Python Setup and Usage

릴리스 *3.14.0a7*

Guido van Rossum and the Python development team

4월 27, 2025

Contents

설명서의 이 부분은 여러 플랫폼에서 파이썬 환경을 설정하고, 인터프리터를 호출하며, 파이썬으로 작업하기 더 쉽게 만드는 것들에 관한 일반적인 정보를 다루는데 할당되었습니다.

CHAPTER 1

명령 줄과 환경

CPython 인터프리터는 명령 줄과 환경에서 다양한 설정을 찾습니다.

다른 구현의 명령 줄 체계는 다를 수 있습니다. 자세한 내용은 implementations 참조하십시오.

1.1 명령 줄

파이썬을 호출할 때 다음 옵션들을 지정할 수 있습니다:

```
python [-bBdEhiIOPqRsSuvVWx?] [-c command | -m module-name | script | - ] [args]
```

물론, 가장 일반적인 사용 사례는 간단한 스크립트 호출입니다:

```
python myscript.py
```

1.1.1 인터페이스 옵션

인터프리터 인터페이스는 유닉스 셸의 인터페이스와 비슷하지만, 몇 가지 추가 호출 방법을 제공합니다:

- When called with standard input connected to a tty device, it prompts for commands and executes them until an EOF (an end-of-file character, you can produce that with `Ctrl-D` on UNIX or `Ctrl-Z, Enter` on Windows) is read. For more on interactive mode, see tut-interac.
- 파일 이름 인자나 파일을 표준 입력으로 사용해서 호출하면, 해당 파일에서 스크립트를 읽고 실행합니다.
- 디렉터리 이름 인자로 호출되면, 해당 디렉터리에서 적절히 이름 붙은 스크립트를 읽고 실행합니다.
- `-c command` 로 호출되면, *command*로 주어지는 파이썬 문장을 실행합니다. 여기서 *command*는 개행 문자로 구분된 여러 개의 문장을 포함할 수 있습니다. 선행 공백은 파이썬 문장에서 중요합니다!
- `-m module-name` 으로 호출되면, 주어진 모듈을 파이썬 모듈 경로에서 찾은 후에 스크립트로 실행합니다.

비대화형 모드에서는, 실행하기 전에 전체 입력을 구문 분석합니다.

인터페이스 옵션은 인터프리터에 의해 소비되는 옵션의 목록을 종료합니다, 뒤따르는 모든 인자는 `sys.argv` 로 들어갑니다 – 첫 번째 요소, 서브 스크립트 0(`sys.argv[0]`)은 프로그램 소스를 반영하는 문자열임에 유의하세요.

-c <command>

command 의 파이썬 코드를 실행합니다. *command* 는 개행 문자로 구분된 하나 이상의 문장일 수 있는데, 일반 모듈 코드에서와같이 선행 공백은 의미가 있습니다.

이 옵션을 주면, `sys.argv` 의 첫 번째 요소는 `"-c"` 가 되고, 현재 디렉터리를 `sys.path` 의 시작 부분에 추가합니다 (그 디렉터리에 있는 모듈을 최상위 모듈로 임포트 할 수 있게 합니다).

`command`를 인자로 감사 이벤트(auditing event) `cpython.run_command`를 발생시킵니다.

버전 3.14.0a7 (unreleased) 에서 변경: *command* is automatically dedented before execution.

-m <module-name>

제공된 이름의 모듈을 `sys.path` 에서 검색하고 그 내용을 `__main__` 모듈로서 실행합니다.

인자가 모듈 이름이기 때문에, 파일 확장자(`.py`)를 주지 않아야 합니다. 모듈 이름은 유효한 절대 파이썬 모듈 이름이어야 하지만, 구현이 항상 이를 강제하는 것은 아닙니다 (예를 들어, 하이픈을 포함하는 이름을 허락할 수도 있습니다).

패키지 이름(이름 공간 패키지 포함)도 허용됩니다. 일반 모듈 대신 패키지 이름이 제공되면, 인터프리터는 `<pkg>.__main__` 을 메인 모듈로 실행합니다. 이 동작은 인터프리터에 스크립트 인자로 전달되는 디렉터리 및 zip 파일의 처리와 의도적으로 유사합니다.

참고

이 옵션은 내장 모듈이나 확장 모듈에는 사용될 수 없는데, 이것들은 파이썬 모듈 파일을 갖고 있지 않기 때문입니다. 그러나, 원래 소스 파일이 없는 사전 컴파일된 모듈에는 여전히 사용할 수 있습니다.

이 옵션을 주면, `sys.argv` 의 첫 번째 요소는 모듈 파일의 전체 경로가 됩니다 (모듈 파일을 찾는 동안에는 첫 번째 요소를 `"-m"` 으로 설정합니다). *-c* 옵션과 마찬가지로, 현재 디렉터리가 `sys.path` 의 시작 부분에 추가됩니다.

-I option can be used to run the script in isolated mode where `sys.path` contains neither the current directory nor the user' s site-packages directory. All `PYTHON*` environment variables are ignored, too.

많은 표준 라이브러리 모듈에는 스크립트로 실행할 때 호출되는 코드가 들어 있습니다. 한 예는 `timeit` 모듈입니다:

```
python -m timeit -s "setup here" "benchmarked code here"
python -m timeit -h # for details
```

`module-name`을 인자로 감사 이벤트(auditing event) `cpython.run_module`을 발생시킵니다.

더 보기

runpy.run_module()
파이썬 코드에서 직접 사용할 수 있는 동등한 기능

PEP 338 – 모듈을 스크립트로 실행하기

버전 3.1에서 변경: `__main__` 서브 모듈을 실행할 패키지 이름을 제공할 수 있습니다.

버전 3.4에서 변경: 이름 공간 패키지도 지원됩니다.

-

표준 입력(`sys.stdin`)에서 명령을 읽습니다. 표준 입력이 터미널이면, *-i* 가 묵시적으로 적용됩니다.

이 옵션을 주면, `sys.argv` 의 첫 번째 요소는 `"-"` 이 되고, 현재 디렉터리가 `sys.path` 의 처음에 추가됩니다.

인자 없이 감사 이벤트(auditing event) `cpython.run_stdin`을 발생시킵니다.

<script>

script 에 담긴 파이썬 코드를 실행합니다. *script* 는 파이썬 파일이나 `__main__.py` 파일이 들어있는 디렉터리나 `__main__.py` 파일을 포함하는 zip 파일을 가리키는 파일 시스템 경로(절대나 상대)여야 합니다.

이 옵션을 주면, `sys.argv` 의 첫 번째 요소는 명령 줄에서 주어진 스크립트 이름이 됩니다.

스크립트 이름이 파이썬 파일을 직접 가리키면, 해당 파일을 포함하는 디렉터리가 `sys.path` 의 시작 부분에 추가되고, 파일은 `__main__` 모듈로 실행됩니다.

스크립트 이름이 디렉터리 나 zip 파일을 가리키면, 스크립트 이름이 `sys.path` 의 시작 부분에 추가되고, 해당 위치의 `__main__.py` 파일을 `__main__` 모듈로 실행합니다.

-I option can be used to run the script in isolated mode where `sys.path` contains neither the script's directory nor the user's site-packages directory. All `PYTHON*` environment variables are ignored, too.

`filename`을 인자로 감사 이벤트(auditing event) `cpython.run_file`을 발생시킵니다.

더 보기

runpy.run_path()
파이썬 코드에서 직접 사용할 수 있는 동등한 기능

인터페이스 옵션을 주지 않으면, *-i* 가 묵시적으로 적용되고, `sys.argv[0]` 는 빈 문자열(`""`)이 되고, 현재 디렉터리가 `sys.path` 의 처음에 추가됩니다. 또한, 플랫폼에서 사용 가능한 경우 (rlcompleter-config 를 참조하세요), 탭 완성 및 히스토리 편집이 자동으로 활성화됩니다.

더 보기

tut-invoking

버전 3.4에서 변경: 탭 완성과 히스토리 편집의 자동 활성화.

1.1.2 일반 옵션

-?

-h

--help

Print a short description of all command line options and corresponding environment variables and exit.

--help-env

Print a short description of Python-specific environment variables and exit.

Added in version 3.11.

--help-xoptions

Print a description of implementation-specific *-X* options and exit.

Added in version 3.11.

--help-all

Print complete usage information and exit.

Added in version 3.11.

-V

--version

파이썬 버전 번호를 출력하고 종료합니다. 출력 예는 다음과 같습니다:

```
Python 3.8.0b2+
```

두 번 지정하면, 다음과 같이 빌드에 관한 추가 정보를 인쇄합니다:

```
Python 3.8.0b2+ (3.8:0c076caaa8, Apr 20 2019, 21:55:00)
[GCC 6.2.0 20161005]
```

Added in version 3.6: `-VV` 옵션.

1.1.3 기타 옵션

-b

Issue a warning when converting `bytes` or `bytearray` to `str` without specifying encoding or comparing `bytes` or `bytearray` with `str` or `bytes` with `int`. Issue an error when the option is given twice (`-bb`).

버전 3.5에서 변경: Affects also comparisons of `bytes` with `int`.

-B

주어지면, 파이썬은 소스 모듈을 임포트 할 때 `.pyc` 파일을 쓰려고 하지 않습니다. *PYTHONDONTWRITEBYTECODE* 도 참조하십시오.

--check-hash-based-pycs `default|always|never`

해시 기반 `.pyc` 파일의 검증 동작을 제어합니다. pyc-invalidation를 참조하세요. `default` 로 설정하면, 검사형과 비검사형 해시 기반 바이트 코드 캐시 파일은 기본 의미에 따라 유효성이 검사됩니다. `always` 로 설정하면, 모든 해시 기반 `.pyc` 파일들은, 검사형과 비검사형을 가리지 않고, 해당 소스 파일에 대해 유효성이 검사됩니다. `never` 로 설정되면, 해시 기반 `.pyc` 파일은 해당 소스 파일에 대해 유효성이 검사되지 않습니다.

타임스탬프 기반 `.pyc` 파일의 의미는 이 옵션의 영향을 받지 않습니다.

-d

Turn on parser debugging output (for expert only). See also the *PYTHONDEBUG* environment variable.

This option requires a *debug build of Python*, otherwise it' s ignored.

-E

Ignore all `PYTHON*` environment variables, e.g. *PYTHONPATH* and *PYTHONHOME*, that might be set.

See also the *-P* and *-I* (isolated) options.

-i

Enter interactive mode after execution.

Using the *-i* option will enter interactive mode in any of the following circumstances:

- When a script is passed as first argument
- When the *-c* option is used
- When the *-m* option is used

Interactive mode will start even when `sys.stdin` does not appear to be a terminal. The *PYTHONSTARTUP* file is not read.

이것은 스크립트가 예외를 발생시킬 때 전역 변수나 스택 트레이스를 검사하는 데 유용할 수 있습니다. *PYTHONINSPECT* 도 참조하십시오.

-I

Run Python in isolated mode. This also implies *-E*, *-P* and *-s* options.

In isolated mode `sys.path` contains neither the script' s directory nor the user' s site-packages directory. All `PYTHON*` environment variables are ignored, too. Further restrictions may be imposed to prevent the user from injecting malicious code.

Added in version 3.4.

-O

assert 문과 `__debug__` 의 값에 대한 조건부 코드를 제거합니다. `.pyc` 확장자 앞에 `.opt-1` 을 추가하여 컴파일된 (바이트 코드) 파일의 이름을 구분합니다 (**PEP 488**을 참조하세요). *PYTHONOPTIMIZE* 도 참조하십시오.

버전 3.5에서 변경: **PEP 488** 에 따라 `.pyc` 파일명을 수정합니다.

-OO

*-O*를 적용하고 독스트링도 버립니다. `.pyc` 확장자 앞에 `.opt-2` 를 추가하여 컴파일 된(바이트 코드) 파일의 이름을 구분합니다 (참조 **PEP 488**을 참조하세요).

버전 3.5에서 변경: **PEP 488** 에 따라 `.pyc` 파일명을 수정합니다.

-P

Don' t prepend a potentially unsafe path to `sys.path`:

- `python -m module` command line: Don' t prepend the current working directory.
- `python script.py` command line: Don' t prepend the script' s directory. If it' s a symbolic link, resolve symbolic links.
- `python -c code` and `python` (REPL) command lines: Don' t prepend an empty string, which means the current working directory.

See also the *PYTHONSAFEPATH* environment variable, and *-E* and *-I* (isolated) options.

Added in version 3.11.

-q

대화형 모드에서도 저작권과 버전 메시지를 표시하지 않습니다.

Added in version 3.2.

-R

해시 무작위화를 켭니다. 이 옵션은 *PYTHONHASHSEED* 환경 변수가 `0` 으로 설정된 경우에만 효과가 있습니다, 해시 무작위화는 기본적으로 활성화되기 때문입니다.

On previous versions of Python, this option turns on hash randomization, so that the `__hash__()` values of str and bytes objects are "salted" with an unpredictable random value. Although they remain constant within an individual Python process, they are not predictable between repeated invocations of Python.

Hash randomization is intended to provide protection against a denial-of-service caused by carefully chosen inputs that exploit the worst case performance of a dict construction, $O(n^2)$ complexity. See http://ocert.org/advisories/ocert-2011-003.html for details.

PYTHONHASHSEED 는 해시 시드 시크릿에 고정값을 설정할 수 있게 합니다.

Added in version 3.2.3.

버전 3.7에서 변경: 이 옵션은 더는 무시되지 않습니다.

-s

사용자 `site-packages` 디렉터리를 `sys.path` 에 추가하지 않습니다.

See also *PYTHONNOUSERSITE*.

> **더 보기**
>
> **PEP 370** – 사용자별 site-packages 디렉터리

-S

site 모듈의 임포트와 이 모듈이 수반하는 sys.path 의 사이트 의존적 조작을 비활성화합니다. 또한 site 가 나중에 명시적으로 임포트될 때도 이 조작을 비활성화합니다 (조작하기를 원하면 site.main() 을 호출하십시오).

-u

stdout 과 stderr 스트림을 버퍼링하지 않도록 만듭니다. 이 옵션은 stdin 스트림에는 영향을 미치지 않습니다.

PYTHONUNBUFFERED 도 참조하세요.

버전 3.7에서 변경: stdout 과 stderr 스트림의 텍스트 계층은 이제 버퍼링 되지 않습니다.

-v

Print a message each time a module is initialized, showing the place (filename or built-in module) from which it is loaded. When given twice (-vv), print a message for each file that is checked for when searching for a module. Also provides information on module cleanup at exit.

버전 3.10에서 변경: The site module reports the site-specific paths and .pth files being processed.

See also *PYTHONVERBOSE*.

-W arg

Warning control. Python' s warning machinery by default prints warning messages to sys.stderr.

가장 단순한 설정은 프로세스가 만드는 모든 경고에 무조건 특정 액션을 적용합니다 (그렇지 않으면 기본적으로 무시되는 경고조차도):

```
-Wdefault  # Warn once per call location
-Werror    # Convert to exceptions
-Walways   # Warn every time
-Wall      # Same as -Walways
-Wmodule   # Warn once per calling module
-Wonce     # Warn once per Python process
-Wignore   # Never warn
```

The action names can be abbreviated as desired and the interpreter will resolve them to the appropriate action name. For example, -Wi is the same as -Wignore.

The full form of argument is:

```
action:message:category:module:lineno
```

Empty fields match all values; trailing empty fields may be omitted. For example -W ignore::DeprecationWarning ignores all DeprecationWarning warnings.

The *action* field is as explained above but only applies to warnings that match the remaining fields.

The *message* field must match the whole warning message; this match is case-insensitive.

The *category* field matches the warning category (ex: DeprecationWarning). This must be a class name; the match test whether the actual warning category of the message is a subclass of the specified warning category.

The *module* field matches the (fully qualified) module name; this match is case-sensitive.

The *lineno* field matches the line number, where zero matches all line numbers and is thus equivalent to an omitted line number.

Multiple *-W* options can be given; when a warning matches more than one option, the action for the last matching option is performed. Invalid *-W* options are ignored (though, a warning message is printed about invalid options when the first warning is issued).

Warnings can also be controlled using the *PYTHONWARNINGS* environment variable and from within a Python program using the warnings module. For example, the warnings.filterwarnings() function can be used to use a regular expression on the warning message.

자세한 내용은 warning-filter와 describing-warning-filters를 참조하십시오.

-x

소스의 첫 번째 줄을 건너 뛰어서, 유닉스 이외의 형식의 `#!cmd` 을 사용할 수 있게 합니다. 이것은 DOS 전용 핵(hack)을 위한 것입니다.

-X

다양한 구현 특정 옵션을 위해 예약되어 있습니다. CPython은 현재 다음과 같은 가능한 값을 정의합니다:

- `-X faulthandler` to enable `faulthandler`. See also *PYTHONFAULTHANDLER*.

  Added in version 3.3.

- `-X showrefcount` to output the total reference count and number of used memory blocks when the program finishes or after each statement in the interactive interpreter. This only works on *debug builds*.

  Added in version 3.4.

- `-X tracemalloc` to start tracing Python memory allocations using the `tracemalloc` module. By default, only the most recent frame is stored in a traceback of a trace. Use `-X tracemalloc=NFRAME` to start tracing with a traceback limit of *NFRAME* frames. See `tracemalloc.start()` and *PYTHONTRACEMALLOC* for more information.

  Added in version 3.4.

- `-X int_max_str_digits` configures the integer string conversion length limitation. See also *PYTHONINTMAXSTRDIGITS*.

  Added in version 3.11.

- `-X importtime` 은 각 임포트가 얼마나 오래 걸렸는지 보여줍니다. 모듈 이름, 누적 시간(중첩된 임포트 포함), 자체 시간(중첩 임포트 제외)을 표시합니다. 다중 스레드 응용 프로그램에서 출력이 깨질 수 있음에 유의하십시오. 일반적인 사용법은 `python3 -X importtime -c 'import asyncio'` 입니다. *PYTHONPROFILEIMPORTTIME* 도 참조하십시오.

