The Python/C API

릴리스 *3.12.9*

Guido van Rossum and the Python development team

4월 08, 2025

Contents

1 소개 **3**
- 1.1 코딩 표준 . . . 3
- 1.2 인클루드 파일 . . . 3
- 1.3 유용한 매크로들 . . . 4
- 1.4 객체, 형 그리고 참조 횟수 . . . 7
- 1.5 예외 . . . 10
- 1.6 파이썬 임베딩하기 . . . 12
- 1.7 디버깅 빌드 . . . 13

2 C API Stability **15**
- 2.1 Unstable C API . . . 15
- 2.2 안정적인 응용 프로그램 바이너리 인터페이스 . . . 15
- 2.3 Platform Considerations . . . 17
- 2.4 Contents of Limited API . . . 17

3 매우 고수준 계층 **41**

4 참조 횟수 **47**

5 예외 처리 **51**
- 5.1 인쇄와 지우기 . . . 51
- 5.2 예외 발생시키기 . . . 52
- 5.3 경고 발행하기 . . . 55
- 5.4 에러 표시기 조회하기 . . . 55
- 5.5 시그널 처리하기 . . . 59
- 5.6 예외 클래스 . . . 60
- 5.7 예외 객체 . . . 60
- 5.8 유니코드 예외 객체 . . . 61
- 5.9 재귀 제어 . . . 62
- 5.10 표준 예외 . . . 63
- 5.11 표준 경고 범주 . . . 64

6 유틸리티 **65**
- 6.1 운영 체제 유틸리티 . . . 65
- 6.2 시스템 함수 . . . 68
- 6.3 프로세스 제어 . . . 70
- 6.4 모듈 임포트 하기 . . . 70
- 6.5 데이터 마샬링 지원 . . . 74
- 6.6 인자 구문 분석과 값 구축 . . . 75
- 6.7 문자열 변환과 포매팅 . . . 83
- 6.8 PyHash API . . . 85

6.9 리플렉션 . . . 85
6.10 코덱 등록소와 지원 함수 . . . 86
6.11 Support for Perf Maps . . . 87

7 추상 객체 계층 89
7.1 객체 프로토콜 . . . 89
7.2 호출 프로토콜 . . . 94
7.3 숫자 프로토콜 . . . 99
7.4 시퀀스 프로토콜 . . . 102
7.5 매핑 프로토콜 . . . 104
7.6 이터레이터 프로토콜 . . . 105
7.7 버퍼 프로토콜 . . . 106
7.8 낡은 버퍼 프로토콜 . . . 113

8 구상 객체 계층 115
8.1 기본 객체 . . . 115
8.2 숫자 객체 . . . 121
8.3 시퀀스 객체 . . . 129
8.4 컨테이너 객체 . . . 153
8.5 함수 객체 . . . 159
8.6 기타 객체 . . . 166

9 초기화, 파이널리제이션 및 스레드 191
9.1 파이썬 초기화 전 . . . 191
9.2 전역 구성 변수 . . . 192
9.3 인터프리터 초기화와 파이널리제이션 . . . 195
9.4 프로세스 전체 매개 변수 . . . 196
9.5 스레드 상태와 전역 인터프리터 록 . . . 200
9.6 서브 인터프리터 지원 . . . 207
9.7 비동기 알림 . . . 210
9.8 프로파일링과 추적 . . . 211
9.9 고급 디버거 지원 . . . 213
9.10 스레드 로컬 저장소 지원 . . . 213

10 파이썬 초기화 구성 217
10.1 Example . . . 217
10.2 PyWideStringList . . . 218
10.3 PyStatus . . . 218
10.4 PyPreConfig . . . 220
10.5 Preinitialize Python with PyPreConfig . . . 221
10.6 PyConfig . . . 222
10.7 PyConfig를 사용한 초기화 . . . 233
10.8 격리된 구성 . . . 234
10.9 파이썬 구성 . . . 235
10.10 Python Path Configuration . . . 235
10.11 Py_RunMain() . . . 236
10.12 Py_GetArgcArgv() . . . 236
10.13 다단계 초기화 비공개 잠정적 API . . . 236

11 메모리 관리 239
11.1 개요 . . . 239
11.2 Allocator Domains . . . 240
11.3 원시 메모리 인터페이스 . . . 240
11.4 메모리 인터페이스 . . . 241
11.5 객체 할당자 . . . 242
11.6 기본 메모리 할당자 . . . 243
11.7 메모리 할당자 사용자 정의 . . . 244
11.8 Debug hooks on the Python memory allocators . . . 245
11.9 pymalloc 할당자 . . . 246

11.10 tracemalloc C API . . . 247
11.11 예 . . . 247

12 객체 구현 지원 **249**
12.1 힙에 객체 할당하기 . . . 249
12.2 공통 객체 구조체 . . . 250
12.3 Type Object Structures . . . 258
12.4 순환 가비지 수집 지원 . . . 294

13 API와 ABI 버전 붙이기 **299**

A 용어집 **301**

B About this documentation **317**
B.1 Contributors to the Python documentation . . . 317

C 역사와 라이센스 **319**
C.1 소프트웨어의 역사 . . . 319
C.2 파이썬에 액세스하거나 사용하기 위한 이용 약관 . . . 320
C.3 포함된 소프트웨어에 대한 라이센스 및 승인 . . . 323

D 저작권 **337**

색인 **339**

이 설명서는 확장 모듈을 작성하거나 파이썬을 내장하고자 하는 C와 C++ 프로그래머가 사용하는 API에 관해 설명합니다. 이 설명서와 쌍을 이루는 extending-index 는 확장 제작의 일반 원칙을 설명하지만, API 함수를 자세하게 설명하지는 않습니다.

CHAPTER 1

소개

파이썬의 애플리케이션 프로그래머용 인터페이스는 다양한 수준에서 C/C++ 프로그래머에게 파이썬 인터프리터에 대한 접근 방법을 제공합니다. 이 API는 C++에서도 동일하게 사용 가능하지만 간결함을 위해 보통 파이썬/C API 로 불립니다. 파이썬/C API를 사용하는 데에는 근본적으로 다른 두 가지 이유가 있습니다. 첫번째 이유는 특정한 목적을 위해 확장 모듈을 작성하기 위해서입니다; 이 확장 모듈들은 파이썬 인터프리터를 확장하는 C 모듈들입니다. 이것이 아마도 가장 흔한 용도일 것입니다. 두번째 이유는 파이썬을 더 큰 애플리케이션의 컴포넌트로 사용하기 위함입니다. 이 기술은 일반적으로 파이썬을 애플리케이션에 임베딩(*embedding*) 하는 것을 말합니다.

확장 모듈을 작성하는 것은 비교적 잘 다듬어진 과정으로, "쿡북" 접근법이 잘 통하며 프로세스를 다소 자동화하는 툴들도 존재합니다. 사람들은 파이썬이 존재한 초기부터 다른 애플리케이션에 파이썬을 임베드해왔으나 파이썬을 임베딩 하는 과정은 확장 모듈을 작성하는 것보다 복잡합니다.

많은 API 함수들은 파이썬을 임베딩하거나 확장하는 것에 무관하게 유용합니다. 더욱이 파이썬을 임베드하는 대부분의 애플리케이션은 커스텀 확장을 제공할 필요성이 있기 때문에 파이썬을 임베드하려고 시도하기 전에 확장을 작성하는 것에 친숙해지는 것이 좋습니다.

1.1 코딩 표준

CPython 에 포함하기 위해 C 코드를 작성하는 경우에는 **PEP 7** 에 정의된 지침과 표준을 따라야 합니다. 이 지침은 기여하고 있는 파이썬 버전과 상관없이 적용됩니다. 최종적으로 파이썬에 기여하는 것을 기대하지 않는 이상 이 규칙을 따르는 것은 제삼자 확장 모듈에는 필수가 아닙니다.

1.2 인클루드 파일

파이썬/C API를 사용하기 위한 모든 함수, 타입 그리고 매크로 정의는 다음 행에 의해 인클루드됩니다.

```
#define PY_SSIZE_T_CLEAN
#include <Python.h>
```

이는 다음과 같은 표준 헤더를 인클루드하는것을 의미합니다: `<stdio.h>`, `<string.h>`, `<errno.h>`, `<limits.h>`, `<assert.h>` 그리고 `<stdlib.h>` (사용 가능한 경우).

참고

파이썬은 일부 시스템의 표준 헤더에 영향을 미치는 전처리기 정의를 정의할 수 있으므로 표준 헤더를 인클루드하기 전에 `Python.h` 를 인클루드해야 합니다.

`Python.h` 를 인클루드하기 전에 항상 `PY_SSIZE_T_CLEAN` 를 정의하는 것을 권장합니다. 이 매크로에 대한 자세한 사항은 인자 구문 분석과 값 구축 을 참조하십시오.

Python.h로 정의된 사용자에게 공개되는 모든 이름들은 (포함된 표준 헤더로 정의된 것은 제외) `Py` 또는 `_Py` 로 시작하는 이름을 가지고 있습니다. `_Py` 로 시작하는 이름들은 파이썬 구현에 의해 내부적으로 사용되며 확장 개발자들에 의해 사용되어서는 안됩니다. 구조체 멤버들은 이름에 접두사가 붙지 않습니다.

참고

사용자 코드는 `Py` 또는 `_Py` 로 시작하는 이름들을 정의해서는 안됩니다. 이것은 읽는 사람을 혼란스럽게 하며 이러한 접두사가 붙는 추가적인 이름을 정의할수도 있는 향후의 파이썬 버전에 대한 사용자 코드의 이식성을 위태롭게 합니다.

The header files are typically installed with Python. On Unix, these are located in the directories *prefix*`/include/pythonversion/` and *exec_prefix*`/include/pythonversion/`, where `prefix` and `exec_prefix` are defined by the corresponding parameters to Python's **`configure`** script and *version* is `'%d.%d' % sys.version_info[:2]`. On Windows, the headers are installed in *prefix*`/include`, where `prefix` is the installation directory specified to the installer.

To include the headers, place both directories (if different) on your compiler's search path for includes. Do *not* place the parent directories on the search path and then use `#include <pythonX.Y/Python.h>`; this will break on multi-platform builds since the platform independent headers under `prefix` include the platform specific headers from `exec_prefix`.

C++ 사용자들은 파이썬/C API 가 C 만을 사용하여 정의되었어도 헤더 파일들이 `extern "C"` 로 진입점을 제대로 선언한다는 점에 유의해야 합니다. C++ 에서 파이썬/C API 를 사용하기 위해 특별한 조치를 취할 필요는 없습니다.

1.3 유용한 매크로들

파이썬 헤더 파일에는 몇 가지 유용한 매크로가 정의되어 있습니다. 대부분은 필요한 곳에 가깝게 정의되어 있습니다. (예를 들어 *Py_RETURN_NONE*) 나머지 더 일반적인 유틸리티들은 여기에 정의되어 있습니다. 아래 목록이 전체 목록은 아닙니다.

PyMODINIT_FUNC

Declare an extension module `PyInit` initialization function. The function return type is *PyObject*`*`. The macro declares any special linkage declarations required by the platform, and for C++ declares the function as `extern "C"`.

The initialization function must be named `PyInit_`*name*, where *name* is the name of the module, and should be the only non-`static` item defined in the module file. Example:

```
static struct PyModuleDef spam_module = {
    PyModuleDef_HEAD_INIT,
    .m_name = "spam",
    ...
};

PyMODINIT_FUNC
PyInit_spam(void)
{
    return PyModule_Create(&spam_module);
}
```

Py_ABS(x)
x 의 절댓값을 반환합니다.

Added in version 3.3.

Py_ALWAYS_INLINE
Ask the compiler to always inline a static inline function. The compiler can ignore it and decides to not inline the function.

It can be used to inline performance critical static inline functions when building Python in debug mode with function inlining disabled. For example, MSC disables function inlining when building in debug mode.

Marking blindly a static inline function with Py_ALWAYS_INLINE can result in worse performances (due to increased code size for example). The compiler is usually smarter than the developer for the cost/benefit analysis.

If Python is built in debug mode (if the *Py_DEBUG* macro is defined), the *Py_ALWAYS_INLINE* macro does nothing.

It must be specified before the function return type. Usage:

```
static inline Py_ALWAYS_INLINE int random(void) { return 4; }
```

Added in version 3.11.

Py_CHARMASK(c)
인자는 문자 또는 [-128, 127] 나 [0, 255] 사이의 정수여야 합니다. 이 매크로는 `unsigned char` 로 캐스팅된 `c` 를 반환합니다

Py_DEPRECATED(version)
폐지(deprecated) 선언에 사용하십시오. 이 매크로는 심볼 이름 앞에 위치해야 합니다

예제:

```
Py_DEPRECATED(3.8) PyAPI_FUNC(int) Py_OldFunction(void);
```

버전 3.8에서 변경: MSVC 지원을 추가했습니다.

Py_GETENV(s)
Like `getenv(s)`, but returns `NULL` if `-E` was passed on the command line (see *PyConfig.use_environment*).

Py_MAX(x, y)
x 와 y 사이의 최댓값을 반환합니다.

Added in version 3.3.

Py_MEMBER_SIZE(type, member)
(`type`) 구조체의 `member` 의 크기를 바이트로 반환합니다.

Added in version 3.6.

Py_MIN(x, y)
x 와 y 사이의 최솟값을 반환합니다.

Added in version 3.3.

Py_NO_INLINE
Disable inlining on a function. For example, it reduces the C stack consumption: useful on LTO+PGO builds which heavily inline code (see bpo-33720).

Usage:

```
Py_NO_INLINE static int random(void) { return 4; }
```

Added in version 3.11.

Py_STRINGIFY(x)

x 를 C 문자열로 변환합니다. 예를 들어 `Py_STRINGIFY(123)` 은 `"123"` 을 반환합니다.

Added in version 3.4.

Py_UNREACHABLE()

의도적으로 도달할 수 없는 코드 경로가 있을 경우에 이 매크로를 사용하십시오. 예를 들어, `switch` 문에서 가능한 모든 값이 `case` 절에서 다뤄지는 경우에 `default:` 절에서 사용할 수 있습니다. `assert(0)` 또는 `abort()` 대신 사용하십시오.

릴리즈 모드에서 이 매크로는 컴파일러가 코드를 최적화하는데 도움이 되며 도달할 수 없는 코드에 대한 경고를 방지합니다. 예를 들어, 이 매크로는 릴리즈 모드에서 GCC의 `__builtin_unreachable()` 로 구현됩니다.

`Py_UNREACHABLE()` 의 용도는 반환하지 않지만 `_Py_NO_RETURN` 을 선언하지 않은 함수를 호출하는 것입니다

코드 경로가 매우 가능성이 낮지만 예외적인 경우에 도달할 수 있는 경우, 이 매크로를 사용해서는 안됩니다. 예를 들어, 메모리가 부족하거나 시스템 콜이 예상 범위를 벗어나는 값을 반환했을 경우에는 호출자에게 에러를 보고하는 것이 좋습니다. 호출자에게 에러를 보고할 수 없는 경우 *Py_FatalError()* 를 사용할 수 있습니다.

Added in version 3.7.

Py_UNUSED(arg)

함수의 미사용 인자에 사용하여 컴파일러 경고를 무시합니다. 예시: `int func(int a, int Py_UNUSED(b)) { return a; }`.

Added in version 3.4.

PyDoc_STRVAR(name, str)

독스트링에서 사용 가능한 `name` 이란 이름의 변수를 생성합니다. 파이썬이 독스트링 없이 빌드되었다면 변수의 값은 비어있을 것입니다.

PEP 7 에 명시된 것처럼 파이썬을 독스트링 없이 빌드하기 위해 *PyDoc_STRVAR* 를 독스트링에 사용하십시오

예제:

```
PyDoc_STRVAR(pop_doc, "Remove and return the rightmost element.");

static PyMethodDef deque_methods[] = {
    // ...
    {"pop", (PyCFunction)deque_pop, METH_NOARGS, pop_doc},
    // ...
}
```

PyDoc_STR(str)

주어진 문자열에 대한 독스트링을 생성합니다. 독스트링이 비활성화 되어있을 경우엔 빈 문자열을 생성합니다.

PEP 7 에 명시된 것처럼 독스트링 없이 파이썬을 빌드할 수 있도록 독스트링을 명시할 때 *PyDoc_STR* 을 사용하십시오.

예제:

```
static PyMethodDef pysqlite_row_methods[] = {
    {"keys", (PyCFunction)pysqlite_row_keys, METH_NOARGS,
        PyDoc_STR("Returns the keys of the row.")},
    {NULL, NULL}
};
```

1.4 객체, 형 그리고 참조 횟수

Most Python/C API functions have one or more arguments as well as a return value of type *PyObject**. This type is a pointer to an opaque data type representing an arbitrary Python object. Since all Python object types are treated the same way by the Python language in most situations (e.g., assignments, scope rules, and argument passing), it is only fitting that they should be represented by a single C type. Almost all Python objects live on the heap: you never declare an automatic or static variable of type *PyObject*, only pointer variables of type *PyObject** can be declared. The sole exception are the type objects; since these must never be deallocated, they are typically static *PyTypeObject* objects.

모든 파이썬 객체는 (파이썬 정수조차도) 형(*type*) 과 참조 횟수(*reference count*) 를 가지고 있습니다. 객체의 형은 객체의 종류를 결정합니다. (예를 들어 정수, 리스트, 또는 사용자 정의 함수 등. types 에 추가적인 형들에 대해 설명되어 있습니다.) 잘 알려진 형에는 객체가 해당 형인지를 확인하는 매크로가 있습니다. 예를 들어 `PyList_Check(a)` 는 *a* 가 가리키는 객체가 파이썬 리스트일 경우에만 참입니다.

1.4.1 참조 횟수

The reference count is important because today' s computers have a finite (and often severely limited) memory size; it counts how many different places there are that have a *strong reference* to an object. Such a place could be another object, or a global (or static) C variable, or a local variable in some C function. When the last *strong reference* to an object is released (i.e. its reference count becomes zero), the object is deallocated. If it contains references to other objects, those references are released. Those other objects may be deallocated in turn, if there are no more references to them, and so on. (There' s an obvious problem with objects that reference each other here; for now, the solution is "don' t do that.")

Reference counts are always manipulated explicitly. The normal way is to use the macro *Py_INCREF()* to take a new reference to an object (i.e. increment its reference count by one), and *Py_DECREF()* to release that reference (i.e. decrement the reference count by one). The *Py_DECREF()* macro is considerably more complex than the incref one, since it must check whether the reference count becomes zero and then cause the object' s deallocator to be called. The deallocator is a function pointer contained in the object' s type structure. The type-specific deallocator takes care of releasing references for other objects contained in the object if this is a compound object type, such as a list, as well as performing any additional finalization that' s needed. There' s no chance that the reference count can overflow; at least as many bits are used to hold the reference count as there are distinct memory locations in virtual memory (assuming `sizeof(Py_ssize_t) >= sizeof(void*)`). Thus, the reference count increment is a simple operation.

It is not necessary to hold a *strong reference* (i.e. increment the reference count) for every local variable that contains a pointer to an object. In theory, the object' s reference count goes up by one when the variable is made to point to it and it goes down by one when the variable goes out of scope. However, these two cancel each other out, so at the end the reference count hasn' t changed. The only real reason to use the reference count is to prevent the object from being deallocated as long as our variable is pointing to it. If we know that there is at least one other reference to the object that lives at least as long as our variable, there is no need to take a new *strong reference* (i.e. increment the reference count) temporarily. An important situation where this arises is in objects that are passed as arguments to C functions in an extension module that are called from Python; the call mechanism guarantees to hold a reference to every argument for the duration of the call.

However, a common pitfall is to extract an object from a list and hold on to it for a while without taking a new reference. Some other operation might conceivably remove the object from the list, releasing that reference, and possibly deallocating it. The real danger is that innocent-looking operations may invoke arbitrary Python code which could do this; there is a code path which allows control to flow back to the user from a *Py_DECREF()*, so almost any operation is potentially dangerous.

A safe approach is to always use the generic operations (functions whose name begins with `PyObject_`, `PyNumber_`, `PySequence_` or `PyMapping_`). These operations always create a new *strong reference* (i.e. increment the reference count) of the object they return. This leaves the caller with the responsibility to call *Py_DECREF()* when they are done with the result; this soon becomes second nature.

참조 횟수 상세

The reference count behavior of functions in the Python/C API is best explained in terms of *ownership of references*. Ownership pertains to references, never to objects (objects are not owned: they are always shared). "Owning a reference" means being responsible for calling Py_DECREF on it when the reference is no longer needed. Ownership can also be transferred, meaning that the code that receives ownership of the reference then becomes responsible for eventually releasing it by calling *Py_DECREF()* or *Py_XDECREF()* when it' s no longer needed—or passing on this responsibility (usually to its caller). When a function passes ownership of a reference on to its caller, the caller is said to receive a *new* reference. When no ownership is transferred, the caller is said to *borrow* the reference. Nothing needs to be done for a *borrowed reference*.

반대로 호출 함수에게 객체에 대한 참조를 넘길 때는 두가지 가능성이 있습니다: 함수는 객체에 대한 참조를 훔칠 수도, 그러지 않을 수도 있습니다. 참조를 훔치는 것은 함수에 참조를 전달할 때 해당 함수가 전달된 참조를 소유한다고 가정하고 더 이상 책임을 지지 않는다는 것을 의미합니다.

참조를 훔치는 함수는 거의 없습니다. 주목할만한 두가지 예외는 *PyList_SetItem()* 과 *PyTuple_SetItem()* 입니다. 이 두가지 함수는 요소에 대한 참조를 훔칩니다(단, 요소를 넣을 튜플이나 리스트에 대한 참조는 훔치지 않습니다.). 이 함수들은 새로 만들어진 객체들로 튜플이나 리스트를 채우는 일반적인 관행 때문에 참조를 훔치도록 설계되었습니다. 예를 들어, 튜플을 만드는 코드 `(1, 2, "three")` 는 다음과 같을 수 있습니다. (잠시 에러 처리는 잊어버리십시오. 더 좋은 방법으로 코딩하는 방법은 아래에 나와 있습니다.)

```
PyObject *t;

t = PyTuple_New(3);
PyTuple_SetItem(t, 0, PyLong_FromLong(1L));
PyTuple_SetItem(t, 1, PyLong_FromLong(2L));
PyTuple_SetItem(t, 2, PyUnicode_FromString("three"));
```

여기서 *PyTuple_SetItem()* 는 *PyLong_FromLong()* 가 반환한 참조를 곧바로 훔칩니다. 객체에 대한 참조가 훔쳐져도 계속 객체를 사용하려면 참조를 훔치는 함수를 호출하기 전에 *Py_INCREF()* 를 다른 참조를 가져오는데 사용하십시오.

덧붙이자면, *PyTuple_SetItem()* 은 튜플에 요소를 넣는 유일한 방법입니다. 튜플은 불변 자료형이기 때문에 *PySequence_SetItem()* 과 *PyObject_SetItem()* 는 튜플에 요소를 넣는 것을 거부합니다. *PyTuple_SetItem()* 은 직접 만들고 있는 튜플에만 사용되어야 합니다.

리스트를 채우는 동일한 의미의 코드는 *PyList_New()* 와 *PyList_SetItem()* 을 사용해 만들 수 있습니다.

하지만 실제로는 이렇게 튜플 또는 리스트를 만들고 채우는 경우는 드뭅니다. 일반적인 객체들을 형식 문자열(*format string*)로 지시되는 C 값으로부터 만들어낼 수 있는 제네릭 함수 *Py_BuildValue()* 가 있습니다. 예를 들어, 위의 두 블록의 코드를 다음 코드로 대체할 수 있습니다. (에러 검사도 처리합니다.)

```
PyObject *tuple, *list;

tuple = Py_BuildValue("(iis)", 1, 2, "three");
list = Py_BuildValue("[iis]", 1, 2, "three");
```

It is much more common to use *PyObject_SetItem()* and friends with items whose references you are only borrowing, like arguments that were passed in to the function you are writing. In that case, their behaviour regarding references is much saner, since you don' t have to take a new reference just so you can give that reference away ("have it be stolen"). For example, this function sets all items of a list (actually, any mutable sequence) to a given item:

```
int
set_all(PyObject *target, PyObject *item)
{
    Py_ssize_t i, n;

    n = PyObject_Length(target);
    if (n < 0)
        return -1;
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    for (i = 0; i < n; i++) {
        PyObject *index = PyLong_FromSsize_t(i);
        if (!index)
            return -1;
        if (PyObject_SetItem(target, index, item) < 0) {
            Py_DECREF(index);
            return -1;
        }
        Py_DECREF(index);
    }
    return 0;
}
```

함수 반환 값에 대해서는 상황이 약간 다릅니다. 대부분의 함수에 참조를 전달해도 해당 참조에 대한 소유권 책임이 바뀌진 않지만 객체에 대한 참조를 제공하는 많은 함수는 참조의 소유권을 제공합니다. 이유는 간단합니다. 대부분의 경우에서 반환된 객체는 즉석에서 생성되고 반환된 참조는 객체에 대한 유일한 참조입니다. 따라서 *PyObject_GetItem()* 과 *PySequence_GetItem()* 처럼 객체에 대한 참조를 반환하는 제네릭 함수들은 언제나 새로운 참조를 반환합니다 (호출자가 객체의 소유자가 됩니다).

함수의 의해 반환된 함수를 소유하고 있는지는 어떤 함수를 호출하느냐에 따라 달라진다는 것을 아는 것이 중요합니다. — 깃털 (함수에 인자로 전달된 객체의 형) 은 해당되지 않습니다! 따라서 *PyList_GetItem()* 를 사용하여 리스트에서 항목을 가져오면 참조를 소유하지 않습니다. — 하지만 동일한 인자를 받는 *PySequence_GetItem()* 를 사용하여 리스트에서 항목을 가져온다면 반환된 객체에 대한 참조를 소유하게 됩니다.

다음은 정수 리스트에 있는 항목의 합계를 구하는 함수를 작성하는 방법의 예시입니다. 한 번은 *PyList_GetItem()* 를 사용하고, 한 번은 *PySequence_GetItem()* 을 사용합니다.

```
long
sum_list(PyObject *list)
{
    Py_ssize_t i, n;
    long total = 0, value;
    PyObject *item;

    n = PyList_Size(list);
    if (n < 0)
        return -1; /* Not a list */
    for (i = 0; i < n; i++) {
        item = PyList_GetItem(list, i); /* Can't fail */
        if (!PyLong_Check(item)) continue; /* Skip non-integers */
        value = PyLong_AsLong(item);
        if (value == -1 && PyErr_Occurred())
            /* Integer too big to fit in a C long, bail out */
            return -1;
        total += value;
    }
    return total;
}
```

```
long
sum_sequence(PyObject *sequence)
{
    Py_ssize_t i, n;
    long total = 0, value;
    PyObject *item;
    n = PySequence_Length(sequence);
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    if (n < 0)
        return -1; /* Has no length */
    for (i = 0; i < n; i++) {
        item = PySequence_GetItem(sequence, i);
        if (item == NULL)
            return -1; /* Not a sequence, or other failure */
        if (PyLong_Check(item)) {
            value = PyLong_AsLong(item);
            Py_DECREF(item);
            if (value == -1 && PyErr_Occurred())
                /* Integer too big to fit in a C long, bail out */
                return -1;
            total += value;
        }
        else {
            Py_DECREF(item); /* Discard reference ownership */
        }
    }
    return total;
}
```

1.4.2 형

There are few other data types that play a significant role in the Python/C API; most are simple C types such as `int`, `long`, `double` and `char*`. A few structure types are used to describe static tables used to list the functions exported by a module or the data attributes of a new object type, and another is used to describe the value of a complex number. These will be discussed together with the functions that use them.

type **`Py_ssize_t`**

Part of the Stable ABI. A signed integral type such that `sizeof(Py_ssize_t) == sizeof(size_t)`. C99 doesn' t define such a thing directly (size_t is an unsigned integral type). See **PEP 353** for details. `PY_SSIZE_T_MAX` is the largest positive value of type *`Py_ssize_t`*.

1.5 예외

파이썬 프로그래머는 특정한 에러 처리가 필요할 경우에만 에러를 처리하면 됩니다. 처리되지 않은 예외는 사용자에게 전달되는 최상위 인터프리터까지 스택 트레이스백과 함께 자동으로 호출자, 호출자의 호출자 등으로 전파됩니다.

그러나 C 프로그래머들에게 에러 검사는 항상 명시적이어야만 합니다. 파이썬/C API의 모든 함수는 해당 함수의 문서에서 명시하지 않는 한 예외를 발생시킬 수 있습니다. 일반적으로 함수에 에러가 발생하면 함수는 예외를 설정하고 소유하고 있는 모든 객체에 대한 참조를 취소하고 에러 표시기를 반환합니다. 달리 문서화되지 않은 경우 표시기는 함수의 반환 형에 따라 `NULL` 또는 `-1` 입니다. 일부 함수는 에러를 의미하는 거짓과 함께 참/거짓의 불리언 결과를 반환합니다. 아주 일부의 함수는 명시적인 에러 표시기가 없거나 모호한 반환값을 가지며 *`PyErr_Occurred()`* 를 사용하여 명시적인 점검을 요구합니다. 이런 예외는 항상 명시적으로 문서화됩니다.

예외 상태는 스레드 별 공간에서 관리됩니다. (스레드를 사용하지 않는 프로그램에서는 전역 공간을 사용한다는 말과 같습니다.) 스레드는 예외가 발생했거나, 발생하지 않았거나의 두가지 상태 중 하나일 수 있습니다. 함수 *`PyErr_Occurred()`* 는 이 상태를 확인하기 위해 사용할 수 있습니다. 해당 함수는 예외가 발생했을 경우 예외 형 객체에 대한 빌린 참조를 반환합니다. 예외가 발생하지 않았을 경우엔 `NULL` 을 반환합니다. 예외 상태를 설정하기 위한 여러가지 함수들이 있습니다: *`PyErr_SetString()`* 는 예외 상태를 설정하기 위해 가장 보편적인 (가장 일반적인 것은 아니지만) 함수입니다. *`PyErr_Clear()`* 는 예외 상태를 지웁니다.

전체 예외 상태는 예외 형, 해당 예외 값, 트레이스백이라는 세가지 객체로 구성됩니다. (셋 모두 `NULL` 일 수 있습니다.) 이 세가지 객체는 파이썬의 `sys.exc_info()` 의 결과와 같은 의미를 가지고 있지만 동일하지는

않습니다. 파이썬 객체는 `try` … `except` 문으로 처리되는 마지막 예외를 표현하는 반면 C 수준 예외는 `sys.exc_info()` 와 그 친구들로 예외를 전송하는 파이썬 바이트코드 인터프리터의 메인 루프에 도달할 때까지 C 함수들 간에 전달되는 동안에만 존재합니다.

파이썬 1.5부터 선호되어 온 파이썬 코드에서의 스레드 안전한 예외 상태 접근 방법은 파이썬 코드를 위해 스레드 별 예외 상태를 반환하는 `sys.exc_info()` 함수를 호출하는 것입니다. 또한 예외 상태에 접근하는 양쪽 방법의 의미도 바뀌어 에러를 포착하는 함수가 호출자의 예외 상태를 보존하기 위해 스레드의 예외를 저장하고 복원합니다. 이는 평범해 보이는 함수가 처리중인 예외를 덮어씌우는 것으로 인한 예외 처리 코드의 흔한 버그를 방지합니다. 또한 트레이스백의 스택 프레임에 의하여 참조되는 객체들에 대해 종종 원하지 않은 수명 증가가 일어나는 것을 방지합니다

일반적으로 어떤 작업을 수행하기 위해 다른 함수를 호출하는 함수는 호출된 함수가 예외를 일으켰는지 확인해야만 하며 만약 예외가 일어났다면 호출자에게 예외 상태를 전달해야 합니다. 소유하고 있는 모든 객체에 대한 참조를 버리고 에러 표시기를 반환해야 하지만 다른 예외를 설정해서는 안됩니다. — 방금 일어난 예외를 덮어씌우고 정확한 에러 원인에 대한 중요한 정보를 잃어버리게 됩니다.

A simple example of detecting exceptions and passing them on is shown in the `sum_sequence()` example above. It so happens that this example doesn' t need to clean up any owned references when it detects an error. The following example function shows some error cleanup. First, to remind you why you like Python, we show the equivalent Python code:

```
def incr_item(dict, key):
    try:
        item = dict[key]
    except KeyError:
        item = 0
    dict[key] = item + 1
```

다음은 같은 의미의 웅장한 C 코드입니다:

```
int
incr_item(PyObject *dict, PyObject *key)
{
    /* Objects all initialized to NULL for Py_XDECREF */
    PyObject *item = NULL, *const_one = NULL, *incremented_item = NULL;
    int rv = -1; /* Return value initialized to -1 (failure) */

    item = PyObject_GetItem(dict, key);
    if (item == NULL) {
        /* Handle KeyError only: */
        if (!PyErr_ExceptionMatches(PyExc_KeyError))
            goto error;

        /* Clear the error and use zero: */
        PyErr_Clear();
        item = PyLong_FromLong(0L);
        if (item == NULL)
            goto error;
    }
    const_one = PyLong_FromLong(1L);
    if (const_one == NULL)
        goto error;

    incremented_item = PyNumber_Add(item, const_one);
    if (incremented_item == NULL)
        goto error;

    if (PyObject_SetItem(dict, key, incremented_item) < 0)
        goto error;
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    rv = 0; /* Success */
    /* Continue with cleanup code */

 error:
    /* Cleanup code, shared by success and failure path */

    /* Use Py_XDECREF() to ignore NULL references */
    Py_XDECREF(item);
    Py_XDECREF(const_one);
    Py_XDECREF(incremented_item);

    return rv; /* -1 for error, 0 for success */
}
```

이 예시는 C `goto` 문의 허용된 사용방법을 보여줍니다! 이 예시는 특정한 예외를 처리하기 위한 *PyErr_ExceptionMatches()* 와 *PyErr_Clear()* 의 사용 방법과 *Py_XDECREF()* 를 사용하여 소유하고 있는 `NULL` 일 수도 있는 참조를 삭제하는 방법을 표현합니다. (이름에 있는 `'X'` 를 주목하십시오. *Py_DECREF()* 는 `NULL` 참조와 마주치면 충돌을 일으킵니다.) 이 예시를 수행하려면 소유하고 있는 참조를 보유하는데 사용하는 변수를 `NULL` 로 초기화하는 것이 중요합니다. 마찬가지로 반환 값은 `-1` (실패) 로 설정되고 마지막 호출이 성공한 뒤에야 성공으로 설정됩니다.

1.6 파이썬 임베딩하기

확장 작성자들과는 달리 파이썬 인터프리터를 임베딩 하는 사람들만이 걱정해야 하는 한가지 중요한 문제는 파이썬 인터프리터의 초기화, 그리고 아마도 마무리일 것입니다. 인터프리터의 대부분의 기능은 인터프리터가 초기화 된 이후에 사용할 수 있습니다.

기본적인 초기화 함수는 *Py_Initialize()* 입니다. 이 함수는 로드된 모듈 테이블을 초기화 하고 기본 모듈인 `builtins`, `__main__`, 그리고 `sys` 를 생성합니다. 또한 모듈 검색 경로 (`sys.path`) 를 초기화합니다

Py_Initialize() does not set the "script argument list" (`sys.argv`). If this variable is needed by Python code that will be executed later, setting *PyConfig.argv* and *PyConfig.parse_argv* must be set: see *Python Initialization Configuration*.

대부분의 시스템에서 (특별히 유닉스와 윈도우는 세부적인 부분이 조금 다르긴 하지만) *Py_Initialize()* 는 파이썬 인터프리터를 기준으로 고정된 위치에 파이썬 라이브러리가 있다고 가정하여 표준 파이썬 인터프리터 실행 파일에 대한 최선의 추측을 바탕으로 바탕으로 모듈 검색 경로를 계산합니다. 특히 셸 명령어 검색 경로 (환경 변수 `PATH`) 에서 `python` 이라는 이름의 실행 파일이 발견되는 부모 디렉터리를 기준으로 `lib/python`*X.Y* 같은 이름을 가진 디렉터리를 찾습니다.

예를 들어 파이썬 실행 파일이 `/usr/local/bin/python` 에서 발견된다면 라이브러리는 `/usr/local/lib/python`*X.Y* 에 있는 것으로 가정합니다. (실제로 이 특정 경로는 `PATH` 를 따라 `python` 이라는 이름의 실행 파일이 발견되지 않을 때 사용되는 "fallback" 경로이기도 합니다.) 유저는 환경 변수 `PYTHONHOME` 를 설정하여 이 동작을 재정의하거나 `PYTHONPATH` 를 설정하여 표준 경로 앞에 추가적인 디렉터리를 추가할 수 있습니다.

파이썬을 임베딩하는 애플리케이션은 *Py_Initialize()* 를 호출하기 전에 `Py_SetProgramName(file)` 을 호출하여 검색을 제어할 수 있습니다. `PYTHONHOME` 는 여전히 이 값을 재정의하며 `PYTHONPATH` 는 여전히 표준 경로 앞에 추가된다는 점에 유의하십시오. 완전한 제어가 필요한 애플리케이션은 *Py_GetPath()*, *Py_GetPrefix()*, *Py_GetExecPrefix()*, 그리고 *Py_GetProgramFullPath()* 의 자체적인 구현을 제공할 필요가 있습니다. (모두 `Modules/getpath.c` 에 정의되어 있습니다.)

가끔은 파이썬을 "uninitialize" 하는 것이 바람직합니다. 예를 들어 애플리케이션이 다시 시작하거나 (*Py_Initialize()* 다시 호출하기) 애플리케이션에서 파이썬의 사용이 끝나 파이썬이 할당한 메모리를 해제하려고 할 수 있습니다. *Py_FinalizeEx()* 를 호출하여 이를 달성할 수 있습니다. 함수 *Py_IsInitialized()* 는 파이썬이 현재 초기화된 상태에 있을 경우 참을 반환합니다. 이 함수들에 대한 자세한 내용은 다른 장에서 제공됩니다. *Py_FinalizeEx()* 가 파이썬 인터프리터가 할당한 모든 메모리를 해제하지는 않는다는 점에 유의해야 합니다. 예를 들어, 현재 확장 모듈에서 할당한 메모리는 해제할 수 없습니다.

1.7 디버깅 빌드

파이썬은 인터프리터와 확장 모듈들에 대한 추가적인 검사를 가능하게 하는 여러 매크로를 사용하여 빌드될 수 있습니다. 이러한 검사는 런타임에 많은 오버헤드를 추가하는 경향이 있으므로 기본적으로 실행되지 않습니다.

A full list of the various types of debugging builds is in the file `Misc/SpecialBuilds.txt` in the Python source distribution. Builds are available that support tracing of reference counts, debugging the memory allocator, or low-level profiling of the main interpreter loop. Only the most frequently used builds will be described in the remainder of this section.

`Py_DEBUG`

Compiling the interpreter with the `Py_DEBUG` macro defined produces what is generally meant by a debug build of Python. `Py_DEBUG` is enabled in the Unix build by adding `--with-pydebug` to the `./configure` command. It is also implied by the presence of the not-Python-specific `_DEBUG` macro. When `Py_DEBUG` is enabled in the Unix build, compiler optimization is disabled.

In addition to the reference count debugging described below, extra checks are performed, see Python Debug Build.

Defining `Py_TRACE_REFS` enables reference tracing (see the `configure --with-trace-refs option`). When defined, a circular doubly linked list of active objects is maintained by adding two extra fields to every *PyObject*. Total allocations are tracked as well. Upon exit, all existing references are printed. (In interactive mode this happens after every statement run by the interpreter.)

자세한 내용은 파이썬 소스 배포판 안의 `Misc/SpecialBuilds.txt` 를 참조하십시오.

CHAPTER 2

C API Stability

Unless documented otherwise, Python' s C API is covered by the Backwards Compatibility Policy, **PEP 387**. Most changes to it are source-compatible (typically by only adding new API). Changing existing API or removing API is only done after a deprecation period or to fix serious issues.

CPython' s Application Binary Interface (ABI) is forward- and backwards-compatible across a minor release (if these are compiled the same way; see *Platform Considerations* below). So, code compiled for Python 3.10.0 will work on 3.10.8 and vice versa, but will need to be compiled separately for 3.9.x and 3.11.x.

There are two tiers of C API with different stability expectations:

- *Unstable API*, may change in minor versions without a deprecation period. It is marked by the `PyUnstable` prefix in names.
- *Limited API*, is compatible across several minor releases. When `Py_LIMITED_API` is defined, only this subset is exposed from `Python.h`.

These are discussed in more detail below.

Names prefixed by an underscore, such as `_Py_InternalState`, are private API that can change without notice even in patch releases. If you need to use this API, consider reaching out to CPython developers to discuss adding public API for your use case.

2.1 Unstable C API

Any API named with the `PyUnstable` prefix exposes CPython implementation details, and may change in every minor release (e.g. from 3.9 to 3.10) without any deprecation warnings. However, it will not change in a bugfix release (e.g. from 3.10.0 to 3.10.1).

It is generally intended for specialized, low-level tools like debuggers.

Projects that use this API are expected to follow CPython development and spend extra effort adjusting to changes.

2.2 안정적인 응용 프로그램 바이너리 인터페이스

For simplicity, this document talks about *extensions*, but the Limited API and Stable ABI work the same way for all uses of the API – for example, embedding Python.

2.2.1 Limited C API

Python 3.2 introduced the *Limited API*, a subset of Python' s C API. Extensions that only use the Limited API can be compiled once and be loaded on multiple versions of Python. Contents of the Limited API are *listed below*.

Py_LIMITED_API

Define this macro before including `Python.h` to opt in to only use the Limited API, and to select the Limited API version.

Define `Py_LIMITED_API` to the value of *PY_VERSION_HEX* corresponding to the lowest Python version your extension supports. The extension will be ABI-compatible with all Python 3 releases from the specified one onward, and can use Limited API introduced up to that version.

Rather than using the `PY_VERSION_HEX` macro directly, hardcode a minimum minor version (e.g. `0x030A0000` for Python 3.10) for stability when compiling with future Python versions.

You can also define `Py_LIMITED_API` to `3`. This works the same as `0x03020000` (Python 3.2, the version that introduced Limited API).

2.2.2 Stable ABI

To enable this, Python provides a *Stable ABI*: a set of symbols that will remain ABI-compatible across Python 3.x versions.

> **참고**
>
> The Stable ABI prevents ABI issues, like linker errors due to missing symbols or data corruption due to changes in structure layouts or function signatures. However, other changes in Python can change the *behavior* of extensions. See Python' s Backwards Compatibility Policy (**PEP 387**) for details.

The Stable ABI contains symbols exposed in the *Limited API*, but also other ones – for example, functions necessary to support older versions of the Limited API.

On Windows, extensions that use the Stable ABI should be linked against `python3.dll` rather than a version-specific library such as `python39.dll`.

On some platforms, Python will look for and load shared library files named with the `abi3` tag (e.g. `mymodule.abi3.so`). It does not check if such extensions conform to a Stable ABI. The user (or their packaging tools) need to ensure that, for example, extensions built with the 3.10+ Limited API are not installed for lower versions of Python.

All functions in the Stable ABI are present as functions in Python' s shared library, not solely as macros. This makes them usable from languages that don' t use the C preprocessor.

2.2.3 Limited API Scope and Performance

The goal for the Limited API is to allow everything that is possible with the full C API, but possibly with a performance penalty.

For example, while *PyList_GetItem()* is available, its "unsafe" macro variant *PyList_GET_ITEM()* is not. The macro can be faster because it can rely on version-specific implementation details of the list object.

Without `Py_LIMITED_API` defined, some C API functions are inlined or replaced by macros. Defining `Py_LIMITED_API` disables this inlining, allowing stability as Python' s data structures are improved, but possibly reducing performance.

By leaving out the `Py_LIMITED_API` definition, it is possible to compile a Limited API extension with a version-specific ABI. This can improve performance for that Python version, but will limit compatibility. Compiling with `Py_LIMITED_API` will then yield an extension that can be distributed where a version-specific one is not available – for example, for prereleases of an upcoming Python version.

2.2.4 Limited API Caveats

Note that compiling with `Py_LIMITED_API` is *not* a complete guarantee that code conforms to the *Limited API* or the *Stable ABI*. `Py_LIMITED_API` only covers definitions, but an API also includes other issues, such as expected semantics.

One issue that `Py_LIMITED_API` does not guard against is calling a function with arguments that are invalid in a lower Python version. For example, consider a function that starts accepting `NULL` for an argument. In Python 3.9, `NULL` now selects a default behavior, but in Python 3.8, the argument will be used directly, causing a `NULL` dereference and crash. A similar argument works for fields of structs.

Another issue is that some struct fields are currently not hidden when `Py_LIMITED_API` is defined, even though they' re part of the Limited API.

For these reasons, we recommend testing an extension with *all* minor Python versions it supports, and preferably to build with the *lowest* such version.

We also recommend reviewing documentation of all used API to check if it is explicitly part of the Limited API. Even with `Py_LIMITED_API` defined, a few private declarations are exposed for technical reasons (or even unintentionally, as bugs).

Also note that the Limited API is not necessarily stable: compiling with `Py_LIMITED_API` with Python 3.8 means that the extension will run with Python 3.12, but it will not necessarily *compile* with Python 3.12. In particular, parts of the Limited API may be deprecated and removed, provided that the Stable ABI stays stable.

2.3 Platform Considerations

ABI stability depends not only on Python, but also on the compiler used, lower-level libraries and compiler options. For the purposes of the *Stable ABI*, these details define a "platform". They usually depend on the OS type and processor architecture

It is the responsibility of each particular distributor of Python to ensure that all Python versions on a particular platform are built in a way that does not break the Stable ABI. This is the case with Windows and macOS releases from `python.org` and many third-party distributors.

2.4 Contents of Limited API

Currently, the *Limited API* includes the following items:

- *PY_VECTORCALL_ARGUMENTS_OFFSET*
- *PyAIter_Check()*
- *PyArg_Parse()*
- *PyArg_ParseTuple()*
- *PyArg_ParseTupleAndKeywords()*
- *PyArg_UnpackTuple()*
- *PyArg_VaParse()*
- *PyArg_VaParseTupleAndKeywords()*
- *PyArg_ValidateKeywordArguments()*
- *PyBaseObject_Type*
- *PyBool_FromLong()*
- *PyBool_Type*
- *PyBuffer_FillContiguousStrides()*

- *PyBuffer_FillInfo()*
- *PyBuffer_FromContiguous()*
- *PyBuffer_GetPointer()*
- *PyBuffer_IsContiguous()*
- *PyBuffer_Release()*
- *PyBuffer_SizeFromFormat()*
- *PyBuffer_ToContiguous()*
- PyByteArrayIter_Type
- *PyByteArray_AsString()*
- *PyByteArray_Concat()*
- *PyByteArray_FromObject()*
- *PyByteArray_FromStringAndSize()*
- *PyByteArray_Resize()*
- *PyByteArray_Size()*
- *PyByteArray_Type*
- PyBytesIter_Type
- *PyBytes_AsString()*
- *PyBytes_AsStringAndSize()*
- *PyBytes_Concat()*
- *PyBytes_ConcatAndDel()*
- PyBytes_DecodeEscape()
- *PyBytes_FromFormat()*
- *PyBytes_FromFormatV()*
- *PyBytes_FromObject()*
- *PyBytes_FromString()*
- *PyBytes_FromStringAndSize()*
- PyBytes_Repr()
- *PyBytes_Size()*
- *PyBytes_Type*
- *PyCFunction*
- *PyCFunctionWithKeywords*
- PyCFunction_Call()
- PyCFunction_GetFlags()
- PyCFunction_GetFunction()
- PyCFunction_GetSelf()
- *PyCFunction_New()*
- *PyCFunction_NewEx()*
- PyCFunction_Type
- *PyCMethod_New()*

- *PyCallIter_New()*
- *PyCallIter_Type*
- *PyCallable_Check()*
- *PyCapsule_Destructor*
- *PyCapsule_GetContext()*
- *PyCapsule_GetDestructor()*
- *PyCapsule_GetName()*
- *PyCapsule_GetPointer()*
- *PyCapsule_Import()*
- *PyCapsule_IsValid()*
- *PyCapsule_New()*
- *PyCapsule_SetContext()*
- *PyCapsule_SetDestructor()*
- *PyCapsule_SetName()*
- *PyCapsule_SetPointer()*
- PyCapsule_Type
- PyClassMethodDescr_Type
- *PyCodec_BackslashReplaceErrors()*
- *PyCodec_Decode()*
- *PyCodec_Decoder()*
- *PyCodec_Encode()*
- *PyCodec_Encoder()*
- *PyCodec_IgnoreErrors()*
- *PyCodec_IncrementalDecoder()*
- *PyCodec_IncrementalEncoder()*
- *PyCodec_KnownEncoding()*
- *PyCodec_LookupError()*
- *PyCodec_NameReplaceErrors()*
- *PyCodec_Register()*
- *PyCodec_RegisterError()*
- *PyCodec_ReplaceErrors()*
- *PyCodec_StreamReader()*
- *PyCodec_StreamWriter()*
- *PyCodec_StrictErrors()*
- *PyCodec_Unregister()*
- *PyCodec_XMLCharRefReplaceErrors()*
- *PyComplex_FromDoubles()*
- *PyComplex_ImagAsDouble()*
- *PyComplex_RealAsDouble()*

- *PyComplex_Type*
- *PyDescr_NewClassMethod()*
- *PyDescr_NewGetSet()*
- *PyDescr_NewMember()*
- *PyDescr_NewMethod()*
- PyDictItems_Type
- PyDictIterItem_Type
- PyDictIterKey_Type
- PyDictIterValue_Type
- PyDictKeys_Type
- *PyDictProxy_New()*
- PyDictProxy_Type
- PyDictRevIterItem_Type
- PyDictRevIterKey_Type
- PyDictRevIterValue_Type
- PyDictValues_Type
- *PyDict_Clear()*
- *PyDict_Contains()*
- *PyDict_Copy()*
- *PyDict_DelItem()*
- *PyDict_DelItemString()*
- *PyDict_GetItem()*
- *PyDict_GetItemString()*
- *PyDict_GetItemWithError()*
- *PyDict_Items()*
- *PyDict_Keys()*
- *PyDict_Merge()*
- *PyDict_MergeFromSeq2()*
- *PyDict_New()*
- *PyDict_Next()*
- *PyDict_SetItem()*
- *PyDict_SetItemString()*
- *PyDict_Size()*
- *PyDict_Type*
- *PyDict_Update()*
- *PyDict_Values()*
- *PyEllipsis_Type*
- PyEnum_Type
- *PyErr_BadArgument()*

- *PyErr_BadInternalCall()*
- *PyErr_CheckSignals()*
- *PyErr_Clear()*
- PyErr_Display()
- *PyErr_DisplayException()*
- *PyErr_ExceptionMatches()*
- *PyErr_Fetch()*
- *PyErr_Format()*
- *PyErr_FormatV()*
- *PyErr_GetExcInfo()*
- *PyErr_GetHandledException()*
- *PyErr_GetRaisedException()*
- *PyErr_GivenExceptionMatches()*
- *PyErr_NewException()*
- *PyErr_NewExceptionWithDoc()*
- *PyErr_NoMemory()*
- *PyErr_NormalizeException()*
- *PyErr_Occurred()*
- *PyErr_Print()*
- *PyErr_PrintEx()*
- PyErr_ProgramText()
- *PyErr_ResourceWarning()*
- *PyErr_Restore()*
- *PyErr_SetExcFromWindowsErr()*
- *PyErr_SetExcFromWindowsErrWithFilename()*
- *PyErr_SetExcFromWindowsErrWithFilenameObject()*
- *PyErr_SetExcFromWindowsErrWithFilenameObjects()*
- *PyErr_SetExcInfo()*
- *PyErr_SetFromErrno()*
- *PyErr_SetFromErrnoWithFilename()*
- *PyErr_SetFromErrnoWithFilenameObject()*
- *PyErr_SetFromErrnoWithFilenameObjects()*
- *PyErr_SetFromWindowsErr()*
- *PyErr_SetFromWindowsErrWithFilename()*
- *PyErr_SetHandledException()*
- *PyErr_SetImportError()*
- *PyErr_SetImportErrorSubclass()*
- *PyErr_SetInterrupt()*
- *PyErr_SetInterruptEx()*

- *PyErr_SetNone()*
- *PyErr_SetObject()*
- *PyErr_SetRaisedException()*
- *PyErr_SetString()*
- *PyErr_SyntaxLocation()*
- *PyErr_SyntaxLocationEx()*
- *PyErr_WarnEx()*
- *PyErr_WarnExplicit()*
- *PyErr_WarnFormat()*
- *PyErr_WriteUnraisable()*
- *PyEval_AcquireLock()*
- *PyEval_AcquireThread()*
- PyEval_CallFunction()
- PyEval_CallMethod()
- PyEval_CallObjectWithKeywords()
- *PyEval_EvalCode()*
- *PyEval_EvalCodeEx()*
- *PyEval_EvalFrame()*
- *PyEval_EvalFrameEx()*
- *PyEval_GetBuiltins()*
- *PyEval_GetFrame()*
- *PyEval_GetFuncDesc()*
- *PyEval_GetFuncName()*
- *PyEval_GetGlobals()*
- *PyEval_GetLocals()*
- *PyEval_InitThreads()*
- *PyEval_ReleaseLock()*
- *PyEval_ReleaseThread()*
- *PyEval_RestoreThread()*
- *PyEval_SaveThread()*
- *PyEval_ThreadsInitialized()*
- PyExc_ArithmeticError
- PyExc_AssertionError
- PyExc_AttributeError
- PyExc_BaseException
- PyExc_BaseExceptionGroup
- PyExc_BlockingIOError
- PyExc_BrokenPipeError
- PyExc_BufferError

- PyExc_BytesWarning
- PyExc_ChildProcessError
- PyExc_ConnectionAbortedError
- PyExc_ConnectionError
- PyExc_ConnectionRefusedError
- PyExc_ConnectionResetError
- PyExc_DeprecationWarning
- PyExc_EOFError
- PyExc_EncodingWarning
- PyExc_EnvironmentError
- PyExc_Exception
- PyExc_FileExistsError
- PyExc_FileNotFoundError
- PyExc_FloatingPointError
- PyExc_FutureWarning
- PyExc_GeneratorExit
- PyExc_IOError
- PyExc_ImportError
- PyExc_ImportWarning
- PyExc_IndentationError
- PyExc_IndexError
- PyExc_InterruptedError
- PyExc_IsADirectoryError
- PyExc_KeyError
- PyExc_KeyboardInterrupt
- PyExc_LookupError
- PyExc_MemoryError
- PyExc_ModuleNotFoundError
- PyExc_NameError
- PyExc_NotADirectoryError
- PyExc_NotImplementedError
- PyExc_OSError
- PyExc_OverflowError
- PyExc_PendingDeprecationWarning
- PyExc_PermissionError
- PyExc_ProcessLookupError
- PyExc_RecursionError
- PyExc_ReferenceError
- PyExc_ResourceWarning

- PyExc_RuntimeError
- PyExc_RuntimeWarning
- PyExc_StopAsyncIteration
- PyExc_StopIteration
- PyExc_SyntaxError
- PyExc_SyntaxWarning
- PyExc_SystemError
- PyExc_SystemExit
- PyExc_TabError
- PyExc_TimeoutError
- PyExc_TypeError
- PyExc_UnboundLocalError
- PyExc_UnicodeDecodeError
- PyExc_UnicodeEncodeError
- PyExc_UnicodeError
- PyExc_UnicodeTranslateError
- PyExc_UnicodeWarning
- PyExc_UserWarning
- PyExc_ValueError
- PyExc_Warning
- PyExc_WindowsError
- PyExc_ZeroDivisionError
- PyExceptionClass_Name()
- *PyException_GetArgs()*
- *PyException_GetCause()*
- *PyException_GetContext()*
- *PyException_GetTraceback()*
- *PyException_SetArgs()*
- *PyException_SetCause()*
- *PyException_SetContext()*
- *PyException_SetTraceback()*
- *PyFile_FromFd()*
- *PyFile_GetLine()*
- *PyFile_WriteObject()*
- *PyFile_WriteString()*
- PyFilter_Type
- *PyFloat_AsDouble()*
- *PyFloat_FromDouble()*
- *PyFloat_FromString()*

- *PyFloat_GetInfo()*
- *PyFloat_GetMax()*
- *PyFloat_GetMin()*
- *PyFloat_Type*
- *PyFrameObject*
- *PyFrame_GetCode()*
- *PyFrame_GetLineNumber()*
- *PyFrozenSet_New()*
- *PyFrozenSet_Type*
- *PyGC_Collect()*
- *PyGC_Disable()*
- *PyGC_Enable()*
- *PyGC_IsEnabled()*
- *PyGILState_Ensure()*
- *PyGILState_GetThisThreadState()*
- *PyGILState_Release()*
- PyGILState_STATE
- *PyGetSetDef*
- PyGetSetDescr_Type
- *PyImport_AddModule()*
- *PyImport_AddModuleObject()*
- *PyImport_AppendInittab()*
- *PyImport_ExecCodeModule()*
- *PyImport_ExecCodeModuleEx()*
- *PyImport_ExecCodeModuleObject()*
- *PyImport_ExecCodeModuleWithPathnames()*
- *PyImport_GetImporter()*
- *PyImport_GetMagicNumber()*
- *PyImport_GetMagicTag()*
- *PyImport_GetModule()*
- *PyImport_GetModuleDict()*
- *PyImport_Import()*
- *PyImport_ImportFrozenModule()*
- *PyImport_ImportFrozenModuleObject()*
- *PyImport_ImportModule()*
- *PyImport_ImportModuleLevel()*
- *PyImport_ImportModuleLevelObject()*
- *PyImport_ImportModuleNoBlock()*
- *PyImport_ReloadModule()*

- *PyIndex_Check()*
- *PyInterpreterState*
- *PyInterpreterState_Clear()*
- *PyInterpreterState_Delete()*
- *PyInterpreterState_Get()*
- *PyInterpreterState_GetDict()*
- *PyInterpreterState_GetID()*
- *PyInterpreterState_New()*
- *PyIter_Check()*
- *PyIter_Next()*
- *PyIter_Send()*
- PyListIter_Type
- PyListRevIter_Type
- *PyList_Append()*
- *PyList_AsTuple()*
- *PyList_GetItem()*
- *PyList_GetSlice()*
- *PyList_Insert()*
- *PyList_New()*
- *PyList_Reverse()*
- *PyList_SetItem()*
- *PyList_SetSlice()*
- *PyList_Size()*
- *PyList_Sort()*
- *PyList_Type*
- *PyLongObject*
- PyLongRangeIter_Type
- *PyLong_AsDouble()*
- *PyLong_AsLong()*
- *PyLong_AsLongAndOverflow()*
- *PyLong_AsLongLong()*
- *PyLong_AsLongLongAndOverflow()*
- *PyLong_AsSize_t()*
- *PyLong_AsSsize_t()*
- *PyLong_AsUnsignedLong()*
- *PyLong_AsUnsignedLongLong()*
- *PyLong_AsUnsignedLongLongMask()*
- *PyLong_AsUnsignedLongMask()*
- *PyLong_AsVoidPtr()*

- *PyLong_FromDouble()*
- *PyLong_FromLong()*
- *PyLong_FromLongLong()*
- *PyLong_FromSize_t()*
- *PyLong_FromSsize_t()*
- *PyLong_FromString()*
- *PyLong_FromUnsignedLong()*
- *PyLong_FromUnsignedLongLong()*
- *PyLong_FromVoidPtr()*
- *PyLong_GetInfo()*
- *PyLong_Type*
- PyMap_Type
- *PyMapping_Check()*
- *PyMapping_GetItemString()*
- *PyMapping_HasKey()*
- *PyMapping_HasKeyString()*
- *PyMapping_Items()*
- *PyMapping_Keys()*
- *PyMapping_Length()*
- *PyMapping_SetItemString()*
- *PyMapping_Size()*
- *PyMapping_Values()*
- *PyMem_Calloc()*
- *PyMem_Free()*
- *PyMem_Malloc()*
- *PyMem_Realloc()*
- *PyMemberDef*
- PyMemberDescr_Type
- *PyMember_GetOne()*
- *PyMember_SetOne()*
- *PyMemoryView_FromBuffer()*
- *PyMemoryView_FromMemory()*
- *PyMemoryView_FromObject()*
- *PyMemoryView_GetContiguous()*
- PyMemoryView_Type
- *PyMethodDef*
- PyMethodDescr_Type
- *PyModuleDef*
- PyModuleDef_Base

- *PyModuleDef_Init()*
- PyModuleDef_Type
- *PyModule_AddFunctions()*
- *PyModule_AddIntConstant()*
- *PyModule_AddObject()*
- *PyModule_AddObjectRef()*
- *PyModule_AddStringConstant()*
- *PyModule_AddType()*
- *PyModule_Create2()*
- *PyModule_ExecDef()*
- *PyModule_FromDefAndSpec2()*
- *PyModule_GetDef()*
- *PyModule_GetDict()*
- *PyModule_GetFilename()*
- *PyModule_GetFilenameObject()*
- *PyModule_GetName()*
- *PyModule_GetNameObject()*
- *PyModule_GetState()*
- *PyModule_New()*
- *PyModule_NewObject()*
- *PyModule_SetDocString()*
- *PyModule_Type*
- *PyNumber_Absolute()*
- *PyNumber_Add()*
- *PyNumber_And()*
- *PyNumber_AsSsize_t()*
- *PyNumber_Check()*
- *PyNumber_Divmod()*
- *PyNumber_Float()*
- *PyNumber_FloorDivide()*
- *PyNumber_InPlaceAdd()*
- *PyNumber_InPlaceAnd()*
- *PyNumber_InPlaceFloorDivide()*
- *PyNumber_InPlaceLshift()*
- *PyNumber_InPlaceMatrixMultiply()*
- *PyNumber_InPlaceMultiply()*
- *PyNumber_InPlaceOr()*
- *PyNumber_InPlacePower()*
- *PyNumber_InPlaceRemainder()*

- *PyNumber_InPlaceRshift()*
- *PyNumber_InPlaceSubtract()*
- *PyNumber_InPlaceTrueDivide()*
- *PyNumber_InPlaceXor()*
- *PyNumber_Index()*
- *PyNumber_Invert()*
- *PyNumber_Long()*
- *PyNumber_Lshift()*
- *PyNumber_MatrixMultiply()*
- *PyNumber_Multiply()*
- *PyNumber_Negative()*
- *PyNumber_Or()*
- *PyNumber_Positive()*
- *PyNumber_Power()*
- *PyNumber_Remainder()*
- *PyNumber_Rshift()*
- *PyNumber_Subtract()*
- *PyNumber_ToBase()*
- *PyNumber_TrueDivide()*
- *PyNumber_Xor()*
- *PyOS_AfterFork()*
- *PyOS_AfterFork_Child()*
- *PyOS_AfterFork_Parent()*
- *PyOS_BeforeFork()*
- *PyOS_CheckStack()*
- *PyOS_FSPath()*
- *PyOS_InputHook*
- PyOS_InterruptOccurred()
- *PyOS_double_to_string()*
- *PyOS_getsig()*
- PyOS_mystricmp()
- PyOS_mystrnicmp()
- *PyOS_setsig()*
- *PyOS_sighandler_t*
- *PyOS_snprintf()*
- *PyOS_string_to_double()*
- *PyOS_strtol()*
- *PyOS_strtoul()*
- *PyOS_vsnprintf()*

- *PyObject*
- *PyObject.ob_refcnt*
- *PyObject.ob_type*
- *PyObject_ASCII()*
- *PyObject_AsCharBuffer()*
- *PyObject_AsFileDescriptor()*
- *PyObject_AsReadBuffer()*
- *PyObject_AsWriteBuffer()*
- *PyObject_Bytes()*
- *PyObject_Call()*
- *PyObject_CallFunction()*
- *PyObject_CallFunctionObjArgs()*
- *PyObject_CallMethod()*
- *PyObject_CallMethodObjArgs()*
- *PyObject_CallNoArgs()*
- *PyObject_CallObject()*
- *PyObject_Calloc()*
- *PyObject_CheckBuffer()*
- *PyObject_CheckReadBuffer()*
- *PyObject_ClearWeakRefs()*
- *PyObject_CopyData()*
- *PyObject_DelItem()*
- *PyObject_DelItemString()*
- *PyObject_Dir()*
- *PyObject_Format()*
- *PyObject_Free()*
- *PyObject_GC_Del()*
- *PyObject_GC_IsFinalized()*
- *PyObject_GC_IsTracked()*
- *PyObject_GC_Track()*
- *PyObject_GC_UnTrack()*
- *PyObject_GenericGetAttr()*
- *PyObject_GenericGetDict()*
- *PyObject_GenericSetAttr()*
- *PyObject_GenericSetDict()*
- *PyObject_GetAIter()*
- *PyObject_GetAttr()*
- *PyObject_GetAttrString()*
- *PyObject_GetBuffer()*

- *PyObject_GetItem()*
- *PyObject_GetIter()*
- *PyObject_GetTypeData()*
- *PyObject_HasAttr()*
- *PyObject_HasAttrString()*
- *PyObject_Hash()*
- *PyObject_HashNotImplemented()*
- *PyObject_Init()*
- *PyObject_InitVar()*
- *PyObject_IsInstance()*
- *PyObject_IsSubclass()*
- *PyObject_IsTrue()*
- *PyObject_Length()*
- *PyObject_Malloc()*
- *PyObject_Not()*
- *PyObject_Realloc()*
- *PyObject_Repr()*
- *PyObject_RichCompare()*
- *PyObject_RichCompareBool()*
- *PyObject_SelfIter()*
- *PyObject_SetAttr()*
- *PyObject_SetAttrString()*
- *PyObject_SetItem()*
- *PyObject_Size()*
- *PyObject_Str()*
- *PyObject_Type()*
- *PyObject_Vectorcall()*
- *PyObject_VectorcallMethod()*
- *PyProperty_Type*
- PyRangeIter_Type
- PyRange_Type
- PyReversed_Type
- *PySeqIter_New()*
- *PySeqIter_Type*
- *PySequence_Check()*
- *PySequence_Concat()*
- *PySequence_Contains()*
- *PySequence_Count()*
- *PySequence_DelItem()*

- *PySequence_DelSlice()*
- *PySequence_Fast()*
- *PySequence_GetItem()*
- *PySequence_GetSlice()*
- PySequence_In()
- *PySequence_InPlaceConcat()*
- *PySequence_InPlaceRepeat()*
- *PySequence_Index()*
- *PySequence_Length()*
- *PySequence_List()*
- *PySequence_Repeat()*
- *PySequence_SetItem()*
- *PySequence_SetSlice()*
- *PySequence_Size()*
- *PySequence_Tuple()*
- PySetIter_Type
- *PySet_Add()*
- *PySet_Clear()*
- *PySet_Contains()*
- *PySet_Discard()*
- *PySet_New()*
- *PySet_Pop()*
- *PySet_Size()*
- *PySet_Type*
- *PySlice_AdjustIndices()*
- *PySlice_GetIndices()*
- *PySlice_GetIndicesEx()*
- *PySlice_New()*
- *PySlice_Type*
- *PySlice_Unpack()*
- *PyState_AddModule()*
- *PyState_FindModule()*
- *PyState_RemoveModule()*
- *PyStructSequence_Desc*
- *PyStructSequence_Field*
- *PyStructSequence_GetItem()*
- *PyStructSequence_New()*
- *PyStructSequence_NewType()*
- *PyStructSequence_SetItem()*

- *PyStructSequence_UnnamedField*
- PySuper_Type
- *PySys_AddWarnOption()*
- *PySys_AddWarnOptionUnicode()*
- *PySys_AddXOption()*
- *PySys_FormatStderr()*
- *PySys_FormatStdout()*
- *PySys_GetObject()*
- *PySys_GetXOptions()*
- PySys_HasWarnOptions()
- *PySys_ResetWarnOptions()*
- *PySys_SetArgv()*
- *PySys_SetArgvEx()*
- *PySys_SetObject()*
- *PySys_SetPath()*
- *PySys_WriteStderr()*
- *PySys_WriteStdout()*
- *PyThreadState*
- *PyThreadState_Clear()*
- *PyThreadState_Delete()*
- *PyThreadState_Get()*
- *PyThreadState_GetDict()*
- *PyThreadState_GetFrame()*
- *PyThreadState_GetID()*
- *PyThreadState_GetInterpreter()*
- *PyThreadState_New()*
- *PyThreadState_SetAsyncExc()*
- *PyThreadState_Swap()*
- PyThread_GetInfo()
- *PyThread_ReInitTLS()*
- PyThread_acquire_lock()
- PyThread_acquire_lock_timed()
- PyThread_allocate_lock()
- *PyThread_create_key()*
- *PyThread_delete_key()*
- *PyThread_delete_key_value()*
- PyThread_exit_thread()
- PyThread_free_lock()
- *PyThread_get_key_value()*

- PyThread_get_stacksize()
- PyThread_get_thread_ident()
- PyThread_get_thread_native_id()
- PyThread_init_thread()
- PyThread_release_lock()
- *PyThread_set_key_value()*
- PyThread_set_stacksize()
- PyThread_start_new_thread()
- *PyThread_tss_alloc()*
- *PyThread_tss_create()*
- *PyThread_tss_delete()*
- *PyThread_tss_free()*
- *PyThread_tss_get()*
- *PyThread_tss_is_created()*
- *PyThread_tss_set()*
- PyTraceBack_Here()
- PyTraceBack_Print()
- PyTraceBack_Type
- PyTupleIter_Type
- *PyTuple_GetItem()*
- *PyTuple_GetSlice()*
- *PyTuple_New()*
- *PyTuple_Pack()*
- *PyTuple_SetItem()*
- *PyTuple_Size()*
- *PyTuple_Type*
- *PyTypeObject*
- *PyType_ClearCache()*
- *PyType_FromMetaclass()*
- *PyType_FromModuleAndSpec()*
- *PyType_FromSpec()*
- *PyType_FromSpecWithBases()*
- *PyType_GenericAlloc()*
- *PyType_GenericNew()*
- *PyType_GetFlags()*
- *PyType_GetModule()*
- *PyType_GetModuleState()*
- *PyType_GetName()*
- *PyType_GetQualName()*

- *PyType_GetSlot()*
- *PyType_GetTypeDataSize()*
- *PyType_IsSubtype()*
- *PyType_Modified()*
- *PyType_Ready()*
- *PyType_Slot*
- *PyType_Spec*
- *PyType_Type*
- *PyUnicodeDecodeError_Create()*
- *PyUnicodeDecodeError_GetEncoding()*
- *PyUnicodeDecodeError_GetEnd()*
- *PyUnicodeDecodeError_GetObject()*
- *PyUnicodeDecodeError_GetReason()*
- *PyUnicodeDecodeError_GetStart()*
- *PyUnicodeDecodeError_SetEnd()*
- *PyUnicodeDecodeError_SetReason()*
- *PyUnicodeDecodeError_SetStart()*
- *PyUnicodeEncodeError_GetEncoding()*
- *PyUnicodeEncodeError_GetEnd()*
- *PyUnicodeEncodeError_GetObject()*
- *PyUnicodeEncodeError_GetReason()*
- *PyUnicodeEncodeError_GetStart()*
- *PyUnicodeEncodeError_SetEnd()*
- *PyUnicodeEncodeError_SetReason()*
- *PyUnicodeEncodeError_SetStart()*
- PyUnicodeIter_Type
- *PyUnicodeTranslateError_GetEnd()*
- *PyUnicodeTranslateError_GetObject()*
- *PyUnicodeTranslateError_GetReason()*
- *PyUnicodeTranslateError_GetStart()*
- *PyUnicodeTranslateError_SetEnd()*
- *PyUnicodeTranslateError_SetReason()*
- *PyUnicodeTranslateError_SetStart()*
- PyUnicode_Append()
- PyUnicode_AppendAndDel()
- *PyUnicode_AsASCIIString()*
- *PyUnicode_AsCharmapString()*
- PyUnicode_AsDecodedObject()
- PyUnicode_AsDecodedUnicode()

- PyUnicode_AsEncodedObject()
- *PyUnicode_AsEncodedString()*
- PyUnicode_AsEncodedUnicode()
- *PyUnicode_AsLatin1String()*
- *PyUnicode_AsMBCSString()*
- *PyUnicode_AsRawUnicodeEscapeString()*
- *PyUnicode_AsUCS4()*
- *PyUnicode_AsUCS4Copy()*
- *PyUnicode_AsUTF16String()*
- *PyUnicode_AsUTF32String()*
- *PyUnicode_AsUTF8AndSize()*
- *PyUnicode_AsUTF8String()*
- *PyUnicode_AsUnicodeEscapeString()*
- *PyUnicode_AsWideChar()*
- *PyUnicode_AsWideCharString()*
- PyUnicode_BuildEncodingMap()
- *PyUnicode_Compare()*
- *PyUnicode_CompareWithASCIIString()*
- *PyUnicode_Concat()*
- *PyUnicode_Contains()*
- *PyUnicode_Count()*
- *PyUnicode_Decode()*
- *PyUnicode_DecodeASCII()*
- *PyUnicode_DecodeCharmap()*
- *PyUnicode_DecodeCodePageStateful()*
- *PyUnicode_DecodeFSDefault()*
- *PyUnicode_DecodeFSDefaultAndSize()*
- *PyUnicode_DecodeLatin1()*
- *PyUnicode_DecodeLocale()*
- *PyUnicode_DecodeLocaleAndSize()*
- *PyUnicode_DecodeMBCS()*
- *PyUnicode_DecodeMBCSStateful()*
- *PyUnicode_DecodeRawUnicodeEscape()*
- *PyUnicode_DecodeUTF16()*
- *PyUnicode_DecodeUTF16Stateful()*
- *PyUnicode_DecodeUTF32()*
- *PyUnicode_DecodeUTF32Stateful()*
- *PyUnicode_DecodeUTF7()*
- *PyUnicode_DecodeUTF7Stateful()*

- *PyUnicode_DecodeUTF8()*
- *PyUnicode_DecodeUTF8Stateful()*
- *PyUnicode_DecodeUnicodeEscape()*
- *PyUnicode_EncodeCodePage()*
- *PyUnicode_EncodeFSDefault()*
- *PyUnicode_EncodeLocale()*
- *PyUnicode_FSConverter()*
- *PyUnicode_FSDecoder()*
- *PyUnicode_Find()*
- *PyUnicode_FindChar()*
- *PyUnicode_Format()*
- *PyUnicode_FromEncodedObject()*
- *PyUnicode_FromFormat()*
- *PyUnicode_FromFormatV()*
- *PyUnicode_FromObject()*
- PyUnicode_FromOrdinal()
- *PyUnicode_FromString()*
- *PyUnicode_FromStringAndSize()*
- *PyUnicode_FromWideChar()*
- *PyUnicode_GetDefaultEncoding()*
- *PyUnicode_GetLength()*
- *PyUnicode_InternFromString()*
- *PyUnicode_InternInPlace()*
- *PyUnicode_IsIdentifier()*
- *PyUnicode_Join()*
- *PyUnicode_Partition()*
- *PyUnicode_RPartition()*
- *PyUnicode_RSplit()*
- *PyUnicode_ReadChar()*
- *PyUnicode_Replace()*
- PyUnicode_Resize()
- *PyUnicode_RichCompare()*
- *PyUnicode_Split()*
- *PyUnicode_Splitlines()*
- *PyUnicode_Substring()*
- *PyUnicode_Tailmatch()*
- *PyUnicode_Translate()*
- *PyUnicode_Type*
- *PyUnicode_WriteChar()*

- *PyVarObject*
- PyVarObject.ob_base
- *PyVarObject.ob_size*
- *PyVectorcall_Call()*
- *PyVectorcall_NARGS()*
- PyWeakReference
- *PyWeakref_GetObject()*
- *PyWeakref_NewProxy()*
- *PyWeakref_NewRef()*
- PyWrapperDescr_Type
- *PyWrapper_New()*
- PyZip_Type
- *Py_AddPendingCall()*
- *Py_AtExit()*
- *Py_BEGIN_ALLOW_THREADS*
- *Py_BLOCK_THREADS*
- *Py_BuildValue()*
- *Py_BytesMain()*
- *Py_CompileString()*
- *Py_DecRef()*
- *Py_DecodeLocale()*
- *Py_END_ALLOW_THREADS*
- *Py_EncodeLocale()*
- *Py_EndInterpreter()*
- *Py_EnterRecursiveCall()*
- *Py_Exit()*
- *Py_FatalError()*
- Py_FileSystemDefaultEncodeErrors
- Py_FileSystemDefaultEncoding
- *Py_Finalize()*
- *Py_FinalizeEx()*
- *Py_GenericAlias()*
- *Py_GenericAliasType*
- *Py_GetBuildInfo()*
- *Py_GetCompiler()*
- *Py_GetCopyright()*
- *Py_GetExecPrefix()*
- *Py_GetPath()*
- *Py_GetPlatform()*

- *Py_GetPrefix()*
- *Py_GetProgramFullPath()*
- *Py_GetProgramName()*
- *Py_GetPythonHome()*
- Py_GetRecursionLimit()
- *Py_GetVersion()*
- Py_HasFileSystemDefaultEncoding
- *Py_IncRef()*
- *Py_Initialize()*
- *Py_InitializeEx()*
- *Py_Is()*
- *Py_IsFalse()*
- *Py_IsInitialized()*
- *Py_IsNone()*
- *Py_IsTrue()*
- *Py_LeaveRecursiveCall()*
- *Py_Main()*
- Py_MakePendingCalls()
- *Py_NewInterpreter()*
- *Py_NewRef()*
- *Py_ReprEnter()*
- *Py_ReprLeave()*
- *Py_SetPath()*
- *Py_SetProgramName()*
- *Py_SetPythonHome()*
- Py_SetRecursionLimit()
- *Py_UCS4*
- *Py_UNBLOCK_THREADS*
- Py_UTF8Mode
- *Py_VaBuildValue()*
- *Py_Version*
- *Py_XNewRef()*
- *Py_buffer*
- Py_intptr_t
- *Py_ssize_t*
- Py_uintptr_t
- *allocfunc*
- *binaryfunc*
- *descrgetfunc*

- *descrsetfunc*
- *destructor*
- *getattrfunc*
- *getattrofunc*
- *getbufferproc*
- *getiterfunc*
- *getter*
- *hashfunc*
- *initproc*
- *inquiry*
- *iternextfunc*
- *lenfunc*
- *newfunc*
- *objobjargproc*
- *objobjproc*
- *releasebufferproc*
- *reprfunc*
- *richcmpfunc*
- *setattrfunc*
- *setattrofunc*
- *setter*
- *ssizeargfunc*
- *ssizeobjargproc*
- ssizessizeargfunc
- ssizessizeobjargproc
- symtable
- *ternaryfunc*
- *traverseproc*
- *unaryfunc*
- *vectorcallfunc*
- *visitproc*

CHAPTER 3

매우 고수준 계층

이 장의 함수들은 파일이나 버퍼에 제공된 파이썬 소스 코드를 실행할 수 있도록 하지만, 인터프리터와 더 세밀한 방식으로 상호 작용하도록 하지는 않습니다.

Several of these functions accept a start symbol from the grammar as a parameter. The available start symbols are *Py_eval_input*, *Py_file_input*, and *Py_single_input*. These are described following the functions which accept them as parameters.

Note also that several of these functions take `FILE*` parameters. One particular issue which needs to be handled carefully is that the `FILE` structure for different C libraries can be different and incompatible. Under Windows (at least), it is possible for dynamically linked extensions to actually use different libraries, so care should be taken that `FILE*` parameters are only passed to these functions if it is certain that they were created by the same library that the Python runtime is using.

int **Py_Main**(int argc, wchar_t **argv)

Part of the Stable ABI. 표준 인터프리터의 메인 프로그램. 이것은 파이썬을 내장하는 프로그램을 위해 제공됩니다. *argc*와 *argv* 매개 변수는 C 프로그램의 `main()` 함수에 전달되는 것과 정확히 일치하도록 준비해야 합니다 (사용자의 로케일에 따라 wchar_t로 변환됩니다). 인자 목록이 수정될 수 있음에 유의해야 합니다 (하지만 인자 목록이 가리키는 문자열의 내용은 수정되지 않습니다). 인터프리터가 정상적으로 (즉, 예외 없이) 종료되면 반환 값은 `0`, 예외로 인해 인터프리터가 종료되면 `1`, 매개 변수 목록이 유효한 파이썬 명령 줄을 나타내지 않으면 `2`가 됩니다.

Note that if an otherwise unhandled `SystemExit` is raised, this function will not return `1`, but exit the process, as long as *PyConfig.inspect* is zero.

int **Py_BytesMain**(int argc, char **argv)

Part of the Stable ABI *since version 3.8.* *Py_Main()*과 유사하지만 *argv*는 바이트 문자열의 배열입니다.

Added in version 3.8.

int **PyRun_AnyFile**(FILE *fp, const char *filename)

아래 *PyRun_AnyFileExFlags()* 의 단순화된 인터페이스입니다. *closeit*은 `0`으로 *flags*는 `NULL`로 설정된 상태로 남겨둡니다.

int **PyRun_AnyFileFlags**(FILE *fp, const char *filename, *PyCompilerFlags* *flags)

아래 *PyRun_AnyFileExFlags()* 의 단순화된 인터페이스입니다. 이것은 *closeit* 인자를 `0`으로 설정된 상태로 남겨둡니다.

int **PyRun_AnyFileEx**(FILE *fp, const char *filename, int closeit)

아래 *PyRun_AnyFileExFlags()* 의 단순화된 인터페이스입니다. 이것은 *flags* 인자를 `NULL`로 설정된 상태로 남겨둡니다.

int **PyRun_AnyFileExFlags**(FILE *fp, const char *filename, int closeit, *PyCompilerFlags* *flags)

If *fp* refers to a file associated with an interactive device (console or terminal input or Unix pseudo-terminal), return the value of *PyRun_InteractiveLoop()*, otherwise return the result of *PyRun_SimpleFile()*. *filename* is decoded from the filesystem encoding (`sys.getfilesystemencoding()`). If *filename* is `NULL`, this function uses `"???"` as the filename. If *closeit* is true, the file is closed before `PyRun_SimpleFileExFlags()` returns.

int **PyRun_SimpleString**(const char *command)

This is a simplified interface to *PyRun_SimpleStringFlags()* below, leaving the *PyCompilerFlags** argument set to `NULL`.

int **PyRun_SimpleStringFlags**(const char *command, *PyCompilerFlags* *flags)

flags 인자에 따라 `__main__` 모듈에서 *command*에 있는 파이썬 소스 코드를 실행합니다. `__main__`이 존재하지 않으면 만듭니다. 성공하면 `0`을, 예외가 발생하면 `-1`을 반환합니다. 에러가 있으면, 예외 정보를 얻을 방법이 없습니다. *flags*의 의미는 아래를 참조하십시오.

Note that if an otherwise unhandled `SystemExit` is raised, this function will not return `-1`, but exit the process, as long as *PyConfig.inspect* is zero.

int **PyRun_SimpleFile**(FILE *fp, const char *filename)

아래 *PyRun_SimpleFileExFlags()* 의 단순화된 인터페이스입니다. *closeit*을 `0`으로, *flags*를 `NULL`로 설정된 상태로 남겨둡니다.

int **PyRun_SimpleFileEx**(FILE *fp, const char *filename, int closeit)

아래 *PyRun_SimpleFileExFlags()* 의 단순화된 인터페이스입니다. *flags*를 `NULL`로 설정된 상태로 남겨둡니다.

int **PyRun_SimpleFileExFlags**(FILE *fp, const char *filename, int closeit, *PyCompilerFlags* *flags)

Similar to *PyRun_SimpleStringFlags()*, but the Python source code is read from *fp* instead of an in-memory string. *filename* should be the name of the file, it is decoded from *filesystem encoding and error handler*. If *closeit* is true, the file is closed before `PyRun_SimpleFileExFlags()` returns.

> **참고**
>
> 윈도우에서, *fp*는 바이너리 모드로 열어야 합니다 (예를 들어 `fopen(filename, "rb")`). 그렇지 않으면, 파이썬은 LF 줄 종료가 있는 스크립트 파일을 올바르게 처리하지 못할 수 있습니다.

int **PyRun_InteractiveOne**(FILE *fp, const char *filename)

아래 *PyRun_InteractiveOneFlags()* 의 단순화된 인터페이스입니다. *flags*를 `NULL`로 설정된 상태로 남겨둡니다.

int **PyRun_InteractiveOneFlags**(FILE *fp, const char *filename, *PyCompilerFlags* *flags)

Read and execute a single statement from a file associated with an interactive device according to the *flags* argument. The user will be prompted using `sys.ps1` and `sys.ps2`. *filename* is decoded from the *filesystem encoding and error handler*.

입력이 성공적으로 실행될 때 `0`을, 예외가 있으면 `-1`을, 또는 구문 분석 에러가 있으면 파이썬의 일부로 배포된 `errcode.h` 인클루드 파일에 있는 에러 코드를 반환합니다. (`errcode.h`는 `Python.h`에서 인클루드하지 않기 때문에 필요하면 특별히 인클루드해야 함에 유의하십시오.)

int **PyRun_InteractiveLoop**(FILE *fp, const char *filename)

아래 *PyRun_InteractiveLoopFlags()* 의 단순화된 인터페이스입니다. *flags*를 `NULL`로 설정된 상태로 남겨둡니다.

int **PyRun_InteractiveLoopFlags**(FILE *fp, const char *filename, *PyCompilerFlags* *flags)

Read and execute statements from a file associated with an interactive device until EOF is reached. The user will be prompted using `sys.ps1` and `sys.ps2`. *filename* is decoded from the *filesystem encoding and error handler*. Returns `0` at EOF or a negative number upon failure.

int (***PyOS_InputHook**)(void)

Part of the Stable ABI. 프로토타입 `int func(void)` 인 함수를 가리키도록 설정할 수 있습니다. 이 함수는 파이썬의 인터프리터 프롬프트가 유휴 상태가 되고 터미널에서 사용자 입력을 기다리려고 할 때 호출됩니다. 반환 값은 무시됩니다. 이 훅을 재정의하는 것은 파이썬 소스 코드의 `Modules/_tkinter.c`에서 한 것처럼 인터프리터의 프롬프트를 다른 이벤트 루프와 통합하는 데 사용될 수 있습니다.

버전 3.12에서 변경: This function is only called from the *main interpreter*.

char *(***PyOS_ReadlineFunctionPointer**)(FILE*, FILE*, const char*)

프로토타입 `char *func(FILE *stdin, FILE *stdout, char *prompt)` 인 함수를 가리키도록 설정하여, 인터프리터의 프롬프트에서 단일 입력 줄을 읽는 데 사용되는 기본 함수를 재정의할 수 있습니다. 이 함수는 `NULL`이 아니면 문자열 *prompt*를 출력한 다음 제공된 표준 입력 파일에서 입력 줄을 읽고 결과 문자열을 반환할 것이라고 기대됩니다. 예를 들어, `readline` 모듈은 이 훅을 설정하여 줄 편집과 탭 완성 기능을 제공합니다.

결과는 *PyMem_RawMalloc()*이나 *PyMem_RawRealloc()*으로 할당된 문자열 이거나, 에러가 발생했으면 `NULL`이어야 합니다.

버전 3.4에서 변경: 결과는 *PyMem_Malloc()*이나 *PyMem_Realloc()*으로 할당하는 대신, *PyMem_RawMalloc()*이나 *PyMem_RawRealloc()*으로 할당해야 합니다.

버전 3.12에서 변경: This function is only called from the *main interpreter*.

PyObject ***PyRun_String**(const char *str, int start, *PyObject* *globals, *PyObject* *locals)

Return value: New reference. 아래 *PyRun_StringFlags()*의 단순화된 인터페이스입니다. 이것은 *flags*를 `NULL`로 설정된 상태로 남겨둡니다.

PyObject ***PyRun_StringFlags**(const char *str, int start, *PyObject* *globals, *PyObject* *locals, *PyCompilerFlags* *flags)

Return value: New reference. *flags*로 지정된 컴파일러 플래그를 사용하여 *globals*과 *locals* 객체로 지정된 컨텍스트에서 *str*에서 파이썬 소스 코드를 실행합니다. *globals*는 딕셔너리이어야 합니다. *locals*는 매핑 프로토콜을 구현하는 모든 객체가 될 수 있습니다. 매개 변수 *start*는 소스 코드를 구문 분석하는 데 사용해야 하는 시작 토큰을 지정합니다.

코드를 실행한 결과를 파이썬 객체로 반환하거나, 예외가 발생하면 `NULL`을 반환합니다.

PyObject ***PyRun_File**(FILE *fp, const char *filename, int start, *PyObject* *globals, *PyObject* *locals)

Return value: New reference. 아래 *PyRun_FileExFlags()*의 단순화된 인터페이스입니다. *closeit*을 `0`으로, *flags*를 `NULL`로 설정된 상태로 남겨둡니다.

PyObject ***PyRun_FileEx**(FILE *fp, const char *filename, int start, *PyObject* *globals, *PyObject* *locals, int closeit)

Return value: New reference. 아래 *PyRun_FileExFlags()*의 단순화된 인터페이스입니다. *flags*를 `NULL`로 설정된 상태로 남겨둡니다.

PyObject ***PyRun_FileFlags**(FILE *fp, const char *filename, int start, *PyObject* *globals, *PyObject* *locals, *PyCompilerFlags* *flags)

Return value: New reference. 아래 *PyRun_FileExFlags()*의 단순화된 인터페이스입니다. *closeit*을 `0`으로 설정된 상태로 남겨둡니다.

PyObject ***PyRun_FileExFlags**(FILE *fp, const char *filename, int start, *PyObject* *globals, *PyObject* *locals, int closeit, *PyCompilerFlags* *flags)

Return value: New reference. Similar to *PyRun_StringFlags()*, but the Python source code is read from *fp* instead of an in-memory string. *filename* should be the name of the file, it is decoded from the *filesystem encoding and error handler*. If *closeit* is true, the file is closed before *PyRun_FileExFlags()* returns.

PyObject ***Py_CompileString**(const char *str, const char *filename, int start)

Return value: New reference. Part of the Stable ABI. 아래 *Py_CompileStringFlags()* 의 단순화된 인터페이스입니다. *flags*를 `NULL`로 설정된 상태로 유지합니다.

PyObject ***Py_CompileStringFlags**(const char *str, const char *filename, int start, *PyCompilerFlags* *flags)

Return value: New reference. 아래 *Py_CompileStringExFlags()* 의 단순화된 인터페이스입니다. *optimize*를 `-1`로 설정된 상태로 유지합니다.

PyObject ***Py_CompileStringObject**(const char *str, *PyObject* *filename, int start, *PyCompilerFlags* *flags, int optimize)

Return value: New reference. Parse and compile the Python source code in *str*, returning the resulting code object. The start token is given by *start*; this can be used to constrain the code which can be compiled and should be *Py_eval_input*, *Py_file_input*, or *Py_single_input*. The filename specified by *filename* is used to construct the code object and may appear in tracebacks or `SyntaxError` exception messages. This returns `NULL` if the code cannot be parsed or compiled.

정수 *optimize*는 컴파일러의 최적화 수준을 지정합니다. `-1` 값은 `-O` 옵션으로 주어진 것처럼 인터프리터의 최적화 수준을 선택합니다. 명시적 수준은 `0` (최적화 없음; `__debug__`가 참), `1` (어서션이 제거되고 `__debug__`가 거짓) 또는 `2` (독스트링도 제거됩니다) 입니다.

Added in version 3.4.

PyObject ***Py_CompileStringExFlags**(const char *str, const char *filename, int start, *PyCompilerFlags* *flags, int optimize)

Return value: New reference. Like *Py_CompileStringObject()*, but *filename* is a byte string decoded from the *filesystem encoding and error handler*.

Added in version 3.2.

PyObject ***PyEval_EvalCode**(*PyObject* *co, *PyObject* *globals, *PyObject* *locals)

Return value: New reference. Part of the Stable ABI. 이것은 코드 객체와 전역 변수 및 지역 변수만 있는, *PyEval_EvalCodeEx()*의 단순화된 인터페이스입니다. 다른 인자는 `NULL`로 설정됩니다.

PyObject ***PyEval_EvalCodeEx**(*PyObject* *co, *PyObject* *globals, *PyObject* *locals, *PyObject* *const *args, int argcount, *PyObject* *const *kws, int kwcount, *PyObject* *const *defs, int defcount, *PyObject* *kwdefs, *PyObject* *closure)

Return value: New reference. Part of the Stable ABI. 주어진 평가를 위한 특정 환경에서, 미리 컴파일된 코드 객체를 평가합니다. 이 환경은 전역 변수의 딕셔너리, 지역 변수의 매핑 객체, 인자의 배열, 키워드와 기본값, 키워드 전용 인자의 기본값 딕셔너리 및 셀의 클로저 튜플로 구성됩니다.

PyObject ***PyEval_EvalFrame**(*PyFrameObject* *f)

Return value: New reference. Part of the Stable ABI. 실행 프레임을 평가합니다. 이전 버전과의 호환성을 위한, *PyEval_EvalFrameEx()*의 단순화된 인터페이스입니다.

PyObject ***PyEval_EvalFrameEx**(*PyFrameObject* *f, int throwflag)

Return value: New reference. Part of the Stable ABI. 이것은 파이썬 인터프리트의 메인, 꾸미지 않은 함수입니다. 실행 프레임 *f*와 연관된 코드 객체가 실행됩니다. 필요에 따라 바이트 코드를 해석하고 호출을 실행합니다. 추가 *throwflag* 매개 변수는 대체로 무시할 수 있습니다 - 참이면, 예외가 즉시 발생하도록 합니다; 제너레이터 객체의 `throw()` 메서드에 사용됩니다.

버전 3.4에서 변경: 이 함수는 이제 활성 예외를 조용히 버리지 않았는지 확인하도록 도우려고 디버그 어서션을 포함합니다.

int **PyEval_MergeCompilerFlags**(*PyCompilerFlags* *cf)

이 함수는 현재 평가 프레임의 플래그를 변경하고, 성공하면 참을, 실패하면 거짓을 반환합니다.

int **Py_eval_input**

격리된 표현식을 위한 파이썬 문법의 시작 기호; *Py_CompileString()*과 함께 사용합니다.

int **Py_file_input**

파일이나 다른 소스에서 읽은 문장의 시퀀스를 위한 파이썬 문법의 시작 기호; *Py_CompileString()*과 함께 사용합니다. 임의로 긴 파이썬 소스 코드를 컴파일할 때 사용하는 기호입니다.

int **Py_single_input**

단일 문장을 위한 파이썬 문법의 시작 기호; *Py_CompileString()*과 함께 사용합니다. 대화식 인터프리터 루프에 사용되는 기호입니다.

struct **PyCompilerFlags**

이것은 컴파일러 플래그를 담는 데 사용되는 구조체입니다. 코드가 컴파일되기만 하는 경우 `int flags`로 전달되고, 코드가 실행되는 경우 `PyCompilerFlags *flags`로 전달됩니다. 이 경우, `from __future__ import`는 *flags*를 수정할 수 있습니다.

Whenever `PyCompilerFlags *flags` is `NULL`, *cf_flags* is treated as equal to `0`, and any modification due to `from __future__ import` is discarded.

int **cf_flags**

컴파일러 플래그.

int **cf_feature_version**

*cf_feature_version*은 부 파이썬 버전입니다. `PY_MINOR_VERSION`으로 초기화되어야 합니다.

The field is ignored by default, it is used if and only if `PyCF_ONLY_AST` flag is set in *cf_flags*.

버전 3.8에서 변경: *cf_feature_version* 필드를 추가했습니다.

int **CO_FUTURE_DIVISION**

*flags*에서 이 비트를 설정하면 **PEP 238**에 따라 나누기 연산자 `/`를 "실수 나누기(true division)"로 해석되도록 합니다.

CHAPTER 4

참조 횟수

The functions and macros in this section are used for managing reference counts of Python objects.

Py_ssize_t **Py_REFCNT**(*PyObject* *o)

Get the reference count of the Python object *o*.

Note that the returned value may not actually reflect how many references to the object are actually held. For example, some objects are "immortal" and have a very high refcount that does not reflect the actual number of references. Consequently, do not rely on the returned value to be accurate, other than a value of 0 or 1.

Use the *Py_SET_REFCNT()* function to set an object reference count.

버전 3.10에서 변경: *Py_REFCNT()* is changed to the inline static function.

버전 3.11에서 변경: The parameter type is no longer `const PyObject*`.

void **Py_SET_REFCNT**(*PyObject* *o, *Py_ssize_t* refcnt)

Set the object *o* reference counter to *refcnt*.

Note that this function has no effect on immortal objects.

Added in version 3.9.

버전 3.12에서 변경: Immortal objects are not modified.

void **Py_INCREF**(*PyObject* *o)

Indicate taking a new *strong reference* to object *o*, indicating it is in use and should not be destroyed.

This function is usually used to convert a *borrowed reference* to a *strong reference* in-place. The *Py_NewRef()* function can be used to create a new *strong reference*.

When done using the object, release it by calling *Py_DECREF()*.

The object must not be `NULL`; if you aren' t sure that it isn' t `NULL`, use *Py_XINCREF()*.

Do not expect this function to actually modify *o* in any way. For at least **some objects**, this function has no effect.

버전 3.12에서 변경: Immortal objects are not modified.

void **Py_XINCREF**(*PyObject* *o)

Similar to *Py_INCREF()*, but the object *o* can be `NULL`, in which case this has no effect.

See also *Py_XNewRef()*.

PyObject ***Py_NewRef**(*PyObject* *o)

Part of the Stable ABI *since version 3.10.* Create a new *strong reference* to an object: call *Py_INCREF()* on *o* and return the object *o*.

When the *strong reference* is no longer needed, *Py_DECREF()* should be called on it to release the reference.

The object *o* must not be NULL; use *Py_XNewRef()* if *o* can be NULL.

For example:

```
Py_INCREF(obj);
self->attr = obj;
```

can be written as:

```
self->attr = Py_NewRef(obj);
```

See also *Py_INCREF()*.

Added in version 3.10.

PyObject ***Py_XNewRef**(*PyObject* *o)

Part of the Stable ABI *since version 3.10.* Similar to *Py_NewRef()*, but the object *o* can be NULL.

If the object *o* is NULL, the function just returns NULL.

Added in version 3.10.

void **Py_DECREF**(*PyObject* *o)

Release a *strong reference* to object *o*, indicating the reference is no longer used.

Once the last *strong reference* is released (i.e. the object' s reference count reaches 0), the object' s type' s deallocation function (which must not be NULL) is invoked.

This function is usually used to delete a *strong reference* before exiting its scope.

The object must not be NULL; if you aren' t sure that it isn' t NULL, use *Py_XDECREF()*.

Do not expect this function to actually modify *o* in any way. For at least **some objects**, this function has no effect.

경고

The deallocation function can cause arbitrary Python code to be invoked (e.g. when a class instance with a `__del__()` method is deallocated). While exceptions in such code are not propagated, the executed code has free access to all Python global variables. This means that any object that is reachable from a global variable should be in a consistent state before *Py_DECREF()* is invoked. For example, code to delete an object from a list should copy a reference to the deleted object in a temporary variable, update the list data structure, and then call *Py_DECREF()* for the temporary variable.

버전 3.12에서 변경: Immortal objects are not modified.

void **Py_XDECREF**(*PyObject* *o)

Similar to *Py_DECREF()*, but the object *o* can be NULL, in which case this has no effect. The same warning from *Py_DECREF()* applies here as well.

void **Py_CLEAR**(*PyObject* *o)

Release a *strong reference* for object *o*. The object may be NULL, in which case the macro has no effect; otherwise the effect is the same as for *Py_DECREF()*, except that the argument is also set to NULL. The warning for *Py_DECREF()* does not apply with respect to the object passed because the macro carefully uses a temporary variable and sets the argument to NULL before releasing the reference.

It is a good idea to use this macro whenever releasing a reference to an object that might be traversed during garbage collection.

버전 3.12에서 변경: The macro argument is now only evaluated once. If the argument has side effects, these are no longer duplicated.

void **Py_IncRef**(*PyObject* *o)

Part of the Stable ABI. Indicate taking a new *strong reference* to object *o*. A function version of `Py_XINCREF()`. It can be used for runtime dynamic embedding of Python.

void **Py_DecRef**(*PyObject* *o)

Part of the Stable ABI. Release a *strong reference* to object *o*. A function version of `Py_XDECREF()`. It can be used for runtime dynamic embedding of Python.

Py_SETREF(dst, src)

Macro safely releasing a *strong reference* to object *dst* and setting *dst* to *src*.

As in case of `Py_CLEAR()`, "the obvious" code can be deadly:

```
Py_DECREF(dst);
dst = src;
```

The safe way is:

```
Py_SETREF(dst, src);
```

That arranges to set *dst* to *src* _before_ releasing the reference to the old value of *dst*, so that any code triggered as a side-effect of *dst* getting torn down no longer believes *dst* points to a valid object.

Added in version 3.6.

버전 3.12에서 변경: The macro arguments are now only evaluated once. If an argument has side effects, these are no longer duplicated.

Py_XSETREF(dst, src)

Variant of `Py_SETREF` macro that uses `Py_XDECREF()` instead of `Py_DECREF()`.

Added in version 3.6.

버전 3.12에서 변경: The macro arguments are now only evaluated once. If an argument has side effects, these are no longer duplicated.

CHAPTER 5

예외 처리

The functions described in this chapter will let you handle and raise Python exceptions. It is important to understand some of the basics of Python exception handling. It works somewhat like the POSIX `errno` variable: there is a global indicator (per thread) of the last error that occurred. Most C API functions don' t clear this on success, but will set it to indicate the cause of the error on failure. Most C API functions also return an error indicator, usually `NULL` if they are supposed to return a pointer, or `-1` if they return an integer (exception: the `PyArg_*` functions return `1` for success and `0` for failure).

구체적으로, 에러 표시기는 세 가지 객체 포인터로 구성됩니다: 예외 형, 예외 값 및 트레이스백 객체. 이러한 포인터들은 설정되지 않으면 `NULL`이 될 수 있습니다 (하지만 일부 조합은 금지되어 있습니다, 예를 들어 예외 형이 `NULL`이면 `NULL`이 아닌 트레이스백을 가질 수 없습니다).

호출한 일부 함수가 실패하여 함수가 실패해야 할 때, 일반적으로 에러 표시기를 설정하지 않습니다; 호출된 함수가 이미 설정했습니다. 에러를 처리하고 예외를 지우거나 보유한 모든 리소스(가령 객체 참조나 메모리 할당)를 정리한 후 반환해야 할 책임이 있습니다; 에러를 처리할 준비가 되지 않았을 때 정상적으로 계속되지 않아야 합니다. 에러로 인해 반환하면, 호출자에게 에러가 설정되었음을 알리는 것이 중요합니다. 에러를 처리하지 않거나 신중하게 전파하지 않으면, 파이썬/C API에 대한 추가 호출이 의도한 대로 작동하지 않을 수 있으며 알 수 없는 방식으로 실패할 수 있습니다.

> **참고**
>
> The error indicator is **not** the result of `sys.exc_info()`. The former corresponds to an exception that is not yet caught (and is therefore still propagating), while the latter returns an exception after it is caught (and has therefore stopped propagating).

5.1 인쇄와 지우기

void **PyErr_Clear**()

Part of the Stable ABI. 에러 표시기를 지웁니다. 에러 표시기가 설정되어 있지 않으면 효과가 없습니다.

void **PyErr_PrintEx**(int set_sys_last_vars)

Part of the Stable ABI. 표준 트레이스백을 `sys.stderr`로 인쇄하고 에러 표시기를 지웁니다. 에러가 `SystemExit`가 **아닌 한**, 이 경우에는 트레이스백이 인쇄되지 않고 파이썬 프로세스는 `SystemExit` 인스턴스에 의해 지정된 에러 코드로 종료됩니다.

에러 표시기가 설정된 경우**에만** 이 함수를 호출하십시오. 그렇지 않으면 치명적인 에러가 발생합니다!

If *set_sys_last_vars* is nonzero, the variable `sys.last_exc` is set to the printed exception. For backwards compatibility, the deprecated variables `sys.last_type`, `sys.last_value` and `sys.last_traceback` are also set to the type, value and traceback of this exception, respectively.

버전 3.12에서 변경: The setting of `sys.last_exc` was added.

void **PyErr_Print**()

Part of the Stable ABI. `PyErr_PrintEx(1)`의 별칭.

void **PyErr_WriteUnraisable**(*PyObject* *obj)

Part of the Stable ABI. 현재 예외와 *obj* 인자를 사용하여 `sys.unraisablehook()`을 호출합니다.

This utility function prints a warning message to `sys.stderr` when an exception has been set but it is impossible for the interpreter to actually raise the exception. It is used, for example, when an exception occurs in an `__del__()` method.

The function is called with a single argument *obj* that identifies the context in which the unraisable exception occurred. If possible, the repr of *obj* will be printed in the warning message. If *obj* is `NULL`, only the traceback is printed.

이 함수를 호출할 때 예외를 설정되어 있어야 합니다.

버전 3.4에서 변경: Print a traceback. Print only traceback if *obj* is `NULL`.

버전 3.8에서 변경: Use `sys.unraisablehook()`.

void **PyErr_DisplayException**(*PyObject* *exc)

Part of the Stable ABI *since version 3.12*. Print the standard traceback display of `exc` to `sys.stderr`, including chained exceptions and notes.

Added in version 3.12.

5.2 예외 발생시키기

이 함수들은 현재 스레드의 에러 표시기를 설정하는 데 도움이 됩니다. 편의를 위해, 이러한 함수 중 일부는 항상 `return` 문에서 사용할 `NULL` 포인터를 반환합니다.

void **PyErr_SetString**(*PyObject* *type, const char *message)

Part of the Stable ABI. This is the most common way to set the error indicator. The first argument specifies the exception type; it is normally one of the standard exceptions, e.g. `PyExc_RuntimeError`. You need not create a new *strong reference* to it (e.g. with *`Py_INCREF()`*). The second argument is an error message; it is decoded from `'utf-8'`.

void **PyErr_SetObject**(*PyObject* *type, *PyObject* *value)

Part of the Stable ABI. 이 함수는 *`PyErr_SetString()`*과 유사하지만, 예외의 "값"에 대해 임의의 파이썬 객체를 지정할 수 있습니다.

PyObject ***PyErr_Format**(*PyObject* *exception, const char *format, ...)

Return value: Always NULL. Part of the Stable ABI. 이 함수는 에러 표시기를 설정하고 `NULL`을 반환합니다. *exception*은 파이썬 예외 클래스여야 합니다. *format*과 후속 매개 변수는 에러 메시지를 포맷하는 데 도움이 됩니다; *`PyUnicode_FromFormat()`*에서와 같은 의미와 값을 갖습니다. *format*은 ASCII 인코딩된 문자열입니다.

PyObject ***PyErr_FormatV**(*PyObject* *exception, const char *format, va_list vargs)

Return value: Always NULL. Part of the Stable ABI *since version 3.5*. *`PyErr_Format()`*과 같지만, 가변 개수의 인자 대신 `va_list` 인자를 취합니다.

Added in version 3.5.

void **PyErr_SetNone**(*PyObject* *type)

Part of the Stable ABI. 이것은 `PyErr_SetObject(type, Py_None)`의 줄임 표현입니다.

int **PyErr_BadArgument**()

Part of the Stable ABI. 이것은 `PyErr_SetString(PyExc_TypeError, message)`의 줄임 표현입니다, 여기서 *message*는 잘못된 인자로 내장 연산이 호출되었음을 나타냅니다. 대부분 내부 용입니다.

PyObject ***PyErr_NoMemory**()

Return value: Always NULL. Part of the Stable ABI. 이것은 `PyErr_SetNone(PyExc_MemoryError)`의 줄임 표현입니다; `NULL`을 반환해서 객체 할당 함수는 메모리가 부족할 때 `return PyErr_NoMemory();`라고 쓸 수 있습니다.

PyObject ***PyErr_SetFromErrno**(*PyObject* *type)

Return value: Always NULL. Part of the Stable ABI. This is a convenience function to raise an exception when a C library function has returned an error and set the C variable `errno`. It constructs a tuple object whose first item is the integer `errno` value and whose second item is the corresponding error message (gotten from `strerror()`), and then calls `PyErr_SetObject(type, object)`. On Unix, when the `errno` value is `EINTR`, indicating an interrupted system call, this calls *`PyErr_CheckSignals()`*, and if that set the error indicator, leaves it set to that. The function always returns `NULL`, so a wrapper function around a system call can write `return PyErr_SetFromErrno(type);` when the system call returns an error.

PyObject ***PyErr_SetFromErrnoWithFilenameObject**(*PyObject* *type, *PyObject* *filenameObject)

Return value: Always NULL. Part of the Stable ABI. Similar to *`PyErr_SetFromErrno()`*, with the additional behavior that if *filenameObject* is not `NULL`, it is passed to the constructor of *type* as a third parameter. In the case of `OSError` exception, this is used to define the `filename` attribute of the exception instance.

PyObject ***PyErr_SetFromErrnoWithFilenameObjects**(*PyObject* *type, *PyObject* *filenameObject, *PyObject* *filenameObject2)

Return value: Always NULL. Part of the Stable ABI *since version 3.7.* *`PyErr_SetFromErrnoWithFilenameObject()`*와 유사하지만, 두 개의 파일명을 취하는 함수가 실패할 때 에러를 발생시키기 위해 두 번째 파일명 객체를 취합니다.

Added in version 3.4.

PyObject ***PyErr_SetFromErrnoWithFilename**(*PyObject* *type, const char *filename)

Return value: Always NULL. Part of the Stable ABI. Similar to *`PyErr_SetFromErrnoWithFilenameObject()`*, but the filename is given as a C string. *filename* is decoded from the *filesystem encoding and error handler*.

PyObject ***PyErr_SetFromWindowsErr**(int ierr)

Return value: Always NULL. Part of the Stable ABI *on Windows since version 3.7.* This is a convenience function to raise `OSError`. If called with *ierr* of `0`, the error code returned by a call to `GetLastError()` is used instead. It calls the Win32 function `FormatMessage()` to retrieve the Windows description of error code given by *ierr* or `GetLastError()`, then it constructs a `OSError` object with the `winerror` attribute set to the error code, the `strerror` attribute set to the corresponding error message (gotten from `FormatMessage()`), and then calls `PyErr_SetObject(PyExc_OSError, object)`. This function always returns `NULL`.

Availability: Windows.

PyObject ***PyErr_SetExcFromWindowsErr**(*PyObject* *type, int ierr)

Return value: Always NULL. Part of the Stable ABI *on Windows since version 3.7.* *`PyErr_SetFromWindowsErr()`*와 유사하며, 발생시킬 예외 형을 지정하는 추가 매개 변수가 있습니다.

Availability: Windows.

PyObject ***PyErr_SetFromWindowsErrWithFilename**(int ierr, const char *filename)

Return value: Always NULL. Part of the Stable ABI *on Windows since version 3.7.* Similar to *`PyErr_SetFromWindowsErr()`*, with the additional behavior that if *filename* is not `NULL`, it is decoded from the filesystem encoding (`os.fsdecode()`) and passed to the constructor of `OSError` as a third parameter to be used to define the `filename` attribute of the exception instance.

Availability: Windows.

PyObject ***PyErr_SetExcFromWindowsErrWithFilenameObject**(*PyObject* *type, int ierr, *PyObject* *filename)

Return value: Always NULL. Part of the Stable ABI *on Windows since version 3.7.* Similar to `PyErr_SetExcFromWindowsErr()`, with the additional behavior that if *filename* is not `NULL`, it is passed to the constructor of `OSError` as a third parameter to be used to define the `filename` attribute of the exception instance.

Availability: Windows.

PyObject ***PyErr_SetExcFromWindowsErrWithFilenameObjects**(*PyObject* *type, int ierr, *PyObject* *filename, *PyObject* *filename2)

Return value: Always NULL. Part of the Stable ABI *on Windows since version 3.7.* `PyErr_SetExcFromWindowsErrWithFilenameObject()`와 유사하지만, 두 번째 파일명 객체를 받아들입니다.

Availability: Windows.

Added in version 3.4.

PyObject ***PyErr_SetExcFromWindowsErrWithFilename**(*PyObject* *type, int ierr, const char *filename)

Return value: Always NULL. Part of the Stable ABI *on Windows since version 3.7.* `PyErr_SetFromWindowsErrWithFilename()`와 유사하며, 발생시킬 예외 형을 지정하는 추가 매개 변수가 있습니다.

Availability: Windows.

PyObject ***PyErr_SetImportError**(*PyObject* *msg, *PyObject* *name, *PyObject* *path)

Return value: Always NULL. Part of the Stable ABI *since version 3.7.* `ImportError`를 발생시키는 편의 함수입니다. *msg*는 예외의 메시지 문자열로 설정됩니다. 둘 다 `NULL`이 될 수 있는, *name*과 *path*는 각각 `ImportError`의 `name`과 `path` 어트리뷰트로 설정됩니다.

Added in version 3.3.

PyObject ***PyErr_SetImportErrorSubclass**(*PyObject* *exception, *PyObject* *msg, *PyObject* *name, *PyObject* *path)

Return value: Always NULL. Part of the Stable ABI *since version 3.6.* `PyErr_SetImportError()` 와 매우 비슷하지만, 이 함수는 발생시킬 `ImportError`의 서브 클래스를 지정할 수 있습니다.

Added in version 3.6.

void **PyErr_SyntaxLocationObject**(*PyObject* *filename, int lineno, int col_offset)

현재 예외에 대한 파일(file), 줄(line) 및 오프셋(offset) 정보를 설정합니다. 현재 예외가 `SyntaxError`가 아니면, 추가 어트리뷰트를 설정하여, 예외 인쇄 하위 시스템이 예외가 `SyntaxError`라고 생각하게 합니다.

Added in version 3.4.

void **PyErr_SyntaxLocationEx**(const char *filename, int lineno, int col_offset)

Part of the Stable ABI *since version 3.7.* Like `PyErr_SyntaxLocationObject()`, but *filename* is a byte string decoded from the *filesystem encoding and error handler*.

Added in version 3.2.

void **PyErr_SyntaxLocation**(const char *filename, int lineno)

Part of the Stable ABI. Like `PyErr_SyntaxLocationEx()`, but the *col_offset* parameter is omitted.

void **PyErr_BadInternalCall**()

Part of the Stable ABI. 이것은 `PyErr_SetString(PyExc_SystemError, message)`의 줄임 표현입니다. 여기서 *message*는 내부 연산(예를 들어 파이썬/C API 함수)이 잘못된 인자로 호출되었음을 나타냅니다. 대부분 내부 용입니다.

5.3 경고 발행하기

이 함수를 사용하여 C 코드에서 경고를 발행하십시오. 파이썬 `warnings` 모듈에서 내보낸 유사한 함수를 미러링합니다. 일반적으로 *sys.stderr*에 경고 메시지를 인쇄합니다; 그러나, 사용자가 경고를 에러로 전환하도록 지정했을 수도 있으며, 이 경우 예외가 발생합니다. 경고 장치의 문제로 인해 이 함수가 예외를 발생시키는 것도 가능합니다. 예외가 발생하지 않으면 반환 값은 `0`이고, 예외가 발생하면 `-1`입니다. (경고 메시지가 실제로 인쇄되는지나 예외의 이유를 확인할 수 없습니다; 이것은 의도적입니다.) 예외가 발생하면, 호출자는 정상적인 예외 처리를 수행해야 합니다 (예를 들어, 소유한 참조를 `Py_DECREF()` 하고 에러값을 반환합니다).

int **PyErr_WarnEx**(*PyObject* *category, const char *message, *Py_ssize_t* stack_level)

Part of the Stable ABI. 경고 메시지를 발행합니다. *category* 인자는 경고 범주(아래를 참조하십시오)나 `NULL`입니다; *message* 인자는 UTF-8로 인코딩된 문자열입니다. *stack_level*은 스택 프레임 수를 제공하는 양수입니다; 해당 스택 프레임에서 현재 실행 중인 코드 줄에서 경고가 발생합니다. *stack_level*이 1이면 `PyErr_WarnEx()`를 호출하는 함수, 2는 그 위의 함수, 등등.

경고 범주는 `PyExc_Warning`의 서브 클래스여야 합니다. `PyExc_Warning`은 `PyExc_Exception`의 서브 클래스입니다; 기본 경고 범주는 `PyExc_RuntimeWarning`입니다. 표준 파이썬 경고 범주는 이름이 표준 경고 범주 에 열거된 전역 변수로 제공됩니다.

경고 제어에 대한 자세한 내용은, `warnings` 모듈 설명서와 명령 줄 설명서에서 `-W` 옵션을 참조하십시오. 경고 제어를 위한 C API는 없습니다.

int **PyErr_WarnExplicitObject**(*PyObject* *category, *PyObject* *message, *PyObject* *filename, int lineno, *PyObject* *module, *PyObject* *registry)

Issue a warning message with explicit control over all warning attributes. This is a straightforward wrapper around the Python function `warnings.warn_explicit()`; see there for more information. The *module* and *registry* arguments may be set to `NULL` to get the default effect described there.

Added in version 3.4.

int **PyErr_WarnExplicit**(*PyObject* *category, const char *message, const char *filename, int lineno, const char *module, *PyObject* *registry)

Part of the Stable ABI. Similar to `PyErr_WarnExplicitObject()` except that *message* and *module* are UTF-8 encoded strings, and *filename* is decoded from the *filesystem encoding and error handler*.

int **PyErr_WarnFormat**(*PyObject* *category, *Py_ssize_t* stack_level, const char *format, ...)

Part of the Stable ABI. `PyErr_WarnEx()`와 유사한 함수지만, `PyUnicode_FromFormat()`을 사용하여 경고 메시지를 포맷합니다. *format*은 ASCII 인코딩된 문자열입니다.

Added in version 3.2.

int **PyErr_ResourceWarning**(*PyObject* *source, *Py_ssize_t* stack_level, const char *format, ...)

Part of the Stable ABI *since version 3.6.* Function similar to `PyErr_WarnFormat()`, but *category* is `ResourceWarning` and it passes *source* to `warnings.WarningMessage`.

Added in version 3.6.

5.4 에러 표시기 조회하기

PyObject ***PyErr_Occurred**()

Return value: Borrowed reference. Part of the Stable ABI. Test whether the error indicator is set. If set, return the exception *type* (the first argument to the last call to one of the `PyErr_Set*` functions or to `PyErr_Restore()`). If not set, return `NULL`. You do not own a reference to the return value, so you do not need to `Py_DECREF()` it.

호출자는 GIL을 보유해야 합니다.

> **참고**
>
> 반환 값을 특정 예외와 비교하지 마십시오; 대신 *PyErr_ExceptionMatches()*를 사용하십시오, 아래를 참조하십시오. (클래스 예외의 경우 예외가 클래스 대신 인스턴스이거나, 예상하는 예외의 서브 클래스일 수 있어서 비교는 실패하기 쉽습니다.)

int **PyErr_ExceptionMatches**(*PyObject* *exc)

Part of the Stable ABI. `PyErr_GivenExceptionMatches(PyErr_Occurred(), exc)`와 동등합니다. 예외가 실제로 설정되었을 때만 호출해야 합니다; 예외가 발생하지 않았으면 메모리 액세스 위반이 발생합니다.

int **PyErr_GivenExceptionMatches**(*PyObject* *given, *PyObject* *exc)

Part of the Stable ABI. *given* 예외가 *exc*의 예외 형과 일치하면 참을 반환합니다. *exc*가 클래스 객체이면, *given*이 서브 클래스의 인스턴스일 때도 참을 반환합니다. *exc*가 튜플이면, 튜플에 있는 모든 예외 형(그리고 서브 튜플도 재귀적으로)을 일치를 위해 검색합니다.

PyObject ***PyErr_GetRaisedException**(void)

Return value: New reference. Part of the Stable ABI *since version 3.12.* Return the exception currently being raised, clearing the error indicator at the same time. Return `NULL` if the error indicator is not set.

This function is used by code that needs to catch exceptions, or code that needs to save and restore the error indicator temporarily.

For example:

```
{
   PyObject *exc = PyErr_GetRaisedException();

   /* ... code that might produce other errors ... */

   PyErr_SetRaisedException(exc);
}
```

> **더 보기**
>
> *PyErr_GetHandledException()*, to save the exception currently being handled.

Added in version 3.12.

void **PyErr_SetRaisedException**(*PyObject* *exc)

Part of the Stable ABI *since version 3.12.* Set *exc* as the exception currently being raised, clearing the existing exception if one is set.

> **경고**
>
> This call steals a reference to *exc*, which must be a valid exception.

Added in version 3.12.

void **PyErr_Fetch**(*PyObject* **ptype, *PyObject* **pvalue, *PyObject* **ptraceback)

Part of the Stable ABI. 버전 3.12부터 폐지됨: Use *PyErr_GetRaisedException()* instead.

주소가 전달된 세 개의 변수로 에러 표시기를 꺼냅니다. 에러 표시기가 설정되지 않았으면, 세 변수를 모두 `NULL`로 설정합니다. 설정되었으면, 지워지고 꺼낸 각 객체에 대한 참조를 여러분이 소유합니다. 값과 트레이스백 객체는 형 객체가 그렇지 않을 때도 `NULL`일 수 있습니다.

참고

This function is normally only used by legacy code that needs to catch exceptions or save and restore the error indicator temporarily.

For example:

```
{
   PyObject *type, *value, *traceback;
   PyErr_Fetch(&type, &value, &traceback);

   /* ... code that might produce other errors ... */

   PyErr_Restore(type, value, traceback);
}
```

void **PyErr_Restore**(*PyObject* *type, *PyObject* *value, *PyObject* *traceback)

Part of the Stable ABI. 버전 3.12부터 폐지됨: Use `PyErr_SetRaisedException()` instead.

Set the error indicator from the three objects, *type*, *value*, and *traceback*, clearing the existing exception if one is set. If the objects are `NULL`, the error indicator is cleared. Do not pass a `NULL` type and non-`NULL` value or traceback. The exception type should be a class. Do not pass an invalid exception type or value. (Violating these rules will cause subtle problems later.) This call takes away a reference to each object: you must own a reference to each object before the call and after the call you no longer own these references. (If you don' t understand this, don' t use this function. I warned you.)

참고

This function is normally only used by legacy code that needs to save and restore the error indicator temporarily. Use `PyErr_Fetch()` to save the current error indicator.

void **PyErr_NormalizeException**(*PyObject* **exc, *PyObject* **val, *PyObject* **tb)

Part of the Stable ABI. 버전 3.12부터 폐지됨: Use `PyErr_GetRaisedException()` instead, to avoid any possible de-normalization.

특정 상황에서, 아래의 `PyErr_Fetch()`가 반환하는 값은 "비 정규화"되었을 수 있습니다. 즉, `*exc`는 클래스 객체이지만 `*val`은 같은 클래스의 인스턴스가 아닙니다. 이 함수는 이 경우 클래스를 인스턴스 화하는 데 사용할 수 있습니다. 값이 이미 정규화되어 있으면, 아무 일도 일어나지 않습니다. 지연된 정규화는 성능 향상을 위해 구현됩니다.

참고

This function *does not* implicitly set the `__traceback__` attribute on the exception value. If setting the traceback appropriately is desired, the following additional snippet is needed:

```
if (tb != NULL) {
  PyException_SetTraceback(val, tb);
}
```

PyObject ***PyErr_GetHandledException**(void)

Part of the Stable ABI *since version 3.11.* Retrieve the active exception instance, as would be returned by `sys.exception()`. This refers to an exception that was *already caught*, not to an exception that was freshly raised. Returns a new reference to the exception or `NULL`. Does not modify the interpreter' s exception state.

> **참고**
>
> This function is not normally used by code that wants to handle exceptions. Rather, it can be used when code needs to save and restore the exception state temporarily. Use *PyErr_SetHandledException()* to restore or clear the exception state.

Added in version 3.11.

void **PyErr_SetHandledException**(*PyObject* *exc)

Part of the Stable ABI *since version 3.11.* Set the active exception, as known from `sys.exception()`. This refers to an exception that was *already caught*, not to an exception that was freshly raised. To clear the exception state, pass `NULL`.

> **참고**
>
> This function is not normally used by code that wants to handle exceptions. Rather, it can be used when code needs to save and restore the exception state temporarily. Use *PyErr_GetHandledException()* to get the exception state.

Added in version 3.11.

void **PyErr_GetExcInfo**(*PyObject* **ptype, *PyObject* **pvalue, *PyObject* **ptraceback)

Part of the Stable ABI *since version 3.7.* Retrieve the old-style representation of the exception info, as known from `sys.exc_info()`. This refers to an exception that was *already caught*, not to an exception that was freshly raised. Returns new references for the three objects, any of which may be `NULL`. Does not modify the exception info state. This function is kept for backwards compatibility. Prefer using *PyErr_GetHandledException()*.

> **참고**
>
> 이 함수는 일반적으로 예외를 처리하려는 코드에서 사용되지 않습니다. 오히려, 코드가 예외 상태를 임시로 저장하고 복원해야 할 때 사용할 수 있습니다. 예외 상태를 복원하거나 지우려면 *PyErr_SetExcInfo()*를 사용하십시오.

Added in version 3.3.

void **PyErr_SetExcInfo**(*PyObject* *type, *PyObject* *value, *PyObject* *traceback)

Part of the Stable ABI *since version 3.7.* Set the exception info, as known from `sys.exc_info()`. This refers to an exception that was *already caught*, not to an exception that was freshly raised. This function steals the references of the arguments. To clear the exception state, pass `NULL` for all three arguments. This function is kept for backwards compatibility. Prefer using *PyErr_SetHandledException()*.

> **참고**
>
> 이 함수는 일반적으로 예외를 처리하려는 코드에서 사용되지 않습니다. 오히려, 코드가 예외 상태를 임시로 저장하고 복원해야 할 때 사용할 수 있습니다. 예외 상태를 읽으려면 *PyErr_GetExcInfo()*를 사용하십시오.

Added in version 3.3.

버전 3.11에서 변경: The `type` and `traceback` arguments are no longer used and can be NULL. The interpreter now derives them from the exception instance (the `value` argument). The function still steals references of all three arguments.

5.5 시그널 처리하기

int **PyErr_CheckSignals**()

Part of the Stable ABI. This function interacts with Python' s signal handling.

If the function is called from the main thread and under the main Python interpreter, it checks whether a signal has been sent to the processes and if so, invokes the corresponding signal handler. If the `signal` module is supported, this can invoke a signal handler written in Python.

The function attempts to handle all pending signals, and then returns `0`. However, if a Python signal handler raises an exception, the error indicator is set and the function returns `-1` immediately (such that other pending signals may not have been handled yet: they will be on the next *PyErr_CheckSignals()* invocation).

If the function is called from a non-main thread, or under a non-main Python interpreter, it does nothing and returns `0`.

This function can be called by long-running C code that wants to be interruptible by user requests (such as by pressing Ctrl-C).

> **참고**
>
> The default Python signal handler for `SIGINT` raises the `KeyboardInterrupt` exception.

void **PyErr_SetInterrupt**()

Part of the Stable ABI. Simulate the effect of a `SIGINT` signal arriving. This is equivalent to `PyErr_SetInterruptEx(SIGINT)`.

> **참고**
>
> This function is async-signal-safe. It can be called without the *GIL* and from a C signal handler.

int **PyErr_SetInterruptEx**(int signum)

Part of the Stable ABI *since version 3.10.* Simulate the effect of a signal arriving. The next time *PyErr_CheckSignals()* is called, the Python signal handler for the given signal number will be called.

This function can be called by C code that sets up its own signal handling and wants Python signal handlers to be invoked as expected when an interruption is requested (for example when the user presses Ctrl-C to interrupt an operation).

If the given signal isn' t handled by Python (it was set to `signal.SIG_DFL` or `signal.SIG_IGN`), it will be ignored.

If *signum* is outside of the allowed range of signal numbers, `-1` is returned. Otherwise, `0` is returned. The error indicator is never changed by this function.

> **참고**
>
> This function is async-signal-safe. It can be called without the *GIL* and from a C signal handler.