  Added in version 3.7.

- `-X dev`: enable Python Development Mode, introducing additional runtime checks that are too expensive to be enabled by default. See also *PYTHONDEVMODE*.

  Added in version 3.7.

- `-X utf8` enables the Python UTF-8 Mode. `-X utf8=0` explicitly disables Python UTF-8 Mode (even when it would otherwise activate automatically). See also *PYTHONUTF8*.

  Added in version 3.7.

- `-X pycache_prefix=PATH`는 `.pyc` 파일을 코드 트리 대신에 지정된 디렉터리를 루트로 하는 병렬 트리에 쓰도록 합니다. *PYTHONPYCACHEPREFIX*도 참조하십시오.

  Added in version 3.8.

- `-X warn_default_encoding` issues a `EncodingWarning` when the locale-specific default encoding is used for opening files. See also *PYTHONWARNDEFAULTENCODING*.

  Added in version 3.10.

- `-X no_debug_ranges` disables the inclusion of the tables mapping extra location information (end line, start column offset and end column offset) to every instruction in code objects. This is useful when smaller code objects and pyc files are desired as well as suppressing the extra visual location indicators when the interpreter displays tracebacks. See also *PYTHONNODEBUGRANGES*.

  Added in version 3.11.

- `-X frozen_modules` determines whether or not frozen modules are ignored by the import machinery. A value of `on` means they get imported and `off` means they are ignored. The default is `on` if this is an installed Python (the normal case). If it' s under development (running from the source tree) then

the default is `off`. Note that the `importlib_bootstrap` and `importlib_bootstrap_external` frozen modules are always used, even if this flag is set to `off`. See also *PYTHON_FROZEN_MODULES*.

Added in version 3.11.

- `-X perf` enables support for the Linux `perf` profiler. When this option is provided, the `perf` profiler will be able to report Python calls. This option is only available on some platforms and will do nothing if is not supported on the current system. The default value is "off". See also *PYTHONPERFSUPPORT* and perf_profiling.

  Added in version 3.12.

- `-X perf_jit` enables support for the Linux `perf` profiler with DWARF support. When this option is provided, the `perf` profiler will be able to report Python calls using DWARF information. This option is only available on some platforms and will do nothing if is not supported on the current system. The default value is "off". See also *PYTHON_PERF_JIT_SUPPORT* and perf_profiling.

  Added in version 3.13.

- `-X disable_remote_debug` disables the remote debugging support as described in **PEP 768**. This includes both the functionality to schedule code for execution in another process and the functionality to receive code for execution in the current process.

  This option is only available on some platforms and will do nothing if is not supported on the current system. See also *PYTHON_DISABLE_REMOTE_DEBUG* and **PEP 768**.

  Added in version 3.14.

- `-X cpu_count=`*n* overrides `os.cpu_count()`, `os.process_cpu_count()`, and `multiprocessing.cpu_count()`. *n* must be greater than or equal to 1. This option may be useful for users who need to limit CPU resources of a container system. See also *PYTHON_CPU_COUNT*. If *n* is `default`, nothing is overridden.

  Added in version 3.13.

- `-X presite=`*package.module* specifies a module that should be imported before the `site` module is executed and before the `__main__` module exists. Therefore, the imported module isn' t `__main__`. This can be used to execute code early during Python initialization. Python needs to be *built in debug mode* for this option to exist. See also *PYTHON_PRESITE*.

  Added in version 3.13.

- `-X gil=`*0*, *1* forces the GIL to be disabled or enabled, respectively. Setting to `0` is only available in builds configured with *--disable-gil*. See also *PYTHON_GIL* and whatsnew313-free-threaded-cpython.

  Added in version 3.13.

- `-X thread_inherit_context=`*0*, *1* causes `Thread` to, by default, use a copy of context of of the caller of `Thread.start()` when starting. Otherwise, threads will start with an empty context. If unset, the value of this option defaults to `1` on free-threaded builds and to `0` otherwise. See also *PYTHON_THREAD_INHERIT_CONTEXT*.

  Added in version 3.14.

- `-X context_aware_warnings=`*0*, *1* causes the `warnings.catch_warnings` context manager to use a `ContextVar` to store warnings filter state. If unset, the value of this option defaults to `1` on free-threaded builds and to `0` otherwise. See also *PYTHON_CONTEXT_AWARE_WARNINGS*.

  Added in version 3.14.

또한 `sys._xoptions` 딕셔너리를 통해 임의의 값을 전달하고 조회할 수 있도록 합니다.

Added in version 3.2.

버전 3.9에서 변경: Removed the `-X showalloccount` option.

버전 3.10에서 변경: Removed the `-X oldparser` option.

1.1.4 Controlling color

The Python interpreter is configured by default to use colors to highlight output in certain situations such as when displaying tracebacks. This behavior can be controlled by setting different environment variables.

Setting the environment variable `TERM` to `dumb` will disable color.

If the `FORCE_COLOR` environment variable is set, then color will be enabled regardless of the value of TERM. This is useful on CI systems which aren' t terminals but can still display ANSI escape sequences.

If the `NO_COLOR` environment variable is set, Python will disable all color in the output. This takes precedence over `FORCE_COLOR`.

All these environment variables are used also by other tools to control color output. To control the color output only in the Python interpreter, the *PYTHON_COLORS* environment variable can be used. This variable takes precedence over `NO_COLOR`, which in turn takes precedence over `FORCE_COLOR`.

1.1.5 사용해서는 안 되는 옵션

-J

Jython 이 사용하기 위해 예약되었습니다.

1.2 환경 변수

이 환경 변수들은 파이썬의 동작에 영향을 주며, -E와 -I 이외의 명령 줄 스위치보다 먼저 처리됩니다. 충돌하면 명령 줄 스위치가 환경 변수에 우선하는 것이 관례입니다.

PYTHONHOME

표준 파이썬 라이브러리의 위치를 변경합니다. 기본적으로, 라이브러리는 *prefix*`/lib/pythonversion`과 *exec_prefix*`/lib/pythonversion`에서 검색되는데, *prefix* 와 *exec_prefix* 는 설치 의존적인 디렉터리이고, 둘 다 기본값은 `/usr/local` 입니다.

PYTHONHOME 이 하나의 디렉터리로 설정되면, 그 값은 *prefix* 와 *exec_prefix* 를 모두 대체합니다. 이들에 대해 다른 값을 지정하려면, *PYTHONHOME* 을 *prefix*`:`*exec_prefix* 로 설정하십시오.

PYTHONPATH

모듈 파일의 기본 검색 경로를 보강합니다. 형식은 셸의 `PATH` 와 같습니다: 하나 이상의 디렉터리 경로명이 `os.pathsep` (예를 들어, 유닉스에서는 콜론, 윈도우에서는 세미콜론) 로 구분됩니다. 존재하지 않는 디렉터리는 조용히 무시됩니다.

일반 디렉터리 외에도, 개별 *PYTHONPATH* 엔트리는 순수 파이썬 모듈(소스 또는 컴파일된 형식)을 포함하는 zip 파일을 가리킬 수 있습니다. 확장 모듈은 zip 파일에서 임포트될 수 없습니다.

기본 검색 경로는 설치 의존적이지만, 일반적으로 *prefix*`/lib/pythonversion`으로 시작합니다 (위의 *PYTHONHOME* 을 참조하세요). 항상 *PYTHONPATH* 에 추가됩니다.

위에서 설명한 대로 인터페이스 옵션 하에서는 *PYTHONPATH* 앞에 검색 경로에 추가 디렉터리가 삽입됩니다. 검색 경로는 파이썬 프로그램 내에서 `sys.path` 변수로 조작할 수 있습니다.

PYTHONSAFEPATH

If this is set to a non-empty string, don' t prepend a potentially unsafe path to `sys.path`: see the *-P* option for details.

Added in version 3.11.

PYTHONPLATLIBDIR

이것을 비어 있지 않은 문자열로 설정하면, `sys.platlibdir` 값을 재정의합니다.

Added in version 3.9.

PYTHONSTARTUP

이것이 읽을 수 있는 파일의 이름이면, 첫 번째 프롬프트가 대화형 모드에 표시되기 전에, 해당 파일의 파이썬 명령이 실행됩니다. 이 파일은 대화형 명령이 실행되는 것과 같은 이름 공간에서 실행되므로,

여기에서 정의되거나 임포트 한 객체를 대화형 세션에서 그대로 사용할 수 있습니다. 이 파일에서 프롬프트 sys.ps1 과 sys.ps2 와 훅 sys.__interactivehook__ 도 바꿀 수 있습니다.

시작 시 호출될 때 filename을 인자로 감사 이벤트(auditing event) cpython.run_startup을 발생시킵니다.

PYTHONOPTIMIZE

비어 있지 않은 문자열로 설정하면 -O 옵션을 지정하는 것과 같습니다. 정수로 설정하면, -O를 여러 번 지정하는 것과 같습니다.

PYTHONBREAKPOINT

설정되면, 점으로 구분된 경로 표기법을 사용하여 콜러블의 이름을 지정합니다. 콜러블을 포함하는 모듈이 임포트 된 후에 콜러블은, 내장 breakpoint() 에 의해 호출되는 sys.breakpointhook() 의 기본 구현이 실행합니다. 설정되지 않았거나 빈 문자열로 설정하면, 값 "pdb.set_trace"와 동등합니다. 문자열 "0"으로 설정하면, sys.breakpointhook() 의 기본 구현은 아무것도 하지 않고 즉시 반환합니다.

Added in version 3.7.

PYTHONDEBUG

비어 있지 않은 문자열로 설정하면, -d 옵션을 지정하는 것과 같습니다. 정수로 설정하면, -d를 여러 번 지정하는 것과 같습니다.

This environment variable requires a *debug build of Python*, otherwise it' s ignored.

PYTHONINSPECT

비어 있지 않은 문자열로 설정하면, -i 옵션을 지정하는 것과 같습니다.

이 변수는 프로그램 종료 시 검사 모드를 강제하기 위해, os.environ 을 사용해서 파이썬 코드에 의해 수정될 수도 있습니다.

인자 없이 감사 이벤트(auditing event) cpython.run_stdin을 발생시킵니다.

버전 3.12.5에서 변경: (also 3.11.10, 3.10.15, 3.9.20, and 3.8.20) Emits audit events.

버전 3.13에서 변경: Uses PyREPL if possible, in which case *PYTHONSTARTUP* is also executed. Emits audit events.

PYTHONUNBUFFERED

비어 있지 않은 문자열로 설정하면, -u 옵션을 지정하는 것과 같습니다.

PYTHONVERBOSE

비어 있지 않은 문자열로 설정하면, -v 옵션을 지정하는 것과 같습니다. 정수로 설정하면 -v를 여러 번 지정하는 것과 같습니다.

PYTHONCASEOK

If this is set, Python ignores case in import statements. This only works on Windows and macOS.

PYTHONDONTWRITEBYTECODE

비어 있지 않은 문자열로 설정되면, 파이썬은 소스 모듈을 임포트 할 때 .pyc 파일을 쓰지 않습니다. 이는 -B 옵션을 지정하는 것과 같습니다.

PYTHONPYCACHEPREFIX

설정되면, 파이썬은 소스 트리 내의 __pycache__ 디렉터리 대신에 이 경로에 있는 미러 디렉터리 트리에 .pyc 파일을 씁니다. 이것은 -X pycache_prefix=PATH 옵션을 지정하는 것과 동등합니다.

Added in version 3.8.

PYTHONHASHSEED

이 변수가 설정되어 있지 않거나 random 으로 설정되면, str과 bytes 객체의 해시 시드에 난수가 사용됩니다.

PYTHONHASHSEED 가 정숫값으로 설정되면, 해시 무작위화가 적용되는 형의 hash()를 생성하기 위한 고정 시드로 사용됩니다.

목적은 인터프리터 자체에 대한 셀프 테스트와 같은 이유로 반복 가능한 해싱을 허용하거나, 파이썬 프로세스 클러스터가 해시값을 공유하도록 허용하는 것입니다.

정수는 [0,4294967295] 범위의 십진수여야 합니다. 값 0을 지정하면 해시 무작위화가 비활성화됩니다.

Added in version 3.2.3.

PYTHONINTMAXSTRDIGITS

If this variable is set to an integer, it is used to configure the interpreter' s global integer string conversion length limitation.

Added in version 3.11.

PYTHONIOENCODING

인터프리터를 실행하기 전에 이것이 설정되면, stdin/stdout/stderr에 사용되는 인코딩을 대체합니다. 문법은 `encodingname:errorhandler` 형식입니다. `encodingname` 과 `:errorhandler` 부분은 모두 선택 사항이며 `str.encode()` 에서와 같은 의미입니다.

stderr의 경우, `:errorhandler` 부분은 무시됩니다; 처리기는 항상 `'backslashreplace'` 입니다.

버전 3.4에서 변경: `encodingname` 부분은 이제 선택적입니다.

버전 3.6에서 변경: Windows에서, *PYTHONLEGACYWINDOWSSTDIO* 도 지정하지 않는 한, 대화형 콘솔 버퍼에서 이 변수로 지정된 인코딩이 무시됩니다. 표준 스트림을 통해 리디렉션 된 파일과 파이프는 영향을 받지 않습니다.

PYTHONNOUSERSITE

설정되면, 파이썬은 사용자 `site-packages` 디렉터리를 `sys.path` 에 추가하지 않습니다.

> **더 보기**
>
> **PEP 370** – 사용자별 site-packages 디렉터리

PYTHONUSERBASE

Defines the `user base directory`, which is used to compute the path of the `user site-packages directory` and installation paths for `python -m pip install --user`.

> **더 보기**
>
> **PEP 370** – 사용자별 site-packages 디렉터리

PYTHONEXECUTABLE

If this environment variable is set, `sys.argv[0]` will be set to its value instead of the value got through the C runtime. Only works on macOS.

PYTHONWARNINGS

-W 옵션과 동등합니다. 쉼표로 구분된 문자열로 설정하면, *-W*를 여러 번 지정하는 것과 같습니다. 목록의 뒷부분에 있는 필터는 목록의 이전 필터보다 우선합니다.

가장 단순한 설정은 프로세스가 만드는 모든 경고에 무조건 특정 액션을 적용합니다 (그렇지 않으면 기본적으로 무시되는 경고조차도):

```
PYTHONWARNINGS=default  # Warn once per call location
PYTHONWARNINGS=error    # Convert to exceptions
PYTHONWARNINGS=always   # Warn every time
PYTHONWARNINGS=all      # Same as PYTHONWARNINGS=always
PYTHONWARNINGS=module   # Warn once per calling module
PYTHONWARNINGS=once     # Warn once per Python process
PYTHONWARNINGS=ignore   # Never warn
```

자세한 내용은 warning-filter와 describing-warning-filters를 참조하십시오.

PYTHONFAULTHANDLER

If this environment variable is set to a non-empty string, `faulthandler.enable()` is called at startup: install a handler for `SIGSEGV`, `SIGFPE`, `SIGABRT`, `SIGBUS` and `SIGILL` signals to dump the Python traceback. This is equivalent to *-X* `faulthandler` option.

Added in version 3.3.

PYTHONTRACEMALLOC

If this environment variable is set to a non-empty string, start tracing Python memory allocations using the `tracemalloc` module. The value of the variable is the maximum number of frames stored in a traceback of a trace. For example, `PYTHONTRACEMALLOC=1` stores only the most recent frame. See the `tracemalloc.start()` function for more information. This is equivalent to setting the *-X* `tracemalloc` option.

Added in version 3.4.

PYTHONPROFILEIMPORTTIME

If this environment variable is set to a non-empty string, Python will show how long each import takes. This is equivalent to setting the *-X* `importtime` option.

Added in version 3.7.

PYTHONASYNCIODEBUG

이 환경 변수가 비어 있지 않은 문자열로 설정되면, `asyncio` 모듈의 디버그 모드를 활성화합니다.

Added in version 3.4.

PYTHONMALLOC

파이썬 메모리 할당자를 설정하거나 디버그 훅을 설치합니다.

파이썬이 사용하는 메모리 할당자를 설정합니다:

- `default`: 기본 메모리 할당자를 사용합니다.
- `malloc`: use the `malloc()` function of the C library for all domains (`PYMEM_DOMAIN_RAW`, `PYMEM_DOMAIN_MEM`, `PYMEM_DOMAIN_OBJ`).
- `pymalloc`: use the pymalloc allocator for `PYMEM_DOMAIN_MEM` and `PYMEM_DOMAIN_OBJ` domains and use the `malloc()` function for the `PYMEM_DOMAIN_RAW` domain.
- `mimalloc`: use the mimalloc allocator for `PYMEM_DOMAIN_MEM` and `PYMEM_DOMAIN_OBJ` domains and use the `malloc()` function for the `PYMEM_DOMAIN_RAW` domain.

Install debug hooks:

- `debug`: 기본 메모리 할당자 위에 디버그 훅을 설치합니다.
- `malloc_debug`: `malloc` 과 같지만, 디버그 훅도 설치합니다.
- `pymalloc_debug`: `pymalloc` 과 같지만, 디버그 훅도 설치합니다.
- `mimalloc_debug`: same as `mimalloc` but also install debug hooks.

Added in version 3.6.

버전 3.7에서 변경: `"default"` 할당자를 추가했습니다.

PYTHONMALLOCSTATS

비어 있지 않은 문자열로 설정되면, 파이썬은 새로운 pymalloc 객체 영역이 생성될 때마다, 그리고 종료할 때 pymalloc 메모리 할당자의 통계를 인쇄합니다.

PYTHONMALLOC 환경 변수를 사용하여 C 라이브러리의 `malloc()` 할당자를 강제로 사용하거나, `pymalloc` 지원 없이 파이썬을 구성하면, 이 변수는 무시됩니다.