Added in version 3.10.

int **PySignal_SetWakeupFd**(int fd)

이 유틸리티 함수는 시그널이 수신될 때마다 시그널 번호가 단일 바이트로 기록되는 파일 기술자를 지정합니다. *fd*는 비 블로킹이어야 합니다. 이전의 파일 기술자를 반환합니다.

값 `-1`은 기능을 비활성화합니다; 이것이 초기 상태입니다. 이것은 파이썬의 `signal.set_wakeup_fd()`와 동등하지만, 에러 검사는 없습니다. *fd*는 유효한 파일 기술자여야 합니다. 함수는 메인 스레드에서만 호출되어야 합니다.

버전 3.5에서 변경: 윈도우에서, 함수는 이제 소켓 핸들도 지원합니다.

5.6 예외 클래스

PyObject ***PyErr_NewException**(const char *name, *PyObject* *base, *PyObject* *dict)

Return value: New reference. Part of the Stable ABI. 이 유틸리티 함수는 새 예외 클래스를 만들고 반환합니다. *name* 인자는 새 예외의 이름, `module.classname` 형식의 C 문자열이어야 합니다. *base*와 *dict* 인자는 일반적으로 `NULL`입니다. 이렇게 하면 `Exception`(C에서 `PyExc_Exception`으로 액세스할 수 있습니다)에서 파생된 클래스 객체가 만들어집니다.

The `__module__` attribute of the new class is set to the first part (up to the last dot) of the *name* argument, and the class name is set to the last part (after the last dot). The *base* argument can be used to specify alternate base classes; it can either be only one class or a tuple of classes. The *dict* argument can be used to specify a dictionary of class variables and methods.

PyObject ***PyErr_NewExceptionWithDoc**(const char *name, const char *doc, *PyObject* *base, *PyObject* *dict)

Return value: New reference. Part of the Stable ABI. 새로운 예외 클래스에 독스트링을 쉽게 부여할 수 있다는 점을 제외하면 *PyErr_NewException()*과 같습니다: *doc*이 `NULL`이 아니면, 예외 클래스에 대한 독스트링으로 사용됩니다.

Added in version 3.2.

5.7 예외 객체

PyObject ***PyException_GetTraceback**(*PyObject* *ex)

Return value: New reference. Part of the Stable ABI. Return the traceback associated with the exception as a new reference, as accessible from Python through the `__traceback__` attribute. If there is no traceback associated, this returns `NULL`.

int **PyException_SetTraceback**(*PyObject* *ex, *PyObject* *tb)

Part of the Stable ABI. 예외와 관련된 트레이스백을 *tb*로 설정합니다. 지우려면 `Py_None`을 사용하십시오.

PyObject ***PyException_GetContext**(*PyObject* *ex)

Return value: New reference. Part of the Stable ABI. Return the context (another exception instance during whose handling *ex* was raised) associated with the exception as a new reference, as accessible from Python through the `__context__` attribute. If there is no context associated, this returns `NULL`.

void **PyException_SetContext**(*PyObject* *ex, *PyObject* *ctx)

Part of the Stable ABI. 예외와 연관된 컨텍스트를 *ctx*로 설정합니다. 지우려면 `NULL`을 사용하십시오. *ctx*가 예외 인스턴스인지 확인하는 형 검사는 없습니다. 이것은 *ctx*에 대한 참조를 훔칩니다.

PyObject ***PyException_GetCause**(*PyObject* *ex)

Return value: New reference. Part of the Stable ABI. Return the cause (either an exception instance, or `None`, set by `raise ... from ...`) associated with the exception as a new reference, as accessible from Python through the `__cause__` attribute.

void **PyException_SetCause**(*PyObject* *ex, *PyObject* *cause)

Part of the Stable ABI. Set the cause associated with the exception to *cause*. Use `NULL` to clear it. There is no type check to make sure that *cause* is either an exception instance or `None`. This steals a reference to *cause*.

The `__suppress_context__` attribute is implicitly set to `True` by this function.

PyObject ***PyException_GetArgs**(*PyObject* *ex)

Return value: New reference. Part of the Stable ABI *since version 3.12.* Return `args` of exception *ex*.

void **PyException_SetArgs**(*PyObject* *ex, *PyObject* *args)

Part of the Stable ABI *since version 3.12.* Set `args` of exception *ex* to *args*.

PyObject ***PyUnstable_Exc_PrepReraiseStar**(*PyObject* *orig, *PyObject* *excs)

> This is *Unstable API*. It may change without warning in minor releases.

Implement part of the interpreter's implementation of `except*`. *orig* is the original exception that was caught, and *excs* is the list of the exceptions that need to be raised. This list contains the unhandled part of *orig*, if any, as well as the exceptions that were raised from the `except*` clauses (so they have a different traceback from *orig*) and those that were reraised (and have the same traceback as *orig*). Return the `ExceptionGroup` that needs to be reraised in the end, or `None` if there is nothing to reraise.

Added in version 3.12.

5.8 유니코드 예외 객체

다음 함수는 C에서 유니코드 예외를 만들고 수정하는 데 사용됩니다.

PyObject ***PyUnicodeDecodeError_Create**(const char *encoding, const char *object, *Py_ssize_t* length, *Py_ssize_t* start, *Py_ssize_t* end, const char *reason)

Return value: New reference. Part of the Stable ABI. *encoding*, *object*, *length*, *start*, *end* 및 *reason* 어트리뷰트를 사용하여 `UnicodeDecodeError` 객체를 만듭니다. *encoding*과 *reason*은 UTF-8로 인코딩된 문자열입니다.

PyObject ***PyUnicodeDecodeError_GetEncoding**(*PyObject* *exc)

PyObject ***PyUnicodeEncodeError_GetEncoding**(*PyObject* *exc)

Return value: New reference. Part of the Stable ABI. 주어진 예외 객체의 *encoding* 어트리뷰트를 반환합니다.

PyObject ***PyUnicodeDecodeError_GetObject**(*PyObject* *exc)

PyObject ***PyUnicodeEncodeError_GetObject**(*PyObject* *exc)

PyObject ***PyUnicodeTranslateError_GetObject**(*PyObject* *exc)

Return value: New reference. Part of the Stable ABI. 주어진 예외 객체의 *object* 어트리뷰트를 반환합니다.

int **PyUnicodeDecodeError_GetStart**(*PyObject* *exc, *Py_ssize_t* *start)

int **PyUnicodeEncodeError_GetStart**(*PyObject* *exc, *Py_ssize_t* *start)

int **PyUnicodeTranslateError_GetStart**(*PyObject* *exc, *Py_ssize_t* *start)

Part of the Stable ABI. 주어진 예외 객체의 *start* 어트리뷰트를 가져와서 **start*에 배치합니다. *start*는 `NULL`이 아니어야 합니다. 성공하면 `0`을, 실패하면 `-1`을 반환합니다.

int **PyUnicodeDecodeError_SetStart**(*PyObject* *exc, *Py_ssize_t* start)

int **PyUnicodeEncodeError_SetStart**(*PyObject* *exc, *Py_ssize_t* start)

int **PyUnicodeTranslateError_SetStart**(*PyObject* *exc, *Py_ssize_t* start)

Part of the Stable ABI. 주어진 예외 객체의 *start* 어트리뷰트를 *start*로 설정합니다. 성공하면 `0`을, 실패하면 `-1`을 반환합니다.

int **PyUnicodeDecodeError_GetEnd**(*PyObject* *exc, *Py_ssize_t* *end)

int **PyUnicodeEncodeError_GetEnd**(*PyObject* *exc, *Py_ssize_t* *end)

int **PyUnicodeTranslateError_GetEnd**(*PyObject* *exc, *Py_ssize_t* *end)

Part of the Stable ABI. 주어진 예외 객체의 *end* 어트리뷰트를 가져와서 **end*에 배치합니다. *end*는 `NULL`이 아니어야 합니다. 성공하면 `0`을, 실패하면 `-1`을 반환합니다.

int **PyUnicodeDecodeError_SetEnd**(*PyObject* *exc, *Py_ssize_t* end)

int **PyUnicodeEncodeError_SetEnd**(*PyObject* *exc, *Py_ssize_t* end)

int **PyUnicodeTranslateError_SetEnd**(*PyObject* *exc, *Py_ssize_t* end)

Part of the Stable ABI. 주어진 예외 객체의 *end* 어트리뷰트를 *end*로 설정합니다. 성공하면 `0`을, 실패하면 `-1`을 반환합니다.

PyObject ***PyUnicodeDecodeError_GetReason**(*PyObject* *exc)

PyObject ***PyUnicodeEncodeError_GetReason**(*PyObject* *exc)

PyObject ***PyUnicodeTranslateError_GetReason**(*PyObject* *exc)

Return value: New reference. Part of the Stable ABI. 주어진 예외 객체의 *reason* 어트리뷰트를 반환합니다.

int **PyUnicodeDecodeError_SetReason**(*PyObject* *exc, const char *reason)

int **PyUnicodeEncodeError_SetReason**(*PyObject* *exc, const char *reason)

int **PyUnicodeTranslateError_SetReason**(*PyObject* *exc, const char *reason)

Part of the Stable ABI. 주어진 예외 객체의 *reason* 어트리뷰트를 *reason*으로 설정합니다. 성공하면 `0`을, 실패하면 `-1`을 반환합니다.

5.9 재귀 제어

이 두 함수는 코어와 확장 모듈 모두에서 C 수준에서 안전한 재귀 호출을 수행하는 방법을 제공합니다. 재귀 코드가 반드시 파이썬 코드를 호출하지 않는 경우 필요합니다 (파이썬 코드는 재귀 깊이를 자동으로 추적합니다). 호출 프로토콜이 재귀 처리를 처리하기 때문에 *tp_call* 구현에도 필요하지 않습니다.

int **Py_EnterRecursiveCall**(const char *where)

Part of the Stable ABI *since version 3.9.* 재귀적 C 수준 호출이 막 수행되려고 하는 지점을 표시합니다.

If `USE_STACKCHECK` is defined, this function checks if the OS stack overflowed using *PyOS_CheckStack()*. If this is the case, it sets a `MemoryError` and returns a nonzero value.

그런 다음 함수는 재귀 제한에 도달했는지 확인합니다. 이 경우, `RecursionError`가 설정되고 0이 아닌 값이 반환됩니다. 그렇지 않으면, 0이 반환됩니다.

*where*는 재귀 깊이 제한으로 인한 `RecursionError` 메시지에 이어붙일 `" in instance check"`와 같은 UTF-8 인코딩된 문자열이어야 합니다.

버전 3.9에서 변경: This function is now also available in the *limited API*.

void **Py_LeaveRecursiveCall**(void)

Part of the Stable ABI *since version 3.9.* *Py_EnterRecursiveCall()*을 종료합니다. *Py_EnterRecursiveCall()*의 각 성공적인 호출마다 한 번씩 호출되어야 합니다.

버전 3.9에서 변경: This function is now also available in the *limited API*.

컨테이너형에 대해 *tp_repr*을 올바르게 구현하려면 특별한 재귀 처리가 필요합니다. 스택을 보호하는 것 외에도, *tp_repr*은 순환을 방지하기 위해 객체를 추적해야 합니다. 다음 두 함수는 이 기능을 쉽게 만듭니다. 사실상, 이들은 `reprlib.recursive_repr()`에 대한 C 동등물입니다.

int **Py_ReprEnter**(*PyObject* *object)

Part of the Stable ABI. 순환을 감지하기 위해 *tp_repr* 구현 시작 시 호출됩니다.

객체가 이미 처리되었으면, 함수는 양의 정수를 반환합니다. 이 경우 *tp_repr* 구현은 순환을 나타내는 문자열 객체를 반환해야 합니다. 예를 들어, `dict` 객체는 `{...}`를 반환하고 `list` 객체는 `[...]`를 반환합니다.

재귀 제한에 도달하면 함수는 음의 정수를 반환합니다. 이 경우 *tp_repr* 구현은 일반적으로 `NULL`을 반환해야 합니다.

그렇지 않으면, 함수는 0을 반환하고 *tp_repr* 구현은 정상적으로 계속될 수 있습니다.

void **Py_ReprLeave**(*PyObject* *object)

Part of the Stable ABI. *Py_ReprEnter()*를 종료합니다. 0을 반환하는 *Py_ReprEnter()* 호출마다 한 번씩 호출해야 합니다.

5.10 표준 예외

All standard Python exceptions are available as global variables whose names are `PyExc_` followed by the Python exception name. These have the type *PyObject**; they are all class objects. For completeness, here are all the variables:

C 이름	파이썬 이름	노트
PyExc_BaseException	BaseException	[1]
PyExc_Exception	Exception	Page 64, 1
PyExc_ArithmeticError	ArithmeticError	Page 64, 1
PyExc_AssertionError	AssertionError	
PyExc_AttributeError	AttributeError	
PyExc_BlockingIOError	BlockingIOError	
PyExc_BrokenPipeError	BrokenPipeError	
PyExc_BufferError	BufferError	
PyExc_ChildProcessError	ChildProcessError	
PyExc_ConnectionAbortedErro	ConnectionAbortedError	
PyExc_ConnectionError	ConnectionError	
PyExc_ConnectionRefusedErro	ConnectionRefusedError	
PyExc_ConnectionResetError	ConnectionResetError	
PyExc_EOFError	EOFError	
PyExc_FileExistsError	FileExistsError	
PyExc_FileNotFoundError	FileNotFoundError	
PyExc_FloatingPointError	FloatingPointError	
PyExc_GeneratorExit	GeneratorExit	
PyExc_ImportError	ImportError	
PyExc_IndentationError	IndentationError	
PyExc_IndexError	IndexError	
PyExc_InterruptedError	InterruptedError	
PyExc_IsADirectoryError	IsADirectoryError	
PyExc_KeyError	KeyError	
PyExc_KeyboardInterrupt	KeyboardInterrupt	
PyExc_LookupError	LookupError	Page 64, 1
PyExc_MemoryError	MemoryError	
PyExc_ModuleNotFoundError	ModuleNotFoundError	
PyExc_NameError	NameError	
PyExc_NotADirectoryError	NotADirectoryError	
PyExc_NotImplementedError	NotImplementedError	
PyExc_OSError	OSError	Page 64, 1
PyExc_OverflowError	OverflowError	
PyExc_PermissionError	PermissionError	
PyExc_ProcessLookupError	ProcessLookupError	
PyExc_RecursionError	RecursionError	
PyExc_ReferenceError	ReferenceError	
PyExc_RuntimeError	RuntimeError	
PyExc_StopAsyncIteration	StopAsyncIteration	
PyExc_StopIteration	StopIteration	
PyExc_SyntaxError	SyntaxError	
PyExc_SystemError	SystemError	
PyExc_SystemExit	SystemExit	
PyExc_TabError	TabError	
PyExc_TimeoutError	TimeoutError	
PyExc_TypeError	TypeError	
PyExc_UnboundLocalError	UnboundLocalError	
PyExc_UnicodeDecodeError	UnicodeDecodeError	
PyExc_UnicodeEncodeError	UnicodeEncodeError	

다음 페이지에 계속

표 1 – 이전 페이지에서 계속

C 이름	파이썬 이름	노트
PyExc_UnicodeError	UnicodeError	
PyExc_UnicodeTranslateError	UnicodeTranslateError	
PyExc_ValueError	ValueError	
PyExc_ZeroDivisionError	ZeroDivisionError	

Added in version 3.3: PyExc_BlockingIOError, PyExc_BrokenPipeError, PyExc_ChildProcessError, PyExc_ConnectionError, PyExc_ConnectionAbortedError, PyExc_ConnectionRefusedError, PyExc_ConnectionResetError, PyExc_FileExistsError, PyExc_FileNotFoundError, PyExc_InterruptedError, PyExc_IsADirectoryError, PyExc_NotADirectoryError, PyExc_PermissionError, PyExc_ProcessLookupError 및 PyExc_TimeoutError 는 **PEP 3151**을 따라 도입되었습니다.

Added in version 3.5: PyExc_StopAsyncIteration 과 PyExc_RecursionError.

Added in version 3.6: PyExc_ModuleNotFoundError.

다음은 PyExc_OSError 에 대한 호환성 별칭입니다:

C 이름	노트
PyExc_EnvironmentError	
PyExc_IOError	
PyExc_WindowsError	[2]

버전 3.3에서 변경: 이러한 별칭은 별도의 예외 형이었습니다.

노트:

5.11 표준 경고 범주

All standard Python warning categories are available as global variables whose names are PyExc_ followed by the Python exception name. These have the type *PyObject**; they are all class objects. For completeness, here are all the variables:

C 이름	파이썬 이름	노트
PyExc_Warning	Warning	[3]
PyExc_BytesWarning	BytesWarning	
PyExc_DeprecationWarning	DeprecationWarning	
PyExc_FutureWarning	FutureWarning	
PyExc_ImportWarning	ImportWarning	
PyExc_PendingDeprecationWarning	PendingDeprecationWarning	
PyExc_ResourceWarning	ResourceWarning	
PyExc_RuntimeWarning	RuntimeWarning	
PyExc_SyntaxWarning	SyntaxWarning	
PyExc_UnicodeWarning	UnicodeWarning	
PyExc_UserWarning	UserWarning	

Added in version 3.2: PyExc_ResourceWarning.

노트:

[1] 이것은 다른 표준 예외에 대한 베이스 클래스입니다.
[2] 윈도우에서만 정의됩니다; 전 처리기 매크로 MS_WINDOWS 가 정의되었는지 테스트하여 이를 사용하는 코드를 보호하십시오.
[3] 이것은 다른 표준 경고 범주의 베이스 클래스입니다.

CHAPTER 6

유틸리티

이 장의 함수들은 C 코드의 플랫폼 간 호환성 개선, C에서 파이썬 모듈 사용, 함수 인자의 구문 분석 및 C 값으로부터 파이썬 값을 구성하는 것에 이르기까지 다양한 유틸리티 작업을 수행합니다.

6.1 운영 체제 유틸리티

PyObject ***PyOS_FSPath**(*PyObject* *path)

Return value: New reference. Part of the Stable ABI *since version 3.6.* Return the file system representation for *path*. If the object is a `str` or `bytes` object, then a new *strong reference* is returned. If the object implements the `os.PathLike` interface, then `__fspath__()` is returned as long as it is a `str` or `bytes` object. Otherwise `TypeError` is raised and `NULL` is returned.

Added in version 3.6.

int **Py_FdIsInteractive**(FILE *fp, const char *filename)

Return true (nonzero) if the standard I/O file *fp* with name *filename* is deemed interactive. This is the case for files for which `isatty(fileno(fp))` is true. If the *`PyConfig.interactive`* is non-zero, this function also returns true if the *filename* pointer is `NULL` or if the name is equal to one of the strings `'<stdin>'` or `'???'`.

This function must not be called before Python is initialized.

void **PyOS_BeforeFork**()

Part of the Stable ABI *on platforms with fork() since version 3.7.* 프로세스 포크 전에 내부 상태를 준비하는 함수. `fork()` 나 현재 프로세스를 복제하는 유사한 함수를 호출하기 전에 호출해야 합니다. `fork()` 가 정의된 시스템에서만 사용 가능합니다.

> **경고**
>
> C `fork()` 호출은 ("메인" 인터프리터의) "메인" 스레드에서만 이루어져야 합니다. `PyOS_BeforeFork()` 도 마찬가지입니다.

Added in version 3.7.

void **PyOS_AfterFork_Parent**()

Part of the Stable ABI *on platforms with fork() since version 3.7.* 프로세스 포크 후 일부 내부 상태를 갱신하는 함수. 프로세스 복제가 성공했는지와 관계없이, `fork()` 나 현재 프로세스를 복제하는 유사한

함수를 호출한 후 부모 프로세스에서 호출해야 합니다. `fork()` 가 정의된 시스템에서만 사용 가능합니다.

> **경고**
>
> C `fork()` 호출은 ("메인" 인터프리터의) "메인" 스레드에서만 이루어져야 합니다. `PyOS_AfterFork_Parent()` 도 마찬가지입니다.

Added in version 3.7.

void **PyOS_AfterFork_Child**()

Part of the Stable ABI *on platforms with fork() since version 3.7.* 프로세스 포크 후 내부 인터프리터 상태를 갱신하는 함수. `fork()` 나 현재 프로세스를 복제하는 유사한 함수를 호출한 후, 프로세스가 파이썬 인터프리터를 다시 호출할 가능성이 있으면 자식 프로세스에서 호출해야 합니다. `fork()` 가 정의된 시스템에서만 사용 가능합니다.

> **경고**
>
> C `fork()` 호출은 ("메인" 인터프리터의) "메인" 스레드에서만 이루어져야 합니다. `PyOS_AfterFork_Child()` 도 마찬가지입니다.

Added in version 3.7.

> **더 보기**
>
> `os.register_at_fork()` 를 사용하면 *PyOS_BeforeFork()*, *PyOS_AfterFork_Parent()* 및 *PyOS_AfterFork_Child()* 에서 호출될 사용자 정의 파이썬 함수를 등록할 수 있습니다.

void **PyOS_AfterFork**()

Part of the Stable ABI *on platforms with fork()*. 프로세스 포크 후 일부 내부 상태를 갱신하는 함수; 파이썬 인터프리터가 계속 사용된다면 새로운 프로세스에서 호출되어야 합니다. 새 실행 파일이 새 프로세스에 로드되면, 이 함수를 호출할 필요가 없습니다.

버전 3.7부터 폐지됨: 이 함수는 *PyOS_AfterFork_Child()* 로 대체되었습니다.

int **PyOS_CheckStack**()

Part of the Stable ABI *on platforms with USE_STACKCHECK since version 3.7.* Return true when the interpreter runs out of stack space. This is a reliable check, but is only available when `USE_STACKCHECK` is defined (currently on certain versions of Windows using the Microsoft Visual C++ compiler). `USE_STACKCHECK` will be defined automatically; you should never change the definition in your own code.

typedef void (***PyOS_sighandler_t**)(int)

Part of the Stable ABI.

PyOS_sighandler_t **PyOS_getsig**(int i)

Part of the Stable ABI. Return the current signal handler for signal *i*. This is a thin wrapper around either `sigaction()` or `signal()`. Do not call those functions directly!

PyOS_sighandler_t **PyOS_setsig**(int i, *PyOS_sighandler_t* h)

Part of the Stable ABI. Set the signal handler for signal *i* to be *h*; return the old signal handler. This is a thin wrapper around either `sigaction()` or `signal()`. Do not call those functions directly!

wchar_t ***Py_DecodeLocale**(const char *arg, size_t *size)

Part of the Stable ABI *since version 3.7.*

경고

This function should not be called directly: use the *PyConfig* API with the *PyConfig_SetBytesString()* function which ensures that *Python is preinitialized*.

This function must not be called before *Python is preinitialized* and so that the LC_CTYPE locale is properly configured: see the *Py_PreInitialize()* function.

Decode a byte string from the *filesystem encoding and error handler*. If the error handler is surrogateescape error handler, undecodable bytes are decoded as characters in range U+DC80..U+DCFF; and if a byte sequence can be decoded as a surrogate character, the bytes are escaped using the surrogateescape error handler instead of decoding them.

새로 할당된 와이드 문자(wide character) 문자열에 대한 포인터를 반환합니다, 메모리를 해제하려면 *PyMem_RawFree()*를 사용하십시오. size가 `NULL`이 아니면, 널 문자를 제외한 와이드 문자 수를 `*size`에 기록합니다.

디코딩 에러나 메모리 할당 에러 시 `NULL`을 반환합니다. *size*가 `NULL`이 아니면, 메모리 에러 시 `*size`가 `(size_t)-1`로 설정되고, 디코딩 에러 시 `(size_t)-2`로 설정됩니다.

The *filesystem encoding and error handler* are selected by *PyConfig_Read()*: see *filesystem_encoding* and *filesystem_errors* members of *PyConfig*.

C 라이브러리에 버그가 없으면, 디코딩 에러가 발생하지 않아야 합니다.

문자열을 바이트열로 다시 인코딩하려면 *Py_EncodeLocale()* 함수를 사용하십시오.

더 보기

*PyUnicode_DecodeFSDefaultAndSize()*와 *PyUnicode_DecodeLocaleAndSize()* 함수.

Added in version 3.5.

버전 3.7에서 변경: The function now uses the UTF-8 encoding in the Python UTF-8 Mode.

버전 3.8에서 변경: The function now uses the UTF-8 encoding on Windows if *PyPreConfig.legacy_windows_fs_encoding* is zero;

char *__Py_EncodeLocale__(const wchar_t *text, size_t *error_pos)

Part of the Stable ABI *since version 3.7.* Encode a wide character string to the *filesystem encoding and error handler*. If the error handler is surrogateescape error handler, surrogate characters in the range U+DC80..U+DCFF are converted to bytes 0x80..0xFF.

Return a pointer to a newly allocated byte string, use *PyMem_Free()* to free the memory. Return `NULL` on encoding error or memory allocation error.

error_pos가 `NULL`이 아니면, `*error_pos`는 성공 시 `(size_t)-1`로 설정되고, 인코딩 에러 시 유효하지 않은 문자의 인덱스로 설정됩니다.

The *filesystem encoding and error handler* are selected by *PyConfig_Read()*: see *filesystem_encoding* and *filesystem_errors* members of *PyConfig*.

바이트열을 와이드 문자 문자열로 다시 디코딩하려면 *Py_DecodeLocale()* 함수를 사용하십시오.

경고

This function must not be called before *Python is preinitialized* and so that the LC_CTYPE locale is properly configured: see the *Py_PreInitialize()* function.

> **더 보기**
>
> *PyUnicode_EncodeFSDefault()*와 *PyUnicode_EncodeLocale()* 함수.

Added in version 3.5.

버전 3.7에서 변경: The function now uses the UTF-8 encoding in the Python UTF-8 Mode.

버전 3.8에서 변경: The function now uses the UTF-8 encoding on Windows if *PyPreConfig.legacy_windows_fs_encoding* is zero.

6.2 시스템 함수

sys 모듈의 기능을 C 코드에서 액세스 할 수 있게 하는 유틸리티 함수입니다. 모두 내부 스레드 상태 구조체에 포함된 현재 인터프리터 스레드의 sys 모듈의 딕셔너리에 작동합니다.

PyObject ***PySys_GetObject**(const char *name)

Return value: Borrowed reference. Part of the Stable ABI. sys 모듈에서 객체 *name*을 반환하거나, 존재하지 않으면 예외를 설정하지 않고 NULL을 반환합니다.

int **PySys_SetObject**(const char *name, *PyObject* *v)

Part of the Stable ABI. *v*가 *NULL*이 아닌 한 sys 모듈의 *name*을 *v*로 설정합니다. NULL이면 *name*은 sys 모듈에서 삭제됩니다. 성공하면 0, 에러 시 -1을 반환합니다.

void **PySys_ResetWarnOptions**()

Part of the Stable ABI. sys.warnoptions를 빈 리스트로 재설정합니다. 이 함수는 *Py_Initialize()* 이전에 호출할 수 있습니다.

void **PySys_AddWarnOption**(const wchar_t *s)

Part of the Stable ABI. This API is kept for backward compatibility: setting *PyConfig.warnoptions* should be used instead, see *Python Initialization Configuration*.

*s*를 sys.warnoptions에 추가합니다. 경고 필터 리스트에 영향을 주려면 *Py_Initialize()* 이전에 이 함수를 호출해야 합니다.

버전 3.11부터 폐지됨.

void **PySys_AddWarnOptionUnicode**(*PyObject* *unicode)

Part of the Stable ABI. This API is kept for backward compatibility: setting *PyConfig.warnoptions* should be used instead, see *Python Initialization Configuration*.

*unicode*를 sys.warnoptions에 추가합니다.

참고: 이 함수는 현재 CPython 구현 외부에서 사용할 수 없습니다. 효과가 있으려면 *Py_Initialize()*에서 warnings를 묵시적으로 임포트 하기 전에 호출해야 하지만, 유니코드 객체를 만들도록 허락할 수 있을 만큼 런타임이 충분히 초기화되기 전에는 호출할 수 없기 때문입니다.

버전 3.11부터 폐지됨.

void **PySys_SetPath**(const wchar_t *path)

Part of the Stable ABI. This API is kept for backward compatibility: setting *PyConfig.module_search_paths* and *PyConfig.module_search_paths_set* should be used instead, see *Python Initialization Configuration*.

sys.path를 플랫폼의 검색 경로 구분자(유닉스에서는 :, 윈도우에서는 ;)로 구분된 경로 리스트여야 하는 *path*에서 찾은 경로의 리스트 객체로 설정합니다.

버전 3.11부터 폐지됨.

void **PySys_WriteStdout**(const char *format, ...)

Part of the Stable ABI. *format*으로 기술되는 출력 문자열을 sys.stdout에 기록합니다. 잘림이 발생하더라도 예외는 발생하지 않습니다 (아래를 참조하십시오).

*format*은 포맷된 출력 문자열의 총 크기를 1000바이트 이하로 제한해야 합니다 – 1000바이트 이후에는, 출력 문자열이 잘립니다. 특히, 이것은 무제한 "%s" 포맷이 있어서는 안 됨을 의미합니다; "%.<N>s"를 사용하여 제한해야 합니다, 여기서 <N>은 <N>에 다른 포맷된 텍스트의 최대 크기를 더할 때 1000바이트를 초과하지 않도록 계산된 십진수입니다. 또한 "%f"도 주의하십시오, 아주 큰 숫자는 수백 자리를 인쇄할 수 있습니다.

문제가 발생하거나, `sys.stdout`가 설정되어 있지 않으면, 포맷된 메시지는 실제(C 수준) *stdout*에 기록됩니다.

void **PySys_WriteStderr**(const char *format, ...)

Part of the Stable ABI. *PySys_WriteStdout()*과 같지만, 대신 `sys.stderr`이나 *stderr*에 씁니다.

void **PySys_FormatStdout**(const char *format, ...)

Part of the Stable ABI. PySys_WriteStdout()과 유사한 함수이지만, 메시지를 *PyUnicode_FromFormatV()*를 사용하여 포맷하고 메시지를 임의의 길이로 자르지 않습니다.

Added in version 3.2.

void **PySys_FormatStderr**(const char *format, ...)

Part of the Stable ABI. *PySys_FormatStdout()*과 같지만, 대신 `sys.stderr`이나 *stderr*에 씁니다.

Added in version 3.2.

void **PySys_AddXOption**(const wchar_t *s)

Part of the Stable ABI *since version 3.7.* This API is kept for backward compatibility: setting *PyConfig.xoptions* should be used instead, see *Python Initialization Configuration*.

*s*를 `-X` 옵션 집합으로 구문 분석하고 *PySys_GetXOptions()*가 반환하는 현재 옵션 매핑에 추가합니다. 이 함수는 *Py_Initialize()* 이전에 호출할 수 있습니다.

Added in version 3.2.

버전 3.11부터 폐지됨.

PyObject ***PySys_GetXOptions**()

Return value: Borrowed reference. Part of the Stable ABI *since version 3.7.* `sys._xoptions`와 유사하게, `-X` 옵션의 현재 딕셔너리를 반환합니다. 에러가 발생하면, `NULL`이 반환되고 예외가 설정됩니다.

Added in version 3.2.

int **PySys_Audit**(const char *event, const char *format, ...)

모든 활성 훅으로 감사 이벤트를 발생시킵니다. 성공 시 0을 반환하고 실패 시 예외를 설정하여 0이 아닌 값을 반환합니다.

훅이 추가되었으면, *format*과 기타 인자를 사용하여 전달할 튜플을 구성합니다. `N` 외에도, *Py_BuildValue()*에서 사용된 것과 같은 포맷 문자를 사용할 수 있습니다. 빌드된 값이 튜플이 아니면, 단일 요소 튜플에 추가됩니다. (`N` 포맷 옵션은 참조를 소비하지만, 이 함수에 대한 인자가 소비될지를 알 방법이 없기 때문에, 사용하면 참조 누수가 발생할 수 있습니다.)

Note that `#` format characters should always be treated as *Py_ssize_t*, regardless of whether `PY_SSIZE_T_CLEAN` was defined.

`sys.audit()`은 파이썬 코드와 동일한 기능을 수행합니다.

Added in version 3.8.

버전 3.8.2에서 변경: Require *Py_ssize_t* for `#` format characters. Previously, an unavoidable deprecation warning was raised.

int **PySys_AddAuditHook**(*Py_AuditHookFunction* hook, void *userData)

Append the callable *hook* to the list of active auditing hooks. Return zero on success and non-zero on failure. If the runtime has been initialized, also set an error on failure. Hooks added through this API are called for all interpreters created by the runtime.

userData 포인터는 훅 함수로 전달됩니다. 훅 함수는 다른 런타임에서 호출될 수 있어서, 이 포인터는 파이썬 상태를 직접 참조하면 안 됩니다.

이 함수는 *Py_Initialize()* 이전에 호출해도 안전합니다. 런타임 초기화 후 호출되면, 기존 감사 훅에 알리고 `Exception`에서 서브 클래싱 된 에러를 발생 시켜 조용히 연산을 중단할 수 있습니다 (다른 에러는 억제되지(silenced) 않습니다).

The hook function is always called with the GIL held by the Python interpreter that raised the event.

감사에 대한 자세한 설명은 **PEP 578**을 참조하십시오. 이벤트를 발생시키는 런타임과 표준 라이브러리의 함수는 감사 이벤트 표에 나열되어 있습니다. 자세한 내용은 각 함수 설명서에 있습니다.

If the interpreter is initialized, this function raises an auditing event `sys.addaudithook` with no arguments. If any existing hooks raise an exception derived from `Exception`, the new hook will not be added and the exception is cleared. As a result, callers cannot assume that their hook has been added unless they control all existing hooks.

typedef int (***Py_AuditHookFunction**)(const char *event, *PyObject* *args, void *userData)

The type of the hook function. *event* is the C string event argument passed to *PySys_Audit()*. *args* is guaranteed to be a *PyTupleObject*. *userData* is the argument passed to PySys_AddAuditHook().

Added in version 3.8.

6.3 프로세스 제어

void **Py_FatalError**(const char *message)

Part of the Stable ABI. Print a fatal error message and kill the process. No cleanup is performed. This function should only be invoked when a condition is detected that would make it dangerous to continue using the Python interpreter; e.g., when the object administration appears to be corrupted. On Unix, the standard C library function `abort()` is called which will attempt to produce a `core` file.

`Py_LIMITED_API` 매크로가 정의되어 있지 않은 한, `Py_FatalError()` 함수는 현재 함수의 이름을 자동으로 로그 하는 매크로로 대체됩니다.

버전 3.9에서 변경: 함수 이름을 자동으로 로그 합니다.

void **Py_Exit**(int status)

Part of the Stable ABI. 현재 프로세스를 종료합니다. 이것은 *Py_FinalizeEx()*를 호출한 다음 표준 C 라이브러리 함수 `exit(status)`를 호출합니다. *Py_FinalizeEx()*가 에러를 표시하면, 종료 상태는 120으로 설정됩니다.

버전 3.6에서 변경: 파이널리제이션에서의 에러가 더는 무시되지 않습니다.

int **Py_AtExit**(void (*func)())

Part of the Stable ABI. *Py_FinalizeEx()*가 호출할 정리 함수를 등록합니다. 정리 함수는 인자 없이 호출되며 값을 반환하지 않아야 합니다. 최대 32개의 정리 함수를 등록할 수 있습니다. 등록이 성공하면, *Py_AtExit()*는 `0`을 반환합니다; 실패하면 `-1`을 반환합니다. 마지막에 등록된 정리 함수가 먼저 호출됩니다. 각 정리 함수는 최대 한 번 호출됩니다. 정리 함수 전에 파이썬의 내부 파이널리제이션이 완료되기 때문에, *func*에서 파이썬 API를 호출하면 안 됩니다.

6.4 모듈 임포트 하기

PyObject ***PyImport_ImportModule**(const char *name)

Return value: New reference. Part of the Stable ABI. This is a wrapper around *PyImport_Import()* which takes a `const char*` as an argument instead of a *PyObject**.

PyObject ***PyImport_ImportModuleNoBlock**(const char *name)

Return value: New reference. Part of the Stable ABI. 이 함수는 *PyImport_ImportModule()*의 폐지된 별칭입니다.

버전 3.3에서 변경: 이 기능은 다른 스레드가 임포트 잠금을 보유한 경우 즉시 실패했었습니다. 그러나 파이썬 3.3에서는, 잠금 방식이 대부분의 목적에서 모듈 단위 잠금으로 전환되었기 때문에, 이 함수의 특수한 동작은 더는 필요하지 않습니다.

PyObject ***PyImport_ImportModuleEx**(const char *name, *PyObject* *globals, *PyObject* *locals, *PyObject* *fromlist)

Return value: New reference. 모듈을 임포트 합니다. 내장 파이썬 함수 `__import__()`를 통해 가장 잘 설명할 수 있습니다.

반환 값은 임포트 된 모듈이나 최상위 패키지에 대한 새로운 참조, 또는 실패 시 예외가 설정된 `NULL`입니다. `__import__()`와 마찬가지로, 비어 있지 않은 *fromlist*가 제공되지 않는 한, 패키지의 서브 모듈이 요청되었을 때의 반환 값은 최상위 패키지입니다.

임포트 실패는 `PyImport_ImportModule()`처럼 불완전한 모듈 객체를 제거합니다.

PyObject ***PyImport_ImportModuleLevelObject**(*PyObject* *name, *PyObject* *globals, *PyObject* *locals, *PyObject* *fromlist, int level)

Return value: New reference. *Part of the* Stable ABI *since version 3.7.* 모듈을 임포트 합니다. 표준 `__import__()` 함수가 이 함수를 직접 호출하기 때문에, 내장 파이썬 함수 `__import__()`를 통해 가장 잘 설명할 수 있습니다.

반환 값은 임포트 된 모듈이나 최상위 패키지에 대한 새로운 참조, 또는 실패 시 예외가 설정된 `NULL`입니다. `__import__()`와 마찬가지로, 비어 있지 않은 *fromlist*가 제공되지 않는 한, 패키지의 서브 모듈이 요청되었을 때의 반환 값은 최상위 패키지입니다.

Added in version 3.3.

PyObject ***PyImport_ImportModuleLevel**(const char *name, *PyObject* *globals, *PyObject* *locals, *PyObject* *fromlist, int level)

Return value: New reference. *Part of the* Stable ABI. `PyImport_ImportModuleLevelObject()`와 비슷하지만, name은 유니코드 객체 대신 UTF-8로 인코딩된 문자열입니다.

버전 3.3에서 변경: *level*의 음수 값은 더는 허용되지 않습니다.

PyObject ***PyImport_Import**(*PyObject* *name)

Return value: New reference. *Part of the* Stable ABI. 이것은 현재 "임포트 훅 함수"를 호출하는 고수준 인터페이스입니다 (명시적인 *level* 0을 사용하는데, 절대 임포트를 뜻합니다). 현재 전역의 `__builtins__`에 있는 `__import__()` 함수를 호출합니다. 이는 현재 환경에 설치된 임포트 훅을 사용하여 임포트가 수행됨을 의미합니다.

이 함수는 항상 절대 임포트를 사용합니다.

PyObject ***PyImport_ReloadModule**(*PyObject* *m)

Return value: New reference. *Part of the* Stable ABI. 모듈을 다시 로드(reload)합니다. 다시 로드된 모듈에 대한 참조를 반환하거나, 실패 시 예외가 설정된 `NULL`을 반환합니다 (이때 모듈은 여전히 존재합니다).

PyObject ***PyImport_AddModuleObject**(*PyObject* *name)

Return value: Borrowed reference. *Part of the* Stable ABI *since version 3.7.* 모듈 이름에 해당하는 모듈 객체를 반환합니다. *name* 인자는 `package.module` 형식일 수 있습니다. 먼저 모듈 딕셔너리에 있는지 확인하고, 없으면 새로 만들어 모듈 딕셔너리에 삽입합니다. 실패 시 예외를 설정하고 `NULL`을 반환합니다.

> **참고**
>
> 이 함수는 모듈을 로드하거나 임포트 하지 않습니다; 모듈이 아직 로드되지 않았으면, 빈 모듈 객체를 얻게 됩니다. 모듈을 임포트 하려면 `PyImport_ImportModule()`이나 그 변형 중 하나를 사용하십시오. *name*에서 점으로 구분된 이름으로 암시된 패키지 구조는 이미 존재하지 않는다면 만들어지지 않습니다.

Added in version 3.3.

PyObject ***PyImport_AddModule**(const char *name)

Return value: Borrowed reference. *Part of the* Stable ABI. `PyImport_AddModuleObject()`와 비슷하지만, name은 유니코드 객체 대신 UTF-8로 인코딩된 문자열입니다.

PyObject ***PyImport_ExecCodeModule**(const char *name, *PyObject* *co)

Return value: New reference. Part of the Stable ABI. Given a module name (possibly of the form `package.module`) and a code object read from a Python bytecode file or obtained from the built-in function `compile()`, load the module. Return a new reference to the module object, or `NULL` with an exception set if an error occurred. *name* is removed from `sys.modules` in error cases, even if *name* was already in `sys.modules` on entry to *PyImport_ExecCodeModule()*. Leaving incompletely initialized modules in `sys.modules` is dangerous, as imports of such modules have no way to know that the module object is an unknown (and probably damaged with respect to the module author' s intents) state.

The module' s `__spec__` and `__loader__` will be set, if not set already, with the appropriate values. The spec' s loader will be set to the module' s `__loader__` (if set) and to an instance of `SourceFileLoader` otherwise.

The module' s `__file__` attribute will be set to the code object' s `co_filename`. If applicable, `__cached__` will also be set.

이 함수는 이미 임포트 되었다면 모듈을 다시 로드합니다. 모듈을 다시 로드하는 의도된 방법은 *PyImport_ReloadModule()*을 참조하십시오.

*name*이 `package.module` 형식의 점으로 구분된 이름을 가리키면, 이미 만들어지지 않은 패키지 구조는 여전히 만들어지지 않습니다.

*PyImport_ExecCodeModuleEx()*와 *PyImport_ExecCodeModuleWithPathnames()*도 참조하십시오.

버전 3.12에서 변경: The setting of `__cached__` and `__loader__` is deprecated. See `ModuleSpec` for alternatives.

PyObject ***PyImport_ExecCodeModuleEx**(const char *name, *PyObject* *co, const char *pathname)

Return value: New reference. Part of the Stable ABI. Like *PyImport_ExecCodeModule()*, but the `__file__` attribute of the module object is set to *pathname* if it is non-`NULL`.

*PyImport_ExecCodeModuleWithPathnames()*도 참조하십시오.

PyObject ***PyImport_ExecCodeModuleObject**(*PyObject* *name, *PyObject* *co, *PyObject* *pathname, *PyObject* *cpathname)

Return value: New reference. Part of the Stable ABI *since version 3.7.* Like *PyImport_ExecCodeModuleEx()*, but the `__cached__` attribute of the module object is set to *cpathname* if it is non-`NULL`. Of the three functions, this is the preferred one to use.

Added in version 3.3.

버전 3.12에서 변경: Setting `__cached__` is deprecated. See `ModuleSpec` for alternatives.

PyObject ***PyImport_ExecCodeModuleWithPathnames**(const char *name, *PyObject* *co, const char *pathname, const char *cpathname)

Return value: New reference. Part of the Stable ABI. *PyImport_ExecCodeModuleObject()*와 유사하지만, *name*, *pathname* 및 *cpathname*은 UTF-8로 인코딩된 문자열입니다. *pathname*의 값이 `NULL`로 설정된 경우 어떤 값이 *cpathname*에서 와야하는지 알아내려고 합니다.

Added in version 3.2.

버전 3.3에서 변경: Uses `imp.source_from_cache()` in calculating the source path if only the bytecode path is provided.

버전 3.12에서 변경: No longer uses the removed `imp` module.

long **PyImport_GetMagicNumber**()

Part of the Stable ABI. 파이썬 바이트 코드 파일(일명 `.pyc` 파일)의 매직 번호(magic number)를 반환합니다. 매직 번호는 바이트 코드 파일의 처음 4바이트에 리틀 엔디안 바이트 순서로 존재해야 합니다. 에러 시 `-1`을 반환합니다.

버전 3.3에서 변경: 실패 시 `-1`을 반환합니다.

const char *PyImport_GetMagicTag()

Part of the Stable ABI. **PEP 3147** 형식 파이썬 바이트 코드 파일 이름의 매직 태그 문자열을 반환합니다. `sys.implementation.cache_tag`의 값은 신뢰할 수 있고 이 함수 대신 사용해야 함에 유의하십시오.

Added in version 3.2.

PyObject *PyImport_GetModuleDict()

Return value: Borrowed reference. Part of the Stable ABI. 모듈 관리에 사용되는 딕셔너리(일명 `sys.modules`)를 반환합니다. 이것은 인터프리터마다 존재하는 변수임에 유의하십시오.

PyObject *PyImport_GetModule(*PyObject* *name)

Return value: New reference. Part of the Stable ABI *since version 3.8.* 주어진 이름으로 이미 임포트 된 모듈을 반환합니다. 모듈이 아직 임포트 되지 않았다면 `NULL`을 반환하지만 에러는 설정하지 않습니다. 조회에 실패하면 `NULL`을 반환하고 에러를 설정합니다.

Added in version 3.7.

PyObject *PyImport_GetImporter(*PyObject* *path)

Return value: New reference. Part of the Stable ABI. Return a finder object for a `sys.path`/`pkg.__path__` item *path*, possibly by fetching it from the `sys.path_importer_cache` dict. If it wasn't yet cached, traverse `sys.path_hooks` until a hook is found that can handle the path item. Return `None` if no hook could; this tells our caller that the *path based finder* could not find a finder for this path item. Cache the result in `sys.path_importer_cache`. Return a new reference to the finder object.

int PyImport_ImportFrozenModuleObject(*PyObject* *name)

Part of the Stable ABI *since version 3.7.* *name*이라는 이름의 프로즌 모듈(frozen module)을 로드합니다. 성공하면 `1`을, 모듈을 찾지 못하면 `0`을, 초기화에 실패하면 예외를 설정하고 `-1`을 반환합니다. 로드가 성공할 때 임포트 된 모듈에 액세스하려면 *PyImport_ImportModule()*을 사용하십시오. (잘못된 이름에 주의하십시오 — 이 함수는 모듈이 이미 임포트 되었을 때 다시 로드합니다.)

Added in version 3.3.

버전 3.4에서 변경: `__file__` 어트리뷰트는 더는 모듈에 설정되지 않습니다.

int PyImport_ImportFrozenModule(const char *name)

Part of the Stable ABI. *PyImport_ImportFrozenModuleObject()*와 비슷하지만, name은 유니코드 객체 대신 UTF-8로 인코딩된 문자열입니다.

struct _frozen

이것은 **freeze** 유틸리티(파이썬 소스 배포의 `Tools/freeze/`를 참조하십시오)가 생성한 프로즌 모듈 디스크립터를 위한 구조체 형 정의입니다. `Include/import.h`에 있는 정의는 다음과 같습니다:

```
struct _frozen {
    const char *name;
    const unsigned char *code;
    int size;
    bool is_package;
};
```

버전 3.11에서 변경: The new `is_package` field indicates whether the module is a package or not. This replaces setting the `size` field to a negative value.

const struct *_frozen* *PyImport_FrozenModules

This pointer is initialized to point to an array of *_frozen* records, terminated by one whose members are all `NULL` or zero. When a frozen module is imported, it is searched in this table. Third-party code could play tricks with this to provide a dynamically created collection of frozen modules.

int PyImport_AppendInittab(const char *name, *PyObject* *(*initfunc)(void))

Part of the Stable ABI. 기존의 내장 모듈 테이블에 단일 모듈을 추가합니다. 이것은 *PyImport_ExtendInittab()*을 감싸는 편리한 래퍼인데, 테이블을 확장할 수 없으면 `-1`을 반환합니다. 새 모듈은 *name*이라는 이름으로 임포트 될 수 있으며, *initfunc* 함수를 처음 시도한 임포트에서 호출되는 초기화 함수로 사용합니다. *Py_Initialize()* 전에 호출해야 합니다.

struct **_inittab**

Structure describing a single entry in the list of built-in modules. Programs which embed Python may use an array of these structures in conjunction with *PyImport_ExtendInittab()* to provide additional built-in modules. The structure consists of two members:

const char ***name**

The module name, as an ASCII encoded string.

PyObject *(***initfunc**)(void)

Initialization function for a module built into the interpreter.

int **PyImport_ExtendInittab**(struct *_inittab* *newtab)

Add a collection of modules to the table of built-in modules. The *newtab* array must end with a sentinel entry which contains `NULL` for the *name* field; failure to provide the sentinel value can result in a memory fault. Returns `0` on success or `-1` if insufficient memory could be allocated to extend the internal table. In the event of failure, no modules are added to the internal table. This must be called before *Py_Initialize()*.

If Python is initialized multiple times, *PyImport_AppendInittab()* or *PyImport_ExtendInittab()* must be called before each Python initialization.

6.5 데이터 마샬링 지원

이러한 루틴은 C 코드가 `marshal` 모듈과 같은 데이터 형식을 사용하여 직렬화된 객체로 작업 할 수 있도록 합니다. 직렬화 형식으로 데이터를 쓰는 함수와 데이터를 다시 읽는 데 사용할 수 있는 추가 함수가 있습니다. 마샬링 된 데이터를 저장하는 데 사용되는 파일은 바이너리 모드로 열어야 합니다.

숫자 값은 최하위 바이트가 먼저 저장됩니다.

The module supports two versions of the data format: version 0 is the historical version, version 1 shares interned strings in the file, and upon unmarshalling. Version 2 uses a binary format for floating-point numbers. `Py_MARSHAL_VERSION` indicates the current file format (currently 2).

void **PyMarshal_WriteLongToFile**(long value, FILE *file, int version)

Marshal a `long` integer, *value*, to *file*. This will only write the least-significant 32 bits of *value*; regardless of the size of the native `long` type. *version* indicates the file format.

This function can fail, in which case it sets the error indicator. Use *PyErr_Occurred()* to check for that.

void **PyMarshal_WriteObjectToFile**(*PyObject* *value, FILE *file, int version)

파이썬 객체 *value*를 *file*로 마샬합니다. *version*은 파일 형식을 나타냅니다.

This function can fail, in which case it sets the error indicator. Use *PyErr_Occurred()* to check for that.

PyObject ***PyMarshal_WriteObjectToString**(*PyObject* *value, int version)

Return value: New reference. 마샬된 *value* 표현을 포함한 바이트열 객체를 반환합니다. *version*은 파일 형식을 나타냅니다.

다음 함수를 사용하면 마샬된 값을 다시 읽을 수 있습니다.

long **PyMarshal_ReadLongFromFile**(FILE *file)

Return a C `long` from the data stream in a `FILE*` opened for reading. Only a 32-bit value can be read in using this function, regardless of the native size of `long`.

에러 시, 적절한 예외(`EOFError`)를 설정하고 `-1`을 반환합니다.

int **PyMarshal_ReadShortFromFile**(FILE *file)

Return a C `short` from the data stream in a `FILE*` opened for reading. Only a 16-bit value can be read in using this function, regardless of the native size of `short`.

에러 시, 적절한 예외(`EOFError`)를 설정하고 `-1`을 반환합니다.

PyObject ***PyMarshal_ReadObjectFromFile**(FILE *file)

Return value: New reference. Return a Python object from the data stream in a `FILE*` opened for reading.

에러 시, 적절한 예외(`EOFError`, `ValueError` 또는 `TypeError`)를 설정하고 `NULL`을 반환합니다.

PyObject ***PyMarshal_ReadLastObjectFromFile**(FILE *file)

Return value: New reference. Return a Python object from the data stream in a `FILE*` opened for reading. Unlike *PyMarshal_ReadObjectFromFile()*, this function assumes that no further objects will be read from the file, allowing it to aggressively load file data into memory so that the de-serialization can operate from data in memory rather than reading a byte at a time from the file. Only use these variant if you are certain that you won' t be reading anything else from the file.

에러 시, 적절한 예외(`EOFError`, `ValueError` 또는 `TypeError`)를 설정하고 `NULL`을 반환합니다.

PyObject ***PyMarshal_ReadObjectFromString**(const char *data, *Py_ssize_t* len)

Return value: New reference. *data*가 가리키는 *len* 바이트를 포함하는 바이트 버퍼의 데이터 스트림에서 파이썬 객체를 반환합니다.

에러 시, 적절한 예외(`EOFError`, `ValueError` 또는 `TypeError`)를 설정하고 `NULL`을 반환합니다.

6.6 인자 구문 분석과 값 구축

These functions are useful when creating your own extension functions and methods. Additional information and examples are available in extending-index.

설명된 이러한 함수 중 처음 세 개인 *PyArg_ParseTuple()*, *PyArg_ParseTupleAndKeywords()* 및 *PyArg_Parse()*는 모두 예상 인자에 관한 사항을 함수에 알리는 데 사용되는 포맷 문자열(*format strings*)을 사용합니다. 포맷 문자열은 이러한 각 함수에 대해 같은 문법을 사용합니다.

6.6.1 인자 구문 분석

포맷 문자열은 0개 이상의 "포맷 단위(format units)"로 구성됩니다. 포맷 단위는 하나의 파이썬 객체를 설명합니다; 일반적으로 단일 문자나 괄호로 묶인 포맷 단위 시퀀스입니다. 몇 가지 예외를 제외하고, 괄호로 묶인 시퀀스가 아닌 포맷 단위는 일반적으로 이러한 함수에 대한 단일 주소 인자에 대응합니다. 다음 설명에서, 인용된(quoted) 형식은 포맷 단위입니다; (둥근) 괄호 안의 항목은 포맷 단위와 일치하는 파이썬 객체 형입니다; [대괄호] 안의 항목은 주소를 전달해야 하는 C 변수의 형입니다.

문자열과 버퍼

이러한 포맷을 사용하면 연속적인 메모리 청크로 객체에 액세스 할 수 있습니다. 반환된 유니코드나 바이트열 영역에 대한 원시 저장소를 제공할 필요가 없습니다.

달리 명시되지 않는 한, 버퍼는 NUL로 종료되지 않습니다.

There are three ways strings and buffers can be converted to C:

- Formats such as `y*` and `s*` fill a *Py_buffer* structure. This locks the underlying buffer so that the caller can subsequently use the buffer even inside a *Py_BEGIN_ALLOW_THREADS* block without the risk of mutable data being resized or destroyed. As a result, **you have to call** *PyBuffer_Release()* after you have finished processing the data (or in any early abort case).
- The `es`, `es#`, `et` and `et#` formats allocate the result buffer. **You have to call** *PyMem_Free()* after you have finished processing the data (or in any early abort case).
- Other formats take a `str` or a read-only *bytes-like object*, such as `bytes`, and provide a `const char *` pointer to its buffer. In this case the buffer is "borrowed": it is managed by the corresponding Python object, and shares the lifetime of this object. You won' t have to release any memory yourself.

  To ensure that the underlying buffer may be safely borrowed, the object' s *PyBufferProcs.bf_releasebuffer* field must be `NULL`. This disallows common mutable objects such as `bytearray`, but also some read-only objects such as `memoryview` of `bytes`.

Besides this `bf_releasebuffer` requirement, there is no check to verify whether the input object is immutable (e.g. whether it would honor a request for a writable buffer, or whether another thread can mutate the data).

> **참고**
>
> For all `#` variants of formats (`s#`, `y#`, etc.), the macro `PY_SSIZE_T_CLEAN` must be defined before including `Python.h`. On Python 3.9 and older, the type of the length argument is *`Py_ssize_t`* if the `PY_SSIZE_T_CLEAN` macro is defined, or int otherwise.

`s` (**`str`**) [**const char ***]
유니코드 객체를 문자열에 대한 C 포인터로 변환합니다. 기존 문자열에 대한 포인터는 여러분이 주소를 전달한 문자 포인터 변수에 저장됩니다. C 문자열은 NUL로 종료됩니다. 파이썬 문자열은 내장된 널 코드 포인트를 포함하지 않아야 합니다; 그렇다면 `ValueError` 예외가 발생합니다. 유니코드 객체는 `'utf-8'` 인코딩을 사용하여 C 문자열로 변환됩니다. 이 변환이 실패하면, `UnicodeError`가 발생합니다.

> **참고**
>
> 이 포맷은 바이트열류 객체를 받아들이지 않습니다. 파일 시스템 경로를 받아들이고 이를 C 문자열로 변환하려면, *`PyUnicode_FSConverter()`*를 *converter*로 `O&` 포맷을 사용하는 것이 좋습니다.

버전 3.5에서 변경: 이전에는, 파이썬 문자열에서 내장된 널 코드 포인트가 발견되면 `TypeError`가 발생했습니다.

`s*` (**`str`** 또는 바이트열류 객체) [**Py_buffer**]
이 포맷은 바이트열류 객체뿐만 아니라 유니코드 객체를 받아들입니다. 호출자가 제공한 *`Py_buffer`* 구조체를 채웁니다. 이 경우 결과 C 문자열은 내장된 NUL 바이트를 포함할 수 있습니다. 유니코드 객체는 `'utf-8'` 인코딩을 사용하여 C 문자열로 변환됩니다.

`s#` (**`str`**, **read-only *bytes-like object***) [**const char ***, ***`Py_ssize_t`***]
Like `s*`, except that it provides a *borrowed buffer*. The result is stored into two C variables, the first one a pointer to a C string, the second one its length. The string may contain embedded null bytes. Unicode objects are converted to C strings using `'utf-8'` encoding.

`z` (**`str`** 또는 **`None`**) [**const char ***]
`s`와 비슷하지만, 파이썬 객체가 `None`일 수도 있는데, 이 경우 C 포인터가 `NULL`로 설정됩니다.

`z*` (**`str`**, 바이트열류 객체 또는 **`None`**) [**Py_buffer**]
`s*`와 비슷하지만, 파이썬 객체는 `None`일 수도 있습니다, 이 경우 *`Py_buffer`* 구조체의 `buf` 멤버가 `NULL`로 설정됩니다.

`z#` (**`str`**, **read-only *bytes-like object*** or **`None`**) [**const char ***, ***`Py_ssize_t`***]
`s#`와 비슷하지만, 파이썬 객체는 `None`일 수도 있습니다, 이 경우 C 포인터가 `NULL`로 설정됩니다.

`y` (읽기 전용 바이트열류 객체) [**const char ***]
This format converts a bytes-like object to a C pointer to a *borrowed* character string; it does not accept Unicode objects. The bytes buffer must not contain embedded null bytes; if it does, a `ValueError` exception is raised.

버전 3.5에서 변경: 이전에는, 바이트열 버퍼에서 내장 널 바이트가 발견되면 `TypeError`가 발생했습니다.

`y*` (바이트열류 객체) [**Py_buffer**]
`s*`의 이 변형은 유니코드 객체가 아니라 바이트열류 객체만 받아들입니다. **바이너리 데이터를 받아들이는 권장 방법입니다.**

`y#` (**read-only *bytes-like object***) [**const char ***, ***`Py_ssize_t`***]
`s#`의 이 변형은 유니코드 객체가 아니라 바이트열류 객체만 받아들입니다.

`S` (**`bytes`**) [**PyBytesObject ***]
Requires that the Python object is a `bytes` object, without attempting any conversion. Raises `TypeError` if

the object is not a bytes object. The C variable may also be declared as *PyObject*.

Y (bytearray) [PyByteArrayObject *]
Requires that the Python object is a bytearray object, without attempting any conversion. Raises TypeError if the object is not a bytearray object. The C variable may also be declared as *PyObject*.

U (str) [PyObject *]
Requires that the Python object is a Unicode object, without attempting any conversion. Raises TypeError if the object is not a Unicode object. The C variable may also be declared as *PyObject*.

w* (읽기-쓰기 바이트열류 객체) [Py_buffer]
이 포맷은 읽기-쓰기 버퍼 인터페이스를 구현하는 모든 객체를 허용합니다. 호출자가 제공한 *Py_buffer* 구조체를 채웁니다. 버퍼에는 내장 널 바이트가 포함될 수 있습니다. 호출자는 버퍼로 할 일을 마치면 *PyBuffer_Release()*를 호출해야 합니다.

es (str) [const char *encoding, char **buffer]
s의 이 변형은 유니코드를 문자 버퍼로 인코딩하는 데 사용됩니다. 내장 NUL 바이트가 포함되지 않은 인코딩된 데이터에 대해서만 작동합니다.

This format requires two arguments. The first is only used as input, and must be a const char* which points to the name of an encoding as a NUL-terminated string, or NULL, in which case 'utf-8' encoding is used. An exception is raised if the named encoding is not known to Python. The second argument must be a char**; the value of the pointer it references will be set to a buffer with the contents of the argument text. The text will be encoded in the encoding specified by the first argument.

*PyArg_ParseTuple()*은 필요한 크기의 버퍼를 할당하고, 인코딩된 데이터를 이 버퍼에 복사하고 새로 할당된 스토리지를 참조하도록 **buffer*를 조정합니다. 호출자는 사용 후에 할당된 버퍼를 해제하기 위해 *PyMem_Free()*를 호출해야 합니다.

et (str, bytes 또는 bytearray) [const char *encoding, char **buffer]
바이트 문자열 객체를 다시 코딩하지 않고 통과시킨다는 점을 제외하면 es와 같습니다. 대신, 구현은 바이트 문자열 객체가 매개 변수로 전달된 인코딩을 사용한다고 가정합니다.

es# (str) [const char *encoding, char **buffer, *Py_ssize_t* *buffer_length]
s#의 이 변형은 유니코드를 문자 버퍼로 인코딩하는 데 사용됩니다. es 포맷과 달리, 이 변형은 NUL 문자를 포함하는 입력 데이터를 허용합니다.

It requires three arguments. The first is only used as input, and must be a const char* which points to the name of an encoding as a NUL-terminated string, or NULL, in which case 'utf-8' encoding is used. An exception is raised if the named encoding is not known to Python. The second argument must be a char**; the value of the pointer it references will be set to a buffer with the contents of the argument text. The text will be encoded in the encoding specified by the first argument. The third argument must be a pointer to an integer; the referenced integer will be set to the number of bytes in the output buffer.

두 가지 작동 모드가 있습니다:

**buffer*가 NULL 포인터를 가리키면, 함수는 필요한 크기의 버퍼를 할당하고, 이 버퍼로 인코딩된 데이터를 복사하고 **buffer*를 새로 할당된 스토리지를 참조하도록 설정합니다. 호출자는 사용 후 할당된 버퍼를 해제하기 위해 *PyMem_Free()*를 호출해야 합니다.

**buffer*가 NULL이 아닌 포인터를 가리키면 (이미 할당된 버퍼), *PyArg_ParseTuple()*은 이 위치를 버퍼로 사용하고 **buffer_length*의 초깃값을 버퍼 크기로 해석합니다. 그런 다음 인코딩된 데이터를 버퍼에 복사하고 NUL 종료합니다. 버퍼가 충분히 크지 않으면, ValueError가 설정됩니다.

두 경우 모두, **buffer_length*는 후행 NUL 바이트를 제외한 인코딩된 데이터의 길이로 설정됩니다.

et# (str, bytes or bytearray) [const char *encoding, char **buffer, *Py_ssize_t* *buffer_length]
바이트 문자열 객체를 다시 코딩하지 않고 통과시킨다는 점을 제외하면 es#와 같습니다. 대신, 구현은 바이트 문자열 객체가 매개 변수로 전달된 인코딩을 사용한다고 가정합니다.

버전 3.12에서 변경: u, u#, Z, and Z# are removed because they used a legacy Py_UNICODE* representation.

숫자

These formats allow representing Python numbers or single characters as C numbers. Formats that require `int`, `float` or `complex` can also use the corresponding special methods `__index__()`, `__float__()` or `__complex__()` to convert the Python object to the required type.

For signed integer formats, `OverflowError` is raised if the value is out of range for the C type. For unsigned integer formats, no range checking is done — the most significant bits are silently truncated when the receiving field is too small to receive the value.

`b` (**`int`**) [**unsigned char**]
Convert a nonnegative Python integer to an unsigned tiny integer, stored in a C `unsigned char`.

`B` (**`int`**) [**unsigned char**]
Convert a Python integer to a tiny integer without overflow checking, stored in a C `unsigned char`.

`h` (**`int`**) [**short int**]
Convert a Python integer to a C `short int`.

`H` (**`int`**) [**unsigned short int**]
Convert a Python integer to a C `unsigned short int`, without overflow checking.

`i` (**`int`**) [**int**]
Convert a Python integer to a plain C `int`.

`I` (**`int`**) [**unsigned int**]
Convert a Python integer to a C `unsigned int`, without overflow checking.

`l` (**`int`**) [**long int**]
Convert a Python integer to a C `long int`.

`k` (**`int`**) [**unsigned long**]
Convert a Python integer to a C `unsigned long` without overflow checking.

`L` (**`int`**) [**long long**]
Convert a Python integer to a C `long long`.

`K` (**`int`**) [**unsigned long long**]
Convert a Python integer to a C `unsigned long long` without overflow checking.

`n` (**`int`**) [*`Py_ssize_t`*]
파이썬 정수를 C *`Py_ssize_t`*로 변환합니다.

`c` (길이 1의 **`bytes`** 또는 **`bytearray`**) [**char**]
Convert a Python byte, represented as a `bytes` or `bytearray` object of length 1, to a C `char`.

버전 3.3에서 변경: `bytearray` 객체를 허용합니다.

`C` (길이 1의 **`str`**) [**int**]
Convert a Python character, represented as a `str` object of length 1, to a C `int`.

`f` (**`float`**) [**float**]
Convert a Python floating-point number to a C `float`.

`d` (**`float`**) [**double**]
Convert a Python floating-point number to a C `double`.

`D` (**`complex`**) [**Py_complex**]
파이썬 복소수를 C *`Py_complex`* 구조체로 변환합니다.

기타 객체

`O` (**object**) [**PyObject ***]
Store a Python object (without any conversion) in a C object pointer. The C program thus receives the actual object that was passed. A new *strong reference* to the object is not created (i.e. its reference count is not increased). The pointer stored is not `NULL`.

O! (**object**) [*typeobject*, **PyObject** *]
Store a Python object in a C object pointer. This is similar to O, but takes two C arguments: the first is the address of a Python type object, the second is the address of the C variable (of type *PyObject**) into which the object pointer is stored. If the Python object does not have the required type, TypeError is raised.

O& (**object**) [*converter*, *address*]
Convert a Python object to a C variable through a *converter* function. This takes two arguments: the first is a function, the second is the address of a C variable (of arbitrary type), converted to void*. The *converter* function in turn is called as follows:

```
status = converter(object, address);
```

where *object* is the Python object to be converted and *address* is the void* argument that was passed to the PyArg_Parse* function. The returned *status* should be 1 for a successful conversion and 0 if the conversion has failed. When the conversion fails, the *converter* function should raise an exception and leave the content of *address* unmodified. If the *converter* returns Py_CLEANUP_SUPPORTED, it may get called a second time if the argument parsing eventually fails, giving the converter a chance to release any memory that it had already allocated. In this second call, the *object* parameter will be NULL; *address* will have the same value as in the original call.

Examples of converters: *PyUnicode_FSConverter()* and *PyUnicode_FSDecoder()*.

버전 3.1에서 변경: Py_CLEANUP_SUPPORTED was added.

p (**bool**) [**int**]
전달된 값의 논리값을 테스트(불리언 **p**redicate)하고 결과를 동등한 C 참/거짓 정숫값으로 변환합니다. 표현식이 참이면 int를 1로, 거짓이면 0으로 설정합니다. 모든 유효한 파이썬 값을 허용합니다. 파이썬이 논리값을 테스트하는 방법에 대한 자세한 내용은 truth를 참조하십시오.

Added in version 3.3.

(items) (**tuple**) [*matching-items*]
객체는 길이가 *items*에 있는 포맷 단위의 수인 파이썬 시퀀스여야 합니다. C 인자들은 *items*의 개별 포맷 단위에 대응해야 합니다. 시퀀스의 포맷 단위는 중첩될 수 있습니다.

몇 가지 다른 문자는 포맷 문자열에서 의미가 있습니다. 중첩된 괄호 안에서는 나타날 수 없습니다. 그들은:

|
파이썬 인자 리스트의 나머지 인자가 선택 사항임을 나타냅니다. 선택적 인자에 해당하는 C 변수는 기본값으로 초기화되어야 합니다 — 선택적 인자가 지정되지 않을 때, *PyArg_ParseTuple()*은 해당 C 변수의 내용을 건드리지 않습니다.

$
PyArg_ParseTupleAndKeywords() 전용: 파이썬 인자 리스트의 나머지 인자가 키워드 전용임을 나타냅니다. 현재, 모든 키워드 전용 인자는 선택적 인자여야 하므로, | 는 항상 포맷 문자열에서 $ 앞에 지정되어야 합니다.

Added in version 3.3.

:
포맷 단위 리스트는 여기에서 끝납니다; 콜론 뒤의 문자열은 에러 메시지에서 함수 이름으로 사용됩니다 (*PyArg_ParseTuple()*이 발생시키는 예외의 "연관된 값").

;
포맷 단위 리스트는 여기에서 끝납니다; 세미콜론 뒤의 문자열은 기본 에러 메시지의 에러 메시지 대신 에러 메시지로 사용됩니다. : 와 ; 는 서로를 배제합니다.

Note that any Python object references which are provided to the caller are *borrowed* references; do not release them (i.e. do not decrement their reference count)!

이러한 함수에 전달되는 추가 인자는 포맷 문자열에 의해 형이 결정되는 변수의 주소여야 합니다; 이들은 입력 튜플의 값을 저장하는 데 사용됩니다. 위의 포맷 단위 리스트에서 설명된 대로, 이러한 매개 변수가 입력값으로 사용되는 몇 가지 경우가 있습니다; 이 경우 해당 포맷 단위에 대해 지정된 것과 일치해야 합니다.

For the conversion to succeed, the *arg* object must match the format and the format must be exhausted. On success, the `PyArg_Parse*` functions return true, otherwise they return false and raise an appropriate exception. When the `PyArg_Parse*` functions fail due to conversion failure in one of the format units, the variables at the addresses corresponding to that and the following format units are left untouched.

API 함수

int **PyArg_ParseTuple**(*PyObject* *args, const char *format, ...)

Part of the Stable ABI. 위치 매개 변수만 지역 변수로 취하는 함수의 매개 변수를 구문 분석합니다. 성공하면 참을 반환합니다; 실패하면, 거짓을 반환하고 적절한 예외를 발생시킵니다.

int **PyArg_VaParse**(*PyObject* *args, const char *format, va_list vargs)

Part of the Stable ABI. 가변 개수의 인자가 아닌 va_list를 받아들인다는 점을 제외하면, *PyArg_ParseTuple()*과 동일합니다.

int **PyArg_ParseTupleAndKeywords**(*PyObject* *args, *PyObject* *kw, const char *format, char *keywords[], ...)

Part of the Stable ABI. 위치와 키워드 매개 변수를 모두 지역 변수로 취하는 함수의 매개 변수를 구문 분석합니다. *keywords* 인자는 키워드 매개 변수 이름의 `NULL`-종료 배열입니다. 빈 이름은 위치-전용 매개 변수를 나타냅니다. 성공하면 참을 반환합니다; 실패하면, 거짓을 반환하고 적절한 예외를 발생시킵니다.

버전 3.6에서 변경: 위치-전용 매개 변수에 대한 지원이 추가되었습니다.

int **PyArg_VaParseTupleAndKeywords**(*PyObject* *args, *PyObject* *kw, const char *format, char *keywords[], va_list vargs)

Part of the Stable ABI. 가변 개수의 인자가 아닌 va_list를 받아들인다는 점을 제외하면, *PyArg_ParseTupleAndKeywords()* 와 동일합니다.

int **PyArg_ValidateKeywordArguments**(*PyObject**)

Part of the Stable ABI. 키워드 인자 딕셔너리의 키가 문자열인지 확인합니다. *PyArg_ParseTupleAndKeywords()* 가 사용되지 않는 경우에만 필요합니다, 여기서는 이미 이 검사를 수행하기 때문입니다.

Added in version 3.2.

int **PyArg_Parse**(*PyObject* *args, const char *format, ...)

Part of the Stable ABI. "이전 스타일" 함수의 인자 리스트를 분해하는 데 사용되는 함수 — 이들은 파이썬 3에서 제거된 `METH_OLDARGS` 매개 변수 구문 분석 메서드를 사용하는 함수입니다. 새 코드에서 매개 변수 구문 분석에 사용하는 것은 권장되지 않고, 표준 인터프리터에 있는 대부분의 코드는 더는 이런 목적으로 사용하지 않도록 수정되었습니다. 그러나, 다른 튜플을 분해하는 편리한 방법으로 남아 있으며, 그런 목적으로 계속 사용할 수 있습니다.

int **PyArg_UnpackTuple**(*PyObject* *args, const char *name, *Py_ssize_t* min, *Py_ssize_t* max, ...)

Part of the Stable ABI. A simpler form of parameter retrieval which does not use a format string to specify the types of the arguments. Functions which use this method to retrieve their parameters should be declared as *METH_VARARGS* in function or method tables. The tuple containing the actual parameters should be passed as *args*; it must actually be a tuple. The length of the tuple must be at least *min* and no more than *max*; *min* and *max* may be equal. Additional arguments must be passed to the function, each of which should be a pointer to a *PyObject** variable; these will be filled in with the values from *args*; they will contain *borrowed references*. The variables which correspond to optional parameters not given by *args* will not be filled in; these should be initialized by the caller. This function returns true on success and false if *args* is not a tuple or contains the wrong number of elements; an exception will be set if there was a failure.

This is an example of the use of this function, taken from the sources for the `_weakref` helper module for weak references:

```
static PyObject *
weakref_ref(PyObject *self, PyObject *args)
{
    PyObject *object;
    PyObject *callback = NULL;
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    PyObject *result = NULL;

    if (PyArg_UnpackTuple(args, "ref", 1, 2, &object, &callback)) {
        result = PyWeakref_NewRef(object, callback);
    }
    return result;
}
```

이 예제에서 *PyArg_UnpackTuple()*에 대한 호출은 *PyArg_ParseTuple()*에 대한 다음 호출과 전적으로 동등합니다:

```
PyArg_ParseTuple(args, "O|O:ref", &object, &callback)
```

6.6.2 값 구축

PyObject ***Py_BuildValue**(const char *format, ...)

Return value: New reference. Part of the Stable ABI. Create a new value based on a format string similar to those accepted by the `PyArg_Parse*` family of functions and a sequence of values. Returns the value or `NULL` in the case of an error; an exception will be raised if `NULL` is returned.

*Py_BuildValue()*는 항상 튜플을 구축하지는 않습니다. 포맷 문자열에 둘 이상의 포맷 단위가 포함되었을 때만 튜플을 구축합니다. 포맷 문자열이 비어 있으면, `None`을 반환합니다; 정확히 하나의 포맷 단위를 포함하면, 해당 포맷 단위가 기술하는 객체가 무엇이건 반환합니다. 크기가 0이나 1인 튜플을 반환하도록 하려면, 포맷 문자열을 괄호로 묶으십시오.

`s`와 `s#` 포맷의 경우처럼, 메모리 버퍼가 데이터를 빌드 객체에 제공하기 위해 매개 변수로 전달될 때, 필요한 데이터가 복사됩니다. 호출자가 제공하는 버퍼는 *Py_BuildValue()*가 만든 객체에 의해 참조되지 않습니다. 즉, 여러분의 코드가 `malloc()`을 호출하고 할당된 메모리를 *Py_BuildValue()*에 전달하면, 인단 *Py_BuildValue()*가 반환되면 여러분이 코드가 해당 메모리에 대해 `free()`를 호출해야 합니다.

다음 설명에서, 인용된 형식은 포맷 단위입니다; (둥근) 괄호 안의 항목은 포맷 단위가 반환할 파이썬 객체 형입니다; [대괄호] 안의 항목은 전달할 C 값의 형입니다.

문자 스페이스, 탭, 콜론 및 쉼표는 포맷 문자열에서 무시됩니다 (하지만 `s#`와 같은 포맷 단위 내에서는 아닙니다). 이것은 긴 포맷 문자열을 좀 더 읽기 쉽게 만드는 데 사용할 수 있습니다.

s (str 또는 None) [const char *]
: `'utf-8'` 인코딩을 사용하여 널-종료 C 문자열을 파이썬 `str` 객체로 변환합니다. C 문자열 포인터가 `NULL`이면, `None`이 사용됩니다.

s# (str or None) [const char *, *Py_ssize_t*]
: `'utf-8'` 인코딩을 사용하여 C 문자열과 그 길이를 파이썬 `str` 객체로 변환합니다. C 문자열 포인터가 `NULL`이면, 길이가 무시되고 `None`이 반환됩니다.

y (bytes) [const char *]
: 이것은 C 문자열을 파이썬 `bytes` 객체로 변환합니다. C 문자열 포인터가 `NULL`이면, `None`이 반환됩니다.

y# (bytes) [const char *, *Py_ssize_t*]
: 이것은 C 문자열과 그 길이를 파이썬 객체로 변환합니다. C 문자열 포인터가 `NULL`이면, `None`이 반환됩니다.

z (str 또는 None) [const char *]
: `s`와 같습니다.

z# (str or None) [const char *, *Py_ssize_t*]
: `s#`과 같습니다.

u (str) [const wchar_t *]
: 유니코드 (UTF-16 또는 UCS-4) 데이터의 널-종료 `wchar_t` 버퍼를 파이썬 유니코드 객체로 변환합니다. 유니코드 버퍼 포인터가 `NULL`이면, `None`이 반환됩니다.

u# (str) [const wchar_t *, *Py_ssize_t*]
유니코드 (UTF-16 또는 UCS-4) 데이터 버퍼와 그 길이를 파이썬 유니코드 객체로 변환합니다. 유니코드 버퍼 포인터가 `NULL`이면, 길이가 무시되고 `None`이 반환됩니다.

U (str 또는 None) [const char *]
`s`와 같습니다.

U# (str or None) [const char *, *Py_ssize_t*]
`s#`과 같습니다.

i (int) [int]
Convert a plain C `int` to a Python integer object.

b (int) [char]
Convert a plain C `char` to a Python integer object.

h (int) [short int]
Convert a plain C `short int` to a Python integer object.

l (int) [long int]
Convert a C `long int` to a Python integer object.

B (int) [unsigned char]
Convert a C `unsigned char` to a Python integer object.

H (int) [unsigned short int]
Convert a C `unsigned short int` to a Python integer object.

I (int) [unsigned int]
Convert a C `unsigned int` to a Python integer object.

k (int) [unsigned long]
Convert a C `unsigned long` to a Python integer object.

L (int) [long long]
Convert a C `long long` to a Python integer object.

K (int) [unsigned long long]
Convert a C `unsigned long long` to a Python integer object.

n (int) [*Py_ssize_t*]
C *Py_ssize_t*를 파이썬 정수로 변환합니다.

c (길이 1의 bytes) [char]
Convert a C `int` representing a byte to a Python `bytes` object of length 1.

C (길이 1의 str) [int]
Convert a C `int` representing a character to Python `str` object of length 1.

d (float) [double]
Convert a C `double` to a Python floating-point number.

f (float) [float]
Convert a C `float` to a Python floating-point number.

D (complex) [Py_complex *]
C *Py_complex* 구조체를 파이썬 복소수로 변환합니다.

O (object) [PyObject *]
Pass a Python object untouched but create a new *strong reference* to it (i.e. its reference count is incremented by one). If the object passed in is a `NULL` pointer, it is assumed that this was caused because the call producing the argument found an error and set an exception. Therefore, *Py_BuildValue()* will return `NULL` but won' t raise an exception. If no exception has been raised yet, `SystemError` is set.

S (object) [PyObject *]
`O`와 같습니다.

N (object) [PyObject *]
: Same as O, except it doesn' t create a new *strong reference*. Useful when the object is created by a call to an object constructor in the argument list.

O& (object) [*converter, anything*]
: Convert *anything* to a Python object through a *converter* function. The function is called with *anything* (which should be compatible with `void*`) as its argument and should return a "new" Python object, or `NULL` if an error occurred.

(items) (tuple) [*matching-items*]
: C값의 시퀀스를 항목 수가 같은 파이썬 튜플로 변환합니다.

[items] (list) [*matching-items*]
: C값의 시퀀스를 항목 수가 같은 파이썬 리스트로 변환합니다.

{items} (dict) [*matching-items*]
: C값의 시퀀스를 파이썬 딕셔너리로 변환합니다. 연속된 C 값의 각 쌍은 딕셔너리에 하나의 항목을 추가하여, 각각 키와 값으로 사용됩니다.

포맷 문자열에 에러가 있으면, `SystemError` 예외가 설정되고 `NULL`이 반환됩니다.

PyObject ***Py_VaBuildValue**(const char *format, va_list vargs)
: *Return value: New reference. Part of the* Stable ABI. 가변 개수의 인자가 아닌 va_list를 받아들인다는 점을 제외하면, *Py_BuildValue()*와 동일합니다.

6.7 문자열 변환과 포매팅

숫자 변환과 포맷된 문자열 출력을 위한 함수.

int **PyOS_snprintf**(char *str, size_t size, const char *format, ...)
: *Part of the* Stable ABI. 포맷 문자열 *format*과 추가 인자에 따라 *size* 바이트를 넘지 않도록 *str*로 출력합니다. 유닉스 매뉴얼 페이지 *snprintf(3)*를 보십시오.

int **PyOS_vsnprintf**(char *str, size_t size, const char *format, va_list va)
: *Part of the* Stable ABI. 포맷 문자열 *format*과 가변 인자 목록 *va*에 따라 *size* 바이트를 넘지 않도록 *str*로 출력합니다. 유닉스 매뉴얼 페이지 *vsnprintf(3)*를 보십시오.

*PyOS_snprintf()*와 *PyOS_vsnprintf()*는 표준 C 라이브러리 함수 `snprintf()`와 `vsnprintf()`를 감쌉니다. 그들의 목적은 경계 조건에서 표준 C 함수가 제공하지 않는 수준의 일관된 동작을 보장하는 것입니다.

The wrappers ensure that `str[size-1]` is always `'\0'` upon return. They never write more than *size* bytes (including the trailing `'\0'`) into str. Both functions require that `str != NULL`, `size > 0`, `format != NULL` and `size < INT_MAX`. Note that this means there is no equivalent to the C99 `n = snprintf(NULL, 0, ...)` which would determine the necessary buffer size.

이 함수들의 반환 값(*rv*)은 다음과 같이 해석되어야 합니다:

- `0 <= rv < size` 일 때, 출력 변환에 성공했으며 *rv* 문자가 *str*에 기록되었습니다 (`str[rv]`의 후행 `'\0'` 바이트 제외).
- `rv >= size` 일 때, 출력 변환이 잘렸고 성공하려면 `rv + 1` 바이트의 버퍼가 필요합니다. `str[size-1]`은 이때 `'\0'`입니다.
- `rv < 0` 일 때, "뭔가 나쁜 일이 일어났습니다." 이때도 `str[size-1]`은 `'\0'`이지만, *str*의 나머지는 정의되지 않습니다. 에러의 정확한 원인은 하부 플랫폼에 따라 다릅니다.

다음 함수는 로케일 독립적인 문자열에서 숫자로의 변환을 제공합니다.

unsigned long **PyOS_strtoul**(const char *str, char **ptr, int base)
: *Part of the* Stable ABI. Convert the initial part of the string in `str` to an `unsigned long` value according to the given `base`, which must be between `2` and `36` inclusive, or be the special value `0`.

  Leading white space and case of characters are ignored. If `base` is zero it looks for a leading `0b`, `0o` or `0x` to tell which base. If these are absent it defaults to `10`. Base must be 0 or between 2 and 36 (inclusive). If `ptr` is non-`NULL` it will contain a pointer to the end of the scan.

If the converted value falls out of range of corresponding return type, range error occurs (`errno` is set to `ERANGE`) and `ULONG_MAX` is returned. If no conversion can be performed, `0` is returned.

See also the Unix man page *strtoul(3)*.

Added in version 3.2.

long **PyOS_strtol**(const char *str, char **ptr, int base)

Part of the Stable ABI. Convert the initial part of the string in `str` to an `long` value according to the given `base`, which must be between `2` and `36` inclusive, or be the special value `0`.

Same as *PyOS_strtoul()*, but return a `long` value instead and `LONG_MAX` on overflows.

See also the Unix man page *strtol(3)*.

Added in version 3.2.

double **PyOS_string_to_double**(const char *s, char **endptr, *PyObject* *overflow_exception)

Part of the Stable ABI. Convert a string `s` to a `double`, raising a Python exception on failure. The set of accepted strings corresponds to the set of strings accepted by Python's `float()` constructor, except that `s` must not have leading or trailing whitespace. The conversion is independent of the current locale.

`endptr`이 `NULL`이면, 전체 문자열을 변환합니다. 문자열이 부동 소수점 숫자의 유효한 표현이 아니면 `ValueError`를 발생시키고 `-1.0`을 반환합니다.

endptr이 `NULL`이 아니면, 가능한 한 많은 문자열을 변환하고 `*endptr`이 변환되지 않은 첫 번째 문자를 가리키도록 설정합니다. 문자열의 초기 세그먼트가 부동 소수점 숫자의 유효한 표현이 아니면, `*endptr`이 문자열의 시작을 가리키도록 설정하고, ValueError를 발생시키고 `-1.0`을 반환합니다.

`s`가 float에 저장하기에 너무 큰 값을 나타낼 때 (예를 들어, 여러 플랫폼에서 `"1e500"`가 그런 문자열입니다), `overflow_exception`가 `NULL`이면 (적절한 부호와 함께) `Py_HUGE_VAL`을 반환하고, 어떤 예외도 설정하지 않습니다. 그렇지 않으면, `overflow_exception`은 파이썬 예외 객체를 가리켜야 합니다; 그 예외를 발생시키고 `-1.0`를 반환합니다. 두 경우 모두, 변환된 값 다음의 첫 번째 문자를 가리키도록 `*endptr`을 설정합니다.

변환 중 다른 에러가 발생하면 (예를 들어 메모리 부족 에러), 적절한 파이썬 예외를 설정하고 `-1.0`을 반환합니다.

Added in version 3.1.

char ***PyOS_double_to_string**(double val, char format_code, int precision, int flags, int *ptype)

Part of the Stable ABI. Convert a `double` *val* to a string using supplied *format_code*, *precision*, and *flags*.

*format_code*는 `'e'`, `'E'`, `'f'`, `'F'`, `'g'`, `'G'` 또는 `'r'` 중 하나여야 합니다. `'r'`의 경우, 제공된 *precision*은 0이어야 하며 무시됩니다. `'r'` 포맷 코드는 표준 `repr()` 형식을 지정합니다.

*flags*는 `Py_DTSF_SIGN`, `Py_DTSF_ADD_DOT_0` 또는 `Py_DTSF_ALT` 값을 0개 이상 함께 or 할 수 있습니다:

- `Py_DTSF_SIGN`은 *val*가 음수가 아닐 때도 항상 반환된 문자열 앞에 부호 문자가 오는 것을 뜻합니다.
- `Py_DTSF_ADD_DOT_0`은 반환된 문자열이 정수처럼 보이지 않도록 하는 것을 뜻합니다.
- `Py_DTSF_ALT`는 "대체" 포매팅 규칙을 적용하는 것을 뜻합니다. 자세한 내용은 *PyOS_snprintf()* `'#'` 지정자에 대한 설명서를 참조하십시오.

*ptype*이 `NULL`이 아니면, 포인터가 가리키는 값은 `Py_DTST_FINITE`, `Py_DTST_INFINITE` 또는 `Py_DTST_NAN` 중 하나로 설정되어, *val*가 각각 유한 수, 무한 수 또는 NaN임을 나타냅니다.

반환 값은 변환된 문자열이 있는 *buffer*에 대한 포인터이거나, 변환에 실패하면 `NULL`입니다. 호출자는 *PyMem_Free()*를 호출하여 반환된 문자열을 해제해야 합니다.

Added in version 3.1.

int **PyOS_stricmp**(const char *s1, const char *s2)

Case insensitive comparison of strings. The function works almost identically to `strcmp()` except that it ignores the case.

int **PyOS_strnicmp**(const char *s1, const char *s2, *Py_ssize_t* size)

Case insensitive comparison of strings. The function works almost identically to `strncmp()` except that it ignores the case.

6.8 PyHash API

See also the *PyTypeObject.tp_hash* member.

type **Py_hash_t**

Hash value type: signed integer.

Added in version 3.2.

type **Py_uhash_t**

Hash value type: unsigned integer.

Added in version 3.2.

type **PyHash_FuncDef**

Hash function definition used by *PyHash_GetFuncDef()*.

const char ***name**

Hash function name (UTF-8 encoded string).

const int **hash_bits**

Internal size of the hash value in bits.

const int **seed_bits**

Size of seed input in bits.

Added in version 3.4.

PyHash_FuncDef ***PyHash_GetFuncDef**(void)

Get the hash function definition.

> **더 보기**
>
> **PEP 456** "Secure and interchangeable hash algorithm".

Added in version 3.4.

6.9 리플렉션

PyObject ***PyEval_GetBuiltins**(void)

Return value: Borrowed reference. Part of the Stable ABI. 현재 실행 프레임이나 현재 실행 중인 프레임이 없으면 스레드 상태의 인터프리터의 builtins의 딕셔너리를 반환합니다.

PyObject ***PyEval_GetLocals**(void)

Return value: Borrowed reference. Part of the Stable ABI. 현재 실행 프레임의 지역 변수 딕셔너리를 반환하거나, 현재 실행 중인 프레임이 없으면 `NULL`을 반환합니다.

PyObject ***PyEval_GetGlobals**(void)

Return value: Borrowed reference. Part of the Stable ABI. 현재 실행 프레임의 전역 변수 딕셔너리를 반환하거나, 현재 실행 중인 프레임이 없으면 `NULL`을 반환합니다.

PyFrameObject ***PyEval_GetFrame**(void)

Return value: Borrowed reference. Part of the Stable ABI. 현재의 스레드 상태의 프레임을 반환합니다. 현재 실행 중의 프레임이 없으면 `NULL`입니다.

*PyThreadState_GetFrame()*도 참조하십시오.

const char *PyEval_GetFuncName(*PyObject* *func)

Part of the Stable ABI. *func*가 함수, 클래스 또는 인스턴스 객체면 *func*의 이름을 반환하고, 그렇지 않으면 *func*의 형의 이름을 반환합니다.

const char *PyEval_GetFuncDesc(*PyObject* *func)

Part of the Stable ABI. *func*의 형에 따라 설명 문자열을 반환합니다. 반환 값에는 함수 및 메서드의 "()", " constructor", " instance" 및 " object" 가 포함됩니다. `PyEval_GetFuncName()`의 결과와 이어붙이면 *func*의 설명이 됩니다.

6.10 코덱 등록소와 지원 함수

int PyCodec_Register(*PyObject* *search_function)

Part of the Stable ABI. 새로운 코덱 검색 함수를 등록합니다.

As side effect, this tries to load the `encodings` package, if not yet done, to make sure that it is always first in the list of search functions.

int PyCodec_Unregister(*PyObject* *search_function)

Part of the Stable ABI *since version 3.10.* Unregister a codec search function and clear the registry' s cache. If the search function is not registered, do nothing. Return 0 on success. Raise an exception and return -1 on error.

Added in version 3.10.

int PyCodec_KnownEncoding(const char *encoding)

Part of the Stable ABI. 지정된 *encoding*에 대해 등록된 코덱이 있는지에 따라 `1` 이나 `0`을 반환합니다. 이 함수는 항상 성공합니다.

PyObject *PyCodec_Encode(*PyObject* *object, const char *encoding, const char *errors)

Return value: New reference. Part of the Stable ABI. 일반 코덱 기반 인코딩 API.

*object*는 *errors*로 정의된 에러 처리 방법을 사용하여 지정된 *encoding*에 대해 발견된 인코더 함수로 전달됩니다. 코덱에 정의된 기본 방법을 사용하기 위해 *errors*가 `NULL` 일 수 있습니다. 인코더를 찾을 수 없으면 `LookupError`를 발생시킵니다.

PyObject *PyCodec_Decode(*PyObject* *object, const char *encoding, const char *errors)

Return value: New reference. Part of the Stable ABI. 일반 코덱 기반 디코딩 API.

*object*는 *errors*로 정의된 에러 처리 방법을 사용하여 지정된 *encoding*에 대해 발견된 디코더 함수로 전달됩니다. 코덱에 정의된 기본 방법을 사용하기 위해 *errors*가 `NULL` 일 수 있습니다. 인코더를 찾을 수 없으면 `LookupError`를 발생시킵니다.

6.10.1 코덱 조회 API

다음 함수에서, *encoding* 문자열은 모두 소문자로 변환되어 조회되므로, 이 메커니즘을 통한 인코딩 조회는 대소문자를 구분하지 않게 됩니다. 코덱이 없으면, `KeyError`가 설정되고 `NULL`이 반환됩니다.

PyObject *PyCodec_Encoder(const char *encoding)

Return value: New reference. Part of the Stable ABI. 주어진 *encoding*에 대한 인코더 함수를 가져옵니다.

PyObject *PyCodec_Decoder(const char *encoding)

Return value: New reference. Part of the Stable ABI. 주어진 *encoding*에 대한 디코더 함수를 가져옵니다.

PyObject *PyCodec_IncrementalEncoder(const char *encoding, const char *errors)

Return value: New reference. Part of the Stable ABI. 지정된 *encoding*에 대한 `IncrementalEncoder` 객체를 가져옵니다.

PyObject *PyCodec_IncrementalDecoder(const char *encoding, const char *errors)

Return value: New reference. Part of the Stable ABI. 지정된 *encoding*에 대한 `IncrementalDecoder` 객체를 가져옵니다.

PyObject ***PyCodec_StreamReader**(const char *encoding, *PyObject* *stream, const char *errors)

Return value: New reference. Part of the Stable ABI. 지정된 *encoding*에 대한 `StreamReader` 팩토리 함수를 가져옵니다.

PyObject ***PyCodec_StreamWriter**(const char *encoding, *PyObject* *stream, const char *errors)

Return value: New reference. Part of the Stable ABI. 지정된 *encoding*에 대한 `StreamWriter` 팩토리 함수를 가져옵니다.

6.10.2 유니코드 인코딩 에러 처리기용 등록소 API

int **PyCodec_RegisterError**(const char *name, *PyObject* *error)

Part of the Stable ABI. 지정된 *name* 으로 에러 처리 콜백 함수 *error*를 등록합니다. 코덱이 인코딩할 수 없는 문자/디코딩할 수 없는 바이트열을 발견하고, 인코드/디코드 함수를 호출할 때 *name*이 error 매개 변수로 지정되었을 때 이 콜백 함수를 호출합니다.

콜백은 하나의 인자로 `UnicodeEncodeError`, `UnicodeDecodeError` 또는 `UnicodeTranslateError`의 인스턴스를 받아들이는데, 문제가 되는 문자나 바이트의 시퀀스와 이들의 원본 문자열에서의 오프셋에 대한 정보를 담고 있습니다 (이 정보를 추출하는 함수는 유니코드 예외 객체를 참조하세요). 콜백은 주어진 예외를 발생시키거나, 문제가 있는 시퀀스의 대체와 원래 문자열에서 인코딩/디코딩을 다시 시작해야 하는 오프셋을 제공하는 정수를 포함하는 두 항목 튜플을 반환해야 합니다.

성공하면 `0`을, 에러면 `-1`을 반환합니다.

PyObject ***PyCodec_LookupError**(const char *name)

Return value: New reference. Part of the Stable ABI. *name* 으로 등록된 에러 처리 콜백 함수를 찾습니다. 특수한 경우로 `NULL`이 전달 될 수 있는데, 이때는 "strict" 에 대한 에러 처리 콜백이 반환됩니다.

PyObject ***PyCodec_StrictErrors**(*PyObject* *exc)

Return value: Always NULL. Part of the Stable ABI. *exc*를 예외로 발생시킵니다.

PyObject ***PyCodec_IgnoreErrors**(*PyObject* *exc)

Return value: New reference. Part of the Stable ABI. 잘못된 입력을 건너뛰고, 유니코드 에러를 무시합니다.

PyObject ***PyCodec_ReplaceErrors**(*PyObject* *exc)

Return value: New reference. Part of the Stable ABI. 유니코드 인코딩 에러를 `?` 나 `U+FFFD`로 치환합니다.

PyObject ***PyCodec_XMLCharRefReplaceErrors**(*PyObject* *exc)

Return value: New reference. Part of the Stable ABI. 유니코드 인코딩 에러를 XML 문자 참조로 치환합니다.

PyObject ***PyCodec_BackslashReplaceErrors**(*PyObject* *exc)

Return value: New reference. Part of the Stable ABI. 유니코드 인코딩 에러를 백 슬래시 이스케이프(`\x`, `\u` 및 `\U`)로 치환합니다.

PyObject ***PyCodec_NameReplaceErrors**(*PyObject* *exc)

Return value: New reference. Part of the Stable ABI *since version 3.7.* 유니코드 인코딩 에러를 `\N{...}` 이스케이프로 치환합니다.

Added in version 3.5.

6.11 Support for Perf Maps

On supported platforms (as of this writing, only Linux), the runtime can take advantage of *perf map files* to make Python functions visible to an external profiling tool (such as perf). A running process may create a file in the `/tmp` directory, which contains entries that can map a section of executable code to a name. This interface is described in the documentation of the Linux Perf tool.

In Python, these helper APIs can be used by libraries and features that rely on generating machine code on the fly.

Note that holding the Global Interpreter Lock (GIL) is not required for these APIs.

int **PyUnstable_PerfMapState_Init**(void)

> This is *Unstable API*. It may change without warning in minor releases.

Open the `/tmp/perf-$pid.map` file, unless it' s already opened, and create a lock to ensure thread-safe writes to the file (provided the writes are done through *PyUnstable_WritePerfMapEntry()*). Normally, there' s no need to call this explicitly; just use *PyUnstable_WritePerfMapEntry()* and it will initialize the state on first call.

Returns `0` on success, `-1` on failure to create/open the perf map file, or `-2` on failure to create a lock. Check `errno` for more information about the cause of a failure.

int **PyUnstable_WritePerfMapEntry**(const void *code_addr, unsigned int code_size, const char *entry_name)

> This is *Unstable API*. It may change without warning in minor releases.

Write one single entry to the `/tmp/perf-$pid.map` file. This function is thread safe. Here is what an example entry looks like:

```
# address       size  name
7f3529fcf759 b     py::bar:/run/t.py
```

Will call *PyUnstable_PerfMapState_Init()* before writing the entry, if the perf map file is not already opened. Returns `0` on success, or the same error codes as *PyUnstable_PerfMapState_Init()* on failure.

void **PyUnstable_PerfMapState_Fini**(void)

> This is *Unstable API*. It may change without warning in minor releases.

Close the perf map file opened by *PyUnstable_PerfMapState_Init()*. This is called by the runtime itself during interpreter shut-down. In general, there shouldn' t be a reason to explicitly call this, except to handle specific scenarios such as forking.

CHAPTER 7

추상 객체 계층

이 장의 함수는 객체의 형과 무관하게, 혹은 광범위한 종류의 객체 형의 (예를 들어, 모든 숫자 형 또는 모든 시퀀스 형) 파이썬 객체와 상호 작용합니다. 적용되지 않는 객체 형에 사용되면, 파이썬 예외가 발생합니다.

*PyList_New()*로 만들었지만, 항목이 아직 NULL이 아닌 값으로 설정되지 않은 리스트 객체와 같이, 제대로 초기화되지 않은 객체에 대해서는 이 함수를 사용할 수 없습니다.

7.1 객체 프로토콜

PyObject ***Py_NotImplemented**

지정된 형 조합에 대해 연산이 구현되지 않았음을 알리는 데 사용되는 NotImplemented 싱글톤.

Py_RETURN_NOTIMPLEMENTED

Properly handle returning *Py_NotImplemented* from within a C function (that is, create a new *strong reference* to NotImplemented and return it).

Py_PRINT_RAW

Flag to be used with multiple functions that print the object (like *PyObject_Print()* and *PyFile_WriteObject()*). If passed, these function would use the str() of the object instead of the repr().

int **PyObject_Print**(*PyObject* *o, FILE *fp, int flags)

Print an object *o*, on file *fp*. Returns -1 on error. The flags argument is used to enable certain printing options. The only option currently supported is *Py_PRINT_RAW*; if given, the str() of the object is written instead of the repr().

int **PyObject_HasAttr**(*PyObject* *o, *PyObject* *attr_name)

Part of the Stable ABI. *o*에 *attr_name* 어트리뷰트가 있으면 1을, 그렇지 않으면 0을 반환합니다. 이것은 파이썬 표현식 hasattr(o, attr_name)과 동등합니다. 이 함수는 항상 성공합니다.

> **참고**
>
> Exceptions that occur when this calls __getattr__() and __getattribute__() methods are silently ignored. For proper error handling, use *PyObject_GetAttr()* instead.

int **PyObject_HasAttrString**(*PyObject* *o, const char *attr_name)

Part of the Stable ABI. This is the same as *PyObject_HasAttr()*, but *attr_name* is specified as a `const char*` UTF-8 encoded bytes string, rather than a *PyObject**.

> **참고**
>
> Exceptions that occur when this calls `__getattr__()` and `__getattribute__()` methods or while creating the temporary `str` object are silently ignored. For proper error handling, use *PyObject_GetAttrString()* instead.

PyObject ***PyObject_GetAttr**(*PyObject* *o, *PyObject* *attr_name)

Return value: New reference. *Part of the* Stable ABI. 객체 *o*에서 *attr_name*이라는 이름의 어트리뷰트를 가져옵니다. 성공하면 어트리뷰트 값을, 실패하면 `NULL`을 반환합니다. 이것은 파이썬 표현식 `o.attr_name`과 동등합니다.

PyObject ***PyObject_GetAttrString**(*PyObject* *o, const char *attr_name)

Return value: New reference. *Part of the* Stable ABI. This is the same as *PyObject_GetAttr()*, but *attr_name* is specified as a `const char*` UTF-8 encoded bytes string, rather than a *PyObject**.

PyObject ***PyObject_GenericGetAttr**(*PyObject* *o, *PyObject* *name)

Return value: New reference. *Part of the* Stable ABI. 형 객체의 `tp_getattro` 슬롯에 배치되는 일반 어트리뷰트 게터(getter) 함수. 객체의 (있다면) `__dict__`에 있는 어트리뷰트뿐만 아니라 객체의 MRO에 있는 클래스의 딕셔너리에 있는 디스크립터를 찾습니다. descriptors에 요약된 것처럼, 데이터 디스크립터는 인스턴스 어트리뷰트보다 우선하지만, 비 데이터 디스크립터는 그렇지 않습니다. 그렇지 않으면, `AttributeError`가 발생합니다.

int **PyObject_SetAttr**(*PyObject* *o, *PyObject* *attr_name, *PyObject* *v)

Part of the Stable ABI. 객체 *o*에 대해, *attr_name*이라는 이름의 어트리뷰트 값을 *v* 값으로 설정합니다. 실패 시 예외를 발생시키고 `-1`을 반환합니다. 성공하면 `0`을 반환합니다. 이것은 파이썬 문장 `o.attr_name = v`와 동등합니다.

If *v* is `NULL`, the attribute is deleted. This behaviour is deprecated in favour of using *PyObject_DelAttr()*, but there are currently no plans to remove it.

int **PyObject_SetAttrString**(*PyObject* *o, const char *attr_name, *PyObject* *v)

Part of the Stable ABI. This is the same as *PyObject_SetAttr()*, but *attr_name* is specified as a `const char*` UTF-8 encoded bytes string, rather than a *PyObject**.

If *v* is `NULL`, the attribute is deleted, but this feature is deprecated in favour of using *PyObject_DelAttrString()*.

The number of different attribute names passed to this function should be kept small, usually by using a statically allocated string as *attr_name*. For attribute names that aren' t known at compile time, prefer calling *PyUnicode_FromString()* and *PyObject_SetAttr()* directly. For more details, see *PyUnicode_InternFromString()*, which may be used internally to create a key object.

int **PyObject_GenericSetAttr**(*PyObject* *o, *PyObject* *name, *PyObject* *value)

Part of the Stable ABI. 형 객체의 *tp_setattro* 슬롯에 배치되는 일반 어트리뷰트 세터(setter)와 딜리터(deleter) 함수. 객체의 MRO에 있는 클래스의 딕셔너리에서 데이터 디스크립터를 찾고, 발견되면 인스턴스 딕셔너리에 있는 어트리뷰트를 설정하거나 삭제하는 것보다 우선합니다. 그렇지 않으면, 객체의 (있다면) `__dict__`에서 어트리뷰트가 설정되거나 삭제됩니다. 성공하면 `0`이 반환되고, 그렇지 않으면 `AttributeError`가 발생하고 `-1`이 반환됩니다.

int **PyObject_DelAttr**(*PyObject* *o, *PyObject* *attr_name)

객체 *o*에 대해, *attr_name*이라는 이름의 어트리뷰트를 삭제합니다. 실패 시 `-1`을 반환합니다. 이것은 파이썬 문장 `del o.attr_name`과 동등합니다.

int **PyObject_DelAttrString**(*PyObject* *o, const char *attr_name)

This is the same as *PyObject_DelAttr()*, but *attr_name* is specified as a `const char*` UTF-8 encoded bytes string, rather than a *PyObject**.

The number of different attribute names passed to this function should be kept small, usually by using a statically allocated string as *attr_name*. For attribute names that aren't known at compile time, prefer calling `PyUnicode_FromString()` and `PyObject_DelAttr()` directly. For more details, see `PyUnicode_InternFromString()`, which may be used internally to create a key object for lookup.

PyObject ***PyObject_GenericGetDict**(*PyObject* *o, void *context)

Return value: New reference. Part of the Stable ABI *since version 3.10.* `__dict__` 디스크립터의 게터(getter)를 위한 일반적인 구현. 필요하면 딕셔너리를 만듭니다.

This function may also be called to get the `__dict__` of the object *o*. Pass `NULL` for *context* when calling it. Since this function may need to allocate memory for the dictionary, it may be more efficient to call `PyObject_GetAttr()` when accessing an attribute on the object.

On failure, returns `NULL` with an exception set.

Added in version 3.3.

int **PyObject_GenericSetDict**(*PyObject* *o, *PyObject* *value, void *context)

Part of the Stable ABI *since version 3.7.* `__dict__` 디스크립터의 세터(setter)를 위한 일반적인 구현. 이 구현은 딕셔너리 삭제를 허락하지 않습니다.

Added in version 3.3.

PyObject ****_PyObject_GetDictPtr**(*PyObject* *obj)

Return a pointer to `__dict__` of the object *obj*. If there is no `__dict__`, return `NULL` without setting an exception.

This function may need to allocate memory for the dictionary, so it may be more efficient to call `PyObject_GetAttr()` when accessing an attribute on the object.

PyObject ***PyObject_RichCompare**(*PyObject* *o1, *PyObject* *o2, int opid)

Return value: New reference. Part of the Stable ABI. Compare the values of *o1* and *o2* using the operation specified by *opid*, which must be one of `Py_LT`, `Py_LE`, `Py_EQ`, `Py_NE`, `Py_GT`, or `Py_GE`, corresponding to `<`, `<=`, `==`, `!=`, `>`, or `>=` respectively. This is the equivalent of the Python expression `o1 op o2`, where `op` is the operator corresponding to *opid*. Returns the value of the comparison on success, or `NULL` on failure.

int **PyObject_RichCompareBool**(*PyObject* *o1, *PyObject* *o2, int opid)

Part of the Stable ABI. Compare the values of *o1* and *o2* using the operation specified by *opid*, like `PyObject_RichCompare()`, but returns `-1` on error, `0` if the result is false, `1` otherwise.

참고

If *o1* and *o2* are the same object, `PyObject_RichCompareBool()` will always return `1` for `Py_EQ` and `0` for `Py_NE`.

PyObject ***PyObject_Format**(*PyObject* *obj, *PyObject* *format_spec)

Part of the Stable ABI. Format *obj* using *format_spec*. This is equivalent to the Python expression `format(obj, format_spec)`.

format_spec may be `NULL`. In this case the call is equivalent to `format(obj)`. Returns the formatted string on success, `NULL` on failure.

PyObject ***PyObject_Repr**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 객체 *o*의 문자열 표현을 계산합니다. 성공하면 문자열 표현을, 실패하면 `NULL`을 반환합니다. 이것은 파이썬 표현식 `repr(o)`와 동등합니다. `repr()` 내장 함수에 의해 호출됩니다.

버전 3.4에서 변경: 이 함수에는 이제 디버그 어서션이 포함되어 있어 활성 예외를 조용히 버리지 않도록 합니다.

PyObject ***PyObject_ASCII**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. `PyObject_Repr()`처럼, 객체 *o*의 문자열 표현을 계산하지만, `\x`, `\u` 또는 `\U` 이스케이프를 사용하여 `PyObject_Repr()`이 반환한 문자열에서 비 ASCII 문자를 이스케이프 합니다. 이것은 파이썬 2에서 `PyObject_Repr()`에 의해 반환된 것과 유사한 문자열을 생성합니다. `ascii()` 내장 함수에 의해 호출됩니다.

PyObject ***PyObject_Str**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 객체 *o*의 문자열 표현을 계산합니다. 성공 시 문자열 표현을, 실패 시 `NULL`을 반환합니다. 이것은 파이썬 표현식 `str(o)`와 동등합니다. `str()` 내장 함수에 의해, 따라서 `print()` 함수에 의해서도 호출됩니다.

버전 3.4에서 변경: 이 함수에는 이제 디버그 어서션이 포함되어 있어 활성 예외를 조용히 버리지 않도록 합니다.

PyObject ***PyObject_Bytes**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 객체 *o*의 바이트열 표현을 계산합니다. 실패하면 `NULL`을, 성공하면 바이트열 객체를 반환됩니다. 이는 *o*가 정수가 아닐 때 파이썬 표현식 `bytes(o)`와 동등합니다. `bytes(o)`와 달리, *o*가 정수이면 0으로 초기화된 바이트열 객체 대신 TypeError가 발생합니다.

int **PyObject_IsSubclass**(*PyObject* *derived, *PyObject* *cls)

Part of the Stable ABI. 클래스 *derived*가 클래스 *cls*와 동일하거나 *cls*에서 파생되었으면 `1`을 반환하고, 그렇지 않으면 `0`을 반환합니다. 에러가 발생하면 `-1`을 반환합니다.

*cls*가 튜플이면, *cls*의 모든 항목에 대해 검사가 수행됩니다. 적어도 하나의 검사에서 `1`을 반환하면 결과는 `1`이 되고, 그렇지 않으면 `0`이 됩니다.

If *cls* has a `__subclasscheck__()` method, it will be called to determine the subclass status as described in **PEP 3119**. Otherwise, *derived* is a subclass of *cls* if it is a direct or indirect subclass, i.e. contained in `cls.__mro__`.

Normally only class objects, i.e. instances of `type` or a derived class, are considered classes. However, objects can override this by having a `__bases__` attribute (which must be a tuple of base classes).

int **PyObject_IsInstance**(*PyObject* *inst, *PyObject* *cls)

Part of the Stable ABI. *inst*가 *cls* 클래스나 *cls*의 서브 클래스의 인스턴스이면 `1`을 반환하고, 그렇지 않으면 `0`을 반환합니다. 에러가 발생하면 `-1`을 반환하고 예외를 설정합니다.

*cls*가 튜플이면, *cls*의 모든 항목에 대해 검사가 수행됩니다. 적어도 하나의 검사에서 `1`을 반환하면 결과는 `1`이 되고, 그렇지 않으면 `0`이 됩니다.

If *cls* has a `__instancecheck__()` method, it will be called to determine the subclass status as described in **PEP 3119**. Otherwise, *inst* is an instance of *cls* if its class is a subclass of *cls*.

An instance *inst* can override what is considered its class by having a `__class__` attribute.

An object *cls* can override if it is considered a class, and what its base classes are, by having a `__bases__` attribute (which must be a tuple of base classes).

Py_hash_t **PyObject_Hash**(*PyObject* *o)

Part of the Stable ABI. 객체 *o*의 해시값을 계산하고 반환합니다. 실패하면 `-1`을 반환합니다. 이것은 파이썬 표현식 `hash(o)`와 동등합니다.

버전 3.2에서 변경: The return type is now Py_hash_t. This is a signed integer the same size as `Py_ssize_t`.

Py_hash_t **PyObject_HashNotImplemented**(*PyObject* *o)

Part of the Stable ABI. Set a `TypeError` indicating that `type(o)` is not *hashable* and return `-1`. This function receives special treatment when stored in a `tp_hash` slot, allowing a type to explicitly indicate to the interpreter that it is not hashable.

int **PyObject_IsTrue**(*PyObject* *o)

Part of the Stable ABI. 객체 *o*를 참으로 간주하면 `1`을, 그렇지 않으면 `0`을 반환합니다. 이것은 파이썬 표현식 `not not o`와 동등합니다. 실패하면 `-1`을 반환합니다.

int **PyObject_Not**(*PyObject* *o)

Part of the Stable ABI. 객체 *o*를 참으로 간주하면 `0`을, 그렇지 않으면 `1`을 반환합니다. 이것은 파이썬 표현식 `not o`와 동등합니다. 실패하면 `-1`을 반환합니다.

PyObject ***PyObject_Type**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. When *o* is non-`NULL`, returns a type object corresponding to the object type of object *o*. On failure, raises `SystemError` and returns `NULL`. This is equivalent to the Python expression `type(o)`. This function creates a new *strong reference* to the return value. There's really no reason to use this function instead of the *Py_TYPE()* function, which returns a pointer of type *PyTypeObject**, except when a new *strong reference* is needed.

int **PyObject_TypeCheck**(*PyObject* *o, *PyTypeObject* *type)

Return non-zero if the object *o* is of type *type* or a subtype of *type*, and `0` otherwise. Both parameters must be non-`NULL`.

Py_ssize_t **PyObject_Size**(*PyObject* *o)

Py_ssize_t **PyObject_Length**(*PyObject* *o)

Part of the Stable ABI. 객체 *o*의 길이를 반환합니다. 객체 *o*가 시퀀스와 매핑 프로토콜을 제공하면, 시퀀스 길이가 반환됩니다. 에러가 발생하면 `-1`이 반환됩니다. 이것은 파이썬 표현식 `len(o)`와 동등합니다.

Py_ssize_t **PyObject_LengthHint**(*PyObject* *o, *Py_ssize_t* defaultvalue)

Return an estimated length for the object *o*. First try to return its actual length, then an estimate using `__length_hint__()`, and finally return the default value. On error return `-1`. This is the equivalent to the Python expression `operator.length_hint(o, defaultvalue)`.

Added in version 3.4.

PyObject ***PyObject_GetItem**(*PyObject* *o, *PyObject* *key)

Return value: New reference. Part of the Stable ABI. 객체 *key*에 해당하는 *o*의 요소를 반환하거나 실패 시 `NULL`을 반환합니다. 이것은 파이썬 표현식 `o[key]`와 동등합니다.

int **PyObject_SetItem**(*PyObject* *o, *PyObject* *key, *PyObject* *v)

Part of the Stable ABI. 객체 *key*를 값 *v*에 매핑합니다. 실패 시 예외를 발생시키고 `-1`을 반환합니다; 성공하면 `0`을 반환합니다. 이것은 파이썬 문장 `o[key] = v`와 동등합니다. 이 함수는 *v*에 대한 참조를 훔치지 않습니다.

int **PyObject_DelItem**(*PyObject* *o, *PyObject* *key)

Part of the Stable ABI. 객체 *o*에서 객체 *key*에 대한 매핑을 제거합니다. 실패하면 `-1`을 반환합니다. 이것은 파이썬 문장 `del o[key]`와 동등합니다.

int **PyObject_DelItemString**(*PyObject* *o, const char *key)

Part of the Stable ABI. This is the same as *PyObject_DelItem()*, but *key* is specified as a `const char*` UTF-8 encoded bytes string, rather than a *PyObject**.

PyObject ***PyObject_Dir**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 이것은 파이썬 표현식 `dir(o)`와 동등하며, 객체 인자에 적합한 문자열의 (비어있을 수 있는) 리스트를 반환하거나, 에러가 있으면 `NULL`을 반환합니다. 인자가 `NULL`이면, 파이썬 `dir()`과 비슷하며, 현재 지역(locals)의 이름들을 반환합니다; 이 경우, 실행 프레임이 활성화되어 있지 않으면 `NULL`이 반환되지만 *PyErr_Occurred()*는 거짓을 반환합니다.

PyObject ***PyObject_GetIter**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 이것은 파이썬 표현식 `iter(o)`와 동등합니다. 객체 인자에 대한 새로운 이터레이터를 반환하거나, 객체가 이미 이터레이터이면 객체 자체를 반환합니다. 객체를 이터레이트 할 수 없으면 `TypeError`를 발생시키고 `NULL`을 반환합니다.

PyObject ***PyObject_SelfIter**(*PyObject* *obj)

Return value: New reference. Part of the Stable ABI. This is equivalent to the Python `__iter__(self): return self` method. It is intended for *iterator* types, to be used in the *PyTypeObject.tp_iter* slot.

PyObject ***PyObject_GetAIter**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI *since version 3.10.* This is the equivalent to the Python expression `aiter(o)`. Takes an `AsyncIterable` object and returns an `AsyncIterator` for it. This is typically a new iterator but if the argument is an `AsyncIterator`, this returns itself. Raises `TypeError` and returns `NULL` if the object cannot be iterated.

Added in version 3.10.

void ***PyObject_GetTypeData**(*PyObject* *o, *PyTypeObject* *cls)

Part of the Stable ABI *since version 3.12.* Get a pointer to subclass-specific data reserved for *cls*.

The object *o* must be an instance of *cls*, and *cls* must have been created using negative *PyType_Spec.basicsize*. Python does not check this.

On error, set an exception and return `NULL`.

Added in version 3.12.

Py_ssize_t **PyType_GetTypeDataSize**(*PyTypeObject* *cls)

Part of the Stable ABI *since version 3.12.* Return the size of the instance memory space reserved for *cls*, i.e. the size of the memory *PyObject_GetTypeData()* returns.

This may be larger than requested using *-PyType_Spec.basicsize*; it is safe to use this larger size (e.g. with `memset()`).

The type *cls* **must** have been created using negative *PyType_Spec.basicsize*. Python does not check this.

On error, set an exception and return a negative value.

Added in version 3.12.

void ***PyObject_GetItemData**(*PyObject* *o)

Get a pointer to per-item data for a class with *Py_TPFLAGS_ITEMS_AT_END*.

On error, set an exception and return `NULL`. `TypeError` is raised if *o* does not have *Py_TPFLAGS_ITEMS_AT_END* set.

Added in version 3.12.

7.2 호출 프로토콜

CPython은 두 가지 호출 프로토콜을 지원합니다: *tp_call*과 벡터콜(vectorcall).

7.2.1 *tp_call* 프로토콜

*tp_call*을 설정하는 클래스의 인스턴스는 콜러블입니다. 슬롯의 서명은 다음과 같습니다:

```
PyObject *tp_call(PyObject *callable, PyObject *args, PyObject *kwargs);
```

파이썬 코드의 `callable(*args, **kwargs)` 와 유사하게, 위치 인자를 위한 튜플과 키워드 인자를 위한 딕셔너리를 사용하여 호출합니다. *args*는 NULL이 아니어야 합니다 (인자가 없으면 빈 튜플을 사용하십시오). 하지만 키워드 인자가 없으면 *kwargs*는 *NULL*일 수 있습니다.

이 규칙은 *tp_call*에서만 사용되는 것이 아닙니다: *tp_new*와 *tp_init*도 인자를 이런 식으로 전달합니다.

To call an object, use *PyObject_Call()* or another *call API*.

7.2.2 벡터콜(Vectorcall) 프로토콜

Added in version 3.9.

벡터콜 프로토콜은 **PEP 590**에서 호출 효율을 높이기 위한 추가 프로토콜로 도입되었습니다.

경험 규칙으로, CPython은 콜러블이 지원하면 내부 호출에 대해 벡터콜을 선호합니다. 그러나 이것은 엄격한 규칙이 아닙니다. 또한, 일부 제삼자 확장은 (`PyObject_Call()`을 사용하지 않고) *tp_call*을 직접

사용합니다. 따라서, 벡터콜을 지원하는 클래스도 *tp_call*을 구현해야 합니다. 또한, 어떤 프로토콜을 사용하는지와 관계없이 콜러블은 동일하게 작동해야 합니다. 이를 위해 권장되는 방법은 *tp_call*을 *PyVectorcall_Call()*로 설정하는 것입니다. 이것이 반복을 처리합니다:

경고

벡터콜을 지원하는 클래스도 같은 의미가 있도록 *tp_call*을 **반드시** 구현해야 합니다.

버전 3.12에서 변경: The *Py_TPFLAGS_HAVE_VECTORCALL* flag is now removed from a class when the class' s `__call__()` method is reassigned. (This internally sets *tp_call* only, and thus may make it behave differently than the vectorcall function.) In earlier Python versions, vectorcall should only be used with *immutable* or static types.

*tp_call*보다 느려진다면 클래스는 벡터콜을 구현해서는 안 됩니다. 예를 들어, 피호출자가 어차피 인자를 인자 튜플과 kwargs 딕셔너리로 변환해야 하면, 벡터콜을 구현할 이유가 없습니다.

Classes can implement the vectorcall protocol by enabling the *Py_TPFLAGS_HAVE_VECTORCALL* flag and setting *tp_vectorcall_offset* to the offset inside the object structure where a *vectorcallfunc* appears. This is a pointer to a function with the following signature:

typedef *PyObject* *(***vectorcallfunc**)(*PyObject* *callable, *PyObject* *const *args, size_t nargsf, *PyObject* *kwnames)

Part of the Stable ABI *since version 3.12.*

- *callable*은 호출되는 객체입니다.
- ***args*는 위치 인자와 그 뒤를 따르는 키워드 인자의 값으로 구성된 C 배열입니다.**
  인자가 없으면 *NULL*일 수 있습니다.
- ***nargsf*는 위치 인자의 수에**
  PY_VECTORCALL_ARGUMENTS_OFFSET flag. To get the actual number of positional arguments from *nargsf*, use *PyVectorcall_NARGS()*.
- ***kwnames*는 키워드 인자의 이름을 포함하는 튜플입니다;**
  다시 말해, kwargs 딕셔너리의 키. 이 이름들은 문자열(`str`이나 서브 클래스의 인스턴스)이어야하며 고유해야 합니다. 키워드 인자가 없으면, *kwnames*는 대신 *NULL*일 수 있습니다.

PY_VECTORCALL_ARGUMENTS_OFFSET

Part of the Stable ABI *since version 3.12.* 이 플래그가 벡터콜 *nargsf* 인자에 설정되면, 피호출자는 일시적으로 `args[-1]`을 변경할 수 있습니다. 즉, *args*는 할당된 벡터에서 인자 1(0이 아닙니다)을 가리킵니다. 피호출자는 반환하기 전에 `args[-1]` 값을 복원해야 합니다.

PyObject_VectorcallMethod() 의 경우, 이 플래그는 대신 `args[0]`이 변경될 수 있음을 의미합니다.

Whenever they can do so cheaply (without additional allocation), callers are encouraged to use *PY_VECTORCALL_ARGUMENTS_OFFSET*. Doing so will allow callables such as bound methods to make their onward calls (which include a prepended *self* argument) very efficiently.

Added in version 3.8.

벡터콜을 구현하는 객체를 호출하려면, 다른 콜러블과 마찬가지로 호출 *API* 함수를 사용하십시오. *PyObject_Vectorcall()*은 일반적으로 가장 효율적입니다.

참고

CPython 3.8에서 벡터콜 API와 관련 함수는 앞에 밑줄이 붙은 이름으로 잠정적으로 사용할 수 있었습니다: `_PyObject_Vectorcall`, `_Py_TPFLAGS_HAVE_VECTORCALL`, `_PyObject_VectorcallMethod`, `_PyVectorcall_Function`, `_PyObject_CallOneArg`, `_PyObject_CallMethodNoArgs`, `_PyObject_CallMethodOneArg`. 또한, `PyObject_VectorcallDict`는 `_PyObject_FastCallDict`로 제공되었습니다. 이전 이름은 여전히 밑줄이 없는 새로운 이름의 별칭으로 정의됩니다.

재귀 제어

*tp_call*을 사용할 때, 피호출자는 재귀에 대해 걱정할 필요가 없습니다: CPython은 *tp_call*을 사용하여 호출한 경우 `Py_EnterRecursiveCall()` 과 `Py_LeaveRecursiveCall()` 을 사용합니다.

효율성을 위해, 벡터콜을 사용하여 호출한 경우에는 그렇지 않습니다: 피호출자는 필요하면 *Py_EnterRecursiveCall* 과 *Py_LeaveRecursiveCall* 을 사용해야 합니다.

벡터콜 지원 API

Py_ssize_t **PyVectorcall_NARGS**(size_t nargsf)

Part of the Stable ABI *since version 3.12.* 벡터콜 *nargsf* 인자가 주어지면, 실제 인자 수를 반환합니다. 현재 다음과 동등합니다:

```
(Py_ssize_t)(nargsf & ~PY_VECTORCALL_ARGUMENTS_OFFSET)
```

그러나, 향후 확장을 위해 `PyVectorcall_NARGS` 함수를 사용해야 합니다.

Added in version 3.8.

vectorcallfunc **PyVectorcall_Function**(*PyObject* *op)

*op*가 벡터콜 프로토콜을 지원하지 않으면 (형이 지원하지 않거나 인스턴스가 지원하지 않기 때문에), *NULL*을 반환합니다. 그렇지 않으면, *op*에 저장된 벡터콜 함수 포인터를 반환합니다. 이 함수는 예외를 발생시키지 않습니다.

이것은 *op*가 벡터콜을 지원하는지를 확인하는 데 주로 유용하며, `PyVectorcall_Function(op) != NULL`을 확인하여 수행 할 수 있습니다.

Added in version 3.9.

PyObject ***PyVectorcall_Call**(*PyObject* *callable, *PyObject* *tuple, *PyObject* *dict)

Part of the Stable ABI *since version 3.12.* 튜플과 딕셔너리에 각각 주어진 위치와 키워드 인자로 *callable*의 *vectorcallfunc* 를 호출합니다.

This is a specialized function, intended to be put in the *tp_call* slot or be used in an implementation of `tp_call`. It does not check the *Py_TPFLAGS_HAVE_VECTORCALL* flag and it does not fall back to `tp_call`.

Added in version 3.8.

7.2.3 객체 호출 API

Various functions are available for calling a Python object. Each converts its arguments to a convention supported by the called object – either *tp_call* or vectorcall. In order to do as little conversion as possible, pick one that best fits the format of data you have available.

다음 표는 사용 가능한 함수를 요약한 것입니다; 자세한 내용은 개별 설명서를 참조하십시오.

함수	콜러블	args	kwargs
PyObject_Call()	PyObject *	튜플	딕셔너리/NULL
PyObject_CallNoArgs()	PyObject *	—	—
PyObject_CallOneArg()	PyObject *	1 객체	—
PyObject_CallObject()	PyObject *	튜플/NULL	—
PyObject_CallFunction()	PyObject *	포맷 (format)	—
PyObject_CallMethod()	obj + char*	포맷 (format)	—
PyObject_CallFunctionObjArgs()	PyObject *	가변 (variadic)	—
PyObject_CallMethodObjArgs()	obj + name	가변 (variadic)	—
PyObject_CallMethodNoArgs()	obj + name	—	—
PyObject_CallMethodOneArg()	obj + name	1 객체	—
PyObject_Vectorcall()	PyObject *	벡터콜	벡터콜
PyObject_VectorcallDict()	PyObject *	벡터콜	딕셔너리/NULL
PyObject_VectorcallMethod()	arg + name	벡터콜	벡터콜

PyObject ***PyObject_Call**(*PyObject* *callable, *PyObject* *args, *PyObject* *kwargs)

Return value: New reference. Part of the Stable ABI. 튜플 *args*로 주어진 인자와 딕셔너리 *kwargs*로 주어진 이름있는 인자로 콜러블 파이썬 객체 *callable*을 호출합니다.

*args*는 *NULL*이 아니어야 합니다 | 인자가 필요 없으면 빈 튜플을 사용하십시오. 이름있는 인자가 필요하지 않으면, *kwargs*는 *NULL*일 수 있습니다.

성공하면 호출 결과를 반환하고, 실패하면 예외를 발생시키고 *NULL*을 반환합니다.

이것은 다음 파이썬 표현식과 동등합니다: `callable(*args, **kwargs)`.

PyObject ***PyObject_CallNoArgs**(*PyObject* *callable)

Return value: New reference. Part of the Stable ABI *since version 3.10.* 인자 없이 콜러블 파이썬 객체 *callable*을 호출합니다. 인자 없이 콜러블 파이썬 객체를 호출하는 가장 효율적인 방법입니다.

성공하면 호출 결과를 반환하고, 실패하면 예외를 발생시키고 *NULL*을 반환합니다.

Added in version 3.9.

PyObject ***PyObject_CallOneArg**(*PyObject* *callable, *PyObject* *arg)

Return value: New reference. 정확히 1개의 위치 인자 *arg*로 키워드 인자 없이 콜러블 파이썬 객체 *callable*을 호출합니다.

성공하면 호출 결과를 반환하고, 실패하면 예외를 발생시키고 *NULL*을 반환합니다.

Added in version 3.9.

PyObject ***PyObject_CallObject**(*PyObject* *callable, *PyObject* *args)

Return value: New reference. Part of the Stable ABI. 튜플 *args*에 의해 주어진 인자로 콜러블 파이썬 객체 *callable*을 호출합니다. 인자가 필요하지 않으면 *args*는 *NULL*일 수 있습니다.

성공하면 호출 결과를 반환하고, 실패하면 예외를 발생시키고 *NULL*을 반환합니다.

이것은 다음 파이썬 표현식과 동등합니다: `callable(*args)`.

PyObject ***PyObject_CallFunction**(*PyObject* *callable, const char *format, ...)

Return value: New reference. Part of the Stable ABI. 가변 개수의 C 인자로 콜러블 파이썬 객체 *callable*을 호출합니다. C 인자는 `Py_BuildValue()` 스타일 포맷 문자열을 사용하여 기술됩니다. format은 *NULL*일 수 있으며, 인자가 제공되지 않음을 나타냅니다.

성공하면 호출 결과를 반환하고, 실패하면 예외를 발생시키고 *NULL*을 반환합니다.

이것은 다음 파이썬 표현식과 동등합니다: `callable(*args)`.

Note that if you only pass `PyObject*` args, `PyObject_CallFunctionObjArgs()` is a faster alternative.

버전 3.4에서 변경: *format*의 형이 `char *`에서 변경되었습니다.

PyObject ***PyObject_CallMethod**(*PyObject* *obj, const char *name, const char *format, ...)

Return value: New reference. Part of the Stable ABI. 가변 개수의 C 인자를 사용하여 객체 *obj*의 *name*이라는 이름의 메서드를 호출합니다. C 인자는 튜플을 생성해야 하는 `Py_BuildValue()` 포맷 문자열로 기술됩니다.

format은 *NULL*일 수 있으며, 인자가 제공되지 않음을 나타냅니다.

성공하면 호출 결과를 반환하고, 실패하면 예외를 발생시키고 *NULL*을 반환합니다.

이것은 다음 파이썬 표현식과 동등합니다: `obj.name(arg1, arg2, ...)`.

Note that if you only pass `PyObject*` args, `PyObject_CallMethodObjArgs()` is a faster alternative.

버전 3.4에서 변경: *name*과 *format*의 형이 `char *`에서 변경되었습니다.

PyObject ***PyObject_CallFunctionObjArgs**(*PyObject* *callable, ...)

Return value: New reference. Part of the Stable ABI. Call a callable Python object *callable*, with a variable number of `PyObject*` arguments. The arguments are provided as a variable number of parameters followed by *NULL*.

성공하면 호출 결과를 반환하고, 실패하면 예외를 발생시키고 *NULL*을 반환합니다.

이것은 다음 파이썬 표현식과 동등합니다: `callable(arg1, arg2, ...)`.

PyObject ***PyObject_CallMethodObjArgs**(*PyObject* *obj, *PyObject* *name, ...)

Return value: New reference. Part of the Stable ABI. Call a method of the Python object *obj*, where the name of the method is given as a Python string object in *name*. It is called with a variable number of *PyObject** arguments. The arguments are provided as a variable number of parameters followed by *NULL*.

성공하면 호출 결과를 반환하고, 실패하면 예외를 발생시키고 *NULL*을 반환합니다.

PyObject ***PyObject_CallMethodNoArgs**(*PyObject* *obj, *PyObject* *name)

인자 없이 파이썬 객체 *obj*의 메서드를 호출합니다. 여기서 메서드 이름은 *name*에서 파이썬 문자열 객체로 제공됩니다.

성공하면 호출 결과를 반환하고, 실패하면 예외를 발생시키고 *NULL*을 반환합니다.

Added in version 3.9.

PyObject ***PyObject_CallMethodOneArg**(*PyObject* *obj, *PyObject* *name, *PyObject* *arg)

단일 위치 인자 *arg*로 파이썬 객체 *obj*의 메서드를 호출합니다. 여기서 메서드 이름은 *name*에서 파이썬 문자열 객체로 제공됩니다.

성공하면 호출 결과를 반환하고, 실패하면 예외를 발생시키고 *NULL*을 반환합니다.

Added in version 3.9.

PyObject ***PyObject_Vectorcall**(*PyObject* *callable, *PyObject* *const *args, size_t nargsf, *PyObject* *kwnames)

Part of the Stable ABI *since version 3.12.* 콜러블 파이썬 객체 *callable*을 호출합니다. 인자는 *vectorcallfunc* 와 같습니다. *callable*이 벡터콜을 지원하면, *callable*에 저장된 벡터콜 함수를 직접 호출합니다.

성공하면 호출 결과를 반환하고, 실패하면 예외를 발생시키고 *NULL*을 반환합니다.

Added in version 3.9.

PyObject ***PyObject_VectorcallDict**(*PyObject* *callable, *PyObject* *const *args, size_t nargsf, *PyObject* *kwdict)

위치 인자가 벡터콜 프로토콜과 정확히 일치하지만 딕셔너리 *kwdict*로 전달된 키워드 인자로 *callable*을 호출합니다. *args* 배열은 위치 인자만 포함합니다.

내부적으로 사용되는 프로토콜과 관계없이, 인자를 변환해야 합니다. 따라서, 이 함수는 호출자에게 이미 키워드 인자로 사용할 준비가 된 딕셔너리가 있지만, 위치 인자에 대한 튜플이 없을 때만 사용해야 합니다.

Added in version 3.9.

PyObject ***PyObject_VectorcallMethod**(*PyObject* *name, *PyObject* *const *args, size_t nargsf, *PyObject* *kwnames)

Part of the Stable ABI *since version 3.12.* Call a method using the vectorcall calling convention. The name of the method is given as a Python string *name*. The object whose method is called is *args*[*0*], and the *args* array starting at *args*[*1*] represents the arguments of the call. There must be at least one positional argument. *nargsf* is the number of positional arguments including *args*[*0*], plus *PY_VECTORCALL_ARGUMENTS_OFFSET* if the value of `args[0]` may temporarily be changed. Keyword arguments can be passed just like in *PyObject_Vectorcall()*.

If the object has the *Py_TPFLAGS_METHOD_DESCRIPTOR* feature, this will call the unbound method object with the full *args* vector as arguments.

성공하면 호출 결과를 반환하고, 실패하면 예외를 발생시키고 *NULL*을 반환합니다.

Added in version 3.9.

7.2.4 호출 지원 API

int **PyCallable_Check**(*PyObject* *o)

Part of the Stable ABI. 객체 *o*가 콜러블 인지 판별합니다. 객체가 콜러블 이면 `1`을, 그렇지 않으면 `0`을 반환합니다. 이 함수는 항상 성공합니다.

7.3 숫자 프로토콜

int **PyNumber_Check**(*PyObject* *o)

Part of the Stable ABI. 객체 *o*가 숫자 프로토콜을 제공하면 `1`을 반환하고, 그렇지 않으면 거짓을 반환합니다. 이 함수는 항상 성공합니다.

버전 3.8에서 변경: *o*가 인덱스 정수면 `1`을 반환합니다.

PyObject ***PyNumber_Add**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. *o1*과 *o2*를 더한 결과나, 실패 시 `NULL`을 반환합니다. 이것은 파이썬 표현식 `o1 + o2`와 동등합니다.

PyObject ***PyNumber_Subtract**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. *o1*에서 *o2*를 뺀 결과나, 실패 시 `NULL`을 반환합니다. 이것은 파이썬 표현식 `o1 - o2`와 동등합니다.

PyObject ***PyNumber_Multiply**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. *o1*과 *o2*를 곱한 결과나, 실패 시 `NULL`을 반환합니다. 이것은 파이썬 표현식 `o1 * o2`와 동등합니다.

PyObject ***PyNumber_MatrixMultiply**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI *since version 3.7*. *o1*과 *o2*를 행렬 곱셈한 결과나, 실패 시 `NULL`을 반환합니다. 이것은 파이썬 표현식 `o1 @ o2`와 동등합니다.

Added in version 3.5.

PyObject ***PyNumber_FloorDivide**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. Return the floor of *o1* divided by *o2*, or `NULL` on failure. This is the equivalent of the Python expression `o1 // o2`.

PyObject ***PyNumber_TrueDivide**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. Return a reasonable approximation for the mathematical value of *o1* divided by *o2*, or `NULL` on failure. The return value is "approximate" because binary floating-point numbers are approximate; it is not possible to represent all real numbers in base two. This function can return a floating-point value when passed two integers. This is the equivalent of the Python expression `o1 / o2`.

PyObject ***PyNumber_Remainder**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. *o1*을 *o2*로 나눈 나머지나, 실패 시 `NULL`을 반환합니다. 이것은 파이썬 표현식 `o1 % o2`와 동등합니다.

PyObject ***PyNumber_Divmod**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. 내장 함수 `divmod()`를 참조하십시오. 실패하면 `NULL`을 반환합니다. 이것은 파이썬 표현식 `divmod(o1, o2)`와 동등합니다.

PyObject ***PyNumber_Power**(*PyObject* *o1, *PyObject* *o2, *PyObject* *o3)

Return value: New reference. Part of the Stable ABI. 내장 함수 `pow()`를 참조하십시오. 실패하면 `NULL`을 반환합니다. 이것은 파이썬 표현식 `pow(o1, o2, o3)`와 동등합니다, 여기서 *o3*는 선택적입니다. *o3*를 무시하려면, 그 자리에 *Py_None*을 전달하십시오 (*o3*에 `NULL`을 전달하면 잘못된 메모리 액세스가 발생합니다).

PyObject ***PyNumber_Negative**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 성공 시 *o*의 음의 값(negation)을, 실패 시 `NULL`을 반환합니다. 이것은 파이썬 표현식 `-o`와 동등합니다.

PyObject ***PyNumber_Positive**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 성공 시 *o*를, 실패 시 NULL을 반환합니다. 이것은 파이썬 표현식 `+o`와 동등합니다.

PyObject ***PyNumber_Absolute**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. *o*의 절댓값이나, 실패 시 NULL을 반환합니다. 이것은 파이썬 표현식 `abs(o)`와 동등합니다.

PyObject ***PyNumber_Invert**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 성공 시 *o*의 비트 반전(bitwise negation)을, 실패 시 NULL을 반환합니다. 이것은 파이썬 표현식 `~o`와 동등합니다.

PyObject ***PyNumber_Lshift**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. 성공 시 *o1*을 *o2*만큼 왼쪽으로 시프트 한 결과를, 실패 시 NULL을 반환합니다. 이것은 파이썬 표현식 `o1 << o2`와 동등합니다.

PyObject ***PyNumber_Rshift**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. 성공 시 *o1*을 *o2*만큼 오른쪽으로 시프트 한 결과를, 실패 시 NULL을 반환합니다. 이것은 파이썬 표현식 `o1 >> o2`와 동등합니다.

PyObject ***PyNumber_And**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. 성공 시 *o1*과 *o2*의 "비트별 논리곱(bitwise and)"을, 실패 시 NULL을 반환합니다. 이것은 파이썬 표현식 `o1 & o2`와 동등합니다.

PyObject ***PyNumber_Xor**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. 성공 시 *o1*과 *o2*의 "비트별 배타적 논리합(bitwise exclusive or)"을, 실패 시 NULL을 반환합니다. 이것은 파이썬 표현식 `o1 ^ o2`와 동등합니다.

PyObject ***PyNumber_Or**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. 성공 시 *o1*과 *o2*의 "비트별 논리합(bitwise or)"을, 실패 시 NULL을 반환합니다. 이것은 파이썬 표현식 `o1 | o2`와 동등합니다.

PyObject ***PyNumber_InPlaceAdd**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. *o1*과 *o2*를 더한 결과나, 실패 시 NULL을 반환합니다. 이 연산은 *o1*이 지원하면 제자리에서(*in-place*) 수행됩니다. 이것은 파이썬 문장 `o1 += o2`와 동등합니다.

PyObject ***PyNumber_InPlaceSubtract**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. *o1*에서 *o2*를 뺀 결과나, 실패 시 NULL을 반환합니다. 이 연산은 *o1*이 지원하면 제자리에서(*in-place*) 수행됩니다. 이것은 파이썬 문장 `o1 -= o2`와 동등합니다.

PyObject ***PyNumber_InPlaceMultiply**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. *o1*과 *o2*를 곱한 결과나, 실패 시 NULL을 반환합니다. 이 연산은 *o1*이 지원하면 제자리에서(*in-place*) 수행됩니다. 이것은 파이썬 문장 `o1 *= o2`와 동등합니다.

PyObject ***PyNumber_InPlaceMatrixMultiply**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI *since version 3.7.* *o1*과 *o2*를 행렬 곱셈한 결과나, 실패 시 NULL을 반환합니다. 이 연산은 *o1*이 지원하면 제자리에서(*in-place*) 수행됩니다. 이것은 파이썬 문장 `o1 @= o2`와 동등합니다.

Added in version 3.5.

PyObject ***PyNumber_InPlaceFloorDivide**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. *o1*을 *o2*로 나눈 수학적 플로어(floor)나, 실패 시 NULL을 반환합니다. 이 연산은 *o1*이 지원하면 제자리에서(*in-place*) 수행됩니다. 이것은 파이썬 문장 `o1 //= o2`와 동등합니다.

PyObject ***PyNumber_InPlaceTrueDivide**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. Return a reasonable approximation for the mathematical value of *o1* divided by *o2*, or `NULL` on failure. The return value is "approximate" because binary floating-point numbers are approximate; it is not possible to represent all real numbers in base two. This function can return a floating-point value when passed two integers. The operation is done *in-place* when *o1* supports it. This is the equivalent of the Python statement `o1 /= o2`.

PyObject ***PyNumber_InPlaceRemainder**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. *o1*을 *o2*로 나눈 나머지나, 실패 시 `NULL`을 반환합니다. 이 연산은 *o1*이 지원하면 제자리에서(*in-place*) 수행됩니다. 이것은 파이썬 문장 `o1 %= o2`와 동등합니다.

PyObject ***PyNumber_InPlacePower**(*PyObject* *o1, *PyObject* *o2, *PyObject* *o3)

Return value: New reference. Part of the Stable ABI. 내장 함수 `pow()`를 참조하십시오. 실패하면 `NULL`을 반환합니다. 이 연산은 *o1*이 지원하면 제자리에서(*in-place*) 수행됩니다. 이것은 o3가 *Py_None*일 때 파이썬 문장 `o1 **= o2`와, 그렇지 않으면 `pow(o1, o2, o3)`의 제자리 변형과 동등합니다. *o3*를 무시하려면, 그 자리에 *Py_None*을 전달하십시오 (*o3*에 `NULL`을 전달하면 잘못된 메모리 액세스가 발생합니다).

PyObject ***PyNumber_InPlaceLshift**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. 성공 시 *o1*을 *o2*만큼 왼쪽으로 시프트 한 결과를, 실패 시 `NULL`을 반환합니다. 이 연산은 *o1*이 지원하면 제자리에서(*in-place*) 수행됩니다. 이것은 파이썬 문장 `o1 <<= o2`와 동등합니다.

PyObject ***PyNumber_InPlaceRshift**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. 성공 시 *o1*을 *o2*만큼 오른쪽으로 시프트 한 결과를, 실패 시 `NULL`을 반환합니다. 이 연산은 *o1*이 지원하면 제자리에서(*in-place*) 수행됩니다. 이것은 파이썬 문장 `o1 >>= o2`와 동등합니다.

PyObject ***PyNumber_InPlaceAnd**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. 성공 시 *o1*과 *o2*의 "비트별 논리곱(bitwise and)"을, 실패 시 `NULL`을 반환합니다. 이 연산은 *o1*이 지원하면 제자리에서(*in-place*) 수행됩니다. 이것은 파이썬 문장 `o1 &= o2`와 동등합니다.

PyObject ***PyNumber_InPlaceXor**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. 성공 시 *o1*과 *o2*의 "비트별 배타적 논리합(bitwise exclusive or)"을, 실패 시 `NULL`을 반환합니다. 이 연산은 *o1*이 지원하면 제자리에서(*in-place*) 수행됩니다. 이것은 파이썬 문장 `o1 ^= o2`와 동등합니다.

PyObject ***PyNumber_InPlaceOr**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. 성공 시 *o1*과 *o2*의 "비트별 논리합(bitwise or)"을, 실패 시 `NULL`을 반환합니다. 이 연산은 *o1*이 지원하면 제자리에서(*in-place*) 수행됩니다. 이것은 파이썬 문장 `o1 |= o2`와 동등합니다.

PyObject ***PyNumber_Long**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 성공 시 정수 객체로 변환된 *o*를, 실패 시 `NULL`을 반환합니다. 이것은 파이썬 표현식 `int(o)`와 동등합니다.

PyObject ***PyNumber_Float**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 성공 시 float 객체로 변환된 *o*를, 실패 시 `NULL`을 반환합니다. 이것은 파이썬 표현식 `float(o)`와 동등합니다.

PyObject ***PyNumber_Index**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 성공 시 파이썬 int로 변환된 *o*를, 실패 시 `NULL`을 반환합니다. 실패 시 `TypeError` 예외가 발생합니다.

버전 3.10에서 변경: The result always has exact type `int`. Previously, the result could have been an instance of a subclass of `int`.

PyObject ***PyNumber_ToBase**(*PyObject* *n, int base)

Return value: New reference. Part of the Stable ABI. 정수 *n*을 진수 *base*를 사용해서 변환한 문자열을 반환합니다. *base* 인자는 2, 8, 10 또는 16중 하나여야 합니다. 진수 2, 8 또는 16의 경우, 반환된 문자열은 `'0b'`, `'0o'` 또는 `'0x'`의 진수 표시자가 각각 앞에 붙습니다. *n*이 파이썬 int가 아니면, 먼저 `PyNumber_Index()`로 변환됩니다.

Py_ssize_t **PyNumber_AsSsize_t**(*PyObject* *o, *PyObject* *exc)

Part of the Stable ABI. Returns *o* converted to a `Py_ssize_t` value if *o* can be interpreted as an integer. If the call fails, an exception is raised and `-1` is returned.

If *o* can be converted to a Python int but the attempt to convert to a `Py_ssize_t` value would raise an `OverflowError`, then the *exc* argument is the type of exception that will be raised (usually `IndexError` or `OverflowError`). If *exc* is `NULL`, then the exception is cleared and the value is clipped to `PY_SSIZE_T_MIN` for a negative integer or `PY_SSIZE_T_MAX` for a positive integer.

int **PyIndex_Check**(*PyObject* *o)

Part of the Stable ABI *since version 3.8.* Returns `1` if *o* is an index integer (has the `nb_index` slot of the `tp_as_number` structure filled in), and `0` otherwise. This function always succeeds.

7.4 시퀀스 프로토콜

int **PySequence_Check**(*PyObject* *o)

Part of the Stable ABI. Return `1` if the object provides the sequence protocol, and `0` otherwise. Note that it returns `1` for Python classes with a `__getitem__()` method, unless they are `dict` subclasses, since in general it is impossible to determine what type of keys the class supports. This function always succeeds.

Py_ssize_t **PySequence_Size**(*PyObject* *o)

Py_ssize_t **PySequence_Length**(*PyObject* *o)

Part of the Stable ABI. 성공 시 시퀀스 *o*의 객체 수를 반환하고, 실패하면 `-1`을 반환합니다. 이것은 파이썬 표현식 `len(o)`와 동등합니다.

PyObject ***PySequence_Concat**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. 성공 시 *o1*와 *o2*의 이어붙이기를 반환하고, 실패하면 `NULL`을 반환합니다. 이것은 파이썬 표현식 `o1 + o2`와 동등합니다.

PyObject ***PySequence_Repeat**(*PyObject* *o, *Py_ssize_t* count)

Return value: New reference. Part of the Stable ABI. 시퀀스 객체 *o*를 *count* 번 반복한 결과를 반환하거나, 실패 시 `NULL`을 반환합니다. 이것은 파이썬 표현식 `o * count`와 동등합니다.

PyObject ***PySequence_InPlaceConcat**(*PyObject* *o1, *PyObject* *o2)

Return value: New reference. Part of the Stable ABI. 성공 시 *o1*와 *o2*의 이어붙이기를 반환하고, 실패하면 `NULL`을 반환합니다. 이 연산은 *o1*가 지원하면 제자리에서(*in-place*) 수행됩니다. 이것은 파이썬 표현식 `o1 += o2`와 동등합니다.

PyObject ***PySequence_InPlaceRepeat**(*PyObject* *o, *Py_ssize_t* count)

Return value: New reference. Part of the Stable ABI. 시퀀스 객체 *o*를 *count* 번 반복한 결과를 반환하거나, 실패 시 `NULL`을 반환합니다. 이 연산은 *o*가 지원하면 제자리에서(*in-place*) 수행됩니다. 이것은 파이썬 표현식 `o *= count`와 동등합니다.

PyObject ***PySequence_GetItem**(*PyObject* *o, *Py_ssize_t* i)

Return value: New reference. Part of the Stable ABI. *o*의 *i* 번째 요소를 반환하거나, 실패하면 `NULL`을 반환합니다. 이것은 파이썬 표현식 `o[i]`와 동등합니다.

PyObject ***PySequence_GetSlice**(*PyObject* *o, *Py_ssize_t* i1, *Py_ssize_t* i2)

Return value: New reference. Part of the Stable ABI. 시퀀스 객체 *o*의 *i1*와 *i2* 사이의 슬라이스를 반환하거나, 실패하면 `NULL`을 반환합니다. 이것은 파이썬 표현식 `o[i1:i2]`와 동등합니다.

int **PySequence_SetItem**(*PyObject* *o, *Py_ssize_t* i, *PyObject* *v)

Part of the Stable ABI. 객체 *v*를 *o*의 *i* 번째 요소에 대입합니다. 실패하면 예외를 발생시키고 `-1`을 반환합니다; 성공하면 `0`을 반환합니다. 이것은 파이썬 문장 `o[i] = v`와 동등합니다. 이 함수는 *v*에 대한 참조를 훔치지 않습니다.

If *v* is `NULL`, the element is deleted, but this feature is deprecated in favour of using *PySequence_DelItem()*.

int **PySequence_DelItem**(*PyObject* *o, *Py_ssize_t* i)

Part of the Stable ABI. *o* 객체의 *i* 번째 요소를 삭제합니다. 실패하면 `-1`을 반환합니다. 이것은 파이썬 문장 `del o[i]`와 동등합니다.

int **PySequence_SetSlice**(*PyObject* *o, *Py_ssize_t* i1, *Py_ssize_t* i2, *PyObject* *v)

Part of the Stable ABI. 시퀀스 객체 *v*를 시퀀스 객체 *o*의 *i1*에서 *i2* 사이의 슬라이스에 대입합니다. 이것은 파이썬 문장 `o[i1:i2] = v`와 동등합니다.

int **PySequence_DelSlice**(*PyObject* *o, *Py_ssize_t* i1, *Py_ssize_t* i2)

Part of the Stable ABI. 시퀀스 객체 *o*의 *i1*에서 *i2* 사이의 슬라이스를 삭제합니다. 실패하면 `-1`을 반환합니다. 이것은 파이썬 문장 `del o[i1:i2]`와 동등합니다.

Py_ssize_t **PySequence_Count**(*PyObject* *o, *PyObject* *value)

Part of the Stable ABI. *o*에 있는 *value*의 수를 반환합니다. 즉, `o[key] == value`를 만족하는 key의 수를 반환합니다. 실패하면 `-1`을 반환합니다. 이것은 파이썬 표현식 `o.count(value)`와 동등합니다.

int **PySequence_Contains**(*PyObject* *o, *PyObject* *value)

Part of the Stable ABI. *o*에 *value*가 있는지 확인합니다. *o*의 항목 중 하나가 *value*와 같으면 `1`을 반환하고, 그렇지 않으면 `0`을 반환합니다. 에러 시 `-1`을 반환합니다. 이는 파이썬 표현식 `value in o`와 동등합니다.

Py_ssize_t **PySequence_Index**(*PyObject* *o, *PyObject* *value)

Part of the Stable ABI. `o[i] == value`을 만족하는 첫 번째 인덱스 *i*를 반환합니다. 에러 시 `-1`을 반환합니다. 이것은 파이썬 표현식 `o.index(value)`와 동등합니다.

PyObject ***PySequence_List**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 시퀀스나 이터러블 *o*와 같은 내용을 가진 리스트 객체를 반환하거나, 실패하면 `NULL`을 반환합니다. 반환된 리스트는 새로운 것으로 보장됩니다. 이것은 파이썬 표현식 `list(o)`와 동등합니다.

PyObject ***PySequence_Tuple**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 시퀀스나 이터러블 *o*와 같은 내용을 가진 튜플 객체를 반환하거나, 실패하면 `NULL`을 반환합니다. *o*가 튜플이면, 새로운 참조가 반환되고, 그렇지 않으면 튜플이 적절한 내용으로 만들어집니다. 이것은 파이썬 표현식 `tuple(o)`와 동등합니다.

PyObject ***PySequence_Fast**(*PyObject* *o, const char *m)

Return value: New reference. Part of the Stable ABI. 시퀀스나 이터러블 *o*를 다른 `PySequence_Fast*` 계열 함수에서 사용할 수 있는 객체로 반환합니다. 객체가 시퀀스나 이터러블이 아니면 *m*을 메시지 텍스트로 사용하여 `TypeError`를 발생시킵니다. 실패 시 `NULL`을 반환합니다.

`PySequence_Fast*` 함수는 *o*가 *PyTupleObject*나 *PyListObject*라고 가정하고 *o*의 데이터 필드에 직접 액세스하기 때문에 이렇게 이름 붙였습니다.

CPython 구현 세부 사항으로, *o*가 이미 시퀀스나 리스트면, 반환됩니다.

Py_ssize_t **PySequence_Fast_GET_SIZE**(*PyObject* *o)

Returns the length of *o*, assuming that *o* was returned by *PySequence_Fast()* and that *o* is not `NULL`. The size can also be retrieved by calling *PySequence_Size()* on *o*, but *PySequence_Fast_GET_SIZE()* is faster because it can assume *o* is a list or tuple.

PyObject ***PySequence_Fast_GET_ITEM**(*PyObject* *o, *Py_ssize_t* i)

Return value: Borrowed reference. *o*의 *i* 번째 요소를 반환하는데, *o*가 *PySequence_Fast()*에 의해 반환되었고, *o*가 `NULL`이 아니며, *i*가 경계 내에 있다고 가정합니다.

PyObject ***PySequence_Fast_ITEMS**(*PyObject* *o)

PyObject 포인터의 하부 배열을 반환합니다. *o*가 *PySequence_Fast()*에 의해 반환되었고, *o*가 NULL이 아니라고 가정합니다.

리스트의 크기가 변경되면, 재할당이 항목 배열을 재배치할 수 있음에 유의하십시오. 따라서, 시퀀스가 변경될 수 없는 문맥에서만 하부 배열 포인터를 사용하십시오.

PyObject ***PySequence_ITEM**(*PyObject* *o, *Py_ssize_t* i)

Return value: New reference. *o*의 *i* 번째 요소를 반환하거나, 실패하면 NULL을 반환합니다. *PySequence_GetItem()*의 빠른 형식이지만, *o*에 대해 *PySequence_Check()*가 참인지 검사하지 않고, 음수 인덱스를 조정하지 않습니다.

7.5 매핑 프로토콜

PyObject_GetItem(), *PyObject_SetItem()* 및 *PyObject_DelItem()*도 참조하십시오.

int **PyMapping_Check**(*PyObject* *o)

Part of the Stable ABI. Return 1 if the object provides the mapping protocol or supports slicing, and 0 otherwise. Note that it returns 1 for Python classes with a `__getitem__()` method, since in general it is impossible to determine what type of keys the class supports. This function always succeeds.

Py_ssize_t **PyMapping_Size**(*PyObject* *o)

Py_ssize_t **PyMapping_Length**(*PyObject* *o)

Part of the Stable ABI. 성공 시 객체 *o*의 키 수를 반환하고, 실패하면 -1을 반환합니다. 이는 파이썬 표현식 `len(o)`와 동등합니다.

PyObject ***PyMapping_GetItemString**(*PyObject* *o, const char *key)

Return value: New reference. *Part of the* Stable ABI. This is the same as *PyObject_GetItem()*, but *key* is specified as a `const char*` UTF-8 encoded bytes string, rather than a *PyObject**.

int **PyMapping_SetItemString**(*PyObject* *o, const char *key, *PyObject* *v)

Part of the Stable ABI. This is the same as *PyObject_SetItem()*, but *key* is specified as a `const char*` UTF-8 encoded bytes string, rather than a *PyObject**.

int **PyMapping_DelItem**(*PyObject* *o, *PyObject* *key)

This is an alias of *PyObject_DelItem()*.

int **PyMapping_DelItemString**(*PyObject* *o, const char *key)

This is the same as *PyObject_DelItem()*, but *key* is specified as a `const char*` UTF-8 encoded bytes string, rather than a *PyObject**.

int **PyMapping_HasKey**(*PyObject* *o, *PyObject* *key)

Part of the Stable ABI. 매핑 객체에 *key* 키가 있으면 1을 반환하고, 그렇지 않으면 0을 반환합니다. 이는 파이썬 표현식 `key in o`와 동등합니다. 이 함수는 항상 성공합니다.

> **참고**
>
> Exceptions which occur when this calls `__getitem__()` method are silently ignored. For proper error handling, use *PyObject_GetItem()* instead.

int **PyMapping_HasKeyString**(*PyObject* *o, const char *key)

Part of the Stable ABI. This is the same as *PyMapping_HasKey()*, but *key* is specified as a `const char*` UTF-8 encoded bytes string, rather than a *PyObject**.

> **참고**
>
> Exceptions that occur when this calls `__getitem__()` method or while creating the temporary `str` object are silently ignored. For proper error handling, use *PyMapping_GetItemString()* instead.

PyObject ***PyMapping_Keys**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 성공하면, 객체 *o*의 키 리스트를 반환합니다. 실패하면, NULL을 반환합니다.

버전 3.7에서 변경: 이전에는 함수가 리스트나 튜플을 반환했습니다.

PyObject ***PyMapping_Values**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 성공하면, 객체 *o*의 값 리스트를 반환합니다. 실패하면, NULL을 반환합니다.

버전 3.7에서 변경: 이전에는 함수가 리스트나 튜플을 반환했습니다.

PyObject ***PyMapping_Items**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 성공하면, 객체 *o*에 있는 항목 리스트를 반환합니다. 여기서 각 항목은 키-값 쌍을 포함하는 튜플입니다. 실패하면, NULL을 반환합니다.

버전 3.7에서 변경: 이전에는 함수가 리스트나 튜플을 반환했습니다.

7.6 이터레이터 프로토콜

특히 이터레이터를 사용하기 위한 두 함수가 있습니다.

int **PyIter_Check**(*PyObject* *o)

Part of the Stable ABI *since version 3.8.* Return non-zero if the object *o* can be safely passed to *PyIter_Next()*, and 0 otherwise. This function always succeeds.

int **PyAIter_Check**(*PyObject* *o)

Part of the Stable ABI *since version 3.10.* Return non-zero if the object *o* provides the AsyncIterator protocol, and 0 otherwise. This function always succeeds.

Added in version 3.10.

PyObject ***PyIter_Next**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. Return the next value from the iterator *o*. The object must be an iterator according to *PyIter_Check()* (it is up to the caller to check this). If there are no remaining values, returns NULL with no exception set. If an error occurs while retrieving the item, returns NULL and passes along the exception.

이터레이터를 이터레이트하는 루프를 작성하려면, C 코드는 이런 식으로 되어야 합니다:

```
PyObject *iterator = PyObject_GetIter(obj);
PyObject *item;

if (iterator == NULL) {
    /* propagate error */
}

while ((item = PyIter_Next(iterator))) {
    /* do something with item */
    ...
    /* release reference when done */
    Py_DECREF(item);
}

Py_DECREF(iterator);

if (PyErr_Occurred()) {
    /* propagate error */
}
else {
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    /* continue doing useful work */
}
```

type **PySendResult**

The enum value used to represent different results of *PyIter_Send()*.

Added in version 3.10.

PySendResult **PyIter_Send**(*PyObject* *iter, *PyObject* *arg, *PyObject* **presult)

Part of the Stable ABI *since version 3.10.* Sends the *arg* value into the iterator *iter*. Returns:

- PYGEN_RETURN if iterator returns. Return value is returned via *presult*.
- PYGEN_NEXT if iterator yields. Yielded value is returned via *presult*.
- PYGEN_ERROR if iterator has raised and exception. *presult* is set to NULL.

Added in version 3.10.

7.7 버퍼 프로토콜

파이썬에서 사용할 수 있는 어떤 객체는 하부 메모리 배열 또는 버퍼에 대한 액세스를 감쌉니다. 이러한 객체에는 내장 bytes 와 bytearray, 그리고 array.array와 같은 일부 확장형이 포함됩니다. 제삼자 라이브러리도 이미지 처리나 수치 해석과 같은 특수한 용도로 자체 형을 정의할 수 있습니다.

이러한 형은 각각 고유의 의미가 있지만, (아마도) 큰 메모리 버퍼에 의해 뒷받침되는 공통된 특징을 공유합니다. 어떤 상황에서는 중간 복사 없이 직접 버퍼에 액세스하는 것이 바람직합니다.

파이썬은 C 수준에서 버퍼 프로토콜 형식으로 이러한 기능을 제공합니다. 이 프로토콜에는 두 가지 측면이 있습니다:

- 생산자 측에서는, 형이 "버퍼 인터페이스"를 내보낼 수 있는데, 그 형의 객체가 하부 버퍼의 정보를 노출할 수 있게 합니다. 이 인터페이스는 버퍼 객체 구조체 절에서 설명됩니다.
- 소비자 측에서는, 객체의 원시 하부 데이터에 대한 포인터를 얻기 위해 여러 가지 방법을 사용할 수 있습니다 (예를 들어 메서드 매개 변수).

bytes 와 bytearray와 같은 간단한 객체는 하부 버퍼를 바이트 지향 형식으로 노출합니다. 다른 형태도 가능합니다; 예를 들어, array.array에 의해 노출되는 요소는 멀티 바이트 값이 될 수 있습니다.

An example consumer of the buffer interface is the write() method of file objects: any object that can export a series of bytes through the buffer interface can be written to a file. While write() only needs read-only access to the internal contents of the object passed to it, other methods such as readinto() need write access to the contents of their argument. The buffer interface allows objects to selectively allow or reject exporting of read-write and read-only buffers.

버퍼 인터페이스의 소비자가 대상 객체에 대해 버퍼를 얻는 방법에는 두 가지가 있습니다:

- 올바른 매개 변수로 *PyObject_GetBuffer()*를 호출합니다;
- y*, w* 또는 s* 형식 코드 중 하나를 사용하여 *PyArg_ParseTuple()* (또는 그 형제 중 하나)을 호출합니다.

두 경우 모두, 버퍼가 더는 필요하지 않으면 *PyBuffer_Release()*를 호출해야 합니다. 그렇게 하지 않으면 자원 누수와 같은 다양한 문제가 발생할 수 있습니다.

7.7.1 버퍼 구조체

버퍼 구조체 (또는 단순히 "버퍼")는 다른 객체의 바이너리 데이터를 파이썬 프로그래머에게 노출하는 방법으로 유용합니다. 또한, 복사 없는 (zero-copy) 슬라이싱 메커니즘으로 사용할 수 있습니다. 메모리 블록을 참조하는 능력을 사용해서, 임의의 데이터를 파이썬 프로그래머에게 아주 쉽게 노출할 수 있습니다. 메모리는 C 확장의 큰 상수 배열일 수 있으며, 운영 체제 라이브러리로 전달되기 전에 조작하기 위한 원시 메모리 블록일 수도 있고, 네이티브 인 메모리 (in-memory) 형식으로 구조화된 데이터를 전달하는 데 사용될 수도 있습니다.

파이썬 인터프리터가 노출하는 대부분의 데이터형과 달리, 버퍼는 *PyObject* 포인터가 아니라 단순한 C 구조체입니다. 이를 통해 매우 간단하게 만들고 복사할 수 있습니다. 버퍼를 감싸는 일반 래퍼가 필요할 때는, 메모리 뷰 객체를 만들 수 있습니다.

제공하는(exporting) 객체를 작성하는 간단한 지침은 버퍼 객체 구조체를 참조하십시오. 버퍼를 얻으려면, *PyObject_GetBuffer()*를 참조하십시오.

type **Py_buffer**

Part of the Stable ABI (*including all members*) *since version 3.11.*

void ***buf**

버퍼 필드에 의해 기술된 논리적 구조의 시작을 가리키는 포인터. 이것은 제공자(exporter)의 하부 물리적 메모리 블록 내의 모든 위치일 수 있습니다. 예를 들어, 음의 *strides*를 사용하면 값이 메모리 블록의 끝을 가리킬 수 있습니다.

연속 배열의 경우, 값은 메모리 블록의 시작을 가리킵니다.

PyObject ***obj**

A new reference to the exporting object. The reference is owned by the consumer and automatically released (i.e. reference count decremented) and set to `NULL` by *PyBuffer_Release()*. The field is the equivalent of the return value of any standard C-API function.

특수한 경우로, *PyMemoryView_FromBuffer()* 나 *PyBuffer_FillInfo()*로 감싸진 임시(*temporary*) 버퍼의 경우, 이 필드는 `NULL`입니다. 일반적으로, 제공하는(exporting) 객체는 이 체계를 사용하지 않아야 합니다.

Py_ssize_t **len**

`product(shape) * itemsize`. 연속 배열의 경우, 하부 메모리 블록의 길이입니다. 불연속 배열의 경우, 연속 표현으로 복사된다면 논리적 구조체가 갖게 될 길이입니다.

`((char *)buf)[0]` 에서 `((char *)buf)[len-1]` 범위의 액세스는 연속성을 보장하는 요청으로 버퍼가 확보된 경우에만 유효합니다. 대부분 이러한 요청은 *PyBUF_SIMPLE* 또는 *PyBUF_WRITABLE*입니다.

int **readonly**

버퍼가 읽기 전용인지를 나타내는 표시기입니다. 이 필드는 *PyBUF_WRITABLE* 플래그로 제어됩니다.

Py_ssize_t **itemsize**

단일 요소의 항목 크기(바이트)입니다. `NULL`이 아닌 *format* 값에 호출된 `struct.calcsize()` 값과 같습니다.

중요한 예외: 소비자가 *PyBUF_FORMAT* 플래그 없이 버퍼를 요청하면, *format*은 `NULL`로 설정되지만, *itemsize*는 여전히 원래 형식의 값을 갖습니다.

*shape*이 있으면, `product(shape) * itemsize == len` 동치가 계속 성립하고 소비자는 *itemsize*를 사용하여 버퍼를 탐색할 수 있습니다.

*PyBUF_SIMPLE*이나 *PyBUF_WRITABLE* 요청의 결과로 *shape*이 `NULL`이면, 소비자는 *itemsize*를 무시하고 `itemsize == 1`로 가정해야 합니다.

char ***format**

A *NULL* terminated string in `struct` module style syntax describing the contents of a single item. If this is `NULL`, `"B"` (unsigned bytes) is assumed.

이 필드는 *PyBUF_FORMAT* 플래그로 제어됩니다.

int **ndim**

The number of dimensions the memory represents as an n-dimensional array. If it is `0`, *buf* points to a single item representing a scalar. In this case, *shape*, *strides* and *suboffsets* MUST be `NULL`. The maximum number of dimensions is given by *PyBUF_MAX_NDIM*.

Py_ssize_t ***shape**

n-차원 배열로 메모리의 모양을 나타내는 길이 *ndim*의 *Py_ssize_t* 배열. `shape[0] * ... * shape[ndim-1] * itemsize`는 *len*과 같아야 합니다.

모양 값은 `shape[n] >= 0`로 제한됩니다. `shape[n] == 0`인 경우는 특별한 주의가 필요합니다. 자세한 정보는 복잡한 배열을 참조하십시오.

shape 배열은 소비자에게 읽기 전용입니다.

Py_ssize_t ***strides**

각 차원에서 새 요소를 가져오기 위해 건너뛸 바이트 수를 제공하는 길이 *ndim*의 *Py_ssize_t* 배열.

스트라이드 값은 임의의 정수일 수 있습니다. 일반 배열의 경우, 스트라이드는 보통 양수이지만, 소비자는 `strides[n] <= 0`인 경우를 처리할 수 있어야 합니다. 자세한 내용은 복잡한 배열을 참조하십시오.

strides 배열은 소비자에게 읽기 전용입니다.

Py_ssize_t ***suboffsets**

길이 *ndim*의 *Py_ssize_t* 배열. `suboffsets[n] >= 0` 면, n 번째 차원을 따라 저장된 값은 포인터이고 서브 오프셋 값은 역참조(de-referencing) 후 각 포인터에 더할 바이트 수를 나타냅니다. 음의 서브 오프셋 값은 역참조(de-referencing)가 발생하지 않아야 함을 나타냅니다(연속 메모리 블록에서의 스트라이드).

모든 서브 오프셋이 음수면 (즉, 역참조가 필요하지 않으면), 이 필드는 `NULL`(기본값) 이어야 합니다.

이 유형의 배열 표현은 파이썬 이미징 라이브러리(PIL)에서 사용됩니다. 이러한 배열 요소에 액세스하는 방법에 대한 자세한 내용은 복잡한 배열을 참조하십시오.

suboffsets 배열은 소비자에게 읽기 전용입니다.

void ***internal**

이것은 제공하는(exporting) 객체에 의해 내부적으로 사용됩니다. 예를 들어, 이것은 제공자(exporter)가 정수로 다시 캐스팅할 수 있으며, 버퍼가 해제될 때 shape, strides 및 suboffsets 배열을 해제해야 하는지에 대한 플래그를 저장하는 데 사용됩니다. 소비자가 이 값을 변경해서는 안 됩니다.

Constants:

PyBUF_MAX_NDIM

The maximum number of dimensions the memory represents. Exporters MUST respect this limit, consumers of multi-dimensional buffers SHOULD be able to handle up to `PyBUF_MAX_NDIM` dimensions. Currently set to 64.

7.7.2 버퍼 요청 유형

버퍼는 대개 *PyObject_GetBuffer()*를 통해 제공하는(exporting) 객체로 버퍼 요청을 보내서 얻습니다. 메모리의 논리적 구조의 복잡성이 크게 다를 수 있으므로, 소비자는 처리할 수 있는 정확한 버퍼 유형을 지정하기 위해 *flags* 인자를 사용합니다.

All *Py_buffer* fields are unambiguously defined by the request type.

요청 독립적 필드

다음 필드는 *flags*의 영향을 받지 않고 항상 올바른 값으로 채워져야 합니다: *obj*, *buf*, *len*, *itemsize*, *ndim*.

readonly, format

PyBUF_WRITABLE

Controls the *readonly* field. If set, the exporter MUST provide a writable buffer or else report failure. Otherwise, the exporter MAY provide either a read-only or writable buffer, but the choice MUST be consistent for all consumers. For example, *PyBUF_SIMPLE* | *PyBUF_WRITABLE* can be used to request a simple writable buffer.

PyBUF_FORMAT

format 필드를 제어합니다. 설정되면, 이 필드를 올바르게 채워야 합니다. 그렇지 않으면, 이 필드는 반드시 NULL 이어야 합니다.

*PyBUF_WRITABLE*은 다음 섹션의 모든 플래그와 | 될 수 있습니다. *PyBUF_SIMPLE*이 0으로 정의되므로, *PyBUF_WRITABLE*은 독립형 플래그로 사용되어 간단한 쓰기 가능한 버퍼를 요청할 수 있습니다.

PyBUF_FORMAT must be |' d to any of the flags except *PyBUF_SIMPLE*, because the latter already implies format B (unsigned bytes). PyBUF_FORMAT cannot be used on its own.

shape, strides, suboffsets

메모리의 논리 구조를 제어하는 플래그는 복잡도가 감소하는 순서로 나열됩니다. 각 플래그는 그 아래에 있는 플래그의 모든 비트를 포함합니다.

요청	shape	strides	suboffsets
PyBUF_INDIRECT	yes	yes	필요하면
PyBUF_STRIDES	yes	yes	NULL
PyBUF_ND	yes	NULL	NULL
PyBUF_SIMPLE	NULL	NULL	NULL

연속성 요청

C 나 포트란 연속성을 명시적으로 요청할 수 있는데, 스트라이드 정보를 포함하기도 그렇지 않기도 합니다. 스트라이드 정보가 없으면, 버퍼는 C-연속이어야 합니다.

요청	shape	strides	suboffsets	연속성
PyBUF_C_CONTIGUOUS	yes	yes	NULL	C
PyBUF_F_CONTIGUOUS	yes	yes	NULL	F
PyBUF_ANY_CONTIGUOUS	yes	yes	NULL	C 또는 F
PyBUF_ND	yes	NULL	NULL	C

복합 요청

모든 가능한 요청은 앞 절의 플래그 조합에 의해 완전히 정의됩니다. 편의상, 버퍼 프로토콜은 자주 사용되는 조합을 단일 플래그로 제공합니다.

다음 표에서 *U*는 정의되지 않은 연속성을 나타냅니다. 소비자는 연속성을 판단하기 위해 *PyBuffer_IsContiguous()*를 호출해야 합니다.

요청	shape	strides	suboffsets	연속성	readonly	format
PyBUF_FULL	yes	yes	필요하면	U	0	yes
PyBUF_FULL_RO	yes	yes	필요하면	U	1 또는 0	yes
PyBUF_RECORDS	yes	yes	NULL	U	0	yes
PyBUF_RECORDS_RO	yes	yes	NULL	U	1 또는 0	yes
PyBUF_STRIDED	yes	yes	NULL	U	0	NULL
PyBUF_STRIDED_RO	yes	yes	NULL	U	1 또는 0	NULL
PyBUF_CONTIG	yes	NULL	NULL	C	0	NULL
PyBUF_CONTIG_RO	yes	NULL	NULL	C	1 또는 0	NULL

7.7.3 복잡한 배열

NumPy-스타일: shape과 strides

NumPy 스타일 배열의 논리적 구조는 *itemsize*, *ndim*, *shape* 및 *strides*로 정의됩니다.

`ndim == 0`이면, *buf*가 가리키는 메모리 위치가 *itemsize* 크기의 스칼라로 해석됩니다. 이 경우, *shape*과 *strides*는 모두 `NULL`입니다.

*strides*가 `NULL`이면, 배열은 표준 n-차원 C 배열로 해석됩니다. 그렇지 않으면, 소비자는 다음과 같이 n-차원 배열에 액세스해야 합니다:

```
ptr = (char *)buf + indices[0] * strides[0] + ... + indices[n-1] * strides[n-1];
item = *((typeof(item) *)ptr);
```

위에서 언급했듯이, *buf*는 실제 메모리 블록 내의 모든 위치를 가리킬 수 있습니다. 제공자(exporter)는 이 함수로 버퍼의 유효성을 검사 할 수 있습니다:

```
def verify_structure(memlen, itemsize, ndim, shape, strides, offset):
    """Verify that the parameters represent a valid array within
       the bounds of the allocated memory:
           char *mem: start of the physical memory block
           memlen: length of the physical memory block
           offset: (char *)buf - mem
    """
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    if offset % itemsize:
        return False
    if offset < 0 or offset+itemsize > memlen:
        return False
    if any(v % itemsize for v in strides):
        return False

    if ndim <= 0:
        return ndim == 0 and not shape and not strides
    if 0 in shape:
        return True

    imin = sum(strides[j]*(shape[j]-1) for j in range(ndim)
               if strides[j] <= 0)
    imax = sum(strides[j]*(shape[j]-1) for j in range(ndim)
               if strides[j] > 0)

    return 0 <= offset+imin and offset+imax+itemsize <= memlen
```

PIL-스타일: shape, strides 및 suboffsets

일반 항목 외에도, PIL 스타일 배열에는 차원의 다음 요소를 가져오기 위해 따라야 하는 포인터가 포함될 수 있습니다. 예를 들어, 일반 3-차원 C 배열 `char v[2][2][3]`는 2개의 2-차원 배열을 가리키는 2개의 포인터 배열로 볼 수도 있습니다: `char (*v[2])[2][3]`. suboffsets 표현에서, 이 두 포인터는 *buf*의 시작 부분에 임베드 될 수 있는데, 메모리의 어느 위치 에나 배치될 수 있는 두 개의 `char x[2][3]` 배열을 가리킵니다.

다음은 `NULL`이 아닌 strides와 suboffsets가 있을 때, N-차원 인덱스가 가리키는 N-차원 배열의 요소에 대한 포인터를 반환하는 함수입니다:

```
void *get_item_pointer(int ndim, void *buf, Py_ssize_t *strides,
                       Py_ssize_t *suboffsets, Py_ssize_t *indices) {
    char *pointer = (char*)buf;
    int i;
    for (i = 0; i < ndim; i++) {
        pointer += strides[i] * indices[i];
        if (suboffsets[i] >=0 ) {
            pointer = *((char**)pointer) + suboffsets[i];
        }
    }
    return (void*)pointer;
}
```

7.7.4 버퍼 관련 함수

int **PyObject_CheckBuffer**(*PyObject* *obj)

Part of the Stable ABI *since version 3.11.* *obj*가 버퍼 인터페이스를 지원하면 `1`을 반환하고, 그렇지 않으면 `0`을 반환합니다. `1`이 반환될 때, *PyObject_GetBuffer()*가 성공할 것이라고 보장하지는 않습니다. 이 함수는 항상 성공합니다.

int **PyObject_GetBuffer**(*PyObject* *exporter, *Py_buffer* *view, int flags)

Part of the Stable ABI *since version 3.11.* Send a request to *exporter* to fill in *view* as specified by *flags*. If the exporter cannot provide a buffer of the exact type, it MUST raise `BufferError`, set `view->obj` to `NULL` and return `-1`.

성공하면, *view*를 채우고, `view->obj`를 *exporter*에 대한 새 참조로 설정하고, 0을 반환합니다. 요청을 단일 객체로 리디렉션하는 연결된(chained) 버퍼 공급자의 경우, `view->obj`는 *exporter* 대신 이 객체를 참조할 수 있습니다 (버퍼 객체 구조체를 보세요).

*PyObject_GetBuffer()*에 대한 성공적인 호출은 *PyBuffer_Release()*에 대한 호출과 쌍을 이루어야 합니다, `malloc()`과 `free()`와 유사합니다. 따라서, 소비자가 버퍼로 작업한 후에는, *PyBuffer_Release()*를 정확히 한 번 호출해야 합니다.

void **PyBuffer_Release**(*Py_buffer* *view)

Part of the Stable ABI *since version 3.11.* Release the buffer *view* and release the *strong reference* (i.e. decrement the reference count) to the view' s supporting object, `view->obj`. This function MUST be called when the buffer is no longer being used, otherwise reference leaks may occur.

*PyObject_GetBuffer()*를 통해 얻지 않은 버퍼에 이 함수를 호출하는 것은 에러입니다.

Py_ssize_t **PyBuffer_SizeFromFormat**(const char *format)

Part of the Stable ABI *since version 3.11.* Return the implied *itemsize* from *format*. On error, raise an exception and return -1.

Added in version 3.9.

int **PyBuffer_IsContiguous**(const *Py_buffer* *view, char order)

Part of the Stable ABI *since version 3.11.* *view*로 정의된 메모리가 C 스타일(*order*가 `'C'`)이나 포트란 스타일(*order*가 `'F'`) 연속이거나 둘 중 하나(*order*가 `'A'`)면 `1`을 반환합니다. 그렇지 않으면 `0`을 반환합니다. 이 함수는 항상 성공합니다.

void ***PyBuffer_GetPointer**(const *Py_buffer* *view, const *Py_ssize_t* *indices)

Part of the Stable ABI *since version 3.11.* 주어진 *view* 내부의 *indices*가 가리키는 메모리 영역을 가져옵니다. *indices*는 `view->ndim` 인덱스의 배열을 가리켜야 합니다.

int **PyBuffer_FromContiguous**(const *Py_buffer* *view, const void *buf, *Py_ssize_t* len, char fort)

Part of the Stable ABI *since version 3.11.* *buf*에 있는 연속된 *len* 바이트를 *view*로 복사합니다. *fort*는 `'C'` 또는 `'F'`(C 스타일 또는 포트란 스타일 순서)일 수 있습니다. 성공하면 `0`이 반환되고, 에러가 있으면 `-1`이 반환됩니다.

int **PyBuffer_ToContiguous**(void *buf, const *Py_buffer* *src, *Py_ssize_t* len, char order)

Part of the Stable ABI *since version 3.11.* *src*에 있는 *len* 바이트를 *buf*에 연속 표현으로 복사합니다. *order*는 `'C'` 또는 `'F'` 또는 `'A'`(C 스타일 또는 포트란 스타일 순서 또는 둘 중 하나)일 수 있습니다. 성공하면 `0`이 반환되고, 에러가 있으면 `-1`이 반환됩니다.

이 함수는 *len* != *src->len*이면 실패합니다.

int **PyObject_CopyData**(*PyObject* *dest, *PyObject* *src)

Part of the Stable ABI *since version 3.11.* Copy data from *src* to *dest* buffer. Can convert between C-style and or Fortran-style buffers.

`0` is returned on success, `-1` on error.

void **PyBuffer_FillContiguousStrides**(int ndims, *Py_ssize_t* *shape, *Py_ssize_t* *strides, int itemsize, char order)

Part of the Stable ABI *since version 3.11.* *strides* 배열을 주어진 요소당 바이트 수와 주어진 shape 으로 연속 (*order*가 `'C'`면 C 스타일, *order*가 `'F'`면 포트란 스타일) 배열의 바이트 스트라이드로 채웁니다.

int **PyBuffer_FillInfo**(*Py_buffer* *view, *PyObject* *exporter, void *buf, *Py_ssize_t* len, int readonly, int flags)

Part of the Stable ABI *since version 3.11.* *readonly*에 따라 쓰기 가능성이 설정된 *len* 크기의 *buf*를 노출하려는 제공자(exporter)에 대한 버퍼 요청을 처리합니다. *buf*는 부호 없는 바이트의 시퀀스로 해석됩니다.

flags 인자는 요청 유형을 나타냅니다. 이 함수는 *buf*가 읽기 전용으로 지정되고 *PyBUF_WRITABLE*이 *flags*에 설정되어 있지 않으면, 항상 플래그가 지정하는 대로 *view*를 채웁니다.

On success, set `view->obj` to a new reference to *exporter* and return 0. Otherwise, raise `BufferError`, set `view->obj` to `NULL` and return `-1`;

이 함수가 *getbufferproc*의 일부로 사용되면, *exporter*가 제공하는(exporting) 객체로 설정되어야 하고, *flags*는 수정되지 않은 채로 전달되어야 합니다. 그렇지 않으면 *exporter*가 `NULL`이어야 합니다.

7.8 낡은 버퍼 프로토콜

버전 3.0부터 폐지됨.

이 함수는 파이썬 2에서 "낡은 버퍼 프로토콜" API 일부분이었습니다. 파이썬 3에서는 이 프로토콜이 더는 존재하지 않지만 2.x 코드 이식을 쉽게 하도록 함수들은 여전히 노출됩니다. 이들은 새 버퍼 프로토콜를 둘러싼 호환성 래퍼 역할을 하지만, 버퍼를 제공할 때 얻은 자원의 수명을 제어할 수는 없습니다.

따라서, *PyObject_GetBuffer()*(또는 `y*` 나 `w*` 포맷 코드를 사용하는 *PyArg_ParseTuple()* 계열의 함수)를 호출하여 개체에 대한 버퍼 뷰를 가져오고, 버퍼 뷰를 해제할 수 있을 때 *PyBuffer_Release()*를 호출하는 것이 좋습니다.

int **PyObject_AsCharBuffer**(*PyObject* *obj, const char **buffer, *Py_ssize_t* *buffer_len)

Part of the Stable ABI. 문자 기반 입력으로 사용할 수 있는 읽기 전용 메모리 위치에 대한 포인터를 반환합니다. *obj* 인자는 단일 세그먼트 문자 버퍼 인터페이스를 지원해야 합니다. 성공하면, `0`을 반환하고, *buffer*를 메모리 위치로 설정하고, *buffer_len*을 버퍼 길이로 설정합니다. 에러 시에, `-1`을 반환하고, `TypeError`를 설정합니다.

int **PyObject_AsReadBuffer**(*PyObject* *obj, const void **buffer, *Py_ssize_t* *buffer_len)

Part of the Stable ABI. 임의의 데이터를 포함하는 읽기 전용 메모리 위치에 대한 포인터를 반환합니다. *obj* 인자는 단일 세그먼트 읽기 가능 버퍼 인터페이스를 지원해야 합니다. 성공하면, `0`을 반환하고, *buffer*를 메모리 위치로 설정하고, *buffer_len*을 버퍼 길이로 설정합니다. 에러 시에, `-1`을 반환하고, `TypeError`를 설정합니다.

int **PyObject_CheckReadBuffer**(*PyObject* *o)

Part of the Stable ABI. *o*가 단일 세그먼트 읽기 가능 버퍼 인터페이스를 지원하면 `1`을 반환합니다. 그렇지 않으면, `0`을 반환합니다. 이 함수는 항상 성공합니다.

이 함수는 버퍼를 가져오고 해제하려고 하며, 해당 함수를 호출하는 동안 발생하는 예외는 억제됨에 유의하십시오. 에러 보고를 받으려면 대신 *PyObject_GetBuffer()*를 사용하십시오.

int **PyObject_AsWriteBuffer**(*PyObject* *obj, void **buffer, *Py_ssize_t* *buffer_len)

Part of the Stable ABI. 쓰기 가능한 메모리 위치에 대한 포인터를 반환합니다. *obj* 인자는 단일 세그먼트, 문자 버퍼 인터페이스를 지원해야 합니다. 성공하면, `0`을 반환하고, *buffer*를 메모리 위치로 설정하고, *buffer_len*을 버퍼 길이로 설정합니다. 에러 시에, `-1`을 반환하고, `TypeError`를 설정합니다.

CHAPTER 8

구상 객체 계층

이 장의 함수는 특정 파이썬 객체 형에게만 적용됩니다. 그들에게 잘못된 형의 객체를 전달하는 것은 좋은 생각이 아닙니다; 파이썬 프로그램에서 객체를 받았는데 올바른 형을 가졌는지 확실하지 않다면, 먼저 형 검사를 수행해야 합니다; 예를 들어, 객체가 딕셔너리인지 확인하려면, *PyDict_Check()*를 사용하십시오. 이 장은 파이썬 객체 형의 "족보" 처럼 구성되어 있습니다.

경고

이 장에서 설명하는 함수는 전달되는 객체의 형을 주의 깊게 검사하지만, 많은 함수는 유효한 객체 대신 전달되는 NULL을 확인하지 않습니다. NULL을 전달하면 메모리 액세스 위반이 발생하고 인터프리터가 즉시 종료될 수 있습니다.

8.1 기본 객체

이 절에서는 파이썬 형 객체와 싱글톤 객체 None에 대해 설명합니다.

8.1.1 형 객체

type **PyTypeObject**

Part of the Limited API (*as an opaque struct*). 내장형을 기술하는 데 사용되는 객체의 C 구조체.

PyTypeObject **PyType_Type**

Part of the Stable ABI. 이것은 형 객체의 형 객체입니다; 파이썬 계층의 type과 같은 객체입니다.

int **PyType_Check**(*PyObject* *o)

객체 *o*가 표준형 객체에서 파생된 형의 인스턴스를 포함하여 형 객체면 0이 아닌 값을 반환합니다. 다른 모든 경우 0을 반환합니다. 이 함수는 항상 성공합니다.

int **PyType_CheckExact**(*PyObject* *o)

객체 *o*가 형 객체이지만, 표준형 객체의 서브 형이 아니면 0이 아닌 값을 반환합니다. 다른 모든 경우 0을 반환합니다. 이 함수는 항상 성공합니다.

unsigned int **PyType_ClearCache**()

Part of the Stable ABI. 내부 조회 캐시를 지웁니다. 현재의 버전 태그를 반환합니다.

unsigned long **PyType_GetFlags**(*PyTypeObject* *type)

Part of the Stable ABI. Return the `tp_flags` member of *type*. This function is primarily meant for use with `Py_LIMITED_API`; the individual flag bits are guaranteed to be stable across Python releases, but access to `tp_flags` itself is not part of the *limited API*.

Added in version 3.2.

버전 3.4에서 변경: 반환형은 이제 `long`이 아니라 `unsigned long`입니다.

PyObject ***PyType_GetDict**(*PyTypeObject* *type)

Return the type object' s internal namespace, which is otherwise only exposed via a read-only proxy (`cls.__dict__`). This is a replacement for accessing `tp_dict` directly. The returned dictionary must be treated as read-only.

This function is meant for specific embedding and language-binding cases, where direct access to the dict is necessary and indirect access (e.g. via the proxy or `PyObject_GetAttr()`) isn' t adequate.

Extension modules should continue to use `tp_dict`, directly or indirectly, when setting up their own types.

Added in version 3.12.

void **PyType_Modified**(*PyTypeObject* *type)

Part of the Stable ABI. 형과 그것의 모든 서브 형에 대한 내부 검색 캐시를 무효로 합니다. 형의 어트리뷰트나 베이스 클래스를 수동으로 수정한 후에는 이 함수를 호출해야 합니다.

int **PyType_AddWatcher**(*PyType_WatchCallback* callback)

Register *callback* as a type watcher. Return a non-negative integer ID which must be passed to future calls to `PyType_Watch()`. In case of error (e.g. no more watcher IDs available), return `-1` and set an exception.

Added in version 3.12.

int **PyType_ClearWatcher**(int watcher_id)

Clear watcher identified by *watcher_id* (previously returned from `PyType_AddWatcher()`). Return `0` on success, `-1` on error (e.g. if *watcher_id* was never registered.)

An extension should never call `PyType_ClearWatcher` with a *watcher_id* that was not returned to it by a previous call to `PyType_AddWatcher()`.

Added in version 3.12.

int **PyType_Watch**(int watcher_id, *PyObject* *type)

Mark *type* as watched. The callback granted *watcher_id* by `PyType_AddWatcher()` will be called whenever `PyType_Modified()` reports a change to *type*. (The callback may be called only once for a series of consecutive modifications to *type*, if `_PyType_Lookup()` is not called on *type* between the modifications; this is an implementation detail and subject to change.)

An extension should never call `PyType_Watch` with a *watcher_id* that was not returned to it by a previous call to `PyType_AddWatcher()`.

Added in version 3.12.

typedef int (***PyType_WatchCallback**)(*PyObject* *type)

Type of a type-watcher callback function.

The callback must not modify *type* or cause `PyType_Modified()` to be called on *type* or any type in its MRO; violating this rule could cause infinite recursion.

Added in version 3.12.

int **PyType_HasFeature**(*PyTypeObject* *o, int feature)

형 객체 *o*가 기능 *feature*를 설정하면 0이 아닌 값을 반환합니다. 형 기능은 단일 비트 플래그로 표시됩니다.

int **PyType_IS_GC**(*PyTypeObject* *o)

Return true if the type object includes support for the cycle detector; this tests the type flag `Py_TPFLAGS_HAVE_GC`.

int **PyType_IsSubtype**(*PyTypeObject* *a, *PyTypeObject* *b)

Part of the Stable ABI. *a*가 *b*의 서브 형이면 참을 반환합니다.

This function only checks for actual subtypes, which means that `__subclasscheck__()` is not called on *b*. Call `PyObject_IsSubclass()` to do the same check that `issubclass()` would do.

PyObject ***PyType_GenericAlloc**(*PyTypeObject* *type, *Py_ssize_t* nitems)

Return value: New reference. Part of the Stable ABI. 형 객체의 `tp_alloc` 슬롯을 위한 일반 처리기. 파이썬의 기본 메모리 할당 메커니즘을 사용하여 새 인스턴스를 할당하고 모든 내용을 `NULL`로 초기화합니다.

PyObject ***PyType_GenericNew**(*PyTypeObject* *type, *PyObject* *args, *PyObject* *kwds)

Return value: New reference. Part of the Stable ABI. 형 객체의 `tp_new` 슬롯을 위한 일반 처리기. 형의 `tp_alloc` 슬롯을 사용하여 새 인스턴스를 만듭니다.

int **PyType_Ready**(*PyTypeObject* *type)

Part of the Stable ABI. 형 개체를 마무리합니다. 초기화를 완료하려면 모든 형 객체에 대해 이 메서드를 호출해야 합니다. 이 함수는 형의 베이스 클래스에서 상속된 슬롯을 추가합니다. 성공 시 `0`을 반환하고, 오류 시 `-1`을 반환하고 예외를 설정합니다.

> **참고**
>
> If some of the base classes implements the GC protocol and the provided type does not include the `Py_TPFLAGS_HAVE_GC` in its flags, then the GC protocol will be automatically implemented from its parents. On the contrary, if the type being created does include `Py_TPFLAGS_HAVE_GC` in its flags then it **must** implement the GC protocol itself by at least implementing the `tp_traverse` handle.

PyObject ***PyType_GetName**(*PyTypeObject* *type)

Return value: New reference. Part of the Stable ABI *since version 3.11.* Return the type' s name. Equivalent to getting the type' s `__name__` attribute.

Added in version 3.11.

PyObject ***PyType_GetQualName**(*PyTypeObject* *type)

Return value: New reference. Part of the Stable ABI *since version 3.11.* Return the type' s qualified name. Equivalent to getting the type' s `__qualname__` attribute.

Added in version 3.11.

void ***PyType_GetSlot**(*PyTypeObject* *type, int slot)

Part of the Stable ABI *since version 3.4.* 지정된 슬롯에 저장된 함수 포인터를 반환합니다. 결과가 `NULL`이면, 슬롯이 `NULL`이거나 함수가 유효하지 않은 매개 변수로 호출되었음을 나타냅니다. 호출자는 일반적으로 결과 포인터를 적절한 함수 형으로 캐스팅합니다.

slot 인자의 가능한 값은 `PyType_Slot.slot`을 참조하십시오.

Added in version 3.4.

버전 3.10에서 변경: `PyType_GetSlot()` can now accept all types. Previously, it was limited to *heap types*.

PyObject ***PyType_GetModule**(*PyTypeObject* *type)

Part of the Stable ABI *since version 3.10.* `PyType_FromModuleAndSpec()`를 사용하여 형을 만들 때 지정된 형과 관련된 모듈 객체를 반환합니다.

주어진 형과 연관된 모듈이 없으면, `TypeError`를 설정하고 `NULL`을 반환합니다.

This function is usually used to get the module in which a method is defined. Note that in such a method, `PyType_GetModule(Py_TYPE(self))` may not return the intended result. `Py_TYPE(self)` may be a *subclass* of the intended class, and subclasses are not necessarily defined in the same module as their superclass. See `PyCMethod` to get the class that defines the method. See `PyType_GetModuleByDef()` for cases when `PyCMethod` cannot be used.

Added in version 3.9.

void *PyType_GetModuleState(*PyTypeObject* *type)

Part of the Stable ABI *since version 3.10.* 주어진 형과 관련된 모듈 객체의 상태를 반환합니다. `PyType_GetModule()` 결과에 `PyModule_GetState()`를 호출하는 바로 가기입니다.

주어진 형과 연관된 모듈이 없으면, `TypeError`를 설정하고 `NULL`을 반환합니다.

*type*에 연관된 모듈이 있지만, 상태가 `NULL`이면, 예외를 설정하지 않고 `NULL`을 반환합니다.

Added in version 3.9.

PyObject *PyType_GetModuleByDef(*PyTypeObject* *type, struct *PyModuleDef* *def)

Find the first superclass whose module was created from the given `PyModuleDef` *def*, and return that module.

If no module is found, raises a `TypeError` and returns `NULL`.

This function is intended to be used together with `PyModule_GetState()` to get module state from slot methods (such as `tp_init` or `nb_add`) and other places where a method' s defining class cannot be passed using the `PyCMethod` calling convention.

Added in version 3.11.

int PyUnstable_Type_AssignVersionTag(*PyTypeObject* *type)

This is *Unstable API*. It may change without warning in minor releases.

Attempt to assign a version tag to the given type.

Returns 1 if the type already had a valid version tag or a new one was assigned, or 0 if a new tag could not be assigned.

Added in version 3.12.

힙에 할당된 형 만들기

다음 함수와 구조체는 힙 형을 만드는 데 사용됩니다.

PyObject *PyType_FromMetaclass(*PyTypeObject* *metaclass, *PyObject* *module, *PyType_Spec* *spec, *PyObject* *bases)

Part of the Stable ABI *since version 3.12.* Create and return a *heap type* from the *spec* (see `Py_TPFLAGS_HEAPTYPE`).

The metaclass *metaclass* is used to construct the resulting type object. When *metaclass* is `NULL`, the metaclass is derived from *bases* (or *Py_tp_base*[*s*] slots if *bases* is `NULL`, see below).

Metaclasses that override `tp_new` are not supported, except if `tp_new` is `NULL`. (For backwards compatibility, other `PyType_From*` functions allow such metaclasses. They ignore `tp_new`, which may result in incomplete initialization. This is deprecated and in Python 3.14+ such metaclasses will not be supported.)

The *bases* argument can be used to specify base classes; it can either be only one class or a tuple of classes. If *bases* is `NULL`, the *Py_tp_bases* slot is used instead. If that also is `NULL`, the *Py_tp_base* slot is used instead. If that also is `NULL`, the new type derives from `object`.

The *module* argument can be used to record the module in which the new class is defined. It must be a module object or `NULL`. If not `NULL`, the module is associated with the new type and can later be retrieved with `PyType_GetModule()`. The associated module is not inherited by subclasses; it must be specified for each class individually.

이 함수는 새로운 형에 `PyType_Ready()`를 호출합니다.

Note that this function does *not* fully match the behavior of calling `type()` or using the `class` statement. With user-provided base types or metaclasses, prefer *calling* `type` (or the metaclass) over `PyType_From*` functions. Specifically:

- `__new__()` is not called on the new class (and it must be set to `type.__new__`).
- `__init__()` is not called on the new class.
- `__init_subclass__()` is not called on any bases.
- `__set_name__()` is not called on new descriptors.

Added in version 3.12.

PyObject ***PyType_FromModuleAndSpec**(*PyObject* *module, *PyType_Spec* *spec, *PyObject* *bases)

Return value: New reference. Part of the Stable ABI *since version 3.10.* Equivalent to `PyType_FromMetaclass(NULL, module, spec, bases)`.

Added in version 3.9.

버전 3.10에서 변경: The function now accepts a single class as the *bases* argument and `NULL` as the `tp_doc` slot.

버전 3.12에서 변경: The function now finds and uses a metaclass corresponding to the provided base classes. Previously, only `type` instances were returned.

The *tp_new* of the metaclass is *ignored*. which may result in incomplete initialization. Creating classes whose metaclass overrides *tp_new* is deprecated and in Python 3.14+ it will be no longer allowed.

PyObject ***PyType_FromSpecWithBases**(*PyType_Spec* *spec, *PyObject* *bases)

Return value: New reference. Part of the Stable ABI *since version 3.3.* Equivalent to `PyType_FromMetaclass(NULL, NULL, spec, bases)`.

Added in version 3.3.

버전 3.12에서 변경: The function now finds and uses a metaclass corresponding to the provided base classes. Previously, only `type` instances were returned.

The *tp_new* of the metaclass is *ignored*. which may result in incomplete initialization. Creating classes whose metaclass overrides *tp_new* is deprecated and in Python 3.14+ it will be no longer allowed.

PyObject ***PyType_FromSpec**(*PyType_Spec* *spec)

Return value: New reference. Part of the Stable ABI. Equivalent to `PyType_FromMetaclass(NULL, NULL, spec, NULL)`.

버전 3.12에서 변경: The function now finds and uses a metaclass corresponding to the base classes provided in *Py_tp_base*[*s*] slots. Previously, only `type` instances were returned.

The *tp_new* of the metaclass is *ignored*. which may result in incomplete initialization. Creating classes whose metaclass overrides *tp_new* is deprecated and in Python 3.14+ it will be no longer allowed.

type **PyType_Spec**

Part of the Stable ABI (*including all members*). 형의 행동을 정의하는 구조체.

const char ***name**

형의 이름, *PyTypeObject.tp_name*을 설정하는 데 사용됩니다.

int **basicsize**

If positive, specifies the size of the instance in bytes. It is used to set *PyTypeObject.tp_basicsize*.

If zero, specifies that *tp_basicsize* should be inherited.

If negative, the absolute value specifies how much space instances of the class need *in addition* to the superclass. Use *PyObject_GetTypeData()* to get a pointer to subclass-specific memory reserved this way. For negative `basicsize`, Python will insert padding when needed to meet *tp_basicsize*' s alignment requirements.

버전 3.12에서 변경: Previously, this field could not be negative.

int **itemsize**

Size of one element of a variable-size type, in bytes. Used to set *PyTypeObject.tp_itemsize*. See `tp_itemsize` documentation for caveats.

If zero, *tp_itemsize* is inherited. Extending arbitrary variable-sized classes is dangerous, since some types use a fixed offset for variable-sized memory, which can then overlap fixed-sized memory used by a subclass. To help prevent mistakes, inheriting `itemsize` is only possible in the following situations:

- The base is not variable-sized (its *tp_itemsize*).
- The requested *PyType_Spec.basicsize* is positive, suggesting that the memory layout of the base class is known.
- The requested *PyType_Spec.basicsize* is zero, suggesting that the subclass does not access the instance' s memory directly.
- With the *Py_TPFLAGS_ITEMS_AT_END* flag.

unsigned int **flags**

형 플래그, *PyTypeObject.tp_flags*를 설정하는 데 사용됩니다.

`Py_TPFLAGS_HEAPTYPE` 플래그가 설정되어 있지 않으면, *PyType_FromSpecWithBases()* 가 자동으로 플래그를 설정합니다.

PyType_Slot ***slots**

PyType_Slot 구조체의 배열. 특수 슬롯값 `{0, NULL}`에 의해 종료됩니다.

Each slot ID should be specified at most once.

type **PyType_Slot**

Part of the Stable ABI (*including all members*). 형의 선택적 기능을 정의하는 구조체, 슬롯 ID와 값 포인터를 포함합니다.

int **slot**

슬롯 ID.

슬롯 ID는 구조체 *PyTypeObject*, *PyNumberMethods*, *PySequenceMethods*, *PyMappingMethods* 및 *PyAsyncMethods* 의 필드 이름에 `Py_` 접두사를 붙인 이름을 사용합니다. 예를 들어, :

- *PyTypeObject.tp_dealloc*을 설정하는 `Py_tp_dealloc`
- *PyNumberMethods.nb_add*를 설정하는 `Py_nb_add`
- *PySequenceMethods.sq_length*를 설정하는 `Py_sq_length`

The following "offset" fields cannot be set using *PyType_Slot*:

- *tp_weaklistoffset* (use *Py_TPFLAGS_MANAGED_WEAKREF* instead if possible)
- *tp_dictoffset* (use *Py_TPFLAGS_MANAGED_DICT* instead if possible)
- *tp_vectorcall_offset* (use `"__vectorcalloffset__"` in *PyMemberDef*)

If it is not possible to switch to a `MANAGED` flag (for example, for vectorcall or to support Python older than 3.12), specify the offset in *Py_tp_members*. See *PyMemberDef documentation* for details.

The following fields cannot be set at all when creating a heap type:

- *tp_vectorcall* (use *tp_new* and/or *tp_init*)
- Internal fields: *tp_dict*, *tp_mro*, *tp_cache*, *tp_subclasses*, and *tp_weaklist*.

Setting `Py_tp_bases` or `Py_tp_base` may be problematic on some platforms. To avoid issues, use the *bases* argument of *PyType_FromSpecWithBases()* instead.

버전 3.9에서 변경: Slots in *PyBufferProcs* may be set in the unlimited API.

버전 3.11에서 변경: *bf_getbuffer* and *bf_releasebuffer* are now available under the *limited API*.

void ***pfunc**

슬롯의 원하는 값입니다. 대부분 이것은 함수에 대한 포인터입니다.

Slots other than `Py_tp_doc` may not be `NULL`.

8.1.2 `None` 객체

Note that the *`PyTypeObject`* for `None` is not directly exposed in the Python/C API. Since `None` is a singleton, testing for object identity (using == in C) is sufficient. There is no `PyNone_Check()` function for the same reason.

PyObject ***Py_None**

The Python `None` object, denoting lack of value. This object has no methods and is immortal.

버전 3.12에서 변경: *`Py_None`* is immortal.

Py_RETURN_NONE

Return *`Py_None`* from a function.

8.2 숫자 객체

8.2.1 정수 객체

모든 정수는 임의의 크기의 "long" 정수 객체로 구현됩니다.

에러 시, 대부분의 `PyLong_As*` API는 숫자와 구별할 수 없는 `(return type)-1`을 반환합니다. 모호성을 제거하려면 *`PyErr_Occurred()`*를 사용하십시오.

type **PyLongObject**

Part of the Limited API (*as an opaque struct*). 이 *`PyObject`*의 서브 형은 파이썬 정수 객체를 나타냅니다.

PyTypeObject **PyLong_Type**

Part of the Stable ABI. 이 *`PyTypeObject`* 인스턴스는 파이썬 정수 형을 나타냅니다. 이것은 파이썬 계층의 `int`와 같은 객체입니다.

int **PyLong_Check**(*PyObject* *p)

인자가 *`PyLongObject`*이나 *`PyLongObject`*의 서브 형이면 참을 반환합니다. 이 함수는 항상 성공합니다.

int **PyLong_CheckExact**(*PyObject* *p)

인자가 *`PyLongObject`* 이지만 *`PyLongObject`*의 서브 형이 아니면 참을 반환합니다. 이 함수는 항상 성공합니다.

PyObject ***PyLong_FromLong**(long v)

Return value: New reference. Part of the Stable ABI. *v*로부터 새 *`PyLongObject`* 객체를 반환하거나, 실패하면 `NULL`을 반환합니다.

The current implementation keeps an array of integer objects for all integers between `-5` and `256`. When you create an int in that range you actually just get back a reference to the existing object.

PyObject ***PyLong_FromUnsignedLong**(unsigned long v)

Return value: New reference. Part of the Stable ABI. Return a new *`PyLongObject`* object from a C `unsigned long`, or `NULL` on failure.

PyObject ***PyLong_FromSsize_t**(*Py_ssize_t* v)

Return value: New reference. Part of the Stable ABI. C *`Py_ssize_t`*로부터 새 *`PyLongObject`* 객체를 반환하거나, 실패하면 `NULL`을 반환합니다.

PyObject ***PyLong_FromSize_t**(size_t v)

Return value: New reference. Part of the Stable ABI. C `size_t`로부터 새 *`PyLongObject`* 객체를 반환하거나, 실패하면 `NULL`을 반환합니다.

PyObject ***PyLong_FromLongLong**(long long v)

Return value: New reference. Part of the Stable ABI. Return a new *PyLongObject* object from a C `long long`, or `NULL` on failure.

PyObject ***PyLong_FromUnsignedLongLong**(unsigned long long v)

Return value: New reference. Part of the Stable ABI. Return a new *PyLongObject* object from a C `unsigned long long`, or `NULL` on failure.

PyObject ***PyLong_FromDouble**(double v)

Return value: New reference. Part of the Stable ABI. *v*의 정수 부분으로부터 새 *PyLongObject* 객체를 반환하거나, 실패하면 `NULL`을 반환합니다.

PyObject ***PyLong_FromString**(const char *str, char **pend, int base)

Return value: New reference. Part of the Stable ABI. Return a new *PyLongObject* based on the string value in *str*, which is interpreted according to the radix in *base*, or `NULL` on failure. If *pend* is non-`NULL`, **pend* will point to the end of *str* on success or to the first character that could not be processed on error. If *base* is `0`, *str* is interpreted using the integers definition; in this case, leading zeros in a non-zero decimal number raises a `ValueError`. If *base* is not `0`, it must be between `2` and `36`, inclusive. Leading and trailing whitespace and single underscores after a base specifier and between digits are ignored. If there are no digits or *str* is not NULL-terminated following the digits and trailing whitespace, `ValueError` will be raised.

> **더 보기**
>
> Python methods `int.to_bytes()` and `int.from_bytes()` to convert a *PyLongObject* to/from an array of bytes in base `256`. You can call those from C using *PyObject_CallMethod()*.

PyObject ***PyLong_FromUnicodeObject**(*PyObject* *u, int base)

Return value: New reference. 문자열 *u*에 있는 유니코드 숫자의 시퀀스를 파이썬 정숫값으로 변환합니다.

Added in version 3.3.

PyObject ***PyLong_FromVoidPtr**(void *p)

Return value: New reference. Part of the Stable ABI. 포인터 *p*로부터 파이썬 정수를 만듭니다. 포인터 값은 *PyLong_AsVoidPtr()*를 사용하여 결괏값에서 조회할 수 있습니다.

long **PyLong_AsLong**(*PyObject* *obj)

Part of the Stable ABI. Return a C `long` representation of *obj*. If *obj* is not an instance of *PyLongObject*, first call its `__index__()` method (if present) to convert it to a *PyLongObject*.

Raise `OverflowError` if the value of *obj* is out of range for a `long`.

에러 시 `-1`을 반환합니다. 모호성을 제거하려면 *PyErr_Occurred()*를 사용하십시오.

버전 3.8에서 변경: Use `__index__()` if available.

버전 3.10에서 변경: This function will no longer use `__int__()`.

long **PyLong_AS_LONG**(*PyObject* *obj)

A *soft deprecated* alias. Exactly equivalent to the preferred `PyLong_AsLong`. In particular, it can fail with `OverflowError` or another exception.

버전 3.14부터 폐지됨: The function is soft deprecated.

long **PyLong_AsLongAndOverflow**(*PyObject* *obj, int *overflow)

Part of the Stable ABI. Return a C `long` representation of *obj*. If *obj* is not an instance of *PyLongObject*, first call its `__index__()` method (if present) to convert it to a *PyLongObject*.

If the value of *obj* is greater than `LONG_MAX` or less than `LONG_MIN`, set **overflow* to `1` or `-1`, respectively, and return `-1`; otherwise, set **overflow* to `0`. If any other exception occurs set **overflow* to `0` and return `-1` as usual.

에러 시 `-1`을 반환합니다. 모호성을 제거하려면 *PyErr_Occurred()*를 사용하십시오.

버전 3.8에서 변경: Use `__index__()` if available.

버전 3.10에서 변경: This function will no longer use `__int__()`.

long long **PyLong_AsLongLong**(*PyObject* *obj)

Part of the Stable ABI. Return a C `long long` representation of *obj*. If *obj* is not an instance of *PyLongObject*, first call its `__index__()` method (if present) to convert it to a *PyLongObject*.

Raise `OverflowError` if the value of *obj* is out of range for a `long long`.

에러 시 `-1`을 반환합니다. 모호성을 제거하려면 *PyErr_Occurred()*를 사용하십시오.

버전 3.8에서 변경: Use `__index__()` if available.

버전 3.10에서 변경: This function will no longer use `__int__()`.

long long **PyLong_AsLongLongAndOverflow**(*PyObject* *obj, int *overflow)

Part of the Stable ABI. Return a C `long long` representation of *obj*. If *obj* is not an instance of *PyLongObject*, first call its `__index__()` method (if present) to convert it to a *PyLongObject*.

If the value of *obj* is greater than `LLONG_MAX` or less than `LLONG_MIN`, set **overflow* to `1` or `-1`, respectively, and return `-1`; otherwise, set **overflow* to `0`. If any other exception occurs set **overflow* to `0` and return `-1` as usual.

에러 시 `-1`을 반환합니다. 모호성을 제거하려면 *PyErr_Occurred()*를 사용하십시오.

Added in version 3.2.

버전 3.8에서 변경: Use `__index__()` if available.

버전 3.10에서 변경: This function will no longer use `__int__()`.

Py_ssize_t **PyLong_AsSsize_t**(*PyObject* *pylong)

Part of the Stable ABI. *pylong*의 C *Py_ssize_t* 표현을 반환합니다. *pylong*은 *PyLongObject*의 인스턴스여야 합니다.

*pylong*의 값이 *Py_ssize_t*의 범위를 벗어나면 `OverflowError`를 발생시킵니다.

에러 시 `-1`을 반환합니다. 모호성을 제거하려면 *PyErr_Occurred()*를 사용하십시오.

unsigned long **PyLong_AsUnsignedLong**(*PyObject* *pylong)

Part of the Stable ABI. Return a C `unsigned long` representation of *pylong*. *pylong* must be an instance of *PyLongObject*.

Raise `OverflowError` if the value of *pylong* is out of range for a `unsigned long`.

에러 시 `(unsigned long)-1`을 반환합니다. 모호성을 제거하려면 *PyErr_Occurred()*를 사용하십시오.

size_t **PyLong_AsSize_t**(*PyObject* *pylong)

Part of the Stable ABI. *pylong*의 C `size_t` 표현을 반환합니다. *pylong*은 *PyLongObject*의 인스턴스여야 합니다.

*pylong*의 값이 `size_t`의 범위를 벗어나면 `OverflowError`를 발생시킵니다.

에러 시 `(size_t)-1`을 반환합니다. 모호성을 제거하려면 *PyErr_Occurred()*를 사용하십시오.

unsigned long long **PyLong_AsUnsignedLongLong**(*PyObject* *pylong)

Part of the Stable ABI. Return a C `unsigned long long` representation of *pylong*. *pylong* must be an instance of *PyLongObject*.

Raise `OverflowError` if the value of *pylong* is out of range for an `unsigned long long`.

에러 시 `(unsigned long long)-1`을 반환합니다. 모호성을 제거하려면 *PyErr_Occurred()*를 사용하십시오.

버전 3.1에서 변경: 음의 *pylong*는 이제 `TypeError`가 아니라 `OverflowError`를 발생시킵니다.

unsigned long **PyLong_AsUnsignedLongMask**(*PyObject* *obj)

Part of the Stable ABI. Return a C `unsigned long` representation of *obj*. If *obj* is not an instance of *PyLongObject*, first call its `__index__()` method (if present) to convert it to a *PyLongObject*.

If the value of *obj* is out of range for an `unsigned long`, return the reduction of that value modulo `ULONG_MAX + 1`.

에러 시 `(unsigned long)-1`을 반환합니다. 모호성을 제거하려면 *PyErr_Occurred()*를 사용하십시오.

버전 3.8에서 변경: Use `__index__()` if available.

버전 3.10에서 변경: This function will no longer use `__int__()`.

unsigned long long **PyLong_AsUnsignedLongLongMask**(*PyObject* *obj)

Part of the Stable ABI. Return a C `unsigned long long` representation of *obj*. If *obj* is not an instance of *PyLongObject*, first call its `__index__()` method (if present) to convert it to a *PyLongObject*.

If the value of *obj* is out of range for an `unsigned long long`, return the reduction of that value modulo `ULLONG_MAX + 1`.

에러 시 `(unsigned long long)-1`을 반환합니다. 모호성을 제거하려면 *PyErr_Occurred()*를 사용하십시오.

버전 3.8에서 변경: Use `__index__()` if available.

버전 3.10에서 변경: This function will no longer use `__int__()`.

double **PyLong_AsDouble**(*PyObject* *pylong)

Part of the Stable ABI. Return a C `double` representation of *pylong*. *pylong* must be an instance of *PyLongObject*.

Raise `OverflowError` if the value of *pylong* is out of range for a `double`.

에러 시 `-1.0`을 반환합니다. 모호성을 제거하려면 *PyErr_Occurred()*를 사용하십시오.

void ***PyLong_AsVoidPtr**(*PyObject* *pylong)

Part of the Stable ABI. Convert a Python integer *pylong* to a C `void` pointer. If *pylong* cannot be converted, an `OverflowError` will be raised. This is only assured to produce a usable `void` pointer for values created with *PyLong_FromVoidPtr()*.

에러 시 `NULL`을 반환합니다. 모호성을 제거하려면 *PyErr_Occurred()*를 사용하십시오.

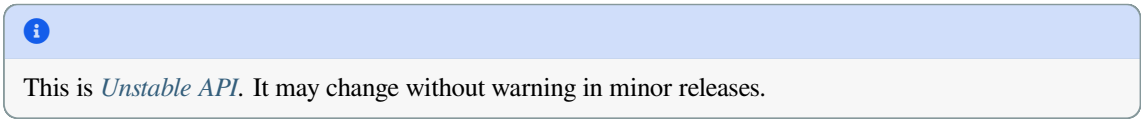
PyObject ***PyLong_GetInfo**(void)

Part of the Stable ABI. On success, return a read only *named tuple*, that holds information about Python' s internal representation of integers. See `sys.int_info` for description of individual fields.

On failure, return `NULL` with an exception set.

Added in version 3.1.

int **PyUnstable_Long_IsCompact**(const *PyLongObject* *op)



This is *Unstable API*. It may change without warning in minor releases.

Return 1 if *op* is compact, 0 otherwise.

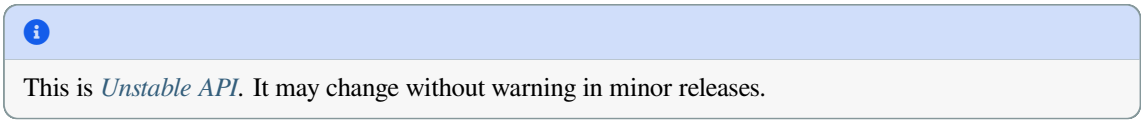
This function makes it possible for performance-critical code to implement a "fast path" for small integers. For compact values use *PyUnstable_Long_CompactValue()*; for others fall back to a *PyLong_As** function or *calling* `int.to_bytes()`.

The speedup is expected to be negligible for most users.

Exactly what values are considered compact is an implementation detail and is subject to change.

Added in version 3.12.

Py_ssize_t **PyUnstable_Long_CompactValue**(const *PyLongObject* *op)



This is *Unstable API*. It may change without warning in minor releases.

If *op* is compact, as determined by `PyUnstable_Long_IsCompact()`, return its value.

Otherwise, the return value is undefined.

Added in version 3.12.

8.2.2 불리언 객체

Booleans in Python are implemented as a subclass of integers. There are only two booleans, `Py_False` and `Py_True`. As such, the normal creation and deletion functions don' t apply to booleans. The following macros are available, however.

PyTypeObject **PyBool_Type**

Part of the Stable ABI. This instance of `PyTypeObject` represents the Python boolean type; it is the same object as `bool` in the Python layer.

int **PyBool_Check**(*PyObject* *o)

*o*가 `PyBool_Type` 형이면 참을 돌려줍니다. 이 함수는 항상 성공합니다.

PyObject ***Py_False**

The Python `False` object. This object has no methods and is immortal.

버전 3.12에서 변경: `Py_False` is immortal.

PyObject ***Py_True**

The Python `True` object. This object has no methods and is immortal.

버전 3.12에서 변경: `Py_True` is immortal.

Py_RETURN_FALSE

Return `Py_False` from a function.

Py_RETURN_TRUE

Return `Py_True` from a function.

PyObject ***PyBool_FromLong**(long v)

Return value: New reference. Part of the Stable ABI. Return `Py_True` or `Py_False`, depending on the truth value of *v*.

8.2.3 Floating-Point Objects

type **PyFloatObject**

This subtype of `PyObject` represents a Python floating-point object.

PyTypeObject **PyFloat_Type**

Part of the Stable ABI. This instance of `PyTypeObject` represents the Python floating-point type. This is the same object as `float` in the Python layer.

int **PyFloat_Check**(*PyObject* *p)

인자가 `PyFloatObject` 나 `PyFloatObject`의 서브 형이면 참을 반환합니다. 이 함수는 항상 성공합니다.

int **PyFloat_CheckExact**(*PyObject* *p)

인자가 *PyFloatObject*이지만 *PyFloatObject*의 서브 형은 아니면 참을 반환합니다. 이 함수는 항상 성공합니다.

PyObject ***PyFloat_FromString**(*PyObject* *str)

Return value: New reference. Part of the Stable ABI. *str*의 문자열 값을 기반으로 *PyFloatObject* 객체를 만들거나, 실패하면 NULL.

PyObject ***PyFloat_FromDouble**(double v)

Return value: New reference. Part of the Stable ABI. *v*로부터 *PyFloatObject* 객체를 만들거나, 실패하면 NULL.

double **PyFloat_AsDouble**(*PyObject* *pyfloat)

Part of the Stable ABI. Return a C double representation of the contents of *pyfloat*. If *pyfloat* is not a Python floating-point object but has a __float__() method, this method will first be called to convert *pyfloat* into a float. If __float__() is not defined then it falls back to __index__(). This method returns -1.0 upon failure, so one should call *PyErr_Occurred()* to check for errors.

버전 3.8에서 변경: Use __index__() if available.

double **PyFloat_AS_DOUBLE**(*PyObject* *pyfloat)

Return a C double representation of the contents of *pyfloat*, but without error checking.

PyObject ***PyFloat_GetInfo**(void)

Return value: New reference. Part of the Stable ABI. float의 정밀도, 최솟값, 최댓값에 관한 정보를 포함한 structseq 인스턴스를 돌려줍니다. 헤더 파일 float.h를 감싸는 얇은 래퍼입니다.

double **PyFloat_GetMax**()

Part of the Stable ABI. Return the maximum representable finite float *DBL_MAX* as C double.

double **PyFloat_GetMin**()

Part of the Stable ABI. Return the minimum normalized positive float *DBL_MIN* as C double.

Pack and Unpack functions

The pack and unpack functions provide an efficient platform-independent way to store floating-point values as byte strings. The Pack routines produce a bytes string from a C double, and the Unpack routines produce a C double from such a bytes string. The suffix (2, 4 or 8) specifies the number of bytes in the bytes string.

On platforms that appear to use IEEE 754 formats these functions work by copying bits. On other platforms, the 2-byte format is identical to the IEEE 754 binary16 half-precision format, the 4-byte format (32-bit) is identical to the IEEE 754 binary32 single precision format, and the 8-byte format to the IEEE 754 binary64 double precision format, although the packing of INFs and NaNs (if such things exist on the platform) isn' t handled correctly, and attempting to unpack a bytes string containing an IEEE INF or NaN will raise an exception.

On non-IEEE platforms with more precision, or larger dynamic range, than IEEE 754 supports, not all values can be packed; on non-IEEE platforms with less precision, or smaller dynamic range, not all values can be unpacked. What happens in such cases is partly accidental (alas).

Added in version 3.11.

Pack functions

The pack routines write 2, 4 or 8 bytes, starting at *p*. *le* is an int argument, non-zero if you want the bytes string in little-endian format (exponent last, at p+1, p+3, or p+6 p+7), zero if you want big-endian format (exponent first, at *p*). The PY_BIG_ENDIAN constant can be used to use the native endian: it is equal to 1 on big endian processor, or 0 on little endian processor.

Return value: 0 if all is OK, -1 if error (and an exception is set, most likely OverflowError).

There are two problems on non-IEEE platforms:

- What this does is undefined if *x* is a NaN or infinity.

- `-0.0` and `+0.0` produce the same bytes string.

int **PyFloat_Pack2**(double x, unsigned char *p, int le)

Pack a C double as the IEEE 754 binary16 half-precision format.

int **PyFloat_Pack4**(double x, unsigned char *p, int le)

Pack a C double as the IEEE 754 binary32 single precision format.

int **PyFloat_Pack8**(double x, unsigned char *p, int le)

Pack a C double as the IEEE 754 binary64 double precision format.

Unpack functions

The unpack routines read 2, 4 or 8 bytes, starting at *p*. *le* is an `int` argument, non-zero if the bytes string is in little-endian format (exponent last, at `p+1`, `p+3` or `p+6` and `p+7`), zero if big-endian (exponent first, at *p*). The `PY_BIG_ENDIAN` constant can be used to use the native endian: it is equal to `1` on big endian processor, or `0` on little endian processor.

Return value: The unpacked double. On error, this is `-1.0` and *PyErr_Occurred()* is true (and an exception is set, most likely `OverflowError`).

Note that on a non-IEEE platform this will refuse to unpack a bytes string that represents a NaN or infinity.

double **PyFloat_Unpack2**(const unsigned char *p, int le)

Unpack the IEEE 754 binary16 half-precision format as a C double.

double **PyFloat_Unpack4**(const unsigned char *p, int le)

Unpack the IEEE 754 binary32 single precision format as a C double.

double **PyFloat_Unpack8**(const unsigned char *p, int le)

Unpack the IEEE 754 binary64 double precision format as a C double.

8.2.4 복소수 객체

파이썬의 복소수 객체는 C API에서 볼 때 두 개의 다른 형으로 구현됩니다: 하나는 파이썬 프로그램에 노출된 파이썬 객체이고, 다른 하나는 실제 복소수 값을 나타내는 C 구조체입니다. API는 두 가지 모두도 작업할 수 있는 함수를 제공합니다.

C 구조체로서의 복소수

매개 변수로 이러한 구조체를 받아들이고 결과로 반환하는 함수는 포인터를 통해 역참조하기보다는 값으로 다룹니다. 이는 API 전체에서 일관됩니다.

type **Py_complex**

The C structure which corresponds to the value portion of a Python complex number object. Most of the functions for dealing with complex number objects use structures of this type as input or output values, as appropriate.

double **real**

double **imag**

The structure is defined as:

```
typedef struct {
    double real;
    double imag;
} Py_complex;
```

Py_complex **_Py_c_sum**(*Py_complex* left, *Py_complex* right)

C *Py_complex* 표현을 사용하여 두 복소수의 합을 반환합니다.

Py_complex **_Py_c_diff**(*Py_complex* left, *Py_complex* right)

C *Py_complex* 표현을 사용하여 두 복소수의 차이를 반환합니다.

Py_complex **_Py_c_neg**(*Py_complex* num)

Return the negation of the complex number *num*, using the C *Py_complex* representation.

Py_complex **_Py_c_prod**(*Py_complex* left, *Py_complex* right)

C *Py_complex* 표현을 사용하여 두 복소수의 곱을 반환합니다.

Py_complex **_Py_c_quot**(*Py_complex* dividend, *Py_complex* divisor)

C *Py_complex* 표현을 사용하여 두 복소수의 몫을 반환합니다.

If *divisor* is null, this method returns zero and sets `errno` to `EDOM`.

Py_complex **_Py_c_pow**(*Py_complex* num, *Py_complex* exp)

C *Py_complex* 표현을 사용하여 *num*의 *exp* 거듭제곱을 반환합니다.

If *num* is null and *exp* is not a positive real number, this method returns zero and sets `errno` to `EDOM`.

파이썬 객체로서의 복소수

type **PyComplexObject**

파이썬 복소수 객체를 나타내는 *PyObject*의 서브 형.

PyTypeObject **PyComplex_Type**

Part of the Stable ABI. 이 *PyTypeObject* 인스턴스는 파이썬 복소수 형을 나타냅니다. 파이썬 계층의 `complex`와 같은 객체입니다.

int **PyComplex_Check**(*PyObject* *p)

인자가 *PyComplexObject* 나 *PyComplexObject*의 서브 형이면 참을 반환합니다. 이 함수는 항상 성공합니다.

int **PyComplex_CheckExact**(*PyObject* *p)

인자가 *PyComplexObject*이지만, *PyComplexObject*의 서브 유형이 아니면 참을 반환합니다. 이 함수는 항상 성공합니다.

PyObject ***PyComplex_FromCComplex**(*Py_complex* v)

Return value: New reference. Create a new Python complex number object from a C *Py_complex* value. Return `NULL` with an exception set on error.

PyObject ***PyComplex_FromDoubles**(double real, double imag)

Return value: New reference. Part of the Stable ABI. Return a new *PyComplexObject* object from *real* and *imag*. Return `NULL` with an exception set on error.

double **PyComplex_RealAsDouble**(*PyObject* *op)

Part of the Stable ABI. Return the real part of *op* as a C `double`.

Upon failure, this method returns `-1.0` with an exception set, so one should call *PyErr_Occurred()* to check for errors.

double **PyComplex_ImagAsDouble**(*PyObject* *op)

Part of the Stable ABI. Return the imaginary part of *op* as a C `double`.

Py_complex **PyComplex_AsCComplex**(*PyObject* *op)

복소수 *op*의 *Py_complex* 값을 반환합니다.

If *op* is not a Python complex number object but has a `__complex__()` method, this method will first be called to convert *op* to a Python complex number object. If `__complex__()` is not defined then it falls back to `__float__()`. If `__float__()` is not defined then it falls back to `__index__()`.

Upon failure, this method returns *Py_complex* with *real* set to `-1.0` and with an exception set, so one should call *PyErr_Occurred()* to check for errors.

버전 3.8에서 변경: Use `__index__()` if available.

8.3 시퀀스 객체

시퀀스 객체에 대한 일반적인 연산은 이전 장에서 논의했습니다; 이 절에서는 파이썬 언어에 고유한 특정 종류의 시퀀스 객체를 다룹니다.

8.3.1 바이트열 객체

These functions raise `TypeError` when expecting a bytes parameter and called with a non-bytes parameter.

type **PyBytesObject**

이 *PyObject*의 서브 형은 파이썬 바이트열 객체를 나타냅니다.

PyTypeObject **PyBytes_Type**

Part of the Stable ABI. 이 *PyTypeObject*의 인스턴스는 파이썬 바이트열 형을 나타냅니다; 파이썬 계층의 `bytes`와 같은 객체입니다.

int **PyBytes_Check**(*PyObject* *o)

객체 *o*가 바이트열 객체이거나 바이트열 형의 서브 형의 인스턴스면 참을 반환합니다. 이 함수는 항상 성공합니다.

int **PyBytes_CheckExact**(*PyObject* *o)

객체 *o*가 바이트열 객체이지만, 바이트열 형의 서브 형의 인스턴스는 아니면 참을 반환합니다. 이 함수는 항상 성공합니다.

PyObject ***PyBytes_FromString**(const char *v)

Return value: New reference. Part of the Stable ABI. 성공하면 값으로 *v* 문자열의 복사본을 갖는 새 바이트열 객체를 반환하고, 실패하면 `NULL`을 반환합니다. 매개 변수 *v*는 `NULL`이 아니어야 합니다; 검사하지 않습니다.

PyObject ***PyBytes_FromStringAndSize**(const char *v, *Py_ssize_t* len)

Return value: New reference. Part of the Stable ABI. 성공하면 값이 *v* 문자열의 복사본이고 길이가 *len*인 새 바이트열 객체를 반환하고, 실패하면 `NULL`을 반환합니다. *v*가 `NULL`이면, 바이트열 객체의 내용은 초기화되지 않습니다.

PyObject ***PyBytes_FromFormat**(const char *format, ...)

Return value: New reference. Part of the Stable ABI. C `printf()`-스타일 *format* 문자열과 가변 개수의 인자를 받아서, 결과 파이썬 바이트열 객체의 크기를 계산하고 그 안에 값이 포맷된 바이트열 객체를 반환합니다. 가변 인자는 C 형이어야 하며 *format* 문자열에 있는 포맷 문자들과 정확히 대응해야 합니다. 허용되는 포맷 문자는 다음과 같습니다:

포맷 문자	형	주석
`%%`	*n/a*	리터럴 % 문자.
`%c`	int	단일 바이트, C int로 표현됩니다.
`%d`	int	`printf("%d")`와 동등합니다.[1]
`%u`	unsigned int	`printf("%u")`와 동등합니다.[1]
`%ld`	long	`printf("%ld")`와 동등합니다.[1]
`%lu`	unsigned long	`printf("%lu")`와 동등합니다.[1]
`%zd`	*Py_ssize_t*	`printf("%zd")`와 동등합니다.[1]
`%zu`	size_t	`printf("%zu")`와 동등합니다.[1]
`%i`	int	`printf("%i")`와 동등합니다.[1]
`%x`	int	`printf("%x")`와 동등합니다.[1]
`%s`	const char*	널-종료 C 문자 배열.
`%p`	const void*	C 포인터의 16진수 표현. 플랫폼의 `printf`가 어떤 결과를 내는지에 상관없이 리터럴 `0x`로 시작함이 보장된다는 점을 제외하고는 거의 `printf("%p")`와 동등합니다.

[1] 정수 지정자 (d, u, ld, lu, zd, zu, i, x)에서: 0-변환 플래그는 정밀도를 지정해도 영향을 미칩니다.

인식할 수 없는 포맷 문자는 포맷 문자열의 나머지 부분이 모두 결과 객체에 그대로 복사되게 만들고, 추가 인자는 무시됩니다.

PyObject ***PyBytes_FromFormatV**(const char *format, va_list vargs)

Return value: New reference. Part of the Stable ABI. 정확히 두 개의 인자를 취한다는 것을 제외하고는 *PyBytes_FromFormat()*과 같습니다.

PyObject ***PyBytes_FromObject**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 버퍼 프로토콜을 구현하는 객체 *o*의 바이트열 표현을 반환합니다.

Py_ssize_t **PyBytes_Size**(*PyObject* *o)

Part of the Stable ABI. 바이트열 객체 *o*의 길이를 반환합니다.

Py_ssize_t **PyBytes_GET_SIZE**(*PyObject* *o)

Similar to *PyBytes_Size()*, but without error checking.

char ***PyBytes_AsString**(*PyObject* *o)

Part of the Stable ABI. *o*의 내용에 대한 포인터를 반환합니다. 포인터는 `len(o) + 1` 바이트로 구성된 *o*의 내부 버퍼를 가리킵니다. 버퍼의 마지막 바이트는 다른 널(null) 바이트가 있는지에 관계없이 항상 널입니다. 객체가 `PyBytes_FromStringAndSize(NULL, size)`를 사용하여 방금 만들어진 경우가 아니면 데이터를 수정해서는 안 됩니다. 할당을 해제해서는 안 됩니다. *o*가 바이트열 객체가 아니면, *PyBytes_AsString()*은 `NULL`을 반환하고 `TypeError`를 발생시킵니다.

char ***PyBytes_AS_STRING**(*PyObject* *string)

Similar to *PyBytes_AsString()*, but without error checking.

int **PyBytes_AsStringAndSize**(*PyObject* *obj, char **buffer, *Py_ssize_t* *length)

Part of the Stable ABI. Return the null-terminated contents of the object *obj* through the output variables *buffer* and *length*. Returns `0` on success.

*length*가 `NULL`이면, 바이트열 객체는 내장된 널 바이트를 포함할 수 없습니다; 만약 그렇다면 함수는 `-1`을 반환하고 `ValueError`를 발생시킵니다.

buffer는 *obj*의 내부 버퍼를 가리키게 되는데, 끝에 추가 널 바이트가 포함됩니다 (*length*에는 포함되지 않습니다). 객체가 `PyBytes_FromStringAndSize(NULL, size)`를 사용하여 방금 만들어진 경우가 아니면 데이터를 수정해서는 안 됩니다. 할당을 해제해서는 안 됩니다. *obj*가 바이트열 객체가 아니면 *PyBytes_AsStringAndSize()*는 `-1`을 반환하고 `TypeError`를 발생시킵니다.

버전 3.5에서 변경: 이전에는, 바이트열 객체에 널 바이트가 포함되어 있으면 `TypeError`가 발생했습니다.

void **PyBytes_Concat**(*PyObject* **bytes, *PyObject* *newpart)

Part of the Stable ABI. *bytes*에 *newpart*의 내용을 덧붙인 새 바이트열 객체를 **bytes*에 만듭니다; 호출자가 새 참조를 소유합니다. *bytes*의 예전 값에 대한 참조를 훔칩니다. 새 객체가 만들어질 수 없으면, *bytes*에 대한 예전 참조는 여전히 버려지고 **bytes*의 값은 `NULL`로 설정됩니다; 적절한 예외가 설정됩니다.

void **PyBytes_ConcatAndDel**(*PyObject* **bytes, *PyObject* *newpart)

Part of the Stable ABI. Create a new bytes object in **bytes* containing the contents of *newpart* appended to *bytes*. This version releases the *strong reference* to *newpart* (i.e. decrements its reference count).

int **_PyBytes_Resize**(*PyObject* **bytes, *Py_ssize_t* newsize)

"불변" 임에도 불구하고 바이트열 객체의 크기를 변경하는 방법. 완전히 새로운 바이트열 객체를 만들 때만 사용하십시오; bytes가 이미 코드의 다른 부분에 알려져 있을 수 있다면 사용하지 마십시오. 입력 바이트열 객체의 참조 횟수가 1이 아닐 때 이 함수를 호출하는 것은 에러입니다. 기존 바이트열 객체의 주소를 lvalue(내용을 기록할 수 있습니다)로 전달하고, 원하는 새 크기를 전달합니다. 성공하면, **bytes*는 크기가 변경된 바이트열 객체를 갖게 되고 `0`이 반환됩니다; **bytes*의 주소는 입력값과 다를 수 있습니다. 재할당이 실패하면, **bytes*에 있는 원래 바이트열 객체는 할당 해제되고, **bytes*가 `NULL`로 설정되고, `MemoryError`가 설정되며 `-1`이 반환됩니다.

8.3.2 바이트 배열 객체

type **PyByteArrayObject**

이 *PyObject*의 서브 형은 파이썬 bytearray 객체를 나타냅니다.

PyTypeObject **PyByteArray_Type**

Part of the Stable ABI. 이 *PyTypeObject* 인스턴스는 파이썬 bytearray 형을 나타냅니다; 파이썬 계층의 `bytearray`와 같은 객체입니다.

형 검사 매크로

int **PyByteArray_Check**(*PyObject* *o)

객체 *o*가 bytearray 객체이거나 bytearray 형의 서브 형 인스턴스면 참을 반환합니다. 이 함수는 항상 성공합니다.

int **PyByteArray_CheckExact**(*PyObject* *o)

객체 *o*가 bytearray 객체이지만, bytearray 형의 서브 형 인스턴스는 아니면 참을 반환합니다. 이 함수는 항상 성공합니다.

직접 API 함수

PyObject ***PyByteArray_FromObject**(*PyObject* *o)

Return value: New reference. Part of the Stable ABI. 버퍼 프로토콜을 구현하는 임의의 객체(*o*)로부터 써서 새로운 bytearray 객체를 돌려줍니다.

On failure, return `NULL` with an exception set.

PyObject ***PyByteArray_FromStringAndSize**(const char *string, *Py_ssize_t* len)

Return value: New reference. Part of the Stable ABI. Create a new bytearray object from *string* and its length, *len*.

On failure, return `NULL` with an exception set.

PyObject ***PyByteArray_Concat**(*PyObject* *a, *PyObject* *b)

Return value: New reference. Part of the Stable ABI. 바이트 배열 *a* 와 *b*를 이어붙여 새로운 bytearray로 반환합니다.

On failure, return `NULL` with an exception set.

Py_ssize_t **PyByteArray_Size**(*PyObject* *bytearray)

Part of the Stable ABI. `NULL` 포인터를 확인한 후 *bytearray*의 크기를 반환합니다.

char ***PyByteArray_AsString**(*PyObject* *bytearray)

Part of the Stable ABI. `NULL` 포인터를 확인한 후 *bytearray*의 내용을 char 배열로 반환합니다. 반환되는 배열에는 항상 여분의 널 바이트가 추가됩니다.

int **PyByteArray_Resize**(*PyObject* *bytearray, *Py_ssize_t* len)

Part of the Stable ABI. *bytearray*의 내부 버퍼의 크기를 *len*으로 조정합니다.

매크로

이 매크로는 속도를 위해 안전을 희생하며 포인터를 확인하지 않습니다.

char ***PyByteArray_AS_STRING**(*PyObject* *bytearray)

Similar to *PyByteArray_AsString()*, but without error checking.

Py_ssize_t **PyByteArray_GET_SIZE**(*PyObject* *bytearray)

Similar to *PyByteArray_Size()*, but without error checking.

8.3.3 유니코드 객체와 코덱

유니코드 객체

파이썬 3.3에서 **PEP 393**을 구현한 이후, 유니코드 객체는 내부적으로 다양한 표현을 사용하여 전체 유니코드 문자 범위를 처리하면서 메모리 효율성을 유지합니다. 모든 코드 포인트가 128, 256 또는 65536 미만인 문자열에 대한 특별한 경우가 있습니다; 그렇지 않으면, 코드 포인트는 1114112 (전체 유니코드 범위) 미만이어야 합니다.

UTF-8 representation is created on demand and cached in the Unicode object.

> **참고**
>
> The *Py_UNICODE* representation has been removed since Python 3.12 with deprecated APIs. See **PEP 623** for more information.

유니코드 형

다음은 파이썬에서 유니코드 구현에 사용되는 기본 유니코드 객체 형입니다:

type **Py_UCS4**

type **Py_UCS2**

type **Py_UCS1**

Part of the Stable ABI. 이 형들은 각각 32비트, 16비트 및 8비트의 문자를 포함하기에 충분한 부호 없는 정수 형을 위한 typedef 입니다. 단일 유니코드 문자를 처리할 때는, *Py_UCS4*를 사용하십시오.

Added in version 3.3.

type **Py_UNICODE**

이것은 플랫폼에 따라 16비트 형이나 32비트 형인 `wchar_t`의 typedef 입니다.

버전 3.3에서 변경: 이전 버전에서, 이것은 빌드 시 파이썬의 "내로우(narrow)" 나 "와이드(wide)" 유니코드 버전 중 어느 것을 선택했는지에 따라 16비트 형이나 32비트 형이었습니다.

type **PyASCIIObject**

type **PyCompactUnicodeObject**

type **PyUnicodeObject**

이 *PyObject* 서브 형들은 파이썬 유니코드 객체를 나타냅니다. 거의 모든 경우에, 유니코드 객체를 처리하는 모든 API 함수가 *PyObject* 포인터를 취하고 반환하므로 직접 사용해서는 안 됩니다.

Added in version 3.3.

PyTypeObject **PyUnicode_Type**

Part of the Stable ABI. 이 *PyTypeObject* 인스턴스는 파이썬 유니코드 형을 나타냅니다. 파이썬 코드에 `str`로 노출됩니다.

The following APIs are C macros and static inlined functions for fast checks and access to internal read-only data of Unicode objects:

int **PyUnicode_Check**(*PyObject* *obj)

Return true if the object *obj* is a Unicode object or an instance of a Unicode subtype. This function always succeeds.

int **PyUnicode_CheckExact**(*PyObject* *obj)

Return true if the object *obj* is a Unicode object, but not an instance of a subtype. This function always succeeds.

int **PyUnicode_READY**(*PyObject* *unicode)

Returns `0`. This API is kept only for backward compatibility.

Added in version 3.3.

버전 3.10부터 폐지됨: This API does nothing since Python 3.12.

Py_ssize_t **PyUnicode_GET_LENGTH**(*PyObject* *unicode)

Return the length of the Unicode string, in code points. *unicode* has to be a Unicode object in the "canonical" representation (not checked).

Added in version 3.3.

Py_UCS1 ***PyUnicode_1BYTE_DATA**(*PyObject* *unicode)

Py_UCS2 ***PyUnicode_2BYTE_DATA**(*PyObject* *unicode)

Py_UCS4 ***PyUnicode_4BYTE_DATA**(*PyObject* *unicode)

Return a pointer to the canonical representation cast to UCS1, UCS2 or UCS4 integer types for direct character access. No checks are performed if the canonical representation has the correct character size; use *PyUnicode_KIND()* to select the right function.

Added in version 3.3.

PyUnicode_1BYTE_KIND

PyUnicode_2BYTE_KIND

PyUnicode_4BYTE_KIND

PyUnicode_KIND() 매크로의 값을 반환합니다.

Added in version 3.3.

버전 3.12에서 변경: PyUnicode_WCHAR_KIND has been removed.

int **PyUnicode_KIND**(*PyObject* *unicode)

Return one of the PyUnicode kind constants (see above) that indicate how many bytes per character this Unicode object uses to store its data. *unicode* has to be a Unicode object in the "canonical" representation (not checked).

Added in version 3.3.

void ***PyUnicode_DATA**(*PyObject* *unicode)

Return a void pointer to the raw Unicode buffer. *unicode* has to be a Unicode object in the "canonical" representation (not checked).

Added in version 3.3.

void **PyUnicode_WRITE**(int kind, void *data, *Py_ssize_t* index, *Py_UCS4* value)

Write into a canonical representation *data* (as obtained with *PyUnicode_DATA()*). This function performs no sanity checks, and is intended for usage in loops. The caller should cache the *kind* value and *data* pointer as obtained from other calls. *index* is the index in the string (starts at 0) and *value* is the new code point value which should be written to that location.

Added in version 3.3.

Py_UCS4 **PyUnicode_READ**(int kind, void *data, *Py_ssize_t* index)

규범적(canonical) 표현 *data*(*PyUnicode_DATA()*로 얻은 대로)에서 코드 포인트를 읽습니다. 검사나 준비(ready) 호출이 수행되지 않습니다.

Added in version 3.3.

Py_UCS4 **PyUnicode_READ_CHAR**(*PyObject* *unicode, *Py_ssize_t* index)

Read a character from a Unicode object *unicode*, which must be in the "canonical" representation. This is less efficient than *PyUnicode_READ()* if you do multiple consecutive reads.

Added in version 3.3.

Py_UCS4 **PyUnicode_MAX_CHAR_VALUE**(*PyObject* *unicode)

Return the maximum code point that is suitable for creating another string based on *unicode*, which must be in the "canonical" representation. This is always an approximation but more efficient than iterating over the string.

Added in version 3.3.

int **PyUnicode_IsIdentifier**(*PyObject* *unicode)

Part of the Stable ABI. 언어 정의에 따라 문자열이 유효한 식별자이면 1을 반환합니다, 섹션 identifiers. 그렇지 않으면 0을 반환합니다.

버전 3.9에서 변경: 문자열이 준비(ready)되지 않았을 때, 이 함수는 더는 *Py_FatalError()*를 호출하지 않습니다.

유니코드 문자 속성

유니코드는 다양한 문자 속성을 제공합니다. 가장 자주 필요한 것은 파이썬 구성에 따라 C 함수에 매핑되는 이러한 매크로를 통해 사용할 수 있습니다.

int **Py_UNICODE_ISSPACE**(*Py_UCS4* ch)

*ch*가 공백 문자인지에 따라 1이나 0을 반환합니다.

int **Py_UNICODE_ISLOWER**(*Py_UCS4* ch)

*ch*가 소문자인지에 따라 1이나 0을 반환합니다.

int **Py_UNICODE_ISUPPER**(*Py_UCS4* ch)

*ch*가 대문자인지에 따라 1이나 0을 반환합니다.

int **Py_UNICODE_ISTITLE**(*Py_UCS4* ch)

*ch*가 제목 케이스 문자인지에 따라 1이나 0을 반환합니다.

int **Py_UNICODE_ISLINEBREAK**(*Py_UCS4* ch)

*ch*가 줄 바꿈 문자인지에 따라 1이나 0을 반환합니다.

int **Py_UNICODE_ISDECIMAL**(*Py_UCS4* ch)

*ch*가 10진수 문자인지에 따라 1이나 0을 반환합니다.

int **Py_UNICODE_ISDIGIT**(*Py_UCS4* ch)

*ch*가 디짓(digit) 문자인지에 따라 1이나 0을 반환합니다.

int **Py_UNICODE_ISNUMERIC**(*Py_UCS4* ch)

*ch*가 숫자(numeric) 문자인지에 따라 1이나 0을 반환합니다.

int **Py_UNICODE_ISALPHA**(*Py_UCS4* ch)

*ch*가 알파벳 문자인지에 따라 1이나 0을 반환합니다.

int **Py_UNICODE_ISALNUM**(*Py_UCS4* ch)

*ch*가 영숫자 문자인지에 따라 1이나 0을 반환합니다.

int **Py_UNICODE_ISPRINTABLE**(*Py_UCS4* ch)

Return 1 or 0 depending on whether *ch* is a printable character, in the sense of `str.isprintable()`.

다음 API는 빠른 직접 문자 변환에 사용할 수 있습니다:

Py_UCS4 **Py_UNICODE_TOLOWER**(*Py_UCS4* ch)

소문자로 변환된 문자 *ch*를 반환합니다.

버전 3.3부터 폐지됨: 이 함수는 간단한 케이스 매핑을 사용합니다.

Py_UCS4 **Py_UNICODE_TOUPPER**(*Py_UCS4* ch)

대문자로 변환된 문자 *ch*를 반환합니다.

버전 3.3부터 폐지됨: 이 함수는 간단한 케이스 매핑을 사용합니다.

Py_UCS4 **Py_UNICODE_TOTITLE**(*Py_UCS4* ch)

제목 케이스로 변환된 문자 *ch*를 반환합니다.

버전 3.3부터 폐지됨: 이 함수는 간단한 케이스 매핑을 사용합니다.

int **Py_UNICODE_TODECIMAL**(*Py_UCS4* ch)

Return the character *ch* converted to a decimal positive integer. Return `-1` if this is not possible. This function does not raise exceptions.

int **Py_UNICODE_TODIGIT**(*Py_UCS4* ch)

Return the character *ch* converted to a single digit integer. Return `-1` if this is not possible. This function does not raise exceptions.

double **Py_UNICODE_TONUMERIC**(*Py_UCS4* ch)

Return the character *ch* converted to a double. Return `-1.0` if this is not possible. This function does not raise exceptions.

다음 API를 사용하여 서로게이트를 다룰 수 있습니다:

int **Py_UNICODE_IS_SURROGATE**(*Py_UCS4* ch)

*ch*가 서로게이트인지 확인합니다 (`0xD800 <= ch <= 0xDFFF`).

int **Py_UNICODE_IS_HIGH_SURROGATE**(*Py_UCS4* ch)

*ch*가 상위 서로게이트인지 확인합니다 (`0xD800 <= ch <= 0xDBFF`).

int **Py_UNICODE_IS_LOW_SURROGATE**(*Py_UCS4* ch)

*ch*가 하위 서로게이트인지 확인합니다 (`0xDC00 <= ch <= 0xDFFF`).

Py_UCS4 **Py_UNICODE_JOIN_SURROGATES**(*Py_UCS4* high, *Py_UCS4* low)

Join two surrogate characters and return a single *Py_UCS4* value. *high* and *low* are respectively the leading and trailing surrogates in a surrogate pair. *high* must be in the range [0xD800; 0xDBFF] and *low* must be in the range [0xDC00; 0xDFFF].

유니코드 문자열 생성과 액세스

유니코드 객체를 만들고 기본 시퀀스 속성에 액세스하려면 다음 API를 사용하십시오:

PyObject ***PyUnicode_New**(*Py_ssize_t* size, *Py_UCS4* maxchar)

Return value: New reference. 새 유니코드 객체를 만듭니다. *maxchar*은 문자열에 배치할 실제 최대 코드 포인트여야 합니다. 근삿값으로, 127, 255, 65535, 1114111 시퀀스에서 가장 가까운 값으로 올림 할 수 있습니다.

이것은 새 유니코드 객체를 할당하는 데 권장되는 방법입니다. 이 함수를 사용하여 만든 객체는 크기를 조정할 수 없습니다.

On error, set an exception and return `NULL`.

Added in version 3.3.

PyObject ***PyUnicode_FromKindAndData**(int kind, const void *buffer, *Py_ssize_t* size)

Return value: New reference. 주어진 *kind*(가능한 값은 *PyUnicode_KIND()*에 의해 반환된 *PyUnicode_1BYTE_KIND* 등입니다)로 새로운 유니코드 객체를 만듭니다. *buffer*는 kind에 따라 문자 당 1, 2 또는 4바이트의 *size* 단위의 배열을 가리켜야 합니다.

If necessary, the input *buffer* is copied and transformed into the canonical representation. For example, if the *buffer* is a UCS4 string (*PyUnicode_4BYTE_KIND*) and it consists only of codepoints in the UCS1 range, it will be transformed into UCS1 (*PyUnicode_1BYTE_KIND*).

Added in version 3.3.

PyObject ***PyUnicode_FromStringAndSize**(const char *str, *Py_ssize_t* size)

Return value: New reference. Part of the Stable ABI. Create a Unicode object from the char buffer *str*. The bytes will be interpreted as being UTF-8 encoded. The buffer is copied into the new object. The return value might be a shared object, i.e. modification of the data is not allowed.

This function raises `SystemError` when:

- *size* < 0,
- *str* is `NULL` and *size* > 0

버전 3.12에서 변경: *str* == `NULL` with *size* > 0 is not allowed anymore.

PyObject ***PyUnicode_FromString**(const char *str)

Return value: New reference. Part of the Stable ABI. Create a Unicode object from a UTF-8 encoded null-terminated char buffer *str*.

PyObject ***PyUnicode_FromFormat**(const char *format, ...)

Return value: New reference. Part of the Stable ABI. Take a C `printf()`-style *format* string and a variable number of arguments, calculate the size of the resulting Python Unicode string and return a string with the values formatted into it. The variable arguments must be C types and must correspond exactly to the format characters in the *format* ASCII-encoded string.

A conversion specifier contains two or more characters and has the following components, which must occur in this order:

1. The `'%'` character, which marks the start of the specifier.
2. Conversion flags (optional), which affect the result of some conversion types.
3. Minimum field width (optional). If specified as an `'*'` (asterisk), the actual width is given in the next argument, which must be of type `int`, and the object to convert comes after the minimum field width and optional precision.
4. Precision (optional), given as a `'.'` (dot) followed by the precision. If specified as `'*'` (an asterisk), the actual precision is given in the next argument, which must be of type `int`, and the value to convert comes after the precision.
5. Length modifier (optional).
6. Conversion type.

The conversion flag characters are:

Flag	Meaning
`0`	The conversion will be zero padded for numeric values.
`-`	The converted value is left adjusted (overrides the `0` flag if both are given).

The length modifiers for following integer conversions (`d`, `i`, `o`, `u`, `x`, or `X`) specify the type of the argument (`int` by default):

Modifier	Types
`l`	`long` or `unsigned long`
`ll`	`long long` or `unsigned long long`
`j`	`intmax_t` or `uintmax_t`
`z`	`size_t` or `ssize_t`
`t`	`ptrdiff_t`

The length modifier `l` for following conversions `s` or `V` specify that the type of the argument is `const wchar_t*`.

The conversion specifiers are:

Conversion Specifier	형	주석
`%`	*n/a*	The literal `%` character.
`d`, `i`	Specified by the length modifier	The decimal representation of a signed C integer.
`u`	Specified by the length modifier	The decimal representation of an unsigned C integer.
`o`	Specified by the length modifier	The octal representation of an unsigned C integer.
`x`	Specified by the length modifier	The hexadecimal representation of an unsigned C integer (lowercase).
`X`	Specified by the length modifier	The hexadecimal representation of an unsigned C integer (uppercase).
`c`	`int`	A single character.
`s`	`const char*` or `const wchar_t*`	널-종료 C 문자 배열.
`p`	`const void*`	The hex representation of a C pointer. Mostly equivalent to `printf("%p")` except that it is guaranteed to start with the literal `0x` regardless of what the platform's `printf` yields.
`A`	*PyObject**	`ascii()`를 호출한 결과.
`U`	*PyObject**	유니코드 객체.
`V`	*PyObject**, `const char*` or `const wchar_t*`	유니코드 객체(`NULL`일 수 있습니다)와 두 번째 매개 변수로서 널-종료 C 문자 배열 (첫 번째 매개 변수가 `NULL`이면 사용됩니다).
`S`	*PyObject**	*PyObject_Str()*을 호출한 결과.
`R`	*PyObject**	*PyObject_Repr()*을 호출한 결과.