버전 3.6에서 변경: 이 변수는 이제 배포 모드로 컴파일된 파이썬에서도 사용할 수 있습니다. 이제 빈 문자열로 설정하면 효과가 없습니다.

PYTHONLEGACYWINDOWSFSENCODING

If set to a non-empty string, the default *filesystem encoding and error handler* mode will revert to their pre-3.6 values of 'mbcs' and 'replace' , respectively. Otherwise, the new defaults 'utf-8' and 'surrogatepass' are used.

This may also be enabled at runtime with `sys._enablelegacywindowsfsencoding()`.

가용성: Windows.

Added in version 3.6: 자세한 내용은 **PEP 529**를 참조하십시오.

PYTHONLEGACYWINDOWSSTDIO

비어 있지 않은 문자열로 설정하면, 새 콘솔 입력기와 출력기를 사용하지 않습니다. 이것은 유니코드 문자가 utf-8을 사용하는 대신 활성 콘솔 코드 페이지에 따라 인코딩됨을 의미합니다.

이 변수는 표준 스트림이 콘솔 버퍼를 참조하는 대신 리디렉트 된 (파일 또는 파이프로) 경우 무시됩니다.

가용성: Windows.

Added in version 3.6.

PYTHONCOERCECLOCALE

값 `0` 으로 설정하면, 주 파이썬 명령 줄 응용 프로그램이 레거시 ASCII 기반 C와 POSIX 로케일을 보다 유능한 UTF-8 기반 대안으로 강제 변환하지 않습니다.

이 변수가 설정되지 않고 (또는 `0` 이외의 값으로 설정되고), 환경 변수에 우선하는 `LC_ALL` 로케일도 설정되지 않고, `LC_CTYPE` 범주에 대해 보고되는 현재 로케일이 기본 `C` 로케일이거나 명시적인 ASCII 기반의 `POSIX` 로케일이면, 파이썬 CLI는 인터프리터 런타임을 로드하기 전에 `LC_CTYPE` 범주에 대해 다음 로케일을 나열된 순서대로 구성하려고 시도합니다:

- `C.UTF-8`
- `C.utf8`
- `UTF-8`

이러한 로케일 범주 중 하나를 설정하는 데 성공하면, 파이썬 런타임이 초기화되기 전에 `LC_CTYPE` 환경 변수도 현재 프로세스 환경에서 적절히 설정됩니다. 이렇게 하면 인터프리터 자신과 같은 프로세스에서 실행되는 다른 로케일 인식 구성 요소(가령 GNU `readline` 라이브러리)가 볼 수 있는 것에 더해, 갱신된 설정을 현재 C 로케일이 아닌 환경을 조회하는 연산(가령 파이썬 자체의 `locale.getdefaultlocale()`)뿐만 아니라, 자식 프로세스에서도 (이 프로세스가 파이썬 인터프리터를 실행하는지에 관계없이) 볼 수 있습니다.

이러한 로케일 중 하나를 구성하면 (명시적으로나 위의 묵시적 로케일 강제 변경을 통해) `sys.stdin` 과 `sys.stdout` 에 대해 `surrogateescape` 에러 처리기를 자동으로 활성화합니다 (`sys.stderr` 는 다른 로케일에서처럼 `backslashreplace` 를 계속 사용합니다). 이 스트림 처리 동작은 평소처럼 *PYTHONIOENCODING*을 사용하여 대체할 수 있습니다.

디버깅을 위해, `PYTHONCOERCECLOCALE=warn` 을 설정하면, 로케일 강제 변경이 일어나거나, 그렇지 않고 강제 변경을 유발할 로케일이 파이썬 런타임이 초기화될 때 여전히 활성 상태면 파이썬은 `stderr` 로 경고 메시지를 보냅니다.

또한, 로케일 강제 변환이 비활성화되거나 적절한 대상 로케일을 찾지 못할 때도, 레거시 ASCII 기반 로케일에서 *PYTHONUTF8* 은 기본적으로 활성화됨에 유의하십시오. 인터프리터가 시스템 인터페이스에 대해 `UTF-8` 대신에 `ASCII` 를 사용하게 하려면, 두 가지 기능을 모두 비활성화시켜야 합니다.

가용성: Unix.

Added in version 3.7: 자세한 내용은 **PEP 538**을 참조하십시오.

PYTHONDEVMODE

If this environment variable is set to a non-empty string, enable Python Development Mode, introducing additional runtime checks that are too expensive to be enabled by default. This is equivalent to setting the *-X* `dev` option.

Added in version 3.7.

PYTHONUTF8

If set to `1`, enable the Python UTF-8 Mode.

If set to `0`, disable the Python UTF-8 Mode.

다른 모든 비어 있지 않은 문자열로 설정하면, 인터프리터를 초기화하는 동안 에러가 발생합니다.

Added in version 3.7.

PYTHONWARNDEFAULTENCODING

If this environment variable is set to a non-empty string, issue a `EncodingWarning` when the locale-specific default encoding is used.

See io-encoding-warning for details.

Added in version 3.10.

PYTHONNODEBUGRANGES

If this variable is set, it disables the inclusion of the tables mapping extra location information (end line, start column offset and end column offset) to every instruction in code objects. This is useful when smaller code objects and pyc files are desired as well as suppressing the extra visual location indicators when the interpreter displays tracebacks.

Added in version 3.11.

PYTHONPERFSUPPORT

If this variable is set to a nonzero value, it enables support for the Linux `perf` profiler so Python calls can be detected by it.

If set to `0`, disable Linux `perf` profiler support.

See also the *-X perf* command-line option and perf_profiling.

Added in version 3.12.

PYTHON_PERF_JIT_SUPPORT

If this variable is set to a nonzero value, it enables support for the Linux `perf` profiler so Python calls can be detected by it using DWARF information.

If set to `0`, disable Linux `perf` profiler support.

See also the *-X perf_jit* command-line option and perf_profiling.

Added in version 3.13.

PYTHON_DISABLE_REMOTE_DEBUG

If this variable is set to a non-empty string, it disables the remote debugging feature described in **PEP 768**. This includes both the functionality to schedule code for execution in another process and the functionality to receive code for execution in the current process.

See also the *-X disable_remote_debug* command-line option.

Added in version 3.14.

PYTHON_CPU_COUNT

If this variable is set to a positive integer, it overrides the return values of `os.cpu_count()` and `os.process_cpu_count()`.

See also the *-X cpu_count* command-line option.

Added in version 3.13.

PYTHON_FROZEN_MODULES

If this variable is set to `on` or `off`, it determines whether or not frozen modules are ignored by the import machinery. A value of `on` means they get imported and `off` means they are ignored. The default is `on` for non-debug builds (the normal case) and `off` for debug builds. Note that the `importlib_bootstrap` and `importlib_bootstrap_external` frozen modules are always used, even if this flag is set to `off`.

See also the *-X frozen_modules* command-line option.

Added in version 3.13.

PYTHON_COLORS

If this variable is set to `1`, the interpreter will colorize various kinds of output. Setting it to `0` deactivates this behavior. See also *Controlling color*.

Added in version 3.13.

PYTHON_BASIC_REPL

If this variable is set to any value, the interpreter will not attempt to load the Python-based *REPL* that requires `curses` and `readline`, and will instead use the traditional parser-based *REPL*.

Added in version 3.13.

PYTHON_HISTORY

This environment variable can be used to set the location of a `.python_history` file (by default, it is `.python_history` in the user' s home directory).

Added in version 3.13.

PYTHON_GIL

If this variable is set to `1`, the global interpreter lock (GIL) will be forced on. Setting it to `0` forces the GIL off (needs Python configured with the *--disable-gil* build option).

See also the *-X gil* command-line option, which takes precedence over this variable, and whatsnew313-free-threaded-cpython.

Added in version 3.13.

PYTHON_THREAD_INHERIT_CONTEXT

If this variable is set to `1` then `Thread` will, by default, use a copy of context of of the caller of `Thread.start()` when starting. Otherwise, new threads will start with an empty context. If unset, this variable defaults to `1` on free-threaded builds and to `0` otherwise. See also *-X thread_inherit_context*.

Added in version 3.14.

PYTHON_CONTEXT_AWARE_WARNINGS

If set to `1` then the `warnings.catch_warnings` context manager will use a `ContextVar` to store warnings filter state. If unset, this variable defaults to `1` on free-threaded builds and to `0` otherwise. See *-X context_aware_warnings*.

Added in version 3.14.

1.2.1 디버그 모드 변수

PYTHONDUMPREFS

설정되면, 파이썬은 인터프리터를 종료한 후에도 살아있는 객체와 참조 횟수를 덤프합니다.

Needs Python configured with the *--with-trace-refs* build option.

PYTHONDUMPREFSFILE

If set, Python will dump objects and reference counts still alive after shutting down the interpreter into a file under the path given as the value to this environment variable.

Needs Python configured with the *--with-trace-refs* build option.

Added in version 3.11.

PYTHON_PRESITE

If this variable is set to a module, that module will be imported early in the interpreter lifecycle, before the `site` module is executed, and before the `__main__` module is created. Therefore, the imported module is not treated as `__main__`.

This can be used to execute code early during Python initialization.

To import a submodule, use `package.module` as the value, like in an import statement.

See also the *-X presite* command-line option, which takes precedence over this variable.

Needs Python configured with the *--with-pydebug* build option.

Added in version 3.13.

CHAPTER 2

유닉스 플랫폼에서 파이썬 사용하기

2.1 최신 버전의 파이썬 내려받기와 설치

2.1.1 리눅스

Python comes preinstalled on most Linux distributions, and is available as a package on all others. However there are certain features you might want to use that are not available on your distro' s package. You can compile the latest version of Python from source.

In the event that the latest version of Python doesn' t come preinstalled and isn' t in the repositories as well, you can make packages for your own distro. Have a look at the following links:

더 보기

https://www.debian.org/doc/manuals/maint-guide/first.en.html
데비안 사용자용

https://en.opensuse.org/Portal:Packaging
OpenSuse 사용자용

https://docs.fedoraproject.org/en-US/package-maintainers/Packaging_Tutorial_GNU_Hello/
Fedora 사용자용

https://slackbook.org/html/package-management-making-packages.html
Slackware 사용자용

Installing IDLE

In some cases, IDLE might not be included in your Python installation.

- For Debian and Ubuntu users:

```
sudo apt update
sudo apt install idle
```

- For Fedora, RHEL, and CentOS users:

```
sudo dnf install python3-idle
```

- For SUSE and OpenSUSE users:

```
sudo zypper install python3-idle
```

- For Alpine Linux users:

```
sudo apk add python3-idle
```

2.1.2 FreeBSD와 OpenBSD

- FreeBSD 사용자, 패키지를 추가하려면 이렇게 하십시오:

```
pkg install python3
```

- OpenBSD 사용자, 패키지를 추가하려면 이렇게 하십시오:

```
pkg_add -r python

pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/4.2/packages/<insert your␣
↪architecture here>/python-<version>.tgz
```

예를 들어 i386 사용자는 이렇게 파이썬 2.5.1 버전을 얻습니다:

```
pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/4.2/packages/i386/python-2.5.1p2.tgz
```

2.2 파이썬 빌드하기

If you want to compile CPython yourself, first thing you should do is get the source. You can download either the latest release' s source or just grab a fresh clone. (If you want to contribute patches, you will need a clone.)

빌드 프로세스는 일반적으로 다음과 같은 명령으로 구성됩니다

```
./configure
make
make install
```

Configuration options and caveats for specific Unix platforms are extensively documented in the README.rst file in the root of the Python source tree.

경고

`make install`은 `python3` 바이너리를 덮어쓰거나 가장 할 수 있습니다. 따라서 `make altinstall`을 `make install` 대신 권장하는데, *exec_prefix*`/bin/python`*version* 만 설치하기 때문입니다.

2.3 파이썬 관련 경로와 파일

These are subject to difference depending on local installation conventions; *prefix* and *exec_prefix* are installation-dependent and should be interpreted as for GNU software; they may be the same.

예를 들어, 대부분 리눅스 시스템에서, 기본값은 모두 `/usr`입니다.

파일/디렉터리	의미
exec_prefix/bin/python3	인터프리터의 권장 위치.
prefix/lib/python*version*, *exec_prefix*/lib/python*version*	표준 모듈을 포함하는 디렉터리의 권장 위치.
prefix/include/python*version*, *exec_prefix*/include/python*version*	파이썬 확장을 개발하고 인터프리터를 내장하는 데 필요한 인클루드 파일을 포함하는 디렉터리의 권장 위치.

2.4 잡동사니

유닉스에서 파이썬 스크립트를 쉽게 사용하려면, 실행 파일로 만들어야 합니다. 예를 들어, 이렇게

```
$ chmod +x script
```

그리고, 스크립트의 상단에 적절한 셔뱅(Shebang) 줄을 넣습니다. 좋은 선택은 대개 이렇습니다

```
#!/usr/bin/env python3
```

이것은 PATH 전체에서 파이썬 인터프리터를 검색합니다. 그러나, 일부 유닉스에는 **env** 명령이 없을 수 있으므로, 인터프리터 경로로 /usr/bin/python3를 하드 코딩해야 할 수 있습니다.

파이썬 스크립트에서 셸 명령을 사용하려면, subprocess 모듈을 보십시오.

2.5 Custom OpenSSL

1. To use your vendor' s OpenSSL configuration and system trust store, locate the directory with openssl.cnf file or symlink in /etc. On most distribution the file is either in /etc/ssl or /etc/pki/tls. The directory should also contain a cert.pem file and/or a certs directory.

   ```
   $ find /etc/ -name openssl.cnf -printf "%h\n"
   /etc/ssl
   ```

2. Download, build, and install OpenSSL. Make sure you use install_sw and not install. The install_sw target does not override openssl.cnf.

   ```
   $ curl -O https://www.openssl.org/source/openssl-VERSION.tar.gz
   $ tar xzf openssl-VERSION
   $ pushd openssl-VERSION
   $ ./config \
       --prefix=/usr/local/custom-openssl \
       --libdir=lib \
       --openssldir=/etc/ssl
   $ make -j1 depend
   $ make -j8
   $ make install_sw
   $ popd
   ```

3. Build Python with custom OpenSSL (see the configure --with-openssl and --with-openssl-rpath options)

   ```
   $ pushd python-3.x.x
   $ ./configure -C \
       --with-openssl=/usr/local/custom-openssl \
       --with-openssl-rpath=auto \
       --prefix=/usr/local/python-3.x.x
   $ make -j8
   $ make altinstall
   ```

참고

Patch releases of OpenSSL have a backwards compatible ABI. You don’ t need to recompile Python to update OpenSSL. It’ s sufficient to replace the custom OpenSSL installation with a newer version.

CHAPTER 3

Configure Python

3.1 Build Requirements

Features and minimum versions required to build CPython:

- A C11 compiler. Optional C11 features are not required.
- On Windows, Microsoft Visual Studio 2017 or later is required.
- Support for IEEE 754 floating-point numbers and floating-point Not-a-Number (NaN).
- Support for threads.
- OpenSSL 1.1.1 is the minimum version and OpenSSL 3.0.16 is the recommended minimum version for the `ssl` and `hashlib` extension modules.
- SQLite 3.15.2 for the `sqlite3` extension module.
- Tcl/Tk 8.5.12 for the `tkinter` module.
- Autoconf 2.72 and aclocal 1.16.5 are required to regenerate the `configure` script.

버전 3.1에서 변경: Tcl/Tk version 8.3.1 is now required.

버전 3.5에서 변경: On Windows, Visual Studio 2015 or later is now required. Tcl/Tk version 8.4 is now required.

버전 3.6에서 변경: Selected C99 features are now required, like `<stdint.h>` and `static inline` functions.

버전 3.7에서 변경: Thread support and OpenSSL 1.0.2 are now required.

버전 3.10에서 변경: OpenSSL 1.1.1 is now required. Require SQLite 3.7.15.

버전 3.11에서 변경: C11 compiler, IEEE 754 and NaN support are now required. On Windows, Visual Studio 2017 or later is required. Tcl/Tk version 8.5.12 is now required for the `tkinter` module.

버전 3.13에서 변경: Autoconf 2.71, aclocal 1.16.5 and SQLite 3.15.2 are now required.

버전 3.14에서 변경: Autoconf 2.72 is now required.

See also **PEP 7** “Style Guide for C Code” and **PEP 11** “CPython platform support”.

3.2 Generated files

To reduce build dependencies, Python source code contains multiple generated files. Commands to regenerate all generated files:

```
make regen-all
make regen-stdlib-module-names
make regen-limited-abi
make regen-configure
```

The `Makefile.pre.in` file documents generated files, their inputs, and tools used to regenerate them. Search for `regen-*` make targets.

3.2.1 configure script

The `make regen-configure` command regenerates the `aclocal.m4` file and the `configure` script using the `Tools/build/regen-configure.sh` shell script which uses an Ubuntu container to get the same tools versions and have a reproducible output.

The container is optional, the following command can be run locally:

```
autoreconf -ivf -Werror
```

The generated files can change depending on the exact `autoconf-archive`, `aclocal` and `pkg-config` versions.

3.3 Configure Options

List all `configure` script options using:

```
./configure --help
```

See also the `Misc/SpecialBuilds.txt` in the Python source distribution.

3.3.1 General Options

--enable-loadable-sqlite-extensions

Support loadable extensions in the `_sqlite` extension module (default is no) of the `sqlite3` module.

See the `sqlite3.Connection.enable_load_extension()` method of the `sqlite3` module.

Added in version 3.6.

--disable-ipv6

Disable IPv6 support (enabled by default if supported), see the `socket` module.

--enable-big-digits=`[15|30]`

Define the size in bits of Python `int` digits: 15 or 30 bits.

By default, the digit size is 30.

Define the `PYLONG_BITS_IN_DIGIT` to `15` or `30`.

See `sys.int_info.bits_per_digit`.

--with-suffix=`SUFFIX`

Set the Python executable suffix to *SUFFIX*.