참고

The width formatter unit is number of characters rather than bytes. The precision formatter unit is number of bytes or `wchar_t` items (if the length modifier `l` is used) for `"%s"` and `"%V"` (if the `PyObject*` argument is `NULL`), and a number of characters for `"%A"`, `"%U"`, `"%S"`, `"%R"` and `"%V"` (if the `PyObject*` argument is not `NULL`).

참고

Unlike to C `printf()` the `0` flag has effect even when a precision is given for integer conversions (`d`, `i`, `u`, `o`, `x`, or `X`).

버전 3.2에서 변경: `"%lld"`와 `"%llu"`에 대한 지원이 추가되었습니다.

버전 3.3에서 변경: `"%li"`, `"%lli"` 및 `"%zi"`에 대한 지원이 추가되었습니다.

버전 3.4에서 변경: `"%s"`, `"%A"`, `"%U"`, `"%V"`, `"%S"`, `"%R"`에 대한 너비와 정밀도 포매터 지원이 추가되었습니다.

버전 3.12에서 변경: Support for conversion specifiers `o` and `X`. Support for length modifiers `j` and `t`. Length modifiers are now applied to all integer conversions. Length modifier `l` is now applied to conversion specifiers `s` and `V`. Support for variable width and precision `*`. Support for flag `-`.

An unrecognized format character now sets a `SystemError`. In previous versions it caused all the rest of the format string to be copied as-is to the result string, and any extra arguments discarded.

PyObject ***PyUnicode_FromFormatV**(const char *format, va_list vargs)

Return value: New reference. Part of the Stable ABI. 정확히 두 개의 인자를 취한다는 점을 제외하면 *PyUnicode_FromFormat()*과 동일합니다.

PyObject ***PyUnicode_FromObject**(*PyObject* *obj)

Return value: New reference. Part of the Stable ABI. Copy an instance of a Unicode subtype to a new true Unicode object if necessary. If *obj* is already a true Unicode object (not a subtype), return a new *strong reference* to the object.

유니코드나 이의 서브 형 이외의 객체는 `TypeError`를 발생시킵니다.

PyObject ***PyUnicode_FromEncodedObject**(*PyObject* *obj, const char *encoding, const char *errors)

Return value: New reference. Part of the Stable ABI. 인코딩된 객체 *obj*를 유니코드 객체로 디코딩합니다.

`bytes`, `bytearray` 및 기타 바이트열류 객체는 주어진 *encoding*에 따라 *errors*로 정의한 에러 처리를 사용하여 디코딩됩니다. 둘 다 `NULL`이 될 수 있고, 이 경우 인터페이스는 기본값을 사용합니다 (자세한 내용은 내장 코덱를 참조하십시오).

유니코드 객체를 포함한 다른 모든 객체는 `TypeError`가 설정되도록 합니다.

API는 에러가 있으면 `NULL`을 반환합니다. 호출자는 반환된 객체의 참조 횟수를 감소시킬 책임이 있습니다.

const char ***PyUnicode_GetDefaultEncoding**(void)

Part of the Stable ABI. Return the name of the default string encoding, `"utf-8"`. See `sys.getdefaultencoding()`.

The returned string does not need to be freed, and is valid until interpreter shutdown.

Py_ssize_t **PyUnicode_GetLength**(*PyObject* *unicode)

Part of the Stable ABI *since version 3.7.* 유니코드 객체의 길이를 코드 포인트로 반환합니다.

On error, set an exception and return `-1`.

Added in version 3.3.

Py_ssize_t **PyUnicode_CopyCharacters**(*PyObject* *to, *Py_ssize_t* to_start, *PyObject* *from, *Py_ssize_t* from_start, *Py_ssize_t* how_many)

Copy characters from one Unicode object into another. This function performs character conversion when necessary and falls back to `memcpy()` if possible. Returns `-1` and sets an exception on error, otherwise returns the number of copied characters.

Added in version 3.3.

Py_ssize_t **PyUnicode_Fill**(*PyObject* *unicode, *Py_ssize_t* start, *Py_ssize_t* length, *Py_UCS4* fill_char)

문자로 문자열을 채웁니다: *fill_char*을 `unicode[start:start+length]`에 씁니다.

*fill_char*이 문자열 최대 문자보다 크거나, 문자열에 둘 이상의 참조가 있으면 실패합니다.

기록된 문자 수를 반환하거나, 에러 시 `-1`을 반환하고 예외를 발생시킵니다.

Added in version 3.3.

int **PyUnicode_WriteChar**(*PyObject* *unicode, *Py_ssize_t* index, *Py_UCS4* character)

Part of the Stable ABI *since version 3.7.* 문자열에 문자를 씁니다. 문자열은 *PyUnicode_New()*를 통해 만들었어야 합니다. 유니코드 문자열은 불변이므로, 문자열을 공유하거나 아직 해시 하지 않아야 합니다.

이 함수는 *unicode*가 유니코드 객체인지, 인덱스가 범위를 벗어났는지, 객체가 안전하게 수정될 수 있는지 (즉, 참조 횟수가 1인지) 확인합니다.

Return `0` on success, `-1` on error with an exception set.

Added in version 3.3.

Py_UCS4 **PyUnicode_ReadChar**(*PyObject* *unicode, *Py_ssize_t* index)

Part of the Stable ABI *since version 3.7.* Read a character from a string. This function checks that *unicode* is a Unicode object and the index is not out of bounds, in contrast to `PyUnicode_READ_CHAR()`, which performs no error checking.

Return character on success, `-1` on error with an exception set.

Added in version 3.3.

PyObject ***PyUnicode_Substring**(*PyObject* *unicode, *Py_ssize_t* start, *Py_ssize_t* end)

Return value: New reference. Part of the Stable ABI *since version 3.7.* Return a substring of *unicode*, from character index *start* (included) to character index *end* (excluded). Negative indices are not supported. On error, set an exception and return `NULL`.

Added in version 3.3.

Py_UCS4 ***PyUnicode_AsUCS4**(*PyObject* *unicode, *Py_UCS4* *buffer, *Py_ssize_t* buflen, int copy_null)

Part of the Stable ABI *since version 3.7.* Copy the string *unicode* into a UCS4 buffer, including a null character, if *copy_null* is set. Returns `NULL` and sets an exception on error (in particular, a `SystemError` if *buflen* is smaller than the length of *unicode*). *buffer* is returned on success.

Added in version 3.3.

Py_UCS4 ***PyUnicode_AsUCS4Copy**(*PyObject* *unicode)

Part of the Stable ABI *since version 3.7.* Copy the string *unicode* into a new UCS4 buffer that is allocated using `PyMem_Malloc()`. If this fails, `NULL` is returned with a `MemoryError` set. The returned buffer always has an extra null code point appended.

Added in version 3.3.

로케일 인코딩

현재 로케일 인코딩을 사용하여 운영 체제에서 온 텍스트를 디코딩 할 수 있습니다.

PyObject ***PyUnicode_DecodeLocaleAndSize**(const char *str, *Py_ssize_t* length, const char *errors)

Return value: New reference. Part of the Stable ABI *since version 3.7.* 안드로이드와 VxWorks의 UTF-8이나 다른 플랫폼의 현재 로케일 인코딩의 문자열을 디코딩합니다. 지원되는 에러 처리기는 `"strict"`와 `"surrogateescape"`(**PEP 383**)입니다. 디코더는 *errors*가 `NULL`이면 `"strict"` 에러 처리기를 사용합니다. *str*은 널 문자로 끝나야 하지만 널 문자를 포함할 수 없습니다.

Use `PyUnicode_DecodeFSDefaultAndSize()` to decode a string from the *filesystem encoding and error handler*.

This function ignores the Python UTF-8 Mode.

> **더 보기**
>
> `Py_DecodeLocale()` 함수.

Added in version 3.3.

버전 3.7에서 변경: 이 함수는 이제 안드로이드를 제외하고 `surrogateescape` 에러 처리기에 현재 로케일 인코딩도 사용합니다. 이전에는, `Py_DecodeLocale()`이 `surrogateescape`에 사용되었고, 현재 로케일 인코딩은 `strict`에 사용되었습니다.

PyObject ***PyUnicode_DecodeLocale**(const char *str, const char *errors)

Return value: New reference. Part of the Stable ABI *since version 3.7.* Similar to `PyUnicode_DecodeLocaleAndSize()`, but compute the string length using `strlen()`.

Added in version 3.3.

PyObject ***PyUnicode_EncodeLocale**(*PyObject* *unicode, const char *errors)

Return value: New reference. Part of the Stable ABI *since version 3.7.* 유니코드 객체를 안드로이드와 VxWorks에서 UTF-8로 인코딩하거나, 다른 플랫폼에서 현재 로케일 인코딩으로 인코딩합니다. 지원되는 에러 처리기는 `"strict"`와 `"surrogateescape"`(**PEP 383**)입니다. 인코더는 *errors*가 `NULL`이면 `"strict"` 에러 처리기를 사용합니다. `bytes` 객체를 반환합니다. *unicode*는 내장된 널 문자를 포함할 수 없습니다.

Use `PyUnicode_EncodeFSDefault()` to encode a string to the *filesystem encoding and error handler*.

This function ignores the Python UTF-8 Mode.

더 보기

`Py_EncodeLocale()` 함수.

Added in version 3.3.

버전 3.7에서 변경: 이 함수는 이제 안드로이드를 제외하고 `surrogateescape` 에러 처리기에 현재 로케일 인코딩도 사용합니다. 이전에는 `Py_EncodeLocale()`이 `surrogateescape`에 사용되었고, 현재 로케일 인코딩은 `strict`에 사용되었습니다.

파일 시스템 인코딩

Functions encoding to and decoding from the *filesystem encoding and error handler* (**PEP 383** and **PEP 529**).

To encode file names to `bytes` during argument parsing, the `"O&"` converter should be used, passing `PyUnicode_FSConverter()` as the conversion function:

int **PyUnicode_FSConverter**(*PyObject* *obj, void *result)

Part of the Stable ABI. *PyArg_Parse* converter*: encode `str` objects – obtained directly or through the `os.PathLike` interface – to `bytes` using `PyUnicode_EncodeFSDefault()`; `bytes` objects are output as-is. *result* must be an address of a C variable of type `PyObject*` (or `PyBytesObject*`). On success, set the variable to a new *strong reference* to a *bytes object* which must be released when it is no longer used and return a non-zero value (`Py_CLEANUP_SUPPORTED`). Embedded null bytes are not allowed in the result. On failure, return `0` with an exception set.

If *obj* is `NULL`, the function releases a strong reference stored in the variable referred by *result* and returns `1`.

Added in version 3.1.

버전 3.6에서 변경: 경로류 객체를 받아들입니다.

To decode file names to `str` during argument parsing, the `"O&"` converter should be used, passing `PyUnicode_FSDecoder()` as the conversion function:

int **PyUnicode_FSDecoder**(*PyObject* *obj, void *result)

Part of the Stable ABI. *PyArg_Parse* converter*: decode `bytes` objects – obtained either directly or indirectly through the `os.PathLike` interface – to `str` using `PyUnicode_DecodeFSDefaultAndSize()`; `str` objects are output as-is. *result* must be an address of a C variable of type `PyObject*` (or `PyUnicodeObject*`). On success, set the variable to a new *strong reference* to a *Unicode object* which must be released when it is no longer used and return a non-zero value (`Py_CLEANUP_SUPPORTED`). Embedded null characters are not allowed in the result. On failure, return `0` with an exception set.

If *obj* is `NULL`, release the strong reference to the object referred to by *result* and return `1`.

Added in version 3.2.

버전 3.6에서 변경: 경로류 객체를 받아들입니다.

PyObject ***PyUnicode_DecodeFSDefaultAndSize**(const char *str, *Py_ssize_t* size)

Return value: New reference. Part of the Stable ABI. Decode a string from the *filesystem encoding and error handler*.

If you need to decode a string from the current locale encoding, use `PyUnicode_DecodeLocaleAndSize()`.

> **더 보기**
>
> *Py_DecodeLocale()* 함수.

버전 3.6에서 변경: The *filesystem error handler* is now used.

PyObject ***PyUnicode_DecodeFSDefault**(const char *str)

Return value: New reference. Part of the Stable ABI. Decode a null-terminated string from the *filesystem encoding and error handler*.

If the string length is known, use *PyUnicode_DecodeFSDefaultAndSize()*.

버전 3.6에서 변경: The *filesystem error handler* is now used.

PyObject ***PyUnicode_EncodeFSDefault**(*PyObject* *unicode)

Return value: New reference. Part of the Stable ABI. Encode a Unicode object to the *filesystem encoding and error handler*, and return `bytes`. Note that the resulting `bytes` object can contain null bytes.

If you need to encode a string to the current locale encoding, use *PyUnicode_EncodeLocale()*.

> **더 보기**
>
> *Py_EncodeLocale()* 함수.

Added in version 3.2.

버전 3.6에서 변경: The *filesystem error handler* is now used.

wchar_t 지원

지원하는 플랫폼에 대한 `wchar_t` 지원:

PyObject ***PyUnicode_FromWideChar**(const wchar_t *wstr, *Py_ssize_t* size)

Return value: New reference. Part of the Stable ABI. Create a Unicode object from the `wchar_t` buffer *wstr* of the given *size*. Passing `-1` as the *size* indicates that the function must itself compute the length, using `wcslen()`. Return `NULL` on failure.

Py_ssize_t **PyUnicode_AsWideChar**(*PyObject* *unicode, wchar_t *wstr, *Py_ssize_t* size)

Part of the Stable ABI. Copy the Unicode object contents into the `wchar_t` buffer *wstr*. At most *size* `wchar_t` characters are copied (excluding a possibly trailing null termination character). Return the number of `wchar_t` characters copied or `-1` in case of an error.

When *wstr* is `NULL`, instead return the *size* that would be required to store all of *unicode* including a terminating null.

Note that the resulting `wchar_t*` string may or may not be null-terminated. It is the responsibility of the caller to make sure that the `wchar_t*` string is null-terminated in case this is required by the application. Also, note that the `wchar_t*` string might contain null characters, which would cause the string to be truncated when used with most C functions.

wchar_t ***PyUnicode_AsWideCharString**(*PyObject* *unicode, *Py_ssize_t* *size)

Part of the Stable ABI *since version 3.7.* Convert the Unicode object to a wide character string. The output string always ends with a null character. If *size* is not `NULL`, write the number of wide characters (excluding the trailing null termination character) into **size*. Note that the resulting `wchar_t` string might contain null characters, which would cause the string to be truncated when used with most C functions. If *size* is `NULL` and the `wchar_t*` string contains null characters a `ValueError` is raised.

Returns a buffer allocated by *PyMem_New* (use *PyMem_Free()* to free it) on success. On error, returns `NULL` and **size* is undefined. Raises a `MemoryError` if memory allocation is failed.

Added in version 3.2.

버전 3.7에서 변경: Raises a `ValueError` if *size* is `NULL` and the `wchar_t*` string contains null characters.

내장 코덱

파이썬은 속도를 위해 C로 작성된 내장 코덱 집합을 제공합니다. 이러한 코덱들은 모두 다음 함수들을 통해 직접 사용할 수 있습니다.

다음 API의 대부분은 두 개의 인자 encoding과 errors를 취하며, 내장 `str()` 문자열 객체 생성자의 것들과 같은 의미입니다.

Setting encoding to `NULL` causes the default encoding to be used which is UTF-8. The file system calls should use *PyUnicode_FSConverter()* for encoding file names. This uses the *filesystem encoding and error handler* internally.

에러 처리는 errors로 설정되는데, 코덱에 대해 정의된 기본 처리를 사용함을 의미하는 `NULL`로 설정될 수도 있습니다. 모든 내장 코덱에 대한 기본 에러 처리는 "strict" 입니다 (`ValueError`가 발생합니다).

The codecs all use a similar interface. Only deviations from the following generic ones are documented for simplicity.

일반 코덱

다음은 일반 코덱 API입니다:

PyObject ***PyUnicode_Decode**(const char *str, *Py_ssize_t* size, const char *encoding, const char *errors)

Return value: New reference. Part of the Stable ABI. Create a Unicode object by decoding *size* bytes of the encoded string *str*. *encoding* and *errors* have the same meaning as the parameters of the same name in the `str()` built-in function. The codec to be used is looked up using the Python codec registry. Return `NULL` if an exception was raised by the codec.

PyObject ***PyUnicode_AsEncodedString**(*PyObject* *unicode, const char *encoding, const char *errors)

Return value: New reference. Part of the Stable ABI. 유니코드 객체를 인코딩하고 결과를 파이썬 bytes 객체로 반환합니다. *encoding*과 *errors*는 유니코드 `encode()` 메서드의 같은 이름의 매개 변수와 같은 의미입니다. 사용할 코덱은 파이썬 코덱 레지스트리를 사용하여 조회됩니다. 코덱에서 예외가 발생하면 `NULL`을 반환합니다.

UTF-8 코덱

다음은 UTF-8 코덱 API입니다:

PyObject ***PyUnicode_DecodeUTF8**(const char *str, *Py_ssize_t* size, const char *errors)

Return value: New reference. Part of the Stable ABI. Create a Unicode object by decoding *size* bytes of the UTF-8 encoded string *str*. Return `NULL` if an exception was raised by the codec.

PyObject ***PyUnicode_DecodeUTF8Stateful**(const char *str, *Py_ssize_t* size, const char *errors, *Py_ssize_t* *consumed)

Return value: New reference. Part of the Stable ABI. *consumed*가 `NULL`이면, *PyUnicode_DecodeUTF8()*처럼 동작합니다. *consumed*가 `NULL`이 아니면, 후행 불완전한 UTF-8 바이트 시퀀스는 에러로 처리되지 않습니다. 이러한 바이트는 디코딩되지 않으며 디코딩된 바이트 수는 *consumed*에 저장됩니다.

PyObject ***PyUnicode_AsUTF8String**(*PyObject* *unicode)

Return value: New reference. Part of the Stable ABI. UTF-8을 사용하여 유니코드 객체를 인코딩하고 결과를 파이썬 bytes 객체로 반환합니다. 에러 처리는 "strict" 입니다. 코덱에서 예외가 발생하면 `NULL`을 반환합니다.

const char ***PyUnicode_AsUTF8AndSize**(*PyObject* *unicode, *Py_ssize_t* *size)

Part of the Stable ABI *since version 3.10.* 유니코드 객체의 UTF-8 인코딩에 대한 포인터를 반환하고, 인코딩된 표현의 크기를 (바이트 단위로) *size*에 저장합니다. *size* 인자는 `NULL`일 수 있습니다; 이 경우 크기가 저장되지 않습니다. 반환된 버퍼에는 다른 널 코드 포인트가 있는지에 관계없이, 항상 추가 널 바이트가 추가됩니다 (*size*에 포함되지 않습니다).

에러가 발생하면, `NULL`이 예외 설정과 함께 반환되고 *size*가 저장되지 않습니다.

This caches the UTF-8 representation of the string in the Unicode object, and subsequent calls will return a pointer to the same buffer. The caller is not responsible for deallocating the buffer. The buffer is deallocated and pointers to it become invalid when the Unicode object is garbage collected.

Added in version 3.3.

버전 3.7에서 변경: 반환형은 이제 `char *`가 아니라 `const char *`입니다.

버전 3.10에서 변경: This function is a part of the *limited API*.

const char ***PyUnicode_AsUTF8**(*PyObject* *unicode)

*PyUnicode_AsUTF8AndSize()*와 같지만, 크기를 저장하지 않습니다.

경고

This function does not have any special behavior for null characters embedded within *unicode*. As a result, strings containing null characters will remain in the returned string, which some C functions might interpret as the end of the string, leading to truncation. If truncation is an issue, it is recommended to use *PyUnicode_AsUTF8AndSize()* instead.

Added in version 3.3.

버전 3.7에서 변경: 반환형은 이제 `char *`가 아니라 `const char *`입니다.

UTF-32 코덱

다음은 UTF-32 코덱 API입니다:

PyObject ***PyUnicode_DecodeUTF32**(const char *str, *Py_ssize_t* size, const char *errors, int *byteorder)

Return value: New reference. Part of the Stable ABI. UTF-32로 인코딩된 버퍼 문자열에서 *size* 바이트를 디코딩하고 해당 유니코드 객체를 반환합니다. *errors*(`NULL`이 아니면)는 에러 처리를 정의합니다. 기본값은 "strict"입니다.

*byteorder*가 `NULL`이 아니면, 디코더는 지정된 바이트 순서를 사용하여 디코딩을 시작합니다:

```
*byteorder == -1: little endian
*byteorder == 0:  native order
*byteorder == 1:  big endian
```

`*byteorder`가 0이고, 입력 데이터의 처음 4바이트가 바이트 순서 표시(BOM)이면, 디코더가 이 바이트 순서로 전환되고 BOM은 결과 유니코드 문자열에 복사되지 않습니다. `*byteorder`가 `-1`이나 `1`이면, 모든 바이트 순서 표시가 출력에 복사됩니다.

완료 후, **byteorder*는 입력 데이터의 끝에서 현재 바이트 순서로 설정됩니다.

*byteorder*가 `NULL`이면, 코덱은 네이티브 순서 모드로 시작합니다.

코덱에서 예외가 발생하면 `NULL`을 반환합니다.

PyObject ***PyUnicode_DecodeUTF32Stateful**(const char *str, *Py_ssize_t* size, const char *errors, int *byteorder, *Py_ssize_t* *consumed)

Return value: New reference. Part of the Stable ABI. *consumed*가 `NULL`이면, *PyUnicode_DecodeUTF32()*처럼 동작합니다. *consumed*가 `NULL`이 아니면, *PyUnicode_DecodeUTF32Stateful()*은 후행 불완전 UTF-32 바이트 시퀀스(가령 4로 나누어떨어지지 않는 바이트 수)를 에러로 처리하지 않습니다. 이러한 바이트는 디코딩되지 않으며 디코딩된 바이트 수는 *consumed*에 저장됩니다.

PyObject ***PyUnicode_AsUTF32String**(*PyObject* *unicode)

Return value: New reference. Part of the Stable ABI. 네이티브 바이트 순서로 UTF-32 인코딩을 사용하여 파이썬 바이트 문자열을 반환합니다. 문자열은 항상 BOM 마크로 시작합니다. 에러 처리는 "strict"입니다. 코덱에서 예외가 발생하면 `NULL`을 반환합니다.

UTF-16 코덱

다음은 UTF-16 코덱 API입니다:

PyObject ***PyUnicode_DecodeUTF16**(const char *str, *Py_ssize_t* size, const char *errors, int *byteorder)

Return value: New reference. Part of the Stable ABI. UTF-16으로 인코딩된 버퍼 문자열에서 *size* 바이트를 디코딩하고 해당 유니코드 객체를 반환합니다. *errors*(`NULL`이 아니면)는 에러 처리를 정의합니다. 기본값은 "strict" 입니다.

*byteorder*가 `NULL`이 아니면, 디코더는 지정된 바이트 순서를 사용하여 디코딩을 시작합니다:

```
*byteorder == -1: little endian
*byteorder == 0:  native order
*byteorder == 1:  big endian
```

`*byteorder`가 0이고, 입력 데이터의 처음 2바이트가 바이트 순서 표시(BOM)이면, 디코더는 이 바이트 순서로 전환되고 BOM은 결과 유니코드 문자열에 복사되지 않습니다. `*byteorder`가 `-1`이나 `1`이면 모든 바이트 순서 표시가 출력에 복사됩니다 (`\ufeff`나 `\ufffe` 문자가 됩니다).

After completion, `*byteorder` is set to the current byte order at the end of input data.

*byteorder*가 `NULL`이면, 코덱은 네이티브 순서 모드로 시작합니다.

코덱에서 예외가 발생하면 `NULL`을 반환합니다.

PyObject ***PyUnicode_DecodeUTF16Stateful**(const char *str, *Py_ssize_t* size, const char *errors, int *byteorder, *Py_ssize_t* *consumed)

Return value: New reference. Part of the Stable ABI. *consumed*가 `NULL`이면, *PyUnicode_DecodeUTF16()*처럼 동작합니다. *consumed*가 `NULL`이 아니면, *PyUnicode_DecodeUTF16Stateful()*은 후행 불완전 UTF-16 바이트 시퀀스(가령 홀수 바이트 수나 분할 서로게이트 쌍)를 에러로 처리하지 않습니다. 이러한 바이트는 디코딩되지 않으며 디코딩된 바이트 수는 *consumed*에 저장됩니다.

PyObject ***PyUnicode_AsUTF16String**(*PyObject* *unicode)

Return value: New reference. Part of the Stable ABI. 네이티브 바이트 순서로 UTF-16 인코딩을 사용하여 파이썬 바이트 문자열을 반환합니다. 문자열은 항상 BOM 마크로 시작합니다. 에러 처리는 "strict" 입니다. 코덱에서 예외가 발생하면 `NULL`을 반환합니다.

UTF-7 코덱

다음은 UTF-7 코덱 API입니다:

PyObject ***PyUnicode_DecodeUTF7**(const char *str, *Py_ssize_t* size, const char *errors)

Return value: New reference. Part of the Stable ABI. Create a Unicode object by decoding *size* bytes of the UTF-7 encoded string *str*. Return `NULL` if an exception was raised by the codec.

PyObject ***PyUnicode_DecodeUTF7Stateful**(const char *str, *Py_ssize_t* size, const char *errors, *Py_ssize_t* *consumed)

Return value: New reference. Part of the Stable ABI. *consumed*가 `NULL`이면, *PyUnicode_DecodeUTF7()*처럼 동작합니다. *consumed*가 `NULL`이 아니면, 후행 불완전한 UTF-7 base-64 섹션은 에러로 처리되지 않습니다. 이러한 바이트는 디코딩되지 않으며 디코딩된 바이트 수는 *consumed*에 저장됩니다.

유니코드 이스케이프 코덱

다음은 "유니코드 이스케이프(Unicode Escape)" 코덱 API입니다:

PyObject ***PyUnicode_DecodeUnicodeEscape**(const char *str, *Py_ssize_t* size, const char *errors)

Return value: New reference. Part of the Stable ABI. Create a Unicode object by decoding *size* bytes of the Unicode-Escape encoded string *str*. Return `NULL` if an exception was raised by the codec.

PyObject ***PyUnicode_AsUnicodeEscapeString**(*PyObject* *unicode)

Return value: New reference. Part of the Stable ABI. 유니코드 이스케이프를 사용하여 유니코드 객체를 인코딩하고 결과를 bytes 객체로 반환합니다. 에러 처리는 "strict" 입니다. 코덱에서 예외가 발생하면 NULL을 반환합니다.

원시 유니코드 이스케이프 코덱

다음은 "원시 유니코드 이스케이프(Raw Unicode Escape)" 코덱 API입니다:

PyObject ***PyUnicode_DecodeRawUnicodeEscape**(const char *str, *Py_ssize_t* size, const char *errors)

Return value: New reference. Part of the Stable ABI. Create a Unicode object by decoding *size* bytes of the Raw-Unicode-Escape encoded string *str*. Return NULL if an exception was raised by the codec.

PyObject ***PyUnicode_AsRawUnicodeEscapeString**(*PyObject* *unicode)

Return value: New reference. Part of the Stable ABI. 원시 유니코드 이스케이프를 사용하여 유니코드 객체를 인코딩하고 결과를 bytes 객체로 반환합니다. 에러 처리는 "strict" 입니다. 코덱에서 예외가 발생하면 NULL을 반환합니다.

Latin-1 코덱

다음은 Latin-1 코덱 API입니다: Latin-1은 처음 256개의 유니코드 서수에 해당하며 인코딩 중에 코덱에서 이들만 허용됩니다.

PyObject ***PyUnicode_DecodeLatin1**(const char *str, *Py_ssize_t* size, const char *errors)

Return value: New reference. Part of the Stable ABI. Create a Unicode object by decoding *size* bytes of the Latin-1 encoded string *str*. Return NULL if an exception was raised by the codec.

PyObject ***PyUnicode_AsLatin1String**(*PyObject* *unicode)

Return value: New reference. Part of the Stable ABI. Latin-1을 사용하여 유니코드 객체를 인코딩하고 결과를 파이썬 bytes 객체로 반환합니다. 에러 처리는 "strict" 입니다. 코덱에서 예외가 발생하면 NULL을 반환합니다.

ASCII 코덱

다음은 ASCII 코덱 API입니다. 7비트 ASCII 데이터만 허용됩니다. 다른 모든 코드는 에러를 생성합니다.

PyObject ***PyUnicode_DecodeASCII**(const char *str, *Py_ssize_t* size, const char *errors)

Return value: New reference. Part of the Stable ABI. Create a Unicode object by decoding *size* bytes of the ASCII encoded string *str*. Return NULL if an exception was raised by the codec.

PyObject ***PyUnicode_AsASCIIString**(*PyObject* *unicode)

Return value: New reference. Part of the Stable ABI. ASCII를 사용하여 유니코드 객체를 인코딩하고 결과를 파이썬 bytes 객체로 반환합니다. 에러 처리는 "strict" 입니다. 코덱에서 예외가 발생하면 NULL을 반환합니다.

문자 맵 코덱

This codec is special in that it can be used to implement many different codecs (and this is in fact what was done to obtain most of the standard codecs included in the encodings package). The codec uses mappings to encode and decode characters. The mapping objects provided must support the __getitem__() mapping interface; dictionaries and sequences work well.

다음은 매핑 코덱 API입니다:

PyObject ***PyUnicode_DecodeCharmap**(const char *str, *Py_ssize_t* length, *PyObject* *mapping, const char *errors)

Return value: New reference. Part of the Stable ABI. Create a Unicode object by decoding *size* bytes of the encoded string *str* using the given *mapping* object. Return NULL if an exception was raised by the codec.

*mapping*이 NULL이면, Latin-1 디코딩이 적용됩니다. 그렇지 않으면 *mapping*은 바이트 서수(0에서 255 사이의 정수)를 유니코드 문자열, 정수(유니코드 서수로 해석됩니다) 또는 None으로 매핑해

야합니다. 매핑되지 않은 데이터 바이트(None, 0xFFFE 또는 '\ufffe'로 매핑되는 것뿐만 아니라, LookupError를 유발하는 것)은 정의되지 않은 매핑으로 처리되어 에러를 발생시킵니다.

PyObject ***PyUnicode_AsCharmapString**(*PyObject* *unicode, *PyObject* *mapping)

Return value: New reference. Part of the Stable ABI. 주어진 *mapping* 객체를 사용하여 유니코드 객체를 인코딩하고 결과를 bytes 객체로 반환합니다. 에러 처리는 "strict" 입니다. 코덱에서 예외가 발생하면 NULL을 반환합니다.

mapping 객체는 유니코드 서수 정수를 bytes 객체, 0에서 255 사이의 정수 또는 None으로 매핑해야 합니다. None에 매핑되는 것뿐만 아니라 매핑되지 않은 문자 서수(LookupError를 유발하는 것)는 "정의되지 않은 매핑"으로 처리되어 에러가 발생합니다.

다음 코덱 API는 유니코드를 유니코드로 매핑한다는 점에서 특별합니다.

PyObject ***PyUnicode_Translate**(*PyObject* *unicode, *PyObject* *table, const char *errors)

Return value: New reference. Part of the Stable ABI. 문자 매핑 테이블을 적용하여 문자열을 변환하고 결과 유니코드 객체를 반환합니다. 코덱에서 예외가 발생하면 NULL을 반환합니다.

매핑 테이블은 유니코드 서수 정수를 유니코드 서수 정수나 None(문자가 삭제되도록 합니다)에 매핑해야 합니다.

Mapping tables need only provide the `__getitem__()` interface; dictionaries and sequences work well. Unmapped character ordinals (ones which cause a LookupError) are left untouched and are copied as-is.

*errors*는 코덱에서의 일반적인 의미입니다. 기본 에러 처리를 사용함을 나타내는 NULL일 수 있습니다.

윈도우 용 MBCS 코덱

다음은 MBCS 코덱 API입니다. 현재 윈도우에서만 사용할 수 있으며 Win32 MBCS 변환기를 사용하여 변환을 구현합니다. MBCS(또는 DBCS)는 단지 하나가 아니라 인코딩 클래스임에 유의하십시오. 대상 인코딩은 코덱을 실행하는 기계의 사용자 설정에 의해 정의됩니다.

PyObject ***PyUnicode_DecodeMBCS**(const char *str, *Py_ssize_t* size, const char *errors)

Return value: New reference. Part of the Stable ABI *on Windows since version 3.7.* Create a Unicode object by decoding *size* bytes of the MBCS encoded string *str*. Return NULL if an exception was raised by the codec.

PyObject ***PyUnicode_DecodeMBCSStateful**(const char *str, *Py_ssize_t* size, const char *errors, *Py_ssize_t* *consumed)

Return value: New reference. Part of the Stable ABI *on Windows since version 3.7.* *consumed*가 NULL이면, *PyUnicode_DecodeMBCS()*처럼 동작합니다. *consumed*가 NULL이 아니면, *PyUnicode_DecodeMBCSStateful()*은 후행 선행(lead) 바이트를 디코딩하지 않고 디코딩된 바이트 수가 *consumed*에 저장됩니다.

PyObject ***PyUnicode_DecodeCodePageStateful**(int code_page, const char *str, *Py_ssize_t* size, const char *errors, *Py_ssize_t* *consumed)

Return value: New reference. Part of the Stable ABI *on Windows since version 3.7.* Similar to *PyUnicode_DecodeMBCSStateful()*, except uses the code page specified by *code_page*.

PyObject ***PyUnicode_AsMBCSString**(*PyObject* *unicode)

Return value: New reference. Part of the Stable ABI *on Windows since version 3.7.* MBCS를 사용하여 유니코드 객체를 인코딩하고 결과를 파이썬 bytes 객체로 반환합니다. 에러 처리는 "strict" 입니다. 코덱에서 예외가 발생하면 NULL을 반환합니다.

PyObject ***PyUnicode_EncodeCodePage**(int code_page, *PyObject* *unicode, const char *errors)

Return value: New reference. Part of the Stable ABI *on Windows since version 3.7.* Encode the Unicode object using the specified code page and return a Python bytes object. Return NULL if an exception was raised by the codec. Use CP_ACP code page to get the MBCS encoder.

Added in version 3.3.

메서드와 슬롯

메서드와 슬롯 함수

다음 API는 입력의 유니코드 객체와 문자열을 (설명에서 문자열이라고 하겠습니다) 처리할 수 있으며 적절하게 유니코드 객체나 정수를 반환합니다.

예외가 발생하면 모두 `NULL`이나 `-1`을 반환합니다.

PyObject ***PyUnicode_Concat**(*PyObject* *left, *PyObject* *right)

Return value: New reference. Part of the Stable ABI. 두 문자열을 이어붙여 하나의 새로운 유니코드 문자열을 제공합니다.

PyObject ***PyUnicode_Split**(*PyObject* *unicode, *PyObject* *sep, *Py_ssize_t* maxsplit)

Return value: New reference. Part of the Stable ABI. 문자열을 분할하여 유니코드 문자열 리스트를 제공합니다. *sep*이 `NULL`이면, 모든 공백 부분 문자열에서 분할이 수행됩니다. 그렇지 않으면, 주어진 구분자에서 분할이 일어납니다. 최대 *maxsplit* 분할이 수행됩니다. 음수이면, 제한이 설정되지 않습니다. 구분자는 결과 리스트에 포함되지 않습니다.

On error, return `NULL` with an exception set.

Equivalent to `str.split()`.

PyObject ***PyUnicode_RSplit**(*PyObject* *unicode, *PyObject* *sep, *Py_ssize_t* maxsplit)

Return value: New reference. Part of the Stable ABI. Similar to *`PyUnicode_Split()`*, but splitting will be done beginning at the end of the string.

On error, return `NULL` with an exception set.

Equivalent to `str.rsplit()`.

PyObject ***PyUnicode_Splitlines**(*PyObject* *unicode, int keepends)

Return value: New reference. Part of the Stable ABI. Split a Unicode string at line breaks, returning a list of Unicode strings. CRLF is considered to be one line break. If *keepends* is `0`, the Line break characters are not included in the resulting strings.

PyObject ***PyUnicode_Partition**(*PyObject* *unicode, *PyObject* *sep)

Return value: New reference. Part of the Stable ABI. Split a Unicode string at the first occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing the string itself, followed by two empty strings.

sep must not be empty.

On error, return `NULL` with an exception set.

Equivalent to `str.partition()`.

PyObject ***PyUnicode_RPartition**(*PyObject* *unicode, *PyObject* *sep)

Return value: New reference. Part of the Stable ABI. Similar to *`PyUnicode_Partition()`*, but split a Unicode string at the last occurrence of *sep*. If the separator is not found, return a 3-tuple containing two empty strings, followed by the string itself.

sep must not be empty.

On error, return `NULL` with an exception set.

Equivalent to `str.rpartition()`.

PyObject ***PyUnicode_Join**(*PyObject* *separator, *PyObject* *seq)

Return value: New reference. Part of the Stable ABI. 주어진 *separator*를 사용하여 문자열 시퀀스를 연결하고 결과 유니코드 문자열을 반환합니다.

Py_ssize_t **PyUnicode_Tailmatch**(*PyObject* *unicode, *PyObject* *substr, *Py_ssize_t* start, *Py_ssize_t* end, int direction)

Part of the Stable ABI. Return `1` if *substr* matches `unicode[start:end]` at the given tail end (*direction* == `-1` means to do a prefix match, *direction* == `1` a suffix match), `0` otherwise. Return `-1` if an error occurred.

Py_ssize_t **PyUnicode_Find**(*PyObject* *unicode, *PyObject* *substr, *Py_ssize_t* start, *Py_ssize_t* end, int direction)

Part of the Stable ABI. Return the first position of *substr* in `unicode[start:end]` using the given *direction* (*direction* == `1` means to do a forward search, *direction* == `-1` a backward search). The return value is the index of the first match; a value of `-1` indicates that no match was found, and `-2` indicates that an error occurred and an exception has been set.

Py_ssize_t **PyUnicode_FindChar**(*PyObject* *unicode, *Py_UCS4* ch, *Py_ssize_t* start, *Py_ssize_t* end, int direction)

Part of the Stable ABI *since version 3.7.* Return the first position of the character *ch* in `unicode[start:end]` using the given *direction* (*direction* == `1` means to do a forward search, *direction* == `-1` a backward search). The return value is the index of the first match; a value of `-1` indicates that no match was found, and `-2` indicates that an error occurred and an exception has been set.

Added in version 3.3.

버전 3.7에서 변경: *start* and *end* are now adjusted to behave like `unicode[start:end]`.

Py_ssize_t **PyUnicode_Count**(*PyObject* *unicode, *PyObject* *substr, *Py_ssize_t* start, *Py_ssize_t* end)

Part of the Stable ABI. Return the number of non-overlapping occurrences of *substr* in `unicode[start:end]`. Return `-1` if an error occurred.

PyObject ***PyUnicode_Replace**(*PyObject* *unicode, *PyObject* *substr, *PyObject* *replstr, *Py_ssize_t* maxcount)

Return value: New reference. Part of the Stable ABI. Replace at most *maxcount* occurrences of *substr* in *unicode* with *replstr* and return the resulting Unicode object. *maxcount* == `-1` means replace all occurrences.

int **PyUnicode_Compare**(*PyObject* *left, *PyObject* *right)

Part of the Stable ABI. 두 문자열을 비교하고 각각 작음, 같음, 큼에 대해 `-1`, `0`, `1`을 반환합니다.

이 함수는 실패 시 `-1`을 반환하므로, 에러를 확인하기 위해 *PyErr_Occurred()*를 호출해야 합니다.

int **PyUnicode_CompareWithASCIIString**(*PyObject* *unicode, const char *string)

Part of the Stable ABI. Compare a Unicode object, *unicode*, with *string* and return `-1`, `0`, `1` for less than, equal, and greater than, respectively. It is best to pass only ASCII-encoded strings, but the function interprets the input string as ISO-8859-1 if it contains non-ASCII characters.

이 함수는 예외를 발생시키지 않습니다.

PyObject ***PyUnicode_RichCompare**(*PyObject* *left, *PyObject* *right, int op)

Return value: New reference. Part of the Stable ABI. 두 유니코드 문자열을 풍부한 비교(rich comparison)하고 다음 중 하나를 반환합니다:

- 예외가 발생하면 `NULL`
- *Py_True* or *Py_False* for successful comparisons
- *Py_NotImplemented* in case the type combination is unknown

Possible values for *op* are *Py_GT*, *Py_GE*, *Py_EQ*, *Py_NE*, *Py_LT*, and *Py_LE*.

PyObject ***PyUnicode_Format**(*PyObject* *format, *PyObject* *args)

Return value: New reference. Part of the Stable ABI. *format*과 *args*에서 새 문자열 객체를 반환합니다; 이것은 `format % args`와 유사합니다.

int **PyUnicode_Contains**(*PyObject* *unicode, *PyObject* *substr)

Part of the Stable ABI. Check whether *substr* is contained in *unicode* and return true or false accordingly.

substr has to coerce to a one element Unicode string. `-1` is returned if there was an error.

void **PyUnicode_InternInPlace**(*PyObject* **p_unicode)

Part of the Stable ABI. Intern the argument **p_unicode* in place. The argument must be the address of a pointer variable pointing to a Python Unicode string object. If there is an existing interned string that is the same as **p_unicode*, it sets **p_unicode* to it (releasing the reference to the old string object and creating a new *strong reference* to the interned string object), otherwise it leaves **p_unicode* alone and interns it.

(Clarification: even though there is a lot of talk about references, think of this function as reference-neutral. You must own the object you pass in; after the call you no longer own the passed-in reference, but you newly own the result.)

This function never raises an exception. On error, it leaves its argument unchanged without interning it.

Instances of subclasses of `str` may not be interned, that is, *PyUnicode_CheckExact*(**p_unicode*) must be true. If it is not, then – as with any other error – the argument is left unchanged.

Note that interned strings are not "immortal". You must keep a reference to the result to benefit from interning.

PyObject ***PyUnicode_InternFromString**(const char *str)

Return value: New reference. Part of the Stable ABI. A combination of *PyUnicode_FromString()* and *PyUnicode_InternInPlace()*, meant for statically allocated strings.

Return a new ("owned") reference to either a new Unicode string object that has been interned, or an earlier interned string object with the same value.

Python may keep a reference to the result, or prevent it from being garbage-collected promptly. For interning an unbounded number of different strings, such as ones coming from user input, prefer calling *PyUnicode_FromString()* and *PyUnicode_InternInPlace()* directly.

8.3.4 튜플 객체

type **PyTupleObject**

이 *PyObject*의 서브 형은 파이썬 튜플 객체를 나타냅니다.

PyTypeObject **PyTuple_Type**

Part of the Stable ABI. 이 *PyTypeObject* 인스턴스는 파이썬 튜플 형을 나타냅니다. 파이썬 계층의 `tuple`과 같은 객체입니다.

int **PyTuple_Check**(*PyObject* *p)

*p*가 튜플 객체이거나 튜플 형의 서브 형의 인스턴스면 참을 돌려줍니다. 이 함수는 항상 성공합니다.

int **PyTuple_CheckExact**(*PyObject* *p)

*p*가 튜플 객체이지만, 튜플 형의 서브 형의 인스턴스는 아니면 참을 돌려줍니다. 이 함수는 항상 성공합니다.

PyObject ***PyTuple_New**(*Py_ssize_t* len)

Return value: New reference. Part of the Stable ABI. Return a new tuple object of size *len*, or `NULL` with an exception set on failure.

PyObject ***PyTuple_Pack**(*Py_ssize_t* n, ...)

Return value: New reference. Part of the Stable ABI. Return a new tuple object of size *n*, or `NULL` with an exception set on failure. The tuple values are initialized to the subsequent *n* C arguments pointing to Python objects. `PyTuple_Pack(2, a, b)` is equivalent to `Py_BuildValue("(OO)", a, b)`.

Py_ssize_t **PyTuple_Size**(*PyObject* *p)

Part of the Stable ABI. Take a pointer to a tuple object, and return the size of that tuple. On error, return `-1` and with an exception set.

Py_ssize_t **PyTuple_GET_SIZE**(*PyObject* *p)

Like *PyTuple_Size()*, but without error checking.

PyObject ***PyTuple_GetItem**(*PyObject* *p, *Py_ssize_t* pos)

Return value: Borrowed reference. Part of the Stable ABI. Return the object at position *pos* in the tuple pointed to by *p*. If *pos* is negative or out of bounds, return `NULL` and set an `IndexError` exception.

The returned reference is borrowed from the tuple *p* (that is: it is only valid as long as you hold a reference to *p*). To get a *strong reference*, use *Py_NewRef(PyTuple_GetItem(...))* or *PySequence_GetItem()*.

PyObject ***PyTuple_GET_ITEM**(*PyObject* *p, *Py_ssize_t* pos)

Return value: Borrowed reference. *PyTuple_GetItem()*와 비슷하지만, 인자를 확인하지 않습니다.

PyObject ***PyTuple_GetSlice**(*PyObject* *p, *Py_ssize_t* low, *Py_ssize_t* high)

Return value: New reference. Part of the Stable ABI. Return the slice of the tuple pointed to by *p* between *low* and *high*, or `NULL` with an exception set on failure.

This is the equivalent of the Python expression `p[low:high]`. Indexing from the end of the tuple is not supported.

int **PyTuple_SetItem**(*PyObject* *p, *Py_ssize_t* pos, *PyObject* *o)

Part of the Stable ABI. *p*가 가리키는 튜플의 *pos* 위치에 객체 *o*에 대한 참조를 삽입합니다. 성공하면 `0`을 반환합니다. *pos*가 범위를 벗어나면, `-1`을 반환하고 `IndexError` 예외를 설정합니다.

> **참고**
>
> 이 함수는 *o*에 대한 참조를 "훔치고" 영향을 받는 위치에서 튜플에 이미 있는 항목에 대한 참조를 버립니다(discard).

void **PyTuple_SET_ITEM**(*PyObject* *p, *Py_ssize_t* pos, *PyObject* *o)

*PyTuple_SetItem()*과 비슷하지만, 에러 검사는 하지 않으며 새로운 튜플을 채울 때*만* 사용해야 합니다.

> **참고**
>
> This function "steals" a reference to *o*, and, unlike *PyTuple_SetItem()*, does *not* discard a reference to any item that is being replaced; any reference in the tuple at position *pos* will be leaked.

int **_PyTuple_Resize**(*PyObject* **p, *Py_ssize_t* newsize)

튜플 크기를 조정하는 데 사용할 수 있습니다. *newsize*는 튜플의 새로운 길이가 됩니다. 튜플은 불변이라고 여겨지므로, 객체에 대해 참조가 하나만 있을 때만 사용해야 합니다. 튜플이 코드의 다른 부분에 이미 알려졌으면 이것을 사용하지 마십시오. 튜플은 항상 끝에서 커지거나 줄어듭니다. 이것을 오래된 튜플을 파괴하고 새 튜플을 만드는 것으로 생각하십시오, 단지 더 효율적일 뿐입니다. 성공하면 `0`을 반환합니다. 클라이언트 코드는, `*p`의 결괏값이 이 함수를 호출하기 전과 같다고 가정해서는 안 됩니다. `*p`가 참조하는 객체가 바뀌면 원래 `*p`는 파괴됩니다. 실패하면, `-1`을 반환하고, `*p`를 `NULL`로 설정하고, `MemoryError` 나 `SystemError`를 발생시킵니다.

8.3.5 구조체 시퀀스 객체

구조체 시퀀스(struct sequence) 객체는 `namedtuple()` 객체의 C 등가물입니다, 즉 어트리뷰트를 통해 항목에 액세스할 수 있는 시퀀스입니다. 구조체 시퀀스를 만들려면, 먼저 특정 구조체 시퀀스 형을 만들어야 합니다.

PyTypeObject ***PyStructSequence_NewType**(*PyStructSequence_Desc* *desc)

Return value: New reference. Part of the Stable ABI. 아래에 설명된 *desc*의 데이터로 새로운 구조체 시퀀스 형을 만듭니다. 결과 형의 인스턴스는 *PyStructSequence_New()*로 만들 수 있습니다.

Return `NULL` with an exception set on failure.

void **PyStructSequence_InitType**(*PyTypeObject* *type, *PyStructSequence_Desc* *desc)

*desc*로 구조체 시퀀스 형 *type*을 재자리에서 초기화합니다.

int **PyStructSequence_InitType2**(*PyTypeObject* *type, *PyStructSequence_Desc* *desc)

Like *PyStructSequence_InitType()*, but returns `0` on success and `-1` with an exception set on failure.

Added in version 3.4.

type **PyStructSequence_Desc**

Part of the Stable ABI (*including all members*). 만들 구조체 시퀀스 형의 메타 정보를 포함합니다.

const char ***name**

Fully qualified name of the type; null-terminated UTF-8 encoded. The name must contain the module name.

const char ***doc**

Pointer to docstring for the type or `NULL` to omit.

PyStructSequence_Field ***fields**

Pointer to `NULL`-terminated array with field names of the new type.

int **n_in_sequence**

Number of fields visible to the Python side (if used as tuple).

type **PyStructSequence_Field**

Part of the Stable ABI (*including all members*). Describes a field of a struct sequence. As a struct sequence is modeled as a tuple, all fields are typed as `PyObject*`. The index in the `fields` array of the `PyStructSequence_Desc` determines which field of the struct sequence is described.

const char ***name**

Name for the field or `NULL` to end the list of named fields, set to `PyStructSequence_UnnamedField` to leave unnamed.

const char ***doc**

Field docstring or `NULL` to omit.

const char *const **PyStructSequence_UnnamedField**

Part of the Stable ABI *since version 3.11.* 이름 없는 상태로 남겨두기 위한 필드 이름의 특수 값.

버전 3.9에서 변경: 형이 `char *`에서 변경되었습니다.

PyObject ***PyStructSequence_New**(*PyTypeObject* *type)

Return value: New reference. Part of the Stable ABI. `PyStructSequence_NewType()`으로 만든 *type*의 인스턴스를 만듭니다.

Return `NULL` with an exception set on failure.

PyObject ***PyStructSequence_GetItem**(*PyObject* *p, *Py_ssize_t* pos)

Return value: Borrowed reference. Part of the Stable ABI. *p*가 가리키는 구조체 시퀀스의 위치 *pos*에 있는 객체를 돌려줍니다. 범위 검사가 수행되지 않습니다.

PyObject ***PyStructSequence_GET_ITEM**(*PyObject* *p, *Py_ssize_t* pos)

Return value: Borrowed reference. `PyStructSequence_GetItem()`과 동등한 매크로.

void **PyStructSequence_SetItem**(*PyObject* *p, *Py_ssize_t* pos, *PyObject* *o)

Part of the Stable ABI. 구조체 시퀀스 *p*의 인덱스 *pos*에 있는 필드를 값 *o*로 설정합니다. `PyTuple_SET_ITEM()`과 마찬가지로, 이것은 새로운 인스턴스를 채울 때만 사용해야 합니다.

> **참고**
>
> 이 함수는 *o*에 대한 참조를 "훔칩니다".

void **PyStructSequence_SET_ITEM**(*PyObject* *p, *Py_ssize_t* *pos, *PyObject* *o)

Similar to `PyStructSequence_SetItem()`, but implemented as a static inlined function.

> **참고**
>
> 이 함수는 *o*에 대한 참조를 "훔칩니다".

8.3.6 리스트 객체

type **PyListObject**

이 *PyObject*의 서브 형은 파이썬 리스트 객체를 나타냅니다.

PyTypeObject **PyList_Type**

Part of the Stable ABI. 이 *PyTypeObject* 인스턴스는 파이썬 리스트 형을 나타냅니다. 이것은 파이썬 계층의 `list`와 같은 객체입니다.

int **PyList_Check**(*PyObject* *p)

*p*가 리스트 객체나 리스트 형의 서브 형 인스턴스면 참을 반환합니다. 이 함수는 항상 성공합니다.

int **PyList_CheckExact**(*PyObject* *p)

*p*가 리스트 객체이지만 리스트 형의 서브 형의 인스턴스가 아니면 참을 반환합니다. 이 함수는 항상 성공합니다.

PyObject ***PyList_New**(*Py_ssize_t* len)

Return value: New reference. Part of the Stable ABI. 성공하면 길이 *len* 인 새 리스트를, 실패하면 `NULL`을 반환합니다.

> **참고**
>
> *len*이 0보다 크면, 반환된 리스트 객체의 항목은 `NULL`로 설정됩니다. 따라서 모든 항목을 *PyList_SetItem()*로 실제 객체로 설정하기 전에 *PySequence_SetItem()*와 같은 추상 API 함수를 사용하거나 파이썬 코드에 객체를 노출할 수 없습니다.

Py_ssize_t **PyList_Size**(*PyObject* *list)

Part of the Stable ABI. *list*에서 리스트 객체의 길이를 반환합니다; 이는 리스트 객체에 대한 `len(list)`와 동등합니다.

Py_ssize_t **PyList_GET_SIZE**(*PyObject* *list)

Similar to *PyList_Size()*, but without error checking.

PyObject ***PyList_GetItem**(*PyObject* *list, *Py_ssize_t* index)

Return value: Borrowed reference. Part of the Stable ABI. *list*가 가리키는 리스트에서 *index* 위치의 객체를 반환합니다. 위치는 음수가 아니어야 합니다; 리스트의 끝에서부터의 인덱싱은 지원되지 않습니다. *index*가 범위를 벗어나면 (<0 또는 >=len(list)), `NULL`을 반환하고 `IndexError` 예외를 설정합니다.

PyObject ***PyList_GET_ITEM**(*PyObject* *list, *Py_ssize_t* i)

Return value: Borrowed reference. Similar to *PyList_GetItem()*, but without error checking.

int **PyList_SetItem**(*PyObject* *list, *Py_ssize_t* index, *PyObject* *item)

Part of the Stable ABI. 리스트의 인덱스 *index*에 있는 항목을 *item*으로 설정합니다. 성공하면 `0`을 반환합니다. *index*가 범위를 벗어나면, `-1`을 반환하고 `IndexError` 예외를 설정합니다.

> **참고**
>
> 이 함수는 *item*에 대한 참조를 "훔치고" 영향을 받는 위치의 리스트에 이미 있는 항목에 대한 참조를 버립니다.

void **PyList_SET_ITEM**(*PyObject* *list, *Py_ssize_t* i, *PyObject* *o)

에러 검사 없는 *PyList_SetItem()*의 매크로 형식. 일반적으로 이전 내용이 없는 새 리스트를 채우는 데 사용됩니다.

> **참고**
>
> 이 매크로는 *item*에 대한 참조를 "훔치고", `PyList_SetItem()`과는 달리 대체되는 항목에 대한 참조를 버리지 않습니다; *list*의 *i* 위치에 있는 참조는 누수를 일으킵니다.

int **PyList_Insert**(*PyObject* *list, *Py_ssize_t* index, *PyObject* *item)

Part of the Stable ABI. 항목 *item*을 리스트 *list*의 인덱스 *index* 앞에 삽입합니다. 성공하면 `0`을 반환합니다; 실패하면 `-1`을 반환하고 예외를 설정합니다. `list.insert(index, item)`에 해당합니다.

int **PyList_Append**(*PyObject* *list, *PyObject* *item)

Part of the Stable ABI. 리스트 *list*의 끝에 객체 *item*을 추가합니다. 성공하면 `0`을 반환합니다; 실패하면 `-1`을 반환하고 예외를 설정합니다. `list.append(item)`에 해당합니다.

PyObject ***PyList_GetSlice**(*PyObject* *list, *Py_ssize_t* low, *Py_ssize_t* high)

Return value: New reference. Part of the Stable ABI. *list*에서 *low*와 *high* 사이에있는 객체들을 포함하는 리스트를 반환합니다. 실패하면 `NULL`을 반환하고 예외를 설정합니다. `list[low:high]`에 해당합니다. 리스트 끝에서부터의 인덱싱은 지원되지 않습니다.

int **PyList_SetSlice**(*PyObject* *list, *Py_ssize_t* low, *Py_ssize_t* high, *PyObject* *itemlist)

Part of the Stable ABI. *low*와 *high* 사이의 *list* 슬라이스를 *itemlist*의 내용으로 설정합니다. `list[low:high] = itemlist`에 해당합니다. *itemlist*는 `NULL`일 수 있는데, 빈 리스트의 대입을 나타냅니다 (슬라이스 삭제). 성공하면 `0`을, 실패하면 `-1`을 반환합니다. 리스트 끝에서부터의 인덱싱은 지원되지 않습니다.

int **PyList_Sort**(*PyObject* *list)

Part of the Stable ABI. *list* 항목을 제자리에서 정렬합니다. 성공하면 `0`을, 실패하면 `-1`을 반환합니다. 이것은 `list.sort()`와 동등합니다.

int **PyList_Reverse**(*PyObject* *list)

Part of the Stable ABI. *list*의 항목을 제자리에서 뒤집습니다. 성공하면 `0`을, 실패하면 `-1`을 반환합니다. 이것은 `list.reverse()`와 동등합니다.

PyObject ***PyList_AsTuple**(*PyObject* *list)

Return value: New reference. Part of the Stable ABI. *list*의 내용을 포함하는 새 튜플 객체를 반환합니다; `tuple(list)`와 동등합니다.

8.4 컨테이너 객체

8.4.1 딕셔너리 객체

type **PyDictObject**

이 `PyObject`의 서브 형은 파이썬 딕셔너리 객체를 나타냅니다.

PyTypeObject **PyDict_Type**

Part of the Stable ABI. 이 `PyTypeObject` 인스턴스는 파이썬 딕셔너리 형을 나타냅니다. 이것은 파이썬 계층의 `dict`와 같은 객체입니다.

int **PyDict_Check**(*PyObject* *p)

*p*가 dict 객체이거나 dict 형의 서브 형의 인스턴스면 참을 반환합니다. 이 함수는 항상 성공합니다.

int **PyDict_CheckExact**(*PyObject* *p)

*p*가 dict 객체이지만, dict 형의 서브 형의 인스턴스는 아니면 참을 반환합니다. 이 함수는 항상 성공합니다.

PyObject ***PyDict_New**()

Return value: New reference. Part of the Stable ABI. 새로운 빈 딕셔너리를 반환하거나, 실패하면 `NULL`을 반환합니다.

PyObject ***PyDictProxy_New**(*PyObject* *mapping)

Return value: New reference. Part of the Stable ABI. 읽기 전용 동작을 강제하는 매핑을 위한 `types.MappingProxyType` 객체를 반환합니다. 이것은 일반적으로 비 동적 클래스 형을 위한 딕셔너리의 수정을 방지하기 위해 뷰를 만드는 데 사용됩니다.

void **PyDict_Clear**(*PyObject* *p)

Part of the Stable ABI. 기존 딕셔너리의 모든 키-값 쌍을 비웁니다.

int **PyDict_Contains**(*PyObject* *p, *PyObject* *key)

Part of the Stable ABI. 딕셔너리 *p*에 *key*가 포함되어 있는지 확인합니다. *p*의 항목이 *key*와 일치하면 `1`을 반환하고, 그렇지 않으면 `0`을 반환합니다. 에러면 `-1`을 반환합니다. 이는 파이썬 표현식 `key in p`와 동등합니다.

PyObject ***PyDict_Copy**(*PyObject* *p)

Return value: New reference. Part of the Stable ABI. *p*와 같은 키-값 쌍을 포함하는 새 딕셔너리를 반환합니다.

int **PyDict_SetItem**(*PyObject* *p, *PyObject* *key, *PyObject* *val)

Part of the Stable ABI. 딕셔너리 *p*에 *val*을 *key* 키로 삽입합니다. *key*는 해시 가능해야 합니다. 그렇지 않으면 `TypeError`가 발생합니다. 성공하면 `0`을, 실패하면 `-1`을 반환합니다. 이 함수는 *val*에 대한 참조를 훔치지 않습니다.

int **PyDict_SetItemString**(*PyObject* *p, const char *key, *PyObject* *val)

Part of the Stable ABI. This is the same as *PyDict_SetItem()*, but *key* is specified as a `const char*` UTF-8 encoded bytes string, rather than a *PyObject**.

int **PyDict_DelItem**(*PyObject* *p, *PyObject* *key)

Part of the Stable ABI. Remove the entry in dictionary *p* with key *key*. *key* must be *hashable*; if it isn' t, `TypeError` is raised. If *key* is not in the dictionary, `KeyError` is raised. Return `0` on success or `-1` on failure.

int **PyDict_DelItemString**(*PyObject* *p, const char *key)

Part of the Stable ABI. This is the same as *PyDict_DelItem()*, but *key* is specified as a `const char*` UTF-8 encoded bytes string, rather than a *PyObject**.

PyObject ***PyDict_GetItem**(*PyObject* *p, *PyObject* *key)

Return value: Borrowed reference. Part of the Stable ABI. 딕셔너리 *p*에서 키가 *key*인 객체를 반환합니다. *key* 키가 없으면 예외를 설정하지 않고 `NULL`을 반환합니다.

> **참고**
>
> Exceptions that occur while this calls `__hash__()` and `__eq__()` methods are silently ignored. Prefer the *PyDict_GetItemWithError()* function instead.

버전 3.10에서 변경: Calling this API without *GIL* held had been allowed for historical reason. It is no longer allowed.

PyObject ***PyDict_GetItemWithError**(*PyObject* *p, *PyObject* *key)

Return value: Borrowed reference. Part of the Stable ABI. 예외를 억제하지 않는 *PyDict_GetItem()*의 변형입니다. 예외가 발생하면 예외를 **설정하고** `NULL`을 반환합니다. 키가 없으면 예외를 설정하지 **않고** `NULL`을 반환합니다.

PyObject ***PyDict_GetItemString**(*PyObject* *p, const char *key)

Return value: Borrowed reference. Part of the Stable ABI. This is the same as *PyDict_GetItem()*, but *key* is specified as a `const char*` UTF-8 encoded bytes string, rather than a *PyObject**.

> **참고**
>
> Exceptions that occur while this calls `__hash__()` and `__eq__()` methods or while creating the temporary `str` object are silently ignored. Prefer using the *PyDict_GetItemWithError()* function with your own *PyUnicode_FromString()* *key* instead.

PyObject ***PyDict_SetDefault**(*PyObject* *p, *PyObject* *key, *PyObject* *defaultobj)

Return value: Borrowed reference. 이것은 파이썬 수준의 `dict.setdefault()`와 같습니다. 존재하면, 딕셔너리 *p*에서 *key*에 해당하는 값을 반환합니다. 키가 dict에 없으면, 값 *defaultobj*로 삽입되고, *defaultobj*가 반환됩니다. 이 함수는 *key*의 해시 함수를 조회 및 삽입을 위해 독립적으로 평가하는 대신 한 번만 평가합니다.

Added in version 3.4.

PyObject ***PyDict_Items**(*PyObject* *p)

Return value: New reference. Part of the Stable ABI. 딕셔너리의 모든 항목을 포함하는 *PyListObject*를 반환합니다.

PyObject ***PyDict_Keys**(*PyObject* *p)

Return value: New reference. Part of the Stable ABI. 딕셔너리의 모든 키를 포함하는 *PyListObject*를 반환합니다.

PyObject ***PyDict_Values**(*PyObject* *p)

Return value: New reference. Part of the Stable ABI. 딕셔너리 *p*의 모든 값을 포함하는 *PyListObject*를 반환합니다.

Py_ssize_t **PyDict_Size**(*PyObject* *p)

Part of the Stable ABI. 딕셔너리에 있는 항목의 수를 반환합니다. 이는 딕셔너리에 대한 `len(p)`와 동등합니다.

int **PyDict_Next**(*PyObject* *p, *Py_ssize_t* *ppos, *PyObject* **pkey, *PyObject* **pvalue)

Part of the Stable ABI. Iterate over all key-value pairs in the dictionary *p*. The *Py_ssize_t* referred to by *ppos* must be initialized to `0` prior to the first call to this function to start the iteration; the function returns true for each pair in the dictionary, and false once all pairs have been reported. The parameters *pkey* and *pvalue* should either point to *PyObject** variables that will be filled in with each key and value, respectively, or may be `NULL`. Any references returned through them are borrowed. *ppos* should not be altered during iteration. Its value represents offsets within the internal dictionary structure, and since the structure is sparse, the offsets are not consecutive.

예를 들면:

```
PyObject *key, *value;
Py_ssize_t pos = 0;

while (PyDict_Next(self->dict, &pos, &key, &value)) {
    /* do something interesting with the values... */
    ...
}
```

딕셔너리 *p*는 이터레이션 중에 변경해서는 안 됩니다. 딕셔너리를 이터레이트 할 때 값을 변경하는 것은 안전하지만, 키 집합이 변경되지 않는 한만 그렇습니다. 예를 들면:

```
PyObject *key, *value;
Py_ssize_t pos = 0;

while (PyDict_Next(self->dict, &pos, &key, &value)) {
    long i = PyLong_AsLong(value);
    if (i == -1 && PyErr_Occurred()) {
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
        return -1;
    }
    PyObject *o = PyLong_FromLong(i + 1);
    if (o == NULL)
        return -1;
    if (PyDict_SetItem(self->dict, key, o) < 0) {
        Py_DECREF(o);
        return -1;
    }
    Py_DECREF(o);
}
```

int **PyDict_Merge**(*PyObject* *a, *PyObject* *b, int override)

Part of the Stable ABI. 매핑 객체 *b*를 이터레이트 하면서, 키-값 쌍을 딕셔너리 *a*에 추가합니다. *b*는 딕셔너리거나 `PyMapping_Keys()`와 `PyObject_GetItem()`를 지원하는 모든 객체일 수 있습니다. *override*가 참이면, *a*에 있는 기존 쌍이 *b*에서 일치하는 키가 있으면 교체되고, 그렇지 않으면 *a*와 일치하는 키가 없을 때만 쌍이 추가됩니다. 성공하면 `0`을 반환하고, 예외가 발생하면 `-1`을 반환합니다.

int **PyDict_Update**(*PyObject* *a, *PyObject* *b)

Part of the Stable ABI. 이는 C에서 `PyDict_Merge(a, b, 1)`와 같고, 두 번째 인자에 "keys" 어트리뷰트가 없을 때 `PyDict_Update()`가 키-값 쌍의 시퀀스에 대해 이터레이트 하지 않는다는 점만 제외하면, 파이썬에서 `a.update(b)`와 유사합니다. 성공하면 `0`을 반환하고, 예외가 발생하면 `-1`을 반환합니다.

int **PyDict_MergeFromSeq2**(*PyObject* *a, *PyObject* *seq2, int override)

Part of the Stable ABI. *seq2*의 키-값 쌍으로 딕셔너리 *a*를 갱신하거나 병합합니다. *seq2*는 키-값 쌍으로 간주하는 길이 2의 이터러블 객체를 생성하는 이터러블 객체여야 합니다. 중복 키가 있으면, *override*가 참이면 마지막이 승리하고, 그렇지 않으면 첫 번째가 승리합니다. 성공 시 `0`을 반환하고, 예외가 발생하면 `-1`을 반환합니다. 동등한 파이썬은 이렇습니다(반환 값 제외)

```
def PyDict_MergeFromSeq2(a, seq2, override):
    for key, value in seq2:
        if override or key not in a:
            a[key] = value
```

int **PyDict_AddWatcher**(*PyDict_WatchCallback* callback)

Register *callback* as a dictionary watcher. Return a non-negative integer id which must be passed to future calls to `PyDict_Watch()`. In case of error (e.g. no more watcher IDs available), return `-1` and set an exception.

Added in version 3.12.

int **PyDict_ClearWatcher**(int watcher_id)

Clear watcher identified by *watcher_id* previously returned from `PyDict_AddWatcher()`. Return `0` on success, `-1` on error (e.g. if the given *watcher_id* was never registered.)

Added in version 3.12.

int **PyDict_Watch**(int watcher_id, *PyObject* *dict)

Mark dictionary *dict* as watched. The callback granted *watcher_id* by `PyDict_AddWatcher()` will be called when *dict* is modified or deallocated. Return `0` on success or `-1` on error.

Added in version 3.12.

int **PyDict_Unwatch**(int watcher_id, *PyObject* *dict)

Mark dictionary *dict* as no longer watched. The callback granted *watcher_id* by `PyDict_AddWatcher()` will no longer be called when *dict* is modified or deallocated. The dict must previously have been watched by this watcher. Return `0` on success or `-1` on error.

Added in version 3.12.

type **PyDict_WatchEvent**

Enumeration of possible dictionary watcher events: `PyDict_EVENT_ADDED`, `PyDict_EVENT_MODIFIED`, `PyDict_EVENT_DELETED`, `PyDict_EVENT_CLONED`, `PyDict_EVENT_CLEARED`, or `PyDict_EVENT_DEALLOCATED`.

Added in version 3.12.

typedef int (***PyDict_WatchCallback**)(*PyDict_WatchEvent* event, *PyObject* *dict, *PyObject* *key, *PyObject* *new_value)

Type of a dict watcher callback function.

If *event* is `PyDict_EVENT_CLEARED` or `PyDict_EVENT_DEALLOCATED`, both *key* and *new_value* will be `NULL`. If *event* is `PyDict_EVENT_ADDED` or `PyDict_EVENT_MODIFIED`, *new_value* will be the new value for *key*. If *event* is `PyDict_EVENT_DELETED`, *key* is being deleted from the dictionary and *new_value* will be `NULL`.

`PyDict_EVENT_CLONED` occurs when *dict* was previously empty and another dict is merged into it. To maintain efficiency of this operation, per-key `PyDict_EVENT_ADDED` events are not issued in this case; instead a single `PyDict_EVENT_CLONED` is issued, and *key* will be the source dictionary.

The callback may inspect but must not modify *dict*; doing so could have unpredictable effects, including infinite recursion. Do not trigger Python code execution in the callback, as it could modify the dict as a side effect.

If *event* is `PyDict_EVENT_DEALLOCATED`, taking a new reference in the callback to the about-to-be-destroyed dictionary will resurrect it and prevent it from being freed at this time. When the resurrected object is destroyed later, any watcher callbacks active at that time will be called again.

Callbacks occur before the notified modification to *dict* takes place, so the prior state of *dict* can be inspected.

If the callback sets an exception, it must return `-1`; this exception will be printed as an unraisable exception using *PyErr_WriteUnraisable()*. Otherwise it should return `0`.

There may already be a pending exception set on entry to the callback. In this case, the callback should return `0` with the same exception still set. This means the callback may not call any other API that can set an exception unless it saves and clears the exception state first, and restores it before returning.

Added in version 3.12.

8.4.2 집합 객체

This section details the public API for `set` and `frozenset` objects. Any functionality not listed below is best accessed using either the abstract object protocol (including *PyObject_CallMethod()*, *PyObject_RichCompareBool()*, *PyObject_Hash()*, *PyObject_Repr()*, *PyObject_IsTrue()*, *PyObject_Print()*, and *PyObject_GetIter()*) or the abstract number protocol (including *PyNumber_And()*, *PyNumber_Subtract()*, *PyNumber_Or()*, *PyNumber_Xor()*, *PyNumber_InPlaceAnd()*, *PyNumber_InPlaceSubtract()*, *PyNumber_InPlaceOr()*, and *PyNumber_InPlaceXor()*).

type **PySetObject**

This subtype of *PyObject* is used to hold the internal data for both `set` and `frozenset` objects. It is like a *PyDictObject* in that it is a fixed size for small sets (much like tuple storage) and will point to a separate, variable sized block of memory for medium and large sized sets (much like list storage). None of the fields of this structure should be considered public and all are subject to change. All access should be done through the documented API rather than by manipulating the values in the structure.

PyTypeObject **PySet_Type**

Part of the Stable ABI. 이것은 파이썬 `set` 형을 나타내는 *PyTypeObject*의 인스턴스입니다.

PyTypeObject **PyFrozenSet_Type**

Part of the Stable ABI. 이것은 파이썬 `frozenset` 형을 나타내는 *PyTypeObject*의 인스턴스입니다.

다음 형 검사 매크로는 모든 파이썬 객체에 대한 포인터에서 작동합니다. 마찬가지로, 생성자 함수는 모든 이터러블 파이썬 객체에서 작동합니다.

int **PySet_Check**(*PyObject* *p)

*p*가 `set` 객체나 서브 형의 인스턴스면 참을 반환합니다. 이 함수는 항상 성공합니다.

int **PyFrozenSet_Check**(*PyObject* *p)

*p*가 `frozenset` 객체나 서브 형의 인스턴스면 참을 반환합니다. 이 함수는 항상 성공합니다.

int **PyAnySet_Check**(*PyObject* *p)

*p*가 `set` 객체, `frozenset` 객체 또는 서브 형의 인스턴스면 참을 반환합니다. 이 함수는 항상 성공합니다.

int **PySet_CheckExact**(*PyObject* *p)

Return true if *p* is a `set` object but not an instance of a subtype. This function always succeeds.

Added in version 3.10.

int **PyAnySet_CheckExact**(*PyObject* *p)

*p*가 `set` 객체나 `frozenset` 객체이지만, 서브 형의 인스턴스는 아니면 참을 반환합니다. 이 함수는 항상 성공합니다.

int **PyFrozenSet_CheckExact**(*PyObject* *p)

*p*가 `frozenset` 객체이지만, 서브 형의 인스턴스는 아니면 참을 반환합니다. 이 함수는 항상 성공합니다.

PyObject ***PySet_New**(*PyObject* *iterable)

Return value: New reference. Part of the Stable ABI. *iterable*에 의해 반환된 객체를 포함하는 새로운 `set`을 반환합니다. *iterable*은 새로운 빈 집합을 만들기 위해 `NULL` 일 수 있습니다. 성공하면 새 집합을, 실패하면 `NULL`을 반환합니다. *iterable*이 실제로 이터러블이 아니면 `TypeError`를 발생시킵니다. 생성자는 집합을 복사할 때도 유용합니다 (`c=set(s)`).

PyObject ***PyFrozenSet_New**(*PyObject* *iterable)

Return value: New reference. Part of the Stable ABI. *iterable*에 의해 반환된 객체를 포함한 새로운 `frozenset`을 반환합니다. *iterable*은 새로운 빈 frozenset을 만들기 위해 `NULL` 일 수 있습니다. 성공하면 새 집합을, 실패하면 `NULL`을 반환합니다. *iterable*이 실제로 이터러블이 아니면 `TypeError`를 발생시킵니다.

`set` 이나 `frozenset`의 인스턴스 또는 그들의 서브 형의 인스턴스에 대해 다음 함수와 매크로를 사용할 수 있습니다.

Py_ssize_t **PySet_Size**(*PyObject* *anyset)

Part of the Stable ABI. Return the length of a `set` or `frozenset` object. Equivalent to `len(anyset)`. Raises a `SystemError` if *anyset* is not a `set`, `frozenset`, or an instance of a subtype.

Py_ssize_t **PySet_GET_SIZE**(*PyObject* *anyset)

에러 검사 없는 *PySet_Size()*의 매크로 형식.

int **PySet_Contains**(*PyObject* *anyset, *PyObject* *key)

Part of the Stable ABI. Return `1` if found, `0` if not found, and `-1` if an error is encountered. Unlike the Python `__contains__()` method, this function does not automatically convert unhashable sets into temporary frozensets. Raise a `TypeError` if the *key* is unhashable. Raise `SystemError` if *anyset* is not a `set`, `frozenset`, or an instance of a subtype.

int **PySet_Add**(*PyObject* *set, *PyObject* *key)

Part of the Stable ABI. Add *key* to a `set` instance. Also works with `frozenset` instances (like *PyTuple_SetItem()* it can be used to fill in the values of brand new frozensets before they are exposed to other code). Return `0` on success or `-1` on failure. Raise a `TypeError` if the *key* is unhashable. Raise a `MemoryError` if there is no room to grow. Raise a `SystemError` if *set* is not an instance of `set` or its subtype.

다음 함수는 `set` 이나 그것의 서브 형의 인스턴스에는 사용할 수 있지만, `frozenset` 이나 그 서브 형의 인스턴스에는 사용할 수 없습니다.

int **PySet_Discard**(*PyObject* *set, *PyObject* *key)

Part of the Stable ABI. Return `1` if found and removed, `0` if not found (no action taken), and `-1` if an error is encountered. Does not raise `KeyError` for missing keys. Raise a `TypeError` if the *key* is unhashable. Unlike the Python `discard()` method, this function does not automatically convert unhashable sets into temporary frozensets. Raise `SystemError` if *set* is not an instance of `set` or its subtype.

PyObject ***PySet_Pop**(*PyObject* *set)

Return value: New reference. Part of the Stable ABI. *set*에 들어있는 임의의 객체에 대한 새 참조를 반환하고, *set*에서 객체를 제거합니다. 실패하면 `NULL`을 반환합니다. 집합이 비어 있으면, `KeyError`를 발생시킵니다. *set*이 `set` 이나 그 서브 형의 인스턴스가 아니면 `SystemError`를 발생시킵니다.

int **PySet_Clear**(*PyObject* *set)

Part of the Stable ABI. Empty an existing set of all elements. Return `0` on success. Return `-1` and raise `SystemError` if *set* is not an instance of `set` or its subtype.

8.5 함수 객체

8.5.1 함수 객체

파이썬 함수와 관련된 몇 가지 함수가 있습니다.

type **PyFunctionObject**

함수에 사용되는 C 구조체.

PyTypeObject **PyFunction_Type**

이것은 *PyTypeObject*의 인스턴스이며 파이썬 함수 형을 나타냅니다. 파이썬 프로그래머에게 `types.FunctionType`으로 노출됩니다.

int **PyFunction_Check**(*PyObject* *o)

*o*가 함수 객체(*PyFunction_Type* 형)면 참을 반환합니다. 매개 변수는 `NULL`이 아니어야 합니다. 이 함수는 항상 성공합니다.

PyObject ***PyFunction_New**(*PyObject* *code, *PyObject* *globals)

Return value: New reference. 코드 객체 *code*와 연관된 새 함수 객체를 반환합니다. *globals*는 함수에서 액세스할 수 있는 전역 변수가 있는 딕셔너리이어야 합니다.

The function' s docstring and name are retrieved from the code object. `__module__` is retrieved from *globals*. The argument defaults, annotations and closure are set to `NULL`. `__qualname__` is set to the same value as the code object' s `co_qualname` field.

PyObject ***PyFunction_NewWithQualName**(*PyObject* *code, *PyObject* *globals, *PyObject* *qualname)

Return value: New reference. As *PyFunction_New()*, but also allows setting the function object' s `__qualname__` attribute. *qualname* should be a unicode object or `NULL`; if `NULL`, the `__qualname__` attribute is set to the same value as the code object' s `co_qualname` field.

Added in version 3.3.

PyObject ***PyFunction_GetCode**(*PyObject* *op)

Return value: Borrowed reference. 함수 객체 *op*와 연관된 코드 객체를 반환합니다.

PyObject ***PyFunction_GetGlobals**(*PyObject* *op)

Return value: Borrowed reference. 함수 객체 *op*와 연관된 전역 딕셔너리를 반환합니다.

PyObject ***PyFunction_GetModule**(*PyObject* *op)

Return value: Borrowed reference. Return a *borrowed reference* to the `__module__` attribute of the function object *op*. It can be *NULL*.

This is normally a `string` containing the module name, but can be set to any other object by Python code.

PyObject ***PyFunction_GetDefaults**(*PyObject* *op)

Return value: Borrowed reference. 함수 객체 *op*의 인자 기본값을 반환합니다. 이는 인자의 튜플이나 `NULL`일 수 있습니다.

int **PyFunction_SetDefaults**(*PyObject* *op, *PyObject* *defaults)

함수 객체 *op*의 인자 기본값을 설정합니다. *defaults*는 `Py_None` 이나 튜플이어야 합니다.

실패하면 `SystemError`를 발생시키고 `-1`을 반환합니다.

void **PyFunction_SetVectorcall**(*PyFunctionObject* *func, *vectorcallfunc* vectorcall)

Set the vectorcall field of a given function object *func*.

Warning: extensions using this API must preserve the behavior of the unaltered (default) vectorcall function!

Added in version 3.12.

PyObject ***PyFunction_GetClosure**(*PyObject* *op)

Return value: Borrowed reference. 함수 객체 *op*와 연관된 클로저를 반환합니다. 이것은 `NULL` 이나 셀 객체의 튜플일 수 있습니다.

int **PyFunction_SetClosure**(*PyObject* *op, *PyObject* *closure)

함수 객체 *op*와 연관된 클로저를 설정합니다. *closure*는 `Py_None` 이나 셀 객체의 튜플이어야 합니다.

실패하면 `SystemError`를 발생시키고 `-1`을 반환합니다.

PyObject ***PyFunction_GetAnnotations**(*PyObject* *op)

Return value: Borrowed reference. 함수 객체 *op*의 어노테이션을 반환합니다. 이것은 가변 딕셔너리나 `NULL` 일 수 있습니다.

int **PyFunction_SetAnnotations**(*PyObject* *op, *PyObject* *annotations)

함수 객체 *op*의 어노테이션을 설정합니다. *annotations*은 딕셔너리나 `Py_None` 이어야 합니다.

실패하면 `SystemError`를 발생시키고 `-1`을 반환합니다.

int **PyFunction_AddWatcher**(*PyFunction_WatchCallback* callback)

Register *callback* as a function watcher for the current interpreter. Return an ID which may be passed to *PyFunction_ClearWatcher()*. In case of error (e.g. no more watcher IDs available), return `-1` and set an exception.

Added in version 3.12.

int **PyFunction_ClearWatcher**(int watcher_id)

Clear watcher identified by *watcher_id* previously returned from *PyFunction_AddWatcher()* for the current interpreter. Return `0` on success, or `-1` and set an exception on error (e.g. if the given *watcher_id* was never registered.)

Added in version 3.12.

type **PyFunction_WatchEvent**

Enumeration of possible function watcher events:

- `PyFunction_EVENT_CREATE`
- `PyFunction_EVENT_DESTROY`
- `PyFunction_EVENT_MODIFY_CODE`
- `PyFunction_EVENT_MODIFY_DEFAULTS`
- `PyFunction_EVENT_MODIFY_KWDEFAULTS`

Added in version 3.12.

typedef int (***PyFunction_WatchCallback**)(*PyFunction_WatchEvent* event, *PyFunctionObject* *func, *PyObject* *new_value)

Type of a function watcher callback function.

If *event* is `PyFunction_EVENT_CREATE` or `PyFunction_EVENT_DESTROY` then *new_value* will be `NULL`. Otherwise, *new_value* will hold a *borrowed reference* to the new value that is about to be stored in *func* for the attribute that is being modified.

The callback may inspect but must not modify *func*; doing so could have unpredictable effects, including infinite recursion.

If *event* is `PyFunction_EVENT_CREATE`, then the callback is invoked after *func* has been fully initialized. Otherwise, the callback is invoked before the modification to *func* takes place, so the prior state of *func* can be inspected. The runtime is permitted to optimize away the creation of function objects when possible. In such cases no event will be emitted. Although this creates the possibility of an observable difference of runtime behavior depending on optimization decisions, it does not change the semantics of the Python code being executed.

If *event* is `PyFunction_EVENT_DESTROY`, Taking a reference in the callback to the about-to-be-destroyed function will resurrect it, preventing it from being freed at this time. When the resurrected object is destroyed later, any watcher callbacks active at that time will be called again.

If the callback sets an exception, it must return `-1`; this exception will be printed as an unraisable exception using *PyErr_WriteUnraisable()*. Otherwise it should return `0`.

There may already be a pending exception set on entry to the callback. In this case, the callback should return `0` with the same exception still set. This means the callback may not call any other API that can set an exception unless it saves and clears the exception state first, and restores it before returning.

Added in version 3.12.

8.5.2 인스턴스 메서드 객체

An instance method is a wrapper for a *PyCFunction* and the new way to bind a *PyCFunction* to a class object. It replaces the former call `PyMethod_New(func, NULL, class)`.

PyTypeObject **PyInstanceMethod_Type**

이 *PyTypeObject* 인스턴스는 파이썬 인스턴스 메서드 형을 나타냅니다. 파이썬 프로그램에는 노출되지 않습니다.

int **PyInstanceMethod_Check**(*PyObject* *o)

*o*가 인스턴스 메서드 객체면 참을 반환합니다 (*PyInstanceMethod_Type* 형입니다). 매개 변수는 `NULL`이 아니어야 합니다. 이 함수는 항상 성공합니다.

PyObject ***PyInstanceMethod_New**(*PyObject* *func)

Return value: New reference. Return a new instance method object, with *func* being any callable object. *func* is the function that will be called when the instance method is called.

PyObject ***PyInstanceMethod_Function**(*PyObject* *im)

Return value: Borrowed reference. 인스턴스 메서드 *im*과 연관된 함수 객체를 반환합니다.

PyObject ***PyInstanceMethod_GET_FUNCTION**(*PyObject* *im)

Return value: Borrowed reference. 오류 검사를 피하는 *PyInstanceMethod_Function()*의 매크로 버전.

8.5.3 메서드 객체

메서드는 연결된(bound) 함수 객체입니다. 메서드는 항상 사용자 정의 클래스의 인스턴스에 연결됩니다. 연결되지 않은(unbound) 메서드(클래스 객체에 연결된 메서드)는 더는 사용할 수 없습니다.

PyTypeObject **PyMethod_Type**

이 *PyTypeObject* 인스턴스는 파이썬 메서드 형을 나타냅니다. 이것은 파이썬 프로그램에 `types.MethodType`로 노출됩니다.

int **PyMethod_Check**(*PyObject* *o)

*o*가 메서드 객체면 참을 반환합니다 (*PyMethod_Type* 형입니다). 매개 변수는 `NULL`이 아니어야 합니다. 이 함수는 항상 성공합니다.

PyObject ***PyMethod_New**(*PyObject* *func, *PyObject* *self)

Return value: New reference. 새로운 메서드 객체를 돌려줍니다. *func*는 임의의 콜러블 객체이며, *self*는 메서드가 연결되어야 할 인스턴스입니다. *func*는 메서드가 호출될 때 호출될 함수입니다. *self*는 `NULL`이 아니어야 합니다.

PyObject ***PyMethod_Function**(*PyObject* *meth)

Return value: Borrowed reference. *meth* 메서드와 연관된 함수 객체를 반환합니다.

PyObject ***PyMethod_GET_FUNCTION**(*PyObject* *meth)

Return value: Borrowed reference. 오류 검사를 피하는 `PyMethod_Function()`의 매크로 버전.

PyObject ***PyMethod_Self**(*PyObject* *meth)

Return value: Borrowed reference. *meth* 메서드와 연관된 인스턴스를 반환합니다.

PyObject ***PyMethod_GET_SELF**(*PyObject* *meth)

Return value: Borrowed reference. 오류 검사를 피하는 `PyMethod_Self()`의 매크로 버전.

8.5.4 셀 객체

"셀" 객체는 여러 스코프에서 참조하는 변수를 구현하는 데 사용됩니다. 이러한 변수마다, 값을 저장하기 위해 셀 객체가 만들어집니다; 값을 참조하는 각 스택 프레임의 지역 변수에는 해당 변수를 사용하는 외부 스코프의 셀에 대한 참조가 포함됩니다. 값에 액세스하면, 셀 객체 자체 대신 셀에 포함된 값이 사용됩니다. 이러한 셀 객체의 역참조(de-referencing)는 생성된 바이트 코드로부터의 지원이 필요합니다; 액세스 시 자동으로 역참조되지 않습니다. 셀 객체는 다른 곳에 유용하지는 않습니다.

type **PyCellObject**

셀 객체에 사용되는 C 구조체.

PyTypeObject **PyCell_Type**

셀 객체에 해당하는 형 객체.

int **PyCell_Check**(*PyObject* *ob)

*ob*가 셀 객체면 참을 반환합니다; *ob*는 `NULL`이 아니어야 합니다. 이 함수는 항상 성공합니다.

PyObject ***PyCell_New**(*PyObject* *ob)

Return value: New reference. *ob* 값을 포함하는 새 셀 객체를 만들고 반환합니다. 매개 변수는 `NULL` 일 수 있습니다.

PyObject ***PyCell_Get**(*PyObject* *cell)

Return value: New reference. Return the contents of the cell *cell*, which can be `NULL`. If *cell* is not a cell object, returns `NULL` with an exception set.

PyObject ***PyCell_GET**(*PyObject* *cell)

Return value: Borrowed reference. 셀 *cell*의 내용을 반환하지만, *cell*이 `NULL`이 아닌지와 셀 객체인지를 확인하지 않습니다.

int **PyCell_Set**(*PyObject* *cell, *PyObject* *value)

Set the contents of the cell object *cell* to *value*. This releases the reference to any current content of the cell. *value* may be `NULL`. *cell* must be non-`NULL`.

On success, return `0`. If *cell* is not a cell object, set an exception and return `-1`.

void **PyCell_SET**(*PyObject* *cell, *PyObject* *value)

셀 객체 *cell*의 값을 *value*로 설정합니다. 참조 횟수는 조정되지 않고, 안전을 위한 검사가 이루어지지 않습니다; *cell*은 `NULL`이 아니어야 하고 셀 객체여야 합니다.

8.5.5 코드 객체

코드 객체는 CPython 구현의 저수준 세부 사항입니다. 각 객체는 아직 함수에 묶여 있지 않은 실행 가능한 코드 덩어리를 나타냅니다.

type **PyCodeObject**

코드 객체를 설명하는 데 사용되는 객체의 C 구조체. 이 형의 필드는 언제든지 변경될 수 있습니다.

PyTypeObject **PyCode_Type**

This is an instance of `PyTypeObject` representing the Python code object.

int **PyCode_Check**(*PyObject* *co)

Return true if *co* is a code object. This function always succeeds.

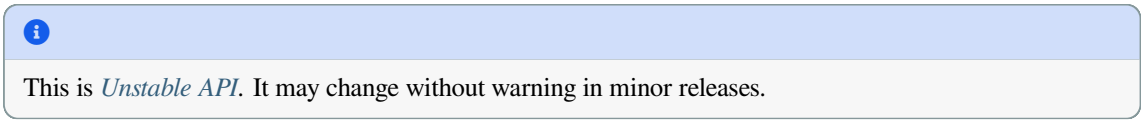
Py_ssize_t **PyCode_GetNumFree**(*PyCodeObject* *co)

Return the number of free variables in a code object.

int **PyCode_GetFirstFree**(*PyCodeObject* *co)

Return the position of the first free variable in a code object.

PyCodeObject ***PyUnstable_Code_New**(int argcount, int kwonlyargcount, int nlocals, int stacksize, int flags, *PyObject* *code, *PyObject* *consts, *PyObject* *names, *PyObject* *varnames, *PyObject* *freevars, *PyObject* *cellvars, *PyObject* *filename, *PyObject* *name, *PyObject* *qualname, int firstlineno, *PyObject* *linetable, *PyObject* *exceptiontable)



This is *Unstable API*. It may change without warning in minor releases.

Return a new code object. If you need a dummy code object to create a frame, use `PyCode_NewEmpty()` instead.

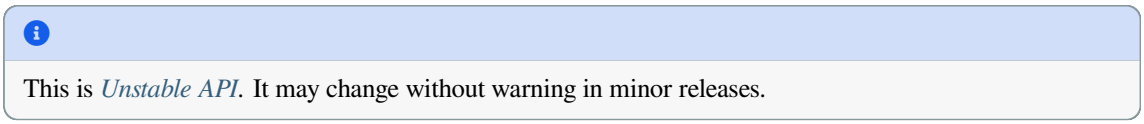
Since the definition of the bytecode changes often, calling `PyUnstable_Code_New()` directly can bind you to a precise Python version.

The many arguments of this function are inter-dependent in complex ways, meaning that subtle changes to values are likely to result in incorrect execution or VM crashes. Use this function only with extreme care.

버전 3.11에서 변경: Added `qualname` and `exceptiontable` parameters.

버전 3.12에서 변경: Renamed from `PyCode_New` as part of *Unstable C API*. The old name is deprecated, but will remain available until the signature changes again.

PyCodeObject ***PyUnstable_Code_NewWithPosOnlyArgs**(int argcount, int posonlyargcount, int kwonlyargcount, int nlocals, int stacksize, int flags, *PyObject* *code, *PyObject* *consts, *PyObject* *names, *PyObject* *varnames, *PyObject* *freevars, *PyObject* *cellvars, *PyObject* *filename, *PyObject* *name, *PyObject* *qualname, int firstlineno, *PyObject* *linetable, *PyObject* *exceptiontable)



This is *Unstable API*. It may change without warning in minor releases.

Similar to `PyUnstable_Code_New()`, but with an extra "posonlyargcount" for positional-only arguments. The same caveats that apply to `PyUnstable_Code_New` also apply to this function.

Added in version 3.8: as `PyCode_NewWithPosOnlyArgs`

버전 3.11에서 변경: Added `qualname` and `exceptiontable` parameters.

버전 3.12에서 변경: Renamed to `PyUnstable_Code_NewWithPosOnlyArgs`. The old name is deprecated, but will remain available until the signature changes again.

PyCodeObject ***PyCode_NewEmpty**(const char *filename, const char *funcname, int firstlineno)

Return value: New reference. Return a new empty code object with the specified filename, function name, and first line number. The resulting code object will raise an `Exception` if executed.

int **PyCode_Addr2Line**(*PyCodeObject* *co, int byte_offset)

Return the line number of the instruction that occurs on or before `byte_offset` and ends after it. If you just need the line number of a frame, use *PyFrame_GetLineNumber()* instead.

For efficiently iterating over the line numbers in a code object, use **the API described in PEP 626**.

int **PyCode_Addr2Location**(*PyObject* *co, int byte_offset, int *start_line, int *start_column, int *end_line, int *end_column)

Sets the passed `int` pointers to the source code line and column numbers for the instruction at `byte_offset`. Sets the value to `0` when information is not available for any particular element.

Returns `1` if the function succeeds and 0 otherwise.

Added in version 3.11.

PyObject ***PyCode_GetCode**(*PyCodeObject* *co)

Equivalent to the Python code `getattr(co, 'co_code')`. Returns a strong reference to a *PyBytesObject* representing the bytecode in a code object. On error, `NULL` is returned and an exception is raised.

This `PyBytesObject` may be created on-demand by the interpreter and does not necessarily represent the bytecode actually executed by CPython. The primary use case for this function is debuggers and profilers.

Added in version 3.11.

PyObject ***PyCode_GetVarnames**(*PyCodeObject* *co)

Equivalent to the Python code `getattr(co, 'co_varnames')`. Returns a new reference to a *PyTupleObject* containing the names of the local variables. On error, `NULL` is returned and an exception is raised.

Added in version 3.11.

PyObject ***PyCode_GetCellvars**(*PyCodeObject* *co)

Equivalent to the Python code `getattr(co, 'co_cellvars')`. Returns a new reference to a *PyTupleObject* containing the names of the local variables that are referenced by nested functions. On error, `NULL` is returned and an exception is raised.

Added in version 3.11.

PyObject ***PyCode_GetFreevars**(*PyCodeObject* *co)

Equivalent to the Python code `getattr(co, 'co_freevars')`. Returns a new reference to a *PyTupleObject* containing the names of the free variables. On error, `NULL` is returned and an exception is raised.

Added in version 3.11.

int **PyCode_AddWatcher**(*PyCode_WatchCallback* callback)

Register *callback* as a code object watcher for the current interpreter. Return an ID which may be passed to *PyCode_ClearWatcher()*. In case of error (e.g. no more watcher IDs available), return `-1` and set an exception.

Added in version 3.12.

int **PyCode_ClearWatcher**(int watcher_id)

Clear watcher identified by *watcher_id* previously returned from *PyCode_AddWatcher()* for the current interpreter. Return `0` on success, or `-1` and set an exception on error (e.g. if the given *watcher_id* was never registered.)

Added in version 3.12.

type **PyCodeEvent**

Enumeration of possible code object watcher events: - `PY_CODE_EVENT_CREATE` - `PY_CODE_EVENT_DESTROY`

Added in version 3.12.

typedef int (***PyCode_WatchCallback**)(*PyCodeEvent* event, *PyCodeObject* *co)

Type of a code object watcher callback function.

If *event* is `PY_CODE_EVENT_CREATE`, then the callback is invoked after *co* has been fully initialized. Otherwise, the callback is invoked before the destruction of *co* takes place, so the prior state of *co* can be inspected.

If *event* is `PY_CODE_EVENT_DESTROY`, taking a reference in the callback to the about-to-be-destroyed code object will resurrect it and prevent it from being freed at this time. When the resurrected object is destroyed later, any watcher callbacks active at that time will be called again.

Users of this API should not rely on internal runtime implementation details. Such details may include, but are not limited to, the exact order and timing of creation and destruction of code objects. While changes in these details may result in differences observable by watchers (including whether a callback is invoked or not), it does not change the semantics of the Python code being executed.

If the callback sets an exception, it must return `-1`; this exception will be printed as an unraisable exception using *`PyErr_WriteUnraisable()`*. Otherwise it should return `0`.

There may already be a pending exception set on entry to the callback. In this case, the callback should return `0` with the same exception still set. This means the callback may not call any other API that can set an exception unless it saves and clears the exception state first, and restores it before returning.

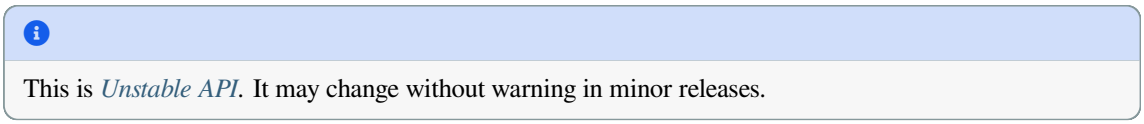
Added in version 3.12.

8.5.6 Extra information

To support low-level extensions to frame evaluation, such as external just-in-time compilers, it is possible to attach arbitrary extra data to code objects.

These functions are part of the unstable C API tier: this functionality is a CPython implementation detail, and the API may change without deprecation warnings.

Py_ssize_t **PyUnstable_Eval_RequestCodeExtraIndex**(*freefunc* free)



This is *Unstable API*. It may change without warning in minor releases.

Return a new an opaque index value used to adding data to code objects.

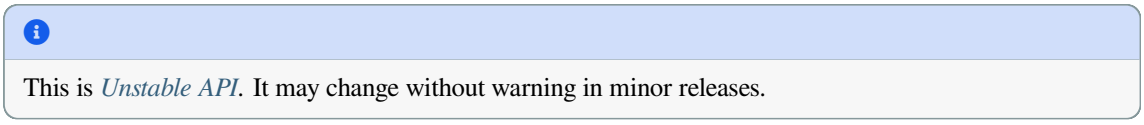
You generally call this function once (per interpreter) and use the result with `PyCode_GetExtra` and `PyCode_SetExtra` to manipulate data on individual code objects.

If *free* is not `NULL`: when a code object is deallocated, *free* will be called on non-`NULL` data stored under the new index. Use *`Py_DecRef()`* when storing *`PyObject`*.

Added in version 3.6: as `_PyEval_RequestCodeExtraIndex`

버전 3.12에서 변경: Renamed to `PyUnstable_Eval_RequestCodeExtraIndex`. The old private name is deprecated, but will be available until the API changes.

int **PyUnstable_Code_GetExtra**(*PyObject* *code, *Py_ssize_t* index, void **extra)



This is *Unstable API*. It may change without warning in minor releases.

Set *extra* to the extra data stored under the given index. Return 0 on success. Set an exception and return -1 on failure.

If no data was set under the index, set *extra* to `NULL` and return 0 without setting an exception.

Added in version 3.6: as `_PyCode_GetExtra`

버전 3.12에서 변경: Renamed to `PyUnstable_Code_GetExtra`. The old private name is deprecated, but will be available until the API changes.

int **PyUnstable_Code_SetExtra**(*PyObject* *code, *Py_ssize_t* index, void *extra)

This is *Unstable API*. It may change without warning in minor releases.

Set the extra data stored under the given index to *extra*. Return 0 on success. Set an exception and return -1 on failure.

Added in version 3.6: as `_PyCode_SetExtra`

버전 3.12에서 변경: Renamed to `PyUnstable_Code_SetExtra`. The old private name is deprecated, but will be available until the API changes.

8.6 기타 객체

8.6.1 파일 객체

These APIs are a minimal emulation of the Python 2 C API for built-in file objects, which used to rely on the buffered I/O (`FILE*`) support from the C standard library. In Python 3, files and streams use the new `io` module, which defines several layers over the low-level unbuffered I/O of the operating system. The functions described below are convenience C wrappers over these new APIs, and meant mostly for internal error reporting in the interpreter; third-party code is advised to access the `io` APIs instead.

PyObject ***PyFile_FromFd**(int fd, const char *name, const char *mode, int buffering, const char *encoding, const char *errors, const char *newline, int closefd)

Return value: New reference. Part of the Stable ABI. 이미 열려있는 파일의 파일 기술자 *fd*로 파이썬 파일 객체를 만듭니다. 인자 *name*, *encoding*, *errors* 및 *newline*은 기본값을 사용하기 위해 `NULL` 일 수 있습니다; *buffering*은 기본값을 사용하기 위해 *-1* 일 수 있습니다. *name*은 무시되고, 이전 버전과의 호환성을 위해 유지됩니다. 실패 시 `NULL`을 반환합니다. 인자에 대한 더 자세한 설명은 `io.open()` 함수 설명서를 참조하십시오.

경고

파이썬 스트림이 자체적인 버퍼링 계층을 가지고 있으므로, OS 수준의 파일 기술자와 혼합하면 여러 예기치 못한 문제가 발생할 수 있습니다 (가령 데이터의 예상치 못한 순서).

버전 3.2에서 변경: *name* 어트리뷰트를 무시합니다.

int **PyObject_AsFileDescriptor**(*PyObject* *p)

Part of the Stable ABI. Return the file descriptor associated with *p* as an `int`. If the object is an integer, its value is returned. If not, the object's `fileno()` method is called if it exists; the method must return an integer, which is returned as the file descriptor value. Sets an exception and returns `-1` on failure.

PyObject ***PyFile_GetLine**(*PyObject* *p, int n)

Return value: New reference. Part of the Stable ABI. `p.readline([n])`과 동등합니다. 이 함수는 객체 *p*에서 한 줄을 읽습니다. *p*는 파일 객체나 `readline()` 메서드가 있는 임의의 객체일 수 있습니다. *n*이 `0`이면, 줄의 길이와 관계없이 정확히 한 줄을 읽습니다. *n*이 `0`보다 크면, *n* 바이트 이상을 파일에서 읽지 않습니다; 불완전한 줄이 반환될 수 있습니다. 두 경우 모두, 파일 끝에 즉시 도달하면 빈

문자열이 반환됩니다. 그러나 *n*이 `0`보다 작으면, 길이와 관계없이 한 줄을 읽지만, 파일 끝에 즉시 도달하면 `EOFError`가 발생합니다.

int **PyFile_SetOpenCodeHook**(*Py_OpenCodeHookFunction* handler)

제공된 handler를 통해 매개 변수를 전달하도록 `io.open_code()`의 일반적인 동작을 재정의합니다.

The *handler* is a function of type:

typedef *PyObject* *(***Py_OpenCodeHookFunction**)(*PyObject**, void*)

Equivalent of `PyObject *(*)(PyObject *path, void *userData)`, where *path* is guaranteed to be `PyUnicodeObject`.

userData 포인터는 훅 함수로 전달됩니다. 훅 함수는 다른 런타임에서 호출될 수 있으므로, 이 포인터는 파이썬 상태를 직접 참조하면 안 됩니다.

이 훅은 의도적으로 임포트 중에 사용되므로, 고정되었거나(frozen) `sys.modules`에 있다고 알려진 경우가 아니라면 훅 실행 중에 새로운 모듈을 임포트하는 것을 피하십시오.

일단 훅이 설정되면, 제거하거나 교체할 수 없으며, 이후의 `PyFile_SetOpenCodeHook()`에 대한 호출은 실패합니다. 실패 시, 함수는 -1을 반환하고 인터프리터가 초기화되었으면 예외를 설정합니다.

이 함수는 `Py_Initialize()` 전에 호출해도 안전합니다.

인자 없이 감사 이벤트 `setopencodehook`을 발생시킵니다.

Added in version 3.8.

int **PyFile_WriteObject**(*PyObject* *obj, *PyObject* *p, int flags)

Part of the Stable ABI. Write object *obj* to file object *p*. The only supported flag for *flags* is `Py_PRINT_RAW`; if given, the `str()` of the object is written instead of the `repr()`. Return `0` on success or `-1` on failure; the appropriate exception will be set.

int **PyFile_WriteString**(const char *s, *PyObject* *p)

Part of the Stable ABI. 문자열 *s*를 파일 객체 *p*에 씁니다. 성공하면 `0`을 반환하고, 실패하면 `-1`을 반환합니다; 적절한 예외가 설정됩니다.

8.6.2 모듈 객체

PyTypeObject **PyModule_Type**

Part of the Stable ABI. 이 `PyTypeObject` 인스턴스는 파이썬 모듈 형을 나타냅니다. 이것은 `types.ModuleType`으로 파이썬 프로그램에 노출됩니다.

int **PyModule_Check**(*PyObject* *p)

*p*가 모듈 객체이거나 모듈 객체의 서브 형이면 참을 반환합니다. 이 함수는 항상 성공합니다.

int **PyModule_CheckExact**(*PyObject* *p)

*p*가 모듈 객체이지만, `PyModule_Type`의 서브 형이 아니면 참을 반환합니다. 이 함수는 항상 성공합니다.

PyObject ***PyModule_NewObject**(*PyObject* *name)

Return value: New reference. Part of the Stable ABI *since version 3.7.* Return a new module object with `module.__name__` set to *name*. The module's `__name__`, `__doc__`, `__package__` and `__loader__` attributes are filled in (all but `__name__` are set to `None`). The caller is responsible for setting a `__file__` attribute.

Return `NULL` with an exception set on error.

Added in version 3.3.

버전 3.4에서 변경: `__package__` and `__loader__` are now set to `None`.

PyObject ***PyModule_New**(const char *name)

Return value: New reference. Part of the Stable ABI. `PyModule_NewObject()`와 비슷하지만, name이 유니코드 객체 대신 UTF-8로 인코딩된 문자열입니다.

PyObject ***PyModule_GetDict**(*PyObject* *module)

Return value: Borrowed reference. Part of the Stable ABI. *module*의 이름 공간을 구현하는 딕셔너리 객체를 반환합니다; 이 객체는 모듈 객체의 `__dict__` 어트리뷰트와 같습니다. *module*이 모듈 객체(또는 모듈 객체의 서브 형)가 아니면, `SystemError`가 발생하고 `NULL`이 반환됩니다.

It is recommended extensions use other `PyModule_*` and `PyObject_*` functions rather than directly manipulate a module's `__dict__`.

PyObject ***PyModule_GetNameObject**(*PyObject* *module)

Return value: New reference. Part of the Stable ABI *since version 3.7.* Return *module*'s `__name__` value. If the module does not provide one, or if it is not a string, `SystemError` is raised and `NULL` is returned.

Added in version 3.3.

const char ***PyModule_GetName**(*PyObject* *module)

Part of the Stable ABI. *PyModule_GetNameObject()*와 비슷하지만 `'utf-8'`로 인코딩된 이름을 반환합니다.

void ***PyModule_GetState**(*PyObject* *module)

Part of the Stable ABI. 모듈의 "상태", 즉 모듈 생성 시 할당된 메모리 블록을 가리키는 포인터나 `NULL`을 반환합니다. *PyModuleDef.m_size*를 참조하십시오.

PyModuleDef ***PyModule_GetDef**(*PyObject* *module)

Part of the Stable ABI. 모듈이 만들어진 *PyModuleDef* 구조체에 대한 포인터나 모듈이 정의에서 만들어지지 않았으면 `NULL`을 반환합니다.

PyObject ***PyModule_GetFilenameObject**(*PyObject* *module)

Return value: New reference. Part of the Stable ABI. Return the name of the file from which *module* was loaded using *module*'s `__file__` attribute. If this is not defined, or if it is not a string, raise `SystemError` and return `NULL`; otherwise return a reference to a Unicode object.

Added in version 3.2.

const char ***PyModule_GetFilename**(*PyObject* *module)

Part of the Stable ABI. *PyModule_GetFilenameObject()*와 비슷하지만 'utf-8'로 인코딩된 파일명을 반환합니다.

버전 3.2부터 폐지됨: *PyModule_GetFilename()* raises `UnicodeEncodeError` on unencodable filenames, use *PyModule_GetFilenameObject()* instead.

C 모듈 초기화

모듈 객체는 일반적으로 확장 모듈(초기화 함수를 내보내는 공유 라이브러리)이나 컴파일된 모듈(초기화 함수가 *PyImport_AppendInittab()*을 사용하여 추가된)에서 만들어집니다. 자세한 내용은 building나 extending-with-embedding를 참조하십시오.

초기화 함수는 모듈 정의 인스턴스를 *PyModule_Create()*에 전달하고 결과 모듈 객체를 반환하거나, 정의 구조체 자체를 반환하여 "다단계 초기화"를 요청할 수 있습니다.

type **PyModuleDef**

Part of the Stable ABI (*including all members*). 모듈 객체를 만드는 데 필요한 모든 정보를 담고 있는 모듈 정의 구조체. 일반적으로 각 모듈에 대해 이 형의 정적으로 초기화된 변수가 하나만 있습니다.

PyModuleDef_Base **m_base**

Always initialize this member to `PyModuleDef_HEAD_INIT`.

const char ***m_name**

새 모듈의 이름.

const char ***m_doc**

모듈의 독스트링; 일반적으로 *PyDoc_STRVAR*로 만들어진 독스트링 변수가 사용됩니다.

Py_ssize_t **m_size**

모듈 상태는 정적 전역이 아닌 *PyModule_GetState()*로 조회할 수 있는 모듈별 메모리 영역에 유지될 수 있습니다. 이것은 여러 서브 인터프리터에서 모듈을 사용하는 것을 안전하게 만듭니다.

This memory area is allocated based on *m_size* on module creation, and freed when the module object is deallocated, after the *m_free* function has been called, if present.

`m_size`를 `-1`로 설정하면 모듈이 전역 상태를 갖기 때문에 서브 인터프리터를 지원하지 않는다는 뜻입니다.

음수가 아닌 값으로 설정하면 모듈을 다시 초기화 할 수 있다는 뜻이며 상태에 필요한 추가 메모리양을 지정합니다. 다단계 초기화에는 음이 아닌 `m_size`가 필요합니다.

자세한 내용은 **PEP 3121**을 참조하십시오.

PyMethodDef ***m_methods**

PyMethodDef 값으로 기술되는 모듈 수준 함수 테이블에 대한 포인터. 함수가 없으면 `NULL`일 수 있습니다.

PyModuleDef_Slot ***m_slots**

다단계 초기화를 위한 슬롯 정의 배열, `{0, NULL}` 항목으로 종료됩니다. 단단계 초기화를 사용할 때, *m_slots*는 `NULL`이어야 합니다.

버전 3.5에서 변경: 버전 3.5 이전에는, 이 멤버가 항상 `NULL`로 설정되었으며, 다음과 같이 정의되었습니다:

inquiry **m_reload**

traverseproc **m_traverse**

모듈 객체의 GC 탐색 중 호출할 탐색 함수나, 필요하지 않으면 `NULL`.

This function is not called if the module state was requested but is not allocated yet. This is the case immediately after the module is created and before the module is executed (*Py_mod_exec* function). More precisely, this function is not called if *m_size* is greater than 0 and the module state (as returned by *PyModule_GetState()*) is `NULL`.

버전 3.9에서 변경: 모듈 상태가 할당되기 전에 더는 호출되지 않습니다.

inquiry **m_clear**

모듈 객체의 GC 정리 중에 호출할 정리(clear) 함수나, 필요하지 않으면 `NULL`.

This function is not called if the module state was requested but is not allocated yet. This is the case immediately after the module is created and before the module is executed (*Py_mod_exec* function). More precisely, this function is not called if *m_size* is greater than 0 and the module state (as returned by *PyModule_GetState()*) is `NULL`.

Like *PyTypeObject.tp_clear*, this function is not *always* called before a module is deallocated. For example, when reference counting is enough to determine that an object is no longer used, the cyclic garbage collector is not involved and *m_free* is called directly.

버전 3.9에서 변경: 모듈 상태가 할당되기 전에 더는 호출되지 않습니다.

freefunc **m_free**

모듈 객체 할당 해제 중에 호출할 함수나, 필요하지 않으면 `NULL`.

This function is not called if the module state was requested but is not allocated yet. This is the case immediately after the module is created and before the module is executed (*Py_mod_exec* function). More precisely, this function is not called if *m_size* is greater than 0 and the module state (as returned by *PyModule_GetState()*) is `NULL`.

버전 3.9에서 변경: 모듈 상태가 할당되기 전에 더는 호출되지 않습니다.

단단계 초기화

모듈 초기화 함수는 모듈 객체를 직접 만들고 반환할 수 있습니다. 이것을 "단단계 초기화"라고 하며, 다음 두 모듈 생성 함수 중 하나를 사용합니다:

PyObject ***PyModule_Create**(*PyModuleDef* *def)

Return value: New reference. Create a new module object, given the definition in *def*. This behaves like *PyModule_Create2()* with *module_api_version* set to PYTHON_API_VERSION.

PyObject ***PyModule_Create2**(*PyModuleDef* *def, int module_api_version)

Return value: New reference. Part of the Stable ABI. *def*의 정의에 따라, API 버전 *module_api_version*을 가정하여 새 모듈 객체를 만듭니다. 해당 버전이 실행 중인 인터프리터 버전과 일치하지 않으면, RuntimeWarning을 발생시킵니다.

Return NULL with an exception set on error.

> **참고**
>
> 이 함수는 대부분 *PyModule_Create()*를 대신 사용해야 합니다; 확실히 필요할 때만 사용하십시오.

Before it is returned from in the initialization function, the resulting module object is typically populated using functions like *PyModule_AddObjectRef()*.

다단계 초기화

An alternate way to specify extensions is to request "multi-phase initialization". Extension modules created this way behave more like Python modules: the initialization is split between the *creation phase*, when the module object is created, and the *execution phase*, when it is populated. The distinction is similar to the __new__() and __init__() methods of classes.

Unlike modules created using single-phase initialization, these modules are not singletons: if the *sys.modules* entry is removed and the module is re-imported, a new module object is created, and the old module is subject to normal garbage collection – as with Python modules. By default, multiple modules created from the same definition should be independent: changes to one should not affect the others. This means that all state should be specific to the module object (using e.g. using *PyModule_GetState()*), or its contents (such as the module's __dict__ or individual classes created with *PyType_FromSpec()*).

다단계 초기화를 사용하여 만들어진 모든 모듈은 서브 인터프리터를 지원할 것으로 기대됩니다. 다중 모듈을 독립적으로 유지하는 것은 일반적으로 이를 달성하기에 충분합니다.

다단계 초기화를 요청하기 위해, 초기화 함수(PyInit_modulename)는 비어 있지 않은 *m_slots*를 가진 *PyModuleDef* 인스턴스를 반환합니다. 반환되기 전에, PyModuleDef 인스턴스를 다음 함수를 사용하여 초기화해야 합니다:

PyObject ***PyModuleDef_Init**(*PyModuleDef* *def)

Return value: Borrowed reference. Part of the Stable ABI *since version 3.5.* 모듈 정의가 형과 참조 횟수를 올바르게 보고하는 올바르게 초기화된 파이썬 객체이게 합니다.

*def*를 PyObject*로 캐스트 하거나, 에러가 발생하면 NULL을 반환합니다.

Added in version 3.5.

모듈 정의의 *m_slots* 멤버는 PyModuleDef_Slot 구조체의 배열을 가리켜야 합니다:

type **PyModuleDef_Slot**

int **slot**

아래 설명된 사용 가능한 값 중에서 선택된, 슬롯 ID.

void ***value**

슬롯 ID에 따라 그 의미가 달라지는, 슬롯의 값.

Added in version 3.5.

m_slots 배열은 id가 0인 슬롯으로 종료해야 합니다.

사용 가능한 슬롯 형은 다음과 같습니다:

Py_mod_create

모듈 객체 자체를 만들기 위해 호출되는 함수를 지정합니다. 이 슬롯의 *value* 포인터는 다음과 같은 서명을 갖는 함수를 가리켜야 합니다:

PyObject ***create_module**(*PyObject* *spec, *PyModuleDef* *def)

이 함수는 **PEP 451**에 정의된 대로, `ModuleSpec` 인스턴스와 모듈 정의를 받습니다. 새 모듈 객체를 반환하거나, 에러를 설정하고 `NULL`을 반환해야 합니다.

이 함수는 최소한으로 유지해야 합니다. 특히 같은 모듈을 다시 임포트 하려고 시도하면 무한 루프가 발생할 수 있어서, 임의의 파이썬 코드를 호출하면 안 됩니다.

하나의 모듈 정의에서 여러 `Py_mod_create` 슬롯을 지정할 수 없습니다.

`Py_mod_create`를 지정하지 않으면, 임포트 절차는 *PyModule_New()*를 사용하여 일반 모듈 객체를 만듭니다. 이름은 정의가 아니라 *spec*에서 취합니다, 확장 모듈이 단일 모듈 정의를 공유하면서 모듈 계층 구조에서 해당 위치에 동적으로 조정되고 심볼릭 링크를 통해 다른 이름으로 임포트 될 수 있도록 하기 위함입니다.

반환된 객체가 *PyModule_Type*의 인스턴스 일 필요는 없습니다. 임포트 관련 어트리뷰트 설정과 읽기를 지원하는 한 모든 형을 사용할 수 있습니다. 그러나, `PyModuleDef`에 `NULL`이 아닌 `m_traverse`, `m_clear`, `m_free`; 0이 아닌 `m_size`; 또는 `Py_mod_create` 이외의 슬롯이 있으면, `PyModule_Type` 인스턴스 만 반환될 수 있습니다.

Py_mod_exec

모듈을 실행하기 위해 호출되는 함수를 지정합니다. 이것은 파이썬 모듈의 코드를 실행하는 것과 동등합니다: 일반적으로, 이 함수는 클래스와 상수를 모듈에 추가합니다. 함수의 서명은 다음과 같습니다:

int **exec_module**(*PyObject* *module)

여러 개의 `Py_mod_exec` 슬롯이 지정되면, *m_slots* 배열에 나타나는 순서대로 처리됩니다.

Py_mod_multiple_interpreters

Specifies one of the following values:

Py_MOD_MULTIPLE_INTERPRETERS_NOT_SUPPORTED

The module does not support being imported in subinterpreters.

Py_MOD_MULTIPLE_INTERPRETERS_SUPPORTED

The module supports being imported in subinterpreters, but only when they share the main interpreter's GIL. (See isolating-extensions-howto.)

Py_MOD_PER_INTERPRETER_GIL_SUPPORTED

The module supports being imported in subinterpreters, even when they have their own GIL. (See isolating-extensions-howto.)

This slot determines whether or not importing this module in a subinterpreter will fail.

Multiple `Py_mod_multiple_interpreters` slots may not be specified in one module definition.

If `Py_mod_multiple_interpreters` is not specified, the import machinery defaults to `Py_MOD_MULTIPLE_INTERPRETERS_SUPPORTED`.

Added in version 3.12.

다단계 초기화에 대한 자세한 내용은 **PEP 489**를 참조하십시오.

저수준 모듈 생성 함수

다단계 초기화를 사용할 때 수면 아래에서는 다음 함수가 호출됩니다. 이들은 직접 사용할 수 있는데, 예를 들어 모듈 객체를 동적으로 생성할 때 그렇습니다. 모듈을 완전히 초기화하려면 `PyModule_FromDefAndSpec`과 `PyModule_ExecDef`를 모두 호출해야 함에 유의하십시오.

PyObject ***PyModule_FromDefAndSpec**(*PyModuleDef* *def, *PyObject* *spec)

Return value: New reference. Create a new module object, given the definition in *def* and the ModuleSpec *spec*. This behaves like *PyModule_FromDefAndSpec2()* with *module_api_version* set to `PYTHON_API_VERSION`.

Added in version 3.5.

PyObject ***PyModule_FromDefAndSpec2**(*PyModuleDef* *def, *PyObject* *spec, int module_api_version)

Return value: New reference. Part of the Stable ABI *since version 3.7.* Create a new module object, given the definition in *def* and the ModuleSpec *spec*, assuming the API version *module_api_version*. If that version does not match the version of the running interpreter, a `RuntimeWarning` is emitted.

Return `NULL` with an exception set on error.

> **참고**
>
> 이 함수는 대부분 *PyModule_FromDefAndSpec()*을 대신 사용해야 합니다; 확실히 필요할 때만 사용하십시오.

Added in version 3.5.

int **PyModule_ExecDef**(*PyObject* *module, *PyModuleDef* *def)

Part of the Stable ABI *since version 3.7. def*에 지정된 모든 실행 슬롯(*Py_mod_exec*)을 처리합니다.

Added in version 3.5.

int **PyModule_SetDocString**(*PyObject* *module, const char *docstring)

Part of the Stable ABI *since version 3.7. module*의 독스트링을 *docstring*으로 설정합니다. 이 함수는 `PyModule_Create`나 `PyModule_FromDefAndSpec`을 사용하여 `PyModuleDef`에서 모듈을 만들 때 자동으로 호출됩니다.

Added in version 3.5.

int **PyModule_AddFunctions**(*PyObject* *module, *PyMethodDef* *functions)

Part of the Stable ABI *since version 3.7.* `NULL` 종료 *functions* 배열의 함수를 *module*에 추가합니다. 개별 항목에 대한 자세한 내용은 *PyMethodDef* 설명서를 참조하십시오 (공유 모듈 이름 공간이 없기 때문에, C로 구현된 모듈 수준 "함수(functions)"는 일반적으로 첫 번째 매개 변수로 모듈을 수신하여, 파이썬 클래스의 인스턴스 메서드와 유사하게 만듭니다). 이 함수는 `PyModule_Create`나 `PyModule_FromDefAndSpec`을 사용하여 `PyModuleDef`에서 모듈을 만들 때 자동으로 호출됩니다.

Added in version 3.5.

지원 함수

모듈 초기화 함수(단단계 초기화를 사용하는 경우)나 모듈 실행 슬롯에서 호출되는 함수(다단계 초기화를 사용하는 경우)는, 모듈 상태 초기화를 도우려고 다음 함수를 사용할 수 있습니다:

int **PyModule_AddObjectRef**(*PyObject* *module, const char *name, *PyObject* *value)

Part of the Stable ABI *since version 3.10.* Add an object to *module* as *name*. This is a convenience function which can be used from the module's initialization function.

On success, return `0`. On error, raise an exception and return `-1`.

Example usage:

```
static int
add_spam(PyObject *module, int value)
{
    PyObject *obj = PyLong_FromLong(value);
    if (obj == NULL) {
        return -1;
    }
    int res = PyModule_AddObjectRef(module, "spam", obj);
    Py_DECREF(obj);
    return res;
 }
```

To be convenient, the function accepts `NULL` *value* with an exception set. In this case, return `-1` and just leave the raised exception unchanged.

The example can also be written without checking explicitly if *obj* is `NULL`:

```
static int
add_spam(PyObject *module, int value)
{
    PyObject *obj = PyLong_FromLong(value);
    int res = PyModule_AddObjectRef(module, "spam", obj);
    Py_XDECREF(obj);
    return res;
 }
```

Note that `Py_XDECREF()` should be used instead of `Py_DECREF()` in this case, since *obj* can be `NULL`.

The number of different *name* strings passed to this function should be kept small, usually by only using statically allocated strings as *name*. For names that aren't known at compile time, prefer calling *PyUnicode_FromString()* and *PyObject_SetAttr()* directly. For more details, see *PyUnicode_InternFromString()*, which may be used internally to create a key object.

Added in version 3.10.

int **PyModule_AddObject**(*PyObject* *module, const char *name, *PyObject* *value)

Part of the Stable ABI. Similar to *PyModule_AddObjectRef()*, but steals a reference to *value* on success (if it returns `0`).

The new *PyModule_AddObjectRef()* function is recommended, since it is easy to introduce reference leaks by misusing the *PyModule_AddObject()* function.

> **참고**
>
> Unlike other functions that steal references, `PyModule_AddObject()` only releases the reference to *value* **on success**.
>
> This means that its return value must be checked, and calling code must *Py_DECREF() value* manually on error.

Example usage:

```
static int
add_spam(PyObject *module, int value)
{
    PyObject *obj = PyLong_FromLong(value);
    if (obj == NULL) {
        return -1;
    }
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    if (PyModule_AddObject(module, "spam", obj) < 0) {
        Py_DECREF(obj);
        return -1;
    }
    // PyModule_AddObject() stole a reference to obj:
    // Py_DECREF(obj) is not needed here
    return 0;
}
```

The example can also be written without checking explicitly if *obj* is `NULL`:

```
static int
add_spam(PyObject *module, int value)
{
    PyObject *obj = PyLong_FromLong(value);
    if (PyModule_AddObject(module, "spam", obj) < 0) {
        Py_XDECREF(obj);
        return -1;
    }
    // PyModule_AddObject() stole a reference to obj:
    // Py_DECREF(obj) is not needed here
    return 0;
}
```

Note that `Py_XDECREF()` should be used instead of `Py_DECREF()` in this case, since *obj* can be `NULL`.

int **PyModule_AddIntConstant**(*PyObject* *module, const char *name, long value)

Part of the Stable ABI. Add an integer constant to *module* as *name*. This convenience function can be used from the module' s initialization function. Return `-1` with an exception set on error, `0` on success.

This is a convenience function that calls *PyLong_FromLong()* and *PyModule_AddObjectRef()*; see their documentation for details.

int **PyModule_AddStringConstant**(*PyObject* *module, const char *name, const char *value)

Part of the Stable ABI. Add a string constant to *module* as *name*. This convenience function can be used from the module' s initialization function. The string *value* must be `NULL`-terminated. Return `-1` with an exception set on error, `0` on success.

This is a convenience function that calls *PyUnicode_InternFromString()* and *PyModule_AddObjectRef()*; see their documentation for details.

PyModule_AddIntMacro(module, macro)

Add an int constant to *module*. The name and the value are taken from *macro*. For example `PyModule_AddIntMacro(module, AF_INET)` adds the int constant *AF_INET* with the value of *AF_INET* to *module*. Return `-1` with an exception set on error, `0` on success.

PyModule_AddStringMacro(module, macro)

*module*에 문자열 상수를 추가합니다.

int **PyModule_AddType**(*PyObject* *module, *PyTypeObject* *type)

Part of the Stable ABI *since version 3.10.* Add a type object to *module*. The type object is finalized by calling internally *PyType_Ready()*. The name of the type object is taken from the last component of *tp_name* after dot. Return `-1` with an exception set on error, `0` on success.

Added in version 3.9.

모듈 조회

단단계 초기화는 현재 인터프리터의 컨텍스트에서 조회할 수 있는 싱글톤 모듈을 만듭니다. 이는 나중에 모듈 정의에 대한 참조만으로 모듈 객체를 검색할 수 있도록 합니다.

이 함수들은 다단계 초기화를 사용하여 만들어진 모듈에서는 작동하지 않습니다. 단일 정의에서 그러한 모듈이 여러 개 만들어질 수 있기 때문입니다.

PyObject ***PyState_FindModule**(*PyModuleDef* *def)

Return value: Borrowed reference. *Part of the* Stable ABI. 현재 인터프리터에 대해 *def*에서 만들어진 모듈 객체를 반환합니다. 이 메서드를 사용하려면 먼저 모듈 객체가 `PyState_AddModule()`로 인터프리터 상태에 연결되어 있어야 합니다. 해당 모듈 객체를 찾을 수 없거나 인터프리터 상태에 아직 연결되지 않았으면, `NULL`을 반환합니다.

int **PyState_AddModule**(*PyObject* *module, *PyModuleDef* *def)

Part of the Stable ABI *since version 3.3.* 함수에 전달된 모듈 객체를 인터프리터 상태에 연결합니다. 이는 `PyState_FindModule()`을 통해 모듈 객체에 액세스 할 수 있도록 합니다.

단단계 초기화를 사용하여 만든 모듈에만 효과가 있습니다.

파이썬은 모듈을 임포트 한 후 자동으로 `PyState_AddModule`을 호출하므로, 모듈 초기화 코드에서 호출하는 것은 불필요합니다 (하지만 무해합니다). 모듈의 자체 초기화 코드가 추후 `PyState_FindModule`을 호출하는 경우에만 명시적인 호출이 필요합니다. 이 함수는 주로 대안 임포트 메커니즘을 구현하기 위한 것입니다 (직접 호출하거나, 필요한 상태 갱신에 대한 자세한 내용에 대해 해당 구현을 참조함으로써).