The default suffix is `.exe` on Windows and macOS (`python.exe` executable), `.js` on Emscripten node, `.html` on Emscripten browser, `.wasm` on WASI, and an empty string on other platforms (`python` executable).

버전 3.11에서 변경: The default suffix on WASM platform is one of `.js`, `.html` or `.wasm`.

--with-tzpath=<list of absolute paths separated by pathsep>

Select the default time zone search path for `zoneinfo.TZPATH`. See the Compile-time configuration of the `zoneinfo` module.

Default: `/usr/share/zoneinfo:/usr/lib/zoneinfo:/usr/share/lib/zoneinfo:/etc/zoneinfo`.

See `os.pathsep` path separator.

Added in version 3.9.

--without-decimal-contextvar

Build the `_decimal` extension module using a thread-local context rather than a coroutine-local context (default), see the `decimal` module.

See `decimal.HAVE_CONTEXTVAR` and the `contextvars` module.

Added in version 3.9.

--with-dbmliborder=<list of backend names>

Override order to check db backends for the `dbm` module

A valid value is a colon (`:`) separated string with the backend names:

- `ndbm`;
- `gdbm`;
- `bdb`.

--without-c-locale-coercion

Disable C locale coercion to a UTF-8 based locale (enabled by default).

Don't define the `PY_COERCE_C_LOCALE` macro.

See *`PYTHONCOERCECLOCALE`* and the **PEP 538**.

--with-platlibdir=DIRNAME

Python library directory name (default is `lib`).

Fedora and SuSE use `lib64` on 64-bit platforms.

See `sys.platlibdir`.

Added in version 3.9.

--with-wheel-pkg-dir=PATH

Directory of wheel packages used by the `ensurepip` module (none by default).

Some Linux distribution packaging policies recommend against bundling dependencies. For example, Fedora installs wheel packages in the `/usr/share/python-wheels/` directory and don't install the `ensurepip._bundled` package.

Added in version 3.10.

--with-pkg-config=[check|yes|no]

Whether configure should use **pkg-config** to detect build dependencies.

- `check` (default): **pkg-config** is optional
- `yes`: **pkg-config** is mandatory
- `no`: configure does not use **pkg-config** even when present

Added in version 3.11.

--enable-pystats

Turn on internal Python performance statistics gathering.

By default, statistics gathering is off. Use `python3 -X pystats` command or set `PYTHONSTATS=1` environment variable to turn on statistics gathering at Python startup.

At Python exit, dump statistics if statistics gathering was on and not cleared.

Effects:

- Add *-X pystats* command line option.
- Add `PYTHONSTATS` environment variable.
- Define the `Py_STATS` macro.
- Add functions to the `sys` module:
  - `sys._stats_on()`: Turns on statistics gathering.
  - `sys._stats_off()`: Turns off statistics gathering.
  - `sys._stats_clear()`: Clears the statistics.
  - `sys._stats_dump()`: Dump statistics to file, and clears the statistics.

The statistics will be dumped to a arbitrary (probably unique) file in `/tmp/py_stats/` (Unix) or `C:\temp\py_stats\` (Windows). If that directory does not exist, results will be printed on stderr.

Use `Tools/scripts/summarize_stats.py` to read the stats.

Statistics:

- Opcode:
  - Specialization: success, failure, hit, deferred, miss, deopt, failures;
  - Execution count;
  - Pair count.
- Call:
  - Inlined Python calls;
  - PyEval calls;
  - Frames pushed;
  - Frame object created;
  - Eval calls: vector, generator, legacy, function VECTORCALL, build class, slot, function "ex", API, method.
- Object:
  - incref and decref;
  - interpreter incref and decref;
  - allocations: all, 512 bytes, 4 kiB, big;
  - free;
  - to/from free lists;
  - dictionary materialized/dematerialized;
  - type cache;
  - optimization attempts;
  - optimization traces created/executed;
  - uops executed.

- Garbage collector:
  - Garbage collections;
  - Objects visited;
  - Objects collected.

Added in version 3.11.

--disable-gil

Enables **experimental** support for running Python without the *global interpreter lock* (GIL): free threading build.

Defines the `Py_GIL_DISABLED` macro and adds `"t"` to `sys.abiflags`.

See whatsnew313-free-threaded-cpython for more detail.

Added in version 3.13.

--enable-experimental-jit=[no|yes|yes-off|interpreter]

Indicate how to integrate the JIT compiler.

- `no` - build the interpreter without the JIT.
- `yes` - build the interpreter with the JIT.
- `yes-off` - build the interpreter with the JIT but disable it by default.
- `interpreter` - build the interpreter without the JIT, but with the tier 2 enabled interpreter.

By convention, `--enable-experimental-jit` is a shorthand for `--enable-experimental-jit=yes`.

> **참고**
>
> When building CPython with JIT enabled, ensure that your system has Python 3.11 or later installed.

Added in version 3.13.

PKG_CONFIG

Path to `pkg-config` utility.

PKG_CONFIG_LIBDIR

PKG_CONFIG_PATH

`pkg-config` options.

3.3.2 C compiler options

CC

C compiler command.

CFLAGS

C compiler flags.

CPP

C preprocessor command.

CPPFLAGS

C preprocessor flags, e.g. `-I`*`include_dir`*.

3.3.3 Linker options

LDFLAGS

Linker flags, e.g. `-Llibrary_directory`.

LIBS

Libraries to pass to the linker, e.g. `-llibrary`.

MACHDEP

Name for machine-dependent library files.

3.3.4 Options for third-party dependencies

Added in version 3.11.

BZIP2_CFLAGS

BZIP2_LIBS

C compiler and linker flags to link Python to `libbz2`, used by `bz2` module, overriding `pkg-config`.

CURSES_CFLAGS

CURSES_LIBS

C compiler and linker flags for `libncurses` or `libncursesw`, used by `curses` module, overriding `pkg-config`.

GDBM_CFLAGS

GDBM_LIBS

C compiler and linker flags for `gdbm`.

LIBB2_CFLAGS

LIBB2_LIBS

C compiler and linker flags for `libb2` (BLAKE2), used by `hashlib` module, overriding `pkg-config`.

LIBEDIT_CFLAGS

LIBEDIT_LIBS

C compiler and linker flags for `libedit`, used by `readline` module, overriding `pkg-config`.

LIBFFI_CFLAGS

LIBFFI_LIBS

C compiler and linker flags for `libffi`, used by `ctypes` module, overriding `pkg-config`.

LIBMPDEC_CFLAGS

LIBMPDEC_LIBS

C compiler and linker flags for `libmpdec`, used by `decimal` module, overriding `pkg-config`.

> **참고**
>
> These environment variables have no effect unless `--with-system-libmpdec` is specified.

LIBLZMA_CFLAGS

LIBLZMA_LIBS

C compiler and linker flags for `liblzma`, used by `lzma` module, overriding `pkg-config`.

LIBREADLINE_CFLAGS

LIBREADLINE_LIBS
C compiler and linker flags for `libreadline`, used by `readline` module, overriding `pkg-config`.

LIBSQLITE3_CFLAGS

LIBSQLITE3_LIBS
C compiler and linker flags for `libsqlite3`, used by `sqlite3` module, overriding `pkg-config`.

LIBUUID_CFLAGS

LIBUUID_LIBS
C compiler and linker flags for `libuuid`, used by `uuid` module, overriding `pkg-config`.

PANEL_CFLAGS

PANEL_LIBS
C compiler and linker flags for PANEL, overriding `pkg-config`.

C compiler and linker flags for `libpanel` or `libpanelw`, used by `curses.panel` module, overriding `pkg-config`.

TCLTK_CFLAGS

TCLTK_LIBS
C compiler and linker flags for TCLTK, overriding `pkg-config`.

ZLIB_CFLAGS

ZLIB_LIBS
C compiler and linker flags for `libzlib`, used by `gzip` module, overriding `pkg-config`.

3.3.5 WebAssembly Options

--enable-wasm-dynamic-linking
Turn on dynamic linking support for WASM.

Dynamic linking enables `dlopen`. File size of the executable increases due to limited dead code elimination and additional features.

Added in version 3.11.

--enable-wasm-pthreads
Turn on pthreads support for WASM.

Added in version 3.11.

3.3.6 Install Options

--prefix=PREFIX
Install architecture-independent files in PREFIX. On Unix, it defaults to `/usr/local`.

This value can be retrieved at runtime using `sys.prefix`.

As an example, one can use `--prefix="$HOME/.local/"` to install a Python in its home directory.

--exec-prefix=EPREFIX
Install architecture-dependent files in EPREFIX, defaults to *--prefix*.

This value can be retrieved at runtime using `sys.exec_prefix`.

--disable-test-modules
Don' t build nor install test modules, like the `test` package or the `_testcapi` extension module (built and installed by default).

Added in version 3.10.

--with-ensurepip=[upgrade|install|no]

Select the `ensurepip` command run on Python installation:

- `upgrade` (default): run `python -m ensurepip --altinstall --upgrade` command.
- `install`: run `python -m ensurepip --altinstall` command;
- `no`: don' t run ensurepip;

Added in version 3.6.

3.3.7 Performance options

Configuring Python using `--enable-optimizations --with-lto` (PGO + LTO) is recommended for best performance. The experimental `--enable-bolt` flag can also be used to improve performance.

--enable-optimizations

Enable Profile Guided Optimization (PGO) using *PROFILE_TASK* (disabled by default).

The C compiler Clang requires `llvm-profdata` program for PGO. On macOS, GCC also requires it: GCC is just an alias to Clang on macOS.

Disable also semantic interposition in libpython if `--enable-shared` and GCC is used: add `-fno-semantic-interposition` to the compiler and linker flags.

> **참고**
>
> During the build, you may encounter compiler warnings about profile data not being available for some source files. These warnings are harmless, as only a subset of the code is exercised during profile data acquisition. To disable these warnings on Clang, manually suppress them by adding `-Wno-profile-instr-unprofiled` to *CFLAGS*.

Added in version 3.6.

버전 3.10에서 변경: Use `-fno-semantic-interposition` on GCC.

PROFILE_TASK

Environment variable used in the Makefile: Python command line arguments for the PGO generation task.

Default: `-m test --pgo --timeout=$(TESTTIMEOUT)`.

Added in version 3.8.

버전 3.13에서 변경: Task failure is no longer ignored silently.

--with-lto=[full|thin|no|yes]

Enable Link Time Optimization (LTO) in any build (disabled by default).

The C compiler Clang requires `llvm-ar` for LTO (`ar` on macOS), as well as an LTO-aware linker (`ld.gold` or `lld`).

Added in version 3.6.

Added in version 3.11: To use ThinLTO feature, use `--with-lto=thin` on Clang.

버전 3.12에서 변경: Use ThinLTO as the default optimization policy on Clang if the compiler accepts the flag.

--enable-bolt

Enable usage of the BOLT post-link binary optimizer (disabled by default).

BOLT is part of the LLVM project but is not always included in their binary distributions. This flag requires that `llvm-bolt` and `merge-fdata` are available.

BOLT is still a fairly new project so this flag should be considered experimental for now. Because this tool operates on machine code its success is dependent on a combination of the build environment + the other

optimization configure args + the CPU architecture, and not all combinations are supported. BOLT versions before LLVM 16 are known to crash BOLT under some scenarios. Use of LLVM 16 or newer for BOLT optimization is strongly encouraged.

The `BOLT_INSTRUMENT_FLAGS` and `BOLT_APPLY_FLAGS` **`configure`** variables can be defined to override the default set of arguments for **`llvm-bolt`** to instrument and apply BOLT data to binaries, respectively.

Added in version 3.12.

`BOLT_APPLY_FLAGS`

Arguments to `llvm-bolt` when creating a BOLT optimized binary.

Added in version 3.12.

`BOLT_INSTRUMENT_FLAGS`

Arguments to `llvm-bolt` when instrumenting binaries.

Added in version 3.12.

`--with-computed-gotos`

Enable computed gotos in evaluation loop (enabled by default on supported compilers).

`--with-tail-call-interp`

Enable interpreters using tail calls in CPython. If enabled, enabling PGO (*`--enable-optimizations`*) is highly recommended. This option specifically requires a C compiler with proper tail call support, and the preserve_none calling convention. For example, Clang 19 and newer supports this feature.

Added in version 3.14.

`--without-mimalloc`

Disable the fast mimalloc allocator (enabled by default).

See also *`PYTHONMALLOC`* environment variable.

`--without-pymalloc`

Disable the specialized Python memory allocator pymalloc (enabled by default).

See also *`PYTHONMALLOC`* environment variable.

`--without-doc-strings`

Disable static documentation strings to reduce the memory footprint (enabled by default). Documentation strings defined in Python are not affected.

Don' t define the `WITH_DOC_STRINGS` macro.

See the `PyDoc_STRVAR()` macro.

`--enable-profiling`

Enable C-level code profiling with `gprof` (disabled by default).

`--with-strict-overflow`

Add `-fstrict-overflow` to the C compiler flags (by default we add `-fno-strict-overflow` instead).

`--without-remote-debug`

Deactivate remote debugging support described in **PEP 768** (enabled by default). When this flag is provided the code that allows the interpreter to schedule the execution of a Python file in a separate process as described in **PEP 768** is not compiled. This includes both the functionality to schedule code to be executed and the functionality to receive code to be executed.

Added in version 3.14.

3.3.8 Python Debug Build

A debug build is Python built with the *`--with-pydebug`* configure option.

Effects of a debug build:

- Display all warnings by default: the list of default warning filters is empty in the `warnings` module.
- Add `d` to `sys.abiflags`.
- Add `sys.gettotalrefcount()` function.
- Add *`-X showrefcount`* command line option.
- Add *`-d`* command line option and *`PYTHONDEBUG`* environment variable to debug the parser.
- Add support for the `__lltrace__` variable: enable low-level tracing in the bytecode evaluation loop if the variable is defined.
- Install debug hooks on memory allocators to detect buffer overflow and other memory errors.
- Define `Py_DEBUG` and `Py_REF_DEBUG` macros.
- Add runtime checks: code surrounded by `#ifdef Py_DEBUG` and `#endif`. Enable `assert(...)` and `_PyObject_ASSERT(...)` assertions: don' t set the `NDEBUG` macro (see also the *`--with-assertions`* configure option). Main runtime checks:
  - Add sanity checks on the function arguments.
  - Unicode and int objects are created with their memory filled with a pattern to detect usage of uninitialized objects.
  - Ensure that functions which can clear or replace the current exception are not called with an exception raised.
  - Check that deallocator functions don' t change the current exception.
  - The garbage collector (`gc.collect()` function) runs some basic checks on objects consistency.
  - The `Py_SAFE_DOWNCAST()` macro checks for integer underflow and overflow when downcasting from wide types to narrow types.

See also the Python Development Mode and the *`--with-trace-refs`* configure option.

버전 3.8에서 변경: Release builds and debug builds are now ABI compatible: defining the `Py_DEBUG` macro no longer implies the `Py_TRACE_REFS` macro (see the *`--with-trace-refs`* option).

3.3.9 Debug options

`--with-pydebug`

Build Python in debug mode: define the `Py_DEBUG` macro (disabled by default).

`--with-trace-refs`

Enable tracing references for debugging purpose (disabled by default).

Effects:

- Define the `Py_TRACE_REFS` macro.
- Add `sys.getobjects()` function.
- Add *`PYTHONDUMPREFS`* environment variable.

The *`PYTHONDUMPREFS`* environment variable can be used to dump objects and reference counts still alive at Python exit.

Statically allocated objects are not traced.

Added in version 3.8.

버전 3.13에서 변경: This build is now ABI compatible with release build and *debug build*.

--with-assertions

Build with C assertions enabled (default is no): `assert(...);` and `_PyObject_ASSERT(...);`.

If set, the `NDEBUG` macro is not defined in the *OPT* compiler variable.

See also the *--with-pydebug* option (*debug build*) which also enables assertions.

Added in version 3.6.

--with-valgrind

Enable Valgrind support (default is no).

--with-dtrace

Enable DTrace support (default is no).

See Instrumenting CPython with DTrace and SystemTap.

Added in version 3.6.

--with-address-sanitizer

Enable AddressSanitizer memory error detector, `asan` (default is no).

Added in version 3.6.

--with-memory-sanitizer

Enable MemorySanitizer allocation error detector, `msan` (default is no).

Added in version 3.6.

--with-undefined-behavior-sanitizer

Enable UndefinedBehaviorSanitizer undefined behaviour detector, `ubsan` (default is no).

Added in version 3.6.

--with-thread-sanitizer

Enable ThreadSanitizer data race detector, `tsan` (default is no).

Added in version 3.13.

3.3.10 Linker options

--enable-shared

Enable building a shared Python library: `libpython` (default is no).

--without-static-libpython

Do not build `libpythonMAJOR.MINOR.a` and do not install `python.o` (built and enabled by default).

Added in version 3.10.

3.3.11 Libraries options

--with-libs`='lib1 ...'`

Link against additional libraries (default is no).

--with-system-expat

Build the `pyexpat` module using an installed `expat` library (default is no).

--with-system-libmpdec

Build the `_decimal` extension module using an installed `mpdecimal` library, see the `decimal` module (default is yes).

Added in version 3.3.

버전 3.13에서 변경: Default to using the installed `mpdecimal` library.

Deprecated since version 3.13, will be removed in version 3.15: A copy of the `mpdecimal` library sources will no longer be distributed with Python 3.15.

> 더 보기
>
> *LIBMPDEC_CFLAGS* and *LIBMPDEC_LIBS*.

--with-readline=readline|editline

Designate a backend library for the `readline` module.

- readline: Use readline as the backend.
- editline: Use editline as the backend.

Added in version 3.10.

--without-readline

Don' t build the `readline` module (built by default).

Don' t define the `HAVE_LIBREADLINE` macro.

Added in version 3.10.

--with-libm=STRING

Override `libm` math library to *STRING* (default is system-dependent).

--with-libc=STRING

Override `libc` C library to *STRING* (default is system-dependent).