호출자는 GIL을 보유해야 합니다.

Return `-1` with an exception set on error, `0` on success.

Added in version 3.3.

int **PyState_RemoveModule**(*PyModuleDef* *def)

Part of the Stable ABI *since version 3.3.* Removes the module object created from *def* from the interpreter state. Return `-1` with an exception set on error, `0` on success.

호출자는 GIL을 보유해야 합니다.

Added in version 3.3.

8.6.3 이터레이터 객체

Python provides two general-purpose iterator objects. The first, a sequence iterator, works with an arbitrary sequence supporting the `__getitem__()` method. The second works with a callable object and a sentinel value, calling the callable for each item in the sequence, and ending the iteration when the sentinel value is returned.

PyTypeObject **PySeqIter_Type**

Part of the Stable ABI. `PySeqIter_New()`와 내장 시퀀스 형에 대한 `iter()` 내장 함수의 단일 인자 형식에 의해 반환된 이터레이터 객체에 대한 형 객체.

int **PySeqIter_Check**(*PyObject* *op)

*op*의 형이 `PySeqIter_Type`이면 참을 돌려줍니다. 이 함수는 항상 성공합니다.

PyObject ***PySeqIter_New**(*PyObject* *seq)

Return value: New reference. *Part of the* Stable ABI. 일반 시퀀스 객체 *seq*와 함께 작동하는 이터레이터를 반환합니다. 시퀀스가 서브스크립션 연산에서 `IndexError`를 일으키면 이터레이션이 끝납니다.

PyTypeObject **PyCallIter_Type**

Part of the Stable ABI. `PyCallIter_New()`와 `iter()` 내장 함수의 두 인자 형식에 의해 반환된 이터레이터 객체에 대한 형 객체.

int **PyCallIter_Check**(*PyObject* *op)

*op*의 형이 `PyCallIter_Type`이면 참을 돌려줍니다. 이 함수는 항상 성공합니다.

PyObject ***PyCallIter_New**(*PyObject* *callable, *PyObject* *sentinel)

Return value: New reference. Part of the Stable ABI. 새로운 이터레이터를 돌려줍니다. 첫 번째 매개 변수 *callable*은 매개 변수 없이 호출할 수 있는 모든 파이썬 콜러블 객체일 수 있습니다; 각 호출은 이터레이션의 다음 항목을 반환해야 합니다. *callable*이 *sentinel*와 같은 값을 반환하면 이터레이션이 종료됩니다.

8.6.4 디스크립터 객체

"디스크립터"는 객체의 일부 어트리뷰트를 기술하는 객체입니다. 그것들은 형 객체의 딕셔너리에 있습니다.

PyTypeObject **PyProperty_Type**

Part of the Stable ABI. 내장 디스크립터 형들을 위한 형 객체.

PyObject ***PyDescr_NewGetSet**(*PyTypeObject* *type, struct *PyGetSetDef* *getset)

Return value: New reference. Part of the Stable ABI.

PyObject ***PyDescr_NewMember**(*PyTypeObject* *type, struct *PyMemberDef* *meth)

Return value: New reference. Part of the Stable ABI.

PyObject ***PyDescr_NewMethod**(*PyTypeObject* *type, struct *PyMethodDef* *meth)

Return value: New reference. Part of the Stable ABI.

PyObject ***PyDescr_NewWrapper**(*PyTypeObject* *type, struct wrapperbase *wrapper, void *wrapped)

Return value: New reference.

PyObject ***PyDescr_NewClassMethod**(*PyTypeObject* *type, *PyMethodDef* *method)

Return value: New reference. Part of the Stable ABI.

int **PyDescr_IsData**(*PyObject* *descr)

Return non-zero if the descriptor objects *descr* describes a data attribute, or `0` if it describes a method. *descr* must be a descriptor object; there is no error checking.

PyObject ***PyWrapper_New**(*PyObject**, *PyObject**)

Return value: New reference. Part of the Stable ABI.

8.6.5 슬라이스 객체

PyTypeObject **PySlice_Type**

Part of the Stable ABI. 슬라이스 객체의 형 객체. 이것은 파이썬 계층의 `slice`와 같습니다.

int **PySlice_Check**(*PyObject* *ob)

*ob*가 슬라이스 객체면 참을 반환합니다. *ob*는 `NULL`이 아니어야 합니다. 이 함수는 항상 성공합니다.

PyObject ***PySlice_New**(*PyObject* *start, *PyObject* *stop, *PyObject* *step)

Return value: New reference. Part of the Stable ABI. Return a new slice object with the given values. The *start*, *stop*, and *step* parameters are used as the values of the slice object attributes of the same names. Any of the values may be `NULL`, in which case the `None` will be used for the corresponding attribute.

Return `NULL` with an exception set if the new object could not be allocated.

int **PySlice_GetIndices**(*PyObject* *slice, *Py_ssize_t* length, *Py_ssize_t* *start, *Py_ssize_t* *stop, *Py_ssize_t* *step)

Part of the Stable ABI. 길이가 *length*인 시퀀스를 가정하여, 슬라이스 객체 *slice*에서 start, stop 및 step 인덱스를 가져옵니다. *length*보다 큰 인덱스를 에러로 처리합니다.

Returns `0` on success and `-1` on error with no exception set (unless one of the indices was not `None` and failed to be converted to an integer, in which case `-1` is returned with an exception set).

이 기능을 사용하고 싶지는 않을 것입니다.

버전 3.2에서 변경: 전에는 *slice* 매개 변수의 매개 변수 형이 `PySliceObject*`였습니다.

int **PySlice_GetIndicesEx**(*PyObject* *slice, *Py_ssize_t* length, *Py_ssize_t* *start, *Py_ssize_t* *stop, *Py_ssize_t* *step, *Py_ssize_t* *slicelength)

Part of the Stable ABI. `PySlice_GetIndices()`를 쓸만하게 대체합니다. 길이가 *length*인 시퀀스를 가정하여, 슬라이스 객체 *slice*에서 start, stop 및 step 인덱스를 가져오고, *slicelength*에 슬라이스의 길이를 저장합니다. 범위를 벗어난 인덱스는 일반 슬라이스의 처리와 일관된 방식으로 잘립니다.

Return `0` on success and `-1` on error with an exception set.

참고

이 함수는 크기를 조정할 수 있는 시퀀스에는 안전하지 않은 것으로 간주합니다. 호출은 `PySlice_Unpack()`와 `PySlice_AdjustIndices()`의 조합으로 대체되어야 합니다. 즉

```
if (PySlice_GetIndicesEx(slice, length, &start, &stop, &step, &slicelength)
↪< 0) {
    // return error
}
```

은 다음으로 대체됩니다

```
if (PySlice_Unpack(slice, &start, &stop, &step) < 0) {
    // return error
}
slicelength = PySlice_AdjustIndices(length, &start, &stop, step);
```

버전 3.2에서 변경: 전에는 *slice* 매개 변수의 매개 변수 형이 `PySliceObject*`였습니다.

버전 3.6.1에서 변경: `Py_LIMITED_API`가 설정되어 있지 않거나 `0x03050400`과 `0x03060000` (포함하지 않음) 사이나 `0x03060100` 이상의 값으로 설정되었으면, `PySlice_GetIndicesEx()`는 `PySlice_Unpack()`과 `PySlice_AdjustIndices()`를 사용하는 매크로로 구현됩니다. 인자 *start*, *stop* 및 *step*는 여러 번 평가됩니다.

버전 3.6.1부터 폐지됨: `Py_LIMITED_API`가 `0x03050400`보다 작거나 `0x03060000`과 `0x03060100` (포함하지 않음) 사이의 값으로 설정되었으면 `PySlice_GetIndicesEx()`는 폐지된 함수입니다.

int **PySlice_Unpack**(*PyObject* *slice, *Py_ssize_t* *start, *Py_ssize_t* *stop, *Py_ssize_t* *step)

Part of the Stable ABI *since version 3.7.* 슬라이스 객체의 start, stop 및 step 데이터 멤버를 C 정수로 추출합니다. `PY_SSIZE_T_MAX`보다 큰 값을 `PY_SSIZE_T_MAX`로 조용히 줄이고, `PY_SSIZE_T_MIN`보다 작은 start 와 stop 값을 `PY_SSIZE_T_MIN`로 조용히 높이고, `-PY_SSIZE_T_MAX`보다 작은 step 값을 `-PY_SSIZE_T_MAX`로 조용히 높입니다.

Return `-1` with an exception set on error, `0` on success.

Added in version 3.6.1.

Py_ssize_t **PySlice_AdjustIndices**(*Py_ssize_t* length, *Py_ssize_t* *start, *Py_ssize_t* *stop, *Py_ssize_t* step)

Part of the Stable ABI *since version 3.7.* 지정된 length 길이의 시퀀스를 가정하여 start/stop 슬라이스 인덱스를 조정합니다. 범위를 벗어난 인덱스는 일반 슬라이스의 처리와 일관된 방식으로 잘립니다.

슬라이스의 길이를 반환합니다. 항상 성공합니다. 파이썬 코드를 호출하지 않습니다.

Added in version 3.6.1.

Ellipsis 객체

PyTypeObject **PyEllipsis_Type**

Part of the Stable ABI. The type of Python `Ellipsis` object. Same as `types.EllipsisType` in the Python layer.

PyObject ***Py_Ellipsis**

The Python `Ellipsis` object. This object has no methods. Like `Py_None`, it is an immortal. singleton object.

버전 3.12에서 변경: `Py_Ellipsis` is immortal.

8.6.6 MemoryView 객체

`memoryview` 객체는 C 수준 버퍼 인터페이스를 다른 객체와 마찬가지로 전달될 수 있는 파이썬 객체로 노출합니다.

PyObject ***PyMemoryView_FromObject**(*PyObject* *obj)

Return value: New reference. Part of the Stable ABI. 버퍼 인터페이스를 제공하는 객체에서 memoryview 객체를 만듭니다. *obj*가 쓰기 가능한 버퍼 제공을 지원하면, memoryview 객체는 읽기/쓰기가 되고, 그렇지 않으면 읽기 전용이거나 제공자의 재량에 따라 읽기/쓰기가 될 수 있습니다.

PyBUF_READ

Flag to request a readonly buffer.

PyBUF_WRITE

Flag to request a writable buffer.

PyObject ***PyMemoryView_FromMemory**(char *mem, *Py_ssize_t* size, int flags)

Return value: New reference. Part of the Stable ABI *since version 3.7. mem*를 하부 버퍼로 사용하여 memoryview 객체를 만듭니다. *flags*는 *PyBUF_READ* 나 *PyBUF_WRITE* 중 하나일 수 있습니다.

Added in version 3.3.

PyObject ***PyMemoryView_FromBuffer**(const *Py_buffer* *view)

Return value: New reference. Part of the Stable ABI *since version 3.11.* 주어진 버퍼 구조체 *view*를 감싸는 memoryview 객체를 만듭니다. 간단한 바이트 버퍼의 경우는, *PyMemoryView_FromMemory()*가 선호되는 함수입니다.

PyObject ***PyMemoryView_GetContiguous**(*PyObject* *obj, int buffertype, char order)

Return value: New reference. Part of the Stable ABI. 버퍼 인터페이스를 정의하는 객체로부터 메모리의 연속 청크('C' 나 'F' ortran *order*로)로 memoryview 객체를 만듭니다. 메모리가 연속적이면 memoryview 객체는 원래 메모리를 가리킵니다. 그렇지 않으면, 복사본이 만들어지고 memoryview는 새 바이트열 객체를 가리킵니다.

buffertype can be one of *PyBUF_READ* or *PyBUF_WRITE*.

int **PyMemoryView_Check**(*PyObject* *obj)

객체 *obj*가 memoryview 객체면 참을 반환합니다. 현재는 `memoryview`의 서브 클래스를 만들 수 없습니다. 이 함수는 항상 성공합니다.

Py_buffer ***PyMemoryView_GET_BUFFER**(*PyObject* *mview)

제공자 버퍼의 memoryview의 비공개 복사본의 포인터를 돌려줍니다. *mview*는 **반드시** memoryview 인스턴스여야 합니다; 이 매크로는 형을 확인하지 않으므로 직접 검사해야 합니다, 그렇지 않으면 충돌 위험이 있습니다.

PyObject ***PyMemoryView_GET_BASE**(*PyObject* *mview)

memoryview가 기반으로 하는 제공자 객체에 대한 포인터나 memoryview가 *PyMemoryView_FromMemory()* 나 *PyMemoryView_FromBuffer()* 함수 중 하나로 만들어졌으면 `NULL`을 반환합니다. *mview*는 **반드시** memoryview 인스턴스여야 합니다.

8.6.7 약한 참조 객체

파이썬은 약한 참조를 1급 객체로 지원합니다. 약한 참조를 직접 구현하는 두 가지 구체적인 객체 형이 있습니다. 첫 번째는 간단한 참조 객체이며, 두 번째는 가능한 한 원래 객체의 프락시 역할을 합니다.

int **PyWeakref_Check**(*PyObject* *ob)

*ob*가 참조 객체나 프락시 객체면 참을 반환합니다. 이 함수는 항상 성공합니다.

int **PyWeakref_CheckRef**(*PyObject* *ob)

*ob*가 참조 객체면 참을 반환합니다. 이 함수는 항상 성공합니다.

int **PyWeakref_CheckProxy**(*PyObject* *ob)

*ob*가 프락시 객체면 참을 반환합니다. 이 함수는 항상 성공합니다.

PyObject ***PyWeakref_NewRef**(*PyObject* *ob, *PyObject* *callback)

Return value: New reference. Part of the Stable ABI. Return a weak reference object for the object *ob*. This will always return a new reference, but is not guaranteed to create a new object; an existing reference object may be returned. The second parameter, *callback*, can be a callable object that receives notification when *ob* is garbage collected; it should accept a single parameter, which will be the weak reference object itself. *callback* may also be `None` or `NULL`. If *ob* is not a weakly referenceable object, or if *callback* is not callable, `None`, or `NULL`, this will return `NULL` and raise `TypeError`.

PyObject ***PyWeakref_NewProxy**(*PyObject* *ob, *PyObject* *callback)

Return value: New reference. Part of the Stable ABI. Return a weak reference proxy object for the object *ob*. This will always return a new reference, but is not guaranteed to create a new object; an existing proxy object may be returned. The second parameter, *callback*, can be a callable object that receives notification when *ob* is garbage collected; it should accept a single parameter, which will be the weak reference object itself. *callback* may also be `None` or `NULL`. If *ob* is not a weakly referenceable object, or if *callback* is not callable, `None`, or `NULL`, this will return `NULL` and raise `TypeError`.

PyObject ***PyWeakref_GetObject**(*PyObject* *ref)

Return value: Borrowed reference. Part of the Stable ABI. Return the referenced object from a weak reference, *ref*. If the referent is no longer live, returns `Py_None`.

> **참고**
>
> This function returns a *borrowed reference* to the referenced object. This means that you should always call *Py_INCREF()* on the object except when it cannot be destroyed before the last usage of the borrowed reference.

PyObject ***PyWeakref_GET_OBJECT**(*PyObject* *ref)

Return value: Borrowed reference. Similar to *PyWeakref_GetObject()*, but does no error checking.

void **PyObject_ClearWeakRefs**(*PyObject* *object)

Part of the Stable ABI. This function is called by the *tp_dealloc* handler to clear weak references.

This iterates through the weak references for *object* and calls callbacks for those references which have one. It returns when all callbacks have been attempted.

8.6.8 캡슐

이 객체 사용에 대한 자세한 정보는 using-capsules를 참조하십시오.

Added in version 3.1.

type **PyCapsule**

This subtype of *PyObject* represents an opaque value, useful for C extension modules who need to pass an opaque value (as a `void*` pointer) through Python code to other C code. It is often used to make a C function pointer defined in one module available to other modules, so the regular import mechanism can be used to access C APIs defined in dynamically loaded modules.

type **PyCapsule_Destructor**

Part of the Stable ABI. 캡슐에 대한 파괴자(destructor) 콜백 형. 이렇게 정의됩니다:

```
typedef void (*PyCapsule_Destructor)(PyObject *);
```

PyCapsule_Destructor 콜백의 의미는 *PyCapsule_New()*를 참조하십시오.

int **PyCapsule_CheckExact**(*PyObject* *p)

인자가 *PyCapsule*이면 참을 돌려줍니다. 이 함수는 항상 성공합니다.

PyObject ***PyCapsule_New**(void *pointer, const char *name, *PyCapsule_Destructor* destructor)

Return value: New reference. Part of the Stable ABI. *pointer*를 캡슐화하는 *PyCapsule*을 만듭니다. *pointer* 인자는 `NULL`이 아닐 수도 있습니다.

실패하면, 예외를 설정하고 NULL을 반환합니다.

name 문자열은 NULL 이나 유효한 C 문자열에 대한 포인터일 수 있습니다. NULL이 아니면, 이 문자열은 캡슐보다 오래 유지되어야 합니다. (*destructor* 내부에서 해제할 수는 있습니다.)

destructor 인자가 NULL이 아니면, 캡슐이 파괴될 때 캡슐을 인자로 호출됩니다.

이 캡슐을 모듈의 어트리뷰트로 저장하려면, *name*을 `modulename.attributename`로 지정해야 합니다. 이렇게 하면 다른 모듈이 *PyCapsule_Import()*를 사용하여 캡슐을 임포트 할 수 있습니다.

void ***PyCapsule_GetPointer**(*PyObject* *capsule, const char *name)

Part of the Stable ABI. 캡슐에 저장된 *pointer*를 가져옵니다. 실패하면, 예외를 설정하고 NULL을 반환합니다.

The *name* parameter must compare exactly to the name stored in the capsule. If the name stored in the capsule is NULL, the *name* passed in must also be NULL. Python uses the C function `strcmp()` to compare capsule names.

PyCapsule_Destructor ***PyCapsule_GetDestructor**(*PyObject* *capsule)

Part of the Stable ABI. 캡슐에 저장된 현재 파괴자를 반환합니다. 실패하면, 예외를 설정하고 NULL을 반환합니다.

캡슐이 NULL 파괴자를 갖는 것은 합법적입니다. 이것은 NULL 반환 코드를 다소 모호하게 만듭니다; 명확히 하려면 *PyCapsule_IsValid()* 나 *PyErr_Occurred()*를 사용하십시오.

void ***PyCapsule_GetContext**(*PyObject* *capsule)

Part of the Stable ABI. 캡슐에 저장된 현재 컨텍스트를 반환합니다. 실패하면, 예외를 설정하고 NULL을 반환합니다.

캡슐이 NULL 컨텍스트를 갖는 것은 합법적입니다. 이것은 NULL 반환 코드를 다소 모호하게 만듭니다; 명확히 하려면 *PyCapsule_IsValid()* 나 *PyErr_Occurred()*를 사용하십시오.

const char ***PyCapsule_GetName**(*PyObject* *capsule)

Part of the Stable ABI. 캡슐에 저장된 현재 이름을 반환합니다. 실패하면, 예외를 설정하고 NULL을 반환합니다.

캡슐이 NULL 이름을 갖는 것은 합법적입니다. 이것은 NULL 반환 코드를 다소 모호하게 만듭니다; 명확히 하려면 *PyCapsule_IsValid()* 나 *PyErr_Occurred()*를 사용하십시오.

void ***PyCapsule_Import**(const char *name, int no_block)

Part of the Stable ABI. Import a pointer to a C object from a capsule attribute in a module. The *name* parameter should specify the full name to the attribute, as in `module.attribute`. The *name* stored in the capsule must match this string exactly.

성공하면 캡슐의 내부 *pointer*를 반환합니다. 실패하면, 예외를 설정하고 NULL를 반환합니다.

버전 3.3에서 변경: *no_block* has no effect anymore.

int **PyCapsule_IsValid**(*PyObject* *capsule, const char *name)

Part of the Stable ABI. *capsule*이 유효한 캡슐인지를 판단합니다. 유효한 캡슐은 NULL이 아니며, *PyCapsule_CheckExact()*를 통과하고, NULL이 아닌 포인터가 저장되며, 내부 이름이 *name* 매개변수와 일치합니다. (캡슐 이름을 비교하는 방법에 대한 정보는 *PyCapsule_GetPointer()*를 참조하십시오.)

In other words, if *PyCapsule_IsValid()* returns a true value, calls to any of the accessors (any function starting with `PyCapsule_Get`) are guaranteed to succeed.

객체가 유효하고 전달된 이름과 일치하면 0이 아닌 값을 반환합니다. 그렇지 않으면 0을 반환합니다. 이 함수는 실패하지 않습니다.

int **PyCapsule_SetContext**(*PyObject* *capsule, void *context)

Part of the Stable ABI. *capsule* 내부의 컨텍스트 포인터를 *context*로 설정합니다.

성공하면 0을 반환합니다. 실패하면 0이 아닌 값을 반환하고 예외를 설정합니다.

int **PyCapsule_SetDestructor**(*PyObject* *capsule, *PyCapsule_Destructor* destructor)

Part of the Stable ABI. *capsule* 내부의 파괴자를 *destructor*로 설정합니다.

성공하면 `0`을 반환합니다. 실패하면 0이 아닌 값을 반환하고 예외를 설정합니다.

int **PyCapsule_SetName**(*PyObject* *capsule, const char *name)

Part of the Stable ABI. *capsule* 내부의 이름을 *name*으로 설정합니다. `NULL`이 아니면, 이름은 캡슐보다 오래 유지되어야 합니다. 캡슐에 저장된 이전 *name*이 `NULL`이 아니면, 이를 해제하려고 시도하지 않습니다.

성공하면 `0`을 반환합니다. 실패하면 0이 아닌 값을 반환하고 예외를 설정합니다.

int **PyCapsule_SetPointer**(*PyObject* *capsule, void *pointer)

Part of the Stable ABI. *capsule* 내부의 void 포인터를 *pointer*로 설정합니다. 포인터는 `NULL`이 아닐 수 있습니다.

성공하면 `0`을 반환합니다. 실패하면 0이 아닌 값을 반환하고 예외를 설정합니다.

8.6.9 Frame Objects

type **PyFrameObject**

Part of the Limited API (*as an opaque struct*). The C structure of the objects used to describe frame objects.

There are no public members in this structure.

버전 3.11에서 변경: The members of this structure were removed from the public C API. Refer to the What's New entry for details.

The *PyEval_GetFrame()* and *PyThreadState_GetFrame()* functions can be used to get a frame object.

See also *Reflection*.

PyTypeObject **PyFrame_Type**

The type of frame objects. It is the same object as `types.FrameType` in the Python layer.

버전 3.11에서 변경: Previously, this type was only available after including `<frameobject.h>`.

int **PyFrame_Check**(*PyObject* *obj)

Return non-zero if *obj* is a frame object.

버전 3.11에서 변경: Previously, this function was only available after including `<frameobject.h>`.

PyFrameObject ***PyFrame_GetBack**(*PyFrameObject* *frame)

Get the *frame* next outer frame.

Return a *strong reference*, or `NULL` if *frame* has no outer frame.

Added in version 3.9.

PyObject ***PyFrame_GetBuiltins**(*PyFrameObject* *frame)

Get the *frame*'s `f_builtins` attribute.

Return a *strong reference*. The result cannot be `NULL`.

Added in version 3.11.

PyCodeObject ***PyFrame_GetCode**(*PyFrameObject* *frame)

Part of the Stable ABI *since version 3.10.* Get the *frame* code.

Return a *strong reference*.

The result (frame code) cannot be `NULL`.

Added in version 3.9.

PyObject ***PyFrame_GetGenerator**(*PyFrameObject* *frame)

Get the generator, coroutine, or async generator that owns this frame, or `NULL` if this frame is not owned by a generator. Does not raise an exception, even if the return value is `NULL`.

Return a *strong reference*, or `NULL`.

Added in version 3.11.

PyObject ***PyFrame_GetGlobals**(*PyFrameObject* *frame)

Get the *frame*' s `f_globals` attribute.

Return a *strong reference*. The result cannot be `NULL`.

Added in version 3.11.

int **PyFrame_GetLasti**(*PyFrameObject* *frame)

Get the *frame*' s `f_lasti` attribute.

Returns -1 if `frame.f_lasti` is `None`.

Added in version 3.11.

PyObject ***PyFrame_GetVar**(*PyFrameObject* *frame, *PyObject* *name)

Get the variable *name* of *frame*.

- Return a *strong reference* to the variable value on success.
- Raise `NameError` and return `NULL` if the variable does not exist.
- Raise an exception and return `NULL` on error.

name type must be a `str`.

Added in version 3.12.

PyObject ***PyFrame_GetVarString**(*PyFrameObject* *frame, const char *name)

Similar to *PyFrame_GetVar()*, but the variable name is a C string encoded in UTF-8.

Added in version 3.12.

PyObject ***PyFrame_GetLocals**(*PyFrameObject* *frame)

Get the *frame*' s `f_locals` attribute (`dict`).

Return a *strong reference*.

Added in version 3.11.

int **PyFrame_GetLineNumber**(*PyFrameObject* *frame)

Part of the Stable ABI *since version 3.10.* Return the line number that *frame* is currently executing.

Internal Frames

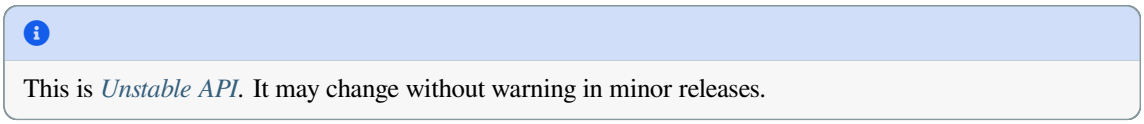
Unless using **PEP 523**, you will not need this.

struct **_PyInterpreterFrame**

The interpreter' s internal frame representation.

Added in version 3.11.

PyObject ***PyUnstable_InterpreterFrame_GetCode**(struct *_PyInterpreterFrame* *frame);

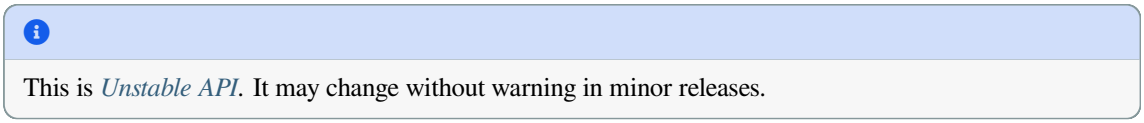


This is *Unstable API*. It may change without warning in minor releases.

Return a *strong reference* to the code object for the frame.

Added in version 3.12.

int **PyUnstable_InterpreterFrame_GetLasti**(struct *_PyInterpreterFrame* *frame);

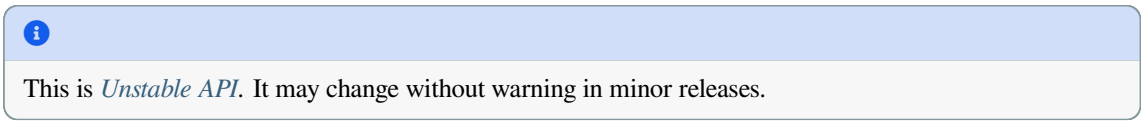


This is *Unstable API*. It may change without warning in minor releases.

Return the byte offset into the last executed instruction.

Added in version 3.12.

int **PyUnstable_InterpreterFrame_GetLine**(struct *_PyInterpreterFrame* *frame);



This is *Unstable API*. It may change without warning in minor releases.

Return the currently executing line number, or -1 if there is no line number.

Added in version 3.12.

8.6.10 제너레이터 객체

제너레이터 객체는 파이썬이 제너레이터 이터레이터를 구현하기 위해 사용하는 객체입니다. 일반적으로 `PyGen_New()` 또는 `PyGen_NewWithQualName()`를 명시적으로 호출하는 것이 아니라, 값을 일드(yield)하는 함수를 이터레이트하여 만들어집니다.

type **PyGenObject**

제너레이터 객체에 사용되는 C 구조체.

PyTypeObject **PyGen_Type**

제너레이터 객체에 해당하는 형 객체

int **PyGen_Check**(*PyObject* *ob)

*ob*가 제너레이터 객체면 참을 돌려줍니다; *ob*는 `NULL`이 아니어야 합니다. 이 함수는 항상 성공합니다.

int **PyGen_CheckExact**(*PyObject* *ob)

*ob*의 형이 `PyGen_Type`이면 참을 돌려줍니다; *ob*는 `NULL`이 아니어야 합니다. 이 함수는 항상 성공합니다.

PyObject ***PyGen_New**(*PyFrameObject* *frame)

Return value: New reference. *frame* 객체에 기반한 새 제너레이터 객체를 만들어 반환합니다. 이 함수는 *frame*에 대한 참조를 훔칩니다. 인자는 `NULL`이 아니어야 합니다.

PyObject ***PyGen_NewWithQualName**(*PyFrameObject* *frame, *PyObject* *name, *PyObject* *qualname)

Return value: New reference. *frame* 객체에 기반한 새 제너레이터 객체를 만들어 반환하는데, `__name__`과 `__qualname__`를 *name* 및 *qualname*로 설정합니다. 이 함수는 *frame*에 대한 참조를 훔칩니다. *frame* 인자는 `NULL`이 아니어야 합니다.

8.6.11 코루틴 객체

Added in version 3.5.

코루틴 객체는 `async` 키워드로 선언된 함수가 반환하는 것입니다.

type **PyCoroObject**

코루틴 객체에 사용되는 C 구조체.

PyTypeObject **PyCoro_Type**

코루틴 객체에 해당하는 형 객체.

int **PyCoro_CheckExact**(*PyObject* *ob)

*ob*의 형이 *PyCoro_Type*이면 참을 반환합니다. *ob*는 NULL일 수 없습니다. 이 함수는 항상 성공합니다.

PyObject ***PyCoro_New**(*PyFrameObject* *frame, *PyObject* *name, *PyObject* *qualname)

Return value: New reference. *frame* 객체를 기반으로 새 코루틴 객체를 만들어서 반환합니다. __name__과 __qualname__은 *name*과 *qualname*로 설정합니다. 이 함수는 *frame*에 대한 참조를 훔칩니다. *frame* 인자는 NULL일 수 없습니다.

8.6.12 컨텍스트 변수 객체

Added in version 3.7.

버전 3.7.1에서 변경:

참고

파이썬 3.7.1에서 모든 컨텍스트 변수 C API의 서명이 *PyContext*, *PyContextVar* 및 *PyContextToken* 대신 *PyObject* 포인터를 사용하도록 **변경되었습니다**, 예를 들어:

```
// in 3.7.0:
PyContext *PyContext_New(void);

// in 3.7.1+:
PyObject *PyContext_New(void);
```

자세한 내용은 bpo-34762를 참조하십시오.

이 절에서는 contextvars 모듈을 위한 공용 C API에 대해 자세히 설명합니다.

type **PyContext**

contextvars.Context 객체를 나타내는 데 사용되는 C 구조체.

type **PyContextVar**

contextvars.ContextVar 객체를 나타내는 데 사용되는 C 구조체.

type **PyContextToken**

contextvars.Token 객체를 나타내는 데 사용되는 C 구조체.

PyTypeObject **PyContext_Type**

context 형을 나타내는 형 객체.

PyTypeObject **PyContextVar_Type**

컨텍스트 변수 형을 나타내는 형 객체.

PyTypeObject **PyContextToken_Type**

컨텍스트 변수 토큰 형을 나타내는 형 객체.

형 검사 매크로:

int **PyContext_CheckExact**(*PyObject* *o)

*o*가 *PyContext_Type* 형이면 참을 돌려줍니다. *o*는 NULL이 아니어야 합니다. 이 함수는 항상 성공합니다.

int **PyContextVar_CheckExact**(*PyObject* *o)

*o*가 *PyContextVar_Type* 형이면 참을 돌려줍니다. *o*는 NULL이 아니어야 합니다. 이 함수는 항상 성공합니다.

int **PyContextToken_CheckExact**(*PyObject* *o)

*o*가 *PyContextToken_Type* 형이면 참을 돌려줍니다. *o*는 NULL이 아니어야 합니다. 이 함수는 항상 성공합니다.

컨텍스트 객체 관리 함수:

PyObject ***PyContext_New**(void)

Return value: New reference. 새로운 빈 컨텍스트 객체를 만듭니다. 에러가 발생하면 NULL를 반환합니다.

PyObject ***PyContext_Copy**(*PyObject* *ctx)

Return value: New reference. 전달된 *ctx* 컨텍스트 객체의 얕은 복사본을 만듭니다. 에러가 발생하면 NULL을 반환합니다.

PyObject ***PyContext_CopyCurrent**(void)

Return value: New reference. 현재 스레드 컨텍스트의 얕은 복사본을 만듭니다. 에러가 발생하면 NULL을 반환합니다.

int **PyContext_Enter**(*PyObject* *ctx)

현재 스레드의 현재 컨텍스트로 *ctx*를 설정합니다. 성공 시 0을 반환하고, 에러 시 -1을 반환합니다.

int **PyContext_Exit**(*PyObject* *ctx)

ctx 컨텍스트를 비활성화하고 이전 컨텍스트를 현재 스레드의 현재 컨텍스트로 복원합니다. 성공 시 0을 반환하고, 에러 시 -1을 반환합니다.

컨텍스트 변수 함수:

PyObject ***PyContextVar_New**(const char *name, *PyObject* *def)

Return value: New reference. Create a new ContextVar object. The *name* parameter is used for introspection and debug purposes. The *def* parameter specifies a default value for the context variable, or NULL for no default. If an error has occurred, this function returns NULL.

int **PyContextVar_Get**(*PyObject* *var, *PyObject* *default_value, *PyObject* **value)

컨텍스트 변수의 값을 가져옵니다. 조회하는 동안 에러가 발생하면 -1을 반환하고, 값이 있는지와 상관없이 에러가 발생하지 않으면 0을 반환합니다.

컨텍스트 변수가 발견되면, *value*는 그것을 가리키는 포인터가 됩니다. 컨텍스트 변수가 발견되지 않으면, *value*는 다음을 가리 킵니다:

- *default_value*, NULL이 아니면;
- *var*의 기본값, NULL이 아니면;
- NULL

Except for NULL, the function returns a new reference.

PyObject ***PyContextVar_Set**(*PyObject* *var, *PyObject* *value)

Return value: New reference. Set the value of *var* to *value* in the current context. Returns a new token object for this change, or NULL if an error has occurred.

int **PyContextVar_Reset**(*PyObject* *var, *PyObject* *token)

var 컨텍스트 변수의 상태를 *token*을 반환한 *PyContextVar_Set()* 호출 전의 상태로 재설정합니다. 이 함수는 성공 시 0을 반환하고, 에러 시 -1을 반환합니다.

8.6.13 DateTime 객체

Various date and time objects are supplied by the datetime module. Before using any of these functions, the header file datetime.h must be included in your source (note that this is not included by Python.h), and the macro PyDateTime_IMPORT must be invoked, usually as part of the module initialisation function. The macro puts a pointer to a C structure into a static variable, PyDateTimeAPI, that is used by the following macros.

type **PyDateTime_Date**

This subtype of *PyObject* represents a Python date object.

type **PyDateTime_DateTime**

This subtype of *PyObject* represents a Python datetime object.

type **PyDateTime_Time**

This subtype of *PyObject* represents a Python time object.

type **PyDateTime_Delta**

This subtype of *PyObject* represents the difference between two datetime values.

PyTypeObject **PyDateTime_DateType**

This instance of *PyTypeObject* represents the Python date type; it is the same object as `datetime.date` in the Python layer.

PyTypeObject **PyDateTime_DateTimeType**

This instance of *PyTypeObject* represents the Python datetime type; it is the same object as `datetime.datetime` in the Python layer.

PyTypeObject **PyDateTime_TimeType**

This instance of *PyTypeObject* represents the Python time type; it is the same object as `datetime.time` in the Python layer.

PyTypeObject **PyDateTime_DeltaType**

This instance of *PyTypeObject* represents Python type for the difference between two datetime values; it is the same object as `datetime.timedelta` in the Python layer.

PyTypeObject **PyDateTime_TZInfoType**

This instance of *PyTypeObject* represents the Python time zone info type; it is the same object as `datetime.tzinfo` in the Python layer.

UTC 싱글톤에 액세스하기 위한 매크로:

PyObject ***PyDateTime_TimeZone_UTC**

UTC를 나타내는 시간대 싱글톤을 반환합니다, `datetime.timezone.utc`와 같은 객체입니다.

Added in version 3.7.

형 검사 매크로:

int **PyDate_Check**(*PyObject* *ob)

Return true if *ob* is of type *PyDateTime_DateType* or a subtype of `PyDateTime_DateType`. *ob* must not be `NULL`. This function always succeeds.

int **PyDate_CheckExact**(*PyObject* *ob)

*ob*가 *PyDateTime_DateType* 형이면 참을 돌려줍니다. *ob*는 `NULL`이 아니어야 합니다. 이 함수는 항상 성공합니다.

int **PyDateTime_Check**(*PyObject* *ob)

Return true if *ob* is of type *PyDateTime_DateTimeType* or a subtype of `PyDateTime_DateTimeType`. *ob* must not be `NULL`. This function always succeeds.

int **PyDateTime_CheckExact**(*PyObject* *ob)

*ob*가 *PyDateTime_DateTimeType* 형이면 참을 돌려줍니다. *ob*는 `NULL`이 아니어야 합니다. 이 함수는 항상 성공합니다.

int **PyTime_Check**(*PyObject* *ob)

Return true if *ob* is of type *PyDateTime_TimeType* or a subtype of `PyDateTime_TimeType`. *ob* must not be `NULL`. This function always succeeds.

int **PyTime_CheckExact**(*PyObject* *ob)

*ob*가 *PyDateTime_TimeType* 형이면 참을 돌려줍니다. *ob*는 `NULL`이 아니어야 합니다. 이 함수는 항상 성공합니다.

int **PyDelta_Check**(*PyObject* *ob)

Return true if *ob* is of type *PyDateTime_DeltaType* or a subtype of PyDateTime_DeltaType. *ob* must not be NULL. This function always succeeds.

int **PyDelta_CheckExact**(*PyObject* *ob)

*ob*가 *PyDateTime_DeltaType* 형이면 참을 돌려줍니다. *ob*는 NULL이 아니어야 합니다. 이 함수는 항상 성공합니다.

int **PyTZInfo_Check**(*PyObject* *ob)

Return true if *ob* is of type *PyDateTime_TZInfoType* or a subtype of PyDateTime_TZInfoType. *ob* must not be NULL. This function always succeeds.

int **PyTZInfo_CheckExact**(*PyObject* *ob)

*ob*가 *PyDateTime_TZInfoType* 형이면 참을 돌려줍니다. *ob*는 NULL이 아니어야 합니다. 이 함수는 항상 성공합니다.

객체를 만드는 매크로:

PyObject ***PyDate_FromDate**(int year, int month, int day)

Return value: New reference. 지정된 년, 월, 일의 datetime.date 객체를 반환합니다.

PyObject ***PyDateTime_FromDateAndTime**(int year, int month, int day, int hour, int minute, int second, int usecond)

Return value: New reference. 지정된 년, 월, 일, 시, 분, 초 및 마이크로초의 datetime.datetime 객체를 반환합니다.

PyObject ***PyDateTime_FromDateAndTimeAndFold**(int year, int month, int day, int hour, int minute, int second, int usecond, int fold)

Return value: New reference. 지정된 년, 월, 일, 시, 분, 초, 마이크로초 및 fold의 datetime.datetime 객체를 반환합니다.

Added in version 3.6.

PyObject ***PyTime_FromTime**(int hour, int minute, int second, int usecond)

Return value: New reference. 지정된 시, 분, 초 및 마이크로초의 datetime.time 객체를 반환합니다.

PyObject ***PyTime_FromTimeAndFold**(int hour, int minute, int second, int usecond, int fold)

Return value: New reference. 지정된 시, 분, 초, 마이크로초 및 fold의 datetime.time 객체를 반환합니다.

Added in version 3.6.

PyObject ***PyDelta_FromDSU**(int days, int seconds, int useconds)

Return value: New reference. 지정된 일, 초 및 마이크로초 수를 나타내는 datetime.timedelta 객체를 반환합니다. 결과 마이크로초와 초가 datetime.timedelta 객체에 관해 설명된 범위에 있도록 정규화가 수행됩니다.

PyObject ***PyTimeZone_FromOffset**(*PyObject* *offset)

Return value: New reference. *offset* 인자로 나타내지는 이름이 없는 고정 오프셋의 datetime.timezone 객체를 돌려줍니다.

Added in version 3.7.

PyObject ***PyTimeZone_FromOffsetAndName**(*PyObject* *offset, *PyObject* *name)

Return value: New reference. *offset* 인자와 tzname *name*으로 나타내지는 고정 오프셋의 datetime.timezone 객체를 돌려줍니다.

Added in version 3.7.

Macros to extract fields from date objects. The argument must be an instance of *PyDateTime_Date*, including subclasses (such as *PyDateTime_DateTime*). The argument must not be NULL, and the type is not checked:

int **PyDateTime_GET_YEAR**(*PyDateTime_Date* *o)

양의 int로, 년을 반환합니다.

int **PyDateTime_GET_MONTH**(*PyDateTime_Date* *o)
1에서 12까지의 int로, 월을 반환합니다.

int **PyDateTime_GET_DAY**(*PyDateTime_Date* *o)
1에서 31까지의 int로, 일을 반환합니다.

Macros to extract fields from datetime objects. The argument must be an instance of *PyDateTime_DateTime*, including subclasses. The argument must not be `NULL`, and the type is not checked:

int **PyDateTime_DATE_GET_HOUR**(*PyDateTime_DateTime* *o)
0부터 23까지의 int로, 시를 반환합니다.

int **PyDateTime_DATE_GET_MINUTE**(*PyDateTime_DateTime* *o)
0부터 59까지의 int로, 분을 반환합니다.

int **PyDateTime_DATE_GET_SECOND**(*PyDateTime_DateTime* *o)
0부터 59까지의 int로, 초를 반환합니다.

int **PyDateTime_DATE_GET_MICROSECOND**(*PyDateTime_DateTime* *o)
0부터 999999까지의 int로, 마이크로초를 반환합니다.

int **PyDateTime_DATE_GET_FOLD**(*PyDateTime_DateTime* *o)
Return the fold, as an int from 0 through 1.

Added in version 3.6.

PyObject ***PyDateTime_DATE_GET_TZINFO**(*PyDateTime_DateTime* *o)
Return the tzinfo (which may be `None`).

Added in version 3.10.

Macros to extract fields from time objects. The argument must be an instance of *PyDateTime_Time*, including subclasses. The argument must not be `NULL`, and the type is not checked:

int **PyDateTime_TIME_GET_HOUR**(*PyDateTime_Time* *o)
0부터 23까지의 int로, 시를 반환합니다.

int **PyDateTime_TIME_GET_MINUTE**(*PyDateTime_Time* *o)
0부터 59까지의 int로, 분을 반환합니다.

int **PyDateTime_TIME_GET_SECOND**(*PyDateTime_Time* *o)
0부터 59까지의 int로, 초를 반환합니다.

int **PyDateTime_TIME_GET_MICROSECOND**(*PyDateTime_Time* *o)
0부터 999999까지의 int로, 마이크로초를 반환합니다.

int **PyDateTime_TIME_GET_FOLD**(*PyDateTime_Time* *o)
Return the fold, as an int from 0 through 1.

Added in version 3.6.

PyObject ***PyDateTime_TIME_GET_TZINFO**(*PyDateTime_Time* *o)
Return the tzinfo (which may be `None`).

Added in version 3.10.

Macros to extract fields from time delta objects. The argument must be an instance of *PyDateTime_Delta*, including subclasses. The argument must not be `NULL`, and the type is not checked:

int **PyDateTime_DELTA_GET_DAYS**(*PyDateTime_Delta* *o)
-999999999에서 999999999까지의 int로, 일 수를 반환합니다.

Added in version 3.3.

int **PyDateTime_DELTA_GET_SECONDS**(*PyDateTime_Delta* *o)

0부터 86399까지의 int로, 초 수를 반환합니다.

Added in version 3.3.

int **PyDateTime_DELTA_GET_MICROSECONDS**(*PyDateTime_Delta* *o)

0에서 999999까지의 int로, 마이크로초 수를 반환합니다.

Added in version 3.3.

DB API를 구현하는 모듈의 편의를 위한 매크로:

PyObject ***PyDateTime_FromTimestamp**(*PyObject* *args)

Return value: New reference. Create and return a new `datetime.datetime` object given an argument tuple suitable for passing to `datetime.datetime.fromtimestamp()`.

PyObject ***PyDate_FromTimestamp**(*PyObject* *args)

Return value: New reference. Create and return a new `datetime.date` object given an argument tuple suitable for passing to `datetime.date.fromtimestamp()`.

8.6.14 Objects for Type Hinting

Various built-in types for type hinting are provided. Currently, two types exist – GenericAlias and Union. Only `GenericAlias` is exposed to C.

PyObject ***Py_GenericAlias**(*PyObject* *origin, *PyObject* *args)

Part of the Stable ABI *since version 3.9.* Create a GenericAlias object. Equivalent to calling the Python class `types.GenericAlias`. The *origin* and *args* arguments set the `GenericAlias` 's `__origin__` and `__args__` attributes respectively. *origin* should be a *`PyTypeObject`*`*`, and *args* can be a *`PyTupleObject`*`*` or any `PyObject*`. If *args* passed is not a tuple, a 1-tuple is automatically constructed and `__args__` is set to `(args,)`. Minimal checking is done for the arguments, so the function will succeed even if *origin* is not a type. The `GenericAlias` 's `__parameters__` attribute is constructed lazily from `__args__`. On failure, an exception is raised and `NULL` is returned.

Here' s an example of how to make an extension type generic:

```
...
static PyMethodDef my_obj_methods[] = {
    // Other methods.
    ...
    {"__class_getitem__", Py_GenericAlias, METH_O|METH_CLASS, "See PEP 585"}
    ...
}
```

더 보기

The data model method `__class_getitem__()`.

Added in version 3.9.

PyTypeObject **Py_GenericAliasType**

Part of the Stable ABI *since version 3.9.* The C type of the object returned by *`Py_GenericAlias()`*. Equivalent to `types.GenericAlias` in Python.

Added in version 3.9.

CHAPTER 9

초기화, 파이널리제이션 및 스레드

파이썬 초기화 구성도 참조하십시오.

9.1 파이썬 초기화 전

파이썬을 내장한 응용 프로그램에서는, 다른 파이썬/C API 함수를 사용하기 전에 *Py_Initialize()* 함수를 호출해야 합니다; 몇 가지 함수와 전역 구성 변수는 예외입니다.

파이썬이 초기화되기 전에 다음 함수를 안전하게 호출할 수 있습니다:

- 구성 함수:
  - *PyImport_AppendInittab()*
  - *PyImport_ExtendInittab()*
  - PyInitFrozenExtensions()
  - *PyMem_SetAllocator()*
  - *PyMem_SetupDebugHooks()*
  - *PyObject_SetArenaAllocator()*
  - *Py_SetPath()*
  - *Py_SetProgramName()*
  - *Py_SetPythonHome()*
  - *Py_SetStandardStreamEncoding()*
  - *PySys_AddWarnOption()*
  - *PySys_AddXOption()*
  - *PySys_ResetWarnOptions()*
- 정보 함수:
  - *Py_IsInitialized()*
  - *PyMem_GetAllocator()*
  - *PyObject_GetArenaAllocator()*

  - *Py_GetBuildInfo()*
  - *Py_GetCompiler()*
  - *Py_GetCopyright()*
  - *Py_GetPlatform()*
  - *Py_GetVersion()*
- 유틸리티:
  - *Py_DecodeLocale()*
- 메모리 할당자:
  - *PyMem_RawMalloc()*
  - *PyMem_RawRealloc()*
  - *PyMem_RawCalloc()*
  - *PyMem_RawFree()*

참고

다음 함수는 *Py_Initialize()* 전에 **호출하면 안 됩니다**: *Py_EncodeLocale()*, *Py_GetPath()*, *Py_GetPrefix()*, *Py_GetExecPrefix()*, *Py_GetProgramFullPath()*, *Py_GetPythonHome()*, *Py_GetProgramName()* 및 *PyEval_InitThreads()*.

9.2 전역 구성 변수

파이썬에는 다양한 기능과 옵션을 제어하기 위한 전역 구성 변수가 있습니다. 기본적으로, 이러한 플래그는 명령 줄 옵션에 의해 제어됩니다.

옵션에 의해 플래그가 설정되면, 플래그 값은 옵션이 설정된 횟수입니다. 예를 들어, `-b`는 *Py_BytesWarningFlag*를 1로 설정하고 `-bb`는 *Py_BytesWarningFlag*를 2로 설정합니다.

int **Py_BytesWarningFlag**

This API is kept for backward compatibility: setting *PyConfig.bytes_warning* should be used instead, see *Python Initialization Configuration*.

`bytes`나 `bytearray`와 `str`을, 또는 `bytes`를 `int`와 비교할 때 경고를 발행합니다. `2`보다 크거나 같으면 에러를 발행합니다.

`-b` 옵션으로 설정합니다.

버전 3.12부터 폐지됨.

int **Py_DebugFlag**

This API is kept for backward compatibility: setting *PyConfig.parser_debug* should be used instead, see *Python Initialization Configuration*.

구문 분석기 디버깅 출력을 켭니다 (전문가 전용, 컴파일 옵션에 의존합니다).

`-d` 옵션과 `PYTHONDEBUG` 환경 변수로 설정됩니다.

버전 3.12부터 폐지됨.

int **Py_DontWriteBytecodeFlag**

This API is kept for backward compatibility: setting *PyConfig.write_bytecode* should be used instead, see *Python Initialization Configuration*.

0이 아닌 값으로 설정하면, 파이썬은 소스 모듈을 임포트 할 때 `.pyc` 파일을 쓰려고 하지 않습니다.

`-B` 옵션과 `PYTHONDONTWRITEBYTECODE` 환경 변수로 설정됩니다.

버전 3.12부터 폐지됨.

int **Py_FrozenFlag**

This API is kept for backward compatibility: setting *PyConfig.pathconfig_warnings* should be used instead, see *Python Initialization Configuration*.

*Py_GetPath()*에서 모듈 검색 경로를 계산할 때 에러 메시지를 표시하지 않습니다.

Private flag used by `_freeze_module` and `frozenmain` programs.

버전 3.12부터 폐지됨.

int **Py_HashRandomizationFlag**

This API is kept for backward compatibility: setting *PyConfig.hash_seed* and *PyConfig.use_hash_seed* should be used instead, see *Python Initialization Configuration*.

`PYTHONHASHSEED` 환경 변수가 비어 있지 않은 문자열로 설정되면 `1`로 설정합니다.

플래그가 0이 아니면, `PYTHONHASHSEED` 환경 변수를 읽어 비밀 해시 시드를 초기화합니다.

버전 3.12부터 폐지됨.

int **Py_IgnoreEnvironmentFlag**

This API is kept for backward compatibility: setting *PyConfig.use_environment* should be used instead, see *Python Initialization Configuration*.

Ignore all `PYTHON*` environment variables, e.g. `PYTHONPATH` and `PYTHONHOME`, that might be set.

`-E`와 `-I` 옵션으로 설정됩니다.

버전 3.12부터 폐지됨.

int **Py_InspectFlag**

This API is kept for backward compatibility: setting *PyConfig.inspect* should be used instead, see *Python Initialization Configuration*.

스크립트가 첫 번째 인자로 전달되거나 `-c` 옵션을 사용할 때, `sys.stdin`가 터미널로 보이지 않더라도 스크립트나 명령을 실행한 후 대화 형 모드로 들어갑니다.

`-i` 옵션과 `PYTHONINSPECT` 환경 변수로 설정됩니다.

버전 3.12부터 폐지됨.

int **Py_InteractiveFlag**

This API is kept for backward compatibility: setting *PyConfig.interactive* should be used instead, see *Python Initialization Configuration*.

`-i` 옵션으로 설정됩니다.

버전 3.12부터 폐지됨.

int **Py_IsolatedFlag**

This API is kept for backward compatibility: setting *PyConfig.isolated* should be used instead, see *Python Initialization Configuration*.

격리 모드로 파이썬을 실행합니다. 격리 모드에서 `sys.path`는 스크립트의 디렉터리도 사용자의 site-packages 디렉터리도 포함하지 않습니다.

`-I` 옵션으로 설정됩니다.

Added in version 3.4.

버전 3.12부터 폐지됨.

int **Py_LegacyWindowsFSEncodingFlag**

This API is kept for backward compatibility: setting *PyPreConfig.legacy_windows_fs_encoding* should be used instead, see *Python Initialization Configuration*.

If the flag is non-zero, use the `mbcs` encoding with `replace` error handler, instead of the UTF-8 encoding with `surrogatepass` error handler, for the *filesystem encoding and error handler*.

PYTHONLEGACYWINDOWSFSENCODING 환경 변수가 비어 있지 않은 문자열로 설정되면 1로 설정합니다.

자세한 내용은 **PEP 529**를 참조하십시오.

Availability: Windows.

버전 3.12부터 폐지됨.

int **Py_LegacyWindowsStdioFlag**

This API is kept for backward compatibility: setting *PyConfig.legacy_windows_stdio* should be used instead, see *Python Initialization Configuration*.

If the flag is non-zero, use io.FileIO instead of io._WindowsConsoleIO for sys standard streams.

PYTHONLEGACYWINDOWSSTDIO 환경 변수가 비어 있지 않은 문자열로 설정되면 1로 설정합니다.

자세한 내용은 **PEP 528**을 참조하십시오.

Availability: Windows.

버전 3.12부터 폐지됨.

int **Py_NoSiteFlag**

This API is kept for backward compatibility: setting *PyConfig.site_import* should be used instead, see *Python Initialization Configuration*.

모듈 site 임포트와 이에 수반되는 sys.path의 사이트 종속적인 조작을 비활성화합니다. 또한 나중에 site를 명시적으로 임포트 할 때도 이러한 조작을 비활성화합니다 (트리거 하려면 site.main() 을 호출하십시오).

-S 옵션으로 설정됩니다.

버전 3.12부터 폐지됨.

int **Py_NoUserSiteDirectory**

This API is kept for backward compatibility: setting *PyConfig.user_site_directory* should be used instead, see *Python Initialization Configuration*.

사용자 site-packages 디렉터리를 sys.path에 추가하지 않습니다.

-s와 -I 옵션, 그리고 PYTHONNOUSERSITE 환경 변수로 설정됩니다.

버전 3.12부터 폐지됨.

int **Py_OptimizeFlag**

This API is kept for backward compatibility: setting *PyConfig.optimization_level* should be used instead, see *Python Initialization Configuration*.

-O 옵션과 PYTHONOPTIMIZE 환경 변수로 설정됩니다.

버전 3.12부터 폐지됨.

int **Py_QuietFlag**

This API is kept for backward compatibility: setting *PyConfig.quiet* should be used instead, see *Python Initialization Configuration*.

대화형 모드에서도 저작권과 버전 메시지를 표시하지 않습니다.

-q 옵션으로 설정됩니다.

Added in version 3.2.

버전 3.12부터 폐지됨.

int **Py_UnbufferedStdioFlag**

This API is kept for backward compatibility: setting *PyConfig.buffered_stdio* should be used instead, see *Python Initialization Configuration*.

stdout과 stderr 스트림을 버퍼링 해제하도록 강제합니다.

-u 옵션과 PYTHONUNBUFFERED 환경 변수로 설정됩니다.

버전 3.12부터 폐지됨.

int **Py_VerboseFlag**

This API is kept for backward compatibility: setting *PyConfig.verbose* should be used instead, see *Python Initialization Configuration*.

모듈이 초기화될 때마다, 로드된 위치(파일명이나 내장 모듈)를 표시하는 메시지를 인쇄합니다. 2 보다 크거나 같으면, 모듈을 검색할 때 검사되는 각 파일에 대한 메시지를 인쇄합니다. 또한 종료 시 모듈 정리에 대한 정보를 제공합니다.

-v 옵션과 PYTHONVERBOSE 환경 변수로 설정됩니다.

버전 3.12부터 폐지됨.

9.3 인터프리터 초기화와 파이널리제이션

void **Py_Initialize**()

Part of the Stable ABI. 파이썬 인터프리터를 초기화합니다. 파이썬을 내장하는 응용 프로그램에서는, 다른 파이썬/C API 함수를 사용하기 전에 호출해야 합니다; 몇 가지 예외는 파이썬 초기화 전을 참조하십시오.

이것은 로드된 모듈의 테이블(sys.modules)을 초기화하고, 기반 모듈 builtins, __main__ 및 sys를 만듭니다. 또한, 모듈 검색 경로(sys.path)를 초기화합니다. sys.argv는 설정하지 않습니다; 이를 위해서는 *PySys_SetArgvEx()* 를 사용하십시오. (*Py_FinalizeEx()* 를 먼저 호출하지 않고) 두 번째로 호출하면 아무런 일도 하지 않습니다. 반환 값이 없습니다; 초기화에 실패하면 치명적인 에러입니다.

Use the *Py_InitializeFromConfig()* function to customize the *Python Initialization Configuration*.

> **참고**
>
> 윈도우에서, 콘솔 모드를 O_TEXT에서 O_BINARY로 변경합니다, C 런타임을 사용하는 콘솔의 비 파이썬 사용에도 영향을 미칩니다.

void **Py_InitializeEx**(int initsigs)

Part of the Stable ABI. 이 함수는 *initsigs*가 1이면 *Py_Initialize()*처럼 작동합니다. *initsigs*가 0이면, 시그널 처리기의 초기화 등록을 건너뛰는데, 파이썬이 내장될 때 유용할 수 있습니다.

Use the *Py_InitializeFromConfig()* function to customize the *Python Initialization Configuration*.

int **Py_IsInitialized**()

Part of the Stable ABI. 파이썬 인터프리터가 초기화되었으면 참(0이 아님)을 반환하고, 그렇지 않으면 거짓(0)을 반환합니다. *Py_FinalizeEx()* 가 호출된 후, *Py_Initialize()*가 다시 호출될 때까지 거짓을 반환합니다.

int **Py_FinalizeEx**()

Part of the Stable ABI *since version 3.6.* Undo all initializations made by *Py_Initialize()* and subsequent use of Python/C API functions, and destroy all sub-interpreters (see *Py_NewInterpreter()* below) that were created and not yet destroyed since the last call to *Py_Initialize()*. Ideally, this frees all memory allocated by the Python interpreter. This is a no-op when called for a second time (without calling *Py_Initialize()* again first).

Since this is the reverse of *Py_Initialize()*, it should be called in the same thread with the same interpreter active. That means the main thread and the main interpreter. This should never be called while *Py_RunMain()* is running.

Normally the return value is 0. If there were errors during finalization (flushing buffered data), -1 is returned.

이 함수는 여러 가지 이유로 제공됩니다. 내장 응용 프로그램이 응용 프로그램 자체를 다시 시작하지 않고 파이썬을 다시 시작하고 싶을 수 있습니다. 동적으로 로드할 수 있는 라이브러리(또는 DLL)에서 파이썬 인터프리터를 로드한 응용 프로그램은 DLL을 언로드 하기 전에 파이썬이 할당한 모든

메모리를 해제하고 싶을 수 있습니다. 응용 프로그램에서 메모리 누수를 찾는 동안 개발자는 응용 프로그램을 종료하기 전에 파이썬에서 할당한 모든 메모리를 해제하고 싶을 것입니다.

Bugs and caveats: The destruction of modules and objects in modules is done in random order; this may cause destructors (`__del__()` methods) to fail when they depend on other objects (even functions) or modules. Dynamically loaded extension modules loaded by Python are not unloaded. Small amounts of memory allocated by the Python interpreter may not be freed (if you find a leak, please report it). Memory tied up in circular references between objects is not freed. Some memory allocated by extension modules may not be freed. Some extensions may not work properly if their initialization routine is called more than once; this can happen if an application calls *Py_Initialize()* and *Py_FinalizeEx()* more than once.

인자 없이 감사 이벤트 `cpython._PySys_ClearAuditHooks` 를 발생시킵니다.

Added in version 3.6.

void **Py_Finalize**()

Part of the Stable ABI. 이것은 *Py_FinalizeEx()*의 이전 버전과 호환되는 반환 값을 무시하는 버전입니다.

9.4 프로세스 전체 매개 변수

int **Py_SetStandardStreamEncoding**(const char *encoding, const char *errors)

This API is kept for backward compatibility: setting *PyConfig.stdio_encoding* and *PyConfig.stdio_errors* should be used instead, see *Python Initialization Configuration*.

이 함수는 (호출한다면) *Py_Initialize()* 전에 호출해야 합니다. `str.encode()` 에서와 같은 의미로, 표준 IO에 사용할 인코딩과 에러 처리를 지정합니다.

`PYTHONIOENCODING` 값을 재정의(overrides)하고, 환경 변수가 작동하지 않을 때 내장(embedding) 코드가 IO 인코딩을 제어할 수 있도록 합니다.

encoding 및/또는 *errors*는 `PYTHONIOENCODING` 및/또는 기본값(다른 설정에 따라 다릅니다)을 사용하기 위해 `NULL`일 수 있습니다.

`sys.stderr`은 이 (또는 다른) 설정과 관계없이 항상 "backslashreplace" 에러 처리기를 사용함에 유의하십시오.

*Py_FinalizeEx()*가 호출되면, 이 함수는 *Py_Initialize()*에 대한 후속 호출에 영향을 미치기 위해 다시 호출되어야 합니다.

성공하면 `0`을 반환하고, 에러 시 (예를 들어 인터프리터가 이미 초기화된 후 호출) 0이 아닌 값을 반환합니다.

Added in version 3.4.

버전 3.11부터 폐지됨.

void **Py_SetProgramName**(const wchar_t *name)

Part of the Stable ABI. This API is kept for backward compatibility: setting *PyConfig.program_name* should be used instead, see *Python Initialization Configuration*.

(호출된다면) 이 함수는 *Py_Initialize()*가 처음으로 호출되기 전에 호출되어야 합니다. 인터프리터에게 프로그램의 `main()` 함수에 대한 `argv[0]` 인자의 값을 알려줍니다 (와이드 문자로 변환됩니다). 이것은 *Py_GetPath()*와 아래의 다른 함수에서 인터프리터 실행 파일과 관련된 파이썬 런타임 라이브러리를 찾는 데 사용됩니다. 기본값은 `'python'`입니다. 인자는 프로그램을 실행하는 동안 내용이 변경되지 않는 정적 저장소의 0으로 끝나는 와이드 문자열을 가리켜야 합니다. 파이썬 인터프리터의 코드는 이 저장소의 내용을 변경하지 않습니다.

Use *Py_DecodeLocale()* to decode a bytes string to get a `wchar_t*` string.

버전 3.11부터 폐지됨.

wchar_t ***Py_GetProgramName**()

Part of the Stable ABI. *Py_SetProgramName()* 으로 설정된 프로그램 이름이나 기본값을 반환합니다. 반환된 문자열은 정적 저장소를 가리킵니다; 호출자는 값을 수정해서는 안 됩니다.

This function should not be called before *Py_Initialize()*, otherwise it returns `NULL`.

버전 3.10에서 변경: It now returns `NULL` if called before *Py_Initialize()*.

wchar_t ***Py_GetPrefix**()

Part of the Stable ABI. Return the *prefix* for installed platform-independent files. This is derived through a number of complicated rules from the program name set with *Py_SetProgramName()* and some environment variables; for example, if the program name is `'/usr/local/bin/python'`, the prefix is `'/usr/local'`. The returned string points into static storage; the caller should not modify its value. This corresponds to the **prefix** variable in the top-level `Makefile` and the `--prefix` argument to the **configure** script at build time. The value is available to Python code as `sys.prefix`. It is only useful on Unix. See also the next function.

This function should not be called before *Py_Initialize()*, otherwise it returns `NULL`.

버전 3.10에서 변경: It now returns `NULL` if called before *Py_Initialize()*.

wchar_t ***Py_GetExecPrefix**()

Part of the Stable ABI. 설치된 플랫폼-종속적 파일에 대한 *exec-prefix*를 반환합니다. 이것은 *Py_SetProgramName()* 으로 설정된 프로그램 이름과 일부 환경 변수의 여러 복잡한 규칙을 통해 파생됩니다; 예를 들어 프로그램 이름이 `'/usr/local/bin/python'`이면, exec-prefix는 `'/usr/local'`입니다. 반환된 문자열은 정적 저장소를 가리킵니다; 호출자는 값을 수정해서는 안 됩니다. 이는 최상위 수준 `Makefile`의 **exec_prefix** 변수와 빌드 시 **configure** 스크립트의 `--exec-prefix` 인자에 해당합니다. 이 값은 파이썬 코드에서 `sys.exec_prefix`로 사용할 수 있습니다. 유닉스에서만 유용합니다.

배경: exec-prefix는 플랫폼 종속적 파일(가령 실행 파일과 공유 라이브러리)이 다른 디렉터리 트리에 설치될 때 prefix와 다릅니다. 일반 설치에서, 플랫폼 종속적 파일은 `/usr/local/plat` 서브 트리에 설치되고 플랫폼 독립적 파일은 `/usr/local`에 설치될 수 있습니다.

일반적으로 말해서, 플랫폼은 하드웨어와 소프트웨어 제품군의 조합입니다, 예를 들어 Solaris 2.x 운영 체제를 실행하는 Sparc 기계들은 같은 플랫폼으로 간주하지만, Solaris 2.x를 실행하는 Intel 기계는 다른 플랫폼이며, 리눅스를 실행하는 Intel 기계는 또 다른 플랫폼입니다. 같은 운영 체제의 서로 다른 주 개정판도 일반적으로 다른 플랫폼을 형성합니다. 비 유닉스 운영 체제는 다른 이야기입니다; 이러한 시스템의 설치 전략이 너무 다르기 때문에 prefix와 exec-prefix는 의미가 없으며, 빈 문자열로 설정됩니다. 컴파일된 파이썬 바이트 코드 파일은 플랫폼 독립적임에 유의하십시오 (그러나 이들을 컴파일하는데 사용된 파이썬 버전에는 종속적입니다!).

시스템 관리자는 `/usr/local/plat`을 각 플랫폼에 대해 다른 파일 시스템으로 사용하면서 플랫폼 간에 `/usr/local`을 공유하도록 **mount** 나 **automount** 프로그램을 구성하는 방법을 알 것입니다.

This function should not be called before *Py_Initialize()*, otherwise it returns `NULL`.

버전 3.10에서 변경: It now returns `NULL` if called before *Py_Initialize()*.

wchar_t ***Py_GetProgramFullPath**()

Part of the Stable ABI. 파이썬 실행 파일의 전체 프로그램 이름을 반환합니다; 이것은 프로그램 이름(위의 *Py_SetProgramName()* 으로 설정됩니다)에서 기본 모듈 검색 경로를 파생하는 부작용으로 계산됩니다. 반환된 문자열은 정적 저장소를 가리킵니다; 호출자는 값을 수정해서는 안 됩니다. 이 값은 파이썬 코드에서 `sys.executable`로 사용할 수 있습니다.

This function should not be called before *Py_Initialize()*, otherwise it returns `NULL`.

버전 3.10에서 변경: It now returns `NULL` if called before *Py_Initialize()*.

wchar_t ***Py_GetPath**()

Part of the Stable ABI. Return the default module search path; this is computed from the program name (set by *Py_SetProgramName()* above) and some environment variables. The returned string consists of a series of directory names separated by a platform dependent delimiter character. The delimiter character is `':'` on Unix and macOS, `';'` on Windows. The returned string points into static storage; the caller should not

modify its value. The list `sys.path` is initialized with this value on interpreter startup; it can be (and usually is) modified later to change the search path for loading modules.

This function should not be called before *`Py_Initialize()`*, otherwise it returns `NULL`.

버전 3.10에서 변경: It now returns `NULL` if called before *`Py_Initialize()`*.

void **`Py_SetPath`**(const wchar_t*)

Part of the Stable ABI *since version 3.7.* This API is kept for backward compatibility: setting *`PyConfig.module_search_paths`* and *`PyConfig.module_search_paths_set`* should be used instead, see *Python Initialization Configuration*.

Set the default module search path. If this function is called before *`Py_Initialize()`*, then *`Py_GetPath()`* won' t attempt to compute a default search path but uses the one provided instead. This is useful if Python is embedded by an application that has full knowledge of the location of all modules. The path components should be separated by the platform dependent delimiter character, which is `':'` on Unix and macOS, `';'` on Windows.

또한 `sys.executable`이 프로그램 전체 경로 (*`Py_GetProgramFullPath()`* 를 참조하십시오)로 설정되고 `sys.prefix`와 `sys.exec_prefix`가 비어있도록 합니다. *`Py_Initialize()`*를 호출한 후 필요할 때 이를 수정하는 것은 호출자에게 달려 있습니다.

Use *`Py_DecodeLocale()`* to decode a bytes string to get a `wchar_*` string.

경로 인자는 내부적으로 복사되므로, 호출이 완료된 후 호출자가 할당 해제할 수 있습니다.

버전 3.8에서 변경: 이제 프로그램 이름 대신 프로그램 전체 경로가 `sys.executable`에 사용됩니다.

버전 3.11부터 폐지됨.

const char ***`Py_GetVersion`**()

Part of the Stable ABI. 이 파이썬 인터프리터의 버전을 반환합니다. 이것은 다음과 같은 문자열입니다

```
"3.0a5+ (py3k:63103M, May 12 2008, 00:53:55) \n[GCC 4.2.3]"
```

The first word (up to the first space character) is the current Python version; the first characters are the major and minor version separated by a period. The returned string points into static storage; the caller should not modify its value. The value is available to Python code as `sys.version`.

See also the *`Py_Version`* constant.

const char ***`Py_GetPlatform`**()

Part of the Stable ABI. Return the platform identifier for the current platform. On Unix, this is formed from the "official" name of the operating system, converted to lower case, followed by the major revision number; e.g., for Solaris 2.x, which is also known as SunOS 5.x, the value is `'sunos5'`. On macOS, it is `'darwin'`. On Windows, it is `'win'`. The returned string points into static storage; the caller should not modify its value. The value is available to Python code as `sys.platform`.

const char ***`Py_GetCopyright`**()

Part of the Stable ABI. 현재 파이썬 버전에 대한 공식 저작권 문자열을 반환합니다, 예를 들어

```
'Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam'
```

반환된 문자열은 정적 저장소를 가리킵니다; 호출자는 값을 수정해서는 안 됩니다. 이 값은 파이썬 코드에서 `sys.copyright`로 사용할 수 있습니다.

const char ***`Py_GetCompiler`**()

Part of the Stable ABI. 현재 파이썬 버전을 빌드하는 데 사용된 컴파일러 표시를 대괄호 감싸서 반환합니다, 예를 들면:

```
"[GCC 2.7.2.2]"
```

반환된 문자열은 정적 저장소를 가리킵니다; 호출자는 값을 수정해서는 안 됩니다. 이 값은 파이썬 코드에서 변수 `sys.version`의 일부로 제공됩니다.

const char *Py_GetBuildInfo()

Part of the Stable ABI. 현재 파이썬 인터프리터 인스턴스의 시퀀스 번호와 빌드 날짜 및 시간에 대한 정보를 반환합니다, 예를 들어

```
"#67, Aug  1 1997, 22:34:28"
```

반환된 문자열은 정적 저장소를 가리킵니다; 호출자는 값을 수정해서는 안 됩니다. 이 값은 파이썬 코드에서 변수 `sys.version`의 일부로 제공됩니다.

void **PySys_SetArgvEx**(int argc, wchar_t **argv, int updatepath)

Part of the Stable ABI. This API is kept for backward compatibility: setting *PyConfig.argv*, *PyConfig.parse_argv* and *PyConfig.safe_path* should be used instead, see *Python Initialization Configuration*.

argc 및 *argv*에 기반해서 `sys.argv`를 설정합니다. 이 매개 변수는 프로그램의 `main()` 함수에 전달된 것과 유사하지만, 첫 번째 항목이 파이썬 인터프리터를 호스팅하는 실행 파일이 아니라 실행될 스크립트 파일을 참조해야 한다는 차이점이 있습니다. 실행할 스크립트가 없으면, *argv*의 첫 번째 항목은 빈 문자열일 수 있습니다. 이 함수가 `sys.argv` 초기화에 실패하면, *Py_FatalError()*를 사용하여 치명적인 조건을 표시합니다.

*updatepath*가 0이면, 여기까지가 이 함수가 하는 모든 일입니다. *updatepath*가 0이 아니면, 함수는 다음 알고리즘에 따라 `sys.path`도 수정합니다:

- 기존 스크립트의 이름이 `argv[0]`으로 전달되면, 스크립트가 있는 디렉터리의 절대 경로가 `sys.path` 앞에 추가됩니다.
- 그렇지 않으면 (즉, *argc*가 `0`이거나 `argv[0]`이 기존 파일 이름을 가리키지 않으면), `sys.path` 앞에 빈 문자열이 추가됩니다, 이는 현재 작업 디렉터리(`"."`)를 앞에 추가하는 것과 같습니다.

Use *Py_DecodeLocale()* to decode a bytes string to get a `wchar_t*` string.

See also *PyConfig.orig_argv* and *PyConfig.argv* members of the *Python Initialization Configuration*.

참고

It is recommended that applications embedding the Python interpreter for purposes other than executing a single script pass `0` as *updatepath*, and update `sys.path` themselves if desired. See **CVE 2008-5983**.

3.1.3 이전 버전에서는, *PySys_SetArgv()*를 호출한 후 첫 번째 `sys.path` 요소를 수동으로 제거하여 같은 효과를 얻을 수 있습니다, 예를 들어 다음을 사용하여:

```
PyRun_SimpleString("import sys; sys.path.pop(0)\n");
```

Added in version 3.1.3.

버전 3.11부터 폐지됨.

void **PySys_SetArgv**(int argc, wchar_t **argv)

Part of the Stable ABI. This API is kept for backward compatibility: setting *PyConfig.argv* and *PyConfig.parse_argv* should be used instead, see *Python Initialization Configuration*.

이 함수는 **python** 인터프리터가 `-I`로 시작되지 않는 한 *updatepath*가 `1`로 설정된 *PySys_SetArgvEx()*처럼 작동합니다.

Use *Py_DecodeLocale()* to decode a bytes string to get a `wchar_t*` string.

See also *PyConfig.orig_argv* and *PyConfig.argv* members of the *Python Initialization Configuration*.

버전 3.4에서 변경: *updatepath* 값은 `-I`에 따라 다릅니다.

버전 3.11부터 폐지됨.

void **Py_SetPythonHome**(const wchar_t *home)

Part of the Stable ABI. This API is kept for backward compatibility: setting *PyConfig.home* should be used instead, see *Python Initialization Configuration*.

기본 "홈" 디렉터리, 즉 표준 파이썬 라이브러리의 위치를 설정합니다. 인자 문자열의 의미는 PYTHONHOME 을 참조하십시오.

인자는 프로그램을 실행하는 동안 내용이 변경되지 않는 정적 저장소에 있는 0으로 끝나는 문자열을 가리켜야 합니다. 파이썬 인터프리터의 코드는 이 저장소의 내용을 변경하지 않습니다.

Use *Py_DecodeLocale()* to decode a bytes string to get a `wchar_t*` string.

버전 3.11부터 폐지됨.

wchar_t ***Py_GetPythonHome**()

Part of the Stable ABI. 기본 "홈", 즉 *Py_SetPythonHome()* 에 대한 이전 호출에서 설정한 값이나 설정되었다면 PYTHONHOME 환경 변수의 값을 반환합니다.

This function should not be called before *Py_Initialize()*, otherwise it returns NULL.

버전 3.10에서 변경: It now returns NULL if called before *Py_Initialize()*.

9.5 스레드 상태와 전역 인터프리터 록

파이썬 인터프리터는 완전히 스레드 안전하지 않습니다. 다중 스레드 파이썬 프로그램을 지원하기 위해, 파이썬 객체에 안전하게 액세스하기 전에 현재 스레드가 보유해야 하는 전역 인터프리터 록 혹은 *GIL*이라고 하는 전역 록이 있습니다. 록 없이는, 가장 간단한 연산조차도 다중 스레드 프로그램에서 문제를 일으킬 수 있습니다: 예를 들어, 두 스레드가 동시에 같은 객체의 참조 횟수를 증가시키면, 참조 횟수가 두 번이 아닌 한 번만 증가할 수 있습니다.

따라서, *GIL*을 획득한 스레드만 파이썬 객체에서 작동하거나 파이썬/C API 함수를 호출할 수 있다는 규칙이 있습니다. 동시 실행을 모방하기 위해 인터프리터는 정기적으로 스레드 전환을 시도합니다 (`sys.setswitchinterval()`을 참조하십시오). 록은 파일 읽기나 쓰기와 같은 잠재적인 블로킹 I/O 연산에 대해서도 해제되므로, 그동안 다른 파이썬 스레드가 실행될 수 있습니다.

파이썬 인터프리터는 *PyThreadState* 라는 데이터 구조체 내에 스레드 별 부기(bookkeeping) 정보를 보관합니다. 현재 *PyThreadState* 를 가리키는 하나의 전역 변수도 있습니다: *PyThreadState_Get()*을 사용하여 얻을 수 있습니다.

9.5.1 확장 코드에서 GIL 해제하기

*GIL*을 조작하는 대부분의 확장 코드는 다음과 같은 간단한 구조로 되어 있습니다:

```
Save the thread state in a local variable.
Release the global interpreter lock.
... Do some blocking I/O operation ...
Reacquire the global interpreter lock.
Restore the thread state from the local variable.
```

이것은 매우 일반적이어서 이를 단순화하기 위해 한 쌍의 매크로가 존재합니다:

```
Py_BEGIN_ALLOW_THREADS
... Do some blocking I/O operation ...
Py_END_ALLOW_THREADS
```

Py_BEGIN_ALLOW_THREADS 매크로는 새 블록을 열고 숨겨진 지역 변수를 선언합니다; *Py_END_ALLOW_THREADS* 매크로는 블록을 닫습니다.

위의 블록은 다음 코드로 확장됩니다:

```
PyThreadState *_save;

_save = PyEval_SaveThread();
... Do some blocking I/O operation ...
PyEval_RestoreThread(_save);
```

이 함수들의 작동 방식은 다음과 같습니다: 전역 인터프리터 록이 현재 스레드 상태에 대한 포인터를 보호하는 데 사용됩니다. 록을 해제하고 스레드 상태를 저장할 때, 록이 해제되기 전에 현재 스레드 상태 포인터를 가져와야 합니다 (다른 스레드가 즉시 록을 획득하고 전역 변수에 자신의 스레드 상태를 저장할 수 있기 때문입니다). 반대로, 록을 획득하고 스레드 상태를 복원할 때, 스레드 상태 포인터를 저장하기 전에 록을 획득해야 합니다.

참고

시스템 I/O 함수 호출은 GIL을 릴리스하는 가장 일반적인 사용 사례이지만, 메모리 버퍼를 통해 작동하는 압축이나 암호화 함수와 같이, 파이썬 객체에 액세스할 필요가 없는 장기 실행 계산을 호출하기 전에도 유용할 수 있습니다. 예를 들어, 표준 `zlib`와 `hashlib` 모듈은 데이터를 압축하거나 해싱할 때 GIL을 해제합니다.

9.5.2 파이썬이 만들지 않은 스레드

전용 파이썬 API(가령 `threading` 모듈)를 사용하여 스레드를 만들면, 스레드 상태가 자동으로 연결되므로 위에 표시된 코드가 올바릅니다. 그러나, 스레드가 C에서 만들어질 때 (예를 들어 자체 스레드 관리 기능이 있는 제삼자 라이브러리에 의해), GIL을 보유하지 않고, 그들을 위한 스레드 상태 구조도 없습니다.

이러한 스레드에서 파이썬 코드를 호출해야 하면 (종종 앞서 언급한 제삼자 라이브러리에서 제공하는 콜백 API의 일부가 됩니다), 먼저 스레드 상태 자료 구조를 만들어서 인터프리터에 이러한 스레드를 등록한 다음, GIL을 획득하고, 마지막으로 파이썬/C API 사용을 시작하기 전에 스레드 상태 포인터를 저장합니다. 완료되면, 스레드 상태 포인터를 재설정하고, GIL을 해제한 다음, 마지막으로 스레드 상태 자료 구조를 해제해야 합니다.

*`PyGILState_Ensure()`*와 *`PyGILState_Release()`* 함수는 위의 모든 작업을 자동으로 수행합니다. C 스레드에서 파이썬을 호출하는 일반적인 관용구는 다음과 같습니다:

```
PyGILState_STATE gstate;
gstate = PyGILState_Ensure();

/* Perform Python actions here. */
result = CallSomeFunction();
/* evaluate result or handle exception */

/* Release the thread. No Python API allowed beyond this point. */
PyGILState_Release(gstate);
```

Note that the `PyGILState_*` functions assume there is only one global interpreter (created automatically by *`Py_Initialize()`*). Python supports the creation of additional interpreters (using *`Py_NewInterpreter()`*), but mixing multiple interpreters and the `PyGILState_*` API is unsupported.

9.5.3 fork()에 대한 주의 사항

스레드에 대해 주목해야 할 또 다른 중요한 점은 C `fork()` 호출 시 스레드의 동작입니다. `fork()`를 사용하는 대부분의 시스템에서는, 프로세스가 포크한 후에 포크를 발행한 스레드만 존재합니다. 이는 록을 처리해야 하는 방법과 CPython 런타임에 저장된 모든 상태 모두에 구체적인 영향을 미칩니다.

The fact that only the "current" thread remains means any locks held by other threads will never be released. Python solves this for `os.fork()` by acquiring the locks it uses internally before the fork, and releasing them afterwards. In addition, it resets any lock-objects in the child. When extending or embedding Python, there is no way to inform Python of additional (non-Python) locks that need to be acquired before or reset after a fork. OS facilities such as `pthread_atfork()` would need to be used to accomplish the same thing. Additionally, when extending or embedding Python, calling `fork()` directly rather than through `os.fork()` (and returning to or calling into Python) may result in a deadlock by one of Python' s internal locks being held by a thread that is defunct after the fork. *`PyOS_AfterFork_Child()`* tries to reset the necessary locks, but is not always able to.

다른 모든 스레드가 사라진다는 사실은 또한 CPython의 런타임 상태가 `os.fork()`와 마찬가지로 적절하게 정리되어야 함을 의미합니다. 이것은 현재 인터프리터와 다른 모든 *`PyInterpreterState`* 객체에 속하는

다른 모든 *PyThreadState* 객체를 파이널리제이션 하는 것을 의미합니다. 이것과 "메인" 인터프리터의 특수한 특성으로 인해, `fork()` 는 CPython 전역 런타임이 원래 초기화된 인터프리터의 "메인" 스레드에서만 호출되어야 합니다. 유일한 예외는 `exec()` 가 그 후에 즉시 호출되는 경우입니다.

9.5.4 고수준 API

다음은 C 확장 코드를 작성하거나 파이썬 인터프리터를 내장할 때 가장 일반적으로 사용되는 형과 함수입니다:

type **PyInterpreterState**

Part of the Limited API (*as an opaque struct*). 이 자료 구조는 여러 협력 스레드가 공유하는 상태를 나타냅니다. 같은 인터프리터에 속하는 스레드는 모듈 관리와 몇 가지 다른 내부 항목을 공유합니다. 이 구조체에는 공개 멤버가 없습니다.

다른 인터프리터에 속한 스레드는 사용 가능한 메모리, 열린 파일 기술자 등과 같은 프로세스 상태를 제외하고는, 처음에는 아무것도 공유하지 않습니다. 전역 인터프리터 록은 어떤 인터프리터에 속해 있는지에 관계없이 모든 스레드에서 공유됩니다.

type **PyThreadState**

Part of the Limited API (*as an opaque struct*). This data structure represents the state of a single thread. The only public data member is:

PyInterpreterState ***interp**

This thread' s interpreter state.

void **PyEval_InitThreads**()

Part of the Stable ABI. 아무것도 하지 않는 폐지된 함수.

파이썬 3.6과 이전 버전에서는, 이 함수가 존재하지 않으면 GIL을 만들었습니다.

버전 3.9에서 변경: 이제 이 함수는 아무 작업도 수행하지 않습니다.

버전 3.7에서 변경: 이 함수는 이제 *Py_Initialize()* 에 의해 호출되어서, 여러분은 더는 직접 호출할 필요가 없습니다.

버전 3.2에서 변경: 이 함수는 더는 *Py_Initialize()* 전에 호출할 수 없습니다.

버전 3.9부터 폐지됨.

int **PyEval_ThreadsInitialized**()

Part of the Stable ABI. *PyEval_InitThreads()* 가 호출되었으면, 0이 아닌 값을 반환합니다. 이 함수는 GIL을 보유하지 않고 호출할 수 있어서, 단일 스레드를 실행할 때 록 API 호출을 회피하는 데 사용할 수 있습니다.

버전 3.7에서 변경: *GIL*은 이제 *Py_Initialize()* 에 의해 초기화됩니다.

버전 3.9부터 폐지됨.

PyThreadState ***PyEval_SaveThread**()

Part of the Stable ABI. (만들었다면) 전역 인터프리터 록을 해제하고 스레드 상태를 `NULL` 로 재설정하고, 이전 스레드 상태(`NULL` 이 아닙니다)를 반환합니다. 록이 만들어졌다면, 현재 스레드가 록을 획득했어야 합니다.

void **PyEval_RestoreThread**(*PyThreadState* *tstate)

Part of the Stable ABI. (만들었다면) 전역 인터프리터 록을 획득하고 스레드 상태를 `NULL` 이 아니어야 하는 *tstate* 로 설정합니다. 록이 만들어졌다면, 현재 스레드가 이를 획득하지 않았어야 합니다, 그렇지 않으면 교착 상태가 발생합니다.

> **참고**
>
> Calling this function from a thread when the runtime is finalizing will terminate the thread, even if the thread was not created by Python. You can use `_Py_IsFinalizing()` or `sys.is_finalizing()`

to check if the interpreter is in process of being finalized before calling this function to avoid unwanted termination.

PyThreadState ***PyThreadState_Get**()

Part of the Stable ABI. 현재 스레드 상태를 반환합니다. 전역 인터프리터 록을 보유해야 합니다. 현재 스레드 상태가 `NULL`이면 치명적인 에러가 발생합니다 (그래서 호출자가 `NULL`을 확인할 필요가 없습니다).

PyThreadState ***PyThreadState_Swap**(*PyThreadState* *tstate)

Part of the Stable ABI. 현재 스레드 상태를 인자 *tstate*(`NULL`일 수 있습니다)가 제공하는 스레드 상태와 스와프합니다. 전역 인터프리터 록을 보유해야 하며 해제되지 않습니다.

다음 함수는 스레드 로컬 저장소를 사용하며, 서브 인터프리터와 호환되지 않습니다:

PyGILState_STATE **PyGILState_Ensure**()

Part of the Stable ABI. 현재 스레드가 파이썬의 현재 상태나 전역 인터프리터 록과 관계없이 파이썬 C API를 호출할 준비가 되었는지 확인합니다. 이것은 각 호출이 *PyGILState_Release()*에 대한 호출과 쌍을 이루는 한 스레드에서 원하는 만큼 여러 번 호출될 수 있습니다. 일반적으로, 스레드 상태가 Release() 전에 이전 상태로 복원되는 한 *PyGILState_Ensure()*와 *PyGILState_Release()* 호출 간에 다른 스레드 관련 API를 사용할 수 있습니다. 예를 들어, *Py_BEGIN_ALLOW_THREADS*와 *Py_END_ALLOW_THREADS* 매크로의 정상적인 사용은 허용됩니다.

반환 값은 *PyGILState_Ensure()*가 호출되었을 때의 스레드 상태에 대한 불투명한 "핸들"이며, 파이썬이 같은 상태에 있도록 하려면 *PyGILState_Release()*로 전달되어야 합니다. 재귀 호출이 허용되더라도, 이 핸들들은 공유할 수 없습니다 - *PyGILState_Ensure()*에 대한 각 고유 호출은 자신의 *PyGILState_Release()*에 대한 호출을 위해 핸들을 저장해야 합니다.

함수가 반환할 때, 현재 스레드는 GIL을 보유하고 임의의 파이썬 코드를 호출할 수 있습니다. 실패는 치명적인 에러입니다.

참고

Calling this function from a thread when the runtime is finalizing will terminate the thread, even if the thread was not created by Python. You can use `_Py_IsFinalizing()` or `sys.is_finalizing()` to check if the interpreter is in process of being finalized before calling this function to avoid unwanted termination.

void **PyGILState_Release**(PyGILState_STATE)

Part of the Stable ABI. 이전에 획득 한 모든 자원을 해제합니다. 이 호출 후에, 파이썬의 상태는 해당 *PyGILState_Ensure()* 호출 이전과 같습니다 (그러나 일반적으로 이 상태는 호출자에게 알려지지 않아서, GILState API를 사용합니다).

*PyGILState_Ensure()*에 대한 모든 호출은 같은 스레드에서 *PyGILState_Release()*에 대한 호출과 쌍을 이뤄야 합니다.

PyThreadState ***PyGILState_GetThisThreadState**()

Part of the Stable ABI. 이 스레드의 현재 스레드 상태를 가져옵니다. 현재 스레드에서 GILState API가 사용되지 않았으면 `NULL`을 반환할 수 있습니다. 메인 스레드에서 자동 스레드 상태 호출(auto-thread-state call)이 수행되지 않은 경우에도, 메인 스레드에는 항상 이러한 스레드 상태가 있음에 유의하십시오. 이것은 주로 도우미/진단 함수입니다.

int **PyGILState_Check**()

현재 스레드가 GIL을 보유하고 있으면 `1`을 반환하고 그렇지 않으면 `0`을 반환합니다. 이 함수는 아무 때나 모든 스레드에서 호출할 수 있습니다. 파이썬 스레드 상태가 초기화되었고 현재 GIL을 보유하고 있을 때만 `1`을 반환합니다. 이것은 주로 도우미/진단 함수입니다. 예를 들어 콜백 컨텍스트나 메모리 할당 함수에서 유용할 수 있는데, GIL이 잠겨 있다는 것을 알면 호출자가 민감한 작업을 수행하거나 그렇지 않으면 다르게 동작하도록 할 수 있습니다.

Added in version 3.4.

다음 매크로는 일반적으로 후행 세미콜론 없이 사용됩니다; 파이썬 소스 배포에서 사용 예를 찾으십시오.

Py_BEGIN_ALLOW_THREADS

Part of the Stable ABI. 이 매크로는 `{ PyThreadState *_save; _save = PyEval_SaveThread();` 로 확장됩니다. 여는 중괄호가 포함되어 있음에 유의하십시오; 뒤따르는 *Py_END_ALLOW_THREADS* 매크로와 일치해야 합니다. 이 매크로에 대한 자세한 내용은 위를 참조하십시오.

Py_END_ALLOW_THREADS

Part of the Stable ABI. 이 매크로는 `PyEval_RestoreThread(_save); }`로 확장됩니다. 닫는 중괄호가 포함되어 있음에 유의하십시오; 이전 *Py_BEGIN_ALLOW_THREADS* 매크로와 일치해야 합니다. 이 매크로에 대한 자세한 내용은 위를 참조하십시오.

Py_BLOCK_THREADS

Part of the Stable ABI. 이 매크로는 `PyEval_RestoreThread(_save);` 로 확장됩니다: 닫는 중괄호가 없는 *Py_END_ALLOW_THREADS*와 동등합니다.

Py_UNBLOCK_THREADS

Part of the Stable ABI. 이 매크로는 `_save = PyEval_SaveThread();` 로 확장됩니다: 여는 중괄호와 변수 선언이 없는 *Py_BEGIN_ALLOW_THREADS*와 동등합니다.

9.5.5 저수준 API

다음 함수는 모두 *Py_Initialize()* 이후에 호출되어야 합니다.

버전 3.7에서 변경: *Py_Initialize()*는 이제 *GIL*을 초기화합니다.

PyInterpreterState ***PyInterpreterState_New**()

Part of the Stable ABI. 새 인터프리터 상태 객체를 만듭니다. 전역 인터프리터 록을 보유할 필요는 없지만, 이 함수에 대한 호출을 직렬화해야 하면 보유할 수 있습니다.

인자 없이 감사 이벤트 `cpython.PyInterpreterState_New`를 발생시킵니다.

void **PyInterpreterState_Clear**(*PyInterpreterState* *interp)

Part of the Stable ABI. 인터프리터 상태 객체의 모든 정보를 재설정합니다. 전역 인터프리터 록을 보유해야 합니다.

인자 없이 감사 이벤트 `cpython.PyInterpreterState_Clear`를 발생시킵니다.

void **PyInterpreterState_Delete**(*PyInterpreterState* *interp)

Part of the Stable ABI. 인터프리터 상태 객체를 파괴합니다. 전역 인터프리터 록은 보유할 필요 없습니다. 인터프리터 상태는 *PyInterpreterState_Clear()*에 대한 이전 호출로 재설정되었어야 합니다.

PyThreadState ***PyThreadState_New**(*PyInterpreterState* *interp)

Part of the Stable ABI. 주어진 인터프리터 객체에 속하는 새 스레드 상태 객체를 만듭니다. 전역 인터프리터 록을 보유할 필요는 없지만, 이 함수에 대한 호출을 직렬화해야 하면 보유할 수 있습니다.

void **PyThreadState_Clear**(*PyThreadState* *tstate)

Part of the Stable ABI. 스레드 상태 객체의 모든 정보를 재설정합니다. 전역 인터프리터 록을 보유해야 합니다.

버전 3.9에서 변경: 이 함수는 이제 `PyThreadState.on_delete` 콜백을 호출합니다. 이전에는, *PyThreadState_Delete()*에서 호출했습니다.

void **PyThreadState_Delete**(*PyThreadState* *tstate)

Part of the Stable ABI. 스레드 상태 객체를 파괴합니다. 전역 인터프리터 록은 보유할 필요 없습니다. 스레드 상태는 *PyThreadState_Clear()*에 대한 이전 호출로 재설정되었어야 합니다.

void **PyThreadState_DeleteCurrent**(void)

Destroy the current thread state and release the global interpreter lock. Like *PyThreadState_Delete()*, the global interpreter lock must be held. The thread state must have been reset with a previous call to *PyThreadState_Clear()*.

PyFrameObject ***PyThreadState_GetFrame**(*PyThreadState* *tstate)

Part of the Stable ABI *since version 3.10.* 파이썬 스레드 상태 *tstate*의 현재 프레임을 가져옵니다.

Return a *strong reference*. Return `NULL` if no frame is currently executing.

*PyEval_GetFrame()*도 참조하십시오.

*tstate*는 `NULL`이 아니어야 합니다.

Added in version 3.9.

uint64_t **PyThreadState_GetID**(*PyThreadState* *tstate)

Part of the Stable ABI *since version 3.10.* 파이썬 스레드 상태 *tstate*의 고유한 스레드 상태 식별자를 가져옵니다.

*tstate*는 `NULL`이 아니어야 합니다.

Added in version 3.9.

PyInterpreterState ***PyThreadState_GetInterpreter**(*PyThreadState* *tstate)

Part of the Stable ABI *since version 3.10.* 파이썬 스레드 상태 *tstate*의 인터프리터를 가져옵니다.

*tstate*는 `NULL`이 아니어야 합니다.

Added in version 3.9.

void **PyThreadState_EnterTracing**(*PyThreadState* *tstate)

Suspend tracing and profiling in the Python thread state *tstate*.

Resume them using the *PyThreadState_LeaveTracing()* function.

Added in version 3.11.

void **PyThreadState_LeaveTracing**(*PyThreadState* *tstate)

Resume tracing and profiling in the Python thread state *tstate* suspended by the *PyThreadState_EnterTracing()* function.

See also *PyEval_SetTrace()* and *PyEval_SetProfile()* functions.

Added in version 3.11.

PyInterpreterState ***PyInterpreterState_Get**(void)

Part of the Stable ABI *since version 3.9.* 현재 인터프리터를 가져옵니다.

현재 파이썬 스레드 상태가 없거나 현재 인터프리터가 없으면 치명적인 에러를 발행합니다. NULL을 반환할 수 없습니다.

호출자는 GIL을 보유해야 합니다.

Added in version 3.9.

int64_t **PyInterpreterState_GetID**(*PyInterpreterState* *interp)

Part of the Stable ABI *since version 3.7.* 인터프리터의 고유 ID를 반환합니다. 그렇게 하는데 에러가 발생하면 `-1`이 반환되고 에러가 설정됩니다.

호출자는 GIL을 보유해야 합니다.

Added in version 3.7.

PyObject ***PyInterpreterState_GetDict**(*PyInterpreterState* *interp)

Part of the Stable ABI *since version 3.8.* 인터프리터별 데이터가 저장될 수 있는 딕셔너리를 반환합니다. 이 함수가 `NULL`을 반환하면 예외는 발생하지 않았고 호출자는 인터프리터별 딕셔너리를 사용할 수 없다고 가정해야 합니다.

이것은 확장이 인터프리터별 상태 정보를 저장하는 데 사용해야 하는 *PyModule_GetState()*를 대체하는 것이 아닙니다.

Added in version 3.8.

typedef *PyObject* *(***_PyFrameEvalFunction**)(*PyThreadState* *tstate, *_PyInterpreterFrame* *frame, int throwflag)

프레임 평가 함수의 형.

throwflag 매개 변수는 제너레이터의 `throw()` 메서드에서 사용됩니다: 0이 아니면, 현재 예외를 처리합니다.

버전 3.9에서 변경: 이제 함수는 *tstate* 매개 변수를 취합니다.

버전 3.11에서 변경: The *frame* parameter changed from `PyFrameObject*` to `_PyInterpreterFrame*`.

_PyFrameEvalFunction **_PyInterpreterState_GetEvalFrameFunc**(*PyInterpreterState* *interp)

프레임 평가 함수를 가져옵니다.

PEP 523 "CPython에 프레임 평가 API 추가"를 참조하십시오.

Added in version 3.9.

void **_PyInterpreterState_SetEvalFrameFunc**(*PyInterpreterState* *interp, *_PyFrameEvalFunction* eval_frame)

프레임 평가 함수를 설정합니다.

PEP 523 "CPython에 프레임 평가 API 추가"를 참조하십시오.

Added in version 3.9.

PyObject ***PyThreadState_GetDict**()

Return value: Borrowed reference. Part of the Stable ABI. 확장이 스레드별 상태 정보를 저장할 수 있는 딕셔너리를 반환합니다. 각 확장은 딕셔너리에 상태를 저장하는 데 사용할 고유 키를 사용해야 합니다. 현재 스레드 상태를 사용할 수 없을 때 이 함수를 호출해도 됩니다. 이 함수가 `NULL`을 반환하면, 예외는 발생하지 않았고 호출자는 현재 스레드 상태를 사용할 수 없다고 가정해야 합니다.

int **PyThreadState_SetAsyncExc**(unsigned long id, *PyObject* *exc)

Part of the Stable ABI. Asynchronously raise an exception in a thread. The *id* argument is the thread id of the target thread; *exc* is the exception object to be raised. This function does not steal any references to *exc*. To prevent naive misuse, you must write your own C extension to call this. Must be called with the GIL held. Returns the number of thread states modified; this is normally one, but will be zero if the thread id isn't found. If *exc* is `NULL`, the pending exception (if any) for the thread is cleared. This raises no exceptions.

버전 3.7에서 변경: The type of the *id* parameter changed from `long` to `unsigned long`.

void **PyEval_AcquireThread**(*PyThreadState* *tstate)

Part of the Stable ABI. 전역 인터프리터 록을 획득하고 현재 스레드 상태를 *tstate*로 설정합니다. *tstate*는 `NULL`이 아니어야 합니다. 록은 이전에 만들어진 것이어야 합니다. 이 스레드에 이미 록이 있으면, 교착 상태가 발생합니다.

> **참고**
>
> Calling this function from a thread when the runtime is finalizing will terminate the thread, even if the thread was not created by Python. You can use `_Py_IsFinalizing()` or `sys.is_finalizing()` to check if the interpreter is in process of being finalized before calling this function to avoid unwanted termination.

버전 3.8에서 변경: *PyEval_RestoreThread()*, *Py_END_ALLOW_THREADS()* 및 *PyGILState_Ensure()*와 일관되도록 갱신되었으며, 인터프리터가 파이널리제이션 하는 동안 호출되면 현재 스레드를 종료합니다.

*PyEval_RestoreThread()*는 (스레드가 초기화되지 않았을 때조차) 항상 사용할 수 있는 고수준 함수입니다.

void **PyEval_ReleaseThread**(*PyThreadState* *tstate)

Part of the Stable ABI. 현재 스레드 상태를 `NULL`로 재설정하고 전역 인터프리터 록을 해제합니다. 록은 이전에 만들어졌어야 하고 현재 스레드가 보유해야 합니다. `NULL`이 아니어야 하는 *tstate* 인자는

현재 스레드 상태를 나타내는지 확인하는 데만 사용됩니다 — 그렇지 않으면, 치명적인 에러가 보고됩니다.

*PyEval_SaveThread()*는 (스레드가 초기화되지 않은 경우에조차) 항상 사용할 수 있는 고수준 함수입니다.

void **PyEval_AcquireLock**()

Part of the Stable ABI. 전역 인터프리터 록을 획득합니다. 록은 이전에 만들어졌어야 합니다. 이 스레드에 이미 록이 있으면, 교착 상태가 발생합니다.

버전 3.2부터 폐지됨: 이 함수는 현재 스레드 상태를 갱신하지 않습니다. 대신 *PyEval_RestoreThread()*나 *PyEval_AcquireThread()*를 사용하십시오.

> **참고**
>
> 런타임이 파이널리제이션 될 때 스레드에서 이 함수를 호출하면, 스레드가 파이썬에 의해 만들어지지 않았더라도 스레드가 종료됩니다. 원치 않는 종료를 방지하려면 `_Py_IsFinalizing()`이나 `sys.is_finalizing()`을 사용하여 이 함수를 호출하기 전에 인터프리터가 파이널리제이션 되고 있는지 확인할 수 있습니다.

버전 3.8에서 변경: *PyEval_RestoreThread()*, *Py_END_ALLOW_THREADS()* 및 *PyGILState_Ensure()*와 일관되도록 갱신되었으며, 인터프리터가 파이널리제이션 하는 동안 호출되면 현재 스레드를 종료합니다.

void **PyEval_ReleaseLock**()

Part of the Stable ABI. 전역 인터프리터 록을 해제합니다. 록은 이전에 만들어졌어야 합니다.

버전 3.2부터 폐지됨: 이 함수는 현재 스레드 상태를 갱신하지 않습니다. 대신 *PyEval_SaveThread()*나 *PyEval_ReleaseThread()*를 사용하십시오.

9.6 서브 인터프리터 지원

대부분의 경우, 단일 파이썬 인터프리터만 내장할 것입니다만, 같은 프로세스, 어쩌면 같은 스레드에서 여러 독립 인터프리터를 만들어야 하는 경우가 있습니다. 서브 인터프리터는 그렇게 할 수 있도록 합니다.

"메인" 인터프리터는 런타임이 초기화될 때 만들어지는 첫 번째 인터프리터입니다. 보통은 프로세스에서 유일한 파이썬 인터프리터입니다. 서브 인터프리터와 달리, 메인 인터프리터는 시그널 처리와 같은 고유한 프로세스 전역 책임을 갖습니다. 또한 런타임 초기화 동안 실행을 담당하며 일반적으로 런타임 파이널리제이션 동안 활성 인터프리터입니다. *PyInterpreterState_Main()* 함수는 그것의 상태에 대한 포인터를 반환합니다.

PyThreadState_Swap() 함수를 사용하여 서브 인터프리터 간에 전환할 수 있습니다. 다음 함수를 사용하여 만들고 파괴할 수 있습니다:

type **PyInterpreterConfig**

Structure containing most parameters to configure a sub-interpreter. Its values are used only in *Py_NewInterpreterFromConfig()* and never modified by the runtime.

Added in version 3.12.

Structure fields:

int **use_main_obmalloc**

If this is `0` then the sub-interpreter will use its own "object" allocator state. Otherwise it will use (share) the main interpreter's.

If this is `0` then *check_multi_interp_extensions* must be `1` (non-zero). If this is `1` then *gil* must not be *PyInterpreterConfig_OWN_GIL*.

int **allow_fork**

If this is `0` then the runtime will not support forking the process in any thread where the sub-interpreter is currently active. Otherwise fork is unrestricted.

Note that the `subprocess` module still works when fork is disallowed.

int **allow_exec**

If this is `0` then the runtime will not support replacing the current process via exec (e.g. `os.execv()`) in any thread where the sub-interpreter is currently active. Otherwise exec is unrestricted.

Note that the `subprocess` module still works when exec is disallowed.

int **allow_threads**

If this is `0` then the sub-interpreter' s `threading` module won' t create threads. Otherwise threads are allowed.

int **allow_daemon_threads**

If this is `0` then the sub-interpreter' s `threading` module won' t create daemon threads. Otherwise daemon threads are allowed (as long as *allow_threads* is non-zero).

int **check_multi_interp_extensions**

If this is `0` then all extension modules may be imported, including legacy (single-phase init) modules, in any thread where the sub-interpreter is currently active. Otherwise only multi-phase init extension modules (see **PEP 489**) may be imported. (Also see *Py_mod_multiple_interpreters*.)

This must be `1` (non-zero) if *use_main_obmalloc* is `0`.

int **gil**

This determines the operation of the GIL for the sub-interpreter. It may be one of the following:

PyInterpreterConfig_DEFAULT_GIL

Use the default selection (*PyInterpreterConfig_SHARED_GIL*).

PyInterpreterConfig_SHARED_GIL

Use (share) the main interpreter' s GIL.

PyInterpreterConfig_OWN_GIL

Use the sub-interpreter' s own GIL.

If this is *PyInterpreterConfig_OWN_GIL* then *PyInterpreterConfig.use_main_obmalloc* must be `0`.

PyStatus **Py_NewInterpreterFromConfig**(*PyThreadState* **tstate_p, const *PyInterpreterConfig* *config)

새 서브 인터프리터를 만듭니다. 이것은 파이썬 코드 실행을 위한 (거의) 완전히 분리된 환경입니다. 특히, 새 인터프리터에는 기본 모듈 `builtins`, `__main__` 및 `sys`를 포함하여, 모든 임포트 된 모듈의 개별, 독립 버전을 갖습니다. 로드된 모듈 테이블(`sys.modules`)과 모듈 검색 경로(`sys.path`)도 별개입니다. 새 환경에는 `sys.argv` 변수가 없습니다. 새로운 표준 I/O 스트림 파일 객체 `sys.stdin`, `sys.stdout` 및 `sys.stderr`을 갖습니다 (단, 같은 하부 파일 기술자를 참조합니다).

The given *config* controls the options with which the interpreter is initialized.

Upon success, *tstate_p* will be set to the first thread state created in the new sub-interpreter. This thread state is made in the current thread state. Note that no actual thread is created; see the discussion of thread states below. If creation of the new interpreter is unsuccessful, *tstate_p* is set to `NULL`; no exception is set since the exception state is stored in the current thread state and there may not be a current thread state.

Like all other Python/C API functions, the global interpreter lock must be held before calling this function and is still held when it returns. Likewise a current thread state must be set on entry. On success, the returned thread state will be set as current. If the sub-interpreter is created with its own GIL then the GIL of the calling interpreter will be released. When the function returns, the new interpreter' s GIL will be held by the current thread and the previously interpreter' s GIL will remain released here.

Added in version 3.12.

Sub-interpreters are most effective when isolated from each other, with certain functionality restricted:

```
PyInterpreterConfig config = {
    .use_main_obmalloc = 0,
    .allow_fork = 0,
    .allow_exec = 0,
    .allow_threads = 1,
    .allow_daemon_threads = 0,
    .check_multi_interp_extensions = 1,
    .gil = PyInterpreterConfig_OWN_GIL,
};
PyThreadState *tstate = NULL;
PyStatus status = Py_NewInterpreterFromConfig(&tstate, &config);
if (PyStatus_Exception(status)) {
    Py_ExitStatusException(status);
}
```

Note that the config is used only briefly and does not get modified. During initialization the config's values are converted into various *PyInterpreterState* values. A read-only copy of the config may be stored internally on the *PyInterpreterState*.

확장 모듈은 다음과 같이 (서브) 인터프리터 간에 공유됩니다:

- 다단계 초기화를 사용하는 모듈의 경우, 예를 들어 *PyModule_FromDefAndSpec()*, 각 인터프리터에 대해 별도의 모듈 객체가 만들어지고 초기화됩니다. C 수준 정적과 전역 변수만 이러한 모듈 객체 간에 공유됩니다.
- 단단계 초기화를 사용하는 모듈의 경우, 예를 들어 *PyModule_Create()*, 특정 확장이 처음 임포트 될 때, 정상적으로 초기화되고, 모듈 딕셔너리의 (얕은) 사본이 저장됩니다. 다른 (서브) 인터프리터가 같은 확장을 임포트 할 때, 새 모듈이 초기화되고 이 복사본의 내용으로 채워집니다; 확장의 `init` 함수는 호출되지 않습니다. 따라서 모듈 딕셔너리의 객체는 (서브) 인터프리터 간에 공유되어, 원치 않는 동작을 일으킬 수 있습니다 (아래 버그와 주의 사항을 참조하십시오).

  이것은 인터프리터가 *Py_FinalizeEx()*와 *Py_Initialize()*를 호출하여 완전히 다시 초기화된 후 확장을 임포트 할 때 일어나는 것과 다름에 유의하십시오; 이 경우, 확장의 `initmodule` 함수가 다시 호출됩니다. 다단계 초기화와 마찬가지로, 이는 C 수준의 정적과 전역 변수만 이러한 모듈 간에 공유됨을 의미합니다.

PyThreadState ***Py_NewInterpreter**(void)

Part of the Stable ABI. Create a new sub-interpreter. This is essentially just a wrapper around *Py_NewInterpreterFromConfig()* with a config that preserves the existing behavior. The result is an unisolated sub-interpreter that shares the main interpreter's GIL, allows fork/exec, allows daemon threads, and allows single-phase init modules.

void **Py_EndInterpreter**(*PyThreadState* *tstate)

Part of the Stable ABI. Destroy the (sub-)interpreter represented by the given thread state. The given thread state must be the current thread state. See the discussion of thread states below. When the call returns, the current thread state is `NULL`. All thread states associated with this interpreter are destroyed. The global interpreter lock used by the target interpreter must be held before calling this function. No GIL is held when it returns.

Py_FinalizeEx() will destroy all sub-interpreters that haven't been explicitly destroyed at that point.

9.6.1 A Per-Interpreter GIL

Using *Py_NewInterpreterFromConfig()* you can create a sub-interpreter that is completely isolated from other interpreters, including having its own GIL. The most important benefit of this isolation is that such an interpreter can execute Python code without being blocked by other interpreters or blocking any others. Thus a single Python process can truly take advantage of multiple CPU cores when running Python code. The isolation also encourages a different approach to concurrency than that of just using threads. (See **PEP 554**.)

Using an isolated interpreter requires vigilance in preserving that isolation. That especially means not sharing any objects or mutable state without guarantees about thread-safety. Even objects that are otherwise immutable (e.g.

`None`, `(1, 5)`) can' t normally be shared because of the refcount. One simple but less-efficient approach around this is to use a global lock around all use of some state (or object). Alternately, effectively immutable objects (like integers or strings) can be made safe in spite of their refcounts by making them "immortal". In fact, this has been done for the builtin singletons, small integers, and a number of other builtin objects.

If you preserve isolation then you will have access to proper multi-core computing without the complications that come with free-threading. Failure to preserve isolation will expose you to the full consequences of free-threading, including races and hard-to-debug crashes.

Aside from that, one of the main challenges of using multiple isolated interpreters is how to communicate between them safely (not break isolation) and efficiently. The runtime and stdlib do not provide any standard approach to this yet. A future stdlib module would help mitigate the effort of preserving isolation and expose effective tools for communicating (and sharing) data between interpreters.

Added in version 3.12.

9.6.2 버그와 주의 사항

서브 인터프리터(및 메인 인터프리터)는 같은 프로세스의 일부이기 때문에, 그들 간의 절연이 완벽하지 않습니다 — 예를 들어, `os.close()`와 같은 저수준 파일 연산을 사용하면 서로의 열린 파일에 (실수로 혹은 악의적으로) 영향을 미칠 수 있습니다. (서브) 인터프리터 간에 확장이 공유되는 방식 때문에, 일부 확장이 제대로 작동하지 않을 수 있습니다; 이것은 특히 단단계 초기화나 (정적) 전역 변수를 사용할 때 특히 그렇습니다. 한 서브 인터프리터에서 만든 객체를 다른 (서브) 인터프리터의 이름 공간에 삽입할 수 있습니다; 가능하면 피해야 합니다.

서브 인터프리터 간에 사용자 정의 함수, 메서드, 인스턴스 또는 클래스를 공유하지 않도록 특별한 주의를 기울여야 합니다. 이러한 객체에 의해 실행되는 임포트 연산은 잘못된 (서브) 인터프리터의 로드된 모듈 딕셔너리에 영향을 미칠 수 있기 때문입니다. 위의 것들에서 접근할 수 있는 객체를 공유하지 않는 것도 마찬가지로 중요합니다.

Also note that combining this functionality with `PyGILState_*` APIs is delicate, because these APIs assume a bijection between Python thread states and OS-level threads, an assumption broken by the presence of sub-interpreters. It is highly recommended that you don' t switch sub-interpreters between a pair of matching *`PyGILState_Ensure()`* and *`PyGILState_Release()`* calls. Furthermore, extensions (such as `ctypes`) using these APIs to allow calling of Python code from non-Python created threads will probably be broken when using sub-interpreters.

9.7 비동기 알림

메인 인터프리터 스레드에 비동기 알림을 보내는 메커니즘이 제공됩니다. 이러한 알림은 함수 포인터와 void 포인터 인자의 형태를 취합니다.

int **`Py_AddPendingCall`**(int (*func)(void*), void *arg)

Part of the Stable ABI. 메인 인터프리터 스레드에서 호출할 함수를 예약합니다. 성공하면 `0`이 반환되고 *func*는 메인 스레드에서 호출되기 위해 큐에 추가됩니다. 실패 시, 예외 설정 없이 `-1`이 반환됩니다.

성공적으로 큐에 넣으면, *func*는 *arg* 인자를 사용하여 결국 메인 인터프리터 스레드에서 호출됩니다. 정상적으로 실행되는 파이썬 코드와 비교할 때 비동기적으로 호출되지만, 다음 두 조건이 모두 충족됩니다:

- 바이트 코드 경계에서;
- 메인 스레드가 전역 인터프리터 록을 보유하면서 (따라서 *func*는 전체 C API를 사용할 수 있습니다).

*func*는 성공하면 `0`을, 실패하면 예외 설정과 함께 `-1`을 반환해야 합니다. *func*는 다른 비동기 알림을 재귀적으로 수행하기 위해 중단되지 않지만, 전역 인터프리터 록이 해제되면 스레드를 전환하기 위해 여전히 중단될 수 있습니다.

이 함수는 실행하는 데 현재 스레드 상태가 필요하지 않으며, 전역 인터프리터 록이 필요하지 않습니다.

서브 인터프리터에서 이 함수를 호출하려면 호출자가 GIL을 보유해야 합니다. 그렇지 않으면, 함수 *func*가 잘못된 인터프리터에서 호출되도록 예약될 수 있습니다.

> **경고**
>
> 이것은 매우 특별한 경우에만 유용한, 저수준 함수입니다. *func*가 가능한 한 빨리 호출된다는 보장은 없습니다. 메인 스레드가 시스템 호출을 실행 중이라 바쁘면, 시스템 호출이 반환되기 전에 *func*가 호출되지 않습니다. 이 함수는 일반적으로 임의의 C 스레드에서 파이썬 코드를 호출하는 데 적합하지 **않습니다**. 대신, *PyGILState API*를 사용하십시오.

Added in version 3.1.

버전 3.9에서 변경: 이 함수가 서브 인터프리터에서 호출되면, *func* 함수는 이제 메인 인터프리터에서 호출되지 않고 서브 인터프리터에서 호출되도록 예약됩니다. 이제 각 서브 인터프리터는 자체 예약된 호출 목록을 갖습니다.

9.8 프로파일링과 추적

파이썬 인터프리터는 프로파일링과 실행 추적 기능을 연결하기 위한 몇 가지 저수준 지원을 제공합니다. 프로파일링, 디버깅 및 커버리지 분석 도구에 사용됩니다.

이 C 인터페이스를 사용하면 프로파일링이나 추적 코드가 파이썬 수준의 콜러블 객체를 통해 호출하는 오버헤드를 피하고, 대신 직접 C 함수를 호출할 수 있습니다. 시설의 필수 어트리뷰트는 변경되지 않았습니다; 인터페이스는 추적 함수를 스레드별로 설치할 수 있도록 하며, 추적 함수에 보고되는 기본 이벤트는 이전 버전의 파이썬 수준 추적 함수에 보고된 것과 같습니다.

typedef int (***Py_tracefunc**)(*PyObject* *obj, *PyFrameObject* *frame, int what, *PyObject* *arg)

The type of the trace function registered using `PyEval_SetProfile()` and `PyEval_SetTrace()`. The first parameter is the object passed to the registration function as *obj*, *frame* is the frame object to which the event pertains, *what* is one of the constants `PyTrace_CALL`, `PyTrace_EXCEPTION`, `PyTrace_LINE`, `PyTrace_RETURN`, `PyTrace_C_CALL`, `PyTrace_C_EXCEPTION`, `PyTrace_C_RETURN`, or `PyTrace_OPCODE`, and *arg* depends on the value of *what*:

*what*의 값	*arg*의 의미
`PyTrace_CALL`	항상 `Py_None`.
`PyTrace_EXCEPTION`	`sys.exc_info()`에서 반환된 예외 정보.
`PyTrace_LINE`	항상 `Py_None`.
`PyTrace_RETURN`	호출자에게 반환되는 값, 또는 예외로 인한 것이면 `NULL`.
`PyTrace_C_CALL`	호출되는 함수 객체.
`PyTrace_C_EXCEPTION`	호출되는 함수 객체.
`PyTrace_C_RETURN`	호출되는 함수 객체.
`PyTrace_OPCODE`	항상 `Py_None`.

int **PyTrace_CALL**

함수나 메서드에 대한 새 호출이 보고되거나, 제너레이터에 대한 새 항목이 보고될 때 `Py_tracefunc` 함수에 대한 *what* 매개 변수의 값. 제너레이터 함수에 대한 이터레이터의 생성은 해당 프레임의 파이썬 바이트 코드로의 제어 전송이 없기 때문에 보고되지 않음에 유의하십시오.

int **PyTrace_EXCEPTION**

예외가 발생했을 때 `Py_tracefunc` 함수에 대한 *what* 매개 변수의 값. 콜백 함수는 실행되는 프레임 내에서 바이트 코드가 처리된 후 예외가 설정될 때 *what*에 대해 이 값으로 호출됩니다. 이것의 효과는 예외 전파로 인해 파이썬 스택이 되감기는 것입니다, 예외가 전파되어 각 프레임으로 반환할 때 콜백이 호출됩니다. 추적 함수만 이러한 이벤트를 수신합니다; 프로파일러에는 필요하지 않습니다.

int **PyTrace_LINE**

The value passed as the *what* parameter to a `Py_tracefunc` function (but not a profiling function) when a

line-number event is being reported. It may be disabled for a frame by setting `f_trace_lines` to *0* on that frame.

int **PyTrace_RETURN**

호출이 반환되려고 할 때 *Py_tracefunc* 함수에 대한 *what* 매개 변수의 값.

int **PyTrace_C_CALL**

C 함수가 호출되려고 할 때 *Py_tracefunc* 함수에 대한 *what* 매개 변수의 값.

int **PyTrace_C_EXCEPTION**

C 함수에서 예외가 발생했을 때 *Py_tracefunc* 함수에 대한 *what* 매개 변수의 값.

int **PyTrace_C_RETURN**

C 함수가 반환했을 때 *Py_tracefunc* 함수에 대한 *what* 매개 변수의 값.

int **PyTrace_OPCODE**

The value for the *what* parameter to *Py_tracefunc* functions (but not profiling functions) when a new opcode is about to be executed. This event is not emitted by default: it must be explicitly requested by setting `f_trace_opcodes` to *1* on the frame.

void **PyEval_SetProfile**(*Py_tracefunc* func, *PyObject* *obj)

Set the profiler function to *func*. The *obj* parameter is passed to the function as its first parameter, and may be any Python object, or `NULL`. If the profile function needs to maintain state, using a different value for *obj* for each thread provides a convenient and thread-safe place to store it. The profile function is called for all monitored events except *PyTrace_LINE* *PyTrace_OPCODE* and *PyTrace_EXCEPTION*.

See also the `sys.setprofile()` function.

호출자는 *GIL*을 보유하고 있어야 합니다.

void **PyEval_SetProfileAllThreads**(*Py_tracefunc* func, *PyObject* *obj)

Like *PyEval_SetProfile()* but sets the profile function in all running threads belonging to the current interpreter instead of the setting it only on the current thread.

호출자는 *GIL*을 보유하고 있어야 합니다.

As *PyEval_SetProfile()*, this function ignores any exceptions raised while setting the profile functions in all threads.

Added in version 3.12.

void **PyEval_SetTrace**(*Py_tracefunc* func, *PyObject* *obj)

Set the tracing function to *func*. This is similar to *PyEval_SetProfile()*, except the tracing function does receive line-number events and per-opcode events, but does not receive any event related to C function objects being called. Any trace function registered using *PyEval_SetTrace()* will not receive *PyTrace_C_CALL*, *PyTrace_C_EXCEPTION* or *PyTrace_C_RETURN* as a value for the *what* parameter.

See also the `sys.settrace()` function.

호출자는 *GIL*을 보유하고 있어야 합니다.

void **PyEval_SetTraceAllThreads**(*Py_tracefunc* func, *PyObject* *obj)

Like *PyEval_SetTrace()* but sets the tracing function in all running threads belonging to the current interpreter instead of the setting it only on the current thread.

호출자는 *GIL*을 보유하고 있어야 합니다.

As *PyEval_SetTrace()*, this function ignores any exceptions raised while setting the trace functions in all threads.

Added in version 3.12.

9.9 고급 디버거 지원

이 함수들은 고급 디버깅 도구에서만 사용하기 위한 것입니다.

PyInterpreterState ***PyInterpreterState_Head**()
인터프리터 상태 객체들의 리스트의 머리에 있는 객체를 반환합니다.

PyInterpreterState ***PyInterpreterState_Main**()
메인 인터프리터 상태 객체를 반환합니다.

PyInterpreterState ***PyInterpreterState_Next**(*PyInterpreterState* *interp)
인터프리터 상태 객체들의 리스트에서 *interp* 이후의 다음 인터프리터 상태 객체를 반환합니다.

PyThreadState ***PyInterpreterState_ThreadHead**(*PyInterpreterState* *interp)
인터프리터 *interp*와 관련된 스레드 리스트에서 첫 번째 `PyThreadState` 객체에 대한 포인터를 반환합니다.

PyThreadState ***PyThreadState_Next**(*PyThreadState* *tstate)
같은 `PyInterpreterState` 객체에 속하는 모든 스레드 객체 리스트에서 *tstate* 이후의 다음 스레드 상태 객체를 반환합니다.

9.10 스레드 로컬 저장소 지원

The Python interpreter provides low-level support for thread-local storage (TLS) which wraps the underlying native TLS implementation to support the Python-level thread local storage API (`threading.local`). The CPython C level APIs are similar to those offered by pthreads and Windows: use a thread key and functions to associate a `void*` value per thread.

이러한 함수를 호출할 때 GIL을 보유할 필요는 없습니다; 그들은 자체 록을 제공합니다.

`Python.h`에는 TLS API 선언이 포함되어 있지 않음에 유의하십시오, 스레드 로컬 저장소를 사용하려면 `pythread.h`를 포함해야 합니다.

참고

None of these API functions handle memory management on behalf of the `void*` values. You need to allocate and deallocate them yourself. If the `void*` values happen to be `PyObject*`, these functions don' t do refcount operations on them either.

9.10.1 스레드별 저장소 (TSS - Thread Specific Storage) API

TSS API is introduced to supersede the use of the existing TLS API within the CPython interpreter. This API uses a new type `Py_tss_t` instead of `int` to represent thread keys.

Added in version 3.7.

더 보기

"CPython의 스레드-로컬 저장소를 위한 새로운 C-API" (**PEP 539**)

type **Py_tss_t**
이 자료 구조는 스레드 키의 상태를 나타내며, 정의는 하부 TLS 구현에 따라 달라질 수 있으며, 키의 초기화 상태를 나타내는 내부 필드가 있습니다. 이 구조체에는 공개 멤버가 없습니다.

*Py_LIMITED_API*가 정의되지 않을 때, `Py_tss_NEEDS_INIT`로 이 형의 정적 할당이 허용됩니다.

Py_tss_NEEDS_INIT
이 매크로는 `Py_tss_t` 변수의 초기화자(initializer)로 확장됩니다. 이 매크로는 *Py_LIMITED_API*에서 정의되지 않음에 유의하십시오.

동적 할당

*Py_LIMITED_API*로 빌드된 확장 모듈에 필요한, 빌드 시점에 구현이 불투명해서 형의 정적 할당이 불가능한 `Py_tss_t`의 동적 할당.

Py_tss_t ***PyThread_tss_alloc**()

Part of the Stable ABI *since version 3.7.* `Py_tss_NEEDS_INIT`로 초기화된 값과 같은 상태의 값을 반환하거나, 동적 할당 실패 시 `NULL`을 반환합니다.

void **PyThread_tss_free**(*Py_tss_t* *key)

Part of the Stable ABI *since version 3.7.* Free the given *key* allocated by `PyThread_tss_alloc()`, after first calling `PyThread_tss_delete()` to ensure any associated thread locals have been unassigned. This is a no-op if the *key* argument is `NULL`.

> **참고**
>
> A freed key becomes a dangling pointer. You should reset the key to `NULL`.

메서드

이 함수들의 매개 변수 *key*는 `NULL`이 아니어야 합니다. 또한, 주어진 `Py_tss_t`가 `PyThread_tss_create()`로 초기화되지 않았으면, `PyThread_tss_set()`과 `PyThread_tss_get()`의 동작은 정의되지 않습니다.

int **PyThread_tss_is_created**(*Py_tss_t* *key)

Part of the Stable ABI *since version 3.7.* 주어진 `Py_tss_t`가 `PyThread_tss_create()`로 초기화되었으면 0이 아닌 값을 반환합니다.

int **PyThread_tss_create**(*Py_tss_t* *key)

Part of the Stable ABI *since version 3.7.* TSS 키 초기화에 성공하면 0 값을 반환합니다. *key* 인자가 가리키는 값이 `Py_tss_NEEDS_INIT`로 초기화되지 않으면 동작이 정의되지 않습니다. 이 함수는 같은 키에서 반복적으로 호출될 수 있습니다 – 이미 초기화된 키에 대해 호출하면 아무런 일도 하지 않으며 즉시 성공을 반환합니다.

void **PyThread_tss_delete**(*Py_tss_t* *key)

Part of the Stable ABI *since version 3.7.* TSS 키를 삭제하여 모든 스레드에서 키와 관련된 값을 잊게 하고, 키의 초기화 상태를 초기화되지 않음으로 변경합니다. 파괴된 키는 `PyThread_tss_create()`로 다시 초기화할 수 있습니다. 이 함수는 같은 키에서 반복적으로 호출될 수 있습니다 – 이미 파괴된 키에 대해 호출하면 아무런 일도 하지 않습니다.

int **PyThread_tss_set**(*Py_tss_t* *key, void *value)

Part of the Stable ABI *since version 3.7.* Return a zero value to indicate successfully associating a `void*` value with a TSS key in the current thread. Each thread has a distinct mapping of the key to a `void*` value.

void ***PyThread_tss_get**(*Py_tss_t* *key)

Part of the Stable ABI *since version 3.7.* Return the `void*` value associated with a TSS key in the current thread. This returns `NULL` if no value is associated with the key in the current thread.

9.10.2 스레드 로컬 저장소 (TLS) API

버전 3.7부터 폐지됨: 이 API는 스레드별 저장소 (*TSS*) *API*로 대체됩니다.