--with-openssl=DIR

Root of the OpenSSL directory.

Added in version 3.7.

--with-openssl-rpath=[no|auto|DIR]

Set runtime library directory (rpath) for OpenSSL libraries:

- `no` (default): don' t set rpath;
- `auto`: auto-detect rpath from *--with-openssl* and `pkg-config`;
- *DIR*: set an explicit rpath.

Added in version 3.10.

3.3.12 Security Options

--with-hash-algorithm=[fnv|siphash13|siphash24]

Select hash algorithm for use in `Python/pyhash.c`:

- `siphash13` (default);
- `siphash24`;
- `fnv`.

Added in version 3.4.

Added in version 3.11: `siphash13` is added and it is the new default.

--with-builtin-hashlib-hashes=md5,sha1,sha256,sha512,sha3,blake2

Built-in hash modules:

- `md5`;
- `sha1`;

- `sha256`;
- `sha512`;
- `sha3` (with shake);
- `blake2`.

Added in version 3.9.

--with-ssl-default-suites=[python|openssl|STRING]

Override the OpenSSL default cipher suites string:

- `python` (default): use Python' s preferred selection;
- `openssl`: leave OpenSSL' s defaults untouched;
- *STRING*: use a custom string

See the `ssl` module.

Added in version 3.7.

버전 3.10에서 변경: The settings `python` and *STRING* also set TLS 1.2 as minimum protocol version.

--disable-safety

Disable compiler options that are recommended by OpenSSF for security reasons with no performance overhead. If this option is not enabled, CPython will be built based on safety compiler options with no slow down. When this option is enabled, CPython will not be built with the compiler options listed below.

The following compiler options are disabled with `--disable-safety`:

- -fstack-protector-strong: Enable run-time checks for stack-based buffer overflows.
- -Wtrampolines: Enable warnings about trampolines that require executable stacks.

Added in version 3.14.

--enable-slower-safety

Enable compiler options that are recommended by OpenSSF for security reasons which require overhead. If this option is not enabled, CPython will not be built based on safety compiler options which performance impact. When this option is enabled, CPython will be built with the compiler options listed below.

The following compiler options are enabled with `--enable-slower-safety`:

- -D_FORTIFY_SOURCE=3: Fortify sources with compile- and run-time checks for unsafe libc usage and buffer overflows.

Added in version 3.14.

3.3.13 macOS Options

See Mac/README.rst.

--enable-universalsdk

--enable-universalsdk=SDKDIR

Create a universal binary build. *SDKDIR* specifies which macOS SDK should be used to perform the build (default is no).

--enable-framework

--enable-framework=INSTALLDIR

Create a Python.framework rather than a traditional Unix install. Optional *INSTALLDIR* specifies the installation path (default is no).

--with-universal-archs=ARCH

Specify the kind of universal binary that should be created. This option is only valid when *--enable-universalsdk* is set.

Options:

- `universal2` (x86-64 and arm64);
- `32-bit` (PPC and i386);
- `64-bit` (PPC64 and x86-64);
- `3-way` (i386, PPC and x86-64);
- `intel` (i386 and x86-64);
- `intel-32` (i386);
- `intel-64` (x86-64);
- `all` (PPC, i386, PPC64 and x86-64).

Note that values for this configuration item are *not* the same as the identifiers used for universal binary wheels on macOS. See the Python Packaging User Guide for details on the packaging platform compatibility tags used on macOS

--with-framework-name=FRAMEWORK

Specify the name for the python framework on macOS only valid when *--enable-framework* is set (default: `Python`).

--with-app-store-compliance

--with-app-store-compliance=PATCH-FILE

The Python standard library contains strings that are known to trigger automated inspection tool errors when submitted for distribution by the macOS and iOS App Stores. If enabled, this option will apply the list of patches that are known to correct app store compliance. A custom patch file can also be specified. This option is disabled by default.

Added in version 3.13.

3.3.14 iOS Options

See iOS/README.rst.

--enable-framework=INSTALLDIR

Create a Python.framework. Unlike macOS, the *INSTALLDIR* argument specifying the installation path is mandatory.

--with-framework-name=FRAMEWORK

Specify the name for the framework (default: `Python`).

3.3.15 Cross Compiling Options

Cross compiling, also known as cross building, can be used to build Python for another CPU architecture or platform. Cross compiling requires a Python interpreter for the build platform. The version of the build Python must match the version of the cross compiled host Python.

--build=BUILD

configure for building on BUILD, usually guessed by **config.guess**.

--host=HOST

cross-compile to build programs to run on HOST (target platform)

--with-build-python=path/to/python

path to build `python` binary for cross compiling

Added in version 3.11.

CONFIG_SITE=file

An environment variable that points to a file with configure overrides.

Example *config.site* file:

```
# config.site-aarch64
ac_cv_buggy_getaddrinfo=no
ac_cv_file__dev_ptmx=yes
ac_cv_file__dev_ptc=no
```

HOSTRUNNER

Program to run CPython for the host platform for cross-compilation.

Added in version 3.11.

Cross compiling example:

```
CONFIG_SITE=config.site-aarch64 ../configure \
    --build=x86_64-pc-linux-gnu \
    --host=aarch64-unknown-linux-gnu \
    --with-build-python=../x86_64/python
```

3.4 Python Build System

3.4.1 Main files of the build system

- `configure.ac` => `configure`;
- `Makefile.pre.in` => `Makefile` (created by `configure`);
- `pyconfig.h` (created by `configure`);
- `Modules/Setup`: C extensions built by the Makefile using `Module/makesetup` shell script;

3.4.2 Main build steps

- C files (`.c`) are built as object files (`.o`).
- A static `libpython` library (`.a`) is created from objects files.
- `python.o` and the static `libpython` library are linked into the final `python` program.
- C extensions are built by the Makefile (see `Modules/Setup`).

3.4.3 Main Makefile targets

make

For the most part, when rebuilding after editing some code or refreshing your checkout from upstream, all you need to do is execute `make`, which (per Make' s semantics) builds the default target, the first one defined in the Makefile. By tradition (including in the CPython project) this is usually the `all` target. The `configure` script expands an `autoconf` variable, `@DEF_MAKE_ALL_RULE@` to describe precisely which targets `make all` will build. The three choices are:

- `profile-opt` (configured with `--enable-optimizations`)
- `build_wasm` (chosen if the host platform matches `wasm32-wasi*` or `wasm32-emscripten`)
- `build_all` (configured without explicitly using either of the others)

Depending on the most recent source file changes, Make will rebuild any targets (object files and executables) deemed out-of-date, including running `configure` again if necessary. Source/target dependencies are many and maintained manually however, so Make sometimes doesn' t have all the information necessary to correctly detect all targets which need to be rebuilt. Depending on which targets aren' t rebuilt, you might experience a number of problems. If you have build or test problems which you can' t otherwise explain, `make clean && make` should work around most dependency problems, at the expense of longer build times.

make platform

Build the `python` program, but don' t build the standard library extension modules. This generates a file named `platform` which contains a single line describing the details of the build platform, e.g., `macosx-14.3-arm64-3.12` or `linux-x86_64-3.13`.

make profile-opt

Build Python using profile-guided optimization (PGO). You can use the configure *`--enable-optimizations`* option to make this the default target of the `make` command (`make all` or just `make`).

make clean

Remove built files.

make distclean

In addition to the work done by `make clean`, remove files created by the configure script. `configure` will have to be run before building again.[1]

make install

Build the `all` target and install Python.

make test

Build the `all` target and run the Python test suite with the `--fast-ci` option without GUI tests. Variables:

- `TESTOPTS`: additional regrtest command-line options.
- `TESTPYTHONOPTS`: additional Python command-line options.
- `TESTTIMEOUT`: timeout in seconds (default: 10 minutes).

make ci

This is similar to `make test`, but uses the `-ugui` to also run GUI tests.

Added in version 3.14.

make buildbottest

This is similar to `make test`, but uses the `--slow-ci` option and default timeout of 20 minutes, instead of `--fast-ci` option.

make regen-all

Regenerate (almost) all generated files. These include (but are not limited to) bytecode cases, and parser generator file. `make regen-stdlib-module-names` and `autoconf` must be run separately for the remaining *generated files*.

[1] `git clean -fdx` is an even more extreme way to "clean" your checkout. It removes all files not known to Git. When bug hunting using `git bisect`, this is recommended between probes to guarantee a completely clean build. **Use with care**, as it will delete all files not checked into Git, including your new, uncommitted work.

3.4.4 C extensions

Some C extensions are built as built-in modules, like the `sys` module. They are built with the `Py_BUILD_CORE_BUILTIN` macro defined. Built-in modules have no `__file__` attribute:

```
>>> import sys
>>> sys
<module 'sys' (built-in)>
>>> sys.__file__
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'sys' has no attribute '__file__'
```

Other C extensions are built as dynamic libraries, like the `_asyncio` module. They are built with the `Py_BUILD_CORE_MODULE` macro defined. Example on Linux x86-64:

```
>>> import _asyncio
>>> _asyncio
<module '_asyncio' from '/usr/lib64/python3.9/lib-dynload/_asyncio.cpython-39-x86_
↪64-linux-gnu.so'>
>>> _asyncio.__file__
'/usr/lib64/python3.9/lib-dynload/_asyncio.cpython-39-x86_64-linux-gnu.so'
```

`Modules/Setup` is used to generate Makefile targets to build C extensions. At the beginning of the files, C extensions are built as built-in modules. Extensions defined after the `*shared*` marker are built as dynamic libraries.

The `PyAPI_FUNC()`, `PyAPI_DATA()` and `PyMODINIT_FUNC` macros of `Include/exports.h` are defined differently depending if the `Py_BUILD_CORE_MODULE` macro is defined:

- Use `Py_EXPORTED_SYMBOL` if the `Py_BUILD_CORE_MODULE` is defined
- Use `Py_IMPORTED_SYMBOL` otherwise.

If the `Py_BUILD_CORE_BUILTIN` macro is used by mistake on a C extension built as a shared library, its `PyInit_xxx()` function is not exported, causing an `ImportError` on import.

3.5 Compiler and linker flags

Options set by the `./configure` script and environment variables and used by `Makefile`.

3.5.1 Preprocessor flags

CONFIGURE_CPPFLAGS

Value of *CPPFLAGS* variable passed to the `./configure` script.

Added in version 3.6.

CPPFLAGS

(Objective) C/C++ preprocessor flags, e.g. `-I`*include_dir* if you have headers in a nonstandard directory *include_dir*.

Both *CPPFLAGS* and *LDFLAGS* need to contain the shell' s value to be able to build extension modules using the directories specified in the environment variables.

BASECPPFLAGS

Added in version 3.4.

PY_CPPFLAGS

Extra preprocessor flags added for building the interpreter object files.

Default: `$ (BASECPPFLAGS) -I. -I$ (srcdir)/Include $ (CONFIGURE_CPPFLAGS) $ (CPPFLAGS).`

Added in version 3.2.

3.5.2 Compiler flags

CC

C compiler command.

Example: `gcc -pthread`.

CXX

C++ compiler command.

Example: `g++ -pthread`.

CFLAGS

C compiler flags.

CFLAGS_NODIST

CFLAGS_NODIST is used for building the interpreter and stdlib C extensions. Use it when a compiler flag should *not* be part of *CFLAGS* once Python is installed (gh-65320).

In particular, *CFLAGS* should not contain:

- the compiler flag `-I` (for setting the search path for include files). The `-I` flags are processed from left to right, and any flags in *CFLAGS* would take precedence over user- and package-supplied `-I` flags.
- hardening flags such as `-Werror` because distributions cannot control whether packages installed by users conform to such heightened standards.

Added in version 3.5.

COMPILEALL_OPTS

Options passed to the `compileall` command line when building PYC files in `make install`. Default: `-j0`.

Added in version 3.12.

EXTRA_CFLAGS

Extra C compiler flags.

CONFIGURE_CFLAGS

Value of *CFLAGS* variable passed to the `./configure` script.

Added in version 3.2.

CONFIGURE_CFLAGS_NODIST

Value of *CFLAGS_NODIST* variable passed to the `./configure` script.

Added in version 3.5.

BASECFLAGS

Base compiler flags.

OPT

Optimization flags.

CFLAGS_ALIASING

Strict or non-strict aliasing flags used to compile `Python/dtoa.c`.

Added in version 3.7.

CCSHARED

Compiler flags used to build a shared library.

For example, `-fPIC` is used on Linux and on BSD.

CFLAGSFORSHARED

Extra C flags added for building the interpreter object files.

Default: `$(CCSHARED)` when *`--enable-shared`* is used, or an empty string otherwise.

PY_CFLAGS

Default: `$(BASECFLAGS) $(OPT) $(CONFIGURE_CFLAGS) $(CFLAGS) $(EXTRA_CFLAGS)`.

PY_CFLAGS_NODIST

Default: `$(CONFIGURE_CFLAGS_NODIST) $(CFLAGS_NODIST) -I$(srcdir)/Include/internal`.

Added in version 3.5.

PY_STDMODULE_CFLAGS

C flags used for building the interpreter object files.

Default: `$(PY_CFLAGS) $(PY_CFLAGS_NODIST) $(PY_CPPFLAGS) $(CFLAGSFORSHARED)`.

Added in version 3.7.

PY_CORE_CFLAGS

Default: `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE`.

Added in version 3.2.

PY_BUILTIN_MODULE_CFLAGS

Compiler flags to build a standard library extension module as a built-in module, like the `posix` module.

Default: `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE_BUILTIN`.

Added in version 3.8.

PURIFY

Purify command. Purify is a memory debugger program.

Default: empty string (not used).

3.5.3 Linker flags

LINKCC

Linker command used to build programs like `python` and `_testembed`.

Default: `$(PURIFY) $(CC)`.

CONFIGURE_LDFLAGS

Value of *`LDFLAGS`* variable passed to the `./configure` script.

Avoid assigning *`CFLAGS`*, *`LDFLAGS`*, etc. so users can use them on the command line to append to these values without stomping the pre-set values.

Added in version 3.2.

LDFLAGS_NODIST

`LDFLAGS_NODIST` is used in the same manner as *`CFLAGS_NODIST`*. Use it when a linker flag should *not* be part of *`LDFLAGS`* once Python is installed (gh-65320).

In particular, *`LDFLAGS`* should not contain:

- the compiler flag `-L` (for setting the search path for libraries). The `-L` flags are processed from left to right, and any flags in *`LDFLAGS`* would take precedence over user- and package-supplied `-L` flags.

CONFIGURE_LDFLAGS_NODIST

Value of *`LDFLAGS_NODIST`* variable passed to the `./configure` script.

Added in version 3.8.

LDFLAGS

Linker flags, e.g. `-Llib_dir` if you have libraries in a nonstandard directory *lib_dir*.

Both *CPPFLAGS* and *LDFLAGS* need to contain the shell' s value to be able to build extension modules using the directories specified in the environment variables.

LIBS

Linker flags to pass libraries to the linker when linking the Python executable.

Example: `-lrt`.

LDSHARED

Command to build a shared library.

Default: `@LDSHARED@ $(PY_LDFLAGS)`.

BLDSHARED

Command to build `libpython` shared library.

Default: `@BLDSHARED@ $(PY_CORE_LDFLAGS)`.

PY_LDFLAGS

Default: `$(CONFIGURE_LDFLAGS) $(LDFLAGS)`.

PY_LDFLAGS_NODIST

Default: `$(CONFIGURE_LDFLAGS_NODIST) $(LDFLAGS_NODIST)`.

Added in version 3.8.

PY_CORE_LDFLAGS

Linker flags used for building the interpreter object files.

Added in version 3.8.

CHAPTER 4

윈도우에서 파이썬 사용하기

이 문서는 Microsoft 윈도우에서 파이썬을 사용할 때 알아야 할 윈도우 특정 동작에 대한 개요를 제공하는 것을 목표로 합니다.

Unlike most Unix systems and services, Windows does not include a system supported installation of Python. To make Python available, the CPython team has compiled Windows installers with every release for many years. These installers are primarily intended to add a per-user installation of Python, with the core interpreter and library being used by a single user. The installer is also able to install for all users of a single machine, and a separate ZIP file is available for application-local distributions.

As specified in **PEP 11**, a Python release only supports a Windows platform while Microsoft considers the platform under extended support. This means that Python 3.14 supports Windows 10 and newer. If you require Windows 7 support, please install Python 3.8. If you require Windows 8.1 support, please install Python 3.12.

윈도우에서 사용할 수 있는 여러 가지 설치 프로그램이 있으며, 각각 나름의 장단점이 있습니다.

전체 설치 프로그램은 모든 구성 요소를 포함하며 모든 종류의 프로젝트에 파이썬을 사용하는 개발자에게 가장 적합한 옵션입니다.

Microsoft Store 패키지 is a simple installation of Python that is suitable for running scripts and packages, and using IDLE or other development environments. It requires Windows 10 and above, but can be safely installed without corrupting other programs. It also provides many convenient commands for launching Python and its tools.

nuget.org 패키지는 지속적 통합 시스템(continuous integration systems)을 위한 경량 설치입니다. 파이썬 패키지를 빌드하거나 스크립트를 실행하는 데 사용할 수 있지만, 업데이트할 수 없으며 사용자 인터페이스 도구가 없습니다.