> **참고**
>
> 이 버전의 API는 `int`로 안전하게 캐스트 할 수 없는 방식으로 네이티브 TLS 키가 정의된 플랫폼을 지원하지 않습니다. 이러한 플랫폼에서, `PyThread_create_key()`는 실패 상태로 즉시 반환되며, 다른 TLS 함수는 이러한 플랫폼에서 모두 아무런 일도 하지 않습니다.

위에서 언급한 호환성 문제로 인해, 이 버전의 API를 새 코드에서 사용해서는 안 됩니다.

int **PyThread_create_key**()

Part of the Stable ABI.

void **PyThread_delete_key**(int key)

Part of the Stable ABI.

int **PyThread_set_key_value**(int key, void *value)

Part of the Stable ABI.

void ***PyThread_get_key_value**(int key)

Part of the Stable ABI.

void **PyThread_delete_key_value**(int key)

Part of the Stable ABI.

void **PyThread_ReInitTLS**()

Part of the Stable ABI.

CHAPTER 10

파이썬 초기화 구성

Added in version 3.8.

Python can be initialized with *Py_InitializeFromConfig()* and the *PyConfig* structure. It can be preinitialized with *Py_PreInitialize()* and the *PyPreConfig* structure.

There are two kinds of configuration:

- The *Python Configuration* can be used to build a customized Python which behaves as the regular Python. For example, environment variables and command line arguments are used to configure Python.
- The *Isolated Configuration* can be used to embed Python into an application. It isolates Python from the system. For example, environment variables are ignored, the LC_CTYPE locale is left unchanged and no signal handler is registered.

The *Py_RunMain()* function can be used to write a customized Python program.

초기화, 파이널리제이션 및 스레드도 참조하십시오.

더 보기

PEP 587 "파이썬 초기화 구성".

10.1 Example

항상 격리 모드에서 실행되는 사용자 정의 파이썬의 예:

```
int main(int argc, char **argv)
{
    PyStatus status;

    PyConfig config;
    PyConfig_InitPythonConfig(&config);
    config.isolated = 1;

    /* Decode command line arguments.
       Implicitly preinitialize Python (in isolated mode). */
    status = PyConfig_SetBytesArgv(&config, argc, argv);
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    if (PyStatus_Exception(status)) {
        goto exception;
    }

    status = Py_InitializeFromConfig(&config);
    if (PyStatus_Exception(status)) {
        goto exception;
    }
    PyConfig_Clear(&config);

    return Py_RunMain();

exception:
    PyConfig_Clear(&config);
    if (PyStatus_IsExit(status)) {
        return status.exitcode;
    }
    /* Display the error message and exit the process with
       non-zero exit code */
    Py_ExitStatusException(status);
}
```

10.2 PyWideStringList

type **PyWideStringList**

`wchar_t*` 문자열의 리스트.

*length*가 0이 아니면, *items*는 `NULL`이 아니어야 하고 모든 문자열은 `NULL`이 아니어야 합니다.

메서드:

PyStatus **PyWideStringList_Append**(*PyWideStringList* *list, const wchar_t *item)

*item*을 *list*에 추가합니다.

이 함수를 호출하려면 파이썬을 사전 초기화해야 합니다.

PyStatus **PyWideStringList_Insert**(*PyWideStringList* *list, *Py_ssize_t* index, const wchar_t *item)

*item*을 *list*의 *index*에 삽입합니다.

*index*가 *list* 길이보다 크거나 같으면, *item*을 *list*에 추가(append)합니다.

index must be greater than or equal to `0`.

이 함수를 호출하려면 파이썬을 사전 초기화해야 합니다.

구조체 필드:

Py_ssize_t **length**

리스트 길이.

wchar_t ***items**

리스트 항목들.

10.3 PyStatus

type **PyStatus**

초기화 함수 상태를 저장하는 구조체: 성공, 에러 또는 종료.

에러의 경우, 에러를 만든 C 함수 이름을 저장할 수 있습니다.

구조체 필드:

int **exitcode**
종료 코드. `exit()` 에 전달된 인자.

const char ***err_msg**
에러 메시지.

const char ***func**
에러를 만든 함수의 이름, `NULL` 일 수 있습니다.

상태를 만드는 함수:

PyStatus **PyStatus_Ok**(void)
성공.

PyStatus **PyStatus_Error**(const char *err_msg)
메시지가 포함된 초기화 에러.

err_msg must not be `NULL`.

PyStatus **PyStatus_NoMemory**(void)
메모리 할당 실패 (메모리 부족).

PyStatus **PyStatus_Exit**(int exitcode)
지정된 종료 코드로 파이썬을 종료합니다.

상태를 처리하는 함수:

int **PyStatus_Exception**(*PyStatus* status)
상태가 에러입니까? 아니면 종료입니까? 참이면, 예외를 처리해야 합니다; 예를 들어 *Py_ExitStatusException()* 을 호출하여.

int **PyStatus_IsError**(*PyStatus* status)
결과가 에러입니까?

int **PyStatus_IsExit**(*PyStatus* status)
결과가 종료입니까?

void **Py_ExitStatusException**(*PyStatus* status)
status 가 종료이면 `exit(exitcode)` 를 호출합니다. *status* 가 에러이면 에러 메시지를 인쇄하고 0이 아닌 종료 코드로 종료합니다. `PyStatus_Exception(status)` 가 0이 아닐 때만 호출해야 합니다.

참고

내부적으로, 파이썬은 `PyStatus.func` 를 설정하는 데는 매크로를 사용하는 반면, `func` 가 `NULL` 로 설정된 상태를 만드는 데는 함수를 사용합니다.

예:

```
PyStatus alloc(void **ptr, size_t size)
{
    *ptr = PyMem_RawMalloc(size);
    if (*ptr == NULL) {
        return PyStatus_NoMemory();
    }
    return PyStatus_Ok();
}

int main(int argc, char **argv)
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
{
    void *ptr;
    PyStatus status = alloc(&ptr, 16);
    if (PyStatus_Exception(status)) {
        Py_ExitStatusException(status);
    }
    PyMem_Free(ptr);
    return 0;
}
```

10.4 PyPreConfig

type **PyPreConfig**

Structure used to preinitialize Python.

사전 구성을 초기화하는 함수:

void **PyPreConfig_InitPythonConfig**(*PyPreConfig* *preconfig)

파이썬 구성으로 사전 구성을 초기화합니다.

void **PyPreConfig_InitIsolatedConfig**(*PyPreConfig* *preconfig)

격리된 구성으로 사전 구성을 초기화합니다.

구조체 필드:

int **allocator**

Name of the Python memory allocators:

- `PYMEM_ALLOCATOR_NOT_SET` (`0`): don' t change memory allocators (use defaults).
- `PYMEM_ALLOCATOR_DEFAULT` (`1`): *default memory allocators*.
- `PYMEM_ALLOCATOR_DEBUG` (`2`): *default memory allocators* with *debug hooks*.
- `PYMEM_ALLOCATOR_MALLOC` (`3`): use `malloc()` of the C library.
- `PYMEM_ALLOCATOR_MALLOC_DEBUG` (`4`): force usage of `malloc()` with *debug hooks*.
- `PYMEM_ALLOCATOR_PYMALLOC` (`5`): *Python pymalloc memory allocator*.
- `PYMEM_ALLOCATOR_PYMALLOC_DEBUG` (`6`): *Python pymalloc memory allocator* with *debug hooks*.

`PYMEM_ALLOCATOR_PYMALLOC` and `PYMEM_ALLOCATOR_PYMALLOC_DEBUG` are not supported if Python is `configured using --without-pymalloc`.

메모리 관리를 참조하십시오.

Default: `PYMEM_ALLOCATOR_NOT_SET`.

int **configure_locale**

Set the LC_CTYPE locale to the user preferred locale.

If equals to `0`, set *`coerce_c_locale`* and *`coerce_c_locale_warn`* members to `0`.

See the *locale encoding*.

Default: `1` in Python config, `0` in isolated config.

int **coerce_c_locale**

If equals to `2`, coerce the C locale.

If equals to `1`, read the LC_CTYPE locale to decide if it should be coerced.

See the *locale encoding*.

Default: `-1` in Python config, `0` in isolated config.

int **coerce_c_locale_warn**

0이 아니면, C 로케일이 강제될 때 경고가 발생합니다.

Default: `-1` in Python config, `0` in isolated config.

int **dev_mode**

Python Development Mode: see *PyConfig.dev_mode*.

Default: `-1` in Python mode, `0` in isolated mode.

int **isolated**

Isolated mode: see *PyConfig.isolated*.

Default: `0` in Python mode, `1` in isolated mode.

int **legacy_windows_fs_encoding**

If non-zero:

- Set *PyPreConfig.utf8_mode* to `0`,
- Set *PyConfig.filesystem_encoding* to `"mbcs"`,
- Set *PyConfig.filesystem_errors* to `"replace"`.

Initialized from the `PYTHONLEGACYWINDOWSFSENCODING` environment variable value.

윈도우에서만 사용 가능합니다. `#ifdef MS_WINDOWS` 매크로는 윈도우 특정 코드에 사용할 수 있습니다.

Default: `0`.

int **parse_argv**

0이 아니면, *Py_PreInitializeFromArgs()*와 *Py_PreInitializeFromBytesArgs()*는 일반 파이썬이 명령 줄 인자를 구문 분석하는 것과 같은 방식으로 `argv` 인자를 구문 분석합니다. 명령 줄 인자를 참조하십시오.

Default: `1` in Python config, `0` in isolated config.

int **use_environment**

Use environment variables? See *PyConfig.use_environment*.

Default: `1` in Python config and `0` in isolated config.

int **utf8_mode**

If non-zero, enable the Python UTF-8 Mode.

Set to `0` or `1` by the `-X utf8` command line option and the `PYTHONUTF8` environment variable.

Also set to `1` if the `LC_CTYPE` locale is `C` or `POSIX`.

Default: `-1` in Python config and `0` in isolated config.

10.5 Preinitialize Python with PyPreConfig

The preinitialization of Python:

- Set the Python memory allocators (*PyPreConfig.allocator*)
- Configure the LC_CTYPE locale (*locale encoding*)
- Set the Python UTF-8 Mode (*PyPreConfig.utf8_mode*)

The current preconfiguration (`PyPreConfig` type) is stored in `_PyRuntime.preconfig`.

파이썬을 사전 초기화하는 함수:

PyStatus **Py_PreInitialize**(const *PyPreConfig* *preconfig)

preconfig 사전 구성에서 파이썬을 사전 초기화합니다.

preconfig must not be `NULL`.

PyStatus **Py_PreInitializeFromBytesArgs**(const *PyPreConfig* *preconfig, int argc, char *const *argv)

preconfig 사전 구성에서 파이썬을 사전 초기화합니다.

Parse *argv* command line arguments (bytes strings) if *parse_argv* of *preconfig* is non-zero.

preconfig must not be `NULL`.

PyStatus **Py_PreInitializeFromArgs**(const *PyPreConfig* *preconfig, int argc, wchar_t *const *argv)

preconfig 사전 구성에서 파이썬을 사전 초기화합니다.

Parse *argv* command line arguments (wide strings) if *parse_argv* of *preconfig* is non-zero.

preconfig must not be `NULL`.

호출자는 *PyStatus_Exception()*과 *Py_ExitStatusException()* 을 사용하여 예외(에러나 종료)를 처리해야 합니다.

For *Python Configuration* (*PyPreConfig_InitPythonConfig()*), if Python is initialized with command line arguments, the command line arguments must also be passed to preinitialize Python, since they have an effect on the pre-configuration like encodings. For example, the `-X utf8` command line option enables the Python UTF-8 Mode.

`PyMem_SetAllocator()` 는 *Py_PreInitialize()* 이후에 *Py_InitializeFromConfig()* 이전에 호출하여 사용자 정의 메모리 할당자를 설치할 수 있습니다. *PyPreConfig.allocator*가 `PYMEM_ALLOCATOR_NOT_SET`으로 설정되면 *Py_PreInitialize()* 전에 호출할 수 있습니다.

Python memory allocation functions like *PyMem_RawMalloc()* must not be used before the Python preinitialization, whereas calling directly `malloc()` and `free()` is always safe. *Py_DecodeLocale()* must not be called before the Python preinitialization.

Example using the preinitialization to enable the Python UTF-8 Mode:

```
PyStatus status;
PyPreConfig preconfig;
PyPreConfig_InitPythonConfig(&preconfig);

preconfig.utf8_mode = 1;

status = Py_PreInitialize(&preconfig);
if (PyStatus_Exception(status)) {
    Py_ExitStatusException(status);
}

/* at this point, Python speaks UTF-8 */

Py_Initialize();
/* ... use Python API here ... */
Py_Finalize();
```

10.6 PyConfig

type **PyConfig**

파이썬을 구성하기 위한 대부분의 파라미터를 포함하는 구조체.

When done, the *PyConfig_Clear()* function must be used to release the configuration memory.

구조체 메서드:

void **PyConfig_InitPythonConfig**(*PyConfig* *config)

Initialize configuration with the *Python Configuration*.

void **PyConfig_InitIsolatedConfig**(*PyConfig* *config)

Initialize configuration with the *Isolated Configuration*.

PyStatus **PyConfig_SetString**(*PyConfig* *config, wchar_t *const *config_str, const wchar_t *str)

와이드 문자열 *str*을 `*config_str`로 복사합니다.

Preinitialize Python if needed.

PyStatus **PyConfig_SetBytesString**(*PyConfig* *config, wchar_t *const *config_str, const char *str)

Decode *str* using `Py_DecodeLocale()` and set the result into `*config_str`.

Preinitialize Python if needed.

PyStatus **PyConfig_SetArgv**(*PyConfig* *config, int argc, wchar_t *const *argv)

Set command line arguments (`argv` member of *config*) from the *argv* list of wide character strings.

Preinitialize Python if needed.

PyStatus **PyConfig_SetBytesArgv**(*PyConfig* *config, int argc, char *const *argv)

Set command line arguments (`argv` member of *config*) from the *argv* list of bytes strings. Decode bytes using `Py_DecodeLocale()`.

Preinitialize Python if needed.

PyStatus **PyConfig_SetWideStringList**(*PyConfig* *config, *PyWideStringList* *list, *Py_ssize_t* length, wchar_t **items)

와이드 문자열 리스트 *list*를 *length*와 *items*로 설정합니다.

Preinitialize Python if needed.

PyStatus **PyConfig_Read**(*PyConfig* *config)

모든 파이썬 구성을 읽습니다.

이미 초기화된 필드는 변경되지 않습니다.

Fields for *path configuration* are no longer calculated or modified when calling this function, as of Python 3.11.

The `PyConfig_Read()` function only parses `PyConfig.argv` arguments once: `PyConfig.parse_argv` is set to `2` after arguments are parsed. Since Python arguments are stripped from `PyConfig.argv`, parsing arguments twice would parse the application options as Python options.

Preinitialize Python if needed.

버전 3.10에서 변경: The `PyConfig.argv` arguments are now only parsed once, `PyConfig.parse_argv` is set to `2` after arguments are parsed, and arguments are only parsed if `PyConfig.parse_argv` equals `1`.

버전 3.11에서 변경: `PyConfig_Read()` no longer calculates all paths, and so fields listed under *Python Path Configuration* may no longer be updated until `Py_InitializeFromConfig()` is called.

void **PyConfig_Clear**(*PyConfig* *config)

구성 메모리를 해제합니다.

Most `PyConfig` methods *preinitialize Python* if needed. In that case, the Python preinitialization configuration (`PyPreConfig`) in based on the `PyConfig`. If configuration fields which are in common with `PyPreConfig` are tuned, they must be set before calling a `PyConfig` method:

- `PyConfig.dev_mode`
- `PyConfig.isolated`
- `PyConfig.parse_argv`
- `PyConfig.use_environment`

Moreover, if *PyConfig_SetArgv()* or *PyConfig_SetBytesArgv()* is used, this method must be called before other methods, since the preinitialization configuration depends on command line arguments (if *parse_argv* is non-zero).

이 메서드의 호출자는 PyStatus_Exception() 과 Py_ExitStatusException() 을 사용하여 예외(에러나 종료)를 처리해야 합니다.

구조체 필드:

PyWideStringList **argv**

Command line arguments: sys.argv.

Set *parse_argv* to 1 to parse *argv* the same way the regular Python parses Python command line arguments and then to strip Python arguments from *argv*.

If *argv* is empty, an empty string is added to ensure that sys.argv always exists and is never empty.

Default: NULL.

See also the *orig_argv* member.

int **safe_path**

If equals to zero, Py_RunMain() prepends a potentially unsafe path to sys.path at startup:

- If *argv[0]* is equal to L"-m" (python -m module), prepend the current working directory.
- If running a script (python script.py), prepend the script' s directory. If it' s a symbolic link, resolve symbolic links.
- Otherwise (python -c code and python), prepend an empty string, which means the current working directory.

Set to 1 by the -P command line option and the PYTHONSAFEPATH environment variable.

Default: 0 in Python config, 1 in isolated config.

Added in version 3.11.

wchar_t ***base_exec_prefix**

sys.base_exec_prefix.

Default: NULL.

Part of the *Python Path Configuration* output.

wchar_t ***base_executable**

Python base executable: sys._base_executable.

Set by the __PYVENV_LAUNCHER__ environment variable.

Set from *PyConfig.executable* if NULL.

Default: NULL.

Part of the *Python Path Configuration* output.

wchar_t ***base_prefix**

sys.base_prefix.

Default: NULL.

Part of the *Python Path Configuration* output.

int **buffered_stdio**

If equals to 0 and *configure_c_stdio* is non-zero, disable buffering on the C streams stdout and stderr.

Set to 0 by the -u command line option and the PYTHONUNBUFFERED environment variable.

stdin은 항상 버퍼링 모드로 열립니다.

Default: 1.

int **bytes_warning**

If equals to 1, issue a warning when comparing bytes or bytearray with str, or comparing bytes with int.

If equal or greater to 2, raise a BytesWarning exception in these cases.

Incremented by the -b command line option.

Default: 0.

int **warn_default_encoding**

If non-zero, emit a EncodingWarning warning when io.TextIOWrapper uses its default encoding. See io-encoding-warning for details.

Default: 0.

Added in version 3.10.

int **code_debug_ranges**

If equals to 0, disables the inclusion of the end line and column mappings in code objects. Also disables traceback printing carets to specific error locations.

Set to 0 by the PYTHONNODEBUGRANGES environment variable and by the -X no_debug_ranges command line option.

Default: 1.

Added in version 3.11.

wchar_t ***check_hash_pycs_mode**

Control the validation behavior of hash-based .pyc files: value of the --check-hash-based-pycs command line option.

Valid values:

- L"always": Hash the source file for invalidation regardless of value of the 'check_source' flag.
- L"never": Assume that hash-based pycs always are valid.
- L"default": The 'check_source' flag in hash-based pycs determines invalidation.

Default: L"default".

See also **PEP 552** "Deterministic pycs".

int **configure_c_stdio**

If non-zero, configure C standard streams:

- On Windows, set the binary mode (O_BINARY) on stdin, stdout and stderr.
- If *buffered_stdio* equals zero, disable buffering of stdin, stdout and stderr streams.
- If *interactive* is non-zero, enable stream buffering on stdin and stdout (only stdout on Windows).

Default: 1 in Python config, 0 in isolated config.

int **dev_mode**

0이 아니면, 파이썬 개발 모드를 활성화합니다.

Set to 1 by the -X dev option and the PYTHONDEVMODE environment variable.

Default: -1 in Python mode, 0 in isolated mode.

int **dump_refs**

Dump Python references?

0이 아니면, 종료 시 여전히 활성 상태인 모든 객체를 덤프합니다.

Set to 1 by the PYTHONDUMPREFS environment variable.

Need a special build of Python with the `Py_TRACE_REFS` macro defined: see the `configure --with-trace-refs option`.

Default: `0`.

wchar_t ***exec_prefix**

The site-specific directory prefix where the platform-dependent Python files are installed: `sys.exec_prefix`.

Default: `NULL`.

Part of the *Python Path Configuration* output.

wchar_t ***executable**

The absolute path of the executable binary for the Python interpreter: `sys.executable`.

Default: `NULL`.

Part of the *Python Path Configuration* output.

int **faulthandler**

Enable faulthandler?

0이 아니면, 시작 시 `faulthandler.enable()`을 호출합니다.

Set to `1` by `-X faulthandler` and the `PYTHONFAULTHANDLER` environment variable.

Default: `-1` in Python mode, `0` in isolated mode.

wchar_t ***filesystem_encoding**

Filesystem encoding: `sys.getfilesystemencoding()`.

On macOS, Android and VxWorks: use `"utf-8"` by default.

On Windows: use `"utf-8"` by default, or `"mbcs"` if *legacy_windows_fs_encoding* of *PyPreConfig* is non-zero.

Default encoding on other platforms:

- `"utf-8"` if *PyPreConfig.utf8_mode* is non-zero.
- `"ascii"` if Python detects that `nl_langinfo(CODESET)` announces the ASCII encoding, whereas the `mbstowcs()` function decodes from a different encoding (usually Latin1).
- `"utf-8"` if `nl_langinfo(CODESET)` returns an empty string.
- Otherwise, use the *locale encoding*: `nl_langinfo(CODESET)` result.

At Python startup, the encoding name is normalized to the Python codec name. For example, `"ANSI_X3.4-1968"` is replaced with `"ascii"`.

See also the *filesystem_errors* member.

wchar_t ***filesystem_errors**

Filesystem error handler: `sys.getfilesystemencodeerrors()`.

On Windows: use `"surrogatepass"` by default, or `"replace"` if *legacy_windows_fs_encoding* of *PyPreConfig* is non-zero.

On other platforms: use `"surrogateescape"` by default.

Supported error handlers:

- `"strict"`
- `"surrogateescape"`
- `"surrogatepass"` (only supported with the UTF-8 encoding)

See also the *filesystem_encoding* member.

unsigned long **hash_seed**

int **use_hash_seed**
무작위 해시 함수 시드.

If *use_hash_seed* is zero, a seed is chosen randomly at Python startup, and *hash_seed* is ignored.

Set by the PYTHONHASHSEED environment variable.

Default *use_hash_seed* value: -1 in Python mode, 0 in isolated mode.

wchar_t ***home**
파이썬 홈 디렉터리.

If *Py_SetPythonHome()* has been called, use its argument if it is not NULL.

Set by the PYTHONHOME environment variable.

Default: NULL.

Part of the *Python Path Configuration* input.

int **import_time**
0이 아니면, 임포트 시간을 프로파일 합니다.

Set the 1 by the -X importtime option and the PYTHONPROFILEIMPORTTIME environment variable.

Default: 0.

int **inspect**
스크립트나 명령을 실행한 후 대화식 모드로 들어갑니다.

If greater than 0, enable inspect: when a script is passed as first argument or the -c option is used, enter interactive mode after executing the script or the command, even when sys.stdin does not appear to be a terminal.

Incremented by the -i command line option. Set to 1 if the PYTHONINSPECT environment variable is non-empty.

Default: 0.

int **install_signal_handlers**
Install Python signal handlers?

Default: 1 in Python mode, 0 in isolated mode.

int **interactive**
If greater than 0, enable the interactive mode (REPL).

Incremented by the -i command line option.

Default: 0.

int **int_max_str_digits**
Configures the integer string conversion length limitation. An initial value of -1 means the value will be taken from the command line or environment or otherwise default to 4300 (sys.int_info.default_max_str_digits). A value of 0 disables the limitation. Values greater than zero but less than 640 (sys.int_info.str_digits_check_threshold) are unsupported and will produce an error.

Configured by the -X int_max_str_digits command line flag or the PYTHONINTMAXSTRDIGITS environment variable.

Default: -1 in Python mode. 4300 (sys.int_info.default_max_str_digits) in isolated mode.

Added in version 3.12.

int **isolated**

If greater than 0, enable isolated mode:

- Set *safe_path* to 1: don' t prepend a potentially unsafe path to sys.path at Python startup, such as the current directory, the script' s directory or an empty string.
- Set *use_environment* to 0: ignore PYTHON environment variables.
- Set *user_site_directory* to 0: don' t add the user site directory to sys.path.
- 파이썬 REPL은 대화식 프롬프트에서 readline을 임포트 하지도 기본 readline 구성을 활성화하지도 않습니다.

Set to 1 by the -I command line option.

Default: 0 in Python mode, 1 in isolated mode.

See also the *Isolated Configuration* and *PyPreConfig.isolated*.

int **legacy_windows_stdio**

If non-zero, use io.FileIO instead of io._WindowsConsoleIO for sys.stdin, sys.stdout and sys.stderr.

Set to 1 if the PYTHONLEGACYWINDOWSSTDIO environment variable is set to a non-empty string.

윈도우에서만 사용 가능합니다. #ifdef MS_WINDOWS 매크로는 윈도우 특정 코드에 사용할 수 있습니다.

Default: 0.

See also the **PEP 528** (Change Windows console encoding to UTF-8).

int **malloc_stats**

0이 아니면, 종료 시 파이썬 *pymalloc* 메모리 할당자에 대한 통계를 덤프합니다.

Set to 1 by the PYTHONMALLOCSTATS environment variable.

The option is ignored if Python is configured using the --without-pymalloc option.

Default: 0.

wchar_t ***platlibdir**

Platform library directory name: sys.platlibdir.

Set by the PYTHONPLATLIBDIR environment variable.

Default: value of the PLATLIBDIR macro which is set by the configure --with-platlibdir option (default: "lib", or "DLLs" on Windows).

Part of the *Python Path Configuration* input.

Added in version 3.9.

버전 3.11에서 변경: This macro is now used on Windows to locate the standard library extension modules, typically under DLLs. However, for compatibility, note that this value is ignored for any non-standard layouts, including in-tree builds and virtual environments.

wchar_t ***pythonpath_env**

Module search paths (sys.path) as a string separated by DELIM (os.pathsep).

Set by the PYTHONPATH environment variable.

Default: NULL.

Part of the *Python Path Configuration* input.

PyWideStringList **module_search_paths**

int **module_search_paths_set**

Module search paths: `sys.path`.

If *module_search_paths_set* is equal to `0`, *Py_InitializeFromConfig()* will replace *module_search_paths* and sets *module_search_paths_set* to `1`.

Default: empty list (`module_search_paths`) and `0` (`module_search_paths_set`).

Part of the *Python Path Configuration* output.

int **optimization_level**

컴파일 최적화 수준:

- `0`: Peephole optimizer, set `__debug__` to `True`.
- `1`: Level 0, remove assertions, set `__debug__` to `False`.
- `2`: Level 1, strip docstrings.

Incremented by the `-O` command line option. Set to the `PYTHONOPTIMIZE` environment variable value.

Default: `0`.

PyWideStringList **orig_argv**

The list of the original command line arguments passed to the Python executable: `sys.orig_argv`.

If *orig_argv* list is empty and *argv* is not a list only containing an empty string, *PyConfig_Read()* copies *argv* into *orig_argv* before modifying *argv* (if *parse_argv* is non-zero).

See also the *argv* member and the *Py_GetArgcArgv()* function.

Default: empty list.

Added in version 3.10.

int **parse_argv**

Parse command line arguments?

If equals to `1`, parse *argv* the same way the regular Python parses command line arguments, and strip Python arguments from *argv*.

The *PyConfig_Read()* function only parses *PyConfig.argv* arguments once: *PyConfig.parse_argv* is set to `2` after arguments are parsed. Since Python arguments are stripped from *PyConfig.argv*, parsing arguments twice would parse the application options as Python options.

Default: `1` in Python mode, `0` in isolated mode.

버전 3.10에서 변경: The *PyConfig.argv* arguments are now only parsed if *PyConfig.parse_argv* equals to `1`.

int **parser_debug**

Parser debug mode. If greater than `0`, turn on parser debugging output (for expert only, depending on compilation options).

Incremented by the `-d` command line option. Set to the `PYTHONDEBUG` environment variable value.

Need a debug build of Python (the `Py_DEBUG` macro must be defined).

Default: `0`.

int **pathconfig_warnings**

If non-zero, calculation of path configuration is allowed to log warnings into `stderr`. If equals to `0`, suppress these warnings.

Default: `1` in Python mode, `0` in isolated mode.

Part of the *Python Path Configuration* input.

버전 3.11에서 변경: Now also applies on Windows.

wchar_t ***prefix**

The site-specific directory prefix where the platform independent Python files are installed: `sys.prefix`.

Default: `NULL`.

Part of the *Python Path Configuration* output.

wchar_t ***program_name**

Program name used to initialize *executable* and in early error messages during Python initialization.

- If `Py_SetProgramName()` has been called, use its argument.
- On macOS, use `PYTHONEXECUTABLE` environment variable if set.
- If the `WITH_NEXT_FRAMEWORK` macro is defined, use `__PYVENV_LAUNCHER__` environment variable if set.
- Use `argv[0]` of *argv* if available and non-empty.
- Otherwise, use `L"python"` on Windows, or `L"python3"` on other platforms.

Default: `NULL`.

Part of the *Python Path Configuration* input.

wchar_t ***pycache_prefix**

Directory where cached `.pyc` files are written: `sys.pycache_prefix`.

Set by the `-X pycache_prefix=PATH` command line option and the `PYTHONPYCACHEPREFIX` environment variable.

`NULL`이면, `sys.pycache_prefix`는 `None`으로 설정됩니다.

Default: `NULL`.

int **quiet**

Quiet mode. If greater than `0`, don' t display the copyright and version at Python startup in interactive mode.

Incremented by the `-q` command line option.

Default: `0`.

wchar_t ***run_command**

Value of the `-c` command line option.

Used by *Py_RunMain()*.

Default: `NULL`.

wchar_t ***run_filename**

Filename passed on the command line: trailing command line argument without `-c` or `-m`. It is used by the *Py_RunMain()* function.

For example, it is set to `script.py` by the `python3 script.py arg` command line.

See also the *PyConfig.skip_source_first_line* option.

Default: `NULL`.

wchar_t ***run_module**

Value of the `-m` command line option.

Used by *Py_RunMain()*.

Default: `NULL`.

int **show_ref_count**

Show total reference count at exit (excluding immortal objects)?

Set to `1` by `-X showrefcount` command line option.

Need a debug build of Python (the `Py_REF_DEBUG` macro must be defined).

Default: `0`.

int **site_import**

시작할 때 `site` 모듈을 임포트 합니까?

If equal to zero, disable the import of the module site and the site-dependent manipulations of `sys.path` that it entails.

Also disable these manipulations if the `site` module is explicitly imported later (call `site.main()` if you want them to be triggered).

Set to `0` by the `-S` command line option.

`sys.flags.no_site` is set to the inverted value of *site_import*.

Default: `1`.

int **skip_source_first_line**

If non-zero, skip the first line of the *PyConfig.run_filename* source.

It allows the usage of non-Unix forms of `#!cmd`. This is intended for a DOS specific hack only.

Set to `1` by the `-x` command line option.

Default: `0`.

wchar_t ***stdio_encoding**

wchar_t ***stdio_errors**

Encoding and encoding errors of `sys.stdin`, `sys.stdout` and `sys.stderr` (but `sys.stderr` always uses `"backslashreplace"` error handler).

If *Py_SetStandardStreamEncoding()* has been called, use its *error* and *errors* arguments if they are not `NULL`.

Use the `PYTHONIOENCODING` environment variable if it is non-empty.

Default encoding:

- `"UTF-8"` if *PyPreConfig.utf8_mode* is non-zero.
- Otherwise, use the *locale encoding*.

Default error handler:

- On Windows: use `"surrogateescape"`.
- `"surrogateescape"` if *PyPreConfig.utf8_mode* is non-zero, or if the LC_CTYPE locale is “C” or “POSIX”.
- `"strict"` otherwise.

int **tracemalloc**

Enable tracemalloc?

0이 아니면, 시작 시 `tracemalloc.start()`를 호출합니다.

Set by `-X tracemalloc=N` command line option and by the `PYTHONTRACEMALLOC` environment variable.

Default: `-1` in Python mode, `0` in isolated mode.

int **perf_profiling**

Enable compatibility mode with the perf profiler?

If non-zero, initialize the perf trampoline. See perf_profiling for more information.

Set by `-X perf` command line option and by the `PYTHONPERFSUPPORT` environment variable.

Default: `-1`.

Added in version 3.12.

int **use_environment**

Use environment variables?

If equals to zero, ignore the environment variables.

Set to `0` by the `-E` environment variable.

Default: `1` in Python config and `0` in isolated config.

int **user_site_directory**

If non-zero, add the user site directory to `sys.path`.

Set to `0` by the `-s` and `-I` command line options.

Set to `0` by the `PYTHONNOUSERSITE` environment variable.

Default: `1` in Python mode, `0` in isolated mode.

int **verbose**

Verbose mode. If greater than `0`, print a message each time a module is imported, showing the place (filename or built-in module) from which it is loaded.

If greater than or equal to `2`, print a message for each file that is checked for when searching for a module. Also provides information on module cleanup at exit.

Incremented by the `-v` command line option.

Set by the `PYTHONVERBOSE` environment variable value.

Default: `0`.

PyWideStringList **warnoptions**

Options of the `warnings` module to build warnings filters, lowest to highest priority: `sys.warnoptions`.

`warnings` 모듈은 `sys.warnoptions`를 역순으로 추가합니다: 마지막 *PyConfig.warnoptions* 항목은 가장 먼저 검사되는 `warnings.filters`의 첫 번째 항목이 됩니다 (가장 높은 우선순위).

The `-W` command line options adds its value to *warnoptions*, it can be used multiple times.

The `PYTHONWARNINGS` environment variable can also be used to add warning options. Multiple options can be specified, separated by commas (`,`).

Default: empty list.

int **write_bytecode**

If equal to `0`, Python won' t try to write `.pyc` files on the import of source modules.

Set to `0` by the `-B` command line option and the `PYTHONDONTWRITEBYTECODE` environment variable.

`sys.dont_write_bytecode`는 *write_bytecode*의 반전된 값으로 초기화됩니다.

Default: `1`.

PyWideStringList **xoptions**

Values of the `-X` command line options: `sys._xoptions`.

Default: empty list.

If *parse_argv* is non-zero, *argv* arguments are parsed the same way the regular Python parses command line arguments, and Python arguments are stripped from *argv*.

The *xoptions* options are parsed to set other options: see the `-X` command line option.

버전 3.9에서 변경: `show_alloc_count` 필드가 제거되었습니다.

10.7 PyConfig를 사용한 초기화

파이썬을 초기화하는 함수:

PyStatus **Py_InitializeFromConfig**(const *PyConfig* *config)

config 구성에서 파이썬을 초기화합니다.

호출자는 *PyStatus_Exception()*과 *Py_ExitStatusException()* 을 사용하여 예외(에러나 종료)를 처리해야 합니다.

If *PyImport_FrozenModules()*, *PyImport_AppendInittab()* or *PyImport_ExtendInittab()* are used, they must be set or called after Python preinitialization and before the Python initialization. If Python is initialized multiple times, *PyImport_AppendInittab()* or *PyImport_ExtendInittab()* must be called before each Python initialization.

The current configuration (`PyConfig` type) is stored in `PyInterpreterState.config`.

프로그램 이름을 설정하는 예:

```
void init_python(void)
{
    PyStatus status;

    PyConfig config;
    PyConfig_InitPythonConfig(&config);

    /* Set the program name. Implicitly preinitialize Python. */
    status = PyConfig_SetString(&config, &config.program_name,
                                L"/path/to/my_program");
    if (PyStatus_Exception(status)) {
        goto exception;
    }

    status = Py_InitializeFromConfig(&config);
    if (PyStatus_Exception(status)) {
        goto exception;
    }
    PyConfig_Clear(&config);
    return;

exception:
    PyConfig_Clear(&config);
    Py_ExitStatusException(status);
}
```

More complete example modifying the default configuration, read the configuration, and then override some parameters. Note that since 3.11, many parameters are not calculated until initialization, and so values cannot be read from the configuration structure. Any values set before initialize is called will be left unchanged by initialization:

```
PyStatus init_python(const char *program_name)
{
    PyStatus status;
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    PyConfig config;
    PyConfig_InitPythonConfig(&config);

    /* Set the program name before reading the configuration
       (decode byte string from the locale encoding).

       Implicitly preinitialize Python. */
    status = PyConfig_SetBytesString(&config, &config.program_name,
                                     program_name);
    if (PyStatus_Exception(status)) {
        goto done;
    }

    /* Read all configuration at once */
    status = PyConfig_Read(&config);
    if (PyStatus_Exception(status)) {
        goto done;
    }

    /* Specify sys.path explicitly */
    /* If you want to modify the default set of paths, finish
       initialization first and then use PySys_GetObject("path") */
    config.module_search_paths_set = 1;
    status = PyWideStringList_Append(&config.module_search_paths,
                                     L"/path/to/stdlib");
    if (PyStatus_Exception(status)) {
        goto done;
    }
    status = PyWideStringList_Append(&config.module_search_paths,
                                     L"/path/to/more/modules");
    if (PyStatus_Exception(status)) {
        goto done;
    }

    /* Override executable computed by PyConfig_Read() */
    status = PyConfig_SetString(&config, &config.executable,
                                L"/path/to/my_executable");
    if (PyStatus_Exception(status)) {
        goto done;
    }

    status = Py_InitializeFromConfig(&config);

done:
    PyConfig_Clear(&config);
    return status;
}
```

10.8 격리된 구성

*PyPreConfig_InitIsolatedConfig()*와 *PyConfig_InitIsolatedConfig()* 함수는 시스템에서 파이썬을 격리하는 구성을 만듭니다. 예를 들어, 파이썬을 응용 프로그램에 내장하기 위해.

This configuration ignores global configuration variables, environment variables, command line arguments (*PyConfig.argv* is not parsed) and user site directory. The C standard streams (ex: `stdout`) and the LC_CTYPE locale are left unchanged. Signal handlers are not installed.

Configuration files are still used with this configuration to determine paths that are unspecified. Ensure *PyConfig.home* is specified to avoid computing the default path configuration.

10.9 파이썬 구성

*PyPreConfig_InitPythonConfig()*와 *PyConfig_InitPythonConfig()* 함수는 일반 파이썬처럼 동작하는 사용자 정의된 파이썬을 빌드하기 위한 구성을 만듭니다.

환경 변수와 명령 줄 인자는 파이썬을 구성하는 데 사용되는 반면, 전역 구성 변수는 무시됩니다.

This function enables C locale coercion (**PEP 538**) and Python UTF-8 Mode (**PEP 540**) depending on the LC_CTYPE locale, `PYTHONUTF8` and `PYTHONCOERCECLOCALE` environment variables.

10.10 Python Path Configuration

*PyConfig*에는 경로 구성을 위한 여러 필드가 포함되어 있습니다:

- 경로 구성 입력:
  - *PyConfig.home*
  - *PyConfig.platlibdir*
  - *PyConfig.pathconfig_warnings*
  - *PyConfig.program_name*
  - *PyConfig.pythonpath_env*
  - 현재 작업 디렉터리: 절대 경로를 얻기 위해
  - (*PyConfig.program_name*에서) 프로그램 전체 경로를 얻기 위한 `PATH` 환경 변수
  - `__PYVENV_LAUNCHER__` 환경 변수
  - (윈도우 전용) HKEY_CURRENT_USER 와 HKEY_LOCAL_MACHINE 의 “SoftwarePythonPythonCoreX.YPythonPath” 아래에 있는 레지스트리의 응용 프로그램 경로 (여기서 X.Y는 파이썬 버전입니다).
- 경로 구성 출력 필드:
  - *PyConfig.base_exec_prefix*
  - *PyConfig.base_executable*
  - *PyConfig.base_prefix*
  - *PyConfig.exec_prefix*
  - *PyConfig.executable*
  - *PyConfig.module_search_paths_set*, *PyConfig.module_search_paths*
  - *PyConfig.prefix*

If at least one “output field” is not set, Python calculates the path configuration to fill unset fields. If *module_search_paths_set* is equal to `0`, *module_search_paths* is overridden and *module_search_paths_set* is set to `1`.

It is possible to completely ignore the function calculating the default path configuration by setting explicitly all path configuration output fields listed above. A string is considered as set even if it is non-empty. `module_search_paths` is considered as set if `module_search_paths_set` is set to `1`. In this case, `module_search_paths` will be used without modification.

Set *pathconfig_warnings* to `0` to suppress warnings when calculating the path configuration (Unix only, Windows does not log any warning).

*base_prefix*나 *base_exec_prefix* 필드가 설정되지 않으면, 각각 *prefix*와 *exec_prefix*의 값을 상속합니다.

*Py_RunMain()*과 *Py_Main()*은 sys.path를 수정합니다:

- *run_filename*이 설정되고 __main__.py 스크립트를 포함하는 디렉터리이면, *run_filename*을 sys.path 앞에 추가합니다.
- *isolated*가 0이면:
  - *run_module*이 설정되면, 현재 디렉터리를 sys.path 앞에 추가합니다. 현재 디렉터리를 읽을 수 없으면 아무것도 하지 않습니다.
  - *run_filename*이 설정되면, 파일명의 디렉터리를 sys.path 앞에 추가합니다.
  - 그렇지 않으면, 빈 문자열을 sys.path 앞에 추가합니다.

*site_import*가 0이 아니면, site 모듈이 sys.path를 수정할 수 있습니다. *user_site_directory*가 0이 아니고 사용자의 site-package 디렉터리가 존재하면, site 모듈은 사용자의 site-package 디렉터리를 sys.path에 추가합니다.

다음과 같은 구성 파일이 경로 구성에 사용됩니다:

- pyvenv.cfg
- ._pth file (ex: python._pth)
- pybuilddir.txt (유닉스 전용)

If a ._pth file is present:

- Set *isolated* to 1.
- Set *use_environment* to 0.
- Set *site_import* to 0.
- Set *safe_path* to 1.

__PYVENV_LAUNCHER__ 환경 변수는 *PyConfig.base_executable*을 설정하는 데 사용됩니다

10.11 Py_RunMain()

int **Py_RunMain**(void)

명령 줄이나 구성에서 지정된 명령 (*PyConfig.run_command*), 스크립트 (*PyConfig.run_filename*) 또는 모듈 (*PyConfig.run_module*)을 실행합니다.

기본적으로, 그리고 -i 옵션을 사용할 때, REPL을 실행합니다.

마지막으로, 파이썬을 파이널라이즈 하고 exit() 함수에 전달할 수 있는 종료 상태를 반환합니다.

*Py_RunMain()*을 사용하여 항상 격리 모드에서 실행되는 사용자 정의 파이썬의 예는 파이썬 구성을 참조하십시오.

10.12 Py_GetArgcArgv()

void **Py_GetArgcArgv**(int *argc, wchar_t ***argv)

파이썬이 수정하기 전의, 원래 명령 줄 인자를 가져옵니다.

See also *PyConfig.orig_argv* member.

10.13 다단계 초기화 비공개 잠정적 API

This section is a private provisional API introducing multi-phase initialization, the core feature of **PEP 432**:

- "핵심(Core)" 초기화 단계, "최소한의 파이썬":

  - 내장형;
  - 내장 예외;
  - 내장과 프로즌 모듈(frozen modules);
  - `sys` 모듈은 부분적으로만 초기화됩니다 (예를 들어: `sys.path`는 아직 존재하지 않습니다).
- "주(Main)" 초기화 단계, 파이썬이 완전히 초기화됩니다:
  - `importlib`를 설치하고 구성합니다;
  - 경로 구성을 적용합니다;
  - 시그널 처리기를 설치합니다;
  - `sys` 모듈 초기화를 완료합니다 (예를 들어: `sys.stdout`과 `sys.path`를 만듭니다);
  - `faulthandler`와 `tracemalloc`과 같은 선택적 기능을 활성화합니다;
  - `site` 모듈을 임포트 합니다;
  - 등등

비공개 잠정적 API:

- `PyConfig._init_main`: if set to `0`, *`Py_InitializeFromConfig()`* stops at the "Core" initialization phase.

PyStatus **_Py_InitializeMain**(void)

"주" 초기화 단계로 이동하여, 파이썬 초기화를 완료합니다.

"핵심" 단계에서는 아무런 모듈도 임포트 하지 않고 `importlib` 모듈이 구성되지 않습니다: 경로 구성은 "주" 단계에서만 적용됩니다. 경로 구성을 재정의하거나 조정하기 위해 파이썬에서 파이썬을 사용자 정의할 수 있으며, 사용자 정의 `sys.meta_path` 임포터(importer)나 임포트 훅 등을 설치할 수 있습니다.

It may become possible to calculate the *Path Configuration* in Python, after the Core phase and before the Main phase, which is one of the **PEP 432** motivation.

"핵심" 단계가 제대로 정의되지 않았습니다: 이 단계에서 무엇을 사용할 수 있고, 무엇이 그렇지 않아야 하는지는 아직 지정되지 않았습니다. API는 비공개이자 잠정적인 것으로 표시됩니다: 적절한 공개 API가 설계될 때까지 언제든지 API를 수정하거나 제거할 수 있습니다.

"핵심"과 "주" 초기화 단계 사이에서 파이썬 코드를 실행하는 예제:

```
void init_python(void)
{
    PyStatus status;

    PyConfig config;
    PyConfig_InitPythonConfig(&config);
    config._init_main = 0;

    /* ... customize 'config' configuration ... */

    status = Py_InitializeFromConfig(&config);
    PyConfig_Clear(&config);
    if (PyStatus_Exception(status)) {
        Py_ExitStatusException(status);
    }

    /* Use sys.stderr because sys.stdout is only created
       by _Py_InitializeMain() */
    int res = PyRun_SimpleString(
        "import sys; "
        "print('Run Python code before _Py_InitializeMain', "
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
                "file=sys.stderr)");
    if (res < 0) {
        exit(1);
    }

    /* ... put more configuration code here ... */

    status = _Py_InitializeMain();
    if (PyStatus_Exception(status)) {
        Py_ExitStatusException(status);
    }
}
```

CHAPTER 11

메모리 관리

11.1 개요

파이썬의 메모리 관리에는 모든 파이썬 객체와 데이터 구조를 포함하는 비공개 힙(private heap)을 수반합니다. 이 비공개 힙의 관리는 파이썬 메모리 관리자에 의해 내부적으로 이루어집니다. 파이썬 메모리 관리자는 공유, 세그먼트 화, 사전 할당 또는 캐싱과 같은 동적 스토리지 관리의 다양한 측면을 처리하는 서로 다른 구성 요소를 가지고 있습니다.

가장 낮은 수준에서, 원시 메모리 할당자는 운영 체제의 메모리 관리자와 상호 작용하여 비공개 힙에 모든 파이썬 관련 데이터를 저장하기에 충분한 공간이 있는지 확인합니다. 원시 메모리 할당자 위에, 여러 개의 객체별 할당자가 같은 힙에서 작동하며 각 객체 형의 특성에 맞는 고유한 메모리 관리 정책을 구현합니다. 예를 들어, 정수는 다른 스토리지 요구 사항과 속도/공간 절충을 의미하므로, 정수 객체는 힙 내에서 문자열, 튜플 또는 딕셔너리와는 다르게 관리됩니다. 따라서 파이썬 메모리 관리자는 일부 작업을 객체별 할당자에게 위임하지만, 후자가 비공개 힙의 경계 내에서 작동하도록 합니다.

파이썬 힙의 관리는 인터프리터 자체에 의해 수행되며, 사용자는 힙 내부의 메모리 블록에 대한 객체 포인터를 규칙적으로 조작하더라도, 사용자가 제어할 수 없다는 것을 이해하는 것이 중요합니다. 파이썬 객체와 기타 내부 버퍼를 위한 힙 공간 할당은 이 설명서에 나열된 파이썬/C API 함수를 통해 파이썬 메모리 관리자의 요청에 따라 수행됩니다.

메모리 손상을 피하고자, 확장 작성자는 C 라이브러리에서 내보낸 함수를 파이썬 객체에 대해 실행하지 않아야 합니다: `malloc()`, `calloc()`, `realloc()` 및 `free()`. 그렇게 한다면, 서로 다른 알고리즘을 구현하고 다른 힘에 작동하기 때문에, C 할당자와 파이썬 메모리 관리자 간에 혼합 호출이 발생하여 치명적인 결과를 초래합니다. 그러나, 다음 예제와 같이 개별 목적으로 C 라이브러리 할당자를 사용하여 메모리 블록을 안전하게 할당하고 해제할 수 있습니다:

```
PyObject *res;
char *buf = (char *) malloc(BUFSIZ); /* for I/O */

if (buf == NULL)
    return PyErr_NoMemory();
...Do some I/O operation involving buf...
res = PyBytes_FromString(buf);
free(buf); /* malloc'ed */
return res;
```

이 예에서, I/O 버퍼에 대한 메모리 요청은 C 라이브러리 할당자에 의해 처리됩니다. 파이썬 메모리 관리자는 결과로 반환되는 바이트열 객체의 할당에만 관여합니다.

In most situations, however, it is recommended to allocate memory from the Python heap specifically because the latter is under control of the Python memory manager. For example, this is required when the interpreter is extended with new object types written in C. Another reason for using the Python heap is the desire to *inform* the Python memory manager about the memory needs of the extension module. Even when the requested memory is used exclusively for internal, highly specific purposes, delegating all memory requests to the Python memory manager causes the interpreter to have a more accurate image of its memory footprint as a whole. Consequently, under certain circumstances, the Python memory manager may or may not trigger appropriate actions, like garbage collection, memory compaction or other preventive procedures. Note that by using the C library allocator as shown in the previous example, the allocated memory for the I/O buffer escapes completely the Python memory manager.

더 보기

`PYTHONMALLOC` 환경 변수를 사용하여 파이썬에서 사용하는 메모리 할당자를 구성할 수 있습니다.

`PYTHONMALLOCSTATS` 환경 변수는 새로운 pymalloc 객체 아레나(arena)가 만들어질 때마다 그리고 종료 시 *pymalloc* 메모리 할당자의 통계를 인쇄하는 데 사용될 수 있습니다.

11.2 Allocator Domains

All allocating functions belong to one of three different "domains" (see also `PyMemAllocatorDomain`). These domains represent different allocation strategies and are optimized for different purposes. The specific details on how every domain allocates memory or what internal functions each domain calls is considered an implementation detail, but for debugging purposes a simplified table can be found at *here*. There is no hard requirement to use the memory returned by the allocation functions belonging to a given domain for only the purposes hinted by that domain (although this is the recommended practice). For example, one could use the memory returned by `PyMem_RawMalloc()` for allocating Python objects or the memory returned by `PyObject_Malloc()` for allocating memory for buffers.

The three allocation domains are:

- Raw domain: intended for allocating memory for general-purpose memory buffers where the allocation *must* go to the system allocator or where the allocator can operate without the *GIL*. The memory is requested directly to the system.
- "Mem" domain: intended for allocating memory for Python buffers and general-purpose memory buffers where the allocation must be performed with the *GIL* held. The memory is taken from the Python private heap.
- Object domain: intended for allocating memory belonging to Python objects. The memory is taken from the Python private heap.

When freeing memory previously allocated by the allocating functions belonging to a given domain,the matching specific deallocating functions must be used. For example, `PyMem_Free()` must be used to free memory allocated using `PyMem_Malloc()`.

11.3 원시 메모리 인터페이스

다음 함수 집합은 시스템 할당자에 대한 래퍼입니다. 이러한 함수는 스레드 안전해서, *GIL*을 유지할 필요는 없습니다.

The *default raw memory allocator* uses the following functions: `malloc()`, `calloc()`, `realloc()` and `free()`; call `malloc(1)` (or `calloc(1, 1)`) when requesting zero bytes.

Added in version 3.4.

void ***PyMem_RawMalloc**(size_t n)

Allocates *n* bytes and returns a pointer of type `void*` to the allocated memory, or `NULL` if the request fails.

0바이트를 요청하면 가능하면 `PyMem_RawMalloc(1)`이 대신 호출된 것처럼 가능하면 고유한 `NULL`이 아닌 포인터를 반환합니다. 메모리는 어떤 식으로든 초기화되지 않습니다.

void *PyMem_RawCalloc(size_t nelem, size_t elsize)

Allocates *nelem* elements each whose size in bytes is *elsize* and returns a pointer of type `void*` to the allocated memory, or `NULL` if the request fails. The memory is initialized to zeros.

0개의 요소나 0바이트 크기의 요소를 요청하면 `PyMem_RawCalloc(1, 1)` 이 대신 호출된 것처럼 가능하면 고유한 `NULL` 이 아닌 포인터를 반환합니다.

Added in version 3.5.

void *PyMem_RawRealloc(void *p, size_t n)

*p*가 가리키는 메모리 블록의 크기를 *n* 바이트로 조정합니다. 내용은 이전과 새로운 크기의 최솟값 내에서는 변경되지 않습니다.

*p*가 `NULL` 이면, 호출은 `PyMem_RawMalloc(n)` 과 동등합니다; *n*이 0과 같으면, 메모리 블록의 크기는 조정되지만 해제되지는 않고, 반환된 포인터는 `NULL` 이 아닙니다.

*p*가 `NULL` 이 아닌 한, *PyMem_RawMalloc()*, *PyMem_RawRealloc()* 또는 *PyMem_RawCalloc()* 에 대한 이전 호출에 의해 반환된 것이어야 합니다.

요청이 실패하면, *PyMem_RawRealloc()* 은 `NULL` 을 반환하고 *p*는 이전 메모리 영역에 대한 유효한 포인터로 유지됩니다.

void PyMem_RawFree(void *p)

*p*가 가리키는 메모리 블록을 해제합니다. *p*는 *PyMem_RawMalloc()*, *PyMem_RawRealloc()* 또는 *PyMem_RawCalloc()* 에 대한 이전 호출로 반환된 것이어야 합니다. 그렇지 않거나 `PyMem_RawFree(p)` 가 앞서 호출되었으면, 정의되지 않은 동작이 일어납니다.

*p*가 `NULL` 이면, 아무 작업도 수행되지 않습니다.

11.4 메모리 인터페이스

ANSI C 표준에 따라 모델링 되었지만 0바이트를 요청할 때의 동작을 지정한 다음 함수 집합은 파이썬 힙에서 메모리를 할당하고 해제하는 데 사용할 수 있습니다.

기본 메모리 할당자는 *pymalloc* 메모리 할당자를 사용합니다.

경고

이 함수를 사용할 때는 *GIL*을 유지해야 합니다.

버전 3.6에서 변경: 기본 할당자는 이제 시스템 `malloc()` 대신 pymalloc 입니다.

void *PyMem_Malloc(size_t n)

Part of the Stable ABI. Allocates *n* bytes and returns a pointer of type `void*` to the allocated memory, or `NULL` if the request fails.

0바이트를 요청하면 `PyMem_Malloc(1)` 이 대신 호출된 것처럼 가능하면 고유한 `NULL` 이 아닌 포인터를 반환합니다. 메모리는 어떤 식으로든 초기화되지 않습니다.

void *PyMem_Calloc(size_t nelem, size_t elsize)

Part of the Stable ABI *since version 3.7.* Allocates *nelem* elements each whose size in bytes is *elsize* and returns a pointer of type `void*` to the allocated memory, or `NULL` if the request fails. The memory is initialized to zeros.

0개의 요소나 0바이트 크기의 요소를 요청하면 `PyMem_Calloc(1, 1)` 이 대신 호출된 것처럼 가능하면 고유한 `NULL` 이 아닌 포인터를 반환합니다.

Added in version 3.5.

void *PyMem_Realloc(void *p, size_t n)

Part of the Stable ABI. *p*가 가리키는 메모리 블록의 크기를 *n* 바이트로 조정합니다. 내용은 이전과 새로운 크기의 최솟값 내에서는 변경되지 않습니다.

*p*가 NULL이면, 호출은 PyMem_Malloc(n)과 동등합니다; 그렇지 않고 *n*이 0과 같으면, 메모리 블록의 크기는 조정되지만 해제되지는 않으며, 반환된 포인터는 NULL이 아닙니다.

*p*가 NULL이 아닌 한, *PyMem_Malloc()*, *PyMem_Realloc()* 또는 *PyMem_Calloc()*에 대한 이전 호출이 반환한 것이어야 합니다.

요청이 실패하면, *PyMem_Realloc()*은 NULL을 반환하고 *p*는 이전 메모리 영역에 대한 유효한 포인터로 유지됩니다.

void **PyMem_Free**(void *p)

Part of the Stable ABI. *p*가 가리키는 메모리 블록을 해제합니다. *p*는 *PyMem_Malloc()*, *PyMem_Realloc()* 또는 *PyMem_Calloc()*에 대한 이전 호출이 반환한 것이어야 합니다. 그렇지 않거나 PyMem_Free(p)가 앞서 호출되었으면 정의되지 않은 동작이 일어납니다.

*p*가 NULL이면, 아무 작업도 수행되지 않습니다.

편의를 위해 다음과 같은 형 지향 매크로가 제공됩니다. *TYPE*이 모든 C 형을 나타냄에 유의하십시오.

PyMem_New(TYPE, n)

Same as *PyMem_Malloc()*, but allocates `(n * sizeof(TYPE))` bytes of memory. Returns a pointer cast to `TYPE*`. The memory will not have been initialized in any way.

PyMem_Resize(p, TYPE, n)

Same as *PyMem_Realloc()*, but the memory block is resized to `(n * sizeof(TYPE))` bytes. Returns a pointer cast to `TYPE*`. On return, *p* will be a pointer to the new memory area, or NULL in the event of failure.

이것은 C 전처리기 매크로입니다; *p*는 항상 다시 대입됩니다. 에러를 처리할 때 메모리 손실을 피하려면 *p*의 원래 값을 보관하십시오.

void **PyMem_Del**(void *p)

*PyMem_Free()*와 같습니다.

또한, 위에 나열된 C API 함수를 사용하지 않고, 파이썬 메모리 할당자를 직접 호출하기 위해 다음 매크로 집합이 제공됩니다. 그러나, 이들을 사용하면 파이썬 버전을 가로지르는 바이너리 호환성이 유지되지 않아서 확장 모듈에서는 폐지되었습니다.

- PyMem_MALLOC(size)
- PyMem_NEW(type, size)
- PyMem_REALLOC(ptr, size)
- PyMem_RESIZE(ptr, type, size)
- PyMem_FREE(ptr)
- PyMem_DEL(ptr)

11.5 객체 할당자

ANSI C 표준에 따라 모델링 되었지만 0바이트를 요청할 때의 동작을 지정한 다음 함수 집합은 파이썬 힙에서 메모리를 할당하고 해제하는 데 사용할 수 있습니다.

참고

There is no guarantee that the memory returned by these allocators can be successfully cast to a Python object when intercepting the allocating functions in this domain by the methods described in the *Customize Memory Allocators* section.

기본 객체 할당자는 *pymalloc* 메모리 할당자를 사용합니다.

경고

이 함수를 사용할 때는 *GIL*을 유지해야 합니다.

void *__PyObject_Malloc__(size_t n)

Part of the Stable ABI. Allocates *n* bytes and returns a pointer of type `void*` to the allocated memory, or `NULL` if the request fails.

0바이트를 요청하면 `PyObject_Malloc(1)` 이 대신 호출된 것처럼 가능하면 고유한 `NULL`이 아닌 포인터를 반환합니다. 메모리는 어떤 식으로든 초기화되지 않습니다.

void *__PyObject_Calloc__(size_t nelem, size_t elsize)

Part of the Stable ABI *since version 3.7.* Allocates *nelem* elements each whose size in bytes is *elsize* and returns a pointer of type `void*` to the allocated memory, or `NULL` if the request fails. The memory is initialized to zeros.

0개의 요소나 0바이트 크기의 요소를 요청하면 `PyObject_Calloc(1, 1)` 이 대신 호출된 것처럼 가능하면 고유한 `NULL`이 아닌 포인터를 반환합니다.

Added in version 3.5.

void *__PyObject_Realloc__(void *p, size_t n)

Part of the Stable ABI. *p*가 가리키는 메모리 블록의 크기를 *n* 바이트로 조정합니다. 내용은 이전과 새로운 크기의 최솟값 내에서는 변경되지 않습니다.

*p*가 `NULL`이면, 호출은 `PyObject_Malloc(n)` 과 동등합니다; 그렇지 않고 *n*이 0과 같으면, 메모리 블록의 크기는 조정되지만 해제되지 않고, 반환된 포인터는 `NULL`이 아닙니다.

*p*가 `NULL`이 아닌 한, *PyObject_Malloc()*, *PyObject_Realloc()* 또는 *PyObject_Calloc()*에 대한 이전 호출에 의해 반환된 것이어야 합니다.

요청이 실패하면, *PyObject_Realloc()*은 `NULL`을 반환하고 *p*는 이전 메모리 영역에 대한 유효한 포인터로 유지됩니다.

void __PyObject_Free__(void *p)

Part of the Stable ABI. *p*가 가리키는 메모리 블록을 해제합니다. 이 블록은 *PyObject_Malloc()*, *PyObject_Realloc()* 또는 *PyObject_Calloc()*에 대한 이전 호출에 의해 반환된 것이어야 합니다. 그렇지 않거나 `PyObject_Free(p)` 가 이전에 호출되었으면 정의되지 않은 동작이 일어납니다.

*p*가 `NULL`이면, 아무 작업도 수행되지 않습니다.

11.6 기본 메모리 할당자

기본 메모리 할당자:

구성	이름	PyMem_RawMallo	PyMem_Malloc	PyOb-ject_Malloc
릴리스 빌드	`"pymalloc"`	`malloc`	`pymalloc`	`pymalloc`
디버그 빌드	`"pymalloc_debug`	`malloc` + 디버그	`pymalloc` + 디버그	`pymalloc` + 디버그
pymalloc 없는 배포 빌드	`"malloc"`	`malloc`	`malloc`	`malloc`
pymalloc 없는 디버그 빌드	`"malloc_debug"`	`malloc` + 디버그	`malloc` + 디버그	`malloc` + 디버그

범례:

- Name: value for `PYTHONMALLOC` environment variable.

- `malloc`: system allocators from the standard C library, C functions: `malloc()`, `calloc()`, `realloc()` and `free()`.
- `pymalloc`: *pymalloc memory allocator*.
- "+ debug": with *debug hooks on the Python memory allocators*.
- "Debug build": Python build in debug mode.

11.7 메모리 할당자 사용자 정의

Added in version 3.4.

type **PyMemAllocatorEx**

Structure used to describe a memory block allocator. The structure has the following fields:

필드	의미
`void *ctx`	첫 번째 인자로 전달된 사용자 컨텍스트
`void* malloc(void *ctx, size_t size)`	메모리 블록을 할당합니다
`void* calloc(void *ctx, size_t nelem, size_t elsize)`	0으로 초기화된 메모리 블록을 할당합니다
`void* realloc(void *ctx, void *ptr, size_t new_size)`	메모리 블록을 할당하거나 크기 조정합니다
`void free(void *ctx, void *ptr)`	메모리 블록을 해제합니다

버전 3.5에서 변경: The `PyMemAllocator` structure was renamed to *PyMemAllocatorEx* and a new `calloc` field was added.

type **PyMemAllocatorDomain**

할당자 도메인을 식별하는 데 사용되는 열거형. 도메인:

PYMEM_DOMAIN_RAW

함수:

- *PyMem_RawMalloc()*
- *PyMem_RawRealloc()*
- *PyMem_RawCalloc()*
- *PyMem_RawFree()*

PYMEM_DOMAIN_MEM

함수:

- *PyMem_Malloc()*,
- *PyMem_Realloc()*
- *PyMem_Calloc()*
- *PyMem_Free()*

PYMEM_DOMAIN_OBJ

함수:

- *PyObject_Malloc()*
- *PyObject_Realloc()*
- *PyObject_Calloc()*
- *PyObject_Free()*

void **PyMem_GetAllocator**(*PyMemAllocatorDomain* domain, *PyMemAllocatorEx* *allocator)

지정된 도메인의 메모리 블록 할당자를 가져옵니다.

void **PyMem_SetAllocator**(*PyMemAllocatorDomain* domain, *PyMemAllocatorEx* *allocator)

지정된 도메인의 메모리 블록 할당자를 설정합니다.

새 할당자는 0바이트를 요청할 때 고유한 `NULL`이 아닌 포인터를 반환해야 합니다.

For the *PYMEM_DOMAIN_RAW* domain, the allocator must be thread-safe: the *GIL* is not held when the allocator is called.

For the remaining domains, the allocator must also be thread-safe: the allocator may be called in different interpreters that do not share a `GIL`.

새 할당자가 훅이 아니면 (이전 할당자를 호출하지 않으면), *PyMem_SetupDebugHooks()* 함수를 호출하여 새 할당자 위에 디버그 훅을 다시 설치해야 합니다.

See also *PyPreConfig.allocator* and *Preinitialize Python with PyPreConfig*.

> **경고**
>
> *PyMem_SetAllocator()* does have the following contract:
>
> - It can be called after *Py_PreInitialize()* and before *Py_InitializeFromConfig()* to install a custom memory allocator. There are no restrictions over the installed allocator other than the ones imposed by the domain (for instance, the Raw Domain allows the allocator to be called without the GIL held). See *the section on allocator domains* for more information.
> - If called after Python has finish initializing (after *Py_InitializeFromConfig()* has been called) the allocator **must** wrap the existing allocator. Substituting the current allocator for some other arbitrary one is **not supported**.

버전 3.12에서 변경: All allocators must be thread-safe.

void **PyMem_SetupDebugHooks**(void)

Setup *debug hooks in the Python memory allocators* to detect memory errors.

11.8 Debug hooks on the Python memory allocators

When Python is built in debug mode, the *PyMem_SetupDebugHooks()* function is called at the *Python preinitialization* to setup debug hooks on Python memory allocators to detect memory errors.

The `PYTHONMALLOC` environment variable can be used to install debug hooks on a Python compiled in release mode (ex: `PYTHONMALLOC=debug`).

The *PyMem_SetupDebugHooks()* function can be used to set debug hooks after calling *PyMem_SetAllocator()*.

These debug hooks fill dynamically allocated memory blocks with special, recognizable bit patterns. Newly allocated memory is filled with the byte `0xCD` (`PYMEM_CLEANBYTE`), freed memory is filled with the byte `0xDD` (`PYMEM_DEADBYTE`). Memory blocks are surrounded by "forbidden bytes" filled with the byte `0xFD` (`PYMEM_FORBIDDENBYTE`). Strings of these bytes are unlikely to be valid addresses, floats, or ASCII strings.

실행 시간 검사:

- Detect API violations. For example, detect if *PyObject_Free()* is called on a memory block allocated by *PyMem_Malloc()*.
- Detect write before the start of the buffer (buffer underflow).
- Detect write after the end of the buffer (buffer overflow).
- Check that the *GIL* is held when allocator functions of *PYMEM_DOMAIN_OBJ* (ex: *PyObject_Malloc()*) and *PYMEM_DOMAIN_MEM* (ex: *PyMem_Malloc()*) domains are called.

에러가 발생하면, 디버그 훅은 tracemalloc 모듈을 사용하여 메모리 블록이 할당된 곳의 트레이스백을 가져옵니다. tracemalloc이 파이썬 메모리 할당을 추적 중이고 메모리 블록이 추적될 때만 트레이스백이 표시됩니다.

Let *S* = `sizeof(size_t)`. `2*S` bytes are added at each end of each block of *N* bytes requested. The memory layout is like so, where p represents the address returned by a malloc-like or realloc-like function (`p[i:j]` means the slice of bytes from `*(p+i)` inclusive up to `*(p+j)` exclusive; note that the treatment of negative indices differs from a Python slice):

p[-2*S:-S]
Number of bytes originally asked for. This is a size_t, big-endian (easier to read in a memory dump).

p[-S]
API identifier (ASCII character):

- `'r'` for *PYMEM_DOMAIN_RAW*.
- `'m'` for *PYMEM_DOMAIN_MEM*.
- `'o'` for *PYMEM_DOMAIN_OBJ*.

p[-S+1:0]
Copies of PYMEM_FORBIDDENBYTE. Used to catch under- writes and reads.

p[0:N]
The requested memory, filled with copies of PYMEM_CLEANBYTE, used to catch reference to uninitialized memory. When a realloc-like function is called requesting a larger memory block, the new excess bytes are also filled with PYMEM_CLEANBYTE. When a free-like function is called, these are overwritten with PYMEM_DEADBYTE, to catch reference to freed memory. When a realloc- like function is called requesting a smaller memory block, the excess old bytes are also filled with PYMEM_DEADBYTE.

p[N:N+S]
Copies of PYMEM_FORBIDDENBYTE. Used to catch over- writes and reads.

p[N+S:N+2*S]
Only used if the `PYMEM_DEBUG_SERIALNO` macro is defined (not defined by default).

A serial number, incremented by 1 on each call to a malloc-like or realloc-like function. Big-endian `size_t`. If "bad memory" is detected later, the serial number gives an excellent way to set a breakpoint on the next run, to capture the instant at which this block was passed out. The static function bumpserialno() in obmalloc.c is the only place the serial number is incremented, and exists so you can set such a breakpoint easily.

A realloc-like or free-like function first checks that the PYMEM_FORBIDDENBYTE bytes at each end are intact. If they' ve been altered, diagnostic output is written to stderr, and the program is aborted via Py_FatalError(). The other main failure mode is provoking a memory error when a program reads up one of the special bit patterns and tries to use it as an address. If you get in a debugger then and look at the object, you' re likely to see that it' s entirely filled with PYMEM_DEADBYTE (meaning freed memory is getting used) or PYMEM_CLEANBYTE (meaning uninitialized memory is getting used).

버전 3.6에서 변경: The *PyMem_SetupDebugHooks()* function now also works on Python compiled in release mode. On error, the debug hooks now use `tracemalloc` to get the traceback where a memory block was allocated. The debug hooks now also check if the GIL is held when functions of *PYMEM_DOMAIN_OBJ* and *PYMEM_DOMAIN_MEM* domains are called.

버전 3.8에서 변경: Byte patterns `0xCB` (`PYMEM_CLEANBYTE`), `0xDB` (`PYMEM_DEADBYTE`) and `0xFB` (`PYMEM_FORBIDDENBYTE`) have been replaced with `0xCD`, `0xDD` and `0xFD` to use the same values than Windows CRT debug `malloc()` and `free()`.

11.9 pymalloc 할당자

Python has a *pymalloc* allocator optimized for small objects (smaller or equal to 512 bytes) with a short lifetime. It uses memory mappings called "arenas" with a fixed size of either 256 KiB on 32-bit platforms or 1 MiB on 64-bit platforms. It falls back to *PyMem_RawMalloc()* and *PyMem_RawRealloc()* for allocations larger than 512 bytes.

pymalloc is the *default allocator* of the `PYMEM_DOMAIN_MEM` (ex: `PyMem_Malloc()`) and `PYMEM_DOMAIN_OBJ` (ex: `PyObject_Malloc()`) domains.

아레나 할당자는 다음 함수를 사용합니다:

- `VirtualAlloc()` and `VirtualFree()` on Windows,
- `mmap()` and `munmap()` if available,
- 그렇지 않으면 `malloc()`과 `free()`

This allocator is disabled if Python is configured with the `--without-pymalloc` option. It can also be disabled at runtime using the `PYTHONMALLOC` environment variable (ex: `PYTHONMALLOC=malloc`).

11.9.1 pymalloc 아레나 할당자 사용자 정의

Added in version 3.4.

type **PyObjectArenaAllocator**

아레나 할당자를 기술하는 데 사용되는 구조체. 이 구조체에는 세 개의 필드가 있습니다:

필드	의미
`void *ctx`	첫 번째 인자로 전달된 사용자 컨텍스트
`void* alloc(void *ctx, size_t size)`	size 바이트의 아레나를 할당합니다
`void free(void *ctx, void *ptr, size_t size)`	아레나를 해제합니다

void **PyObject_GetArenaAllocator**(*PyObjectArenaAllocator* *allocator)

아레나 할당자를 얻습니다.

void **PyObject_SetArenaAllocator**(*PyObjectArenaAllocator* *allocator)

아레나 할당자를 설정합니다.

11.10 tracemalloc C API

Added in version 3.7.

int **PyTraceMalloc_Track**(unsigned int domain, uintptr_t ptr, size_t size)

`tracemalloc` 모듈에서 할당된 메모리 블록을 추적합니다.

성공하면 `0`을 반환하고, 에러가 발생하면 (추적을 저장하기 위한 메모리를 할당하지 못했습니다) `-1`을 반환합니다. tracemalloc이 비활성화되었으면 `-2`를 반환합니다.

메모리 블록이 이미 추적되면, 기존 추적을 갱신합니다.

int **PyTraceMalloc_Untrack**(unsigned int domain, uintptr_t ptr)

`tracemalloc` 모듈에서 할당된 메모리 블록을 추적 해제합니다. 블록이 추적되지 않으면 아무것도 하지 않습니다.

tracemalloc이 비활성화되었으면 `-2`를 반환하고, 그렇지 않으면 `0`을 반환합니다.

11.11 예

다음은 개요 섹션에서 따온 예제입니다. I/O 버퍼가 첫 번째 함수 집합을 사용하여 파이썬 힙에서 할당되도록 다시 작성되었습니다:

```
PyObject *res;
char *buf = (char *) PyMem_Malloc(BUFSIZ); /* for I/O */
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
if (buf == NULL)
    return PyErr_NoMemory();
/* ...Do some I/O operation involving buf... */
res = PyBytes_FromString(buf);
PyMem_Free(buf); /* allocated with PyMem_Malloc */
return res;
```

형 지향 함수 집합을 사용하는 같은 코드입니다:

```
PyObject *res;
char *buf = PyMem_New(char, BUFSIZ); /* for I/O */

if (buf == NULL)
    return PyErr_NoMemory();
/* ...Do some I/O operation involving buf... */
res = PyBytes_FromString(buf);
PyMem_Del(buf); /* allocated with PyMem_New */
return res;
```

위의 두 가지 예에서, 버퍼는 항상 같은 집합에 속하는 함수를 통해 조작됨에 유의하십시오. 실제로, 서로 다른 할당자를 혼합할 위험이 최소로 줄어들도록, 주어진 메모리 블록에 대해 같은 메모리 API 패밀리를 사용하는 것은 필수입니다. 다음 코드 시퀀스에는 두 개의 에러가 있으며, 그중 하나는 서로 다른 힙에서 작동하는 두 개의 다른 할당자를 혼합하기 때문에 치명적(*fatal*)인 것으로 표시됩니다.

```
char *buf1 = PyMem_New(char, BUFSIZ);
char *buf2 = (char *) malloc(BUFSIZ);
char *buf3 = (char *) PyMem_Malloc(BUFSIZ);
...
PyMem_Del(buf3);  /* Wrong -- should be PyMem_Free() */
free(buf2);       /* Right -- allocated via malloc() */
free(buf1);       /* Fatal -- should be PyMem_Del()  */
```

In addition to the functions aimed at handling raw memory blocks from the Python heap, objects in Python are allocated and released with *PyObject_New*, *PyObject_NewVar* and *PyObject_Del()*.

이것들은 C로 새로운 객체 형을 정의하고 구현하는 것에 대한 다음 장에서 설명될 것입니다.

CHAPTER 12

객체 구현 지원

이 장에서는 새 객체 형을 정의할 때 사용되는 함수, 형 및 매크로에 관해 설명합니다.

12.1 힙에 객체 할당하기

PyObject ***_PyObject_New**(*PyTypeObject* *type)

Return value: New reference.

PyVarObject ***_PyObject_NewVar**(*PyTypeObject* *type, *Py_ssize_t* size)

Return value: New reference.

PyObject ***PyObject_Init**(*PyObject* *op, *PyTypeObject* *type)

Return value: Borrowed reference. Part of the Stable ABI. Initialize a newly allocated object *op* with its type and initial reference. Returns the initialized object. Other fields of the object are not affected.

PyVarObject ***PyObject_InitVar**(*PyVarObject* *op, *PyTypeObject* *type, *Py_ssize_t* size)

Return value: Borrowed reference. Part of the Stable ABI. 이것은 `PyObject_Init()`가 수행하는 모든 작업을 수행하고, 가변 크기 객체의 길이 정보도 초기화합니다.

PyObject_New(TYPE, typeobj)

Allocate a new Python object using the C structure type *TYPE* and the Python type object *typeobj* (`PyTypeObject*`). Fields not defined by the Python object header are not initialized. The caller will own the only reference to the object (i.e. its reference count will be one). The size of the memory allocation is determined from the `tp_basicsize` field of the type object.

Note that this function is unsuitable if *typeobj* has `Py_TPFLAGS_HAVE_GC` set. For such objects, use `PyObject_GC_New()` instead.

PyObject_NewVar(TYPE, typeobj, size)

Allocate a new Python object using the C structure type *TYPE* and the Python type object *typeobj* (`PyTypeObject*`). Fields not defined by the Python object header are not initialized. The allocated memory allows for the *TYPE* structure plus *size* (`Py_ssize_t`) fields of the size given by the `tp_itemsize` field of *typeobj*. This is useful for implementing objects like tuples, which are able to determine their size at construction time. Embedding the array of fields into the same allocation decreases the number of allocations, improving the memory management efficiency.

Note that this function is unsuitable if *typeobj* has `Py_TPFLAGS_HAVE_GC` set. For such objects, use `PyObject_GC_NewVar()` instead.

void **PyObject_Del**(void *op)

Releases memory allocated to an object using *PyObject_New* or *PyObject_NewVar*. This is normally called from the *tp_dealloc* handler specified in the object' s type. The fields of the object should not be accessed after this call as the memory is no longer a valid Python object.

PyObject **_Py_NoneStruct**

파이썬에서 `None`으로 노출되는 객체. 이 객체에 대한 포인터로 평가되는 *Py_None* 매크로를 사용해서 액세스해야 합니다.

더 보기

PyModule_Create()
확장 모듈을 할당하고 만듭니다.

12.2 공통 객체 구조체

파이썬의 객체 형 정의에 사용되는 많은 구조체가 있습니다. 이 섹션에서는 이러한 구조체와 사용 방법에 관해 설명합니다.

12.2.1 기본 객체 형과 매크로

All Python objects ultimately share a small number of fields at the beginning of the object' s representation in memory. These are represented by the *PyObject* and *PyVarObject* types, which are defined, in turn, by the expansions of some macros also used, whether directly or indirectly, in the definition of all other Python objects. Additional macros can be found under *reference counting*.

type **PyObject**

Part of the Limited API. (*Only some members are part of the stable ABI.*) All object types are extensions of this type. This is a type which contains the information Python needs to treat a pointer to an object as an object. In a normal "release" build, it contains only the object' s reference count and a pointer to the corresponding type object. Nothing is actually declared to be a *PyObject*, but every pointer to a Python object can be cast to a *PyObject**. Access to the members must be done by using the macros *Py_REFCNT* and *Py_TYPE*.

type **PyVarObject**

Part of the Limited API. (*Only some members are part of the stable ABI.*) This is an extension of *PyObject* that adds the *ob_size* field. This is only used for objects that have some notion of *length*. This type does not often appear in the Python/C API. Access to the members must be done by using the macros *Py_REFCNT*, *Py_TYPE*, and *Py_SIZE*.

PyObject_HEAD

길이가 변하지 않는 객체를 나타내는 새로운 형을 선언할 때 사용되는 매크로입니다. PyObject_HEAD 매크로는 다음과 같이 확장됩니다:

```
PyObject ob_base;
```

위의 *PyObject* 설명서를 참조하십시오.

PyObject_VAR_HEAD

인스턴스마다 길이가 다른 객체를 나타내는 새로운 형을 선언할 때 사용되는 매크로입니다. PyObject_VAR_HEAD 매크로는 다음과 같이 확장됩니다:

```
PyVarObject ob_base;
```

위의 *PyVarObject* 설명서를 참조하십시오.

PyTypeObject **PyBaseObject_Type**

Part of the Stable ABI. The base class of all other objects, the same as `object` in Python.

int **Py_Is**(*PyObject* *x, *PyObject* *y)

Part of the Stable ABI *since version 3.10.* Test if the *x* object is the *y* object, the same as `x is y` in Python.

Added in version 3.10.

int **Py_IsNone**(*PyObject* *x)

Part of the Stable ABI *since version 3.10.* Test if an object is the `None` singleton, the same as `x is None` in Python.

Added in version 3.10.

int **Py_IsTrue**(*PyObject* *x)

Part of the Stable ABI *since version 3.10.* Test if an object is the `True` singleton, the same as `x is True` in Python.

Added in version 3.10.

int **Py_IsFalse**(*PyObject* *x)

Part of the Stable ABI *since version 3.10.* Test if an object is the `False` singleton, the same as `x is False` in Python.

Added in version 3.10.

PyTypeObject ***Py_TYPE**(*PyObject* *o)

Get the type of the Python object *o*.

Return a *borrowed reference*.

Use the *Py_SET_TYPE()* function to set an object type.

버전 3.11에서 변경: *Py_TYPE()* is changed to an inline static function. The parameter type is no longer `const PyObject*`.

int **Py_IS_TYPE**(*PyObject* *o, *PyTypeObject* *type)

객체 *o*의 형이 *type*이면 0이 아닌 값을 반환합니다. 그렇지 않으면 0을 반환합니다. `Py_TYPE(o) == type`과 동등합니다.

Added in version 3.9.

void **Py_SET_TYPE**(*PyObject* *o, *PyTypeObject* *type)

객체 *o*의 형을 *type*으로 설정합니다.

Added in version 3.9.

Py_ssize_t **Py_SIZE**(*PyVarObject* *o)

Get the size of the Python object *o*.

Use the *Py_SET_SIZE()* function to set an object size.

버전 3.11에서 변경: *Py_SIZE()* is changed to an inline static function. The parameter type is no longer `const PyVarObject*`.

void **Py_SET_SIZE**(*PyVarObject* *o, *Py_ssize_t* size)

객체 *o*의 크기를 *size*로 설정합니다.

Added in version 3.9.

PyObject_HEAD_INIT(type)

이것은 새로운 *PyObject* 형의 초기화 값으로 확장되는 매크로입니다. 이 매크로는 다음으로 확장됩니다:

```
_PyObject_EXTRA_INIT
1, type,
```

PyVarObject_HEAD_INIT(type, size)

This is a macro which expands to initialization values for a new *PyVarObject* type, including the *ob_size* field. This macro expands to:

```
_PyObject_EXTRA_INIT
1, type, size,
```

12.2.2 함수와 메서드 구현

type **PyCFunction**

Part of the Stable ABI. Type of the functions used to implement most Python callables in C. Functions of this type take two *PyObject** parameters and return one such value. If the return value is NULL, an exception shall have been set. If not NULL, the return value is interpreted as the return value of the function as exposed in Python. The function must return a new reference.

함수 서명은 다음과 같습니다:

```
PyObject *PyCFunction(PyObject *self,
                      PyObject *args);
```

type **PyCFunctionWithKeywords**

Part of the Stable ABI. Type of the functions used to implement Python callables in C with signature *METH_VARARGS | METH_KEYWORDS*. The function signature is:

```
PyObject *PyCFunctionWithKeywords(PyObject *self,
                                  PyObject *args,
                                  PyObject *kwargs);
```

type **_PyCFunctionFast**

Type of the functions used to implement Python callables in C with signature *METH_FASTCALL*. The function signature is:

```
PyObject *_PyCFunctionFast(PyObject *self,
                           PyObject *const *args,
                           Py_ssize_t nargs);
```

type **_PyCFunctionFastWithKeywords**

Type of the functions used to implement Python callables in C with signature *METH_FASTCALL | METH_KEYWORDS*. The function signature is:

```
PyObject *_PyCFunctionFastWithKeywords(PyObject *self,
                                       PyObject *const *args,
                                       Py_ssize_t nargs,
                                       PyObject *kwnames);
```

type **PyCMethod**

Type of the functions used to implement Python callables in C with signature *METH_METHOD | METH_FASTCALL | METH_KEYWORDS*. The function signature is:

```
PyObject *PyCMethod(PyObject *self,
                    PyTypeObject *defining_class,
                    PyObject *const *args,
                    Py_ssize_t nargs,
                    PyObject *kwnames)
```

Added in version 3.9.

type **PyMethodDef**

Part of the Stable ABI (*including all members*). 확장형의 메서드를 기술하는 데 사용되는 구조체. 이 구조체에는 네 개의 필드가 있습니다:

const char ***ml_name**

Name of the method.

PyCFunction **ml_meth**

Pointer to the C implementation.

int **ml_flags**

Flags bits indicating how the call should be constructed.

const char ***ml_doc**

Points to the contents of the docstring.

The *ml_meth* is a C function pointer. The functions may be of different types, but they always return *PyObject**. If the function is not of the *PyCFunction*, the compiler will require a cast in the method table. Even though *PyCFunction* defines the first parameter as *PyObject**, it is common that the method implementation uses the specific C type of the *self* object.

The *ml_flags* field is a bitfield which can include the following flags. The individual flags indicate either a calling convention or a binding convention.

다음과 같은 호출 규칙이 있습니다:

METH_VARARGS

This is the typical calling convention, where the methods have the type *PyCFunction*. The function expects two *PyObject** values. The first one is the *self* object for methods; for module functions, it is the module object. The second parameter (often called *args*) is a tuple object representing all arguments. This parameter is typically processed using *PyArg_ParseTuple()* or *PyArg_UnpackTuple()*.

METH_KEYWORDS

Can only be used in certain combinations with other flags: *METH_VARARGS* | *METH_KEYWORDS*, *METH_FASTCALL* | *METH_KEYWORDS* and *METH_METHOD* | *METH_FASTCALL* | *METH_KEYWORDS*.

METH_VARARGS* | *METH_KEYWORDS

이러한 플래그가 있는 메서드는 *PyCFunctionWithKeywords* 형이어야 합니다. 이 함수는 세 개의 매개 변수를 기대합니다: *self*, *args*, *kwargs*. 여기서 *kwargs*는 모든 키워드 인자의 딕셔너리이거나 키워드 인자가 없으면 NULL일 수 있습니다. 매개 변수는 일반적으로 *PyArg_ParseTupleAndKeywords()*를 사용하여 처리됩니다.

METH_FASTCALL

Fast calling convention supporting only positional arguments. The methods have the type *_PyCFunctionFast*. The first parameter is *self*, the second parameter is a C array of *PyObject** values indicating the arguments and the third parameter is the number of arguments (the length of the array).

Added in version 3.7.

버전 3.10에서 변경: METH_FASTCALL is now part of the *stable ABI*.

METH_FASTCALL* | *METH_KEYWORDS

Extension of *METH_FASTCALL* supporting also keyword arguments, with methods of type *_PyCFunctionFastWithKeywords*. Keyword arguments are passed the same way as in the *vectorcall protocol*: there is an additional fourth *PyObject** parameter which is a tuple representing the names of the keyword arguments (which are guaranteed to be strings) or possibly NULL if there are no keywords. The values of the keyword arguments are stored in the *args* array, after the positional arguments.

Added in version 3.7.

METH_METHOD

Can only be used in the combination with other flags: *METH_METHOD* | *METH_FASTCALL* | *METH_KEYWORDS*.

METH_METHOD | *METH_FASTCALL* | *METH_KEYWORDS*
: Extension of *METH_FASTCALL | METH_KEYWORDS* supporting the *defining class*, that is, the class that contains the method in question. The defining class might be a superclass of `Py_TYPE(self)`.

  메서드는 *PyCMethod* 형이어야 하는데, `self` 뒤에 `defining_class` 인자가 추가된 `METH_FASTCALL | METH_KEYWORDS`와 같습니다.

  Added in version 3.9.

METH_NOARGS
: Methods without parameters don' t need to check whether arguments are given if they are listed with the *METH_NOARGS* flag. They need to be of type *PyCFunction*. The first parameter is typically named *self* and will hold a reference to the module or object instance. In all cases the second parameter will be `NULL`.

  The function must have 2 parameters. Since the second parameter is unused, *Py_UNUSED* can be used to prevent a compiler warning.

METH_O
: Methods with a single object argument can be listed with the *METH_O* flag, instead of invoking *PyArg_ParseTuple()* with a `"O"` argument. They have the type *PyCFunction*, with the *self* parameter, and a *PyObject** parameter representing the single argument.

이 두 상수는 호출 규칙을 나타내는 데 사용되지 않고 클래스의 메서드와 함께 사용할 때 바인딩을 나타냅니다. 모듈에 정의된 함수에는 사용할 수 없습니다. 이러한 플래그 중 최대 하나를 주어진 메서드에 대해 설정할 수 있습니다.

METH_CLASS
: 메서드로 형의 인스턴스가 아닌 형 객체가 첫 번째 매개 변수로 전달됩니다. `classmethod()` 내장 함수를 사용할 때 만들어지는 것과 유사한 클래스 메서드(*class methods*)를 만드는 데 사용됩니다.

METH_STATIC
: 메서드로 형의 인스턴스가 아닌 `NULL`이 첫 번째 매개 변수로 전달됩니다. `staticmethod()` 내장 함수를 사용할 때 만들어지는 것과 유사한 정적 메서드(*static methods*)를 만드는 데 사용됩니다.

하나의 다른 상수는 같은 메서드 이름을 가진 다른 정의 대신 메서드가 로드되는지를 제어합니다.

METH_COEXIST
: The method will be loaded in place of existing definitions. Without *METH_COEXIST*, the default is to skip repeated definitions. Since slot wrappers are loaded before the method table, the existence of a *sq_contains* slot, for example, would generate a wrapped method named `__contains__()` and preclude the loading of a corresponding PyCFunction with the same name. With the flag defined, the PyCFunction will be loaded in place of the wrapper object and will co-exist with the slot. This is helpful because calls to PyCFunctions are optimized more than wrapper object calls.

PyObject ***PyCMethod_New**(*PyMethodDef* *ml, *PyObject* *self, *PyObject* *module, *PyTypeObject* *cls)
: *Return value: New reference. Part of the* Stable ABI *since version 3.9.* Turn *ml* into a Python *callable* object. The caller must ensure that *ml* outlives the *callable*. Typically, *ml* is defined as a static variable.

  The *self* parameter will be passed as the *self* argument to the C function in `ml->ml_meth` when invoked. *self* can be `NULL`.

  The *callable* object' s `__module__` attribute can be set from the given *module* argument. *module* should be a Python string, which will be used as name of the module the function is defined in. If unavailable, it can be set to `None` or `NULL`.

  > **더 보기**
  >
  > `function.__module__`

  The *cls* parameter will be passed as the *defining_class* argument to the C function. Must be set if *METH_METHOD* is set on `ml->ml_flags`.

  Added in version 3.9.

PyObject ***PyCFunction_NewEx**(*PyMethodDef* *ml, *PyObject* *self, *PyObject* *module)

Return value: New reference. Part of the Stable ABI. Equivalent to `PyCMethod_New(ml, self, module, NULL)`.

PyObject ***PyCFunction_New**(*PyMethodDef* *ml, *PyObject* *self)

Return value: New reference. Part of the Stable ABI *since version 3.4.* Equivalent to `PyCMethod_New(ml, self, NULL, NULL)`.

12.2.3 확장형의 어트리뷰트 액세스

type **PyMemberDef**

Part of the Stable ABI (*including all members*). Structure which describes an attribute of a type which corresponds to a C struct member. When defining a class, put a NULL-terminated array of these structures in the *tp_members* slot.

Its fields are, in order:

const char ***name**

Name of the member. A NULL value marks the end of a `PyMemberDef[]` array.

The string should be static, no copy is made of it.

int **type**

The type of the member in the C struct. See *Member types* for the possible values.

Py_ssize_t **offset**

The offset in bytes that the member is located on the type' s object struct.

int **flags**

Zero or more of the *Member flags*, combined using bitwise OR.

const char ***doc**

The docstring, or NULL. The string should be static, no copy is made of it. Typically, it is defined using *PyDoc_STR*.

By default (when *flags* is `0`), members allow both read and write access. Use the *Py_READONLY* flag for read-only access. Certain types, like *Py_T_STRING*, imply *Py_READONLY*. Only *Py_T_OBJECT_EX* (and legacy *T_OBJECT*) members can be deleted.

For heap-allocated types (created using *PyType_FromSpec()* or similar), `PyMemberDef` may contain a definition for the special member `"__vectorcalloffset__"`, corresponding to *tp_vectorcall_offset* in type objects. These must be defined with `Py_T_PYSSIZET` and `Py_READONLY`, for example:

```
static PyMemberDef spam_type_members[] = {
    {"__vectorcalloffset__", Py_T_PYSSIZET,
     offsetof(Spam_object, vectorcall), Py_READONLY},
    {NULL}  /* Sentinel */
};
```

(You may need to `#include <stddef.h>` for `offsetof()`.)

The legacy offsets *tp_dictoffset* and *tp_weaklistoffset* can be defined similarly using `"__dictoffset__"` and `"__weaklistoffset__"` members, but extensions are strongly encouraged to use *Py_TPFLAGS_MANAGED_DICT* and *Py_TPFLAGS_MANAGED_WEAKREF* instead.

버전 3.12에서 변경: `PyMemberDef` is always available. Previously, it required including `"structmember.h"`.

PyObject ***PyMember_GetOne**(const char *obj_addr, struct *PyMemberDef* *m)

Part of the Stable ABI. Get an attribute belonging to the object at address *obj_addr*. The attribute is described by `PyMemberDef` *m*. Returns `NULL` on error.

버전 3.12에서 변경: `PyMember_GetOne` is always available. Previously, it required including `"structmember.h"`.

int **PyMember_SetOne**(char *obj_addr, struct *PyMemberDef* *m, *PyObject* *o)

Part of the Stable ABI. Set an attribute belonging to the object at address *obj_addr* to object *o*. The attribute to set is described by `PyMemberDef` *m*. Returns `0` if successful and a negative value on failure.

버전 3.12에서 변경: `PyMember_SetOne` is always available. Previously, it required including `"structmember.h"`.

Member flags

The following flags can be used with *`PyMemberDef.flags`*:

Py_READONLY

Not writable.

Py_AUDIT_READ

Emit an `object.__getattr__` audit event before reading.

Py_RELATIVE_OFFSET

Indicates that the *`offset`* of this `PyMemberDef` entry indicates an offset from the subclass-specific data, rather than from `PyObject`.

Can only be used as part of *`Py_tp_members` `slot`* when creating a class using negative *`basicsize`*. It is mandatory in that case.

This flag is only used in *`PyType_Slot`*. When setting *`tp_members`* during class creation, Python clears it and sets *`PyMemberDef.offset`* to the offset from the `PyObject` struct.

버전 3.10에서 변경: The `RESTRICTED`, `READ_RESTRICTED` and `WRITE_RESTRICTED` macros available with `#include "structmember.h"` are deprecated. `READ_RESTRICTED` and `RESTRICTED` are equivalent to *`Py_AUDIT_READ`*; `WRITE_RESTRICTED` does nothing.

버전 3.12에서 변경: The `READONLY` macro was renamed to *`Py_READONLY`*. The `PY_AUDIT_READ` macro was renamed with the `Py_` prefix. The new names are now always available. Previously, these required `#include "structmember.h"`. The header is still available and it provides the old names.

Member types

`PyMemberDef.type` can be one of the following macros corresponding to various C types. When the member is accessed in Python, it will be converted to the equivalent Python type. When it is set from Python, it will be converted back to the C type. If that is not possible, an exception such as `TypeError` or `ValueError` is raised.

Unless marked (D), attributes defined this way cannot be deleted using e.g. `del` or `delattr()`.

매크로 이름	C 형	Python type
Py_T_BYTE	`char`	`int`
Py_T_SHORT	`short`	`int`
Py_T_INT	`int`	`int`
Py_T_LONG	`long`	`int`
Py_T_LONGLONG	`long long`	`int`
Py_T_UBYTE	`unsigned char`	`int`
Py_T_UINT	`unsigned int`	`int`
Py_T_USHORT	`unsigned short`	`int`
Py_T_ULONG	`unsigned long`	`int`
Py_T_ULONGLONG	`unsigned long long`	`int`
Py_T_PYSSIZET	*Py_ssize_t*	`int`
Py_T_FLOAT	`float`	`float`
Py_T_DOUBLE	`double`	`float`
Py_T_BOOL	`char` (written as 0 or 1)	`bool`
Py_T_STRING	`const char*` (*)	`str` (RO)
Py_T_STRING_INPLACE	`const char[]` (*)	`str` (RO)
Py_T_CHAR	`char` (0-127)	`str` (**)
Py_T_OBJECT_EX	*PyObject**	`object` (D)

(*): Zero-terminated, UTF8-encoded C string. With `Py_T_STRING` the C representation is a pointer; with `Py_T_STRING_INPLACE` the string is stored directly in the structure.

(**): String of length 1. Only ASCII is accepted.

(RO): Implies *Py_READONLY*.

(D): Can be deleted, in which case the pointer is set to NULL. Reading a NULL pointer raises AttributeError.

Added in version 3.12: In previous versions, the macros were only available with #include "structmember.h" and were named without the Py_ prefix (e.g. as T_INT). The header is still available and contains the old names, along with the following deprecated types:

T_OBJECT

Like Py_T_OBJECT_EX, but NULL is converted to None. This results in surprising behavior in Python: deleting the attribute effectively sets it to None.

T_NONE

Always None. Must be used with *Py_READONLY*.

Defining Getters and Setters

type **PyGetSetDef**

Part of the Stable ABI (*including all members*). 형에 대한 프로퍼티 같은 액세스를 정의하는 구조체. *PyTypeObject.tp_getset* 슬롯에 대한 설명도 참조하십시오.

const char ***name**

어트리뷰트 이름

getter **get**

C function to get the attribute.

setter **set**

Optional C function to set or delete the attribute. If NULL, the attribute is read-only.

const char ***doc**

선택적 독스트링

void ***closure**

Optional user data pointer, providing additional data for getter and setter.

typedef *PyObject* *(***getter**)(*PyObject**, void*)

Part of the Stable ABI. The get function takes one *PyObject** parameter (the instance) and a user data pointer (the associated closure):

성공하면 새 참조를 반환하고, 실패하면 설정된 예외와 함께 NULL을 반환해야 합니다.

typedef int (***setter**)(*PyObject**, *PyObject**, void*)

Part of the Stable ABI. set functions take two *PyObject** parameters (the instance and the value to be set) and a user data pointer (the associated closure):

어트리뷰트를 삭제해야 하는 경우 두 번째 매개 변수는 NULL입니다. 성공하면 0을, 실패하면 설정된 예외와 함께 -1을 반환해야 합니다.

12.3 Type Object Structures

Perhaps one of the most important structures of the Python object system is the structure that defines a new type: the *PyTypeObject* structure. Type objects can be handled using any of the PyObject_* or PyType_* functions, but do not offer much that' s interesting to most Python applications. These objects are fundamental to how objects behave, so they are very important to the interpreter itself and to any extension module that implements new types.

형 객체는 대부분 표준형보다 상당히 큽니다. 크기가 큰 이유는 각 형 객체가 많은 수의 값을 저장하기 때문인데, 주로 C 함수 포인터이고 각기 형의 기능 중 작은 부분을 구현합니다. 이 섹션에서는 형 객체의 필드를 자세히 살펴봅니다. 필드는 구조체에서 나타나는 순서대로 설명됩니다.

다음의 간략 참조 외에도, 예 섹션은 *PyTypeObject*의 의미와 사용에 대한 통찰을 제공합니다.

12.3.1 간략 참조

"tp 슬롯"

PyTypeObject 슬롯[Page 260, 1]	형	특수 메서드/어트리뷰트	정보[Page 260, 2]			
			C	T	D	I
<R> *tp_name*	const char *	__name__	X	X		
tp_basicsize	*Py_ssize_t*		X	X		X
tp_itemsize	*Py_ssize_t*			X		X
tp_dealloc	*destructor*		X	X		X
tp_vectorcall_offset	*Py_ssize_t*			X		X
(*tp_getattr*)	*getattrfunc*	__getattribute__, __getattr__				G
(*tp_setattr*)	*setattrfunc*	__setattr__, __delattr__				G
tp_as_async	*PyAsyncMethods* *	서브 슬롯				%
tp_repr	*reprfunc*	__repr__	X	X		X
tp_as_number	*PyNumberMethods* *	서브 슬롯				%
tp_as_sequence	*PySequenceMethods* *	서브 슬롯				%
tp_as_mapping	*PyMappingMethods* *	서브 슬롯				%
tp_hash	*hashfunc*	__hash__	X			G
tp_call	*ternaryfunc*	__call__		X		X
tp_str	*reprfunc*	__str__	X			X
tp_getattro	*getattrofunc*	__getattribute__, __getattr__	X	X		G
tp_setattro	*setattrofunc*	__setattr__, __delattr__	X	X		G
tp_as_buffer	*PyBufferProcs* *					%
tp_flags	unsigned long		X	X		?
tp_doc	const char *	__doc__	X	X		
tp_traverse	*traverseproc*			X		G
tp_clear	*inquiry*			X		G
tp_richcompare	*richcmpfunc*	__lt__, __le__, __eq__, __ne__, __gt__, __ge__	X			G
(*tp_weaklistoffset*)	*Py_ssize_t*			X		?
tp_iter	*getiterfunc*	__iter__				X
tp_iternext	*iternextfunc*	__next__				X
tp_methods	*PyMethodDef* []		X	X		
tp_members	*PyMemberDef* []			X		
tp_getset	*PyGetSetDef* []		X	X		
tp_base	*PyTypeObject* *	__base__			X	
tp_dict	*PyObject* *	__dict__			?	
tp_descr_get	*descrgetfunc*	__get__				X
tp_descr_set	*descrsetfunc*	__set__, __delete__				X
(*tp_dictoffset*)	*Py_ssize_t*			X		?
tp_init	*initproc*	__init__	X	X		X
tp_alloc	*allocfunc*		X		?	?
tp_new	*newfunc*	__new__	X	X	?	?
tp_free	*freefunc*		X	X	?	?
tp_is_gc	*inquiry*			X		X
<*tp_bases*>	*PyObject* *	__bases__			~	
<*tp_mro*>	*PyObject* *	__mro__			~	
[*tp_cache*]	*PyObject* *					
[*tp_subclasses*]	void *	__subclasses__				
[*tp_weaklist*]	*PyObject* *					
(*tp_del*)	*destructor*					
[*tp_version_tag*]	unsigned int					
tp_finalize	*destructor*	__del__				X
tp_vectorcall	*vectorcallfunc*					

다음 페이지에 계속

표 1 – 이전 페이지에서 계속

PyTypeObject 슬롯[1]	형	특수 메서드/어트리뷰트	정보[2] C T D I
[*tp_watched*]	unsigned char		

서브 슬롯

슬롯	형	특수 메서드
am_await	*unaryfunc*	__await__
am_aiter	*unaryfunc*	__aiter__
am_anext	*unaryfunc*	__anext__
am_send	*sendfunc*	
nb_add	*binaryfunc*	__add__ __radd__
nb_inplace_add	*binaryfunc*	__iadd__
nb_subtract	*binaryfunc*	__sub__ __rsub__
nb_inplace_subtract	*binaryfunc*	__isub__
nb_multiply	*binaryfunc*	__mul__ __rmul__
nb_inplace_multiply	*binaryfunc*	__imul__
nb_remainder	*binaryfunc*	__mod__ __rmod__
nb_inplace_remainder	*binaryfunc*	__imod__
nb_divmod	*binaryfunc*	__divmod__ __rdivmod__
nb_power	*ternaryfunc*	__pow__ __rpow__
nb_inplace_power	*ternaryfunc*	__ipow__
nb_negative	*unaryfunc*	__neg__
nb_positive	*unaryfunc*	__pos__
nb_absolute	*unaryfunc*	__abs__
nb_bool	*inquiry*	__bool__
nb_invert	*unaryfunc*	__invert__
nb_lshift	*binaryfunc*	__lshift__ __rlshift__
nb_inplace_lshift	*binaryfunc*	__ilshift__
nb_rshift	*binaryfunc*	__rshift__ __rrshift__
nb_inplace_rshift	*binaryfunc*	__irshift__
nb_and	*binaryfunc*	__and__ __rand__

다음 페이지에 계속

[1] (): A slot name in parentheses indicates it is (effectively) deprecated.
<>: Names in angle brackets should be initially set to NULL and treated as read-only.
[]: Names in square brackets are for internal use only.
<R> (as a prefix) means the field is required (must be non-NULL).

[2] 열:
"O": set on *PyBaseObject_Type*
"T": set on *PyType_Type*
"D": 기본값 (슬롯이 NULL로 설정된 경우)

```
X - PyType_Ready sets this value if it is NULL
~ - PyType_Ready always sets this value (it should be NULL)
? - PyType_Ready may set this value depending on other slots

Also see the inheritance column ("I").
```

"I": 상속

```
X - type slot is inherited via *PyType_Ready* if defined with a *NULL* value
% - the slots of the sub-struct are inherited individually
G - inherited, but only in combination with other slots; see the slot's description
? - it's complicated; see the slot's description
```

일부 슬롯은 일반 어트리뷰트 조회 체인을 통해 효과적으로 상속됨에 유의하십시오.

표 2 – 이전 페이지에서 계속

슬롯	형	특수 메서드
nb_inplace_and	*binaryfunc*	__iand__
nb_xor	*binaryfunc*	__xor__ __rxor__
nb_inplace_xor	*binaryfunc*	__ixor__
nb_or	*binaryfunc*	__or__ __ror__
nb_inplace_or	*binaryfunc*	__ior__
nb_int	*unaryfunc*	__int__
nb_reserved	void *	
nb_float	*unaryfunc*	__float__
nb_floor_divide	*binaryfunc*	__floordiv__
nb_inplace_floor_divide	*binaryfunc*	__ifloordiv__
nb_true_divide	*binaryfunc*	__truediv__
nb_inplace_true_divide	*binaryfunc*	__itruediv__
nb_index	*unaryfunc*	__index__
nb_matrix_multiply	*binaryfunc*	__matmul__ __rmatmul__
nb_inplace_matrix_multiply	*binaryfunc*	__imatmul__
mp_length	*lenfunc*	__len__
mp_subscript	*binaryfunc*	__getitem__
mp_ass_subscript	*objobjargproc*	__setitem__, __delitem__
sq_length	*lenfunc*	__len__
sq_concat	*binaryfunc*	__add__
sq_repeat	*ssizeargfunc*	__mul__
sq_item	*ssizeargfunc*	__getitem__
sq_ass_item	*ssizeobjargproc*	__setitem__ __delitem__
sq_contains	*objobjproc*	__contains__
sq_inplace_concat	*binaryfunc*	__iadd__
sq_inplace_repeat	*ssizeargfunc*	__imul__
bf_getbuffer	*getbufferproc()*	
bf_releasebuffer	*releasebufferproc()*	

슬롯 typedef

typedef	매개 변수 형	반환형
allocfunc	*PyTypeObject* * *Py_ssize_t*	*PyObject* *
destructor	*PyObject* *	void
freefunc	void *	void
traverseproc	*PyObject* * *visitproc* void *	int
newfunc	*PyTypeObject* * *PyObject* * *PyObject* *	*PyObject* *
initproc	*PyObject* * *PyObject* * *PyObject* *	int
reprfunc	*PyObject* *	*PyObject* *
getattrfunc	*PyObject* * const char *	*PyObject* *
setattrfunc	*PyObject* * const char * *PyObject* *	int
getattrofunc	*PyObject* * *PyObject* *	*PyObject* *
setattrofunc	*PyObject* * *PyObject* * *PyObject* *	int
descrgetfunc	*PyObject* * *PyObject* * *PyObject* *	*PyObject* *
descrsetfunc	*PyObject* * *PyObject* * *PyObject* *	int
hashfunc	*PyObject* *	Py_hash_t
richcmpfunc		*PyObject* *

자세한 내용은 아래 슬롯 형 *typedef*를 참조하십시오.

12.3.2 PyTypeObject 정의

The structure definition for *PyTypeObject* can be found in `Include/cpython/object.h`. For convenience of reference, this repeats the definition found there:

```
typedef struct _typeobject {
    PyObject_VAR_HEAD
    const char *tp_name; /* For printing, in format "<module>.<name>" */
    Py_ssize_t tp_basicsize, tp_itemsize; /* For allocation */

    /* Methods to implement standard operations */

    destructor tp_dealloc;
    Py_ssize_t tp_vectorcall_offset;
    getattrfunc tp_getattr;
    setattrfunc tp_setattr;
    PyAsyncMethods *tp_as_async; /* formerly known as tp_compare (Python 2)
                                    or tp_reserved (Python 3) */
    reprfunc tp_repr;

    /* Method suites for standard classes */

    PyNumberMethods *tp_as_number;
    PySequenceMethods *tp_as_sequence;
    PyMappingMethods *tp_as_mapping;

    /* More standard operations (here for binary compatibility) */

    hashfunc tp_hash;
    ternaryfunc tp_call;
    reprfunc tp_str;
    getattrofunc tp_getattro;
    setattrofunc tp_setattro;

    /* Functions to access object as input/output buffer */
    PyBufferProcs *tp_as_buffer;

    /* Flags to define presence of optional/expanded features */
    unsigned long tp_flags;

    const char *tp_doc; /* Documentation string */

    /* Assigned meaning in release 2.0 */
    /* call function for all accessible objects */
    traverseproc tp_traverse;

    /* delete references to contained objects */
    inquiry tp_clear;

    /* Assigned meaning in release 2.1 */
    /* rich comparisons */
    richcmpfunc tp_richcompare;

    /* weak reference enabler */
    Py_ssize_t tp_weaklistoffset;
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    /* Iterators */
    getiterfunc tp_iter;
    iternextfunc tp_iternext;

    /* Attribute descriptor and subclassing stuff */
    struct PyMethodDef *tp_methods;
    struct PyMemberDef *tp_members;
    struct PyGetSetDef *tp_getset;
    // Strong reference on a heap type, borrowed reference on a static type
    struct _typeobject *tp_base;
    PyObject *tp_dict;
    descrgetfunc tp_descr_get;
    descrsetfunc tp_descr_set;
    Py_ssize_t tp_dictoffset;
    initproc tp_init;
    allocfunc tp_alloc;
    newfunc tp_new;
    freefunc tp_free; /* Low-level free-memory routine */
    inquiry tp_is_gc; /* For PyObject_IS_GC */
    PyObject *tp_bases;
    PyObject *tp_mro; /* method resolution order */
    PyObject *tp_cache;
    PyObject *tp_subclasses;
    PyObject *tp_weaklist;
    destructor tp_del;

    /* Type attribute cache version tag. Added in version 2.6 */
    unsigned int tp_version_tag;

    destructor tp_finalize;
    vectorcallfunc tp_vectorcall;

    /* bitset of which type-watchers care about this type */
    unsigned char tp_watched;
} PyTypeObject;
```

12.3.3 PyObject 슬롯

The type object structure extends the *PyVarObject* structure. The *ob_size* field is used for dynamic types (created by `type_new()`, usually called from a class statement). Note that *PyType_Type* (the metatype) initializes *tp_itemsize*, which means that its instances (i.e. type objects) *must* have the *ob_size* field.

Py_ssize_t *PyObject*.**ob_refcnt**

Part of the Stable ABI. This is the type object's reference count, initialized to `1` by the `PyObject_HEAD_INIT` macro. Note that for *statically allocated type objects*, the type's instances (objects whose *ob_type* points back to the type) do *not* count as references. But for *dynamically allocated type objects*, the instances *do* count as references.

계승:

이 필드는 서브 형에 의해 상속되지 않습니다.

PyTypeObject **PyObject*.**ob_type**

Part of the Stable ABI. 이것은 형의 형, 즉 메타 형(metatype)입니다. `PyObject_HEAD_INIT` 매크로에 대한 인자로 초기화되며, 값은 일반적으로 `&PyType_Type`이어야 합니다. 그러나, (적어도) 윈도우에서 사용 가능해야 하는 동적으로 로드 가능한 확장 모듈의 경우, 컴파일러는 유효한 초기화자가 아니라고 불평합니다. 따라서, 규칙은 `NULL`을 `PyObject_HEAD_INIT` 매크로로 전달하고, 다른 작업

을 수행하기 전에 모듈의 초기화 함수 시작에서 필드를 명시적으로 초기화하는 것입니다. 이것은 일반적으로 다음과 같이 수행됩니다:

```
Foo_Type.ob_type = &PyType_Type;
```

This should be done before any instances of the type are created. *PyType_Ready()* checks if *ob_type* is NULL, and if so, initializes it to the *ob_type* field of the base class. *PyType_Ready()* will not change this field if it is non-zero.

계승:

이 필드는 서브 형으로 상속됩니다.

PyObject **PyObject._ob_next*

PyObject **PyObject._ob_prev*

These fields are only present when the macro `Py_TRACE_REFS` is defined (see the `configure --with-trace-refs option`).

Their initialization to NULL is taken care of by the `PyObject_HEAD_INIT` macro. For *statically allocated objects*, these fields always remain NULL. For *dynamically allocated objects*, these two fields are used to link the object into a doubly linked list of *all* live objects on the heap.

This could be used for various debugging purposes; currently the only uses are the `sys.getobjects()` function and to print the objects that are still alive at the end of a run when the environment variable `PYTHONDUMPREFS` is set.

계승:

이 필드는 서브 형에 의해 상속되지 않습니다.

12.3.4 PyVarObject 슬롯

Py_ssize_t *PyVarObject*.**ob_size**

Part of the Stable ABI. For *statically allocated type objects*, this should be initialized to zero. For *dynamically allocated type objects*, this field has a special internal meaning.

This field should be accessed using the *Py_SIZE()* and *Py_SET_SIZE()* macros.

계승:

이 필드는 서브 형에 의해 상속되지 않습니다.

12.3.5 PyTypeObject 슬롯

Each slot has a section describing inheritance. If *PyType_Ready()* may set a value when the field is set to NULL then there will also be a "Default" section. (Note that many fields set on *PyBaseObject_Type* and *PyType_Type* effectively act as defaults.)

const char **PyTypeObject*.**tp_name**

Pointer to a NUL-terminated string containing the name of the type. For types that are accessible as module globals, the string should be the full module name, followed by a dot, followed by the type name; for built-in types, it should be just the type name. If the module is a submodule of a package, the full package name is part of the full module name. For example, a type named `T` defined in module `M` in subpackage `Q` in package `P` should have the *tp_name* initializer `"P.Q.M.T"`.

For *dynamically allocated type objects*, this should just be the type name, and the module name explicitly stored in the type dict as the value for key `'__module__'`.

For *statically allocated type objects*, the *tp_name* field should contain a dot. Everything before the last dot is made accessible as the `__module__` attribute, and everything after the last dot is made accessible as the `__name__` attribute.

If no dot is present, the entire *tp_name* field is made accessible as the `__name__` attribute, and the `__module__` attribute is undefined (unless explicitly set in the dictionary, as explained above). This means

your type will be impossible to pickle. Additionally, it will not be listed in module documentations created with pydoc.

이 필드는 NULL이 아니어야 합니다. *PyTypeObject()*에서 유일하게 필요한 필드입니다 (잠재적인 *tp_itemsize*를 제외하고).

계승:

이 필드는 서브 형에 의해 상속되지 않습니다.

Py_ssize_t *PyTypeObject*.**tp_basicsize**

Py_ssize_t *PyTypeObject*.**tp_itemsize**

이 필드를 사용하면 형 인스턴스의 크기를 바이트 단위로 계산할 수 있습니다.

There are two kinds of types: types with fixed-length instances have a zero `tp_itemsize` field, types with variable-length instances have a non-zero `tp_itemsize` field. For a type with fixed-length instances, all instances have the same size, given in `tp_basicsize`. (Exceptions to this rule can be made using *PyUnstable_Object_GC_NewWithExtraData()*.)

For a type with variable-length instances, the instances must have an *ob_size* field, and the instance size is `tp_basicsize` plus N times `tp_itemsize`, where N is the "length" of the object.

Functions like *PyObject_NewVar()* will take the value of N as an argument, and store in the instance' s *ob_size* field. Note that the *ob_size* field may later be used for other purposes. For example, `int` instances use the bits of *ob_size* in an implementation-defined way; the underlying storage and its size should be acessed using `PyLong_Export()`.

참고

The *ob_size* field should be accessed using the *Py_SIZE()* and *Py_SET_SIZE()* macros.

Also, the presence of an *ob_size* field in the instance layout doesn' t mean that the instance structure is variable-length. For example, the `list` type has fixed-length instances, yet those instances have a *ob_size* field. (As with `int`, avoid reading lists' `ob_size` directly. Call *PyList_Size()* instead.)

The `tp_basicsize` includes size needed for data of the type' s *tp_base*, plus any extra data needed by each instance.

The correct way to set `tp_basicsize` is to use the `sizeof` operator on the struct used to declare the instance layout. This struct must include the struct used to declare the base type. In other words, `tp_basicsize` must be greater than or equal to the base' s `tp_basicsize`.

Since every type is a subtype of `object`, this struct must include *PyObject* or *PyVarObject* (depending on whether *ob_size* should be included). These are usually defined by the macro *PyObject_HEAD* or *PyObject_VAR_HEAD*, respectively.

The basic size does not include the GC header size, as that header is not part of *PyObject_HEAD*.

For cases where struct used to declare the base type is unknown, see *PyType_Spec.basicsize* and *PyType_FromMetaclass()*.

Notes about alignment:

- `tp_basicsize` must be a multiple of `_Alignof(PyObject)`. When using `sizeof` on a `struct` that includes *PyObject_HEAD*, as recommended, the compiler ensures this. When not using a C `struct`, or when using compiler extensions like `__attribute__((packed))`, it is up to you.
- If the variable items require a particular alignment, `tp_basicsize` and `tp_itemsize` must each be a multiple of that alignment. For example, if a type' s variable part stores a `double`, it is your responsibility that both fields are a multiple of `_Alignof(double)`.

계승:

These fields are inherited separately by subtypes. (That is, if the field is set to zero, *PyType_Ready()* will copy the value from the base type, indicating that the instances do not need additional storage.)

If the base type has a non-zero *tp_itemsize*, it is generally not safe to set *tp_itemsize* to a different non-zero value in a subtype (though this depends on the implementation of the base type).

destructor *PyTypeObject*.**tp_dealloc**

인스턴스 파괴자(destructor) 함수에 대한 포인터. (싱글톤 None과 Ellipsis의 경우처럼) 형이 해당 인스턴스가 할당 해제되지 않도록 보장하지 않는 한, 이 함수를 정의해야 합니다. 함수 서명은 다음과 같습니다:

```
void tp_dealloc(PyObject *self);
```

The destructor function is called by the *Py_DECREF()* and *Py_XDECREF()* macros when the new reference count is zero. At this point, the instance is still in existence, but there are no references to it. The destructor function should free all references which the instance owns, free all memory buffers owned by the instance (using the freeing function corresponding to the allocation function used to allocate the buffer), and call the type' s *tp_free* function. If the type is not subtypable (doesn' t have the *Py_TPFLAGS_BASETYPE* flag bit set), it is permissible to call the object deallocator directly instead of via *tp_free*. The object deallocator should be the one used to allocate the instance; this is normally *PyObject_Del()* if the instance was allocated using *PyObject_New* or *PyObject_NewVar*, or *PyObject_GC_Del()* if the instance was allocated using *PyObject_GC_New* or *PyObject_GC_NewVar*.

If the type supports garbage collection (has the *Py_TPFLAGS_HAVE_GC* flag bit set), the destructor should call *PyObject_GC_UnTrack()* before clearing any member fields.

```
static void foo_dealloc(foo_object *self) {
    PyObject_GC_UnTrack(self);
    Py_CLEAR(self->ref);
    Py_TYPE(self)->tp_free((PyObject *)self);
}
```

Finally, if the type is heap allocated (*Py_TPFLAGS_HEAPTYPE*), the deallocator should release the owned reference to its type object (via *Py_DECREF()*) after calling the type deallocator. In order to avoid dangling pointers, the recommended way to achieve this is:

```
static void foo_dealloc(foo_object *self) {
    PyTypeObject *tp = Py_TYPE(self);
    // free references and buffers here
    tp->tp_free(self);
    Py_DECREF(tp);
}
```

경고

In a garbage collected Python, tp_dealloc may be called from any Python thread, not just the thread which created the object (if the object becomes part of a refcount cycle, that cycle might be collected by a garbage collection on any thread). This is not a problem for Python API calls, since the thread on which tp_dealloc is called will own the Global Interpreter Lock (GIL). However, if the object being destroyed in turn destroys objects from some other C or C++ library, care should be taken to ensure that destroying those objects on the thread which called tp_dealloc will not violate any assumptions of the library.

계승:

이 필드는 서브 형으로 상속됩니다.

Py_ssize_t *PyTypeObject*.**tp_vectorcall_offset**

간단한 *tp_call*의 더 효율적인 대안인 벡터콜(*vectorcall*) 프로토콜을 사용하여 객체를 호출하는 것을 구현하는 인스턴스별 함수에 대한 선택적 오프셋입니다.

This field is only used if the flag *Py_TPFLAGS_HAVE_VECTORCALL* is set. If so, this must be a positive integer containing the offset in the instance of a *vectorcallfunc* pointer.

The *vectorcallfunc* pointer may be NULL, in which case the instance behaves as if *Py_TPFLAGS_HAVE_VECTORCALL* was not set: calling the instance falls back to *tp_call*.

Py_TPFLAGS_HAVE_VECTORCALL을 설정하는 모든 클래스는 *tp_call*도 설정해야 하고, 해당 동작이 *vectorcallfunc* 함수와 일관되도록 만들어야 합니다. *tp_call*을 *PyVectorcall_Call()*로 설정하면 됩니다:

버전 3.8에서 변경: 버전 3.8 이전에는, 이 슬롯의 이름이 tp_print였습니다. 파이썬 2.x에서는, 파일로 인쇄하는 데 사용되었습니다. 파이썬 3.0에서 3.7까지는, 사용되지 않았습니다.

버전 3.12에서 변경: Before version 3.12, it was not recommended for *mutable heap types* to implement the vectorcall protocol. When a user sets __call__ in Python code, only *tp_call* is updated, likely making it inconsistent with the vectorcall function. Since 3.12, setting __call__ will disable vectorcall optimization by clearing the *Py_TPFLAGS_HAVE_VECTORCALL* flag.

계승:

This field is always inherited. However, the *Py_TPFLAGS_HAVE_VECTORCALL* flag is not always inherited. If it' s not set, then the subclass won' t use *vectorcall*, except when *PyVectorcall_Call()* is explicitly called.

getattrfunc *PyTypeObject*.**tp_getattr**

get-attribute-string 함수에 대한 선택적 포인터.

이 필드는 폐지되었습니다. 정의될 때, *tp_getattro* 함수와 같게 작동하지만, 어트리뷰트 이름을 제공하기 위해 파이썬 문자열 객체 대신 C 문자열을 받아들이는 함수를 가리켜야 합니다.

계승:

Group: *tp_getattr*, *tp_getattro*

이 필드는 *tp_getattro*와 함께 서브 형에 의해 상속됩니다: 서브 형은 서브 형의 *tp_getattr*과 *tp_getattro*가 모두 NULL일 때 베이스형에서 *tp_getattr*과 *tp_getattro*를 모두 상속합니다.

setattrfunc *PyTypeObject*.**tp_setattr**

어트리뷰트 설정과 삭제를 위한 함수에 대한 선택적 포인터.

이 필드는 폐지되었습니다. 정의될 때, *tp_setattro* 함수와 같게 작동하지만, 어트리뷰트 이름을 제공하기 위해 파이썬 문자열 객체 대신 C 문자열을 받아들이는 함수를 가리켜야 합니다.

계승:

Group: *tp_setattr*, *tp_setattro*

이 필드는 *tp_setattro*와 함께 서브 형에 의해 상속됩니다. 서브 형은 서브 형의 *tp_setattr*과 *tp_setattro*가 모두 NULL일 때 베이스형에서 *tp_setattr*과 *tp_setattro*를 모두 상속합니다.

PyAsyncMethods **PyTypeObject*.**tp_as_async**

C 수준에서 어웨이터블과 비동기 이터레이터 프로토콜을 구현하는 객체에만 관련된 필드를 포함하는 추가 구조체에 대한 포인터. 자세한 내용은 비동기 객체 구조체를 참조하십시오.

Added in version 3.5: 이전에는 tp_compare와 tp_reserved라고 했습니다.

계승:

tp_as_async 필드는 상속되지 않지만, 포함된 필드는 개별적으로 상속됩니다.

reprfunc *PyTypeObject*.**tp_repr**

내장 함수 repr()을 구현하는 함수에 대한 선택적 포인터.

서명은 *PyObject_Repr()*과 같습니다:

```
PyObject *tp_repr(PyObject *self);
```

함수는 문자열이나 유니코드 객체를 반환해야 합니다. 이상적으로, 이 함수는 eval()에 전달될 때 적합한 환경이 주어지면 같은 값을 가진 객체를 반환하는 문자열을 반환해야 합니다. 이것이 가능하지 않으면, '<'로 시작하고 '>'로 끝나는 문자열을 반환해야 하는데, 이 문자열에서 객체의 형과 값을 모두 추론할 수 있어야 합니다.

계승:

이 필드는 서브 형으로 상속됩니다.

기본값:

이 필드를 설정하지 않으면, `<%s object at %p>` 형식의 문자열이 반환됩니다. 여기서 `%s`는 형 이름으로, `%p`는 객체의 메모리 주소로 치환됩니다.

PyNumberMethods ******PyTypeObject*`.tp_as_number`

숫자 프로토콜을 구현하는 객체에만 관련된 필드를 포함하는 추가 구조체에 대한 포인터. 이 필드는 숫자 객체 구조체에서 설명합니다.

계승:

tp_as_number 필드는 상속되지 않지만, 포함된 필드는 개별적으로 상속됩니다.

PySequenceMethods ******PyTypeObject*`.tp_as_sequence`

시퀀스 프로토콜을 구현하는 객체에만 관련된 필드를 포함하는 추가 구조체에 대한 포인터. 이 필드는 시퀀스 객체 구조체에서 설명합니다.

계승:

tp_as_sequence 필드는 상속되지 않지만, 포함된 필드는 개별적으로 상속됩니다.

PyMappingMethods ******PyTypeObject*`.tp_as_mapping`

매핑 프로토콜을 구현하는 객체에만 관련된 필드를 포함하는 추가 구조체에 대한 포인터. 이 필드는 매핑 객체 구조체에서 설명합니다.

계승:

tp_as_mapping 필드는 상속되지 않지만, 포함된 필드는 개별적으로 상속됩니다.

hashfunc *PyTypeObject*`.tp_hash`

내장 함수 `hash()`를 구현하는 함수에 대한 선택적 포인터.

서명은 *PyObject_Hash()*와 같습니다:

```
Py_hash_t tp_hash(PyObject *);
```

`-1` 값은 정상적인 반환 값으로 반환되지 않아야 합니다; 해시값을 계산하는 동안 에러가 발생하면 함수는 예외를 설정하고 `-1`을 반환해야 합니다.

When this field is not set (*and* *tp_richcompare* is not set), an attempt to take the hash of the object raises `TypeError`. This is the same as setting it to *PyObject_HashNotImplemented()*.

이 필드는 부모 형에서 해시 메서드의 상속을 차단하기 위해 *PyObject_HashNotImplemented()*로 명시적으로 설정할 수 있습니다. 이것은 파이썬 수준에서의 `__hash__ = None`과 동등한 것으로 해석되어, `isinstance(o, collections.Hashable)`이 `False`를 올바르게 반환하게 합니다. 반대의 경우도 마찬가지입니다 - 파이썬 수준의 클래스에서 `__hash__ = None`을 설정하면 `tp_hash` 슬롯이 *PyObject_HashNotImplemented()*로 설정됩니다.

계승:

Group: *tp_hash*, *tp_richcompare*

이 필드는 *tp_richcompare*와 함께 서브 형에 의해 상속됩니다: 서브 형의 *tp_richcompare*와 *tp_hash*가 모두 `NULL`일 때, 서브 형은 *tp_richcompare*와 *tp_hash*를 모두 상속합니다.

ternaryfunc *PyTypeObject*`.tp_call`

객체 호출을 구현하는 함수에 대한 선택적 포인터. 객체가 콜러블이 아니면 `NULL`이어야 합니다. 서명은 *PyObject_Call()*과 같습니다:

```
PyObject *tp_call(PyObject *self, PyObject *args, PyObject *kwargs);
```

계승:

이 필드는 서브 형으로 상속됩니다.

reprfunc *PyTypeObject*.**tp_str**

내장 연산 `str()`을 구현하는 함수에 대한 선택적 포인터. (`str`는 이제 형이며, `str()`은 그 형의 생성자를 호출함에 유의하십시오. 이 생성자는 *PyObject_Str()*를 호출하여 실제 작업을 수행하고, *PyObject_Str()*은 이 처리기를 호출합니다.)