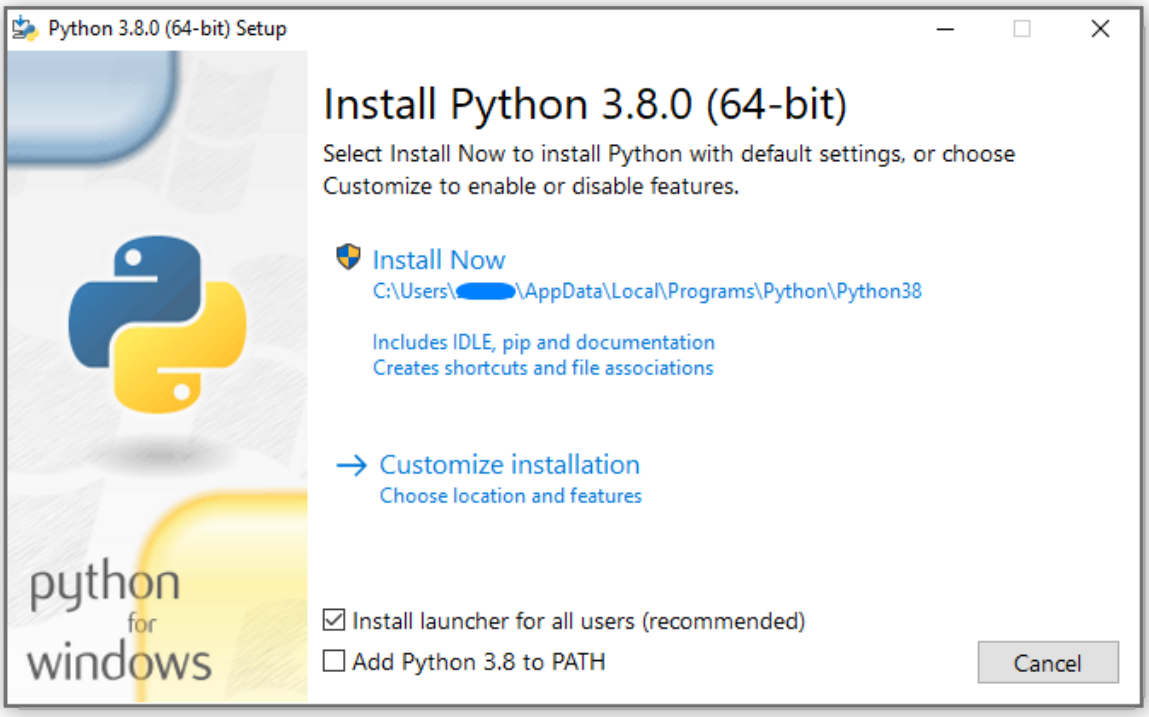
내장 가능한 패키지는 더 큰 응용 프로그램에 내장하기에 적합한 파이썬의 최소 패키지입니다.

4.1 전체 설치 프로그램

4.1.1 설치 단계

네 개의 파이썬 3.14 설치 프로그램을 다운로드할 수 있습니다 - 각각 32비트와 64비트 버전의 인터프리터를 지원하는 두 가지. 웹 설치 프로그램(*web installer*)은 작은 초기 다운로드이며, 필요한 구성 요소를 자동으로 다운로드합니다. 오프라인 설치 프로그램(*offline installer*)에는 기본 설치에 필요한 구성 요소가 포함되어 있으며 선택적 기능을 위해서만 인터넷 연결이 필요합니다. 설치 중 다운로드를 피하는 다른 방법은 다운로드 없이 설치하기를 참조하십시오.

설치 프로그램을 시작한 후, 두 가지 옵션 중 하나를 선택할 수 있습니다:



"Install Now"를 선택하면:

- 관리자(administrator)일 필요는 없습니다 (C 런타임 라이브러리에 대한 시스템 업데이트가 필요하거나 모든 사용자를 위해 윈도우 용 파이썬 런처를 설치하지 않는 한)
- 파이썬이 사용자 디렉터리에 설치됩니다
- 윈도우 용 파이썬 런처는 첫 페이지 하단의 옵션에 따라 설치됩니다
- 표준 라이브러리, 테스트 스위트, 런처 및 pip가 설치됩니다
- 선택하면, 설치 디렉터리가 `PATH`에 추가됩니다
- 바로 가기는 현재 사용자에게만 표시됩니다

"Customize installation"을 선택하면 설치할 기능, 설치 위치 및 다른 옵션이나 설치 후 작업을 선택할 수 있습니다. 디버깅 심볼이나 바이너리를 설치하려면, 이 옵션을 사용해야 합니다.

모든 사용자 설치를 수행하려면, "Customize installation"을 선택해야 합니다. 이 경우:

- 관리자 자격 증명이나 승인을 제공해야 할 수 있습니다.
- 파이썬은 Program Files 디렉터리에 설치됩니다
- 윈도우 용 파이썬 런처는 Windows 디렉터리에 설치됩니다
- 설치 중에 선택적 기능을 선택할 수 있습니다
- 표준 라이브러리는 바이트 코드로 사전 컴파일될 수 있습니다
- 선택하면, 설치 디렉터리가 시스템 `PATH`에 추가됩니다
- 모든 사용자가 바로 가기를 사용할 수 있습니다

4.1.2 MAX_PATH 제한 제거하기

윈도우는 역사적으로 경로 길이를 260자로 제한했습니다. 이는 이보다 긴 경로는 결정(resolve)되지 않고 에러가 발생함을 의미합니다.

최신 버전의 윈도우에서는, 이 제한을 약 32,000자로 확장할 수 있습니다. 관리자는 "Enable Win32 long paths" 그룹 정책을 활성화하거나, 레지스트리 키 `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem`에서 `LongPathsEnabled`를 `1`로 설정해야 합니다.

이는 `open()` 함수, `os` 모듈 및 대부분의 다른 경로 기능이 260자보다 긴 경로를 받아들이고 반환할 수 있도록 합니다.

위 옵션을 변경한 후에는, 추가 구성이 필요하지 않습니다.

버전 3.6에서 변경: 긴 경로에 대한 지원이 파이썬에서 활성화되었습니다.

4.1.3 UI 없이 설치하기

설치 프로그램 UI에서 사용할 수 있는 모든 옵션은 명령 줄에서도 지정할 수 있어서, 스크립팅 된 설치 프로그램이 사용자 상호 작용 없이 여러 컴퓨터에서 설치를 복제할 수 있도록 합니다. 이러한 옵션은 일부 기본값을 변경하기 위해 UI를 억제하지 않고 설정할 수도 있습니다.

The following options (found by executing the installer with `/?`) can be passed into the installer:

이름	설명
/passive	to display progress without requiring user interaction
/quiet	to install/uninstall without displaying any UI
/simple	to prevent user customization
/uninstall	to remove Python (without confirmation)
/layout [directory]	to pre-download all components
/log [filename]	to specify log files location

다른 모든 옵션은 `name=value`로 전달됩니다. 여기서 value는 일반적으로 기능을 비활성화하려면 `0`, 기능을 활성화하려면 `1` 또는 경로입니다. 사용 가능한 옵션의 전체 목록은 다음과 같습니다.

이름	설명	기본값
InstallAllUsers	시스템 전체 설치를 수행합니다.	0
TargetDir	설치 디렉터리	InstallAllUsers 에 따라 선택됩니다
DefaultAllUsersTargetDir	모든 사용자 설치를 위한 기본 설치 디렉터리	`%ProgramFiles%\Python X.Y` 또는 `%ProgramFiles(x86)%\Python X.Y`
DefaultJustForMeTargetDir	현재 사용자 전용 설치를 위한 기본 설치 디렉터리	`%LocalAppData%\Programs\Python\PythonXY` or `%LocalAppData%\Programs\Python\PythonXY-32` or `%LocalAppData%\Programs\Python\PythonXY-64`
DefaultCustomTargetDir	UI에 표시되는 기본 사용자 지정 설치 디렉터리	(비어있음)
AssociateFiles	런처도 설치되었으면 파일 연결을 만듭니다.	1
CompileAll	모든 `.py` 파일을 `.pyc`로 컴파일합니다.	0
PrependPath	Prepend install and Scripts directories to `PATH` and add `.PY` to `PATHEXT`	0
AppendPath	Append install and Scripts directories to `PATH` and add `.PY` to `PATHEXT`	0
Shortcuts	설치되면, 인터프리터, 설명서 및 IDLE에 대한 바로 가기를 만듭니다.	1
Include_doc	파이썬 매뉴얼을 설치합니다	1
Include_debu	디버그 바이너리를 설치합니다	0
Include_dev	Install developer headers and libraries. Omitting this may lead to an unusable installation.	1
Include_exe	Install `python.exe` and related files. Omitting this may lead to an unusable installation.	1
Include_launc	윈도우 용 파이썬 런처를 설치합니다.	1
InstallLauncherAllUsers	Installs the launcher for all users. Also requires `Include_launcher` to be set to 1	1
Include_lib	Install standard library and extension modules. Omitting this may lead to an unusable installation.	1
Include_pip	번들로 제공되는 pip와 setuptools를 설치합니다	1
Include_symb	Install debugging symbols (`*.pdb`)	0
Include_tcltk	Tcl/Tk 지원과 IDLE을 설치합니다	1
Include_test	표준 라이브러리 테스트 스위트를 설치합니다	1
Include_tools	유틸리티 스크립트를 설치합니다	1
LauncherO	런처만 설치합니다. 이것은 대부분의 다른 옵션보다 우선합니다.	0
SimpleInstall	대부분의 설치 UI를 비활성화합니다	0

예를 들어 기본, 시스템 전체 파이썬 설치를 조용히 설치하려면, (관리자 권한 명령 프롬프트에서) 다음 명령을 사용할 수 있습니다:

```
python-3.9.0.exe /quiet InstallAllUsers=1 PrependPath=1 Include_test=0
```

사용자가 테스트 스위트 없이 파이썬의 개인용 사본을 쉽게 설치할 수 있도록, 다음 명령으로 바로 가기를 제공할 수 있습니다. 이렇게 하면 단순화된 초기 페이지가 표시되고 사용자 정의가 허용되지 않습니다:

```
python-3.9.0.exe InstallAllUsers=0 Include_launcher=0 Include_test=0
    SimpleInstall=1 SimpleInstallDescription="Just for me, no test suite."
```

(런처를 생략하면 파일 연결도 생략되며, 런처가 포함된 시스템 전체 설치가 있을 때, 사용자별 설치에만 권장됨에 유의하십시오.)

위에 나열된 옵션은 실행 파일과 함께 `unattend.xml`이라는 파일로 제공될 수도 있습니다. 이 파일은 옵션과 값 목록을 지정합니다. 값이 어트리뷰트로 제공되면, 가능한 경우 숫자로 변환됩니다. 엘리먼트 텍스트로 제공된 값은 항상 문자열로 남아 있습니다. 이 예제 파일은 이전 예제와 같은 옵션을 설정합니다:

```
<Options>
    <Option Name="InstallAllUsers" Value="no" />
    <Option Name="Include_launcher" Value="0" />
    <Option Name="Include_test" Value="no" />
    <Option Name="SimpleInstall" Value="yes" />
    <Option Name="SimpleInstallDescription">Just for me, no test suite</Option>
</Options>
```

4.1.4 다운로드 없이 설치하기

파이썬의 일부 기능은 초기 설치 프로그램 다운로드에 포함되어 있지 않아서, 이러한 기능을 선택하면 인터넷 연결이 필요할 수 있습니다. 이러한 요구를 피하고자, 필요에 따라 모든 가능한 구성 요소를 내려받아 선택한 기능과 관계없이 더는 인터넷 연결이 필요하지 않은 완전한 레이아웃을 만들 수 있습니다. 이 다운로드는 필요한 것보다 클 수 있지만, 많은 수의 설치를 수행할 경우 로컬로 캐시 된 사본을 보유하는 것이 매우 유용합니다.

가능한 모든 파일을 다운로드하려면 명령 프롬프트에서 다음 명령을 실행하십시오. `python-3.9.0.exe`를 설치 프로그램의 실제 이름으로 대체하고, 같은 이름을 가진 파일 간의 충돌을 방지하기 위해 자체 디렉터리에 레이아웃을 만들어야 합니다.

```
python-3.9.0.exe /layout [optional target directory]
```

`/quiet` 옵션을 지정하여 진행률 표시를 숨길 수도 있습니다.

4.1.5 설치 수정하기

일단 파이썬이 설치되면, 윈도우의 일부인 Programs and Features 도구를 통해 기능을 추가하거나 제거할 수 있습니다. Python 항목을 선택하고 "Uninstall/Change"를 선택하여 유지 관리 모드로 설치 프로그램을 엽니다.

"Modify"는 체크 박스를 수정하여 기능을 추가하거나 제거하도록 합니다 - 변경되지 않은 체크 박스는 아무것도 설치하거나 제거하지 않습니다. 이 모드에서는 설치 디렉터리와 같은 일부 옵션을 변경할 수 없습니다; 이를 수정하려면 파이썬을 완전히 제거한 다음 다시 설치해야 합니다.

"Repair"는 현재 설정을 사용하여 설치되어야 하는 모든 파일을 확인하고 제거되거나 수정된 파일을 대체합니다.

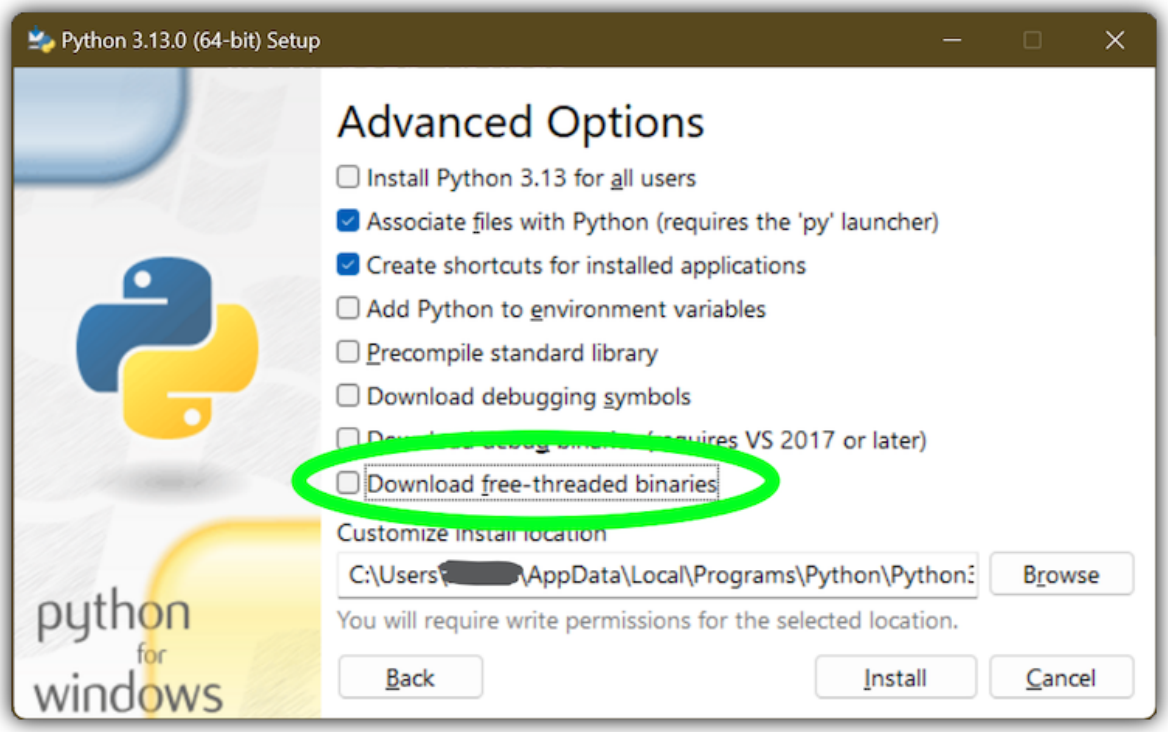
"Uninstall"은 파이썬을 완전히 제거하는데, Programs and Features에 자체 항목이 있는 윈도우 용 파이썬 런처는 제외합니다.

4.1.6 Installing Free-threaded Binaries

Added in version 3.13: (Experimental)

> **참고**
>
> Everything described in this section is considered experimental, and should be expected to change in future releases.

To install pre-built binaries with free-threading enabled (see **PEP 703**), you should select “Customize installation”. The second page of options includes the “Download free-threaded binaries” checkbox.



Selecting this option will download and install additional binaries to the same location as the main Python install. The main executable is called `python3.13t.exe`, and other binaries either receive a `t` suffix or a full ABI suffix. Python source files and bundled third-party dependencies are shared with the main install.

The free-threaded version is registered as a regular Python install with the tag `3.13t` (with a `-32` or `-arm64` suffix as normal for those platforms). This allows tools to discover it, and for the 윈도우 용 파이썬 런처 to support `py.exe -3.13t`. Note that the launcher will interpret `py.exe -3` (or a `python3` shebang) as “the latest 3.x install”, which will prefer the free-threaded binaries over the regular ones, while `py.exe -3.13` will not. If you use the short style of option, you may prefer to not install the free-threaded binaries at this time.

To specify the install option at the command line, use `Include_freethreaded=1`. See 다운로드 없이 설치하기 for instructions on pre-emptively downloading the additional binaries for offline install. The options to include debug symbols and binaries also apply to the free-threaded builds.

Free-threaded binaries are also available *on nuget.org*.

4.2 Microsoft Store 패키지

Added in version 3.7.2.

Microsoft Store 패키지는 쉽게 설치할 수 있는 파이썬 인터프리터로, 주로 예를 들어 학생의 대화형 사용을 목적으로 합니다.

패키지를 설치하려면, 최신 윈도우 10 업데이트가 있는지 확인하고 "Python 3.14"를(을) 위한 Microsoft Store 앱을 검색하십시오. 선택한 앱이 Python Software Foundation에서 게시되었는지 확인하고, 설치합니다.

경고

파이썬은 Microsoft Store에서 항상 무료로 제공됩니다. 비용을 지불하라는 요청을 받으면, 올바른 패키지를 선택하지 않은 것입니다.

설치 후, 파이썬은 Start에서 찾아서 시작할 수 있습니다. 또는, 모든 명령 프롬프트 또는 PowerShell 세션에서 `python`을 입력하여 사용할 수 있습니다. 또한, `pip`나 `idle`을 입력하여 pip와 IDLE을 사용할 수 있습니다. IDLE은 Start에서도 찾을 수 있습니다.