서명은 *PyObject_Str()*과 같습니다:

```
PyObject *tp_str(PyObject *self);
```

함수는 문자열이나 유니코드 객체를 반환해야 합니다. 다른 것 중에서도, `print()` 함수에 의해 사용될 표현이기 때문에, 객체의 "친숙한" 문자열 표현이어야 합니다.

계승:

이 필드는 서브 형으로 상속됩니다.

기본값:

이 필드를 설정하지 않으면, 문자열 표현을 반환하기 위해 *PyObject_Repr()*이 호출됩니다.

getattrofunc *PyTypeObject*.**tp_getattro**

어트리뷰트 읽기(get-attribute) 함수에 대한 선택적 포인터.

서명은 *PyObject_GetAttr()*과 같습니다:

```
PyObject *tp_getattro(PyObject *self, PyObject *attr);
```

일반적으로 이 필드를 *PyObject_GenericGetAttr()*로 설정하는 것이 편리합니다, 객체 어트리뷰트를 찾는 일반적인 방법을 구현합니다.

계승:

Group: *tp_getattr*, *tp_getattro*

이 필드는 *tp_getattr*과 함께 서브 형에 의해 상속됩니다: 서브 형의 *tp_getattr*과 *tp_getattro*가 모두 `NULL`일 때 서브 형은 베이스형에서 *tp_getattr*과 *tp_getattro*를 모두 상속합니다.

기본값:

PyBaseObject_Type uses *PyObject_GenericGetAttr()*.

setattrofunc *PyTypeObject*.**tp_setattro**

어트리뷰트 설정과 삭제를 위한 함수에 대한 선택적 포인터.

서명은 *PyObject_SetAttr()*과 같습니다:

```
int tp_setattro(PyObject *self, PyObject *attr, PyObject *value);
```

또한, *value*를 `NULL`로 설정하여 어트리뷰트를 삭제하는 것을 반드시 지원해야 합니다. 일반적으로 이 필드를 *PyObject_GenericSetAttr()*로 설정하는 것이 편리합니다, 객체 어트리뷰트를 설정하는 일반적인 방법을 구현합니다.

계승:

Group: *tp_setattr*, *tp_setattro*

이 필드는 *tp_setattr*과 함께 서브 형에 의해 상속됩니다: 서브 형의 *tp_setattr*과 *tp_setattro*가 모두 `NULL`일 때, 서브 형은 베이스형에서 *tp_setattr*과 *tp_setattro*를 모두 상속합니다.

기본값:

PyBaseObject_Type uses *PyObject_GenericSetAttr()*.

PyBufferProcs **PyTypeObject*.**tp_as_buffer**

버퍼 인터페이스를 구현하는 객체에만 관련된 필드를 포함하는 추가 구조체에 대한 포인터. 이 필드는 버퍼 객체 구조체에서 설명합니다.

계승:

tp_as_buffer 필드는 상속되지 않지만, 포함된 필드는 개별적으로 상속됩니다.

unsigned long *PyTypeObject*.**tp_flags**

이 필드는 다양한 플래그의 비트 마스크입니다. 일부 플래그는 특정 상황에 대한 변형 의미론을 나타냅니다; 다른 것들은 역사적으로 항상 존재하지는 않았던 형 객체(또는 *tp_as_number*, *tp_as_sequence*, *tp_as_mapping* 및 *tp_as_buffer*를 통해 참조되는 확장 구조체)의 특정 필드가 유효함을 나타내는 데 사용됩니다; 이러한 플래그 비트가 없으면, 이것이 보호하는 형 필드에 액세스하지 말아야 하며 대신 0이나 `NULL` 값을 갖는 것으로 간주해야 합니다.

계승:

Inheritance of this field is complicated. Most flag bits are inherited individually, i.e. if the base type has a flag bit set, the subtype inherits this flag bit. The flag bits that pertain to extension structures are strictly inherited if the extension structure is inherited, i.e. the base type' s value of the flag bit is copied into the subtype together with a pointer to the extension structure. The *Py_TPFLAGS_HAVE_GC* flag bit is inherited together with the *tp_traverse* and *tp_clear* fields, i.e. if the *Py_TPFLAGS_HAVE_GC* flag bit is clear in the subtype and the *tp_traverse* and *tp_clear* fields in the subtype exist and have `NULL` values. .. XXX are most flag bits *really* inherited individually?

기본값:

PyBaseObject_Type uses `Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE`.

비트 마스크:

다음 비트 마스크가 현재 정의되어 있습니다; 이들은 `|` 연산자로 함께 OR 하여 *tp_flags* 필드의 값을 형성할 수 있습니다. 매크로 *PyType_HasFeature()*는 형과 플래그 값 *tp*와 *f*를 취하고 `tp->tp_flags & f`가 0이 아닌지 확인합니다.

Py_TPFLAGS_HEAPTYPE

This bit is set when the type object itself is allocated on the heap, for example, types created dynamically using *PyType_FromSpec()*. In this case, the *ob_type* field of its instances is considered a reference to the type, and the type object is INCREF' ed when a new instance is created, and DECREF' ed when an instance is destroyed (this does not apply to instances of subtypes; only the type referenced by the instance' s ob_type gets INCREF' ed or DECREF' ed). Heap types should also *support garbage collection* as they can form a reference cycle with their own module object.

계승:

???

Py_TPFLAGS_BASETYPE

이 비트는 형을 다른 형의 베이스형으로 사용할 수 있을 때 설정됩니다. 이 비트가 설정되지 않으면 이 형으로 서브 형을 만들 수 없습니다 (Java의 "final" 클래스와 유사합니다).

계승:

???

Py_TPFLAGS_READY

이 비트는 *PyType_Ready()*에 의해 형 객체가 완전히 초기화될 때 설정됩니다.

계승:

???

Py_TPFLAGS_READYING

이 비트는 *PyType_Ready()*가 형 객체를 초기화하는 동안 설정됩니다.

계승:

???

Py_TPFLAGS_HAVE_GC

This bit is set when the object supports garbage collection. If this bit is set, instances must be created using *PyObject_GC_New* and destroyed using *PyObject_GC_Del()*. More information in section 순

환 가비지 수집 지원. This bit also implies that the GC-related fields *tp_traverse* and *tp_clear* are present in the type object.

계승:

Group: *Py_TPFLAGS_HAVE_GC*, *tp_traverse*, *tp_clear*

The *Py_TPFLAGS_HAVE_GC* flag bit is inherited together with the *tp_traverse* and *tp_clear* fields, i.e. if the *Py_TPFLAGS_HAVE_GC* flag bit is clear in the subtype and the *tp_traverse* and *tp_clear* fields in the subtype exist and have `NULL` values.

Py_TPFLAGS_DEFAULT

This is a bitmask of all the bits that pertain to the existence of certain fields in the type object and its extension structures. Currently, it includes the following bits: `Py_TPFLAGS_HAVE_STACKLESS_EXTENSION`.

계승:

???

Py_TPFLAGS_METHOD_DESCRIPTOR

이 비트는 객체가 연결되지 않은 메서드(unbound method)처럼 동작함을 나타냅니다.

이 플래그가 `type(meth)` 에 설정되면:

- `meth.__get__(obj, cls)(*args, **kwds)`(`obj` 가 None 이 아닐 때) 는 `meth(obj, *args, **kwds)` 와 동등해야 합니다.
- `meth.__get__(None, cls)(*args, **kwds)` 는 `meth(*args, **kwds)` 와 동등해야 합니다.

이 플래그는 `obj.meth()` 와 같은 일반적인 메서드 호출에 대한 최적화를 가능하게 합니다: `obj.meth` 에 대한 임시 "연결된 메서드(bound method)" 객체를 만들지 않습니다.

Added in version 3.8.

계승:

This flag is never inherited by types without the *Py_TPFLAGS_IMMUTABLETYPE* flag set. For extension types, it is inherited whenever *tp_descr_get* is inherited.

Py_TPFLAGS_MANAGED_DICT

This bit indicates that instances of the class have a *~object.__dict__* attribute, and that the space for the dictionary is managed by the VM.

If this flag is set, *Py_TPFLAGS_HAVE_GC* should also be set.

Added in version 3.12.

계승:

This flag is inherited unless the *tp_dictoffset* field is set in a superclass.

Py_TPFLAGS_MANAGED_WEAKREF

This bit indicates that instances of the class should be weakly referenceable.

Added in version 3.12.

계승:

This flag is inherited unless the *tp_weaklistoffset* field is set in a superclass.

Py_TPFLAGS_ITEMS_AT_END

Only usable with variable-size types, i.e. ones with non-zero *tp_itemsize*.

Indicates that the variable-sized portion of an instance of this type is at the end of the instance' s memory area, at an offset of `Py_TYPE(obj)->tp_basicsize` (which may be different in each subclass).

When setting this flag, be sure that all superclasses either use this memory layout, or are not variable-sized. Python does not check this.

Added in version 3.12.

계승:

This flag is inherited.

Py_TPFLAGS_LONG_SUBCLASS

Py_TPFLAGS_LIST_SUBCLASS

Py_TPFLAGS_TUPLE_SUBCLASS

Py_TPFLAGS_BYTES_SUBCLASS

Py_TPFLAGS_UNICODE_SUBCLASS

Py_TPFLAGS_DICT_SUBCLASS

Py_TPFLAGS_BASE_EXC_SUBCLASS

Py_TPFLAGS_TYPE_SUBCLASS

이 플래그는 *PyLong_Check()* 와 같은 함수에서 형이 내장형의 서브 클래스인지 신속하게 판별하는 데 사용됩니다; 이러한 특정 검사는 *PyObject_IsInstance()*와 같은 일반 검사보다 빠릅니다. 내장에서 상속된 사용자 정의 형은 *tp_flags*를 적절하게 설정해야 합니다, 그렇지 않으면 그러한 형과 상호 작용하는 코드가 사용되는 검사의 유형에 따라 다르게 작동합니다.

Py_TPFLAGS_HAVE_FINALIZE

이 비트는 *tp_finalize* 슬롯이 형 구조체에 있을 때 설정됩니다.

Added in version 3.4.

버전 3.8부터 폐지됨: 인터프리터는 *tp_finalize* 슬롯이 항상 형 구조체에 있다고 가정하기 때문에, 이 플래그는 더는 필요하지 않습니다.

Py_TPFLAGS_HAVE_VECTORCALL

이 비트는 클래스가 벡터콜 프로토콜을 구현할 때 설정됩니다. 자세한 내용은 *tp_vectorcall_offset*을 참조하십시오.

계승:

This bit is inherited if *tp_call* is also inherited.

Added in version 3.9.

버전 3.12에서 변경: This flag is now removed from a class when the class' s `__call__()` method is reassigned.

This flag can now be inherited by mutable classes.

Py_TPFLAGS_IMMUTABLETYPE

This bit is set for type objects that are immutable: type attributes cannot be set nor deleted.

PyType_Ready() automatically applies this flag to *static types*.

계승:

This flag is not inherited.

Added in version 3.10.

Py_TPFLAGS_DISALLOW_INSTANTIATION

Disallow creating instances of the type: set *tp_new* to NULL and don' t create the `__new__` key in the type dictionary.

The flag must be set before creating the type, not after. For example, it must be set before *PyType_Ready()* is called on the type.

The flag is set automatically on *static types* if *tp_base* is NULL or `&PyBaseObject_Type` and *tp_new* is NULL.

계승:

This flag is not inherited. However, subclasses will not be instantiable unless they provide a non-NULL *tp_new* (which is only possible via the C API).

참고

To disallow instantiating a class directly but allow instantiating its subclasses (e.g. for an *abstract base class*), do not use this flag. Instead, make *tp_new* only succeed for subclasses.

Added in version 3.10.

Py_TPFLAGS_MAPPING

This bit indicates that instances of the class may match mapping patterns when used as the subject of a `match` block. It is automatically set when registering or subclassing `collections.abc.Mapping`, and unset when registering `collections.abc.Sequence`.

참고

Py_TPFLAGS_MAPPING and *Py_TPFLAGS_SEQUENCE* are mutually exclusive; it is an error to enable both flags simultaneously.

계승:

This flag is inherited by types that do not already set *Py_TPFLAGS_SEQUENCE*.

더 보기

PEP 634 – Structural Pattern Matching: Specification

Added in version 3.10.

Py_TPFLAGS_SEQUENCE

This bit indicates that instances of the class may match sequence patterns when used as the subject of a `match` block. It is automatically set when registering or subclassing `collections.abc.Sequence`, and unset when registering `collections.abc.Mapping`.

참고

Py_TPFLAGS_MAPPING and *Py_TPFLAGS_SEQUENCE* are mutually exclusive; it is an error to enable both flags simultaneously.

계승:

This flag is inherited by types that do not already set *Py_TPFLAGS_MAPPING*.

더 보기

PEP 634 – Structural Pattern Matching: Specification

Added in version 3.10.

Py_TPFLAGS_VALID_VERSION_TAG

Internal. Do not set or unset this flag. To indicate that a class has changed call *PyType_Modified()*

> **경고**
>
> This flag is present in header files, but is an internal feature and should not be used. It will be removed in a future version of CPython

const char *PyTypeObject.**tp_doc**

An optional pointer to a NUL-terminated C string giving the docstring for this type object. This is exposed as the `__doc__` attribute on the type and instances of the type.

계승:

이 필드는 서브 형에 의해 상속되지 않습니다.

traverseproc PyTypeObject.**tp_traverse**

An optional pointer to a traversal function for the garbage collector. This is only used if the *Py_TPFLAGS_HAVE_GC* flag bit is set. The signature is:

```
int tp_traverse(PyObject *self, visitproc visit, void *arg);
```

파이썬의 가비지 수집 체계에 대한 자세한 정보는 섹션 순환 가비지 수집 지원에서 찾을 수 있습니다.

The *tp_traverse* pointer is used by the garbage collector to detect reference cycles. A typical implementation of a *tp_traverse* function simply calls *Py_VISIT()* on each of the instance's members that are Python objects that the instance owns. For example, this is function `local_traverse()` from the `_thread` extension module:

```
static int
local_traverse(localobject *self, visitproc visit, void *arg)
{
    Py_VISIT(self->args);
    Py_VISIT(self->kw);
    Py_VISIT(self->dict);
    return 0;
}
```

*Py_VISIT()*는 참조 순환에 참여할 수 있는 멤버에 대해서만 호출됨에 유의하십시오. `self->key` 멤버도 있지만, `NULL`이나 파이썬 문자열만 가능해서 참조 순환의 일부가 될 수 없습니다.

반면에, 멤버가 사이클의 일부가 될 수 없다는 것을 알고 있더라도, 디버깅 지원을 위해 `gc` 모듈의 `get_referents()` 함수가 그것을 포함하도록 어쨌거나 방문하고 싶을 수 있습니다.

> **경고**
>
> When implementing *tp_traverse*, only the members that the instance *owns* (by having *strong references* to them) must be visited. For instance, if an object supports weak references via the *tp_weaklist* slot, the pointer supporting the linked list (what *tp_weaklist* points to) must **not** be visited as the instance does not directly own the weak references to itself (the weakreference list is there to support the weak reference machinery, but the instance has no strong reference to the elements inside it, as they are allowed to be removed even if the instance is still alive).

*Py_VISIT()*는 `local_traverse()`의 *visit*와 *arg* 매개 변수가 이 이름일 것을 요구합니다; 다른 이름을 붙이지 마십시오.

Instances of *heap-allocated types* hold a reference to their type. Their traversal function must therefore either visit *Py_TYPE(self)*, or delegate this responsibility by calling `tp_traverse` of another heap-allocated type (such as a heap-allocated superclass). If they do not, the type object may not be garbage-collected.

버전 3.9에서 변경: 힙 할당 형은 `tp_traverse`에서 `Py_TYPE(self)`를 방문할 것으로 기대됩니다. 이전 버전의 파이썬에서는, 버그 40217로 인해, 이렇게 하면 서브 클래스에서 충돌이 발생할 수 있습니다.

계승:

Group: *Py_TPFLAGS_HAVE_GC*, *tp_traverse*, *tp_clear*

This field is inherited by subtypes together with *tp_clear* and the *Py_TPFLAGS_HAVE_GC* flag bit: the flag bit, *tp_traverse*, and *tp_clear* are all inherited from the base type if they are all zero in the subtype.

inquiry *PyTypeObject*.**tp_clear**

An optional pointer to a clear function for the garbage collector. This is only used if the *Py_TPFLAGS_HAVE_GC* flag bit is set. The signature is:

```
int tp_clear(PyObject *);
```

tp_clear 멤버 함수는 가비지 수집기에서 감지한 순환 가비지에서 참조 순환을 끊는 데 사용됩니다. 종합하여, 시스템의 모든 *tp_clear* 함수가 결합하여 모든 참조 순환을 끊어야 합니다. 이것은 미묘합니다, 확신이 서지 않으면 *tp_clear* 함수를 제공하십시오. 예를 들어, 튜플 형은 *tp_clear* 함수를 구현하지 않습니다. 튜플만으로는 참조 순환이 구성될 수 없음을 증명할 수 있기 때문입니다. 따라서 다른 형의 *tp_clear* 함수만으로 튜플을 포함하는 순환을 끊기에 충분해야 합니다. 이것은 그리 자명하지 않으며, *tp_clear*를 구현하지 않아도 좋을 만한 이유는 거의 없습니다.

*tp_clear*의 구현은 다음 예제와 같이 파이썬 객체일 수 있는 자신의 멤버에 대한 인스턴스의 참조를 삭제하고 해당 멤버에 대한 포인터를 `NULL`로 설정해야 합니다:

```
static int
local_clear(localobject *self)
{
    Py_CLEAR(self->key);
    Py_CLEAR(self->args);
    Py_CLEAR(self->kw);
    Py_CLEAR(self->dict);
    return 0;
}
```

The *Py_CLEAR()* macro should be used, because clearing references is delicate: the reference to the contained object must not be released (via *Py_DECREF()*) until after the pointer to the contained object is set to `NULL`. This is because releasing the reference may cause the contained object to become trash, triggering a chain of reclamation activity that may include invoking arbitrary Python code (due to finalizers, or weakref callbacks, associated with the contained object). If it' s possible for such code to reference *self* again, it' s important that the pointer to the contained object be `NULL` at that time, so that *self* knows the contained object can no longer be used. The *Py_CLEAR()* macro performs the operations in a safe order.

Note that *tp_clear* is not *always* called before an instance is deallocated. For example, when reference counting is enough to determine that an object is no longer used, the cyclic garbage collector is not involved and *tp_dealloc* is called directly.

tp_clear 함수의 목표는 참조 순환을 끊는 것이기 때문에, 참조 순환에 참여할 수 없는 파이썬 문자열이나 파이썬 정수와 같은 포함된 객체를 정리할 필요는 없습니다. 반면에, 포함된 모든 파이썬 객체를 정리하고, 형의 *tp_dealloc* 함수가 *tp_clear*를 호출하도록 작성하는 것이 편리할 수 있습니다.

파이썬의 가비지 수집 체계에 대한 자세한 정보는 섹션 순환 가비지 수집 지원에서 찾을 수 있습니다.

계승:

Group: *Py_TPFLAGS_HAVE_GC*, *tp_traverse*, *tp_clear*

This field is inherited by subtypes together with *tp_traverse* and the *Py_TPFLAGS_HAVE_GC* flag bit: the flag bit, *tp_traverse*, and *tp_clear* are all inherited from the base type if they are all zero in the subtype.

richcmpfunc *PyTypeObject*.**tp_richcompare**

풍부한 비교 함수(rich comparison function)에 대한 선택적 포인터. 서명은 다음과 같습니다:

```
PyObject *tp_richcompare(PyObject *self, PyObject *other, int op);
```

첫 번째 매개 변수는 *PyTypeObject*에 의해 정의된 형의 인스턴스임이 보장됩니다.

이 함수는 비교 결과(일반적으로 `Py_True`나 `Py_False`)를 반환해야 합니다. 비교가 정의되어 있지 않으면, `Py_NotImplemented`를 반환하고, 다른 에러가 발생하면 `NULL`을 반환하고 예외 조건을 설정해야 합니다.

다음 상수는 *tp_richcompare*와 *PyObject_RichCompare()*의 세 번째 인자로 사용되도록 정의됩니다:

상수	비교
Py_LT	<
Py_LE	<=
Py_EQ	==
Py_NE	!=
Py_GT	>
Py_GE	>=

풍부한 비교 함수를 쉽게 작성할 수 있도록 다음 매크로가 정의됩니다:

Py_RETURN_RICHCOMPARE(VAL_A, VAL_B, op)

비교 결과에 따라, 함수에서 `Py_True`나 `Py_False`를 반환합니다. VAL_A와 VAL_B는 C 비교 연산자로 순서를 정할 수 있어야 합니다 (예를 들어, C int나 float일 수 있습니다). 세 번째 인자는 *PyObject_RichCompare()*에서처럼 요청된 연산을 지정합니다.

The returned value is a new *strong reference*.

에러가 발생하면, 예외를 설정하고 함수에서 `NULL`을 반환합니다.

Added in version 3.7.

계승:

Group: *tp_hash*, *tp_richcompare*

이 필드는 *tp_hash*와 함께 서브 형에 의해 상속됩니다. 서브 형의 *tp_richcompare*와 *tp_hash*가 모두 `NULL`이면 서브 형은 *tp_richcompare*와 *tp_hash*를 상속합니다.

기본값:

PyBaseObject_Type provides a *tp_richcompare* implementation, which may be inherited. However, if only *tp_hash* is defined, not even the inherited function is used and instances of the type will not be able to participate in any comparisons.

Py_ssize_t *PyTypeObject*.**tp_weaklistoffset**

While this field is still supported, *Py_TPFLAGS_MANAGED_WEAKREF* should be used instead, if at all possible.

If the instances of this type are weakly referenceable, this field is greater than zero and contains the offset in the instance structure of the weak reference list head (ignoring the GC header, if present); this offset is used by *PyObject_ClearWeakRefs()* and the `PyWeakref_*` functions. The instance structure needs to include a field of type *PyObject** which is initialized to `NULL`.

이 필드를 *tp_weaklist*와 혼동하지 마십시오; 그것은 형 객체 자체에 대한 약한 참조의 리스트 헤드입니다.

It is an error to set both the *Py_TPFLAGS_MANAGED_WEAKREF* bit and *tp_weaklistoffset*.

계승:

이 필드는 서브 형에 의해 상속되지만, 아래 나열된 규칙을 참조하십시오. 서브 형이 이 오프셋을 재정의할 수 있습니다; 이는 서브 형이 베이스형과 다른 약한 참조 리스트 헤드를 사용함을 의미합니다. 리스트 헤드는 항상 *tp_weaklistoffset*을 통해 발견되므로, 문제가 되지 않습니다.

기본값:

If the *Py_TPFLAGS_MANAGED_WEAKREF* bit is set in the *tp_flags* field, then *tp_weaklistoffset* will be set to a negative value, to indicate that it is unsafe to use this field.

getiterfunc *PyTypeObject*.**tp_iter**

An optional pointer to a function that returns an *iterator* for the object. Its presence normally signals that the instances of this type are *iterable* (although sequences may be iterable without this function).

이 함수는 *PyObject_GetIter()*와 같은 서명을 갖습니다:

```
PyObject *tp_iter(PyObject *self);
```

계승:

이 필드는 서브 형으로 상속됩니다.

iternextfunc *PyTypeObject*.**tp_iternext**

An optional pointer to a function that returns the next item in an *iterator*. The signature is:

```
PyObject *tp_iternext(PyObject *self);
```

이터레이터가 소진되면 `NULL`을 반환해야 합니다; `StopIteration` 예외가 설정될 수도, 그렇지 않을 수도 있습니다. 다른 에러가 발생하면, 역시 `NULL`을 반환해야 합니다. 그 존재는 이 형의 인스턴스가 이터레이터라는 신호입니다.

이터레이터 형은 *tp_iter* 함수도 정의해야 하며, 해당 함수는 (새 이터레이터 인스턴스가 아닌) 이터레이터 인스턴스 자체를 반환해야 합니다.

이 함수는 *PyIter_Next()*와 같은 서명을 갖습니다.

계승:

이 필드는 서브 형으로 상속됩니다.

struct *PyMethodDef* **PyTypeObject*.**tp_methods**

이 형의 일반 메서드를 선언하는 *PyMethodDef* 구조체의 정적 `NULL`-종료 배열에 대한 선택적 포인터.

배열의 항목마다, 메서드 디스크립터를 포함하는 형의 딕셔너리(아래 *tp_dict*를 참조하십시오)에 항목이 추가됩니다.

계승:

이 필드는 서브 형에 의해 상속되지 않습니다 (메서드는 다른 메커니즘을 통해 상속됩니다).

struct *PyMemberDef* **PyTypeObject*.**tp_members**

이 형의 인스턴스의 일반 데이터 멤버(필드나 슬롯)를 선언하는 *PyMemberDef* 구조체의 정적 `NULL`-종료 배열에 대한 선택적 포인터.

배열의 항목마다, 멤버 디스크립터를 포함하는 형의 딕셔너리(아래 *tp_dict*를 참조하십시오)에 항목이 추가됩니다.

계승:

이 필드는 서브 형에 의해 상속되지 않습니다 (멤버는 다른 메커니즘을 통해 상속됩니다).

struct *PyGetSetDef* **PyTypeObject.tp_getset*

이 형의 인스턴스의 계산된 어트리뷰트를 선언하는 *PyGetSetDef* 구조체의 정적 NULL-종료 배열에 대한 선택적 포인터.

배열의 항목마다, getset 디스크립터를 포함하는 형의 딕셔너리(아래 *tp_dict*를 참조하십시오)에 항목이 추가됩니다.

계승:

이 필드는 서브 형에 의해 상속되지 않습니다(계산된 어트리뷰트는 다른 메커니즘을 통해 상속됩니다).

PyTypeObject **PyTypeObject.tp_base**

형 속성이 상속되는 베이스형에 대한 선택적 포인터. 이 수준에서는, 단일 상속만 지원됩니다; 다중 상속은 메타 형을 호출하여 형 객체를 동적으로 작성해야 합니다.

> **참고**
>
> 슬롯 초기화에는 전역 초기화 규칙이 적용됩니다. C99에서는 초기화자가 "주소 상수(address constants)"여야 합니다. 포인터로 묵시적으로 변환되는 *PyType_GenericNew()*와 같은 함수 지정자는 유효한 C99 주소 상수입니다.
>
> However, the unary '&' operator applied to a non-static variable like *PyBaseObject_Type* is not required to produce an address constant. Compilers may support this (gcc does), MSVC does not. Both compilers are strictly standard conforming in this particular behavior.
>
> 결과적으로, *tp_base*는 확장 모듈의 초기화 함수에서 설정되어야 합니다.

계승:

이 필드는 서브 형에 의해 상속되지 않습니다(명백히).

기본값:

이 필드의 기본값은 `&PyBaseObject_Type`입니다(파이썬 프로그래머에게는 `object` 형으로 알려져 있습니다).

PyObject **PyTypeObject.tp_dict**

형의 딕셔너리는 *PyType_Ready()*에 의해 여기에 저장됩니다.

This field should normally be initialized to NULL before PyType_Ready is called; it may also be initialized to a dictionary containing initial attributes for the type. Once *PyType_Ready()* has initialized the type, extra attributes for the type may be added to this dictionary only if they don' t correspond to overloaded operations (like `__add__()`). Once initialization for the type has finished, this field should be treated as read-only.

Some types may not store their dictionary in this slot. Use *PyType_GetDict()* to retrieve the dictionary for an arbitrary type.

버전 3.12에서 변경: Internals detail: For static builtin types, this is always NULL. Instead, the dict for such types is stored on `PyInterpreterState`. Use *PyType_GetDict()* to get the dict for an arbitrary type.

계승:

이 필드는 서브 형에 의해 상속되지 않습니다(여기에 정의된 어트리뷰트는 다른 메커니즘을 통해 상속됩니다).

기본값:

이 필드가 NULL이면, *PyType_Ready()*는 새 딕셔너리를 할당합니다.

> **경고**
>
> *PyDict_SetItem()*을 사용하거나 다른 식으로 딕셔너리 C-API로 *tp_dict*를 수정하는 것은 안전하지 않습니다.

descrgetfunc *PyTypeObject*.**tp_descr_get**

"디스크립터 get" 함수에 대한 선택적 포인터.

함수 서명은 다음과 같습니다:

```
PyObject * tp_descr_get(PyObject *self, PyObject *obj, PyObject *type);
```

계승:

이 필드는 서브 형으로 상속됩니다.

descrsetfunc *PyTypeObject*.**tp_descr_set**

디스크립터 값을 설정하고 삭제하기 위한 함수에 대한 선택적 포인터.

함수 서명은 다음과 같습니다:

```
int tp_descr_set(PyObject *self, PyObject *obj, PyObject *value);
```

value 인자는 값을 삭제하기 위해 `NULL`로 설정됩니다.

계승:

이 필드는 서브 형으로 상속됩니다.

Py_ssize_t *PyTypeObject*.**tp_dictoffset**

While this field is still supported, *Py_TPFLAGS_MANAGED_DICT* should be used instead, if at all possible.

이 형의 인스턴스에 인스턴스 변수를 포함하는 딕셔너리가 있으면, 이 필드는 0이 아니며 인스턴스 변수 딕셔너리 형의 인스턴스에서의 오프셋을 포함합니다; 이 오프셋은 *PyObject_GenericGetAttr()*에서 사용됩니다.

이 필드를 *tp_dict*와 혼동하지 마십시오; 그것은 형 객체 자체의 어트리뷰트에 대한 딕셔너리입니다.

The value specifies the offset of the dictionary from the start of the instance structure.

The *tp_dictoffset* should be regarded as write-only. To get the pointer to the dictionary call *PyObject_GenericGetDict()*. Calling *PyObject_GenericGetDict()* may need to allocate memory for the dictionary, so it is may be more efficient to call *PyObject_GetAttr()* when accessing an attribute on the object.

It is an error to set both the *Py_TPFLAGS_MANAGED_DICT* bit and *tp_dictoffset*.

계승:

This field is inherited by subtypes. A subtype should not override this offset; doing so could be unsafe, if C code tries to access the dictionary at the previous offset. To properly support inheritance, use *Py_TPFLAGS_MANAGED_DICT*.

기본값:

This slot has no default. For *static types*, if the field is `NULL` then no `__dict__` gets created for instances.

If the *Py_TPFLAGS_MANAGED_DICT* bit is set in the *tp_dict* field, then *tp_dictoffset* will be set to `-1`, to indicate that it is unsafe to use this field.

initproc *PyTypeObject*.**tp_init**

인스턴스 초기화 함수에 대한 선택적 포인터.

This function corresponds to the `__init__()` method of classes. Like `__init__()`, it is possible to create an instance without calling `__init__()`, and it is possible to reinitialize an instance by calling its `__init__()` method again.

함수 서명은 다음과 같습니다:

```
int tp_init(PyObject *self, PyObject *args, PyObject *kwds);
```

The self argument is the instance to be initialized; the *args* and *kwds* arguments represent positional and keyword arguments of the call to `__init__()`.

`NULL`이 아닐 때, *tp_init* 함수는 형을 호출하여 인스턴스를 정상적으로 만들 때, 형의 *tp_new* 함수가 형의 인스턴스를 반환한 후 호출됩니다. *tp_new* 함수가 원래 형의 서브 형이 아닌 다른 형의 인스턴스를 반환하면, 아무런 *tp_init* 함수도 호출되지 않습니다; *tp_new*가 원래 형의 서브 형 인스턴스를 반환하면, 서브 형의 *tp_init*가 호출됩니다.

성공하면 `0`을 반환하고, 에러 시에는 `-1`을 반환하고 예외를 설정합니다.

계승:

이 필드는 서브 형으로 상속됩니다.

기본값:

For *static types* this field does not have a default.

allocfunc *PyTypeObject*.**tp_alloc**

인스턴스 할당 함수에 대한 선택적 포인터.

함수 서명은 다음과 같습니다:

```
PyObject *tp_alloc(PyTypeObject *self, Py_ssize_t nitems);
```

계승:

이 필드는 정적 서브 형에 의해 상속되지만, 동적 서브 형(클래스 문으로 만들어진 서브 형)에는 상속되지 않습니다.

기본값:

동적 서브 형의 경우, 이 필드는 표준 힙 할당 전략을 강제하기 위해 항상 *PyType_GenericAlloc()*으로 설정됩니다.

For static subtypes, *PyBaseObject_Type* uses *PyType_GenericAlloc()*. That is the recommended value for all statically defined types.

newfunc *PyTypeObject*.**tp_new**

인스턴스 생성 함수에 대한 선택적 포인터.

함수 서명은 다음과 같습니다:

```
PyObject *tp_new(PyTypeObject *subtype, PyObject *args, PyObject *kwds);
```

subtype 인자는 만들어지고 있는 객체의 형입니다; *args*와 *kwds* 인자는 형 호출의 위치와 키워드 인자를 나타냅니다. *subtype*이 *tp_new* 함수가 호출되는 형과 같을 필요는 없음에 유의하십시오; 이 형의 서브 형일 수 있습니다 (하지만 관련이 없는 형은 아닙니다).

tp_new 함수는 객체에 공간을 할당하기 위해 `subtype->tp_alloc(subtype, nitems)`를 호출해야 하고, 그런 다음 꼭 필요한 만큼만 추가 초기화를 수행해야 합니다. 안전하게 무시하거나 반복할 수 있는 초기화는 *tp_init* 처리기에 배치해야 합니다. 간단한 규칙은, 불변 형의 경우 모든 초기화가 *tp_new*에서 수행되어야 하고, 가변형의 경우 대부분 초기화는 *tp_init*로 미뤄져야 합니다.

Set the *Py_TPFLAGS_DISALLOW_INSTANTIATION* flag to disallow creating instances of the type in Python.

계승:

This field is inherited by subtypes, except it is not inherited by *static types* whose *tp_base* is `NULL` or `&PyBaseObject_Type`.

기본값:

For *static types* this field has no default. This means if the slot is defined as `NULL`, the type cannot be called to create new instances; presumably there is some other way to create instances, like a factory function.

freefunc *PyTypeObject*.**tp_free**

인스턴스 할당 해제 함수에 대한 선택적 포인터. 서명은 다음과 같습니다:

```
void tp_free(void *self);
```

이 서명과 호환되는 초기화자는 *PyObject_Free()* 입니다.

계승:

이 필드는 정적 서브 형에 의해 상속되지만, 동적 서브 형(클래스 문으로 만들어진 서브 형)에는 상속되지 않습니다.

기본값:

In dynamic subtypes, this field is set to a deallocator suitable to match *PyType_GenericAlloc()* and the value of the *Py_TPFLAGS_HAVE_GC* flag bit.

For static subtypes, *PyBaseObject_Type* uses *PyObject_Del()*.

inquiry *PyTypeObject*.**tp_is_gc**

가비지 수집기에서 호출되는 함수에 대한 선택적 포인터.

The garbage collector needs to know whether a particular object is collectible or not. Normally, it is sufficient to look at the object' s type' s *tp_flags* field, and check the *Py_TPFLAGS_HAVE_GC* flag bit. But some types have a mixture of statically and dynamically allocated instances, and the statically allocated instances are not collectible. Such types should define this function; it should return `1` for a collectible instance, and `0` for a non-collectible instance. The signature is:

```
int tp_is_gc(PyObject *self);
```

(The only example of this are types themselves. The metatype, *PyType_Type*, defines this function to distinguish between statically and *dynamically allocated types*.)

계승:

이 필드는 서브 형으로 상속됩니다.

기본값:

This slot has no default. If this field is `NULL`, *Py_TPFLAGS_HAVE_GC* is used as the functional equivalent.

PyObject **PyTypeObject*.**tp_bases**

베이스형의 튜플.

This field should be set to `NULL` and treated as read-only. Python will fill it in when the type is *initialized*.

For dynamically created classes, the `Py_tp_bases` *slot* can be used instead of the *bases* argument of *PyType_FromSpecWithBases()*. The argument form is preferred.

> **경고**
>
> Multiple inheritance does not work well for statically defined types. If you set `tp_bases` to a tuple, Python will not raise an error, but some slots will only be inherited from the first base.

계승:

이 필드는 상속되지 않습니다.

PyObject **PyTypeObject*.**tp_mro**

형 자체에서 시작하여 `object`로 끝나는 확장된 베이스형 집합을 포함하는 튜플.

This field should be set to `NULL` and treated as read-only. Python will fill it in when the type is *initialized*.

계승:

이 필드는 상속되지 않습니다; *PyType_Ready()*에 의해 새로 계산됩니다.

PyObject ******PyTypeObject*.**tp_cache**

사용되지 않습니다. 내부 전용.

계승:

이 필드는 상속되지 않습니다.

void ******PyTypeObject*.**tp_subclasses**

A collection of subclasses. Internal use only. May be an invalid pointer.

To get a list of subclasses, call the Python method `__subclasses__()`.

버전 3.12에서 변경: For some types, this field does not hold a valid *PyObject**. The type was changed to `void*` to indicate this.

계승:

이 필드는 상속되지 않습니다.

PyObject ******PyTypeObject*.**tp_weaklist**

이 형 객체에 대한 약한 참조를 위한 약한 참조 리스트 헤드. 상속되지 않습니다. 내부 전용.

버전 3.12에서 변경: Internals detail: For the static builtin types this is always `NULL`, even if weakrefs are added. Instead, the weakrefs for each are stored on `PyInterpreterState`. Use the public C-API or the internal `_PyObject_GET_WEAKREFS_LISTPTR()` macro to avoid the distinction.

계승:

이 필드는 상속되지 않습니다.

destructor *PyTypeObject*.**tp_del**

이 필드는 폐지되었습니다. 대신 *tp_finalize*를 사용하십시오.

unsigned int *PyTypeObject*.**tp_version_tag**

메서드 캐시에 인덱싱하는 데 사용됩니다. 내부 전용.

계승:

이 필드는 상속되지 않습니다.

destructor *PyTypeObject*.**tp_finalize**

인스턴스 파이널리제이션 함수에 대한 선택적 포인터. 서명은 다음과 같습니다:

```
void tp_finalize(PyObject *self);
```

*tp_finalize*가 설정되면, 인터프리터는 인스턴스를 파이널라이즈 할 때 이를 한 번 호출합니다. 가비지 수집기(인스턴스가 격리된 참조 순환의 일부인 경우)나 객체가 할당 해제되기 직전에 호출됩니다. 어느 쪽이든, 참조 순환을 끊기 전에 호출되어 정상 상태에 있는 객체를 보도록 보장합니다.

*tp_finalize*는 현재 예외 상태를 변경하지 않아야 합니다; 따라서 사소하지 않은 파이널라이저를 작성하는 권장 방법은 다음과 같습니다:

```
static void
local_finalize(PyObject *self)
{
    PyObject *error_type, *error_value, *error_traceback;

    /* Save the current exception, if any. */
    PyErr_Fetch(&error_type, &error_value, &error_traceback);

    /* ... */

    /* Restore the saved exception. */
    PyErr_Restore(error_type, error_value, error_traceback);
}
```

계승:

이 필드는 서브 형으로 상속됩니다.

Added in version 3.4.

버전 3.8에서 변경: Before version 3.8 it was necessary to set the *Py_TPFLAGS_HAVE_FINALIZE* flags bit in order for this field to be used. This is no longer required.

더 보기

"안전한 객체 파이널리제이션" (**PEP 442**)

vectorcallfunc *PyTypeObject*.**tp_vectorcall**

Vectorcall function to use for calls of this type object. In other words, it is used to implement *vectorcall* for `type.__call__`. If `tp_vectorcall` is `NULL`, the default call implementation using `__new__()` and `__init__()` is used.

계승:

이 필드는 상속되지 않습니다.

Added in version 3.9: (필드는 3.8부터 존재하지만 3.9부터 사용됩니다)

unsigned char *PyTypeObject*.**tp_watched**

Internal. Do not use.

Added in version 3.12.

12.3.6 Static Types

전통적으로, C 코드에서 정의된 형은 정적(*static*)입니다. 즉 정적 *PyTypeObject* 구조체는 코드에서 직접 정의되고 *PyType_Ready()*를 사용하여 초기화됩니다.

결과적으로 파이썬에서 정의된 형에 비해 형이 제한됩니다:

- 정적 형은 하나의 베이스로 제한됩니다. 즉, 다중 상속을 사용할 수 없습니다.
- 정적 형 객체(그러나 이들의 인스턴스는 아닙니다)는 불변입니다. 파이썬에서 형 객체의 어트리뷰트를 추가하거나 수정할 수 없습니다.
- 정적 형 객체는 서브 인터프리터에서 공유되므로, 서브 인터프리터 관련 상태를 포함하지 않아야 합니다.

Also, since *PyTypeObject* is only part of the *Limited API* as an opaque struct, any extension modules using static types must be compiled for a specific Python minor version.

12.3.7 힙 형

An alternative to *static types* is *heap-allocated types*, or *heap types* for short, which correspond closely to classes created by Python's `class` statement. Heap types have the *Py_TPFLAGS_HEAPTYPE* flag set.

This is done by filling a *PyType_Spec* structure and calling *PyType_FromSpec()*, *PyType_FromSpecWithBases()*, *PyType_FromModuleAndSpec()*, or *PyType_FromMetaclass()*.

12.3.8 숫자 객체 구조체

type **PyNumberMethods**

이 구조체는 객체가 숫자 프로토콜을 구현하는 데 사용하는 함수에 대한 포인터를 담습니다. 각 함수는 숫자 프로토콜 섹션에서 설명하는 유사한 이름의 함수가 사용합니다.

구조체 정의는 다음과 같습니다:

```
typedef struct {
     binaryfunc nb_add;
     binaryfunc nb_subtract;
     binaryfunc nb_multiply;
     binaryfunc nb_remainder;
     binaryfunc nb_divmod;
     ternaryfunc nb_power;
     unaryfunc nb_negative;
     unaryfunc nb_positive;
     unaryfunc nb_absolute;
     inquiry nb_bool;
     unaryfunc nb_invert;
     binaryfunc nb_lshift;
     binaryfunc nb_rshift;
     binaryfunc nb_and;
     binaryfunc nb_xor;
     binaryfunc nb_or;
     unaryfunc nb_int;
     void *nb_reserved;
     unaryfunc nb_float;

     binaryfunc nb_inplace_add;
     binaryfunc nb_inplace_subtract;
     binaryfunc nb_inplace_multiply;
     binaryfunc nb_inplace_remainder;
     ternaryfunc nb_inplace_power;
     binaryfunc nb_inplace_lshift;
     binaryfunc nb_inplace_rshift;
     binaryfunc nb_inplace_and;
     binaryfunc nb_inplace_xor;
     binaryfunc nb_inplace_or;

     binaryfunc nb_floor_divide;
     binaryfunc nb_true_divide;
     binaryfunc nb_inplace_floor_divide;
     binaryfunc nb_inplace_true_divide;

     unaryfunc nb_index;

     binaryfunc nb_matrix_multiply;
     binaryfunc nb_inplace_matrix_multiply;
} PyNumberMethods;
```

참고

이항과 삼항 함수는 모든 피연산자의 형을 확인하고, 필요한 변환을 구현해야합니다 (적어도 피연산자 중 하나는 정의된 형의 인스턴스입니다). 주어진 피연산자에 대해 연산이 정의되지 않으면, 이항과 삼항 함수는 `Py_NotImplemented`를 반환해야하며, 다른 에러가 발생하면 `NULL`을 반환하고 예외를 설정해야 합니다.

참고

The *nb_reserved* field should always be `NULL`. It was previously called `nb_long`, and was renamed in Python 3.0.1.

binaryfunc *PyNumberMethods*.**nb_add**

binaryfunc *PyNumberMethods*.**nb_subtract**

binaryfunc *PyNumberMethods*.**nb_multiply**

binaryfunc *PyNumberMethods*.**nb_remainder**

binaryfunc *PyNumberMethods*.**nb_divmod**

ternaryfunc *PyNumberMethods*.**nb_power**

unaryfunc *PyNumberMethods*.**nb_negative**

unaryfunc *PyNumberMethods*.**nb_positive**

unaryfunc *PyNumberMethods*.**nb_absolute**

inquiry *PyNumberMethods*.**nb_bool**

unaryfunc *PyNumberMethods*.**nb_invert**

binaryfunc *PyNumberMethods*.**nb_lshift**

binaryfunc *PyNumberMethods*.**nb_rshift**

binaryfunc *PyNumberMethods*.**nb_and**

binaryfunc *PyNumberMethods*.**nb_xor**

binaryfunc *PyNumberMethods*.**nb_or**

unaryfunc *PyNumberMethods*.**nb_int**

void **PyNumberMethods*.**nb_reserved**

unaryfunc *PyNumberMethods*.**nb_float**

binaryfunc *PyNumberMethods*.**nb_inplace_add**

binaryfunc *PyNumberMethods*.**nb_inplace_subtract**

binaryfunc *PyNumberMethods*.**nb_inplace_multiply**

binaryfunc *PyNumberMethods*.**nb_inplace_remainder**

ternaryfunc *PyNumberMethods*.**nb_inplace_power**

binaryfunc *PyNumberMethods*.**nb_inplace_lshift**

binaryfunc *PyNumberMethods*.**nb_inplace_rshift**

binaryfunc *PyNumberMethods*.**nb_inplace_and**

binaryfunc *PyNumberMethods*.**nb_inplace_xor**

binaryfunc *PyNumberMethods*.**nb_inplace_or**

binaryfunc *PyNumberMethods*.**nb_floor_divide**

binaryfunc *PyNumberMethods*.**nb_true_divide**

binaryfunc *PyNumberMethods*.**nb_inplace_floor_divide**

binaryfunc *PyNumberMethods*.**nb_inplace_true_divide**

unaryfunc *PyNumberMethods*.**nb_index**

binaryfunc *PyNumberMethods*.**nb_matrix_multiply**

binaryfunc *PyNumberMethods*.**nb_inplace_matrix_multiply**

12.3.9 매핑 객체 구조체

type **PyMappingMethods**

이 구조체에는 객체가 매핑 프로토콜을 구현하는 데 사용하는 함수에 대한 포인터를 담습니다. 세 개의 멤버가 있습니다:

lenfunc *PyMappingMethods*.**mp_length**

이 함수는 *PyMapping_Size()*와 *PyObject_Size()*에서 사용되며, 같은 서명을 갖습니다. 객체에 길이가 정의되어 있지 않으면 이 슬롯을 NULL로 설정할 수 있습니다.

binaryfunc *PyMappingMethods*.**mp_subscript**

이 함수는 *PyObject_GetItem()*과 *PySequence_GetSlice()*에서 사용되며, PyObject_GetItem()과 같은 서명을 갖습니다. *PyMapping_Check()* 함수가 1을 반환하려면, 이 슬롯을 채워야합니다, 그렇지 않으면 NULL일 수 있습니다.

objobjargproc *PyMappingMethods*.**mp_ass_subscript**

This function is used by *PyObject_SetItem()*, *PyObject_DelItem()*, *PySequence_SetSlice()* and *PySequence_DelSlice()*. It has the same signature as PyObject_SetItem(), but *v* can also be set to NULL to delete an item. If this slot is NULL, the object does not support item assignment and deletion.

12.3.10 시퀀스 객체 구조체

type **PySequenceMethods**

이 구조체는 객체가 시퀀스 프로토콜을 구현하는 데 사용하는 함수에 대한 포인터를 담습니다.

lenfunc *PySequenceMethods*.**sq_length**

이 함수는 *PySequence_Size()*와 *PyObject_Size()*에서 사용되며, 같은 서명을 갖습니다. 또한 *sq_item*과 *sq_ass_item* 슬롯을 통해 음수 인덱스를 처리하는 데 사용됩니다.

binaryfunc *PySequenceMethods*.**sq_concat**

이 함수는 *PySequence_Concat()*에서 사용되며 같은 서명을 갖습니다. *nb_add* 슬롯을 통해 숫자 덧셈을 시도한 후, + 연산자에서도 사용됩니다.

ssizeargfunc *PySequenceMethods*.**sq_repeat**

이 함수는 *PySequence_Repeat()*에서 사용되며 같은 서명을 갖습니다. *nb_multiply* 슬롯을 통해 숫자 곱셈을 시도한 후, * 연산자에서도 사용됩니다.

ssizeargfunc *PySequenceMethods*.**sq_item**

이 함수는 *PySequence_GetItem()*에서 사용되며 같은 서명을 갖습니다. *mp_subscript* 슬롯을 통해 서브스크립션(subscription)을 시도한 후, *PyObject_GetItem()*에서도 사용됩니다. *PySequence_Check()* 함수가 1을 반환하려면, 이 슬롯을 채워야합니다, 그렇지 않으면 NULL일 수 있습니다.

Negative indexes are handled as follows: if the *sq_length* slot is filled, it is called and the sequence length is used to compute a positive index which is passed to *sq_item*. If sq_length is NULL, the index is passed as is to the function.

ssizeobjargproc *PySequenceMethods*.**sq_ass_item**

이 함수는 *PySequence_SetItem()*에서 사용되며 같은 서명을 갖습니다. *mp_ass_subscript* 슬롯을 통해 항목 대입과 삭제를 시도한 후, *PyObject_SetItem()*과 *PyObject_DelItem()*에서도 사용됩니다. 객체가 항목 대입과 삭제를 지원하지 않으면 이 슬롯은 NULL로 남겨 둘 수 있습니다.

objobjproc *PySequenceMethods*.**sq_contains**

이 함수는 *PySequence_Contains()*에서 사용될 수 있으며 같은 서명을 갖습니다. 이 슬롯은 NULL로 남겨 둘 수 있습니다, 이때 PySequence_Contains()는 일치하는 것을 찾을 때까지 시퀀스를 단순히 탐색합니다.

binaryfunc *PySequenceMethods*.**sq_inplace_concat**

이 함수는 *PySequence_InPlaceConcat()*에서 사용되며 같은 서명을 갖습니다. 첫 번째 피연산자를 수정하고 그것을 반환해야 합니다. 이 슬롯은 NULL로 남겨 둘 수 있으며, 이때 PySequence_InPlaceConcat()은 *PySequence_Concat()*으로 폴백 됩니다. *nb_inplace_add* 슬롯을 통해 숫자 제자리 덧셈을 시도한 후, 증분 대입 +=에서 사용됩니다.

ssizeargfunc *PySequenceMethods*.**sq_inplace_repeat**

이 함수는 *PySequence_InPlaceRepeat()*에서 사용되며 같은 서명을 갖습니다. 첫 번째 피연산자를 수정하고 그것을 반환해야 합니다. 이 슬롯은 NULL로 남겨 둘 수 있으며, 이때 PySequence_InPlaceRepeat()는 *PySequence_Repeat()*로 폴백 됩니다. *nb_inplace_multiply* 슬롯을 통해 숫자 제자리 곱셈을 시도한 후, 증분 대입 *=에서도 사용됩니다.

12.3.11 버퍼 객체 구조체

type **PyBufferProcs**

이 구조체는 버퍼 프로토콜에 필요한 함수에 대한 포인터를 담습니다. 프로토콜은 제공자(exporter) 객체가 내부 데이터를 소비자 객체에 노출하는 방법을 정의합니다.

getbufferproc *PyBufferProcs*.**bf_getbuffer**

이 함수의 서명은 다음과 같습니다:

```
int (PyObject *exporter, Py_buffer *view, int flags);
```

*view*를 채우기 위해 *exporter*에 대한 *flags*에 지정된 요청을 처리합니다. 포인트 (3) 을 제외하고, 이 함수의 구현은 다음 단계를 반드시 수행해야 합니다:

(1) Check if the request can be met. If not, raise BufferError, set view->obj to NULL and return -1.

(2) 요청된 필드를 채웁니다.

(3) 내보내기 횟수에 대한 내부 카운터를 증가시킵니다.

(4) Set view->obj to *exporter* and increment view->obj.

(5) 0을 반환합니다.

*exporter*가 버퍼 공급자의 체인이나 트리의 일부이면, 두 가지 주요 체계를 사용할 수 있습니다:

- Re-export: Each member of the tree acts as the exporting object and sets view->obj to a new reference to itself.
- Redirect: The buffer request is redirected to the root object of the tree. Here, view->obj will be a new reference to the root object.

*view*의 개별 필드는 섹션 버퍼 구조체에 설명되어 있으며, 제공자가 특정 요청에 응답해야 하는 규칙은 섹션 버퍼 요청 유형에 있습니다.

Py_buffer 구조체에서 가리키는 모든 메모리는 제공자에게 속하며 남은 소비자가 없어질 때까지 유효해야 합니다. *format*, *shape*, *strides*, *suboffsets* 및 *internal*은 소비자에게는 읽기 전용입니다.

*PyBuffer_FillInfo()*는 모든 요청 유형을 올바르게 처리하면서 간단한 바이트열 버퍼를 쉽게 노출할 수 있는 방법을 제공합니다.

*PyObject_GetBuffer()*는 이 함수를 감싸는 소비자 용 인터페이스입니다.

releasebufferproc *PyBufferProcs*.**bf_releasebuffer**

이 함수의 서명은 다음과 같습니다:

```
void (PyObject *exporter, Py_buffer *view);
```

버퍼 자원 해제 요청을 처리합니다. 자원을 해제할 필요가 없으면, *PyBufferProcs.bf_releasebuffer*는 NULL일 수 있습니다. 그렇지 않으면, 이 함수의 표준 구현은 다음과 같은 선택적 단계를 수행합니다:

(1) 내보내기 횟수에 대한 내부 카운터를 줄입니다.

(2) 카운터가 0이면, *view*와 관련된 모든 메모리를 해제합니다.

제공자는 반드시 *internal* 필드를 사용하여 버퍼 특정 자원을 추적해야 합니다. 이 필드는 변경되지 않고 유지됨이 보장되지만, 소비자는 원래 버퍼의 사본을 *view* 인자로 전달할 수 있습니다.

This function MUST NOT decrement `view->obj`, since that is done automatically in *PyBuffer_Release()* (this scheme is useful for breaking reference cycles).

*PyBuffer_Release()*는 이 기능을 감싸는 소비자 용 인터페이스입니다.

12.3.12 비동기 객체 구조체

Added in version 3.5.

type **PyAsyncMethods**

이 구조체는 어웨이터블와 비동기 이터레이터 객체를 구현하는 데 필요한 함수에 대한 포인터를 담습니다.

구조체 정의는 다음과 같습니다:

```
typedef struct {
    unaryfunc am_await;
    unaryfunc am_aiter;
    unaryfunc am_anext;
    sendfunc am_send;
} PyAsyncMethods;
```

unaryfunc *PyAsyncMethods*.**am_await**

이 함수의 서명은 다음과 같습니다:

```
PyObject *am_await(PyObject *self);
```

The returned object must be an *iterator*, i.e. *PyIter_Check()* must return `1` for it.

객체가 어웨이터블이 아니면 이 슬롯을 NULL로 설정할 수 있습니다.

unaryfunc *PyAsyncMethods*.**am_aiter**

이 함수의 서명은 다음과 같습니다:

```
PyObject *am_aiter(PyObject *self);
```

Must return an *asynchronous iterator* object. See `__anext__()` for details.

객체가 비동기 이터레이션 프로토콜을 구현하지 않으면 이 슬롯은 NULL로 설정될 수 있습니다.

unaryfunc *PyAsyncMethods*.**am_anext**

이 함수의 서명은 다음과 같습니다:

```
PyObject *am_anext(PyObject *self);
```

Must return an *awaitable* object. See `__anext__()` for details. This slot may be set to NULL.

sendfunc *PyAsyncMethods*.**am_send**

이 함수의 서명은 다음과 같습니다:

```
PySendResult am_send(PyObject *self, PyObject *arg, PyObject **result);
```

See *PyIter_Send()* for details. This slot may be set to `NULL`.

Added in version 3.10.

12.3.13 슬롯 형 typedef

typedef *PyObject* *(***allocfunc**)(*PyTypeObject* *cls, *Py_ssize_t* nitems)

Part of the Stable ABI. The purpose of this function is to separate memory allocation from memory initialization. It should return a pointer to a block of memory of adequate length for the instance, suitably aligned, and initialized to zeros, but with *ob_refcnt* set to `1` and *ob_type* set to the type argument. If the type' s *tp_itemsize* is non-zero, the object' s *ob_size* field should be initialized to *nitems* and the length of the allocated memory block should be `tp_basicsize + nitems*tp_itemsize`, rounded up to a multiple of `sizeof(void*)`; otherwise, *nitems* is not used and the length of the block should be *tp_basicsize*.

이 함수는 다른 인스턴스 초기화를 수행하지 않아야 합니다, 추가 메모리를 할당도 안 됩니다; 그것은 *tp_new*에 의해 수행되어야 합니다.

typedef void (***destructor**)(*PyObject**)

Part of the Stable ABI.

typedef void (***freefunc**)(void*)

*tp_free*를 참조하십시오.

typedef *PyObject* *(***newfunc**)(*PyTypeObject**, *PyObject**, *PyObject**)

Part of the Stable ABI. *tp_new*를 참조하십시오.

typedef int (***initproc**)(*PyObject**, *PyObject**, *PyObject**)

Part of the Stable ABI. *tp_init*를 참조하십시오.

typedef *PyObject* *(***reprfunc**)(*PyObject**)

Part of the Stable ABI. *tp_repr*을 참조하십시오.

typedef *PyObject* *(***getattrfunc**)(*PyObject* *self, char *attr)

Part of the Stable ABI. 객체의 명명된 어트리뷰트 값을 반환합니다.

typedef int (***setattrfunc**)(*PyObject* *self, char *attr, *PyObject* *value)

Part of the Stable ABI. 객체의 명명된 어트리뷰트 값을 설정합니다. 어트리뷰트를 삭제하려면 value 인자가 `NULL`로 설정됩니다.

typedef *PyObject* *(***getattrofunc**)(*PyObject* *self, *PyObject* *attr)

Part of the Stable ABI. 객체의 명명된 어트리뷰트 값을 반환합니다.

*tp_getattro*를 참조하십시오.

typedef int (***setattrofunc**)(*PyObject* *self, *PyObject* *attr, *PyObject* *value)

Part of the Stable ABI. 객체의 명명된 어트리뷰트 값을 설정합니다. 어트리뷰트를 삭제하려면 value 인자가 `NULL`로 설정됩니다.

*tp_setattro*를 참조하십시오.

typedef *PyObject* *(***descrgetfunc**)(*PyObject**, *PyObject**, *PyObject**)

Part of the Stable ABI. See *tp_descr_get*.

typedef int (***descrsetfunc**)(*PyObject**, *PyObject**, *PyObject**)

Part of the Stable ABI. See *tp_descr_set*.

typedef *Py_hash_t* (***hashfunc**)(*PyObject**)

Part of the Stable ABI. *tp_hash*를 참조하십시오.

typedef *PyObject* *(***richcmpfunc**)(*PyObject**, *PyObject**, int)

Part of the Stable ABI. `tp_richcompare`를 참조하십시오.

typedef *PyObject* *(***getiterfunc**)(*PyObject**)

Part of the Stable ABI. `tp_iter`를 참조하십시오.

typedef *PyObject* *(***iternextfunc**)(*PyObject**)

Part of the Stable ABI. `tp_iternext`를 참조하십시오.

typedef *Py_ssize_t* (***lenfunc**)(*PyObject**)

Part of the Stable ABI.

typedef int (***getbufferproc**)(*PyObject**, *Py_buffer**, int)

Part of the Stable ABI *since version 3.12.*

typedef void (***releasebufferproc**)(*PyObject**, *Py_buffer**)

Part of the Stable ABI *since version 3.12.*

typedef *PyObject* *(***unaryfunc**)(*PyObject**)

Part of the Stable ABI.

typedef *PyObject* *(***binaryfunc**)(*PyObject**, *PyObject**)

Part of the Stable ABI.

typedef *PySendResult* (***sendfunc**)(*PyObject**, *PyObject**, *PyObject***)

See `am_send`.

typedef *PyObject* *(***ternaryfunc**)(*PyObject**, *PyObject**, *PyObject**)

Part of the Stable ABI.

typedef *PyObject* *(***ssizeargfunc**)(*PyObject**, *Py_ssize_t*)

Part of the Stable ABI.

typedef int (***ssizeobjargproc**)(*PyObject**, *Py_ssize_t*, *PyObject**)

Part of the Stable ABI.

typedef int (***objobjproc**)(*PyObject**, *PyObject**)

Part of the Stable ABI.

typedef int (***objobjargproc**)(*PyObject**, *PyObject**, *PyObject**)

Part of the Stable ABI.

12.3.14 예

다음은 파이썬 형 정의의 간단한 예입니다. 여기에는 여러분이 만날 수 있는 일반적인 사용법이 포함됩니다. 일부는 까다로운 코너 사례를 보여줍니다. 더 많은 예제, 실용 정보 및 자습서는 defining-new-types와 new-types-topics를 참조하십시오.

A basic *static type*:

```
typedef struct {
    PyObject_HEAD
    const char *data;
} MyObject;

static PyTypeObject MyObject_Type = {
    PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "mymod.MyObject",
    .tp_basicsize = sizeof(MyObject),
    .tp_doc = PyDoc_STR("My objects"),
    .tp_new = myobj_new,
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    .tp_dealloc = (destructor)myobj_dealloc,
    .tp_repr = (reprfunc)myobj_repr,
};
```

더 상세한 초기화자를 사용하는 이전 코드(특히 CPython 코드 베이스에서)를 찾을 수도 있습니다:

```
static PyTypeObject MyObject_Type = {
    PyVarObject_HEAD_INIT(NULL, 0)
    "mymod.MyObject",               /* tp_name */
    sizeof(MyObject),               /* tp_basicsize */
    0,                              /* tp_itemsize */
    (destructor)myobj_dealloc,      /* tp_dealloc */
    0,                              /* tp_vectorcall_offset */
    0,                              /* tp_getattr */
    0,                              /* tp_setattr */
    0,                              /* tp_as_async */
    (reprfunc)myobj_repr,           /* tp_repr */
    0,                              /* tp_as_number */
    0,                              /* tp_as_sequence */
    0,                              /* tp_as_mapping */
    0,                              /* tp_hash */
    0,                              /* tp_call */
    0,                              /* tp_str */
    0,                              /* tp_getattro */
    0,                              /* tp_setattro */
    0,                              /* tp_as_buffer */
    0,                              /* tp_flags */
    PyDoc_STR("My objects"),        /* tp_doc */
    0,                              /* tp_traverse */
    0,                              /* tp_clear */
    0,                              /* tp_richcompare */
    0,                              /* tp_weaklistoffset */
    0,                              /* tp_iter */
    0,                              /* tp_iternext */
    0,                              /* tp_methods */
    0,                              /* tp_members */
    0,                              /* tp_getset */
    0,                              /* tp_base */
    0,                              /* tp_dict */
    0,                              /* tp_descr_get */
    0,                              /* tp_descr_set */
    0,                              /* tp_dictoffset */
    0,                              /* tp_init */
    0,                              /* tp_alloc */
    myobj_new,                      /* tp_new */
};
```

약한 참조, 인스턴스 딕셔너리 및 해싱을 지원하는 형:

```
typedef struct {
    PyObject_HEAD
    const char *data;
} MyObject;

static PyTypeObject MyObject_Type = {
    PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "mymod.MyObject",
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    .tp_basicsize = sizeof(MyObject),
    .tp_doc = PyDoc_STR("My objects"),
    .tp_flags = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE |
        Py_TPFLAGS_HAVE_GC | Py_TPFLAGS_MANAGED_DICT |
        Py_TPFLAGS_MANAGED_WEAKREF,
    .tp_new = myobj_new,
    .tp_traverse = (traverseproc)myobj_traverse,
    .tp_clear = (inquiry)myobj_clear,
    .tp_alloc = PyType_GenericNew,
    .tp_dealloc = (destructor)myobj_dealloc,
    .tp_repr = (reprfunc)myobj_repr,
    .tp_hash = (hashfunc)myobj_hash,
    .tp_richcompare = PyBaseObject_Type.tp_richcompare,
};
```

A str subclass that cannot be subclassed and cannot be called to create instances (e.g. uses a separate factory func) using *Py_TPFLAGS_DISALLOW_INSTANTIATION* flag:

```
typedef struct {
    PyUnicodeObject raw;
    char *extra;
} MyStr;

static PyTypeObject MyStr_Type = {
    PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "mymod.MyStr",
    .tp_basicsize = sizeof(MyStr),
    .tp_base = NULL,  // set to &PyUnicode_Type in module init
    .tp_doc = PyDoc_STR("my custom str"),
    .tp_flags = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_DISALLOW_INSTANTIATION,
    .tp_repr = (reprfunc)myobj_repr,
};
```

The simplest *static type* with fixed-length instances:

```
typedef struct {
    PyObject_HEAD
} MyObject;

static PyTypeObject MyObject_Type = {
    PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "mymod.MyObject",
};
```

The simplest *static type* with variable-length instances:

```
typedef struct {
    PyObject_VAR_HEAD
    const char *data[1];
} MyObject;

static PyTypeObject MyObject_Type = {
    PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "mymod.MyObject",
    .tp_basicsize = sizeof(MyObject) - sizeof(char *),
    .tp_itemsize = sizeof(char *),
};
```

12.4 순환 가비지 수집 지원

순환 참조를 포함하는 가비지를 탐지하고 수집하는 파이썬의 지원은 역시 컨테이너일 수 있는 다른 객체의 "컨테이너" 인 객체 형의 지원이 필요합니다. 다른 객체에 대한 참조를 저장하지 않거나, 원자 형(가령 숫자나 문자열)에 대한 참조만 저장하는 형은 가비지 수집에 대한 어떤 명시적인 지원을 제공할 필요가 없습니다.

To create a container type, the `tp_flags` field of the type object must include the `Py_TPFLAGS_HAVE_GC` and provide an implementation of the `tp_traverse` handler. If instances of the type are mutable, a `tp_clear` implementation must also be provided.

`Py_TPFLAGS_HAVE_GC`
: 이 플래그가 설정된 형의 객체는 여기에 설명된 규칙을 준수해야 합니다. 편의를 위해 이러한 객체를 컨테이너 객체라고 하겠습니다.

컨테이너형의 생성자는 두 가지 규칙을 준수해야 합니다:

1. The memory for the object must be allocated using `PyObject_GC_New` or `PyObject_GC_NewVar`.
2. 다른 컨테이너에 대한 참조를 포함할 수 있는 모든 필드가 초기화되면, `PyObject_GC_Track()`를 호출해야 합니다.

마찬가지로, 객체의 할당해제자(deallocator)는 비슷한 규칙 쌍을 준수해야 합니다:

1. 다른 컨테이너를 참조하는 필드가 무효화 되기 전에, `PyObject_GC_UnTrack()`를 호출해야 합니다.
2. 객체의 메모리는 `PyObject_GC_Del()`를 사용하여 할당 해제되어야 합니다.

> **경고**
>
> If a type adds the Py_TPFLAGS_HAVE_GC, then it *must* implement at least a `tp_traverse` handler or explicitly use one from its subclass or subclasses.
>
> When calling `PyType_Ready()` or some of the APIs that indirectly call it like `PyType_FromSpecWithBases()` or `PyType_FromSpec()` the interpreter will automatically populate the `tp_flags`, `tp_traverse` and `tp_clear` fields if the type inherits from a class that implements the garbage collector protocol and the child class does *not* include the `Py_TPFLAGS_HAVE_GC` flag.

`PyObject_GC_New`(TYPE, typeobj)
: Analogous to `PyObject_New` but for container objects with the `Py_TPFLAGS_HAVE_GC` flag set.

`PyObject_GC_NewVar`(TYPE, typeobj, size)
: Analogous to `PyObject_NewVar` but for container objects with the `Py_TPFLAGS_HAVE_GC` flag set.

PyObject ***PyUnstable_Object_GC_NewWithExtraData**(*PyTypeObject* *type, size_t extra_size)

> This is *Unstable API*. It may change without warning in minor releases.

Analogous to `PyObject_GC_New` but allocates *extra_size* bytes at the end of the object (at offset `tp_basicsize`). The allocated memory is initialized to zeros, except for the `Python object header`.

The extra data will be deallocated with the object, but otherwise it is not managed by Python.

> **경고**
>
> The function is marked as unstable because the final mechanism for reserving extra data after an instance is not yet decided. For allocating a variable number of fields, prefer using `PyVarObject` and `tp_itemsize` instead.

Added in version 3.12.

PyObject_GC_Resize(TYPE, op, newsize)

Resize an object allocated by *PyObject_NewVar*. Returns the resized object of type `TYPE*` (refers to any C type) or `NULL` on failure.

op must be of type *PyVarObject** and must not be tracked by the collector yet. *newsize* must be of type *Py_ssize_t*.

void **PyObject_GC_Track**(*PyObject* *op)

Part of the Stable ABI. 수집기가 추적하는 컨테이너 객체 집합에 객체 *op*를 추가합니다. 수집기는 예기치 않은 시간에 실행될 수 있으므로 추적되는 동안 객체가 유효해야 합니다. *tp_traverse* 처리기가 탐색하는 모든 필드가 유효해지면 호출해야 합니다, 보통 생성자의 끝부분 근처입니다.

int **PyObject_IS_GC**(*PyObject* *obj)

객체가 가비지 수거기 프로토콜을 구현하면 0이 아닌 값을 반환하고, 그렇지 않으면 0을 반환합니다.

이 함수가 0을 반환하면 가비지 수거기가 객체를 추적할 수 없습니다.

int **PyObject_GC_IsTracked**(*PyObject* *op)

Part of the Stable ABI *since version 3.9.* *op*의 객체 형이 GC 프로토콜을 구현하고 *op*가 현재 가비지 수거기가 추적 중이면 1을 반환하고 그렇지 않으면 0을 반환합니다.

이것은 파이썬 함수 `gc.is_tracked()`에 해당합니다.

Added in version 3.9.

int **PyObject_GC_IsFinalized**(*PyObject* *op)

Part of the Stable ABI *since version 3.9.* *op*의 객체 형이 GC 프로토콜을 구현하고 가비지 수거기가 *op*를 이미 파이널라이즈 했으면 1을 반환하고 그렇지 않으면 0을 반환합니다.

이것은 파이썬 함수 `gc.is_finalized()`에 해당합니다.

Added in version 3.9.

void **PyObject_GC_Del**(void *op)

Part of the Stable ABI. Releases memory allocated to an object using *PyObject_GC_New* or *PyObject_GC_NewVar*.

void **PyObject_GC_UnTrack**(void *op)

Part of the Stable ABI. 수집기가 추적하는 컨테이너 객체 집합에서 *op* 객체를 제거합니다. *PyObject_GC_Track()*를 이 객체에 대해 다시 호출하여 추적 객체 집합에 다시 추가할 수 있음에 유의하십시오. 할당해제자(*tp_dealloc* 처리기)는 *tp_traverse* 처리기에서 사용하는 필드가 무효화되기 전에 객체에 대해 이 함수를 호출해야 합니다.

버전 3.8에서 변경: The `_PyObject_GC_TRACK()` and `_PyObject_GC_UNTRACK()` macros have been removed from the public C API.

tp_traverse 처리기는 다음과 같은 형의 함수 매개 변수를 받아들입니다:

typedef int (***visitproc**)(*PyObject* *object, void *arg)

Part of the Stable ABI. *tp_traverse* 처리기에 전달되는 방문자 함수의 형. 이 함수는 탐색하는 객체를 *object*로, *tp_traverse* 처리기의 세 번째 매개 변수를 *arg*로 호출되어야 합니다. 파이썬 코어는 순환 가비지 탐지를 구현하기 위해 여러 방문자 함수를 사용합니다; 사용자가 자신의 방문자 함수를 작성해야 할 필요는 없습니다.

tp_traverse 처리기는 다음 형이어야 합니다:

typedef int (***traverseproc**)(*PyObject* *self, *visitproc* visit, void *arg)

Part of the Stable ABI. 컨테이너 객체의 탐색 함수입니다. 구현은 *self*에 직접 포함된 각 객체에 대해 *visit* 함수를 호출해야 하며, *visit*에 대한 매개 변수는 포함된 객체와 처리기로 전달된 *arg* 값입니다. *visit* 함수는 `NULL` object 인자로 호출하면 안 됩니다. *visit*가 0이 아닌 값을 반환하면 그 값이 즉시 반환되어야 합니다.

tp_traverse 처리기 작성을 단순화하기 위해, *Py_VISIT()* 매크로가 제공됩니다. 이 매크로를 사용하려면, *tp_traverse* 구현은 인자의 이름을 정확히 *visit* 와 *arg*로 지정해야 합니다:

void **Py_VISIT**(*PyObject* *o)

*o*가 NULL이 아니면, *o* 와 *arg* 인자로 *visit* 콜백을 호출합니다. *visit*가 0이 아닌 값을 반환하면, 그것을 반환합니다. 이 매크로를 사용하면, *tp_traverse* 처리기가 다음과 같아집니다:

```
static int
my_traverse(Noddy *self, visitproc visit, void *arg)
{
    Py_VISIT(self->foo);
    Py_VISIT(self->bar);
    return 0;
}
```

tp_clear 처리기는 *inquiry* 형이거나 객체가 불변이면 NULL이어야 합니다.

typedef int (***inquiry**)(*PyObject* *self)

Part of the Stable ABI. 참조 순환을 생성했을 수 있는 참조를 삭제합니다. 불변 객체는 참조 순환을 직접 생성할 수 없으므로, 이 메서드를 정의 할 필요가 없습니다. 이 메서드를 호출한 후에도 객체가 유효해야 합니다 (단지 참조에 대해 *Py_DECREF()*를 호출하지 마십시오). 이 객체가 참조 순환에 참여하고 있음을 수집기가 감지하면 이 메서드를 호출합니다.

12.4.1 Controlling the Garbage Collector State

The C-API provides the following functions for controlling garbage collection runs.

Py_ssize_t **PyGC_Collect**(void)

Part of the Stable ABI. Perform a full garbage collection, if the garbage collector is enabled. (Note that `gc.collect()` runs it unconditionally.)

Returns the number of collected + unreachable objects which cannot be collected. If the garbage collector is disabled or already collecting, returns `0` immediately. Errors during garbage collection are passed to `sys.unraisablehook`. This function does not raise exceptions.

int **PyGC_Enable**(void)

Part of the Stable ABI *since version 3.10.* Enable the garbage collector: similar to `gc.enable()`. Returns the previous state, 0 for disabled and 1 for enabled.

Added in version 3.10.

int **PyGC_Disable**(void)

Part of the Stable ABI *since version 3.10.* Disable the garbage collector: similar to `gc.disable()`. Returns the previous state, 0 for disabled and 1 for enabled.

Added in version 3.10.

int **PyGC_IsEnabled**(void)

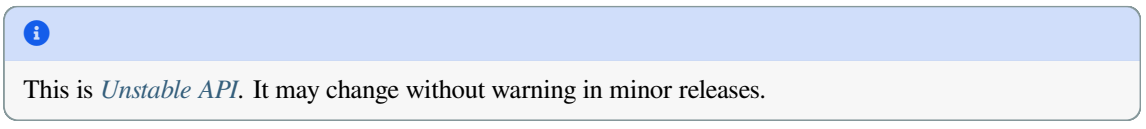
Part of the Stable ABI *since version 3.10.* Query the state of the garbage collector: similar to `gc.isenabled()`. Returns the current state, 0 for disabled and 1 for enabled.

Added in version 3.10.

12.4.2 Querying Garbage Collector State

The C-API provides the following interface for querying information about the garbage collector.

void **PyUnstable_GC_VisitObjects**(*gcvisitobjects_t* callback, void *arg)



This is *Unstable API*. It may change without warning in minor releases.

Run supplied *callback* on all live GC-capable objects. *arg* is passed through to all invocations of *callback*.

> **경고**
>
> If new objects are (de)allocated by the callback it is undefined if they will be visited.
>
> Garbage collection is disabled during operation. Explicitly running a collection in the callback may lead to undefined behaviour e.g. visiting the same objects multiple times or not at all.

Added in version 3.12.

typedef int (***gcvisitobjects_t**)(*PyObject* *object, void *arg)

Type of the visitor function to be passed to *`PyUnstable_GC_VisitObjects()`*. *arg* is the same as the *arg* passed to `PyUnstable_GC_VisitObjects`. Return `0` to continue iteration, return `1` to stop iteration. Other return values are reserved for now so behavior on returning anything else is undefined.

Added in version 3.12.

CHAPTER 13

API와 ABI 버전 붙이기

CPython exposes its version number in the following macros. Note that these correspond to the version code is **built** with, not necessarily the version used at **run time**.

See *C API Stability* for a discussion of API and ABI stability across versions.

PY_MAJOR_VERSION

The `3` in `3.4.1a2`.

PY_MINOR_VERSION

The `4` in `3.4.1a2`.

PY_MICRO_VERSION

The `1` in `3.4.1a2`.

PY_RELEASE_LEVEL

The `a` in `3.4.1a2`. This can be `0xA` for alpha, `0xB` for beta, `0xC` for release candidate or `0xF` for final.

PY_RELEASE_SERIAL

The `2` in `3.4.1a2`. Zero for final releases.

PY_VERSION_HEX

The Python version number encoded in a single integer.

The underlying version information can be found by treating it as a 32 bit number in the following manner:

바이트	비트 (빅 엔디안 순서)	뜻	Value for `3.4.1a2`
1	1-8	`PY_MAJOR_VERSION`	`0x03`
2	9-16	`PY_MINOR_VERSION`	`0x04`
3	17-24	`PY_MICRO_VERSION`	`0x01`
4	25-28	`PY_RELEASE_LEVEL`	`0xA`
	29-32	`PY_RELEASE_SERIAL`	`0x2`

Thus `3.4.1a2` is hexversion `0x030401a2` and `3.10.0` is hexversion `0x030a00f0`.

Use this for numeric comparisons, e.g. `#if PY_VERSION_HEX >= ...`.

This version is also available via the symbol *Py_Version*.

const unsigned long **Py_Version**

Part of the Stable ABI *since version 3.11.* The Python runtime version number encoded in a single constant integer, with the same format as the *PY_VERSION_HEX* macro. This contains the Python version used at run time.

Added in version 3.11.

모든 주어진 매크로는 Include/patchlevel.h에 정의됩니다.

APPENDIX A

용어집

>>>
대화형 셸의 기본 파이썬 프롬프트. 인터프리터에서 대화형으로 실행될 수 있는 코드 예에서 자주 볼 수 있습니다.

...
다음과 같은 것들을 가리킬 수 있습니다:

- 들여쓰기 된 코드 블록의 코드를 입력할 때, 쌍을 이루는 구분자 (괄호, 대괄호, 중괄호) 안에 코드를 입력할 때, 데코레이터 지정 후의 대화형 셸의 기본 파이썬 프롬프트.
- `Ellipsis` 내장 상수.

2to3
파이썬 2.x 코드를 파이썬 3.x 코드로 변환하려고 시도하는 도구인데, 소스를 구문 분석하고 구문 분석 트리를 탐색해서 감지할 수 있는 대부분의 비호환성을 다룹니다.

2to3 는 표준 라이브러리에서 `lib2to3` 로 제공됩니다; 독립적으로 실행할 수 있는 스크립트는 `Tools/scripts/2to3` 로 제공됩니다. 2to3-reference을 보세요.

abstract base class (추상 베이스 클래스)
추상 베이스 클래스는 `hasattr()` 같은 다른 테크닉들이 불편하거나 미묘하게 잘못된 (예를 들어, 매직 메서드) 경우, 인터페이스를 정의하는 방법을 제공함으로써 덕 타이핑 을 보완합니다. ABC는 가상 서브 클래스를 도입하는데, 클래스를 계승하지 않으면서도 `isinstance()` 와 `issubclass()` 에 의해 감지될 수 있는 클래스들입니다; `abc` 모듈 설명서를 보세요. 파이썬에는 많은 내장 ABC 들이 따라오는데 다음과 같은 것들이 있습니다: 자료 구조 (`collections.abc` 모듈에서), 숫자 (`numbers` 모듈에서), 스트림 (`io` 모듈에서), 임포트 파인더와 로더 (`importlib.abc` 모듈에서). `abc` 모듈을 사용해서 자신만의 ABC를 만들 수도 있습니다.

annotation (어노테이션)
관습에 따라 형 힌트 로 사용되는 변수, 클래스 어트리뷰트 또는 함수 매개변수 나 반환 값과 연결된 레이블입니다.

지역 변수의 어노테이션은 실행 시간에 액세스할 수 없지만, 전역 변수, 클래스 속성 및 함수의 어노테이션은 각각 모듈, 클래스, 함수의 `__annotations__` 특수 어트리뷰트에 저장됩니다.

See *variable annotation*, *function annotation*, **PEP 484** and **PEP 526**, which describe this functionality. Also see annotations-howto for best practices on working with annotations.

argument (인자)
함수를 호출할 때 함수 (또는 메서드) 로 전달되는 값. 두 종류의 인자가 있습니다:

- 키워드 인자 (*keyword argument*): 함수 호출 때 식별자가 앞에 붙은 인자 (예를 들어, `name=`) 또는 `**` 를 앞에 붙인 딕셔너리로 전달되는 인자. 예를 들어, 다음과 같은 `complex()` 호출에서 `3` 과 `5` 는 모두 키워드 인자입니다:

```
complex(real=3, imag=5)
complex(**{'real': 3, 'imag': 5})
```

- 위치 인자 (*positional argument*): 키워드 인자가 아닌 인자. 위치 인자들은 인자 목록의 처음에 나오거나 이터러블 의 앞에 `*` 를 붙여 전달할 수 있습니다. 예를 들어, 다음과 같은 호출에서 `3` 과 `5` 는 모두 위치 인자입니다.

```
complex(3, 5)
complex(*(3, 5))
```

인자는 함수 바디의 이름 붙은 지역 변수에 대입됩니다. 이 대입에 적용되는 규칙들에 대해서는 calls 절을 보세요. 문법적으로, 어떤 표현식이건 인자로 사용될 수 있습니다; 구해진 값이 지역 변수에 대입됩니다.

용어집의 매개변수 항목과 FAQ 질문 인자와 매개변수의 차이 와 **PEP 362**도 보세요.

asynchronous context manager (비동기 컨텍스트 관리자)
An object which controls the environment seen in an `async with` statement by defining `__aenter__()` and `__aexit__()` methods. Introduced by **PEP 492**.

asynchronous generator (비동기 제너레이터)
비동기 제너레이터 이터레이터 를 돌려주는 함수. `async def` 로 정의되는 코루틴 함수처럼 보이는데, `async for` 루프가 사용할 수 있는 일련의 값들을 만드는 `yield` 표현식을 포함한다는 점이 다릅니다.

보통 비동기 제너레이터 함수를 가리키지만, 어떤 문맥에서는 비동기 제너레이터 이터레이터 를 가리킵니다. 의도하는 의미가 명확하지 않은 경우는, 완전한 용어를 써서 모호함을 없앱니다.

비동기 제너레이터 함수는 `await` 표현식과, `async for` 문과, `async with` 문을 포함할 수 있습니다.

asynchronous generator iterator (비동기 제너레이터 이터레이터)
비동기 제너레이터 함수가 만드는 객체.

This is an *asynchronous iterator* which when called using the `__anext__()` method returns an awaitable object which will execute the body of the asynchronous generator function until the next `yield` expression.

Each `yield` temporarily suspends processing, remembering the execution state (including local variables and pending try-statements). When the *asynchronous generator iterator* effectively resumes with another awaitable returned by `__anext__()`, it picks up where it left off. See **PEP 492** and **PEP 525**.

asynchronous iterable (비동기 이터러블)
An object, that can be used in an `async for` statement. Must return an *asynchronous iterator* from its `__aiter__()` method. Introduced by **PEP 492**.

asynchronous iterator (비동기 이터레이터)
An object that implements the `__aiter__()` and `__anext__()` methods. `__anext__()` must return an *awaitable* object. `async for` resolves the awaitables returned by an asynchronous iterator' s `__anext__()` method until it raises a `StopAsyncIteration` exception. Introduced by **PEP 492**.

attribute (어트리뷰트)
A value associated with an object which is usually referenced by name using dotted expressions. For example, if an object *o* has an attribute *a* it would be referenced as *o.a*.

It is possible to give an object an attribute whose name is not an identifier as defined by identifiers, for example using `setattr()`, if the object allows it. Such an attribute will not be accessible using a dotted expression, and would instead need to be retrieved with `getattr()`.

awaitable (어웨이터블)
An object that can be used in an `await` expression. Can be a *coroutine* or an object with an `__await__()` method. See also **PEP 492**.

BDFL
자비로운 종신 독재자 (Benevolent Dictator For Life), 즉 Guido van Rossum, 파이썬의 창시자.

binary file (바이너리 파일)
A *file object* able to read and write *bytes-like objects*. Examples of binary files are files opened in binary mode (`'rb'`, `'wb'` or `'rb+'`), `sys.stdin.buffer`, `sys.stdout.buffer`, and instances of `io.BytesIO` and `gzip.GzipFile`.

`str` 객체를 읽고 쓸 수 있는 파일 객체에 대해서는 텍스트 파일 도 참조하세요.

borrowed reference
In Python' s C API, a borrowed reference is a reference to an object, where the code using the object does not own the reference. It becomes a dangling pointer if the object is destroyed. For example, a garbage collection can remove the last *strong reference* to the object and so destroy it.

Calling `Py_INCREF()` on the *borrowed reference* is recommended to convert it to a *strong reference* in-place, except when the object cannot be destroyed before the last usage of the borrowed reference. The `Py_NewRef()` function can be used to create a new *strong reference*.

bytes-like object (바이트열류 객체)
버퍼 프로토콜 를 지원하고 C-연속 버퍼를 익스포트 할 수 있습니다. 여러 공통 `memoryview` 객체들은 물론이고 `bytes`, `bytearray`, `array.array` 객체들을 포함합니다. 바이트열류 객체들은 바이너리 데이터를 다루는 여러 가지 연산들에 사용될 수 있습니다; 압축, 바이너리 파일로 저장, 소켓을 통한 전송 같은 것들이 있습니다.

어떤 연산들은 바이너리 데이터가 가변적일 필요가 있습니다. 이런 경우에 설명서는 종종 "읽고-쓰기 바이트열류 객체"라고 표현합니다. 가변 버퍼 객체의 예로는 `bytearray` 와 `bytearray` 의 `memoryview` 가 있습니다. 다른 연산들은 바이너리 데이터가 불변 객체 ("읽기 전용 바이트열류 객체")에 저장되도록 요구합니다; 이런 것들의 예로는 `bytes`와 `bytes` 객체의 `memoryview` 가 있습니다.

bytecode (바이트 코드)
파이썬 소스 코드는 바이트 코드로 컴파일되는데, CPython 인터프리터에서 파이썬 프로그램의 내부 표현입니다. 바이트 코드는 `.pyc` 파일에 캐시 되어, 같은 파일을 두 번째 실행할 때 더 빨라지게 만듭니다 (소스에서 바이트 코드로의 재컴파일을 피할 수 있습니다). 이 "중간 언어" 는 각 바이트 코드에 대응하는 기계를 실행하는 가상 기계 에서 실행된다고 말합니다. 바이트 코드는 서로 다른 파이썬 가상 기계에서 작동할 것으로 기대하지도, 파이썬 배포 간에 안정적이지도 않다는 것에 주의해야 합니다.

바이트 코드 명령어들의 목록은 dis 모듈 설명서에 나옵니다.

callable
A callable is an object that can be called, possibly with a set of arguments (see *argument*), with the following syntax:

```
callable(argument1, argument2, argumentN)
```

A *function*, and by extension a *method*, is a callable. An instance of a class that implements the `__call__()` method is also a callable.

callback (콜백)
인자로 전달되는 미래의 어느 시점에서 실행될 서브 루틴 함수.

class (클래스)
사용자 정의 객체들을 만들기 위한 주형. 클래스 정의는 보통 클래스의 인스턴스를 대상으로 연산하는 메서드 정의들을 포함합니다.

class variable (클래스 변수)
클래스에서 정의되고 클래스 수준 (즉, 클래스의 인스턴스에서가 아니라) 에서만 수정되는 변수.

complex number (복소수)
익숙한 실수 시스템의 확장인데, 모든 숫자가 실수부와 허수부의 합으로 표현됩니다. 허수부는 실수에 허수 단위 (`-1`의 제곱근)를 곱한 것인데, 종종 수학에서는 `i`로, 공학에서는 `j`로 표기합니다. 파이썬은 후자의 표기법을 쓰는 복소수를 기본 지원합니다; 허수부는 `j` 접미사를 붙여서 표기합니다,

예를 들어, `3+1j`. `math` 모듈의 복소수 버전이 필요하면, `cmath`를 사용합니다. 복소수의 활용은 꽤 수준 높은 수학적 기능입니다. 필요하다고 느끼지 못한다면, 거의 확실히 무시해도 좋습니다.

context manager (컨텍스트 관리자)
An object which controls the environment seen in a `with` statement by defining `__enter__()` and `__exit__()` methods. See **PEP 343**.

context variable (컨텍스트 변수)
컨텍스트에 따라 다른 값을 가질 수 있는 변수. 이는 각 실행 스레드가 변수에 대해 다른 값을 가질 수 있는 스레드-로컬 저장소와 비슷합니다. 그러나, 컨텍스트 변수를 통해, 하나의 실행 스레드에 여러 컨텍스트가 있을 수 있으며 컨텍스트 변수의 주 용도는 동시성 비동기 태스크에서 변수를 추적하는 것입니다. `contextvars`를 참조하십시오.

contiguous (연속)
버퍼는 정확히 *C*-연속(*C-contiguous*)이거나 포트란 연속(*Fortran contiguous*)일 때 연속이라고 여겨집니다. 영차원 버퍼는 C-연속이면서 포트란 연속입니다. 일차원 배열에서, 항목들은 서로에 인접하고, 0에서 시작하는 오름차순 인덱스의 순서대로 메모리에 배치되어야 합니다. 다차원 C-연속 배열에서, 메모리 주소의 순서대로 항목들을 방문할 때 마지막 인덱스가 가장 빨리 변합니다. 하지만, 포트란 연속 배열에서는, 첫 번째 인덱스가 가장 빨리 변합니다.

coroutine (코루틴)
코루틴은 서브루틴의 더 일반화된 형태입니다. 서브루틴은 한 지점에서 진입하고 다른 지점에서 탈출합니다. 코루틴은 여러 다른 지점에서 진입하고, 탈출하고, 재개할 수 있습니다. 이것들은 `async def` 문으로 구현할 수 있습니다. **PEP 492**를 보세요.

coroutine function (코루틴 함수)
코루틴 객체를 돌려주는 함수. 코루틴 함수는 `async def` 문으로 정의될 수 있고, `await` 와 `async for`와 `async with` 키워드를 포함할 수 있습니다. 이것들은 **PEP 492** 에 의해 도입되었습니다.

CPython
파이썬 프로그래밍 언어의 규범적인 구현인데, python.org에서 배포됩니다. 이 구현을 Jython 이나 IronPython 과 같은 다른 것들과 구별할 필요가 있을 때 용어 "CPython" 이 사용됩니다.

decorator (데코레이터)
다른 함수를 돌려주는 함수인데, 보통 `@wrapper` 문법을 사용한 함수 변환으로 적용됩니다. 데코레이터의 흔한 예는 `classmethod()` 과 `staticmethod()` 입니다.

데코레이터 문법은 단지 편의 문법일 뿐입니다. 다음 두 함수 정의는 의미상으로 동등합니다:

```
def f(arg):
    ...
f = staticmethod(f)

@staticmethod
def f(arg):
    ...
```

같은 개념이 클래스에도 존재하지만, 덜 자주 쓰입니다. 데코레이터에 대한 더 자세한 내용은 함수 정의 와 클래스 정의 의 설명서를 보면 됩니다.

descriptor (디스크립터)
Any object which defines the methods `__get__()`, `__set__()`, or `__delete__()`. When a class attribute is a descriptor, its special binding behavior is triggered upon attribute lookup. Normally, using *a.b* to get, set or delete an attribute looks up the object named *b* in the class dictionary for *a*, but if *b* is a descriptor, the respective descriptor method gets called. Understanding descriptors is a key to a deep understanding of Python because they are the basis for many features including functions, methods, properties, class methods, static methods, and reference to super classes.

디스크립터의 메서드들에 대한 자세한 내용은 descriptors나 디스크립터 사용법 안내서에 나옵니다.

dictionary (딕셔너리)
An associative array, where arbitrary keys are mapped to values. The keys can be any object with `__hash__()` and `__eq__()` methods. Called a hash in Perl.

dictionary comprehension (딕셔너리 컴프리헨션)
이터러블에 있는 요소 전체나 일부를 처리하고 결과를 담은 딕셔너리를 반환하는 간결한 방법. `results = {n: n ** 2 for n in range(10)}`은 값 `n ** 2`에 매핑된 키 `n`을 포함하는 딕셔너리를 생성합니다. comprehensions을 참조하십시오.

dictionary view (딕셔너리 뷰)
`dict.keys()`, `dict.values()`, `dict.items()` 메서드가 돌려주는 객체들을 딕셔너리 뷰라고 부릅니다. 이것들은 딕셔너리 항목들에 대한 동적인 뷰를 제공하는데, 딕셔너리가 변경될 때, 뷰가 이 변화를 반영한다는 뜻입니다. 딕셔너리 뷰를 완전한 리스트로 바꾸려면 `list(dictview)`를 사용하면 됩니다. dict-views를 보세요.

docstring (독스트링)
A string literal which appears as the first expression in a class, function or module. While ignored when the suite is executed, it is recognized by the compiler and put into the `__doc__` attribute of the enclosing class, function or module. Since it is available via introspection, it is the canonical place for documentation of the object.

duck-typing (덕 타이핑)
올바른 인터페이스를 가졌는지 판단하는데 객체의 형을 보지 않는 프로그래밍 스타일; 대신, 단순히 메서드나 어트리뷰트가 호출되거나 사용됩니다 ("오리처럼 보이고 오리처럼 꽥꽥댄다면, 그것은 오리다.") 특정한 형 대신에 인터페이스를 강조함으로써, 잘 설계된 코드는 다형적인 치환을 허락함으로써 유연성을 개선할 수 있습니다. 덕 타이핑은 `type()` 이나 `isinstance()` 을 사용한 검사를 피합니다. (하지만, 덕 타이핑이 추상 베이스 클래스 로 보완될 수 있음에 유의해야 합니다.) 대신에, `hasattr()` 검사나 *EAFP* 프로그래밍을 씁니다.

EAFP
허락보다는 용서를 구하기가 쉽다 (Easier to ask for forgiveness than permission). 이 흔히 볼 수 있는 파이썬 코딩 스타일은, 올바른 키나 어트리뷰트의 존재를 가정하고, 그 가정이 틀리면 예외를 잡습니다. 이 깔끔하고 빠른 스타일은 많은 `try`와 `except` 문의 존재로 특징지어집니다. 이 테크닉은 C와 같은 다른 많은 언어에서 자주 사용되는 *LBYL* 스타일과 대비됩니다.

expression (표현식)
어떤 값으로 구해질 수 있는 문법적인 조각. 다른 말로 표현하면, 표현식은 리터럴, 이름, 어트리뷰트 액세스, 연산자, 함수들과 같은 값을 돌려주는 표현 요소들을 쌓아 올린 것입니다. 다른 많은 언어와 대조적으로, 모든 언어 구성물들이 표현식인 것은 아닙니다. `while`처럼, 표현식으로 사용할 수 없는 문장 들이 있습니다. 대입 또한 문장이고, 표현식이 아닙니다.

extension module (확장 모듈)
C 나 C++로 작성된 모듈인데, 파이썬의 C API를 사용해서 핵심이나 사용자 코드와 상호 작용합니다.

f-string (f-문자열)
`'f'` 나 `'F'` 를 앞에 붙인 문자열 리터럴들을 흔히 "f-문자열" 이라고 부르는데, 포맷 문자열 리터럴의 줄임말입니다. **PEP 498** 을 보세요.

file object (파일 객체)
An object exposing a file-oriented API (with methods such as `read()` or `write()`) to an underlying resource. Depending on the way it was created, a file object can mediate access to a real on-disk file or to another type of storage or communication device (for example standard input/output, in-memory buffers, sockets, pipes, etc.). File objects are also called *file-like objects* or *streams*.

실제로는 세 부류의 파일 객체들이 있습니다. 날(raw) 바이너리 파일, 버퍼드(buffered) 바이너리 파일, 텍스트 파일. 이들의 인터페이스는 `io` 모듈에서 정의됩니다. 파일 객체를 만드는 규범적인 방법은 `open()` 함수를 쓰는 것입니다.

file-like object (파일류 객체)
파일 객체 의 비슷한 말.

filesystem encoding and error handler
Encoding and error handler used by Python to decode bytes from the operating system and encode Unicode to the operating system.

The filesystem encoding must guarantee to successfully decode all bytes below 128. If the file system encoding fails to provide this guarantee, API functions can raise `UnicodeError`.

The `sys.getfilesystemencoding()` and `sys.getfilesystemencodeerrors()` functions can be used to get the filesystem encoding and error handler.

The *filesystem encoding and error handler* are configured at Python startup by the *`PyConfig_Read()`* function: see *`filesystem_encoding`* and *`filesystem_errors`* members of *`PyConfig`*.

See also the *locale encoding*.

finder (파인더)
임포트될 모듈을 위한 로더 를 찾으려고 시도하는 객체.

There are two types of finder: *meta path finders* for use with `sys.meta_path`, and *path entry finders* for use with `sys.path_hooks`.

See finders-and-loaders and `importlib` for much more detail.

floor division (정수 나눗셈)
가장 가까운 정수로 내림하는 수학적 나눗셈. 정수 나눗셈 연산자는 `//` 다. 예를 들어, 표현식 `11 // 4` 의 값은 `2`가 되지만, 실수 나눗셈은 `2.75`를 돌려줍니다. `(-11) // 4` 가 `-2.75`를 내림 한 `-3`이 됨에 유의해야 합니다. **PEP 238**을 보세요.

function (함수)
호출자에게 어떤 값을 돌려주는 일련의 문장들. 없거나 그 이상의 인자 가 전달될 수 있는데, 바디의 실행에 사용될 수 있습니다. 매개변수 와 메서드 와 function 섹션도 보세요.

function annotation (함수 어노테이션)
함수 매개변수나 반환 값의 어노테이션.

함수 어노테이션은 일반적으로 형 힌트 로 사용됩니다: 예를 들어, 이 함수는 두 개의 `int` 인자를 받아들일 것으로 기대되고, 동시에 `int` 반환 값을 줄 것으로 기대됩니다:

```
def sum_two_numbers(a: int, b: int) -> int:
   return a + b
```

함수 어노테이션 문법은 function 절에서 설명합니다.

See *variable annotation* and **PEP 484**, which describe this functionality. Also see annotations-howto for best practices on working with annotations.

__future__
A future statement, `from __future__ import <feature>`, directs the compiler to compile the current module using syntax or semantics that will become standard in a future release of Python. The `__future__` module documents the possible values of *feature*. By importing this module and evaluating its variables, you can see when a new feature was first added to the language and when it will (or did) become the default:

```
>>> import __future__
>>> __future__.division
_Feature((2, 2, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0), 8192)
```

garbage collection (가비지 수거)
더 사용되지 않는 메모리를 반납하는 절차. 파이썬은 참조 횟수 추적과 참조 순환을 감지하고 끊을 수 있는 순환 가비지 수거기를 통해 가비지 수거를 수행합니다. 가비지 수거기는 `gc` 모듈을 사용해서 제어할 수 있습니다.

generator (제너레이터)
제너레이터 이터레이터 를 돌려주는 함수. 일반 함수처럼 보이는데, 일련의 값들을 만드는 `yield` 표현식을 포함한다는 점이 다릅니다. 이 값들은 for-루프로 사용하거나 `next()` 함수로 한 번에 하나씩 꺼낼 수 있습니다.

보통 제너레이터 함수를 가리키지만, 어떤 문맥에서는 제너레이터 이터레이터 를 가리킵니다. 의도하는 의미가 명확하지 않은 경우는, 완전한 용어를 써서 모호함을 없앱니다.

generator iterator (제너레이터 이터레이터)
제너레이터 함수가 만드는 객체.

Each `yield` temporarily suspends processing, remembering the execution state (including local variables and pending try-statements). When the *generator iterator* resumes, it picks up where it left off (in contrast to functions which start fresh on every invocation).

generator expression (제너레이터 표현식)
이터레이터를 돌려주는 표현식. 루프 변수와 범위를 정의하는 `for` 절과 생략 가능한 `if` 절이 뒤에 붙는 일반 표현식 처럼 보입니다. 결합한 표현식은 둘러싼 함수를 위한 값들을 만들어냅니다:

```
>>> sum(i*i for i in range(10))         # sum of squares 0, 1, 4, ... 81
285
```

generic function (제네릭 함수)
같은 연산을 서로 다른 형들에 대해 구현한 여러 함수로 구성된 함수. 호출 때 어떤 구현이 사용될지는 디스패치 알고리즘에 의해 결정됩니다.

싱글 디스패치 용어집 항목과 `functools.singledispatch()` 데코레이터와 **PEP 443**도 보세요.

generic type (제네릭 형)
A *type* that can be parameterized; typically a container class such as `list` or `dict`. Used for *type hints* and *annotations*.

For more details, see generic alias types, **PEP 483**, **PEP 484**, **PEP 585**, and the `typing` module.

GIL
전역 인터프리터 록 을 보세요.

global interpreter lock (전역 인터프리터 록)
한 번에 오직 하나의 스레드가 파이썬 바이트 코드 를 실행하도록 보장하기 위해 *CPython* 인터프리터가 사용하는 메커니즘. (`dict`와 같은 중요한 내장형들을 포함하는) 객체 모델이 묵시적으로 동시 액세스에 대해 안전하도록 만들어서 CPython 구현을 단순하게 만듭니다. 인터프리터 전체를 잠그는 것은 인터프리터를 다중스레드화하기 쉽게 만드는 대신, 다중 프로세서 기계가 제공하는 병렬성의 많은 부분을 희생합니다.

However, some extension modules, either standard or third-party, are designed so as to release the GIL when doing computationally intensive tasks such as compression or hashing. Also, the GIL is always released when doing I/O.

(훨씬 더 미세하게 공유 데이터를 잠그는) "스레드에 자유로운(free-threaded)" 인터프리터를 만들고자 하는 과거의 노력은 성공적이지 못했는데, 흔한 단일 프로세서 경우의 성능 저하가 심하기 때문입니다. 이 성능 이슈를 극복하는 것은 구현을 훨씬 복잡하게 만들어서 유지 비용이 더 들어갈 것으로 여겨지고 있습니다.

hash-based pyc (해시 기반 pyc)
유효성을 판별하기 위해 해당 소스 파일의 최종 수정 시간이 아닌 해시를 사용하는 바이트 코드 캐시 파일. pyc-invalidation을 참조하세요.

hashable (해시 가능)
An object is *hashable* if it has a hash value which never changes during its lifetime (it needs a `__hash__()` method), and can be compared to other objects (it needs an `__eq__()` method). Hashable objects which compare equal must have the same hash value.

해시 가능성은 객체를 딕셔너리의 키나 집합의 멤버로 사용할 수 있게 하는데, 이 자료 구조들이 내부적으로 해시값을 사용하기 때문입니다.

대부분 파이썬의 불변 내장 객체들은 해시 가능합니다; (리스트나 딕셔너리 같은) 가변 컨테이너들은 그렇지 않습니다; (튜플이나 frozenset 같은) 불변 컨테이너들은 그들의 요소들이 해시 가능할 때만 해시 가능합니다. 사용자 정의 클래스의 인스턴스 객체들은 기본적으로 해시 가능합니다. (자기 자신을 제외하고는) 모두 다르다고 비교되고, 해시값은 `id()`로 부터 만들어집니다.

IDLE
An Integrated Development and Learning Environment for Python. idle is a basic editor and interpreter environment which ships with the standard distribution of Python.

immortal
Immortal objects are a CPython implementation detail introduced in **PEP 683**.

If an object is immortal, its *reference count* is never modified, and therefore it is never deallocated while the interpreter is running. For example, `True` and `None` are immortal in CPython.

immutable (불변)
고정된 값을 갖는 객체. 불변 객체는 숫자, 문자열, 튜플을 포함합니다. 이런 객체들은 변경될 수 없습니다. 새 값을 저장하려면 새 객체를 만들어야 합니다. 변하지 않는 해시값이 있어야 하는 곳에서 중요한 역할을 합니다, 예를 들어, 딕셔너리의 키.

import path (임포트 경로)
경로 기반 파인더 가 임포트 할 모듈을 찾기 위해 검색하는 장소들 (또는 경로 엔트리) 의 목록. 임포트 하는 동안, 이 장소들의 목록은 보통 `sys.path` 로부터 옵니다, 하지만 서브 패키지의 경우 부모 패키지의 `__path__` 어트리뷰트로부터 올 수도 있습니다.

importing (임포팅)
한 모듈의 파이썬 코드가 다른 모듈의 파이썬 코드에서 사용될 수 있도록 하는 절차.

importer (임포터)
모듈을 찾기도 하고 로드 하기도 하는 객체; 동시에 파인더 이자 로더 객체입니다.

interactive (대화형)
파이썬은 대화형 인터프리터를 갖고 있는데, 인터프리터 프롬프트에서 문장과 표현식을 입력할 수 있고, 즉각 실행된 결과를 볼 수 있다는 뜻입니다. 인자 없이 단지 `python`을 실행하세요 (컴퓨터의 주메뉴에서 선택하는 것도 가능할 수 있습니다). 새 아이디어를 검사하거나 모듈과 패키지를 들여다보는 매우 강력한 방법입니다 (`help(x)` 를 기억하세요).

interpreted (인터프리티드)
바이트 코드 컴파일러의 존재 때문에 그 구분이 흐릿해지기는 하지만, 파이썬은 컴파일 언어가 아니라 인터프리터 언어입니다. 이것은 명시적으로 실행 파일을 만들지 않고도, 소스 파일을 직접 실행할 수 있다는 뜻입니다. 그 프로그램이 좀 더 천천히 실행되기는 하지만, 인터프리터 언어는 보통 컴파일 언어보다 짧은 개발/디버깅 주기를 갖습니다. 대화형 도 보세요.

interpreter shutdown (인터프리터 종료)
종료하라는 요청을 받을 때, 파이썬 인터프리터는 특별한 시기에 진입하는데, 모듈이나 여러 가지 중요한 내부 구조들과 같은 모든 할당된 자원들을 단계적으로 반납합니다. 또한, 가비지 수거기 를 여러 번 호출합니다. 사용자 정의 파괴자나 weakref 콜백에 있는 코드들의 실행을 시작시킬 수 있습니다. 종료 시기 동안 실행되는 코드는 다양한 예외들을 만날 수 있는데, 그것이 의존하는 자원들이 더 기능하지 않을 수 있기 때문입니다 (흔한 예는 라이브러리 모듈이나 경고 장치들입니다).

인터프리터 종료의 주된 원인은 실행되는 `__main__` 모듈이나 스크립트가 실행을 끝내는 것입니다.

iterable (이터러블)
An object capable of returning its members one at a time. Examples of iterables include all sequence types (such as `list`, `str`, and `tuple`) and some non-sequence types like `dict`, *file objects*, and objects of any classes you define with an `__iter__()` method or with a `__getitem__()` method that implements *sequence* semantics.

Iterables can be used in a `for` loop and in many other places where a sequence is needed (`zip()`, `map()`, ···). When an iterable object is passed as an argument to the built-in function `iter()`, it returns an iterator for the object. This iterator is good for one pass over the set of values. When using iterables, it is usually not necessary to call `iter()` or deal with iterator objects yourself. The `for` statement does that automatically for you, creating a temporary unnamed variable to hold the iterator for the duration of the loop. See also *iterator*, *sequence*, and *generator*.

iterator (이터레이터)
An object representing a stream of data. Repeated calls to the iterator' s `__next__()` method (or passing it to the built-in function `next()`) return successive items in the stream. When no more data are available a `StopIteration` exception is raised instead. At this point, the iterator object is exhausted and any further calls to its `__next__()` method just raise `StopIteration` again. Iterators are required to have an `__iter__()` method that returns the iterator object itself so every iterator is also iterable and may be used in most places where other iterables are accepted. One notable exception is code which attempts multiple iteration passes. A container object (such as a `list`) produces a fresh new iterator each time you pass it to the `iter()` function or use it in a `for` loop. Attempting this with an iterator will just return the same exhausted iterator object used in the previous iteration pass, making it appear like an empty container.

typeiter 에 더 자세한 내용이 있습니다.

CPython 구현 상세: CPython does not consistently apply the requirement that an iterator define `__iter__()`.

key function (키 함수)
키 함수 또는 콜레이션(collation) 함수는 정렬(sorting)이나 배열(ordering)에 사용되는 값을 돌려주는 콜러블입니다. 예를 들어, `locale.strxfrm()` 은 로케일 특정 방식을 따르는 정렬 키를 만드는 데 사용됩니다.

파이썬의 많은 도구가 요소들이 어떻게 순서 지어지고 묶이는지를 제어하기 위해 키 함수를 받아들입니다. 이런 것들에는 `min()`, `max()`, `sorted()`, `list.sort()`, `heapq.merge()`, `heapq.nsmallest()`, `heapq.nlargest()`, `itertools.groupby()` 이 있습니다.

There are several ways to create a key function. For example. the `str.lower()` method can serve as a key function for case insensitive sorts. Alternatively, a key function can be built from a `lambda` expression such as `lambda r: (r[0], r[2])`. Also, `operator.attrgetter()`, `operator.itemgetter()`, and `operator.methodcaller()` are three key function constructors. See the Sorting HOW TO for examples of how to create and use key functions.

keyword argument (키워드 인자)
인자 를 보세요.

lambda (람다)
호출될 때 값이 구해지는 하나의 표현식 으로 구성된 이름 없는 인라인 함수. 람다 함수를 만드는 문법은 `lambda [parameters]: expression` 입니다.

LBYL
뛰기 전에 보라 (Look before you leap). 이 코딩 스타일은 호출이나 조회를 하기 전에 명시적으로 사전 조건들을 검사합니다. 이 스타일은 *EAFP* 접근법과 대비되고, 많은 `if` 문의 존재로 특징지어집니다.

다중 스레드 환경에서, LBYL 접근법은 "보기"와 "뛰기" 간에 경쟁 조건을 만들게 될 위험이 있습니다. 예를 들어, 코드 `if key in mapping: return mapping[key]` 는 검사 후에, 하지만 조회 전에, 다른 스레드가 *key*를 *mapping*에서 제거하면 실패할 수 있습니다. 이런 이슈는 록이나 EAFP 접근법을 사용함으로써 해결될 수 있습니다.

lexical analyzer
Formal name for the *tokenizer*; see *token*.

list (리스트)
A built-in Python *sequence*. Despite its name it is more akin to an array in other languages than to a linked list since access to elements is $O(1)$.

list comprehension (리스트 컴프리헨션)
시퀀스의 요소들 전부 또는 일부를 처리하고 그 결과를 리스트로 돌려주는 간결한 방법. `result = ['{:#04x}'.format(x) for x in range(256) if x % 2 == 0]` 는 0에서 255 사이에 있는 짝수들의 16진수 (0x..) 들을 포함하는 문자열의 리스트를 만듭니다. `if` 절은 생략할 수 있습니다. 생략하면, `range(256)` 에 있는 모든 요소가 처리됩니다.

loader (로더)
An object that loads a module. It must define the `exec_module()` and `create_module()` methods to implement the `Loader` interface. A loader is typically returned by a *finder*. See also:

- finders-and-loaders
- `importlib.abc.Loader`
- **PEP 302**

locale encoding
On Unix, it is the encoding of the LC_CTYPE locale. It can be set with `locale.setlocale(locale.LC_CTYPE, new_locale)`.

On Windows, it is the ANSI code page (ex: `"cp1252"`).

On Android and VxWorks, Python uses `"utf-8"` as the locale encoding.

`locale.getencoding()` can be used to get the locale encoding.

See also the *filesystem encoding and error handler*.

magic method (매직 메서드)
특수 메서드 의 비공식적인 비슷한 말.

mapping (매핑)
A container object that supports arbitrary key lookups and implements the methods specified in the `collections.abc.Mapping` or `collections.abc.MutableMapping` abstract base classes. Examples include `dict`, `collections.defaultdict`, `collections.OrderedDict` and `collections.Counter`.

meta path finder (메타 경로 파인더)
`sys.meta_path` 의 검색이 돌려주는 파인더. 메타 경로 파인더는 경로 엔트리 파인더 와 관련되어 있기는 하지만 다릅니다.

메타 경로 파인더가 구현하는 메서드들에 대해서는 `importlib.abc.MetaPathFinder` 를 보면 됩니다.

metaclass (메타 클래스)
클래스의 클래스. 클래스 정의는 클래스 이름, 클래스 딕셔너리, 베이스 클래스들의 목록을 만듭니다. 메타 클래스는 이 세 인자를 받아서 클래스를 만드는 책임을 집니다. 대부분의 객체 지향형 프로그래밍 언어들은 기본 구현을 제공합니다. 파이썬을 특별하게 만드는 것은 커스텀 메타 클래스를 만들 수 있다는 것입니다. 대부분 사용자에게는 이 도구가 전혀 필요 없지만, 필요가 생길 때, 메타 클래스는 강력하고 우아한 해법을 제공합니다. 어트리뷰트 액세스의 로깅(logging), 스레드 안전성의 추가, 객체 생성 추적, 싱글톤 구현과 많은 다른 작업에 사용됐습니다.

metaclasses 에서 더 자세한 내용을 찾을 수 있습니다.

method (메서드)
클래스 바디 안에서 정의되는 함수. 그 클래스의 인스턴스의 어트리뷰트로서 호출되면, 그 메서드는 첫 번째 인자 (보통 `self` 라고 불린다) 로 인스턴스 객체를 받습니다. 함수 와 중첩된 스코프 를 보세요.

method resolution order (메서드 결정 순서)
Method Resolution Order is the order in which base classes are searched for a member during lookup. See python_2.3_mro for details of the algorithm used by the Python interpreter since the 2.3 release.

module (모듈)
파이썬 코드의 조직화 단위를 담당하는 객체. 모듈은 임의의 파이썬 객체들을 담는 이름 공간을 갖습니다. 모듈은 임포팅 절차에 의해 파이썬으로 로드됩니다.

패키지 도 보세요.

module spec (모듈 스펙)
모듈을 로드하는데 사용되는 임포트 관련 정보들을 담고 있는 이름 공간. `importlib.machinery.ModuleSpec` 의 인스턴스.

See also module-specs.

MRO
메서드 결정 순서 를 보세요.

mutable (가변)
가변 객체는 값이 변할 수 있지만 `id()` 는 일정하게 유지합니다. 불변 도 보세요.

named tuple (네임드 튜플)
"named tuple(네임드 튜플)" 이라는 용어는 튜플에서 상속하고 이름 붙은 어트리뷰트를 사용하여 인덱스 할 수 있는 요소에 액세스 할 수 있는 모든 형이나 클래스에 적용됩니다. 형이나 클래스에는 다른 기능도 있을 수 있습니다.

`time.localtime()` 과 `os.stat()` 가 반환한 값을 포함하여, 여러 내장형이 네임드 튜플입니다. 또 다른 예는 `sys.float_info` 입니다:

```
>>> sys.float_info[1]                    # indexed access
1024
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
>>> sys.float_info.max_exp              # named field access
1024
>>> isinstance(sys.float_info, tuple)   # kind of tuple
True
```

Some named tuples are built-in types (such as the above examples). Alternatively, a named tuple can be created from a regular class definition that inherits from `tuple` and that defines named fields. Such a class can be written by hand, or it can be created by inheriting `typing.NamedTuple`, or with the factory function `collections.namedtuple()`. The latter techniques also add some extra methods that may not be found in hand-written or built-in named tuples.

namespace (이름 공간)
변수가 저장되는 장소. 이름 공간은 딕셔너리로 구현됩니다. 객체에 중첩된 이름 공간 (메서드 에서) 뿐만 아니라 지역, 전역, 내장 이름 공간이 있습니다. 이름 공간은 이름 충돌을 방지해서 모듈성을 지원합니다. 예를 들어, 함수 `builtins.open` 과 `os.open()` 은 그들의 이름 공간에 의해 구별됩니다. 또한, 이름 공간은 어떤 모듈이 함수를 구현하는지를 분명하게 만들어서 가독성과 유지 보수성에 도움을 줍니다. 예를 들어, `random.seed()` 또는 `itertools.islice()` 라고 쓰면 그 함수들이 각각 `random` 과 `itertools` 모듈에 의해 구현되었음이 명확해집니다.

namespace package (이름 공간 패키지)
A *package* which serves only as a container for subpackages. Namespace packages may have no physical representation, and specifically are not like a *regular package* because they have no `__init__.py` file.

Namespace packages allow several individually installable packages to have a common parent package. Otherwise, it is recommended to use a *regular package*.

For more information, see **PEP 420** and reference-namespace-package.

모듈 도 보세요.

nested scope (중첩된 스코프)
둘러싼 정의에서 변수를 참조하는 능력. 예를 들어, 다른 함수 내부에서 정의된 함수는 바깥 함수에 있는 변수들을 참조할 수 있습니다. 중첩된 스코프는 기본적으로는 참조만 가능할 뿐, 대입은 되지 않는다는 것에 주의해야 합니다. 지역 변수들은 가장 내부의 스코프에서 읽고 씁니다. 마찬가지로, 전역 변수들은 전역 이름 공간에서 읽고 씁니다. `nonlocal` 은 바깥 스코프에 쓰는 것을 허락합니다.

new-style class (뉴스타일 클래스)
Old name for the flavor of classes now used for all class objects. In earlier Python versions, only new-style classes could use Python' s newer, versatile features like `__slots__`, descriptors, properties, `__getattribute__()`, class methods, and static methods.

object (객체)
상태 (어트리뷰트나 값) 를 갖고 동작 (메서드) 이 정의된 모든 데이터. 또한, 모든 뉴스타일 클래스 의 최종적인 베이스 클래스입니다.

package (패키지)
A Python *module* which can contain submodules or recursively, subpackages. Technically, a package is a Python module with a `__path__` attribute.

정규 패키지 와 이름 공간 패키지 도 보세요.

parameter (매개변수)
함수 (또는 메서드) 정의에서 함수가 받을 수 있는 인자 (또는 어떤 경우 인자들) 를 지정하는 이름 붙은 엔티티. 다섯 종류의 매개변수가 있습니다:

- 위치-키워드 (*positional-or-keyword*): 위치 인자 나 키워드 인자 로 전달될 수 있는 인자를 지정합니다. 이것이 기본 형태의 매개변수입니다, 예를 들어 다음에서 *foo* 와 *bar*:

```
def func(foo, bar=None): ...
```

- 위치-전용 (*positional-only*): 위치로만 제공될 수 있는 인자를 지정합니다. 위치 전용 매개변수는 함수 정의의 매개변수 목록에 / 문자를 포함하고 그 뒤에 정의할 수 있습니다, 예를 들어 다음에서 *posonly1* 과 *posonly2*:

```
def func(posonly1, posonly2, /, positional_or_keyword): ...
```

- 키워드-전용 (*keyword-only*): 키워드로만 제공될 수 있는 인자를 지정합니다. 키워드-전용 매개변수는 함수 정의의 매개변수 목록에서 앞에 하나의 가변-위치 매개변수나 `*`를 그대로 포함해서 정의할 수 있습니다. 예를 들어, 다음에서 *kw_only1* 와 *kw_only2*:

```
def func(arg, *, kw_only1, kw_only2): ...
```

- 가변-위치 (*var-positional*): (다른 매개변수들에 의해서 이미 받아들여진 위치 인자들에 더해) 제공될 수 있는 위치 인자들의 임의의 시퀀스를 지정합니다. 이런 매개변수는 매개변수 이름에 `*` 를 앞에 붙여서 정의될 수 있습니다, 예를 들어 다음에서 *args*:

```
def func(*args, **kwargs): ...
```

- 가변-키워드 (*var-keyword*): (다른 매개변수들에 의해서 이미 받아들여진 키워드 인자들에 더해) 제공될 수 있는 임의의 개수 키워드 인자들을 지정합니다. 이런 매개변수는 매개변수 이름에 `**`를 앞에 붙여서 정의될 수 있습니다, 예를 들어 위의 예에서 *kwargs*.

매개변수는 선택적 인자들을 위한 기본값뿐만 아니라 선택적이거나 필수 인자들을 지정할 수 있습니다.

인자 용어집 항목, 인자와 매개변수의 차이에 나오는 FAQ 질문, `inspect.Parameter` 클래스, function 절, **PEP 362**도 보세요.

path entry (경로 엔트리)
경로 기반 파인더 가 임포트 할 모듈들을 찾기 위해 참고하는 임포트 경로 상의 하나의 장소.

path entry finder (경로 엔트리 파인더)
`sys.path_hooks` 에 있는 콜러블 (즉, 경로 엔트리 훅) 이 돌려주는 파인더 인데, 주어진 경로 엔트리로 모듈을 찾는 방법을 알고 있습니다.

경로 엔트리 파인더들이 구현하는 메서드들은 `importlib.abc.PathEntryFinder` 에 나옵니다.

path entry hook (경로 엔트리 훅)
A callable on the `sys.path_hooks` list which returns a *path entry finder* if it knows how to find modules on a specific *path entry*.

path based finder (경로 기반 파인더)
기본 메타 경로 파인더들 중 하나인데, 임포트 경로 에서 모듈을 찾습니다.

path-like object (경로류 객체)
파일 시스템 경로를 나타내는 객체. 경로류 객체는 경로를 나타내는 `str` 나 `bytes` 객체이거나 `os.PathLike` 프로토콜을 구현하는 객체입니다. `os.PathLike` 프로토콜을 지원하는 객체는 `os.fspath()` 함수를 호출해서 `str` 나 `bytes` 파일 시스템 경로로 변환될 수 있습니다; 대신 `os.fsdecode()` 와 `os.fsencode()` 는 각각 `str` 나 `bytes` 결과를 보장하는데 사용될 수 있습니다. **PEP 519**로 도입되었습니다.

PEP
파이썬 개선 제안. PEP는 파이썬 커뮤니티에 정보를 제공하거나 파이썬 또는 그 프로세스 또는 환경에 대한 새로운 기능을 설명하는 설계 문서입니다. PEP는 제안된 기능에 대한 간결한 기술 사양 및 근거를 제공해야 합니다.

PEP는 주요 새로운 기능을 제안하고 문제에 대한 커뮤니티 입력을 수집하며 파이썬에 들어간 설계 결정을 문서로 만들기 위한 기본 메커니즘입니다. PEP 작성자는 커뮤니티 내에서 합의를 구축하고 반대 의견을 문서화 할 책임이 있습니다.

PEP 1 참조하세요.

portion (포션)
PEP 420 에서 정의한 것처럼, 이름 공간 패키지에 이바지하는 하나의 디렉터리에 들어있는 파일들의 집합 (zip 파일에 저장되는 것도 가능합니다).

positional argument (위치 인자)
인자 를 보세요.

provisional API (잠정 API)
잠정 API는 표준 라이브러리의 과거 호환성 보장으로부터 신중히 제외된 것입니다. 인터페이스의 큰 변화가 예상되지는 않지만, 잠정적이라고 표시되는 한, 코어 개발자들이 필요하다고 생각한다면 과거 호환성이 유지되지 않는 변경이 일어날 수 있습니다. 그런 변경은 불필요한 방식으로 일어나지는 않을 것입니다 — API를 포함하기 전에 놓친 중대하고 근본적인 결함이 발견된 경우에만 일어날 것입니다.

잠정 API에서조차도, 과거 호환성이 유지되지 않는 변경은 "최후의 수단"으로 여겨집니다 - 모든 식별된 문제들에 대해 과거 호환성을 유지하는 해법을 찾으려는 모든 시도가 선행됩니다.

이 절차는 표준 라이브러리가 오랜 시간 동안 잘못된 설계 오류에 발목 잡히지 않고 발전할 수 있도록 만듭니다. 더 자세한 내용은 **PEP 411**을 보면 됩니다.

provisional package (잠정 패키지)
잠정 *API* 를 보세요.

Python 3000 (파이썬 3000)
파이썬 3.x 배포 라인의 별명 (버전 3의 배포가 먼 미래의 이야기던 시절에 만들어진 이름이다.) 이것을 "Py3k" 로 줄여 쓰기도 합니다.

Pythonic (파이썬다운)
다른 언어들에서 일반적인 개념들을 사용해서 코드를 구현하는 대신, 파이썬 언어에서 가장 자주 사용되는 이디엄들을 가까이 따르는 아이디어나 코드 조각. 예를 들어, 파이썬에서 자주 쓰는 이디엄은 `for` 문을 사용해서 이터러블의 모든 요소로 루핑하는 것입니다. 다른 많은 언어에는 이런 종류의 구성물이 없으므로, 파이썬에 익숙하지 않은 사람들은 대신에 숫자 카운터를 사용하기도 합니다:

```
for i in range(len(food)):
    print(food[i])
```

더 깔끔한, 파이썬다운 방법은 이렇습니다:

```
for piece in food:
    print(piece)
```

qualified name (정규화된 이름)
모듈의 전역 스코프에서 모듈에 정의된 클래스, 함수, 메서드에 이르는 "경로"를 보여주는 점으로 구분된 이름. **PEP 3155** 에서 정의됩니다. 최상위 함수와 클래스의 경우에, 정규화된 이름은 객체의 이름과 같습니다:

```
>>> class C:
...     class D:
...         def meth(self):
...             pass
...
>>> C.__qualname__
'C'
>>> C.D.__qualname__
'C.D'
>>> C.D.meth.__qualname__
'C.D.meth'
```

모듈을 가리키는데 사용될 때, 완전히 정규화된 이름(*fully qualified name*)은 모든 부모 패키지들을 포함해서 모듈로 가는 점으로 분리된 이름을 의미합니다, 예를 들어, `email.mime.text`:

```
>>> import email.mime.text
>>> email.mime.text.__name__
'email.mime.text'
```

reference count (참조 횟수)
The number of references to an object. When the reference count of an object drops to zero, it is deallocated. Some objects are "immortal" and have reference counts that are never modified, and therefore the objects are

never deallocated. Reference counting is generally not visible to Python code, but it is a key element of the *CPython* implementation. Programmers can call the `sys.getrefcount()` function to return the reference count for a particular object.

regular package (정규 패키지)
`__init__.py` 파일을 포함하는 디렉터리와 같은 전통적인 패키지.

이름 공간 패키지 도 보세요.

__slots__
클래스 내부의 선언인데, 인스턴스 어트리뷰트들을 위한 공간을 미리 선언하고 인스턴스 딕셔너리를 제거함으로써 메모리를 절감하는 효과를 줍니다. 인기 있기는 하지만, 이 테크닉은 올바르게 사용하기가 좀 까다로운 편이라서, 메모리에 민감한 응용 프로그램에서 많은 수의 인스턴스가 있는 특별한 경우로 한정하는 것이 좋습니다.

sequence (시퀀스)
An *iterable* which supports efficient element access using integer indices via the `__getitem__()` special method and defines a `__len__()` method that returns the length of the sequence. Some built-in sequence types are `list`, `str`, `tuple`, and `bytes`. Note that `dict` also supports `__getitem__()` and `__len__()`, but is considered a mapping rather than a sequence because the lookups use arbitrary *hashable* keys rather than integers.

The `collections.abc.Sequence` abstract base class defines a much richer interface that goes beyond just `__getitem__()` and `__len__()`, adding `count()`, `index()`, `__contains__()`, and `__reversed__()`. Types that implement this expanded interface can be registered explicitly using `register()`. For more documentation on sequence methods generally, see Common Sequence Operations.

set comprehension (집합 컴프리헨션)
이터러블에 있는 요소 전체나 일부를 처리하고 결과를 담은 집합을 반환하는 간결한 방법. `results = {c for c in 'abracadabra' if c not in 'abc'}`는 문자열의 집합 `{'r', 'd'}`를 생성합니다. comprehensions을 참조하십시오.

single dispatch (싱글 디스패치)
구현이 하나의 인자의 형에 기초해서 결정되는 제네릭 함수 디스패치의 한 형태.

slice (슬라이스)
보통 시퀀스 의 일부를 포함하는 객체. 슬라이스는 서브 스크립트 표기법을 사용해서 만듭니다. `variable_name[1:3:5]` 처럼, `[]` 안에서 여러 개의 숫자를 콜론으로 분리합니다. 대괄호 (서브 스크립트) 표기법은 내부적으로 `slice` 객체를 사용합니다.

soft deprecated
A soft deprecated API should not be used in new code, but it is safe for already existing code to use it. The API remains documented and tested, but will not be enhanced further.