세 가지 명령 모두 버전 번호 접미사와 함께 사용할 수 있습니다, 예를 들어, `python.exe`뿐만 아니라 `python3.exe`와 `python3.x.exe` (여기서 `3.x`는 여러분이 시작하려는 특정 버전입니다, 가령 3.14). Start를 통해 "Manage App Execution Aliases"를 열어 각 명령과 연결된 파이썬 버전을 선택합니다. `pip`와 `idle`이 선택된 `python` 버전과 일치하도록 하는 것이 좋습니다.

`python -m venv`로 가상 환경을 만들고 활성화하여 정상적으로 사용할 수 있습니다.

다른 버전의 파이썬을 설치하고 `PATH` 변수에 추가했으면, Microsoft Store에서 제공하는 것이 아니라 그것이 `python.exe`로 사용될 수 있습니다. 새 설치에 액세스하려면, `python3.exe`나 `python3.x.exe`를 사용하십시오.

`py.exe` 런처는 이 파이썬 설치를 감지하지만, 기존 설치 프로그램으로부터의 설치를 선호합니다.

파이썬을 제거하려면, Settings를 열고 Apps and Features를 사용하거나, Start에서 Python을 찾은 다음 마우스 오른쪽 단추를 클릭하여 Uninstall 을 선택합니다. 제거하면 이 파이썬 설치에 직접 설치한 모든 패키지가 제거되지만, 가상 환경은 제거되지 않습니다.

4.2.1 Known issues

Redirection of local data, registry, and temporary paths

Because of restrictions on Microsoft Store apps, Python scripts may not have full write access to shared locations such as `TEMP` and the registry. Instead, it will write to a private copy. If your scripts must modify the shared locations, you will need to install the full installer.

At runtime, Python will use a private copy of well-known Windows folders and the registry. For example, if the environment variable `%APPDATA%` is `c:\Users\<user>\AppData\`, then when writing to `C:\Users\<user>\AppData\Local` will write to `C:\Users\<user>\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.8_qbz5n2kfra8p0\LocalCache\Local\`.

When reading files, Windows will return the file from the private folder, or if that does not exist, the real Windows directory. For example reading `C:\Windows\System32` returns the contents of `C:\Windows\System32` plus the contents of `C:\Program Files\WindowsApps\package_name\VFS\SystemX86`.

You can find the real path of any existing file using `os.path.realpath()`:

```
>>> import os
>>> test_file = 'C:\\Users\\example\\AppData\\Local\\test.txt'
>>> os.path.realpath(test_file)
'C:\\Users\\example\\AppData\\Local\\Packages\\PythonSoftwareFoundation.Python.3.8_
↪qbz5n2kfra8p0\\LocalCache\\Local\\test.txt'
```

When writing to the Windows Registry, the following behaviors exist:

- Reading from `HKLM\\Software` is allowed and results are merged with the `registry.dat` file in the package.
- Writing to `HKLM\\Software` is not allowed if the corresponding key/value exists, i.e. modifying existing keys.

- Writing to `HKLM\\Software` is allowed as long as a corresponding key/value does not exist in the package and the user has the correct access permissions.

For more detail on the technical basis for these limitations, please consult Microsoft' s documentation on packaged full-trust apps, currently available at docs.microsoft.com/en-us/windows/msix/desktop/desktop-to-uwp-behind-the-scenes

4.3 nuget.org 패키지

Added in version 3.5.2.

nuget.org 패키지는 시스템 전체에 파이썬이 설치되지 않은 지속적인 통합과 빌드 시스템에 사용하기 위한 축소된 크기의 파이썬 환경입니다. 너겟은 "NET 용 패키지 관리자" 이지만, 빌드 타임 도구가 포함된 패키지에서도 완벽하게 작동합니다.

너겟 사용에 대한 최신 정보를 보려면 nuget.org를 방문하십시오. 다음은 파이썬 개발자에게 충분한 요약입니다.

`nuget.exe` 명령 줄 도구는, 예를 들어 curl이나 PowerShell을 사용하여 `https://aka.ms/nugetclidl`에서 직접 다운로드할 수 있습니다. 이 도구를 사용하면 다음과 같이 64비트나 32비트 컴퓨터용 파이썬의 최신 버전이 설치됩니다:

```
nuget.exe install python -ExcludeVersion -OutputDirectory .
nuget.exe install pythonx86 -ExcludeVersion -OutputDirectory .
```

To select a particular version, add a `-Version 3.x.y`. The output directory may be changed from `.`, and the package will be installed into a subdirectory. By default, the subdirectory is named the same as the package, and without the `-ExcludeVersion` option this name will include the specific version installed. Inside the subdirectory is a `tools` directory that contains the Python installation:

```
# Without -ExcludeVersion
> .\python.3.5.2\tools\python.exe -V
Python 3.5.2

# With -ExcludeVersion
> .\python\tools\python.exe -V
Python 3.5.2
```

일반적으로, 너겟 패키지는 업그레이드할 수 없으며, 최신 버전을 나란히 설치하고 전체 경로를 사용하여 참조해야 합니다. 또는, 패키지 디렉터리를 수동으로 삭제하고 다시 설치하십시오. 많은 CI 시스템은 빌드 간에 파일을 보존하지 않으면 이 작업을 자동으로 수행합니다.

`tools` 디렉터리와 함께 `build\native` 디렉터리가 있습니다. 여기에는 파이썬 설치를 참조하기 위해 C++ 프로젝트에서 사용할 수 있는 MSBuild 속성 파일 `python.props`가 포함되어 있습니다. 설정을 포함하면 빌드에서 자동으로 헤더와 임포트 라이브러리를 사용합니다.

The package information pages on nuget.org are www.nuget.org/packages/python for the 64-bit version, www.nuget.org/packages/pythonx86 for the 32-bit version, and www.nuget.org/packages/pythonarm64 for the ARM64 version

4.3.1 Free-threaded packages

Added in version 3.13: (Experimental)

> **참고**
>
> Everything described in this section is considered experimental, and should be expected to change in future releases.

Packages containing free-threaded binaries are named python-freethreaded for the 64-bit version, pythonx86-freethreaded for the 32-bit version, and pythonarm64-freethreaded for the ARM64 version. These packages contain both the `python3.13t.exe` and `python.exe` entry points, both of which run free threaded.

4.4 내장 가능한 패키지

Added in version 3.5.

내장된 배포는 최소 파이썬 환경을 포함하는 ZIP 파일입니다. 최종 사용자가 직접 액세스하기보다는, 다른 응용 프로그램의 일부로 작동하기 위한 것입니다.

When extracted, the embedded distribution is (almost) fully isolated from the user' s system, including environment variables, system registry settings, and installed packages. The standard library is included as pre-compiled and optimized `.pyc` files in a ZIP, and `python3.dll`, `python37.dll`, `python.exe` and `pythonw.exe` are all provided. Tcl/tk (including all dependents, such as Idle), pip and the Python documentation are not included.

> **참고**
>
> The embedded distribution does not include the Microsoft C Runtime and it is the responsibility of the application installer to provide this. The runtime may have already been installed on a user' s system previously or automatically via Windows Update, and can be detected by finding `ucrtbase.dll` in the system directory.

제삼자 패키지는 내장된 배포와 함께 응용 프로그램 설치 프로그램이 설치해야 합니다. 일반 파이썬 설치처럼 종속성을 관리하기 위해 pip를 사용하는 것은 이 배포에서 지원되지 않지만, 주의를 기울이면 자동 업데이트를 위해 pip를 포함하고 사용할 수 있습니다. 일반적으로, 제삼자 패키지는 개발자가 사용자에게 업데이트를 제공하기 전에 최신 버전과의 호환성을 보장할 수 있도록 응용 프로그램의 일부로 처리되어야 합니다 ("벤더링(vendoring)").

이 배포에 권장되는 두 가지 사용 사례가 아래에 설명되어 있습니다.

4.4.1 파이썬 응용 프로그램

파이썬으로 작성된 응용 프로그램이 반드시 사용자가 그 사실을 인식하도록 할 필요는 없습니다. 이 경우 내장된 배포를 사용하여 설치 패키지에 파이썬의 내부 버전을 포함할 수 있습니다. 얼마나 투명해야 하는지 (또는 반대로, 얼마나 전문적으로 보여야 하는지)에 따라, 두 가지 옵션이 있습니다.

특수 실행 파일을 런처로 사용하려면 약간의 코딩이 필요하지만, 사용자에게 가장 투명한 경험을 제공합니다. 사용자 정의된 런처를 사용하면, 프로그램이 파이썬에서 실행되고 있다는 명백한 표시가 없습니다: 아이콘을 사용자 정의하고, 회사와 버전 정보를 지정할 수 있으며 파일 연결이 제대로 작동합니다. 대부분의 경우, 사용자 정의 런처는 하드 코딩된 명령 줄을 사용하여 `Py_Main`을 호출할 수 있어야 합니다.

더 간단한 방법은 필요한 명령 줄 인자를 사용하여 `python.exe`나 `pythonw.exe`를 직접 호출하는 배치 파일이나 생성된 바로 가기를 제공하는 것입니다. 이 경우, 응용 프로그램은 실제 이름이 아닌 파이썬으로 표시되며, 사용자는 실행 중인 다른 파이썬 프로세스나 파일 연결과 구별하는 데 어려움을 겪을 수 있습니다.

후자의 접근 방식에서는, 패키지를 파이썬 실행 파일과 함께 디렉터리로 설치하여 경로에서 사용할 수 있도록 해야 합니다. 특수 런처를 사용하면, 응용 프로그램을 시작하기 전에 검색 경로를 지정할 수 있어서 패키지를 다른 위치에 배치할 수 있습니다.

4.4.2 파이썬 내장하기

네이티브 코드로 작성된 응용 프로그램에는 종종 어떤 형태의 스크립팅 언어가 필요하며, 내장된 파이썬 배포를 이러한 목적으로 사용할 수 있습니다. 일반적으로, 대부분의 응용 프로그램은 네이티브 코드로 되어 있으며, 일부가 `python.exe`를 호출하거나 `python3.dll`을 직접 사용합니다. 두 경우 모두, 내장된 배포를 응용 프로그램 설치의 하위 디렉터리로 추출하면 로드할 수 있는 파이썬 인터프리터를 제공하기에 충분합니다.

응용 프로그램 사용과 마찬가지로, 인터프리터를 초기화하기 전에 검색 경로를 지정할 기회가 있어서 패키지를 임의의 위치에 설치할 수 있습니다. 그 외에는, 내장된 배포와 일반 설치를 사용하는 것 간에 근본적인 차이점은 없습니다.

4.5 대체 번들

표준 CPython 배포 외에도, 추가 기능을 포함하는 수정된 패키지가 있습니다. 다음은 많이 사용되는 버전과 주요 기능 목록입니다:

ActivePython
: 다중 플랫폼 호환성, 설명서, PyWin32가 있는 설치 프로그램

Anaconda
: 인기 있는 과학 모듈(가령 numpy, scipy 및 pandas)과 `conda` 패키지 관리자.

Enthought Deployment Manager
: "The Next Generation Python Environment and Package Manager".

  Previously Enthought provided Canopy, but it reached end of life in 2016.

WinPython
: 사전 빌드된 과학 패키지와 패키지 빌드를 위한 도구가 포함된 윈도우 전용 배포.

이러한 패키지들은 최신 버전의 파이썬이나 기타 라이브러리를 포함하지 않을 수 있으며, 핵심 파이썬 팀에서 유지 관리하거나 지원하지 않음에 유의하십시오.

4.6 파이썬 구성하기

명령 프롬프트에서 파이썬을 편리하게 실행하려면, 윈도우에서 일부 기본 환경 변수를 변경하는 것을 고려할 수 있습니다. 설치 프로그램이 PATH와 PATHEXT 변수를 구성하는 옵션을 제공하지만, 이는 시스템 전체의 단일 설치에서만 신뢰할 수 있습니다. 여러 버전의 파이썬을 정기적으로 사용하면, 윈도우 용 파이썬 런처 사용을 고려하십시오.

4.6.1 보충 설명: 환경 변수 설정하기

윈도우에서는 환경 변수를 사용자 수준과 시스템 수준 모두에서 영구적으로 구성하거나, 명령 프롬프트에서 일시적으로 구성할 수 있습니다.

환경 변수를 임시로 설정하려면, 명령 프롬프트를 열고 **set** 명령을 사용하십시오:

```
C:\>set PATH=C:\Program Files\Python 3.9;%PATH%
C:\>set PYTHONPATH=%PYTHONPATH%;C:\My_python_lib
C:\>python
```

이러한 변경은 해당 콘솔에서 실행되는 모든 추가 명령에 적용되며, 콘솔에서 시작된 모든 응용 프로그램에 상속됩니다.

백분율 기호 안에 변수 이름을 포함하면 기존 값으로 확장되어, 시작이나 끝부분에 새 값을 추가할 수 있습니다. **python.exe**를 포함하는 디렉터리를 시작 부분에 추가하여 `PATH`를 수정하는 것은 올바른 버전의 파이썬이 실행되도록 하는 일반적인 방법입니다.

기본 환경 변수를 영구적으로 수정하려면, Start를 클릭하고 'edit environment variables' 를 검색하거나, 시스템 속성, *Advanced system settings*를 열고 *Environment Variables* 버튼을 클릭합니다. 이 대화 상자에서, 사용자와 시스템 변수를 추가하거나 수정할 수 있습니다. 시스템 변수를 변경하려면 컴퓨터에 제한 없이 액세스해야 합니다 (즉 관리자 권한).

> **참고**
>
> 윈도우는 사용자 변수를 시스템 변수 뒤에 이어붙이므로, `PATH`를 수정할 때 예기치 않은 결과가 발생할 수 있습니다.

The *PYTHONPATH* variable is used by all versions of Python, so you should not permanently configure it unless the listed paths only include code that is compatible with all of your installed Python versions.

더 보기

https://learn.microsoft.com/windows/win32/procthread/environment-variables
Overview of environment variables on Windows

https://learn.microsoft.com/windows-server/administration/windows-commands/set_1
The `set` command, for temporarily modifying environment variables

https://learn.microsoft.com/windows-server/administration/windows-commands/setx
The `setx` command, for permanently modifying environment variables

4.6.2 파이썬 실행 파일 찾기

버전 3.5에서 변경.

자동으로 만들어진 파이썬 인터프리터를 위한 시작 메뉴 항목을 사용하는 것 외에도, 명령 프롬프트에서 파이썬을 시작하고 싶을 수 있습니다. 설치 프로그램에는 이를 설정할 수 있는 옵션이 있습니다.

설치 프로그램의 첫 페이지에서, "Add Python to PATH" 라는 옵션을 선택하여 설치 프로그램이 `PATH`에 설치 위치를 추가하도록 할 수 있습니다. `Scripts\` 폴더의 위치도 추가됩니다. 이는 인터프리터를 실행하기 위해 **python**을 입력하고, 패키지 설치 프로그램을 실행하려면 **pip**를 입력할 수 있도록 합니다. 따라서, 명령 줄 옵션으로 스크립트를 실행할 수도 있습니다, 명령 줄 설명서를 참조하십시오.

설치 시 이 옵션을 활성화하지 않았으면, 언제든지 설치 프로그램을 다시 실행하고, Modify를 선택한 다음, 활성화할 수 있습니다. 또는, 보충 설명: 환경 변수 설정하기의 지침을 사용하여 `PATH`를 수동으로 수정할 수 있습니다. 파이썬 설치 디렉터리를 포함하도록 `PATH` 환경 변수를 설정해야 하며, 다른 항목과 세미콜론으로 구분됩니다. 예제 변수는 다음과 같습니다 (처음 두 항목이 이미 존재한다고 가정합니다):

```
C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files\Python 3.9
```

4.7 UTF-8 모드

Added in version 3.7.

Windows still uses legacy encodings for the system encoding (the ANSI Code Page). Python uses it for the default encoding of text files (e.g. `locale.getencoding()`).

UTF-8은 인터넷과 WSL(Windows Subsystem for Linux)을 포함한 대부분의 유닉스 시스템에서 널리 사용되기 때문에 문제가 발생할 수 있습니다.

You can use the Python UTF-8 Mode to change the default text encoding to UTF-8. You can enable the Python UTF-8 Mode via the `-X utf8` command line option, or the `PYTHONUTF8=1` environment variable. See *PYTHONUTF8* for enabling UTF-8 mode, and 보충 설명: 환경 변수 설정하기 for how to modify environment variables.

When the Python UTF-8 Mode is enabled, you can still use the system encoding (the ANSI Code Page) via the "mbcs" codec.

기본 환경 변수에 `PYTHONUTF8=1`을 추가하면 시스템의 모든 파이썬 3.7+ 응용 프로그램에 영향을 줍니다. 레거시 시스템 인코딩에 의존하는 파이썬 3.7+ 응용 프로그램이 있으면 환경 변수를 임시로 설정하거나 `-X utf8` 명령 줄 옵션을 사용하는 것이 좋습니다.

참고

UTF-8 모드가 비활성화된 경우에도, 파이썬은 윈도우에서 기본적으로 다음을 위해 UTF-8을 사용합니다:

- 표준 I/O를 포함한 콘솔 I/O (자세한 내용은 **PEP 528**을 참조하십시오).
- The *filesystem encoding* (see **PEP 529** for details).

4.8 윈도우 용 파이썬 런처

Added in version 3.3.

윈도우 용 파이썬 런처는 다양한 파이썬 버전을 찾고 실행하는 데 도움이 되는 유틸리티입니다. 스크립트 (또는 명령 줄)가 특정 파이썬 버전에 대한 선호를 나타내도록 허용하고, 해당 버전을 찾아 실행합니다.

`PATH` 변수와 달리, 런처는 가장 적합한 파이썬 버전을 올바르게 선택합니다. 시스템 전체 설치보다 사용자별 설치를 선호하며, 가장 최근에 설치된 버전을 사용하기보다 언어 버전별로 순서를 매깁니다.