Soft deprecation, unlike normal deprecation, does not plan on removing the API and will not emit warnings.

See PEP 387: Soft Deprecation.

special method (특수 메서드)
파이썬이 형에 어떤 연산을, 덧셈 같은, 실행할 때 묵시적으로 호출되는 메서드. 이런 메서드는 두 개의 밑줄로 시작하고 끝나는 이름을 갖고 있습니다. 특수 메서드는 specialnames 에 문서로 만들어져 있습니다.

statement (문장)
문장은 스위트 (코드의 "블록(block)") 를 구성하는 부분입니다. 문장은 표현식 이거나 키워드를 사용하는 여러 가지 구조물 중의 하나입니다. 가령 `if`, `while`, `for`.

static type checker
An external tool that reads Python code and analyzes it, looking for issues such as incorrect types. See also *type hints* and the `typing` module.

strong reference
In Python' s C API, a strong reference is a reference to an object which is owned by the code holding the reference. The strong reference is taken by calling *`Py_INCREF()`* when the reference is created and released with *`Py_DECREF()`* when the reference is deleted.

The *Py_NewRef()* function can be used to create a strong reference to an object. Usually, the *Py_DECREF()* function must be called on the strong reference before exiting the scope of the strong reference, to avoid leaking one reference.

See also *borrowed reference*.

text encoding (텍스트 인코딩)
A string in Python is a sequence of Unicode code points (in range `U+0000–U+10FFFF`). To store or transfer a string, it needs to be serialized as a sequence of bytes.

Serializing a string into a sequence of bytes is known as "encoding", and recreating the string from the sequence of bytes is known as "decoding".

There are a variety of different text serialization codecs, which are collectively referred to as "text encodings".

text file (텍스트 파일)
`str` 객체를 읽고 쓸 수 있는 파일 객체. 종종, 텍스트 파일은 실제로는 바이트 지향 데이터스트림을 액세스하고 텍스트 인코딩 을 자동 처리합니다. 텍스트 파일의 예로는 텍스트 모드 (`'r'` 또는 `'w'`) 로 열린 파일, `sys.stdin`, `sys.stdout`, `io.StringIO` 의 인스턴스를 들 수 있습니다.

바이트열류 객체 를 읽고 쓸 수 있는 파일 객체에 대해서는 바이너리 파일 도 참조하세요.

token
A small unit of source code, generated by the lexical analyzer (also called the *tokenizer*). Names, numbers, strings, operators, newlines and similar are represented by tokens.

The `tokenize` module exposes Python' s lexical analyzer. The `token` module contains information on the various types of tokens.

triple-quoted string (삼중 따옴표 된 문자열)
따옴표 (") 나 작은따옴표 (') 세 개로 둘러싸인 문자열. 그냥 따옴표 하나로 둘러싸인 문자열에 없는 기능을 제공하지는 않지만, 여러 가지 이유에서 쓸모가 있습니다. 이스케이프 되지 않은 작은따옴표나 큰따옴표를 문자열 안에 포함할 수 있도록 하고, 연결 문자를 쓰지 않고도 여러 줄에 걸칠 수 있는데, 독스트링을 쓸 때 특히 쓸모 있습니다.

type (형)
The type of a Python object determines what kind of object it is; every object has a type. An object' s type is accessible as its `__class__` attribute or can be retrieved with `type(obj)`.

type alias (형 에일리어스)
형을 식별자에 대입하여 만들어지는 형의 동의어.

형 에일리어스는 형 힌트를 단순화하는 데 유용합니다. 예를 들면:

```
def remove_gray_shades(
        colors: list[tuple[int, int, int]]) -> list[tuple[int, int, int]]:
    pass
```

는 다음과 같이 더 읽기 쉽게 만들 수 있습니다:

```
Color = tuple[int, int, int]

def remove_gray_shades(colors: list[Color]) -> list[Color]:
    pass
```

이 기능을 설명하는 `typing`과 **PEP 484**를 참조하세요.

type hint (형 힌트)
변수, 클래스 어트리뷰트 및 함수 매개변수 나 반환 값의 기대되는 형을 지정하는 어노테이션.

Type hints are optional and are not enforced by Python but they are useful to *static type checkers*. They can also aid IDEs with code completion and refactoring.

지역 변수를 제외하고, 전역 변수, 클래스 어트리뷰트 및 함수의 형 힌트는 `typing.get_type_hints()`를 사용하여 액세스할 수 있습니다.

이 기능을 설명하는 `typing`과 **PEP 484**를 참조하세요.

universal newlines (유니버설 줄 넘김)
다음과 같은 것들을 모두 줄의 끝으로 인식하는, 텍스트 스트림을 해석하는 태도: 유닉스 개행 문자 관례 `'\n'`, 윈도우즈 관례 `'\r\n'`, 예전의 매킨토시 관례 `'\r'`. 추가적인 사용에 관해서는 `bytes.splitlines()` 뿐만 아니라 **PEP 278** 와 **PEP 3116** 도 보세요.

variable annotation (변수 어노테이션)
변수 또는 클래스 어트리뷰트의 어노테이션.

변수 또는 클래스 어트리뷰트에 어노테이션을 달 때 대입은 선택 사항입니다:

```
class C:
    field: 'annotation'
```

변수 어노테이션은 일반적으로 형 힌트로 사용됩니다: 예를 들어, 이 변수는 `int` 값을 가질 것으로 기대됩니다:

```
count: int = 0
```

변수 어노테이션 문법은 섹션 annassign 에서 설명합니다.

See *function annotation*, **PEP 484** and **PEP 526**, which describe this functionality. Also see annotations-howto for best practices on working with annotations.

virtual environment (가상 환경)
파이썬 사용자와 응용 프로그램이, 같은 시스템에서 실행되는 다른 파이썬 응용 프로그램들의 동작에 영향을 주지 않으면서, 파이썬 배포 패키지들을 설치하거나 업그레이드하는 것을 가능하게 하는, 협력적으로 격리된 실행 환경.

`venv` 도 보세요.

virtual machine (가상 기계)
소프트웨어만으로 정의된 컴퓨터. 파이썬의 가상 기계는 바이트 코드 컴파일러가 출력하는 바이트 코드를 실행합니다.

Zen of Python (파이썬 젠)
파이썬 디자인 원리와 철학들의 목록인데, 언어를 이해하고 사용하는 데 도움이 됩니다. 이 목록은 대화형 프롬프트에서 "`import this`" 를 입력하면 보입니다.

APPENDIX B

About this documentation

Python' s documentation is generated from reStructuredText sources using Sphinx, a documentation generator originally created for Python and now maintained as an independent project.

설명서와 이를 위한 툴체인 개발은 파이썬 자체와 마찬가지로 전적으로 자원봉사자의 노력입니다. 기여하고 싶다면, 참여 방법에 대한 정보는 reporting-bugs 페이지를 참고하십시오. 새로운 자원봉사자는 언제나 환영합니다!

다음 분들에게 많은 감사를 드립니다:

- Fred L. Drake, Jr., the creator of the original Python documentation toolset and author of much of the content;
- the Docutils project for creating reStructuredText and the Docutils suite;
- Fredrik Lundh for his Alternative Python Reference project from which Sphinx got many good ideas.

B.1 Contributors to the Python documentation

많은 사람이 파이썬 언어, 파이썬 표준 라이브러리 및 파이썬 설명서에 기여했습니다. 기여자의 부분적인 목록은 파이썬 소스 배포판의 Misc/ACKS 를 참조하십시오.

파이썬이 이런 멋진 설명서를 갖게 된 것은 파이썬 커뮤니티의 입력과 기여 때문입니다 – 감사합니다!

APPENDIX C

역사와 라이센스

C.1 소프트웨어의 역사

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see https://www.cwi.nl) in the Netherlands as a successor of a language called ABC. Guido remains Python' s principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see https://www.cnri.reston.va.us) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations, which became Zope Corporation. In 2001, the Python Software Foundation (PSF, see https://www.python.org/psf/) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation was a sponsoring member of the PSF.

All Python releases are Open Source (see https://opensource.org for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

배포판	파생된 곳	해	소유자	GPL-compatible? (1)
0.9.0 ~ 1.2	n/a	1991-1995	CWI	yes
1.3 ~ 1.5.2	1.2	1995-1999	CNRI	yes
1.6	1.5.2	2000	CNRI	no
2.0	1.6	2000	BeOpen.com	no
1.6.1	1.6	2001	CNRI	yes (2)
2.1	2.0+1.6.1	2001	PSF	no
2.0.1	2.0+1.6.1	2001	PSF	yes
2.1.1	2.1+2.0.1	2001	PSF	yes
2.1.2	2.1.1	2002	PSF	yes
2.1.3	2.1.2	2002	PSF	yes
2.2 이상	2.1.1	2001-현재	PSF	yes

참고

(1) GPL-compatible doesn' t mean that we' re distributing Python under the GPL. All Python licenses, unlike the GPL, let you distribute a modified version without making your changes open source. The GPL-

compatible licenses make it possible to combine Python with other software that is released under the GPL; the others don' t.

(2) According to Richard Stallman, 1.6.1 is not GPL-compatible, because its license has a choice of law clause. According to CNRI, however, Stallman' s lawyer has told CNRI' s lawyer that 1.6.1 is "not incompatible" with the GPL.

Guido의 지도하에 이 배포를 가능하게 만든 많은 외부 자원봉사자들에게 감사드립니다.

C.2 파이썬에 액세스하거나 사용하기 위한 이용 약관

Python software and documentation are licensed under the Python Software Foundation License Version 2.

Starting with Python 3.8.6, examples, recipes, and other code in the documentation are dual licensed under the PSF License Version 2 and the *Zero-Clause BSD license*.

파이썬에 통합된 일부 소프트웨어에는 다른 라이센스가 적용됩니다. 라이센스는 해당 라이센스에 해당하는 코드와 함께 나열됩니다. 이러한 라이센스의 불완전한 목록은 포함된 소프트웨어에 대한 라이센스 및 승인를 참조하십시오.

C.2.1 PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2

```
1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and
   the Individual or Organization ("Licensee") accessing and otherwise using this
   software ("Python") in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSF hereby
   grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce,
   analyze, test, perform and/or display publicly, prepare derivative works,
   distribute, and otherwise use Python alone or in any derivative
   version, provided, however, that PSF's License Agreement and PSF's notice of
   copyright, i.e., "Copyright © 2001-2023 Python Software Foundation; All Rights
   Reserved" are retained in Python alone or in any derivative version
   prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or
   incorporates Python or any part thereof, and wants to make the
   derivative work available to others as provided herein, then Licensee hereby
   agrees to include in any such work a brief summary of the changes made to␣
↪Python.

4. PSF is making Python available to Licensee on an "AS IS" basis.
   PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED.  BY WAY OF
   EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR
   WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE
   USE OF PYTHON WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON
   FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF
   MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON, OR ANY DERIVATIVE
   THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of
   its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship
   of agency, partnership, or joint venture between PSF and Licensee.  This License
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
   Agreement does not grant permission to use PSF trademarks or trade name in a
   trademark sense to endorse or promote products or services of Licensee, or any
   third party.

8. By copying, installing or otherwise using Python, Licensee agrees
   to be bound by the terms and conditions of this License Agreement.
```

C.2.2 BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

```
1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at
   160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization
   ("Licensee") accessing and otherwise using this software in source or binary
   form and its associated documentation ("the Software").

2. Subject to the terms and conditions of this BeOpen Python License Agreement,
   BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license
   to reproduce, analyze, test, perform and/or display publicly, prepare derivative
   works, distribute, and otherwise use the Software alone or in any derivative
   version, provided, however, that the BeOpen Python License is retained in the
   Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an "AS IS" basis.
   BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED.  BY WAY OF
   EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR
   WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE
   USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR
   ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING,
   MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF
   ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate upon a material breach of
   its terms and conditions.

6. This License Agreement shall be governed by and interpreted in all respects
   by the law of the State of California, excluding conflict of law provisions.
   Nothing in this License Agreement shall be deemed to create any relationship of
   agency, partnership, or joint venture between BeOpen and Licensee.  This License
   Agreement does not grant permission to use BeOpen trademarks or trade names in a
   trademark sense to endorse or promote products or services of Licensee, or any
   third party.  As an exception, the "BeOpen Python" logos available at
   http://www.pythonlabs.com/logos.html may be used according to the permissions
   granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be
   bound by the terms and conditions of this License Agreement.
```

C.2.3 CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

```
1. This LICENSE AGREEMENT is between the Corporation for National Research
   Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191
   ("CNRI"), and the Individual or Organization ("Licensee") accessing and
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
   otherwise using Python 1.6.1 software in source or binary form and its
   associated documentation.

2. Subject to the terms and conditions of this License Agreement, CNRI hereby
   grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce,
   analyze, test, perform and/or display publicly, prepare derivative works,
   distribute, and otherwise use Python 1.6.1 alone or in any derivative version,
   provided, however, that CNRI's License Agreement and CNRI's notice of copyright,
   i.e., "Copyright © 1995-2001 Corporation for National Research Initiatives; All
   Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version
   prepared by Licensee.  Alternately, in lieu of CNRI's License Agreement,
   Licensee may substitute the following text (omitting the quotes): "Python 1.6.1
   is made available subject to the terms and conditions in CNRI's License
   Agreement.  This Agreement together with Python 1.6.1 may be located on the
   internet using the following unique, persistent identifier (known as a handle):
   1895.22/1013.  This Agreement may also be obtained from a proxy server on the
   internet using the following URL: http://hdl.handle.net/1895.22/1013".

3. In the event Licensee prepares a derivative work that is based on or
   incorporates Python 1.6.1 or any part thereof, and wants to make the derivative
   work available to others as provided herein, then Licensee hereby agrees to
   include in any such work a brief summary of the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis.  CNRI
   MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED.  BY WAY OF EXAMPLE,
   BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY
   OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF
   PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR
   ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF
   MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE
   THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of
   its terms and conditions.

7. This License Agreement shall be governed by the federal intellectual property
   law of the United States, including without limitation the federal copyright
   law, and, to the extent such U.S. federal law does not apply, by the law of the
   Commonwealth of Virginia, excluding Virginia's conflict of law provisions.
   Notwithstanding the foregoing, with regard to derivative works based on Python
   1.6.1 that incorporate non-separable material that was previously distributed
   under the GNU General Public License (GPL), the law of the Commonwealth of
   Virginia shall govern this License Agreement only as to issues arising under or
   with respect to Paragraphs 4, 5, and 7 of this License Agreement.  Nothing in
   this License Agreement shall be deemed to create any relationship of agency,
   partnership, or joint venture between CNRI and Licensee.  This License Agreement
   does not grant permission to use CNRI trademarks or trade name in a trademark
   sense to endorse or promote products or services of Licensee, or any third
   party.

8. By clicking on the "ACCEPT" button where indicated, or by copying, installing
   or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and
   conditions of this License Agreement.
```

C.2.4 CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

```
Copyright © 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The
Netherlands.  All rights reserved.

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted, provided that
the above copyright notice appear in all copies and that both that copyright
notice and this permission notice appear in supporting documentation, and that
the name of Stichting Mathematisch Centrum or CWI not be used in advertising or
publicity pertaining to distribution of the software without specific, written
prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO
EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT
OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE,
DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS
SOFTWARE.
```

C.2.5 ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON DOCUMENTATION

```
Permission to use, copy, modify, and/or distribute this software for any
purpose with or without fee is hereby granted.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH
REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT,
INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM
LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

C.3 포함된 소프트웨어에 대한 라이센스 및 승인

이 섹션은 파이썬 배포판에 포함된 제삼자 소프트웨어에 대한 불완전하지만 늘어나고 있는 라이센스와 승인의 목록입니다.

C.3.1 메르센 트위스터

The `_random` C extension underlying the `random` module includes code based on a download from http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html. The following are the verbatim comments from the original code:

```
A C-program for MT19937, with initialization improved 2002/1/26.
Coded by Takuji Nishimura and Makoto Matsumoto.

Before using, initialize the state by using init_genrand(seed)
or init_by_array(init_key, key_length).

Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura,
All rights reserved.

Redistribution and use in source and binary forms, with or without
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
modification, are permitted provided that the following conditions
are met:

 1. Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

 2. Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

 3. The names of its contributors may not be used to endorse or promote
    products derived from this software without specific prior written
    permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


Any feedback is very welcome.
http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
email: m-mat @ math.sci.hiroshima-u.ac.jp (remove space)
```

C.3.2 소켓

The `socket` module uses the functions, `getaddrinfo()`, and `getnameinfo()`, which are coded in separate source files from the WIDE Project, https://www.wide.ad.jp/.

```
Copyright (C) 1995, 1996, 1997, and 1998 WIDE Project.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. Neither the name of the project nor the names of its contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE PROJECT AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE PROJECT OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

C.3.3 비동기 소켓 서비스

The `test.support.asynchat` and `test.support.asyncore` modules contain the following notice:

```
Copyright 1996 by Sam Rushing

                        All Rights Reserved

Permission to use, copy, modify, and distribute this software and
its documentation for any purpose and without fee is hereby
granted, provided that the above copyright notice appear in all
copies and that both that copyright notice and this permission
notice appear in supporting documentation, and that the name of Sam
Rushing not be used in advertising or publicity pertaining to
distribution of the software without specific, written prior
permission.

SAM RUSHING DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN
NO EVENT SHALL SAM RUSHING BE LIABLE FOR ANY SPECIAL, INDIRECT OR
CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS
OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```

C.3.4 쿠키 관리

`http.cookies` 모듈은 다음과 같은 주의 사항을 포함합니다:

```
Copyright 2000 by Timothy O'Malley <timo@alum.mit.edu>

               All Rights Reserved

Permission to use, copy, modify, and distribute this software
and its documentation for any purpose and without fee is hereby
granted, provided that the above copyright notice appear in all
copies and that both that copyright notice and this permission
notice appear in supporting documentation, and that the name of
Timothy O'Malley  not be used in advertising or publicity
pertaining to distribution of the software without specific, written
prior permission.

Timothy O'Malley DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS, IN NO EVENT SHALL Timothy O'Malley BE LIABLE FOR
ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

C.3.5 실행 추적

`trace` 모듈은 다음과 같은 주의 사항을 포함합니다:

```
portions copyright 2001, Autonomous Zones Industries, Inc., all rights...
err...  reserved and offered to the public under the terms of the
Python 2.2 license.
Author: Zooko O'Whielacronx
http://zooko.com/
mailto:zooko@zooko.com

Copyright 2000, Mojam Media, Inc., all rights reserved.
Author: Skip Montanaro

Copyright 1999, Bioreason, Inc., all rights reserved.
Author: Andrew Dalke

Copyright 1995-1997, Automatrix, Inc., all rights reserved.
Author: Skip Montanaro

Copyright 1991-1995, Stichting Mathematisch Centrum, all rights reserved.


Permission to use, copy, modify, and distribute this Python software and
its associated documentation for any purpose without fee is hereby
granted, provided that the above copyright notice appears in all copies,
and that both that copyright notice and this permission notice appear in
supporting documentation, and that the name of neither Automatrix,
Bioreason or Mojam Media be used in advertising or publicity pertaining to
distribution of the software without specific, written prior permission.
```

C.3.6 UUencode 및 UUdecode 함수

`uu` 모듈은 다음과 같은 주의 사항을 포함합니다:

```
Copyright 1994 by Lance Ellinghouse
Cathedral City, California Republic, United States of America.
                       All Rights Reserved
Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation, and that the name of Lance Ellinghouse
not be used in advertising or publicity pertaining to distribution
of the software without specific, written prior permission.
LANCE ELLINGHOUSE DISCLAIMS ALL WARRANTIES WITH REGARD TO
THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS, IN NO EVENT SHALL LANCE ELLINGHOUSE CENTRUM BE LIABLE
FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT
OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
Modified by Jack Jansen, CWI, July 1995:
- Use binascii module to do the actual line-by-line conversion
  between ascii and binary. This results in a 1000-fold speedup. The C
  version is still 5 times faster, though.
- Arguments more compliant with Python standard
```

C.3.7 XML 원격 프로시저 호출

xmlrpc.client 모듈은 다음과 같은 주의 사항을 포함합니다:

```
    The XML-RPC client interface is

Copyright (c) 1999-2002 by Secret Labs AB
Copyright (c) 1999-2002 by Fredrik Lundh

By obtaining, using, and/or copying this software and/or its
associated documentation, you agree that you have read, understood,
and will comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this software and
its associated documentation for any purpose and without fee is
hereby granted, provided that the above copyright notice appears in
all copies, and that both that copyright notice and this permission
notice appear in supporting documentation, and that the name of
Secret Labs AB or the author not be used in advertising or publicity
pertaining to distribution of the software without specific, written
prior permission.

SECRET LABS AB AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD
TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANT-
ABILITY AND FITNESS.  IN NO EVENT SHALL SECRET LABS AB OR THE AUTHOR
BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
OF THIS SOFTWARE.
```

C.3.8 test_epoll

The test.test_epoll module contains the following notice:

```
Copyright (c) 2001-2006 Twisted Matrix Laboratories.

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

C.3.9 Select kqueue

`select` 모듈은 kqueue 인터페이스에 대해 다음과 같은 주의 사항을 포함합니다:

```
Copyright (c) 2000 Doug White, 2006 James Knight, 2007 Christian Heimes
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

C.3.10 SipHash24

파일 `Python/pyhash.c` 에는 Dan Bernstein의 SipHash24 알고리즘의 Marek Majkowski의 구현이 포함되어 있습니다. 여기에는 다음과 같은 내용이 포함되어 있습니다:

```
<MIT License>
Copyright (c) 2013  Marek Majkowski <marek@popcount.org>

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.
</MIT License>

Original location:
   https://github.com/majek/csiphash/
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
Solution inspired by code from:
   Samuel Neves (supercop/crypto_auth/siphash24/little)
   djb (supercop/crypto_auth/siphash24/little2)
   Jean-Philippe Aumasson (https://131002.net/siphash/siphash24.c)
```

C.3.11 strtod 와 dtoa

The file `Python/dtoa.c`, which supplies C functions dtoa and strtod for conversion of C doubles to and from strings, is derived from the file of the same name by David M. Gay, currently available from https://web.archive.org/web/20220517033456/http://www.netlib.org/fp/dtoa.c. The original file, as retrieved on March 16, 2009, contains the following copyright and licensing notice:

```
/****************************************************************
 *
 * The author of this software is David M. Gay.
 *
 * Copyright (c) 1991, 2000, 2001 by Lucent Technologies.
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose without fee is hereby granted, provided that this entire notice
 * is included in all copies of any software which is or includes a copy
 * or modification of this software and in all copies of the supporting
 * documentation for such software.
 *
 * THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED
 * WARRANTY.  IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY
 * REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY
 * OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.
 *
 ***************************************************************/
```

C.3.12 OpenSSL

The modules `hashlib`, `posix`, `ssl`, `crypt` use the OpenSSL library for added performance if made available by the operating system. Additionally, the Windows and macOS installers for Python may include a copy of the OpenSSL libraries, so we include a copy of the OpenSSL license here. For the OpenSSL 3.0 release, and later releases derived from that, the Apache License v2 applies:

```
                                 Apache License
                           Version 2.0, January 2004
                        https://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
```

C.3.13 expat

The `pyexpat` extension is built using an included copy of the expat sources unless the build is configured `--with-system-expat`:

```
Copyright (c) 1998, 1999, 2000 Thai Open Source Software Center Ltd
                               and Clark Cooper

Permission is hereby granted, free of charge, to any person obtaining
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

C.3.14 libffi

The `_ctypes` C extension underlying the `ctypes` module is built using an included copy of the libffi sources unless the build is configured `--with-system-libffi`:

```
Copyright (c) 1996-2008  Red Hat, Inc and others.

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT.  IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
```

C.3.15 zlib

`zlib` 확장은 시스템에서 발견된 zlib 버전이 너무 오래되어서 빌드에 사용될 수 없으면, 포함된 zlib 소스 사본을 사용하여 빌드됩니다:

```
Copyright (C) 1995-2011 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied
warranty.  In no event will the authors be held liable for any damages
arising from the use of this software.
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not
   claim that you wrote the original software. If you use this software
   in a product, an acknowledgment in the product documentation would be
   appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be
   misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly        Mark Adler
jloup@gzip.org          madler@alumni.caltech.edu
```

C.3.16 cfuhash

`tracemalloc` 에 의해 사용되는 해시 테이블의 구현은 cfuhash 프로젝트를 기반으로 합니다:

```
Copyright (c) 2005 Don Owens
All rights reserved.

This code is released under the BSD license:

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

  * Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

  * Redistributions in binary form must reproduce the above
    copyright notice, this list of conditions and the following
    disclaimer in the documentation and/or other materials provided
    with the distribution.

  * Neither the name of the author nor the names of its
    contributors may be used to endorse or promote products derived
    from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
```

C.3.17 libmpdec

The `_decimal` C extension underlying the `decimal` module is built using an included copy of the libmpdec library unless the build is configured `--with-system-libmpdec`:

```
Copyright (c) 2008-2020 Stefan Krah. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

C.3.18 W3C C14N 테스트 스위트

The C14N 2.0 test suite in the `test` package (`Lib/test/xmltestdata/c14n-20/`) was retrieved from the W3C website at https://www.w3.org/TR/xml-c14n2-testcases/ and is distributed under the 3-clause BSD license:

```
Copyright (c) 2013 W3C(R) (MIT, ERCIM, Keio, Beihang),
All Rights Reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

* Redistributions of works must retain the original copyright notice,
  this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the original copyright
  notice, this list of conditions and the following disclaimer in the
  documentation and/or other materials provided with the distribution.
* Neither the name of the W3C nor the names of its contributors may be
  used to endorse or promote products derived from this work without
  specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

C.3.19 Audioop

The audioop module uses the code base in g771.c file of the SoX project. https://sourceforge.net/projects/sox/files/sox/12.17.7/sox-12.17.7.tar.gz

> This source code is a product of Sun Microsystems, Inc. and is provided for unrestricted use. Users may copy or modify this source code without charge.
>
> SUN SOURCE CODE IS PROVIDED AS IS WITH NO WARRANTIES OF ANY KIND INCLUDING THE WARRANTIES OF DESIGN, MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE, OR ARISING FROM A COURSE OF DEALING, USAGE OR TRADE PRACTICE.
>
> Sun source code is provided with no support and without any obligation on the part of Sun Microsystems, Inc. to assist in its use, correction, modification or enhancement.
>
> SUN MICROSYSTEMS, INC. SHALL HAVE NO LIABILITY WITH RESPECT TO THE INFRINGEMENT OF COPYRIGHTS, TRADE SECRETS OR ANY PATENTS BY THIS SOFTWARE OR ANY PART THEREOF.
>
> In no event will Sun Microsystems, Inc. be liable for any lost revenue or profits or other special, indirect and consequential damages, even if Sun has been advised of the possibility of such damages.
>
> Sun Microsystems, Inc. 2550 Garcia Avenue Mountain View, California 94043

C.3.20 asyncio

Parts of the `asyncio` module are incorporated from uvloop 0.16, which is distributed under the MIT license:

```
Copyright (c) 2015-2021 MagicStack Inc.  http://magic.io

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

APPENDIX D

저작권

파이썬과 이 설명서는:

Copyright © 2001-2023 Python Software Foundation. All rights reserved.

Copyright © 2000 BeOpen.com. All rights reserved.

Copyright © 1995-2000 Corporation for National Research Initiatives. All rights reserved.

Copyright © 1991-1995 Stichting Mathematisch Centrum. All rights reserved.

전체 라이센스 및 사용 권한 정보는 역사와 라이센스 에서 제공합니다.

색인

알파벳 이외

... , **301**
2to3, **301**
>>>, **301**
__all__ (*package variable*), 70
__dict__ (*module attribute*), 168
__doc__ (*module attribute*), 167
__file__ (*module attribute*), 167, 168
__future__, **306**
__import__
    built-in function, 71
__loader__ (*module attribute*), 167
__main__
    module, 12, 195, 208, 209
__name__ (*module attribute*), 167, 168
__package__ (*module attribute*), 167
__PYVENV_LAUNCHER__, 224, 230
__slots__, **314**
_frozen (*C struct*), 73
_inittab (*C struct*), 74
_inittab.initfunc (*C member*), 74
_inittab.name (*C member*), 74
_Py_c_diff (*C function*), 127
_Py_c_neg (*C function*), 128
_Py_c_pow (*C function*), 128
_Py_c_prod (*C function*), 128
_Py_c_quot (*C function*), 128
_Py_c_sum (*C function*), 127
_Py_InitializeMain (*C function*), 237
_Py_NoneStruct (*C var*), 250
_PyBytes_Resize (*C function*), 130
_PyCFunctionFast (*C type*), 252
_PyCFunctionFastWithKeywords (*C type*), 252
_PyCode_GetExtra (*C function*), 166
_PyCode_SetExtra (*C function*), 166
_PyEval_RequestCodeExtraIndex (*C function*), 165
_PyFrameEvalFunction (*C type*), 205
_PyInterpreterFrame (*C struct*), 182
_PyInterpreterState_GetEvalFrameFunc (*C function*), 206
_PyInterpreterState_SetEvalFrameFunc (*C function*), 206
_PyObject_GetDictPtr (*C function*), 91
_PyObject_New (*C function*), 249
_PyObject_NewVar (*C function*), 249
_PyTuple_Resize (*C function*), 150
_thread
    module, 202

A

abort (*C function*), 70
abs
    built-in function, 100
abstract base class (추상 베이스 클래스), **301**
allocfunc (*C type*), 290
annotation (어노테이션), **301**
argument (인자), **301**
argv (*in module sys*), 199
ascii
    built-in function, 92
asynchronous context manager (비동기 컨텍스트 관리자), **302**
asynchronous generator (비동기 제너레이터), **302**
asynchronous generator iterator (비동기 제너레이터 이터레이터), **302**
asynchronous iterable (비동기 이터러블), **302**
asynchronous iterator (비동기 이터레이터), **302**
attribute (어트리뷰트), **302**
awaitable (어웨이터블), **302**

B

BDFL, **303**
binary file (바이너리 파일), **303**
binaryfunc (*C type*), 291
borrowed reference, **303**
buffer interface
    (see buffer protocol), 106
buffer object
    (see buffer protocol), 106
buffer protocol, 106
built-in function
    __import__, 71
    abs, 100
    ascii, 92

bytes, 92
classmethod, 254
compile, 72
divmod, 99
float, 101
hash, 92, 269
int, 101
len, 93, 102, 104, 152, 155, 158
pow, 99, 101
repr, 91, 268
staticmethod, 254
tuple, 103, 153
type, 93
builtins
module, 12, 195, 208, 209
bytearray
object, 131
bytecode (바이트 코드), **303**
bytes
built-in function, 92
object, 129
bytes-like object (바이트열류 객체), **303**

C

callable, **303**
callback (콜백), **303**
calloc (*C function*), 239
Capsule
object, 179
C-contiguous, 109, 304
class (클래스), **303**
class variable (클래스 변수), **303**
classmethod
built-in function, 254
cleanup functions, 70
close (*in module os*), 209
CO_FUTURE_DIVISION (*C var*), 45
code object, 162
Common Vulnerabilities and Exposures
CVE 2008-5983, 199
compile
built-in function, 72
complex number
object, 127
complex number (복소수), **303**
context manager (컨텍스트 관리자), **304**
context variable (컨텍스트 변수), **304**
contiguous, 109
contiguous (연속), **304**
copyright (*in module sys*), 198
coroutine (코루틴), **304**
coroutine function (코루틴 함수), **304**
CPython, **304**

D

decorator (데코레이터), **304**
descrgetfunc (*C type*), 290
descriptor (디스크립터), **304**
descrsetfunc (*C type*), 290
destructor (*C type*), 290
dictionary
object, 153
dictionary (딕셔너리), **304**
dictionary comprehension (딕셔너리 컴프리헨션), **305**
dictionary view (딕셔너리 뷰), **305**
divmod
built-in function, 99
docstring (독스트링), **305**
duck-typing (덕 타이핑), **305**

E

EAFP, **305**
EOFError (*built-in exception*), 166
exc_info (*in module sys*), 11
executable (*in module sys*), 197
exit (*C function*), 70
expression (표현식), **305**
extension module (확장 모듈), **305**

F

f-string (*f*-문자열), **305**
file
object, 166
file object (파일 객체), **305**
file-like object (파일류 객체), **305**
filesystem encoding and error handler, **305**
finder (파인더), **306**
float
built-in function, 101
floating-point
object, 125
floor division (정수 나눗셈), **306**
Fortran contiguous, 109, 304
free (*C function*), 239
freefunc (*C type*), 290
freeze utility, 73
frozenset
object, 157
function
object, 159
function (함수), **306**
function annotation (함수 어노테이션), **306**

G

garbage collection (가비지 수거), **306**
gcvisitobjects_t (*C type*), 297
generator (제너레이터), **306**
generator expression (제너레이터 표현식), **307**
generator iterator (제너레이터 이터레이터), **306**
generic function (제네릭 함수), **307**
generic type (제네릭 형), **307**
getattrfunc (*C type*), 290
getattrofunc (*C type*), 290
getbufferproc (*C type*), 291

getiterfunc (*C type*), 291
getter (*C type*), 258
GIL, **307**
global interpreter lock, 200
global interpreter lock (전역 인터프리터 록), **307**

H

hash
    built-in function, 92, 269
hash-based pyc (해시 기반 *pyc*), **307**
hashable (해시 가능), **307**
hashfunc (*C type*), 290

I

IDLE, **307**
immortal, **307**
immutable (불변), **308**
import path (임포트 경로), **308**
importer (임포터), **308**
importing (임포팅), **308**
incr_item(), 11, 12
initproc (*C type*), 290
inquiry (*C type*), 296
instancemethod
    object, 161
int
    built-in function, 101
integer
    object, 121
interactive (대화형), **308**
interpreted (인터프리티드), **308**
interpreter lock, 200
interpreter shutdown (인터프리터 종료), **308**
iterable (이터러블), **308**
iterator (이터레이터), **308**
iternextfunc (*C type*), 291

K

key function (키 함수), **309**
KeyboardInterrupt (*built-in exception*), 59
keyword argument (키워드 인자), **309**

L

lambda (람다), **309**
LBYL, **309**
len
    built-in function, 93, 102, 104, 152, 155, 158
lenfunc (*C type*), 291
lexical analyzer, **309**
list
    object, 152
list (리스트), **309**
list comprehension (리스트 컴프리헨션), **309**
loader (로더), **309**
locale encoding, **309**
lock, interpreter, 200
long integer
    object, 121
LONG_MAX (*C macro*), 122

M

magic
    method (메서드), 310
magic method (매직 메서드), **310**
main(), 196, 199
malloc (*C function*), 239
mapping
    object, 153
mapping (매핑), **310**
memoryview
    object, 177
meta path finder (메타 경로 파인더), **310**
metaclass (메타 클래스), **310**
METH_CLASS (*C macro*), 254
METH_COEXIST (*C macro*), 254
METH_FASTCALL (*C macro*), 253
METH_KEYWORDS (*C macro*), 253
METH_METHOD (*C macro*), 253
METH_NOARGS (*C macro*), 254
METH_O (*C macro*), 254
METH_STATIC (*C macro*), 254
METH_VARARGS (*C macro*), 253
method
    object, 161
method (메서드), **310**
    magic, 310
    special, 314
method resolution order (메서드 결정 순서), **310**
MethodType (*in module types*), 159, 161
module
    __main__, 12, 195, 208, 209
    _thread, 202
    builtins, 12, 195, 208, 209
    object, 167
    search path, 12, 195, 197, 198
    signal, 59
    sys, 12, 195, 208, 209
module (모듈), **310**
module spec (모듈 스펙), **310**
modules (*in module sys*), 70, 195
ModuleType (*in module types*), 167
MRO, **310**
mutable (가변), **310**

N

named tuple (네임드 튜플), **310**
namespace (이름 공간), **311**
namespace package (이름 공간 패키지), **311**
nested scope (중첩된 스코프), **311**
new-style class (뉴스타일 클래스), **311**
newfunc (*C type*), 290
None
    object, 121

numeric
    object, 121

O

object
    bytearray, 131
    bytes, 129
    Capsule, 179
    code, 162
    complex number, 127
    dictionary, 153
    file, 166
    floating-point, 125
    frozenset, 157
    function, 159
    instancemethod, 161
    integer, 121
    list, 152
    long integer, 121
    mapping, 153
    memoryview, 177
    method, 161
    module, 167
    None, 121
    numeric, 121
    sequence, 129
    set, 157
    tuple, 149
    type, 7, 115
object (객체), **311**
objobjargproc (*C type*), 291
objobjproc (*C type*), 291
OverflowError (*built-in exception*), 122, 123

P

package (패키지), **311**
package variable
    __all__, 70
parameter (매개변수), **311**
PATH, 12
path
    module search, 12, 195, 197, 198
path (*in module sys*), 12, 195, 197, 198
path based finder (경로 기반 파인더), **312**
path entry (경로 엔트리), **312**
path entry finder (경로 엔트리 파인더), **312**
path entry hook (경로 엔트리 훅), **312**
path-like object (경로류 객체), **312**
PEP, **312**
platform (*in module sys*), 198
portion (포션), **312**
positional argument (위치 인자), **312**
pow
    built-in function, 99, 101
provisional API (잠정 *API*), **313**
provisional package (잠정 패키지), **313**
Py_ABS (*C macro*), 4
Py_AddPendingCall (*C function*), 210
Py_ALWAYS_INLINE (*C macro*), 5
Py_AtExit (*C function*), 70
Py_AUDIT_READ (*C macro*), 256
Py_AuditHookFunction (*C type*), 70
Py_BEGIN_ALLOW_THREADS (*C macro*), 200, 204
Py_BLOCK_THREADS (*C macro*), 204
Py_buffer (*C type*), 107
Py_buffer.buf (*C member*), 107
Py_buffer.format (*C member*), 107
Py_buffer.internal (*C member*), 108
Py_buffer.itemsize (*C member*), 107
Py_buffer.len (*C member*), 107
Py_buffer.ndim (*C member*), 107
Py_buffer.obj (*C member*), 107
Py_buffer.readonly (*C member*), 107
Py_buffer.shape (*C member*), 107
Py_buffer.strides (*C member*), 108
Py_buffer.suboffsets (*C member*), 108
Py_BuildValue (*C function*), 81
Py_BytesMain (*C function*), 41
Py_BytesWarningFlag (*C var*), 192
Py_CHARMASK (*C macro*), 5
Py_CLEANUP_SUPPORTED (*C macro*), 79
Py_CLEAR (*C function*), 48
Py_CompileString (*C function*), 4345
Py_CompileStringExFlags (*C function*), 44
Py_CompileStringFlags (*C function*), 43
Py_CompileStringObject (*C function*), 44
Py_complex (*C type*), 127
Py_complex.imag (*C member*), 127
Py_complex.real (*C member*), 127
Py_DEBUG (*C macro*), 13
Py_DebugFlag (*C var*), 192
Py_DecodeLocale (*C function*), 66
Py_DECREF (*C function*), 7, 48
Py_DecRef (*C function*), 49
Py_DEPRECATED (*C macro*), 5
Py_DontWriteBytecodeFlag (*C var*), 192
Py_Ellipsis (*C var*), 177
Py_EncodeLocale (*C function*), 67
Py_END_ALLOW_THREADS (*C macro*), 200, 204
Py_EndInterpreter (*C function*), 209
Py_EnterRecursiveCall (*C function*), 62
Py_EQ (*C macro*), 277
Py_eval_input (*C var*), 44
Py_Exit (*C function*), 70
Py_ExitStatusException (*C function*), 219
Py_False (*C var*), 125
Py_FatalError (*C function*), 70
Py_FatalError(), 199
Py_FdIsInteractive (*C function*), 65
Py_file_input (*C var*), 44
Py_Finalize (*C function*), 196
Py_FinalizeEx (*C function*), 70, 195, 209
Py_FrozenFlag (*C var*), 193
Py_GE (*C macro*), 277
Py_GenericAlias (*C function*), 189
Py_GenericAliasType (*C var*), 189

Py_GetArgcArgv (*C function*), 236
Py_GetBuildInfo (*C function*), 198
Py_GetCompiler (*C function*), 198
Py_GetCopyright (*C function*), 198
Py_GETENV (*C macro*), 5
Py_GetExecPrefix (*C function*), 12, 197
Py_GetPath (*C function*), 12, 197
Py_GetPath(), 196, 198
Py_GetPlatform (*C function*), 198
Py_GetPrefix (*C function*), 12, 197
Py_GetProgramFullPath (*C function*), 12, 197
Py_GetProgramName (*C function*), 196
Py_GetPythonHome (*C function*), 200
Py_GetVersion (*C function*), 198
Py_GT (*C macro*), 277
Py_hash_t (*C type*), 85
Py_HashRandomizationFlag (*C var*), 193
Py_IgnoreEnvironmentFlag (*C var*), 193
Py_INCREF (*C function*), 7, 47
Py_IncRef (*C function*), 49
Py_Initialize (*C function*), 12, 195, 209
Py_Initialize(), 196
Py_InitializeEx (*C function*), 195
Py_InitializeFromConfig (*C function*), 233
Py_InspectFlag (*C var*), 193
Py_InteractiveFlag (*C var*), 193
Py_Is (*C function*), 250
Py_IS_TYPE (*C function*), 251
Py_IsFalse (*C function*), 251
Py_IsInitialized (*C function*), 12, 195
Py_IsNone (*C function*), 251
Py_IsolatedFlag (*C var*), 193
Py_IsTrue (*C function*), 251
Py_LE (*C macro*), 277
Py_LeaveRecursiveCall (*C function*), 62
Py_LegacyWindowsFSEncodingFlag (*C var*), 193
Py_LegacyWindowsStdioFlag (*C var*), 194
Py_LIMITED_API (*C macro*), 16
Py_LT (*C macro*), 277
Py_Main (*C function*), 41
PY_MAJOR_VERSION (*C macro*), 299
Py_MAX (*C macro*), 5
Py_MEMBER_SIZE (*C macro*), 5
PY_MICRO_VERSION (*C macro*), 299
Py_MIN (*C macro*), 5
PY_MINOR_VERSION (*C macro*), 299
Py_mod_create (*C macro*), 171
Py_mod_exec (*C macro*), 171
Py_mod_multiple_interpreters (*C macro*), 171
Py_MOD_MULTIPLE_INTERPRETERS_NOT_SUPPORTED (*C macro*), 171
Py_MOD_MULTIPLE_INTERPRETERS_SUPPORTED (*C macro*), 171
Py_MOD_PER_INTERPRETER_GIL_SUPPORTED (*C macro*), 171
Py_NE (*C macro*), 277
Py_NewInterpreter (*C function*), 209
Py_NewInterpreterFromConfig (*C function*), 208
Py_NewRef (*C function*), 47
Py_NO_INLINE (*C macro*), 5
Py_None (*C var*), 121
Py_NoSiteFlag (*C var*), 194
Py_NotImplemented (*C var*), 89
Py_NoUserSiteDirectory (*C var*), 194
Py_OpenCodeHookFunction (*C type*), 167
Py_OptimizeFlag (*C var*), 194
Py_PreInitialize (*C function*), 221
Py_PreInitializeFromArgs (*C function*), 222
Py_PreInitializeFromBytesArgs (*C function*), 222
Py_PRINT_RAW (*C macro*), 89, 167
Py_QuietFlag (*C var*), 194
Py_READONLY (*C macro*), 256
Py_REFCNT (*C function*), 47
Py_RELATIVE_OFFSET (*C macro*), 256
PY_RELEASE_LEVEL (*C macro*), 299
PY_RELEASE_SERIAL (*C macro*), 299
Py_ReprEnter (*C function*), 62
Py_ReprLeave (*C function*), 62
Py_RETURN_FALSE (*C macro*), 125
Py_RETURN_NONE (*C macro*), 121
Py_RETURN_NOTIMPLEMENTED (*C macro*), 89
Py_RETURN_RICHCOMPARE (*C macro*), 277
Py_RETURN_TRUE (*C macro*), 125
Py_RunMain (*C function*), 236
Py_SET_REFCNT (*C function*), 47
Py_SET_SIZE (*C function*), 251
Py_SET_TYPE (*C function*), 251
Py_SetPath (*C function*), 198
Py_SetPath(), 197
Py_SetProgramName (*C function*), 12, 196
Py_SetProgramName(), 195, 197
Py_SetPythonHome (*C function*), 199
Py_SETREF (*C macro*), 49
Py_SetStandardStreamEncoding (*C function*), 196
Py_single_input (*C var*), 44
Py_SIZE (*C function*), 251
Py_ssize_t (*C type*), 10
PY_SSIZE_T_MAX (*C macro*), 123
Py_STRINGIFY (*C macro*), 5
Py_T_BOOL (*C macro*), 257
Py_T_BYTE (*C macro*), 257
Py_T_CHAR (*C macro*), 257
Py_T_DOUBLE (*C macro*), 257
Py_T_FLOAT (*C macro*), 257
Py_T_INT (*C macro*), 257
Py_T_LONG (*C macro*), 257
Py_T_LONGLONG (*C macro*), 257
Py_T_OBJECT_EX (*C macro*), 257
Py_T_PYSSIZET (*C macro*), 257
Py_T_SHORT (*C macro*), 257
Py_T_STRING (*C macro*), 257
Py_T_STRING_INPLACE (*C macro*), 257
Py_T_UBYTE (*C macro*), 257
Py_T_UINT (*C macro*), 257
Py_T_ULONG (*C macro*), 257

Py_T_ULONGLONG (*C macro*), 257
Py_T_USHORT (*C macro*), 257
Py_TPFLAGS_BASE_EXC_SUBCLASS (*C macro*), 273
Py_TPFLAGS_BASETYPE (*C macro*), 271
Py_TPFLAGS_BYTES_SUBCLASS (*C macro*), 273
Py_TPFLAGS_DEFAULT (*C macro*), 272
Py_TPFLAGS_DICT_SUBCLASS (*C macro*), 273
Py_TPFLAGS_DISALLOW_INSTANTIATION (*C macro*), 273
Py_TPFLAGS_HAVE_FINALIZE (*C macro*), 273
Py_TPFLAGS_HAVE_GC (*C macro*), 271
Py_TPFLAGS_HAVE_VECTORCALL (*C macro*), 273
Py_TPFLAGS_HEAPTYPE (*C macro*), 271
Py_TPFLAGS_IMMUTABLETYPE (*C macro*), 273
Py_TPFLAGS_ITEMS_AT_END (*C macro*), 272
Py_TPFLAGS_LIST_SUBCLASS (*C macro*), 273
Py_TPFLAGS_LONG_SUBCLASS (*C macro*), 273
Py_TPFLAGS_MANAGED_DICT (*C macro*), 272
Py_TPFLAGS_MANAGED_WEAKREF (*C macro*), 272
Py_TPFLAGS_MAPPING (*C macro*), 274
Py_TPFLAGS_METHOD_DESCRIPTOR (*C macro*), 272
Py_TPFLAGS_READY (*C macro*), 271
Py_TPFLAGS_READYING (*C macro*), 271
Py_TPFLAGS_SEQUENCE (*C macro*), 274
Py_TPFLAGS_TUPLE_SUBCLASS (*C macro*), 273
Py_TPFLAGS_TYPE_SUBCLASS (*C macro*), 273
Py_TPFLAGS_UNICODE_SUBCLASS (*C macro*), 273
Py_TPFLAGS_VALID_VERSION_TAG (*C macro*), 274
Py_tracefunc (*C type*), 211
Py_True (*C var*), 125
Py_tss_NEEDS_INIT (*C macro*), 213
Py_tss_t (*C type*), 213
Py_TYPE (*C function*), 251
Py_UCS1 (*C type*), 132
Py_UCS2 (*C type*), 132
Py_UCS4 (*C type*), 132
Py_uhash_t (*C type*), 85
Py_UNBLOCK_THREADS (*C macro*), 204
Py_UnbufferedStdioFlag (*C var*), 194
Py_UNICODE (*C type*), 132
Py_UNICODE_IS_HIGH_SURROGATE (*C function*), 135
Py_UNICODE_IS_LOW_SURROGATE (*C function*), 135
Py_UNICODE_IS_SURROGATE (*C function*), 135
Py_UNICODE_ISALNUM (*C function*), 134
Py_UNICODE_ISALPHA (*C function*), 134
Py_UNICODE_ISDECIMAL (*C function*), 134
Py_UNICODE_ISDIGIT (*C function*), 134
Py_UNICODE_ISLINEBREAK (*C function*), 134
Py_UNICODE_ISLOWER (*C function*), 134
Py_UNICODE_ISNUMERIC (*C function*), 134
Py_UNICODE_ISPRINTABLE (*C function*), 134
Py_UNICODE_ISSPACE (*C function*), 134
Py_UNICODE_ISTITLE (*C function*), 134
Py_UNICODE_ISUPPER (*C function*), 134
Py_UNICODE_JOIN_SURROGATES (*C function*), 135
Py_UNICODE_TODECIMAL (*C function*), 134
Py_UNICODE_TODIGIT (*C function*), 135
Py_UNICODE_TOLOWER (*C function*), 134
Py_UNICODE_TONUMERIC (*C function*), 135
Py_UNICODE_TOTITLE (*C function*), 134
Py_UNICODE_TOUPPER (*C function*), 134
Py_UNREACHABLE (*C macro*), 6
Py_UNUSED (*C macro*), 6
Py_VaBuildValue (*C function*), 83
PY_VECTORCALL_ARGUMENTS_OFFSET (*C macro*), 95
Py_VerboseFlag (*C var*), 195
Py_Version (*C var*), 299
PY_VERSION_HEX (*C macro*), 299
Py_VISIT (*C function*), 295
Py_XDECREF (*C function*), 12, 48
Py_XINCREF (*C function*), 47
Py_XNewRef (*C function*), 48
Py_XSETREF (*C macro*), 49
PyAIter_Check (*C function*), 105
PyAnySet_Check (*C function*), 158
PyAnySet_CheckExact (*C function*), 158
PyArg_Parse (*C function*), 80
PyArg_ParseTuple (*C function*), 80
PyArg_ParseTupleAndKeywords (*C function*), 80
PyArg_UnpackTuple (*C function*), 80
PyArg_ValidateKeywordArguments (*C function*), 80
PyArg_VaParse (*C function*), 80
PyArg_VaParseTupleAndKeywords (*C function*), 80
PyASCIIObject (*C type*), 132
PyAsyncMethods (*C type*), 289
PyAsyncMethods.am_aiter (*C member*), 289
PyAsyncMethods.am_anext (*C member*), 289
PyAsyncMethods.am_await (*C member*), 289
PyAsyncMethods.am_send (*C member*), 289
PyBaseObject_Type (*C var*), 250
PyBool_Check (*C function*), 125
PyBool_FromLong (*C function*), 125
PyBool_Type (*C var*), 125
PyBUF_ANY_CONTIGUOUS (*C macro*), 109
PyBUF_C_CONTIGUOUS (*C macro*), 109
PyBUF_CONTIG (*C macro*), 110
PyBUF_CONTIG_RO (*C macro*), 110
PyBUF_F_CONTIGUOUS (*C macro*), 109
PyBUF_FORMAT (*C macro*), 109
PyBUF_FULL (*C macro*), 110
PyBUF_FULL_RO (*C macro*), 110
PyBUF_INDIRECT (*C macro*), 109
PyBUF_MAX_NDIM (*C macro*), 108
PyBUF_ND (*C macro*), 109
PyBUF_READ (*C macro*), 178
PyBUF_RECORDS (*C macro*), 110
PyBUF_RECORDS_RO (*C macro*), 110
PyBUF_SIMPLE (*C macro*), 109
PyBUF_STRIDED (*C macro*), 110
PyBUF_STRIDED_RO (*C macro*), 110
PyBUF_STRIDES (*C macro*), 109
PyBUF_WRITABLE (*C macro*), 109
PyBUF_WRITE (*C macro*), 178
PyBuffer_FillContiguousStrides (*C function*), 112

PyBuffer_FillInfo (*C function*), 112
PyBuffer_FromContiguous (*C function*), 112
PyBuffer_GetPointer (*C function*), 112
PyBuffer_IsContiguous (*C function*), 112
PyBuffer_Release (*C function*), 112
PyBuffer_SizeFromFormat (*C function*), 112
PyBuffer_ToContiguous (*C function*), 112
PyBufferProcs (*C type*), 106, 288
PyBufferProcs.bf_getbuffer (*C member*), 288
PyBufferProcs.bf_releasebuffer (*C member*), 288
PyByteArray_AS_STRING (*C function*), 131
PyByteArray_AsString (*C function*), 131
PyByteArray_Check (*C function*), 131
PyByteArray_CheckExact (*C function*), 131
PyByteArray_Concat (*C function*), 131
PyByteArray_FromObject (*C function*), 131
PyByteArray_FromStringAndSize (*C function*), 131
PyByteArray_GET_SIZE (*C function*), 131
PyByteArray_Resize (*C function*), 131
PyByteArray_Size (*C function*), 131
PyByteArray_Type (*C var*), 131
PyByteArrayObject (*C type*), 131
PyBytes_AS_STRING (*C function*), 130
PyBytes_AsString (*C function*), 130
PyBytes_AsStringAndSize (*C function*), 130
PyBytes_Check (*C function*), 129
PyBytes_CheckExact (*C function*), 129
PyBytes_Concat (*C function*), 130
PyBytes_ConcatAndDel (*C function*), 130
PyBytes_FromFormat (*C function*), 129
PyBytes_FromFormatV (*C function*), 130
PyBytes_FromObject (*C function*), 130
PyBytes_FromString (*C function*), 129
PyBytes_FromStringAndSize (*C function*), 129
PyBytes_GET_SIZE (*C function*), 130
PyBytes_Size (*C function*), 130
PyBytes_Type (*C var*), 129
PyBytesObject (*C type*), 129
PyCallable_Check (*C function*), 99
PyCallIter_Check (*C function*), 175
PyCallIter_New (*C function*), 175
PyCallIter_Type (*C var*), 175
PyCapsule (*C type*), 179
PyCapsule_CheckExact (*C function*), 179
PyCapsule_Destructor (*C type*), 179
PyCapsule_GetContext (*C function*), 180
PyCapsule_GetDestructor (*C function*), 180
PyCapsule_GetName (*C function*), 180
PyCapsule_GetPointer (*C function*), 180
PyCapsule_Import (*C function*), 180
PyCapsule_IsValid (*C function*), 180
PyCapsule_New (*C function*), 179
PyCapsule_SetContext (*C function*), 180
PyCapsule_SetDestructor (*C function*), 180
PyCapsule_SetName (*C function*), 181
PyCapsule_SetPointer (*C function*), 181
PyCell_Check (*C function*), 162
PyCell_GET (*C function*), 162
PyCell_Get (*C function*), 162
PyCell_New (*C function*), 162
PyCell_SET (*C function*), 162
PyCell_Set (*C function*), 162
PyCell_Type (*C var*), 162
PyCellObject (*C type*), 162
PyCFunction (*C type*), 252
PyCFunction_New (*C function*), 255
PyCFunction_NewEx (*C function*), 254
PyCFunctionWithKeywords (*C type*), 252
PyCMethod (*C type*), 252
PyCMethod_New (*C function*), 254
PyCode_Addr2Line (*C function*), 164
PyCode_Addr2Location (*C function*), 164
PyCode_AddWatcher (*C function*), 164
PyCode_Check (*C function*), 163
PyCode_ClearWatcher (*C function*), 164
PyCode_GetCellvars (*C function*), 164
PyCode_GetCode (*C function*), 164
PyCode_GetFirstFree (*C function*), 163
PyCode_GetFreevars (*C function*), 164
PyCode_GetNumFree (*C function*), 163
PyCode_GetVarnames (*C function*), 164
PyCode_New (*C function*), 163
PyCode_NewEmpty (*C function*), 163
PyCode_NewWithPosOnlyArgs (*C function*), 163
PyCode_Type (*C var*), 163
PyCode_WatchCallback (*C type*), 165
PyCodec_BackslashReplaceErrors (*C function*), 87
PyCodec_Decode (*C function*), 86
PyCodec_Decoder (*C function*), 86
PyCodec_Encode (*C function*), 86
PyCodec_Encoder (*C function*), 86
PyCodec_IgnoreErrors (*C function*), 87
PyCodec_IncrementalDecoder (*C function*), 86
PyCodec_IncrementalEncoder (*C function*), 86
PyCodec_KnownEncoding (*C function*), 86
PyCodec_LookupError (*C function*), 87
PyCodec_NameReplaceErrors (*C function*), 87
PyCodec_Register (*C function*), 86
PyCodec_RegisterError (*C function*), 87
PyCodec_ReplaceErrors (*C function*), 87
PyCodec_StreamReader (*C function*), 86
PyCodec_StreamWriter (*C function*), 87
PyCodec_StrictErrors (*C function*), 87
PyCodec_Unregister (*C function*), 86
PyCodec_XMLCharRefReplaceErrors (*C function*), 87
PyCodeEvent (*C type*), 164
PyCodeObject (*C type*), 162
PyCompactUnicodeObject (*C type*), 132
PyCompilerFlags (*C struct*), 45
PyCompilerFlags.cf_feature_version (*C member*), 45
PyCompilerFlags.cf_flags (*C member*), 45

PyComplex_AsCComplex (*C function*), 128
PyComplex_Check (*C function*), 128
PyComplex_CheckExact (*C function*), 128
PyComplex_FromCComplex (*C function*), 128
PyComplex_FromDoubles (*C function*), 128
PyComplex_ImagAsDouble (*C function*), 128
PyComplex_RealAsDouble (*C function*), 128
PyComplex_Type (*C var*), 128
PyComplexObject (*C type*), 128
PyConfig (*C type*), 222
PyConfig_Clear (*C function*), 223
PyConfig_InitIsolatedConfig (*C function*), 223
PyConfig_InitPythonConfig (*C function*), 222
PyConfig_Read (*C function*), 223
PyConfig_SetArgv (*C function*), 223
PyConfig_SetBytesArgv (*C function*), 223
PyConfig_SetBytesString (*C function*), 223
PyConfig_SetString (*C function*), 223
PyConfig_SetWideStringList (*C function*), 223
PyConfig.argv (*C member*), 224
PyConfig.base_exec_prefix (*C member*), 224
PyConfig.base_executable (*C member*), 224
PyConfig.base_prefix (*C member*), 224
PyConfig.buffered_stdio (*C member*), 224
PyConfig.bytes_warning (*C member*), 225
PyConfig.check_hash_pycs_mode (*C member*), 225
PyConfig.code_debug_ranges (*C member*), 225
PyConfig.configure_c_stdio (*C member*), 225
PyConfig.dev_mode (*C member*), 225
PyConfig.dump_refs (*C member*), 225
PyConfig.exec_prefix (*C member*), 226
PyConfig.executable (*C member*), 226
PyConfig.faulthandler (*C member*), 226
PyConfig.filesystem_encoding (*C member*), 226
PyConfig.filesystem_errors (*C member*), 226
PyConfig.hash_seed (*C member*), 226
PyConfig.home (*C member*), 227
PyConfig.import_time (*C member*), 227
PyConfig.inspect (*C member*), 227
PyConfig.install_signal_handlers (*C member*), 227
PyConfig.int_max_str_digits (*C member*), 227
PyConfig.interactive (*C member*), 227
PyConfig.isolated (*C member*), 227
PyConfig.legacy_windows_stdio (*C member*), 228
PyConfig.malloc_stats (*C member*), 228
PyConfig.module_search_paths (*C member*), 228
PyConfig.module_search_paths_set (*C member*), 228
PyConfig.optimization_level (*C member*), 229
PyConfig.orig_argv (*C member*), 229
PyConfig.parse_argv (*C member*), 229
PyConfig.parser_debug (*C member*), 229
PyConfig.pathconfig_warnings (*C member*), 229
PyConfig.perf_profiling (*C member*), 231
PyConfig.platlibdir (*C member*), 228
PyConfig.prefix (*C member*), 229
PyConfig.program_name (*C member*), 230
PyConfig.pycache_prefix (*C member*), 230
PyConfig.pythonpath_env (*C member*), 228
PyConfig.quiet (*C member*), 230
PyConfig.run_command (*C member*), 230
PyConfig.run_filename (*C member*), 230
PyConfig.run_module (*C member*), 230
PyConfig.safe_path (*C member*), 224
PyConfig.show_ref_count (*C member*), 230
PyConfig.site_import (*C member*), 231
PyConfig.skip_source_first_line (*C member*), 231
PyConfig.stdio_encoding (*C member*), 231
PyConfig.stdio_errors (*C member*), 231
PyConfig.tracemalloc (*C member*), 231
PyConfig.use_environment (*C member*), 232
PyConfig.use_hash_seed (*C member*), 227
PyConfig.user_site_directory (*C member*), 232
PyConfig.verbose (*C member*), 232
PyConfig.warn_default_encoding (*C member*), 225
PyConfig.warnoptions (*C member*), 232
PyConfig.write_bytecode (*C member*), 232
PyConfig.xoptions (*C member*), 232
PyContext (*C type*), 184
PyContext_CheckExact (*C function*), 184
PyContext_Copy (*C function*), 185
PyContext_CopyCurrent (*C function*), 185
PyContext_Enter (*C function*), 185
PyContext_Exit (*C function*), 185
PyContext_New (*C function*), 185
PyContext_Type (*C var*), 184
PyContextToken (*C type*), 184
PyContextToken_CheckExact (*C function*), 184
PyContextToken_Type (*C var*), 184
PyContextVar (*C type*), 184
PyContextVar_CheckExact (*C function*), 184
PyContextVar_Get (*C function*), 185
PyContextVar_New (*C function*), 185
PyContextVar_Reset (*C function*), 185
PyContextVar_Set (*C function*), 185
PyContextVar_Type (*C var*), 184
PyCoro_CheckExact (*C function*), 184
PyCoro_New (*C function*), 184
PyCoro_Type (*C var*), 183
PyCoroObject (*C type*), 183
PyDate_Check (*C function*), 186
PyDate_CheckExact (*C function*), 186
PyDate_FromDate (*C function*), 187
PyDate_FromTimestamp (*C function*), 189
PyDateTime_Check (*C function*), 186
PyDateTime_CheckExact (*C function*), 186
PyDateTime_Date (*C type*), 185
PyDateTime_DATE_GET_FOLD (*C function*), 188
PyDateTime_DATE_GET_HOUR (*C function*), 188
PyDateTime_DATE_GET_MICROSECOND (*C function*), 188

PyDateTime_DATE_GET_MINUTE (*C function*), 188
PyDateTime_DATE_GET_SECOND (*C function*), 188
PyDateTime_DATE_GET_TZINFO (*C function*), 188
PyDateTime_DateTime (*C type*), 185
PyDateTime_DateTimeType (*C var*), 186
PyDateTime_DateType (*C var*), 186
PyDateTime_Delta (*C type*), 186
PyDateTime_DELTA_GET_DAYS (*C function*), 188
PyDateTime_DELTA_GET_MICROSECONDS (*C function*), 189
PyDateTime_DELTA_GET_SECONDS (*C function*), 188
PyDateTime_DeltaType (*C var*), 186
PyDateTime_FromDateAndTime (*C function*), 187
PyDateTime_FromDateAndTimeAndFold (*C function*), 187
PyDateTime_FromTimestamp (*C function*), 189
PyDateTime_GET_DAY (*C function*), 188
PyDateTime_GET_MONTH (*C function*), 187
PyDateTime_GET_YEAR (*C function*), 187
PyDateTime_Time (*C type*), 186
PyDateTime_TIME_GET_FOLD (*C function*), 188
PyDateTime_TIME_GET_HOUR (*C function*), 188
PyDateTime_TIME_GET_MICROSECOND (*C function*), 188
PyDateTime_TIME_GET_MINUTE (*C function*), 188
PyDateTime_TIME_GET_SECOND (*C function*), 188
PyDateTime_TIME_GET_TZINFO (*C function*), 188
PyDateTime_TimeType (*C var*), 186
PyDateTime_TimeZone_UTC (*C var*), 186
PyDateTime_TZInfoType (*C var*), 186
PyDelta_Check (*C function*), 186
PyDelta_CheckExact (*C function*), 187
PyDelta_FromDSU (*C function*), 187
PyDescr_IsData (*C function*), 176
PyDescr_NewClassMethod (*C function*), 176
PyDescr_NewGetSet (*C function*), 176
PyDescr_NewMember (*C function*), 176
PyDescr_NewMethod (*C function*), 176
PyDescr_NewWrapper (*C function*), 176
PyDict_AddWatcher (*C function*), 156
PyDict_Check (*C function*), 153
PyDict_CheckExact (*C function*), 153
PyDict_Clear (*C function*), 154
PyDict_ClearWatcher (*C function*), 156
PyDict_Contains (*C function*), 154
PyDict_Copy (*C function*), 154
PyDict_DelItem (*C function*), 154
PyDict_DelItemString (*C function*), 154
PyDict_GetItem (*C function*), 154
PyDict_GetItemString (*C function*), 154
PyDict_GetItemWithError (*C function*), 154
PyDict_Items (*C function*), 155
PyDict_Keys (*C function*), 155
PyDict_Merge (*C function*), 156
PyDict_MergeFromSeq2 (*C function*), 156
PyDict_New (*C function*), 153
PyDict_Next (*C function*), 155
PyDict_SetDefault (*C function*), 155
PyDict_SetItem (*C function*), 154
PyDict_SetItemString (*C function*), 154
PyDict_Size (*C function*), 155
PyDict_Type (*C var*), 153
PyDict_Unwatch (*C function*), 156
PyDict_Update (*C function*), 156
PyDict_Values (*C function*), 155
PyDict_Watch (*C function*), 156
PyDict_WatchCallback (*C type*), 157
PyDict_WatchEvent (*C type*), 156
PyDictObject (*C type*), 153
PyDictProxy_New (*C function*), 153
PyDoc_STR (*C macro*), 6
PyDoc_STRVAR (*C macro*), 6
PyEllipsis_Type (*C var*), 177
PyErr_BadArgument (*C function*), 53
PyErr_BadInternalCall (*C function*), 54
PyErr_CheckSignals (*C function*), 59
PyErr_Clear (*C function*), 10, 12, 51
PyErr_DisplayException (*C function*), 52
PyErr_ExceptionMatches (*C function*), 12, 56
PyErr_Fetch (*C function*), 56
PyErr_Format (*C function*), 52
PyErr_FormatV (*C function*), 52
PyErr_GetExcInfo (*C function*), 58
PyErr_GetHandledException (*C function*), 57
PyErr_GetRaisedException (*C function*), 56
PyErr_GivenExceptionMatches (*C function*), 56
PyErr_NewException (*C function*), 60
PyErr_NewExceptionWithDoc (*C function*), 60
PyErr_NoMemory (*C function*), 53
PyErr_NormalizeException (*C function*), 57
PyErr_Occurred (*C function*), 10, 55
PyErr_Print (*C function*), 52
PyErr_PrintEx (*C function*), 51
PyErr_ResourceWarning (*C function*), 55
PyErr_Restore (*C function*), 57
PyErr_SetExcFromWindowsErr (*C function*), 53
PyErr_SetExcFromWindowsErrWithFilename (*C function*), 54
PyErr_SetExcFromWindowsErrWithFilenameObject (*C function*), 54
PyErr_SetExcFromWindowsErrWithFilenameObjects (*C function*), 54
PyErr_SetExcInfo (*C function*), 58
PyErr_SetFromErrno (*C function*), 53
PyErr_SetFromErrnoWithFilename (*C function*), 53
PyErr_SetFromErrnoWithFilenameObject (*C function*), 53
PyErr_SetFromErrnoWithFilenameObjects (*C function*), 53
PyErr_SetFromWindowsErr (*C function*), 53
PyErr_SetFromWindowsErrWithFilename (*C function*), 53
PyErr_SetHandledException (*C function*), 58
PyErr_SetImportError (*C function*), 54
PyErr_SetImportErrorSubclass (*C function*), 54

PyErr_SetInterrupt (*C function*), 59
PyErr_SetInterruptEx (*C function*), 59
PyErr_SetNone (*C function*), 52
PyErr_SetObject (*C function*), 52
PyErr_SetRaisedException (*C function*), 56
PyErr_SetString (*C function*), 10, 52
PyErr_SyntaxLocation (*C function*), 54
PyErr_SyntaxLocationEx (*C function*), 54
PyErr_SyntaxLocationObject (*C function*), 54
PyErr_WarnEx (*C function*), 55
PyErr_WarnExplicit (*C function*), 55
PyErr_WarnExplicitObject (*C function*), 55
PyErr_WarnFormat (*C function*), 55
PyErr_WriteUnraisable (*C function*), 52
PyEval_AcquireLock (*C function*), 207
PyEval_AcquireThread (*C function*), 206
PyEval_AcquireThread(), 202
PyEval_EvalCode (*C function*), 44
PyEval_EvalCodeEx (*C function*), 44
PyEval_EvalFrame (*C function*), 44
PyEval_EvalFrameEx (*C function*), 44
PyEval_GetBuiltins (*C function*), 85
PyEval_GetFrame (*C function*), 85
PyEval_GetFuncDesc (*C function*), 86
PyEval_GetFuncName (*C function*), 85
PyEval_GetGlobals (*C function*), 85
PyEval_GetLocals (*C function*), 85
PyEval_InitThreads (*C function*), 202
PyEval_InitThreads(), 195
PyEval_MergeCompilerFlags (*C function*), 44
PyEval_ReleaseLock (*C function*), 207
PyEval_ReleaseThread (*C function*), 206
PyEval_ReleaseThread(), 202
PyEval_RestoreThread (*C function*), 201, 202
PyEval_RestoreThread(), 202
PyEval_SaveThread (*C function*), 201, 202
PyEval_SaveThread(), 202
PyEval_SetProfile (*C function*), 212
PyEval_SetProfileAllThreads (*C function*), 212
PyEval_SetTrace (*C function*), 212
PyEval_SetTraceAllThreads (*C function*), 212
PyEval_ThreadsInitialized (*C function*), 202
PyExc_ArithmeticError (*C var*), 63
PyExc_AssertionError (*C var*), 63
PyExc_AttributeError (*C var*), 63
PyExc_BaseException (*C var*), 63
PyExc_BlockingIOError (*C var*), 63
PyExc_BrokenPipeError (*C var*), 63
PyExc_BufferError (*C var*), 63
PyExc_BytesWarning (*C var*), 64
PyExc_ChildProcessError (*C var*), 63
PyExc_ConnectionAbortedError (*C var*), 63
PyExc_ConnectionError (*C var*), 63
PyExc_ConnectionRefusedError (*C var*), 63
PyExc_ConnectionResetError (*C var*), 63
PyExc_DeprecationWarning (*C var*), 64
PyExc_EnvironmentError (*C var*), 64
PyExc_EOFError (*C var*), 63
PyExc_Exception (*C var*), 63
PyExc_FileExistsError (*C var*), 63
PyExc_FileNotFoundError (*C var*), 63
PyExc_FloatingPointError (*C var*), 63
PyExc_FutureWarning (*C var*), 64
PyExc_GeneratorExit (*C var*), 63
PyExc_ImportError (*C var*), 63
PyExc_ImportWarning (*C var*), 64
PyExc_IndentationError (*C var*), 63
PyExc_IndexError (*C var*), 63
PyExc_InterruptedError (*C var*), 63
PyExc_IOError (*C var*), 64
PyExc_IsADirectoryError (*C var*), 63
PyExc_KeyboardInterrupt (*C var*), 63
PyExc_KeyError (*C var*), 63
PyExc_LookupError (*C var*), 63
PyExc_MemoryError (*C var*), 63
PyExc_ModuleNotFoundError (*C var*), 63
PyExc_NameError (*C var*), 63
PyExc_NotADirectoryError (*C var*), 63
PyExc_NotImplementedError (*C var*), 63
PyExc_OSError (*C var*), 63
PyExc_OverflowError (*C var*), 63
PyExc_PendingDeprecationWarning (*C var*), 64
PyExc_PermissionError (*C var*), 63
PyExc_ProcessLookupError (*C var*), 63
PyExc_RecursionError (*C var*), 63
PyExc_ReferenceError (*C var*), 63
PyExc_ResourceWarning (*C var*), 64
PyExc_RuntimeError (*C var*), 63
PyExc_RuntimeWarning (*C var*), 64
PyExc_StopAsyncIteration (*C var*), 63
PyExc_StopIteration (*C var*), 63
PyExc_SyntaxError (*C var*), 63
PyExc_SyntaxWarning (*C var*), 64
PyExc_SystemError (*C var*), 63
PyExc_SystemExit (*C var*), 63
PyExc_TabError (*C var*), 63
PyExc_TimeoutError (*C var*), 63
PyExc_TypeError (*C var*), 63
PyExc_UnboundLocalError (*C var*), 63
PyExc_UnicodeDecodeError (*C var*), 63
PyExc_UnicodeEncodeError (*C var*), 63
PyExc_UnicodeError (*C var*), 63
PyExc_UnicodeTranslateError (*C var*), 63
PyExc_UnicodeWarning (*C var*), 64
PyExc_UserWarning (*C var*), 64
PyExc_ValueError (*C var*), 63
PyExc_Warning (*C var*), 64
PyExc_WindowsError (*C var*), 64
PyExc_ZeroDivisionError (*C var*), 63
PyException_GetArgs (*C function*), 60
PyException_GetCause (*C function*), 60
PyException_GetContext (*C function*), 60
PyException_GetTraceback (*C function*), 60
PyException_SetArgs (*C function*), 60
PyException_SetCause (*C function*), 60
PyException_SetContext (*C function*), 60

PyException_SetTraceback (*C function*), 60
PyFile_FromFd (*C function*), 166
PyFile_GetLine (*C function*), 166
PyFile_SetOpenCodeHook (*C function*), 167
PyFile_WriteObject (*C function*), 167
PyFile_WriteString (*C function*), 167
PyFloat_AS_DOUBLE (*C function*), 126
PyFloat_AsDouble (*C function*), 126
PyFloat_Check (*C function*), 125
PyFloat_CheckExact (*C function*), 125
PyFloat_FromDouble (*C function*), 126
PyFloat_FromString (*C function*), 126
PyFloat_GetInfo (*C function*), 126
PyFloat_GetMax (*C function*), 126
PyFloat_GetMin (*C function*), 126
PyFloat_Pack2 (*C function*), 127
PyFloat_Pack4 (*C function*), 127
PyFloat_Pack8 (*C function*), 127
PyFloat_Type (*C var*), 125
PyFloat_Unpack2 (*C function*), 127
PyFloat_Unpack4 (*C function*), 127
PyFloat_Unpack8 (*C function*), 127
PyFloatObject (*C type*), 125
PyFrame_Check (*C function*), 181
PyFrame_GetBack (*C function*), 181
PyFrame_GetBuiltins (*C function*), 181
PyFrame_GetCode (*C function*), 181
PyFrame_GetGenerator (*C function*), 181
PyFrame_GetGlobals (*C function*), 182
PyFrame_GetLasti (*C function*), 182
PyFrame_GetLineNumber (*C function*), 182
PyFrame_GetLocals (*C function*), 182
PyFrame_GetVar (*C function*), 182
PyFrame_GetVarString (*C function*), 182
PyFrame_Type (*C var*), 181
PyFrameObject (*C type*), 181
PyFrozenSet_Check (*C function*), 158
PyFrozenSet_CheckExact (*C function*), 158
PyFrozenSet_New (*C function*), 158
PyFrozenSet_Type (*C var*), 157
PyFunction_AddWatcher (*C function*), 160
PyFunction_Check (*C function*), 159
PyFunction_ClearWatcher (*C function*), 160
PyFunction_GetAnnotations (*C function*), 160
PyFunction_GetClosure (*C function*), 160
PyFunction_GetCode (*C function*), 159
PyFunction_GetDefaults (*C function*), 159
PyFunction_GetGlobals (*C function*), 159
PyFunction_GetModule (*C function*), 159
PyFunction_New (*C function*), 159
PyFunction_NewWithQualName (*C function*), 159
PyFunction_SetAnnotations (*C function*), 160
PyFunction_SetClosure (*C function*), 160
PyFunction_SetDefaults (*C function*), 159
PyFunction_SetVectorcall (*C function*), 160
PyFunction_Type (*C var*), 159
PyFunction_WatchCallback (*C type*), 160
PyFunction_WatchEvent (*C type*), 160
PyFunctionObject (*C type*), 159
PyGC_Collect (*C function*), 296
PyGC_Disable (*C function*), 296
PyGC_Enable (*C function*), 296
PyGC_IsEnabled (*C function*), 296
PyGen_Check (*C function*), 183
PyGen_CheckExact (*C function*), 183
PyGen_New (*C function*), 183
PyGen_NewWithQualName (*C function*), 183
PyGen_Type (*C var*), 183
PyGenObject (*C type*), 183
PyGetSetDef (*C type*), 258
PyGetSetDef.closure (*C member*), 258
PyGetSetDef.doc (*C member*), 258
PyGetSetDef.get (*C member*), 258
PyGetSetDef.name (*C member*), 258
PyGetSetDef.set (*C member*), 258
PyGILState_Check (*C function*), 203
PyGILState_Ensure (*C function*), 203
PyGILState_GetThisThreadState (*C function*), 203
PyGILState_Release (*C function*), 203
PyHash_FuncDef (*C type*), 85
PyHash_FuncDef.hash_bits (*C member*), 85
PyHash_FuncDef.name (*C member*), 85
PyHash_FuncDef.seed_bits (*C member*), 85
PyHash_GetFuncDef (*C function*), 85
PyImport_AddModule (*C function*), 71
PyImport_AddModuleObject (*C function*), 71
PyImport_AppendInittab (*C function*), 73
PyImport_ExecCodeModule (*C function*), 71
PyImport_ExecCodeModuleEx (*C function*), 72
PyImport_ExecCodeModuleObject (*C function*), 72
PyImport_ExecCodeModuleWithPathnames (*C function*), 72
PyImport_ExtendInittab (*C function*), 74
PyImport_FrozenModules (*C var*), 73
PyImport_GetImporter (*C function*), 73
PyImport_GetMagicNumber (*C function*), 72
PyImport_GetMagicTag (*C function*), 72
PyImport_GetModule (*C function*), 73
PyImport_GetModuleDict (*C function*), 73
PyImport_Import (*C function*), 71
PyImport_ImportFrozenModule (*C function*), 73
PyImport_ImportFrozenModuleObject (*C function*), 73
PyImport_ImportModule (*C function*), 70
PyImport_ImportModuleEx (*C function*), 71
PyImport_ImportModuleLevel (*C function*), 71
PyImport_ImportModuleLevelObject (*C function*), 71
PyImport_ImportModuleNoBlock (*C function*), 70
PyImport_ReloadModule (*C function*), 71
PyIndex_Check (*C function*), 102
PyInstanceMethod_Check (*C function*), 161
PyInstanceMethod_Function (*C function*), 161
PyInstanceMethod_GET_FUNCTION (*C function*), 161

PyInstanceMethod_New (*C function*), 161
PyInstanceMethod_Type (*C var*), 161
PyInterpreterConfig (*C type*), 207
PyInterpreterConfig_DEFAULT_GIL (*C macro*), 208
PyInterpreterConfig_OWN_GIL (*C macro*), 208
PyInterpreterConfig_SHARED_GIL (*C macro*), 208
PyInterpreterConfig.allow_daemon_threads (*C member*), 208
PyInterpreterConfig.allow_exec (*C member*), 208
PyInterpreterConfig.allow_fork (*C member*), 207
PyInterpreterConfig.allow_threads (*C member*), 208
PyInterpreterConfig.check_multi_interp_extensions (*C member*), 208
PyInterpreterConfig.gil (*C member*), 208
PyInterpreterConfig.use_main_obmalloc (*C member*), 207
PyInterpreterState (*C type*), 202
PyInterpreterState_Clear (*C function*), 204
PyInterpreterState_Delete (*C function*), 204
PyInterpreterState_Get (*C function*), 205
PyInterpreterState_GetDict (*C function*), 205
PyInterpreterState_GetID (*C function*), 205
PyInterpreterState_Head (*C function*), 213
PyInterpreterState_Main (*C function*), 213
PyInterpreterState_New (*C function*), 204
PyInterpreterState_Next (*C function*), 213
PyInterpreterState_ThreadHead (*C function*), 213
PyIter_Check (*C function*), 105
PyIter_Next (*C function*), 105
PyIter_Send (*C function*), 106
PyList_Append (*C function*), 153
PyList_AsTuple (*C function*), 153
PyList_Check (*C function*), 152
PyList_CheckExact (*C function*), 152
PyList_GET_ITEM (*C function*), 152
PyList_GET_SIZE (*C function*), 152
PyList_GetItem (*C function*), 9, 152
PyList_GetSlice (*C function*), 153
PyList_Insert (*C function*), 153
PyList_New (*C function*), 152
PyList_Reverse (*C function*), 153
PyList_SET_ITEM (*C function*), 152
PyList_SetItem (*C function*), 8, 152
PyList_SetSlice (*C function*), 153
PyList_Size (*C function*), 152
PyList_Sort (*C function*), 153
PyList_Type (*C var*), 152
PyListObject (*C type*), 152
PyLong_AS_LONG (*C function*), 122
PyLong_AsDouble (*C function*), 124
PyLong_AsLong (*C function*), 122
PyLong_AsLongAndOverflow (*C function*), 122
PyLong_AsLongLong (*C function*), 123
PyLong_AsLongLongAndOverflow (*C function*), 123
PyLong_AsSize_t (*C function*), 123
PyLong_AsSsize_t (*C function*), 123
PyLong_AsUnsignedLong (*C function*), 123
PyLong_AsUnsignedLongLong (*C function*), 123
PyLong_AsUnsignedLongLongMask (*C function*), 124
PyLong_AsUnsignedLongMask (*C function*), 123
PyLong_AsVoidPtr (*C function*), 124
PyLong_Check (*C function*), 121
PyLong_CheckExact (*C function*), 121
PyLong_FromDouble (*C function*), 122
PyLong_FromLong (*C function*), 121
PyLong_FromLongLong (*C function*), 121
PyLong_FromSize_t (*C function*), 121
PyLong_FromSsize_t (*C function*), 121
PyLong_FromString (*C function*), 122
PyLong_FromUnicodeObject (*C function*), 122
PyLong_FromUnsignedLong (*C function*), 121
PyLong_FromUnsignedLongLong (*C function*), 122
PyLong_FromVoidPtr (*C function*), 122
PyLong_GetInfo (*C function*), 124
PyLong_Type (*C var*), 121
PyLongObject (*C type*), 121
PyMapping_Check (*C function*), 104
PyMapping_DelItem (*C function*), 104
PyMapping_DelItemString (*C function*), 104
PyMapping_GetItemString (*C function*), 104
PyMapping_HasKey (*C function*), 104
PyMapping_HasKeyString (*C function*), 104
PyMapping_Items (*C function*), 105
PyMapping_Keys (*C function*), 105
PyMapping_Length (*C function*), 104
PyMapping_SetItemString (*C function*), 104
PyMapping_Size (*C function*), 104
PyMapping_Values (*C function*), 105
PyMappingMethods (*C type*), 287
PyMappingMethods.mp_ass_subscript (*C member*), 287
PyMappingMethods.mp_length (*C member*), 287
PyMappingMethods.mp_subscript (*C member*), 287
PyMarshal_ReadLastObjectFromFile (*C function*), 75
PyMarshal_ReadLongFromFile (*C function*), 74
PyMarshal_ReadObjectFromFile (*C function*), 74
PyMarshal_ReadObjectFromString (*C function*), 75
PyMarshal_ReadShortFromFile (*C function*), 74
PyMarshal_WriteLongToFile (*C function*), 74
PyMarshal_WriteObjectToFile (*C function*), 74
PyMarshal_WriteObjectToString (*C function*), 74
PyMem_Calloc (*C function*), 241
PyMem_Del (*C function*), 242
PYMEM_DOMAIN_MEM (*C macro*), 244
PYMEM_DOMAIN_OBJ (*C macro*), 244
PYMEM_DOMAIN_RAW (*C macro*), 244

PyMem_Free (*C function*), 242
PyMem_GetAllocator (*C function*), 244
PyMem_Malloc (*C function*), 241
PyMem_New (*C macro*), 242
PyMem_RawCalloc (*C function*), 240
PyMem_RawFree (*C function*), 241
PyMem_RawMalloc (*C function*), 240
PyMem_RawRealloc (*C function*), 241
PyMem_Realloc (*C function*), 241
PyMem_Resize (*C macro*), 242
PyMem_SetAllocator (*C function*), 245
PyMem_SetupDebugHooks (*C function*), 245
PyMemAllocatorDomain (*C type*), 244
PyMemAllocatorEx (*C type*), 244
PyMember_GetOne (*C function*), 255
PyMember_SetOne (*C function*), 256
PyMemberDef (*C type*), 255
PyMemberDef.doc (*C member*), 255
PyMemberDef.flags (*C member*), 255
PyMemberDef.name (*C member*), 255
PyMemberDef.offset (*C member*), 255
PyMemberDef.type (*C member*), 255
PyMemoryView_Check (*C function*), 178
PyMemoryView_FromBuffer (*C function*), 178
PyMemoryView_FromMemory (*C function*), 178
PyMemoryView_FromObject (*C function*), 178
PyMemoryView_GET_BASE (*C function*), 178
PyMemoryView_GET_BUFFER (*C function*), 178
PyMemoryView_GetContiguous (*C function*), 178
PyMethod_Check (*C function*), 161
PyMethod_Function (*C function*), 162
PyMethod_GET_FUNCTION (*C function*), 162
PyMethod_GET_SELF (*C function*), 162
PyMethod_New (*C function*), 161
PyMethod_Self (*C function*), 162
PyMethod_Type (*C var*), 161
PyMethodDef (*C type*), 252
PyMethodDef.ml_doc (*C member*), 253
PyMethodDef.ml_flags (*C member*), 253
PyMethodDef.ml_meth (*C member*), 253
PyMethodDef.ml_name (*C member*), 253
PyMODINIT_FUNC (*C macro*), 4
PyModule_AddFunctions (*C function*), 172
PyModule_AddIntConstant (*C function*), 174
PyModule_AddIntMacro (*C macro*), 174
PyModule_AddObject (*C function*), 173
PyModule_AddObjectRef (*C function*), 172
PyModule_AddStringConstant (*C function*), 174
PyModule_AddStringMacro (*C macro*), 174
PyModule_AddType (*C function*), 174
PyModule_Check (*C function*), 167
PyModule_CheckExact (*C function*), 167
PyModule_Create (*C function*), 170
PyModule_Create2 (*C function*), 170
PyModule_ExecDef (*C function*), 172
PyModule_FromDefAndSpec (*C function*), 172
PyModule_FromDefAndSpec2 (*C function*), 172
PyModule_GetDef (*C function*), 168
PyModule_GetDict (*C function*), 167
PyModule_GetFilename (*C function*), 168
PyModule_GetFilenameObject (*C function*), 168
PyModule_GetName (*C function*), 168
PyModule_GetNameObject (*C function*), 168
PyModule_GetState (*C function*), 168
PyModule_New (*C function*), 167
PyModule_NewObject (*C function*), 167
PyModule_SetDocString (*C function*), 172
PyModule_Type (*C var*), 167
PyModuleDef (*C type*), 168
PyModuleDef_Init (*C function*), 170
PyModuleDef_Slot (*C type*), 170
PyModuleDef_Slot.slot (*C member*), 170
PyModuleDef_Slot.value (*C member*), 170
PyModuleDef.m_base (*C member*), 168
PyModuleDef.m_clear (*C member*), 169
PyModuleDef.m_doc (*C member*), 168
PyModuleDef.m_free (*C member*), 169
PyModuleDef.m_methods (*C member*), 169
PyModuleDef.m_name (*C member*), 168
PyModuleDef.m_size (*C member*), 168
PyModuleDef.m_slots (*C member*), 169
PyModuleDef.m_slots.m_reload (*C member*), 169
PyModuleDef.m_traverse (*C member*), 169
PyNumber_Absolute (*C function*), 100
PyNumber_Add (*C function*), 99
PyNumber_And (*C function*), 100
PyNumber_AsSsize_t (*C function*), 102
PyNumber_Check (*C function*), 99
PyNumber_Divmod (*C function*), 99
PyNumber_Float (*C function*), 101
PyNumber_FloorDivide (*C function*), 99
PyNumber_Index (*C function*), 101
PyNumber_InPlaceAdd (*C function*), 100
PyNumber_InPlaceAnd (*C function*), 101
PyNumber_InPlaceFloorDivide (*C function*), 100
PyNumber_InPlaceLshift (*C function*), 101
PyNumber_InPlaceMatrixMultiply (*C function*), 100
PyNumber_InPlaceMultiply (*C function*), 100
PyNumber_InPlaceOr (*C function*), 101
PyNumber_InPlacePower (*C function*), 101
PyNumber_InPlaceRemainder (*C function*), 101
PyNumber_InPlaceRshift (*C function*), 101
PyNumber_InPlaceSubtract (*C function*), 100
PyNumber_InPlaceTrueDivide (*C function*), 100
PyNumber_InPlaceXor (*C function*), 101
PyNumber_Invert (*C function*), 100
PyNumber_Long (*C function*), 101
PyNumber_Lshift (*C function*), 100
PyNumber_MatrixMultiply (*C function*), 99
PyNumber_Multiply (*C function*), 99
PyNumber_Negative (*C function*), 99
PyNumber_Or (*C function*), 100
PyNumber_Positive (*C function*), 99
PyNumber_Power (*C function*), 99
PyNumber_Remainder (*C function*), 99

PyNumber_Rshift (*C function*), 100
PyNumber_Subtract (*C function*), 99
PyNumber_ToBase (*C function*), 101
PyNumber_TrueDivide (*C function*), 99
PyNumber_Xor (*C function*), 100
PyNumberMethods (*C type*), 284
PyNumberMethods.nb_absolute (*C member*), 286
PyNumberMethods.nb_add (*C member*), 285
PyNumberMethods.nb_and (*C member*), 286
PyNumberMethods.nb_bool (*C member*), 286
PyNumberMethods.nb_divmod (*C member*), 286
PyNumberMethods.nb_float (*C member*), 286
PyNumberMethods.nb_floor_divide (*C member*), 286
PyNumberMethods.nb_index (*C member*), 286
PyNumberMethods.nb_inplace_add (*C member*), 286
PyNumberMethods.nb_inplace_and (*C member*), 286
PyNumberMethods.nb_inplace_floor_divide (*C member*), 286
PyNumberMethods.nb_inplace_lshift (*C member*), 286
PyNumberMethods.nb_inplace_matrix_multiply (*C member*), 287
PyNumberMethods.nb_inplace_multiply (*C member*), 286
PyNumberMethods.nb_inplace_or (*C member*), 286
PyNumberMethods.nb_inplace_power (*C member*), 286
PyNumberMethods.nb_inplace_remainder (*C member*), 286
PyNumberMethods.nb_inplace_rshift (*C member*), 286
PyNumberMethods.nb_inplace_subtract (*C member*), 286
PyNumberMethods.nb_inplace_true_divide (*C member*), 286
PyNumberMethods.nb_inplace_xor (*C member*), 286
PyNumberMethods.nb_int (*C member*), 286
PyNumberMethods.nb_invert (*C member*), 286
PyNumberMethods.nb_lshift (*C member*), 286
PyNumberMethods.nb_matrix_multiply (*C member*), 287
PyNumberMethods.nb_multiply (*C member*), 286
PyNumberMethods.nb_negative (*C member*), 286
PyNumberMethods.nb_or (*C member*), 286
PyNumberMethods.nb_positive (*C member*), 286
PyNumberMethods.nb_power (*C member*), 286
PyNumberMethods.nb_remainder (*C member*), 286
PyNumberMethods.nb_reserved (*C member*), 286
PyNumberMethods.nb_rshift (*C member*), 286
PyNumberMethods.nb_subtract (*C member*), 286
PyNumberMethods.nb_true_divide (*C member*), 286
PyNumberMethods.nb_xor (*C member*), 286
PyObject (*C type*), 250
PyObject_AsCharBuffer (*C function*), 113
PyObject_ASCII (*C function*), 91
PyObject_AsFileDescriptor (*C function*), 166
PyObject_AsReadBuffer (*C function*), 113
PyObject_AsWriteBuffer (*C function*), 113
PyObject_Bytes (*C function*), 92
PyObject_Call (*C function*), 96
PyObject_CallFunction (*C function*), 97
PyObject_CallFunctionObjArgs (*C function*), 97
PyObject_CallMethod (*C function*), 97
PyObject_CallMethodNoArgs (*C function*), 98
PyObject_CallMethodObjArgs (*C function*), 98
PyObject_CallMethodOneArg (*C function*), 98
PyObject_CallNoArgs (*C function*), 97
PyObject_CallObject (*C function*), 97
PyObject_Calloc (*C function*), 243
PyObject_CallOneArg (*C function*), 97
PyObject_CheckBuffer (*C function*), 111
PyObject_CheckReadBuffer (*C function*), 113
PyObject_ClearWeakRefs (*C function*), 179
PyObject_CopyData (*C function*), 112
PyObject_Del (*C function*), 249
PyObject_DelAttr (*C function*), 90
PyObject_DelAttrString (*C function*), 90
PyObject_DelItem (*C function*), 93
PyObject_DelItemString (*C function*), 93
PyObject_Dir (*C function*), 93
PyObject_Format (*C function*), 91
PyObject_Free (*C function*), 243
PyObject_GC_Del (*C function*), 295
PyObject_GC_IsFinalized (*C function*), 295
PyObject_GC_IsTracked (*C function*), 295
PyObject_GC_New (*C macro*), 294
PyObject_GC_NewVar (*C macro*), 294
PyObject_GC_Resize (*C macro*), 295
PyObject_GC_Track (*C function*), 295
PyObject_GC_UnTrack (*C function*), 295
PyObject_GenericGetAttr (*C function*), 90
PyObject_GenericGetDict (*C function*), 91
PyObject_GenericSetAttr (*C function*), 90
PyObject_GenericSetDict (*C function*), 91
PyObject_GetAIter (*C function*), 93
PyObject_GetArenaAllocator (*C function*), 247
PyObject_GetAttr (*C function*), 90
PyObject_GetAttrString (*C function*), 90
PyObject_GetBuffer (*C function*), 111
PyObject_GetItem (*C function*), 93
PyObject_GetItemData (*C function*), 94
PyObject_GetIter (*C function*), 93
PyObject_GetTypeData (*C function*), 94
PyObject_HasAttr (*C function*), 89
PyObject_HasAttrString (*C function*), 89
PyObject_Hash (*C function*), 92
PyObject_HashNotImplemented (*C function*), 92
PyObject_HEAD (*C macro*), 250
PyObject_HEAD_INIT (*C macro*), 251
PyObject_Init (*C function*), 249

PyObject_InitVar (*C function*), 249
PyObject_IS_GC (*C function*), 295
PyObject_IsInstance (*C function*), 92
PyObject_IsSubclass (*C function*), 92
PyObject_IsTrue (*C function*), 92
PyObject_Length (*C function*), 93
PyObject_LengthHint (*C function*), 93
PyObject_Malloc (*C function*), 243
PyObject_New (*C macro*), 249
PyObject_NewVar (*C macro*), 249
PyObject_Not (*C function*), 92
PyObject._ob_next (*C member*), 265
PyObject._ob_prev (*C member*), 265
PyObject_Print (*C function*), 89
PyObject_Realloc (*C function*), 243
PyObject_Repr (*C function*), 91
PyObject_RichCompare (*C function*), 91
PyObject_RichCompareBool (*C function*), 91
PyObject_SelfIter (*C function*), 93
PyObject_SetArenaAllocator (*C function*), 247
PyObject_SetAttr (*C function*), 90
PyObject_SetAttrString (*C function*), 90
PyObject_SetItem (*C function*), 93
PyObject_Size (*C function*), 93
PyObject_Str (*C function*), 92
PyObject_Type (*C function*), 93
PyObject_TypeCheck (*C function*), 93
PyObject_VAR_HEAD (*C macro*), 250
PyObject_Vectorcall (*C function*), 98
PyObject_VectorcallDict (*C function*), 98
PyObject_VectorcallMethod (*C function*), 98
PyObjectArenaAllocator (*C type*), 247
PyObject.ob_refcnt (*C member*), 264
PyObject.ob_type (*C member*), 264
PyOS_AfterFork (*C function*), 66
PyOS_AfterFork_Child (*C function*), 66
PyOS_AfterFork_Parent (*C function*), 65
PyOS_BeforeFork (*C function*), 65
PyOS_CheckStack (*C function*), 66
PyOS_double_to_string (*C function*), 84
PyOS_FSPath (*C function*), 65
PyOS_getsig (*C function*), 66
PyOS_InputHook (*C var*), 42
PyOS_ReadlineFunctionPointer (*C var*), 43
PyOS_setsig (*C function*), 66
PyOS_sighandler_t (*C type*), 66
PyOS_snprintf (*C function*), 83
PyOS_stricmp (*C function*), 84
PyOS_string_to_double (*C function*), 84
PyOS_strnicmp (*C function*), 84
PyOS_strtol (*C function*), 84
PyOS_strtoul (*C function*), 83
PyOS_vsnprintf (*C function*), 83
PyPreConfig (*C type*), 220
PyPreConfig_InitIsolatedConfig (*C function*), 220
PyPreConfig_InitPythonConfig (*C function*), 220
PyPreConfig.allocator (*C member*), 220
PyPreConfig.coerce_c_locale (*C member*), 220
PyPreConfig.coerce_c_locale_warn (*C member*), 221
PyPreConfig.configure_locale (*C member*), 220
PyPreConfig.dev_mode (*C member*), 221
PyPreConfig.isolated (*C member*), 221
PyPreConfig.legacy_windows_fs_encoding (*C member*), 221
PyPreConfig.parse_argv (*C member*), 221
PyPreConfig.use_environment (*C member*), 221
PyPreConfig.utf8_mode (*C member*), 221
PyProperty_Type (*C var*), 176
PyRun_AnyFile (*C function*), 41
PyRun_AnyFileEx (*C function*), 41
PyRun_AnyFileExFlags (*C function*), 42
PyRun_AnyFileFlags (*C function*), 41
PyRun_File (*C function*), 43
PyRun_FileEx (*C function*), 43
PyRun_FileExFlags (*C function*), 43
PyRun_FileFlags (*C function*), 43
PyRun_InteractiveLoop (*C function*), 42
PyRun_InteractiveLoopFlags (*C function*), 42
PyRun_InteractiveOne (*C function*), 42
PyRun_InteractiveOneFlags (*C function*), 42
PyRun_SimpleFile (*C function*), 42
PyRun_SimpleFileEx (*C function*), 42
PyRun_SimpleFileExFlags (*C function*), 42
PyRun_SimpleString (*C function*), 42
PyRun_SimpleStringFlags (*C function*), 42
PyRun_String (*C function*), 43
PyRun_StringFlags (*C function*), 43
PySendResult (*C type*), 106
PySeqIter_Check (*C function*), 175
PySeqIter_New (*C function*), 175
PySeqIter_Type (*C var*), 175
PySequence_Check (*C function*), 102
PySequence_Concat (*C function*), 102
PySequence_Contains (*C function*), 103
PySequence_Count (*C function*), 103
PySequence_DelItem (*C function*), 103
PySequence_DelSlice (*C function*), 103
PySequence_Fast (*C function*), 103
PySequence_Fast_GET_ITEM (*C function*), 103
PySequence_Fast_GET_SIZE (*C function*), 103
PySequence_Fast_ITEMS (*C function*), 103
PySequence_GetItem (*C function*), 9, 102
PySequence_GetSlice (*C function*), 102
PySequence_Index (*C function*), 103
PySequence_InPlaceConcat (*C function*), 102
PySequence_InPlaceRepeat (*C function*), 102
PySequence_ITEM (*C function*), 104
PySequence_Length (*C function*), 102
PySequence_List (*C function*), 103
PySequence_Repeat (*C function*), 102
PySequence_SetItem (*C function*), 102
PySequence_SetSlice (*C function*), 103
PySequence_Size (*C function*), 102
PySequence_Tuple (*C function*), 103

PySequenceMethods (*C type*), 287
PySequenceMethods.sq_ass_item (*C member*), 287
PySequenceMethods.sq_concat (*C member*), 287
PySequenceMethods.sq_contains (*C member*), 287
PySequenceMethods.sq_inplace_concat (*C member*), 288
PySequenceMethods.sq_inplace_repeat (*C member*), 288
PySequenceMethods.sq_item (*C member*), 287
PySequenceMethods.sq_length (*C member*), 287
PySequenceMethods.sq_repeat (*C member*), 287
PySet_Add (*C function*), 158
PySet_Check (*C function*), 157
PySet_CheckExact (*C function*), 158
PySet_Clear (*C function*), 159
PySet_Contains (*C function*), 158
PySet_Discard (*C function*), 158
PySet_GET_SIZE (*C function*), 158
PySet_New (*C function*), 158
PySet_Pop (*C function*), 159
PySet_Size (*C function*), 158
PySet_Type (*C var*), 157
PySetObject (*C type*), 157
PySignal_SetWakeupFd (*C function*), 59
PySlice_AdjustIndices (*C function*), 177
PySlice_Check (*C function*), 176
PySlice_GetIndices (*C function*), 176
PySlice_GetIndicesEx (*C function*), 176
PySlice_New (*C function*), 176
PySlice_Type (*C var*), 176
PySlice_Unpack (*C function*), 177
PyState_AddModule (*C function*), 175
PyState_FindModule (*C function*), 175
PyState_RemoveModule (*C function*), 175
PyStatus (*C type*), 218
PyStatus_Error (*C function*), 219
PyStatus_Exception (*C function*), 219
PyStatus_Exit (*C function*), 219
PyStatus_IsError (*C function*), 219
PyStatus_IsExit (*C function*), 219
PyStatus_NoMemory (*C function*), 219
PyStatus_Ok (*C function*), 219
PyStatus.err_msg (*C member*), 219
PyStatus.exitcode (*C member*), 219
PyStatus.func (*C member*), 219
PyStructSequence_Desc (*C type*), 150
PyStructSequence_Desc.doc (*C member*), 151
PyStructSequence_Desc.fields (*C member*), 151
PyStructSequence_Desc.n_in_sequence (*C member*), 151
PyStructSequence_Desc.name (*C member*), 150
PyStructSequence_Field (*C type*), 151
PyStructSequence_Field.doc (*C member*), 151
PyStructSequence_Field.name (*C member*), 151
PyStructSequence_GET_ITEM (*C function*), 151
PyStructSequence_GetItem (*C function*), 151
PyStructSequence_InitType (*C function*), 150
PyStructSequence_InitType2 (*C function*), 150
PyStructSequence_New (*C function*), 151
PyStructSequence_NewType (*C function*), 150
PyStructSequence_SET_ITEM (*C function*), 151
PyStructSequence_SetItem (*C function*), 151
PyStructSequence_UnnamedField (*C var*), 151
PySys_AddAuditHook (*C function*), 69
PySys_AddWarnOption (*C function*), 68
PySys_AddWarnOptionUnicode (*C function*), 68
PySys_AddXOption (*C function*), 69
PySys_Audit (*C function*), 69
PySys_FormatStderr (*C function*), 69
PySys_FormatStdout (*C function*), 69
PySys_GetObject (*C function*), 68
PySys_GetXOptions (*C function*), 69
PySys_ResetWarnOptions (*C function*), 68
PySys_SetArgv (*C function*), 195, 199
PySys_SetArgvEx (*C function*), 195, 199
PySys_SetObject (*C function*), 68
PySys_SetPath (*C function*), 68
PySys_WriteStderr (*C function*), 69
PySys_WriteStdout (*C function*), 68
Python 3000 (파이썬 *3000*), **313**
Python 향상 제안
    PEP 1, 312
    PEP 7, 3, 6
    PEP 238, 45, 306
    PEP 278, 316
    PEP 302, 309
    PEP 343, 304
    PEP 353, 10
    PEP 362, 302, 312
    PEP 383, 139, 140
    PEP 387, 15, 16
    PEP 393, 132
    PEP 411, 313
    PEP 420, 311, 312
    PEP 432, 236, 237
    PEP 442, 284
    PEP 443, 307
    PEP 451, 171
    PEP 456, 85
    PEP 483, 307
    PEP 484, 301, 306, 307, 315, 316
    PEP 489, 171, 208
    PEP 492, 302, 304
    PEP 498, 305
    PEP 519, 312
    PEP 523, 182, 206
    PEP 525, 302
    PEP 526, 301, 316
    PEP 528, 194, 228
    PEP 529, 140, 194
    PEP 538, 235
    PEP 539, 213
    PEP 540, 235
    PEP 552, 225

PEP 554, 209
PEP 578, 70
PEP 585, 307
PEP 587, 217
PEP 590, 94
PEP 623, 132
PEP 0626#out-of-process-debuggers-and-profilers, 164
PEP 634, 274
PEP 0683, 47, 48, 307
PEP 3116, 316
PEP 3119, 92
PEP 3121, 169
PEP 3147, 73
PEP 3151, 64
PEP 3155, 313
PYTHONCOERCECLOCALE, 235
PYTHONDEBUG, 192, 229
PYTHONDEVMODE, 225
PYTHONDONTWRITEBYTECODE, 192, 232
PYTHONDUMPREFS, 225, 265
PYTHONEXECUTABLE, 230
PYTHONFAULTHANDLER, 226
PYTHONHASHSEED, 193, 227
PYTHONHOME, 12, 193, 200, 227
Pythonic (파이썬다운), **313**
PYTHONINSPECT, 193, 227
PYTHONINTMAXSTRDIGITS, 227
PYTHONIOENCODING, 196, 231
PYTHONLEGACYWINDOWSFSENCODING, 194, 221
PYTHONLEGACYWINDOWSSTDIO, 194, 228
PYTHONMALLOC, 240, 243, 245, 247
PYTHONMALLOCSTATS, 228, 240
PYTHONNODEBUGRANGES, 225
PYTHONNOUSERSITE, 194, 232
PYTHONOPTIMIZE, 194, 229
PYTHONPATH, 12, 193, 228
PYTHONPERFSUPPORT, 232
PYTHONPLATLIBDIR, 228
PYTHONPROFILEIMPORTTIME, 227
PYTHONPYCACHEPREFIX, 230
PYTHONSAFEPATH, 224
PYTHONTRACEMALLOC, 231
PYTHONUNBUFFERED, 194, 224
PYTHONUTF8, 221, 235
PYTHONVERBOSE, 195, 232
PYTHONWARNINGS, 232
PyThread_create_key (*C function*), 214
PyThread_delete_key (*C function*), 215
PyThread_delete_key_value (*C function*), 215
PyThread_get_key_value (*C function*), 215
PyThread_ReInitTLS (*C function*), 215
PyThread_set_key_value (*C function*), 215
PyThread_tss_alloc (*C function*), 214
PyThread_tss_create (*C function*), 214
PyThread_tss_delete (*C function*), 214
PyThread_tss_free (*C function*), 214
PyThread_tss_get (*C function*), 214
PyThread_tss_is_created (*C function*), 214
PyThread_tss_set (*C function*), 214
PyThreadState (*C type*), 200, 202
PyThreadState_Clear (*C function*), 204
PyThreadState_Delete (*C function*), 204
PyThreadState_DeleteCurrent (*C function*), 204
PyThreadState_EnterTracing (*C function*), 205
PyThreadState_Get (*C function*), 203
PyThreadState_GetDict (*C function*), 206
PyThreadState_GetFrame (*C function*), 204
PyThreadState_GetID (*C function*), 205
PyThreadState_GetInterpreter (*C function*), 205
PyThreadState_LeaveTracing (*C function*), 205
PyThreadState_New (*C function*), 204
PyThreadState_Next (*C function*), 213
PyThreadState_SetAsyncExc (*C function*), 206
PyThreadState_Swap (*C function*), 203
PyThreadState.interp (*C member*), 202
PyTime_Check (*C function*), 186
PyTime_CheckExact (*C function*), 186
PyTime_FromTime (*C function*), 187
PyTime_FromTimeAndFold (*C function*), 187
PyTimeZone_FromOffset (*C function*), 187
PyTimeZone_FromOffsetAndName (*C function*), 187
PyTrace_C_CALL (*C var*), 212
PyTrace_C_EXCEPTION (*C var*), 212
PyTrace_C_RETURN (*C var*), 212
PyTrace_CALL (*C var*), 211
PyTrace_EXCEPTION (*C var*), 211
PyTrace_LINE (*C var*), 211
PyTrace_OPCODE (*C var*), 212
PyTrace_RETURN (*C var*), 212
PyTraceMalloc_Track (*C function*), 247
PyTraceMalloc_Untrack (*C function*), 247
PyTuple_Check (*C function*), 149
PyTuple_CheckExact (*C function*), 149
PyTuple_GET_ITEM (*C function*), 149
PyTuple_GET_SIZE (*C function*), 149
PyTuple_GetItem (*C function*), 149
PyTuple_GetSlice (*C function*), 149
PyTuple_New (*C function*), 149
PyTuple_Pack (*C function*), 149
PyTuple_SET_ITEM (*C function*), 150
PyTuple_SetItem (*C function*), 8, 150
PyTuple_Size (*C function*), 149
PyTuple_Type (*C var*), 149
PyTupleObject (*C type*), 149
PyType_AddWatcher (*C function*), 116
PyType_Check (*C function*), 115
PyType_CheckExact (*C function*), 115
PyType_ClearCache (*C function*), 115
PyType_ClearWatcher (*C function*), 116
PyType_FromMetaclass (*C function*), 118
PyType_FromModuleAndSpec (*C function*), 119
PyType_FromSpec (*C function*), 119
PyType_FromSpecWithBases (*C function*), 119
PyType_GenericAlloc (*C function*), 117
PyType_GenericNew (*C function*), 117

PyType_GetDict (*C function*), 116
PyType_GetFlags (*C function*), 115
PyType_GetModule (*C function*), 117
PyType_GetModuleByDef (*C function*), 118
PyType_GetModuleState (*C function*), 117
PyType_GetName (*C function*), 117
PyType_GetQualName (*C function*), 117
PyType_GetSlot (*C function*), 117
PyType_GetTypeDataSize (*C function*), 94
PyType_HasFeature (*C function*), 116
PyType_IS_GC (*C function*), 116
PyType_IsSubtype (*C function*), 116
PyType_Modified (*C function*), 116
PyType_Ready (*C function*), 117
PyType_Slot (*C type*), 120
PyType_Slot.pfunc (*C member*), 120
PyType_Slot.slot (*C member*), 120
PyType_Spec (*C type*), 119
PyType_Spec.basicsize (*C member*), 119
PyType_Spec.flags (*C member*), 120
PyType_Spec.itemsize (*C member*), 119
PyType_Spec.name (*C member*), 119
PyType_Spec.slots (*C member*), 120
PyType_Type (*C var*), 115
PyType_Watch (*C function*), 116
PyType_WatchCallback (*C type*), 116
PyTypeObject (*C type*), 115
PyTypeObject.tp_alloc (*C member*), 281
PyTypeObject.tp_as_async (*C member*), 268
PyTypeObject.tp_as_buffer (*C member*), 270
PyTypeObject.tp_as_mapping (*C member*), 269
PyTypeObject.tp_as_number (*C member*), 269
PyTypeObject.tp_as_sequence (*C member*), 269
PyTypeObject.tp_base (*C member*), 279
PyTypeObject.tp_bases (*C member*), 282
PyTypeObject.tp_basicsize (*C member*), 266
PyTypeObject.tp_cache (*C member*), 282
PyTypeObject.tp_call (*C member*), 269
PyTypeObject.tp_clear (*C member*), 276
PyTypeObject.tp_dealloc (*C member*), 267
PyTypeObject.tp_del (*C member*), 283
PyTypeObject.tp_descr_get (*C member*), 280
PyTypeObject.tp_descr_set (*C member*), 280
PyTypeObject.tp_dict (*C member*), 279
PyTypeObject.tp_dictoffset (*C member*), 280
PyTypeObject.tp_doc (*C member*), 275
PyTypeObject.tp_finalize (*C member*), 283
PyTypeObject.tp_flags (*C member*), 271
PyTypeObject.tp_free (*C member*), 281
PyTypeObject.tp_getattr (*C member*), 268
PyTypeObject.tp_getattro (*C member*), 270
PyTypeObject.tp_getset (*C member*), 278
PyTypeObject.tp_hash (*C member*), 269
PyTypeObject.tp_init (*C member*), 280
PyTypeObject.tp_is_gc (*C member*), 282
PyTypeObject.tp_itemsize (*C member*), 266
PyTypeObject.tp_iter (*C member*), 278
PyTypeObject.tp_iternext (*C member*), 278
PyTypeObject.tp_members (*C member*), 278
PyTypeObject.tp_methods (*C member*), 278
PyTypeObject.tp_mro (*C member*), 282
PyTypeObject.tp_name (*C member*), 265
PyTypeObject.tp_new (*C member*), 281
PyTypeObject.tp_repr (*C member*), 268
PyTypeObject.tp_richcompare (*C member*), 276
PyTypeObject.tp_setattr (*C member*), 268
PyTypeObject.tp_setattro (*C member*), 270
PyTypeObject.tp_str (*C member*), 269
PyTypeObject.tp_subclasses (*C member*), 283
PyTypeObject.tp_traverse (*C member*), 275
PyTypeObject.tp_vectorcall (*C member*), 284
PyTypeObject.tp_vectorcall_offset (*C member*), 267
PyTypeObject.tp_version_tag (*C member*), 283
PyTypeObject.tp_watched (*C member*), 284
PyTypeObject.tp_weaklist (*C member*), 283
PyTypeObject.tp_weaklistoffset (*C member*), 277
PyTZInfo_Check (*C function*), 187
PyTZInfo_CheckExact (*C function*), 187
PyUnicode_1BYTE_DATA (*C function*), 133
PyUnicode_1BYTE_KIND (*C macro*), 133
PyUnicode_2BYTE_DATA (*C function*), 133
PyUnicode_2BYTE_KIND (*C macro*), 133
PyUnicode_4BYTE_DATA (*C function*), 133
PyUnicode_4BYTE_KIND (*C macro*), 133
PyUnicode_AsASCIIString (*C function*), 145
PyUnicode_AsCharmapString (*C function*), 146
PyUnicode_AsEncodedString (*C function*), 142
PyUnicode_AsLatin1String (*C function*), 145
PyUnicode_AsMBCSString (*C function*), 146
PyUnicode_AsRawUnicodeEscapeString (*C function*), 145
PyUnicode_AsUCS4 (*C function*), 139
PyUnicode_AsUCS4Copy (*C function*), 139
PyUnicode_AsUnicodeEscapeString (*C function*), 144
PyUnicode_AsUTF8 (*C function*), 143
PyUnicode_AsUTF8AndSize (*C function*), 142
PyUnicode_AsUTF8String (*C function*), 142
PyUnicode_AsUTF16String (*C function*), 144
PyUnicode_AsUTF32String (*C function*), 143
PyUnicode_AsWideChar (*C function*), 141
PyUnicode_AsWideCharString (*C function*), 141
PyUnicode_Check (*C function*), 132
PyUnicode_CheckExact (*C function*), 132
PyUnicode_Compare (*C function*), 148
PyUnicode_CompareWithASCIIString (*C function*), 148
PyUnicode_Concat (*C function*), 147
PyUnicode_Contains (*C function*), 148
PyUnicode_CopyCharacters (*C function*), 138
PyUnicode_Count (*C function*), 148
PyUnicode_DATA (*C function*), 133
PyUnicode_Decode (*C function*), 142
PyUnicode_DecodeASCII (*C function*), 145

PyUnicode_DecodeCharmap (*C function*), 145
PyUnicode_DecodeCodePageStateful (*C function*), 146
PyUnicode_DecodeFSDefault (*C function*), 141
PyUnicode_DecodeFSDefaultAndSize (*C function*), 140
PyUnicode_DecodeLatin1 (*C function*), 145
PyUnicode_DecodeLocale (*C function*), 139
PyUnicode_DecodeLocaleAndSize (*C function*), 139
PyUnicode_DecodeMBCS (*C function*), 146
PyUnicode_DecodeMBCSStateful (*C function*), 146
PyUnicode_DecodeRawUnicodeEscape (*C function*), 145
PyUnicode_DecodeUnicodeEscape (*C function*), 144
PyUnicode_DecodeUTF7 (*C function*), 144
PyUnicode_DecodeUTF7Stateful (*C function*), 144
PyUnicode_DecodeUTF8 (*C function*), 142
PyUnicode_DecodeUTF8Stateful (*C function*), 142
PyUnicode_DecodeUTF16 (*C function*), 144
PyUnicode_DecodeUTF16Stateful (*C function*), 144
PyUnicode_DecodeUTF32 (*C function*), 143
PyUnicode_DecodeUTF32Stateful (*C function*), 143
PyUnicode_EncodeCodePage (*C function*), 146
PyUnicode_EncodeFSDefault (*C function*), 141
PyUnicode_EncodeLocale (*C function*), 139
PyUnicode_Fill (*C function*), 138
PyUnicode_Find (*C function*), 147
PyUnicode_FindChar (*C function*), 148
PyUnicode_Format (*C function*), 148
PyUnicode_FromEncodedObject (*C function*), 138
PyUnicode_FromFormat (*C function*), 136
PyUnicode_FromFormatV (*C function*), 137
PyUnicode_FromKindAndData (*C function*), 135
PyUnicode_FromObject (*C function*), 138
PyUnicode_FromString (*C function*), 136
PyUnicode_FromStringAndSize (*C function*), 135
PyUnicode_FromWideChar (*C function*), 141
PyUnicode_FSConverter (*C function*), 140
PyUnicode_FSDecoder (*C function*), 140
PyUnicode_GET_LENGTH (*C function*), 132
PyUnicode_GetDefaultEncoding (*C function*), 138
PyUnicode_GetLength (*C function*), 138
PyUnicode_InternFromString (*C function*), 149
PyUnicode_InternInPlace (*C function*), 148
PyUnicode_IsIdentifier (*C function*), 133
PyUnicode_Join (*C function*), 147
PyUnicode_KIND (*C function*), 133
PyUnicode_MAX_CHAR_VALUE (*C function*), 133
PyUnicode_New (*C function*), 135
PyUnicode_Partition (*C function*), 147
PyUnicode_READ (*C function*), 133
PyUnicode_READ_CHAR (*C function*), 133
PyUnicode_ReadChar (*C function*), 138
PyUnicode_READY (*C function*), 132
PyUnicode_Replace (*C function*), 148
PyUnicode_RichCompare (*C function*), 148
PyUnicode_RPartition (*C function*), 147
PyUnicode_RSplit (*C function*), 147
PyUnicode_Split (*C function*), 147
PyUnicode_Splitlines (*C function*), 147
PyUnicode_Substring (*C function*), 139
PyUnicode_Tailmatch (*C function*), 147
PyUnicode_Translate (*C function*), 146
PyUnicode_Type (*C var*), 132
PyUnicode_WRITE (*C function*), 133
PyUnicode_WriteChar (*C function*), 138
PyUnicodeDecodeError_Create (*C function*), 61
PyUnicodeDecodeError_GetEncoding (*C function*), 61
PyUnicodeDecodeError_GetEnd (*C function*), 61
PyUnicodeDecodeError_GetObject (*C function*), 61
PyUnicodeDecodeError_GetReason (*C function*), 62
PyUnicodeDecodeError_GetStart (*C function*), 61
PyUnicodeDecodeError_SetEnd (*C function*), 61
PyUnicodeDecodeError_SetReason (*C function*), 62
PyUnicodeDecodeError_SetStart (*C function*), 61
PyUnicodeEncodeError_GetEncoding (*C function*), 61
PyUnicodeEncodeError_GetEnd (*C function*), 61
PyUnicodeEncodeError_GetObject (*C function*), 61
PyUnicodeEncodeError_GetReason (*C function*), 62
PyUnicodeEncodeError_GetStart (*C function*), 61
PyUnicodeEncodeError_SetEnd (*C function*), 61
PyUnicodeEncodeError_SetReason (*C function*), 62
PyUnicodeEncodeError_SetStart (*C function*), 61
PyUnicodeObject (*C type*), 132
PyUnicodeTranslateError_GetEnd (*C function*), 61
PyUnicodeTranslateError_GetObject (*C function*), 61
PyUnicodeTranslateError_GetReason (*C function*), 62
PyUnicodeTranslateError_GetStart (*C function*), 61
PyUnicodeTranslateError_SetEnd (*C function*), 61
PyUnicodeTranslateError_SetReason (*C function*), 62
PyUnicodeTranslateError_SetStart (*C function*), 61
PyUnstable, 15
PyUnstable_Code_GetExtra (*C function*), 165
PyUnstable_Code_New (*C function*), 163
PyUnstable_Code_NewWithPosOnlyArgs (*C function*), 163
PyUnstable_Code_SetExtra (*C function*), 166

PyUnstable_Eval_RequestCodeExtraIndex (*C function*), 165
PyUnstable_Exc_PrepReraiseStar (*C function*), 60
PyUnstable_GC_VisitObjects (*C function*), 296
PyUnstable_InterpreterFrame_GetCode (*C function*), 182
PyUnstable_InterpreterFrame_GetLasti (*C function*), 183
PyUnstable_InterpreterFrame_GetLine (*C function*), 183
PyUnstable_Long_CompactValue (*C function*), 125
PyUnstable_Long_IsCompact (*C function*), 124
PyUnstable_Object_GC_NewWithExtraData (*C function*), 294
PyUnstable_PerfMapState_Fini (*C function*), 88
PyUnstable_PerfMapState_Init (*C function*), 87
PyUnstable_Type_AssignVersionTag (*C function*), 118
PyUnstable_WritePerfMapEntry (*C function*), 88
PyVarObject (*C type*), 250
PyVarObject_HEAD_INIT (*C macro*), 251
PyVarObject.ob_size (*C member*), 265
PyVectorcall_Call (*C function*), 96
PyVectorcall_Function (*C function*), 96
PyVectorcall_NARGS (*C function*), 96
PyWeakref_Check (*C function*), 178
PyWeakref_CheckProxy (*C function*), 178
PyWeakref_CheckRef (*C function*), 178
PyWeakref_GET_OBJECT (*C function*), 179
PyWeakref_GetObject (*C function*), 179
PyWeakref_NewProxy (*C function*), 179
PyWeakref_NewRef (*C function*), 178
PyWideStringList (*C type*), 218
PyWideStringList_Append (*C function*), 218
PyWideStringList_Insert (*C function*), 218
PyWideStringList.items (*C member*), 218
PyWideStringList.length (*C member*), 218
PyWrapper_New (*C function*), 176

Q

qualified name (정규화된 이름), **313**

R

READ_RESTRICTED (*C macro*), 256
READONLY (*C macro*), 256
realloc (*C function*), 239
reference count (참조 횟수), **313**
regular package (정규 패키지), **314**
releasebufferproc (*C type*), 291
repr
    built-in function, 91, 268
reprfunc (*C type*), 290
RESTRICTED (*C macro*), 256
richcmpfunc (*C type*), 290

S

sdterr
    stdin stdout, 196
search
    path, module, 12, 195, 197, 198
sendfunc (*C type*), 291
sequence
    object, 129
sequence (시퀀스), **314**
set
    object, 157
set comprehension (집합 컴프리헨션), **314**
set_all(), 9
setattrfunc (*C type*), 290
setattrofunc (*C type*), 290
setswitchinterval (*in module sys*), 200
setter (*C type*), 258
SIGINT (*C macro*), 59
signal
    module, 59
single dispatch (싱글 디스패치), **314**
SIZE_MAX (*C macro*), 123
slice (슬라이스), **314**
soft deprecated, **314**
special
    method (메서드), 314
special method (특수 메서드), **314**
ssizeargfunc (*C type*), 291
ssizeobjargproc (*C type*), 291
statement (문장), **314**
static type checker, **314**
staticmethod
    built-in function, 254
stderr (*in module sys*), 208, 209
stdin
    stdout sdterr, 196
stdin (*in module sys*), 208, 209
stdout
    sdterr, stdin, 196
stdout (*in module sys*), 208, 209
strerror (*C function*), 53
string
    PyObject_Str (*C function*), 92
strong reference, **314**
structmember.h, 258
sum_list(), 9
sum_sequence(), 10, 11
sys
    module, 12, 195, 208, 209
SystemError (*built-in exception*), 168

T

T_BOOL (*C macro*), 258
T_BYTE (*C macro*), 258
T_CHAR (*C macro*), 258
T_DOUBLE (*C macro*), 258
T_FLOAT (*C macro*), 258
T_INT (*C macro*), 258
T_LONG (*C macro*), 258
T_LONGLONG (*C macro*), 258

T_NONE (*C macro*), 258
T_OBJECT (*C macro*), 258
T_OBJECT_EX (*C macro*), 258
T_PYSSIZET (*C macro*), 258
T_SHORT (*C macro*), 258
T_STRING (*C macro*), 258
T_STRING_INPLACE (*C macro*), 258
T_UBYTE (*C macro*), 258
T_UINT (*C macro*), 258
T_ULONG (*C macro*), 258
T_ULONGULONG (*C macro*), 258
T_USHORT (*C macro*), 258
ternaryfunc (*C type*), 291
text encoding (텍스트 인코딩), **315**
text file (텍스트 파일), **315**
token, **315**
traverseproc (*C type*), 295
triple-quoted string (삼중 따옴표 된 문자열), **315**
tuple
    built-in function, 103, 153
    object, 149
type
    built-in function, 93
    object, 7, 115
type (형), **315**
type alias (형 에일리어스), **315**
type hint (형 힌트), **315**

U

ULONG_MAX (*C macro*), 123
unaryfunc (*C type*), 291
universal newlines (유니버설 줄 넘김), **316**
USE_STACKCHECK (*C macro*), 66

V

variable annotation (변수 어노테이션), **316**
vectorcallfunc (*C type*), 95
version (*in module sys*), 198, 199
virtual environment (가상 환경), **316**
virtual machine (가상 기계), **316**
visitproc (*C type*), 295

W

WRITE_RESTRICTED (*C macro*), 256

Y

환경 변수
    __PYVENV_LAUNCHER__, 224, 230
    PATH, 12
    PYTHONCOERCECLOCALE, 235
    PYTHONDEBUG, 192, 229
    PYTHONDEVMODE, 225
    PYTHONDONTWRITEBYTECODE, 192, 232
    PYTHONDUMPREFS, 225, 265
    PYTHONEXECUTABLE, 230
    PYTHONFAULTHANDLER, 226
    PYTHONHASHSEED, 193, 227
    PYTHONHOME, 12, 193, 200, 227
    PYTHONINSPECT, 193, 227
    PYTHONINTMAXSTRDIGITS, 227
    PYTHONIOENCODING, 196, 231
    PYTHONLEGACYWINDOWSFSENCODING, 194, 221
    PYTHONLEGACYWINDOWSSTDIO, 194, 228
    PYTHONMALLOC, 240, 243, 245, 247
    PYTHONMALLOCSTATS, 228, 240
    PYTHONNODEBUGRANGES, 225
    PYTHONNOUSERSITE, 194, 232
    PYTHONOPTIMIZE, 194, 229
    PYTHONPATH, 12, 193, 228
    PYTHONPERFSUPPORT, 232
    PYTHONPLATLIBDIR, 228
    PYTHONPROFILEIMPORTTIME, 227
    PYTHONPYCACHEPREFIX, 230
    PYTHONSAFEPATH, 224
    PYTHONTRACEMALLOC, 231
    PYTHONUNBUFFERED, 194, 224
    PYTHONUTF8, 221, 235
    PYTHONVERBOSE, 195, 232
    PYTHONWARNINGS, 232

Z

Zen of Python (파이썬 젠), **316**