런처는 원래 **PEP 397**에서 지정되었습니다.

4.8.1 시작하기

명령 줄에서

버전 3.6에서 변경.

System-wide installations of Python 3.3 and later will put the launcher on your `PATH`. The launcher is compatible with all available versions of Python, so it does not matter which version is installed. To check that the launcher is available, execute the following command in Command Prompt:

```
py
```

설치한 최신 버전의 파이썬이 시작되어야 합니다 - 정상적으로 종료할 수 있으며, 지정된 추가 명령 줄 인자가 파이썬으로 직접 전송됩니다.

If you have multiple versions of Python installed (e.g., 3.7 and 3.14) you will have noticed that Python 3.14 was started - to launch Python 3.7, try the command:

```
py -3.7
```

If you want the latest version of Python 2 you have installed, try the command:

```
py -2
```

If you see the following error, you do not have the launcher installed:

```
'py' is not recognized as an internal or external command,
operable program or batch file.
```

The command:

```
py --list
```

displays the currently installed version(s) of Python.

The `-x.y` argument is the short form of the `-V:Company/Tag` argument, which allows selecting a specific Python runtime, including those that may have come from somewhere other than python.org. Any runtime registered by following **PEP 514** will be discoverable. The `--list` command lists all available runtimes using the `-V:` format.

When using the `-V:` argument, specifying the Company will limit selection to runtimes from that provider, while specifying only the Tag will select from all providers. Note that omitting the slash implies a tag:

```
# Select any '3.*' tagged runtime
py -V:3
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
# Select any 'PythonCore' released runtime
py -V:PythonCore/

# Select PythonCore's latest Python 3 runtime
py -V:PythonCore/3
```

The short form of the argument (`-3`) only ever selects from core Python releases, and not other distributions. However, the longer form (`-V:3`) will select from any.

The Company is matched on the full string, case-insensitive. The Tag is matched on either the full string, or a prefix, provided the next character is a dot or a hyphen. This allows `-V:3.1` to match `3.1-32`, but not `3.10`. Tags are sorted using numerical ordering (`3.10` is newer than `3.1`), but are compared using text (`-V:3.01` does not match `3.1`).

가상 환경

Added in version 3.5.

런처가 명시적인 파이썬 버전 지정 없이 실행되고, 가상 환경(표준 라이브러리 `venv` 모듈이나 외부 `virtualenv` 도구로 생성된)이 활성화되었으면, 런처는 전역 인터프리터가 아닌 가상 환경의 인터프리터를 실행합니다. 전역 인터프리터를 실행하려면 가상 환경을 비활성화하거나, 전역 파이썬 버전을 명시적으로 지정하십시오.

스크립트에서

테스트 파이썬 스크립트를 만들어 봅시다 - 다음 내용으로 `hello.py`라는 파일을 만듭니다.

```
#! python
import sys
sys.stdout.write("hello from Python %s\n" % (sys.version,))
```

From the directory in which hello.py lives, execute the command:

```
py hello.py
```

최신 파이썬 2.x 설치의 버전 번호가 인쇄됨을 알 수 있습니다. 이제 첫 번째 줄을 다음과 같이 변경합니다:

```
#! python3
```

Re-executing the command should now print the latest Python 3.x information. As with the above command-line examples, you can specify a more explicit version qualifier. Assuming you have Python 3.7 installed, try changing the first line to `#! python3.7` and you should find the 3.7 version information printed.

대화 형 사용과 달리, 장식 없는 "python"은 설치된 파이썬 2.x의 최신 버전을 사용합니다. 이는 이전 버전과의 호환성과 유닉스와의 호환성을 위한 것입니다, 여기서 `python` 명령은 일반적으로 파이썬 2를 참조합니다.

파일 연결에서

런처는 설치 시 파이썬 파일(즉 `.py`, `.pyw`, `.pyc` 파일)과 연결되어 있어야 합니다. 즉, 윈도우 탐색기에서 이러한 파일 중 하나를 더블 클릭하면 런처가 사용되므로, 위에서 설명한 것과 같은 기능을 사용하여 스크립트에서 사용해야 하는 버전을 지정할 수 있습니다.

이것의 주요 이점은 첫 번째 줄의 내용에 따라 단일 런처가 동시에 여러 파이썬 버전을 지원할 수 있다는 것입니다.

4.8.2 셔뱅 줄

스크립트 파일의 첫 번째 줄이 `#!`로 시작하면 "셔뱅(shebang)" 줄이라고 합니다. 리눅스와 기타 유닉스 류 운영 체제는 이러한 줄을 기본적으로 지원하며 스크립트 실행 방법을 나타내기 위해 이러한 시스템에서 일반적으로 사용됩니다. 이 런처를 사용하면 윈도우에서 파이썬 스크립트로 같은 기능을 사용할 수 있으며 위의 예제는 그 사용법을 보여줍니다.

파이썬 스크립트의 셔뱅 줄을 유닉스와 윈도우 간에 이식성 있도록 하기 위해, 이 런처는 사용할 인터프리터를 지정하는 여러 '가상' 명령을 지원합니다. 지원되는 가상 명령은 다음과 같습니다:

- `/usr/bin/env`
- `/usr/bin/python`
- `/usr/local/bin/python`
- `python`

예를 들어, 스크립트의 첫 번째 줄이 다음과 같이 시작하면

```
#! /usr/bin/python
```

The default Python or an active virtual environment will be located and used. As many Python scripts written to work on Unix will already have this line, you should find these scripts can be used by the launcher without modification. If you are writing a new script on Windows which you hope will be useful on Unix, you should use one of the shebang lines starting with `/usr`.

Any of the above virtual commands can be suffixed with an explicit version (either just the major version, or the major and minor version). Furthermore the 32-bit version can be requested by adding "-32" after the minor version. I.e. `/usr/bin/python3.7-32` will request usage of the 32-bit Python 3.7. If a virtual environment is active, the version will be ignored and the environment will be used.

Added in version 3.7: 파이썬 런처 3.7부터는 "-64" 접미사로 64비트 버전을 요청할 수 있습니다. 또한 부 버전 없이 주 버전과 아키텍처를 지정할 수 있습니다 (즉 `/usr/bin/python3-64`).

버전 3.11에서 변경: The "-64" suffix is deprecated, and now implies "any architecture that is not provably i386/32-bit". To request a specific environment, use the new `-V:`*`TAG`* argument with the complete tag.

버전 3.13에서 변경: Virtual commands referencing `python` now prefer an active virtual environment rather than searching `PATH`. This handles cases where the shebang specifies `/usr/bin/env python3` but `python3.exe` is not present in the active environment.

The `/usr/bin/env` form of shebang line has one further special property. Before looking for installed Python interpreters, this form will search the executable `PATH` for a Python executable matching the name provided as the first argument. This corresponds to the behaviour of the Unix `env` program, which performs a `PATH` search. If an executable matching the first argument after the `env` command cannot be found, but the argument starts with `python`, it will be handled as described for the other virtual commands. The environment variable `PYLAUNCHER_NO_SEARCH_PATH` may be set (to any value) to skip this search of `PATH`.

Shebang lines that do not match any of these patterns are looked up in the `[commands]` section of the launcher's *.INI file*. This may be used to handle certain commands in a way that makes sense for your system. The name of the command must be a single argument (no spaces in the shebang executable), and the value substituted is the full path to the executable (additional arguments specified in the .INI will be quoted as part of the filename).

```
[commands]
/bin/xpython=C:\Program Files\XPython\python.exe
```

Any commands not found in the .INI file are treated as **Windows** executable paths that are absolute or relative to the directory containing the script file. This is a convenience for Windows-only scripts, such as those generated by an installer, since the behavior is not compatible with Unix-style shells. These paths may be quoted, and may include multiple arguments, after which the path to the script and any additional arguments will be appended.

4.8.3 셔뱅 줄의 인자

셔뱅 줄은 또한 파이썬 인터프리터에 전달할 추가 옵션을 지정할 수 있습니다. 예를 들어, 다음과 같은 셔뱅 줄이 있으면:

```
#! /usr/bin/python -v
```

파이썬은 `-v` 옵션으로 시작됩니다

4.8.4 사용자 정의

INI 파일을 통한 사용자 정의

Two .ini files will be searched by the launcher - `py.ini` in the current user's application data directory (`%LOCALAPPDATA%` or `$env:LocalAppData`) and `py.ini` in the same directory as the launcher. The same .ini files are used for both the 'console' version of the launcher (i.e. py.exe) and for the 'windows' version (i.e. pyw.exe).

"응용 프로그램 디렉터리"에 지정된 사용자 정의가 실행 파일 옆에 있는 사용자 지정보다 우선하므로, 런처 옆에 있는 .ini 파일에 대한 쓰기 권한이 없는 사용자는 전역 .ini 파일에서 명령을 재정의할 수 있습니다.

기본 파이썬 버전 사용자 정의

때에 따라, 버전 한정자를 명령에 포함하여 명령에서 사용할 파이썬 버전을 지정할 수 있습니다. 버전 한정자는 주 버전 번호로 시작하며 선택적으로 마침표('.')와 부 버전 지정자가 올 수 있습니다. 또한 "-32"나 "-64"를 추가하여 32비트나 64비트 구현을 요청할지를 지정할 수 있습니다.

예를 들어, `#!python`의 셔뱅 줄에는 버전 한정자가 없는 반면, `#!python3`에는 주 버전 만 지정하는 버전 한정자가 있습니다.

명령에 버전 한정자가 없으면, 환경 변수 `PY_PYTHON`을 설정하여 기본 버전 한정자를 지정할 수 있습니다. 설정되지 않으면, 기본값은 "3"입니다. 변수는 "3", "3.7", "3.7-32" 또는 "3.7-64"와 같이 명령 줄에서 전달할 수 있는 모든 값을 지정할 수 있습니다. ("-64" 옵션은 파이썬 3.7 이상에 포함된 런처에서만 사용할 수 있음에 유의하십시오.)

부 버전 한정자가 없으면, 환경 변수 `PY_PYTHON{major}`(여기서 `{major}`는 위에서 결정된 현재 주 버전 한정자입니다)를 설정하여 전체 버전을 지정할 수 있습니다. 그러한 옵션이 없으면, 런처는 설치된 파이썬 버전을 열거하고 주 버전에 대해 발견된 최신 부 릴리스를 사용합니다. 이것은 보장되지는 않지만, 해당 제품군에서 가장 최근에 설치된 버전일 가능성이 높습니다.

같은 (major.minor) 파이썬 버전의 32비트와 64비트 구현이 모두 설치된 64비트 윈도우에서는 항상 64비트 버전이 선호됩니다. 이는 32비트와 64비트 런처 구현 모두에 해당합니다 - 32비트 런처는 가능하면 지정된 버전의 64비트 파이썬 설치를 실행하는 것을 선호합니다. 따라서 런처의 동작은 PC에 설치된 버전만 알고 설치 순서와 관계없이 (즉, 32비트나 64비트 버전의 파이썬과 해당 런처가 마지막에 설치되었는지 알지 못하고) 예측할 수 있습니다). 위에서 언급했듯이, 선택적 "-32"나 "-64" 접미사를 버전 지정자에 사용하여 이 동작을 변경할 수 있습니다.

예:

- 관련 옵션이 설정되지 않으면, `python`과 `python2` 명령은 설치된 최신 파이썬 2.x 버전을 사용하고 `python3` 명령은 설치된 최신 파이썬 3.x를 사용합니다.
- The command `python3.7` will not consult any options at all as the versions are fully specified.
- `PY_PYTHON=3`이면, `python`과 `python3` 명령은 모두 최신 설치된 파이썬 3 버전을 사용합니다.
- If `PY_PYTHON=3.7-32`, the command `python` will use the 32-bit implementation of 3.7 whereas the command `python3` will use the latest installed Python (PY_PYTHON was not considered at all as a major version was specified.)
- If `PY_PYTHON=3` and `PY_PYTHON3=3.7`, the commands `python` and `python3` will both use specifically 3.7

환경 변수 외에도, 런처에서 사용하는 .INI 파일에서 같은 설정을 구성할 수 있습니다. INI 파일의 섹션은 `[defaults]`라고 하며 키 이름은 선행 `PY_` 접두어가 없는 환경 변수와 같습니다 (그리고 INI 파일의 키 이름은 대소 문자를 구분하지 않음에 유의하십시오.) 환경 변수의 내용은 INI 파일에 지정된 것을 재정의합니다.

예를 들면:

- Setting `PY_PYTHON=3.7` is equivalent to the INI file containing:

```
[defaults]
python=3.7
```

- Setting `PY_PYTHON=3` and `PY_PYTHON3=3.7` is equivalent to the INI file containing:

```
[defaults]
python=3
python3=3.7
```

4.8.5 진단

If an environment variable `PYLAUNCHER_DEBUG` is set (to any value), the launcher will print diagnostic information to stderr (i.e. to the console). While this information manages to be simultaneously verbose *and* terse, it should allow you to see what versions of Python were located, why a particular version was chosen and the exact command-line used to execute the target Python. It is primarily intended for testing and debugging.

4.8.6 Dry Run

If an environment variable `PYLAUNCHER_DRYRUN` is set (to any value), the launcher will output the command it would have run, but will not actually launch Python. This may be useful for tools that want to use the launcher to detect and then launch Python directly. Note that the command written to standard output is always encoded using UTF-8, and may not render correctly in the console.

4.8.7 Install on demand

If an environment variable `PYLAUNCHER_ALLOW_INSTALL` is set (to any value), and the requested Python version is not installed but is available on the Microsoft Store, the launcher will attempt to install it. This may require user interaction to complete, and you may need to run the command again.

An additional `PYLAUNCHER_ALWAYS_INSTALL` variable causes the launcher to always try to install Python, even if it is detected. This is mainly intended for testing (and should be used with `PYLAUNCHER_DRYRUN`).

4.8.8 Return codes

The following exit codes may be returned by the Python launcher. Unfortunately, there is no way to distinguish these from the exit code of Python itself.

The names of codes are as used in the sources, and are only for reference. There is no way to access or resolve them apart from reading this page. Entries are listed in alphabetical order of names.

이름	Value	설명
RC_BAD_VENV_CFG	107	A `pyvenv.cfg` was found but is corrupt.
RC_CREATE_PROCESS	101	Failed to launch Python.
RC_INSTALLING	111	An install was started, but the command will need to be re-run after it completes.
RC_INTERNAL_ERROR	109	Unexpected error. Please report a bug.
RC_NO_COMMANDLINE	108	Unable to obtain command line from the operating system.
RC_NO_PYTHON	103	Unable to locate the requested version.
RC_NO_VENV_CFG	106	A `pyvenv.cfg` was required but not found.

4.9 모듈 찾기

These notes supplement the description at sys-path-init with detailed Windows notes.

`._pth` 파일이 없을 때, 윈도우에서 `sys.path`를 채우는 방법은 다음과 같습니다:

- 시작에 현재 디렉터리에 해당하는 빈 항목이 추가됩니다.
- 환경 변수에 설명된 대로, 환경 변수 *PYTHONPATH*가 존재하면, 해당 항목이 다음에 추가됩니다. 윈도우에서, 이 변수의 경로는 드라이브 식별자(`C:\` 등)에 사용되는 콜론과 구별하기 위해 세미콜론으로 구분되어야 합니다.
- 추가 "응용 프로그램 경로"는 `HKEY_CURRENT_USER`와 `HKEY_LOCAL_MACHINE` 하이브 모두의 아래에 `\SOFTWARE\Python\PythonCore{version}\PythonPath`의 하위 키로 레지스트리에 추가될 수 있습니다. 세미콜론으로 구분된 경로 문자열을 기본값으로 사용하는 하위 키는 각 경로가 `sys.path`에 추가되도록 합니다. (알려진 모든 설치 프로그램은 HKLM 만 사용하므로, HKCU는 일반적으로 비어 있음에 유의하십시오.)
- 환경 변수 *PYTHONHOME*이 설정되면, "파이썬 홈"으로 간주합니다. 그렇지 않으면, 메인 파이썬 실행 파일의 경로를 사용하여 "랜드마크 파일"(`Lib\os.py`나 `pythonXY.zip`)을 찾아 "파이썬 홈"을 추론합니다. 파이썬 홈이 발견되면, `sys.path`에 추가되는 관련 하위 디렉터리(`Lib`, `plat-win` 등)는 해당 폴더를 기반으로 합니다. 그렇지 않으면, 핵심 파이썬 경로가 레지스트리에 저장된 PythonPath에서 구성됩니다.
- 파이썬 홈을 찾을 수 없고, 환경에 *PYTHONPATH*가 지정되어 있지 않고, 레지스트리 항목을 찾을 수 없으면, 상대 항목의 기본 경로(예를 들어 `.\Lib;.\plat-win` 등)가 사용됩니다.

`pyvenv.cfg` 파일이 메인 실행 파일과 함께 또는 실행 파일보다 한 수준 위의 디렉터리에서 발견되면, 다음 변형이 적용됩니다:

- `home`이 절대 경로이고 *PYTHONHOME*이 설정되지 않으면, 홈 위치를 추론할 때 메인 실행 파일에 대한 경로 대신 이 경로가 사용됩니다.

이 모든 것의 최종 결과는 다음과 같습니다:

- `python.exe` 또는 기본 파이썬 디렉터리 (설치된 버전 또는 PCbuild 디렉터리에서 직접)에서 다른 .exe를 실행할 때 핵심 경로가 추론되고 레지스트리의 핵심 경로가 무시됩니다. 레지스트리의 다른 "응용 프로그램 경로"는 항상 읽습니다.
- 파이썬이 다른 .exe(다른 디렉터리, COM을 통한 내장, 등)에서 호스팅 될 때, "파이썬 홈"이 추론되지 않아서, 레지스트리의 핵심 경로가 사용됩니다. 레지스트리의 다른 "응용 프로그램 경로"는 항상 읽힙니다.
- 파이썬이 홈을 찾을 수 없고 레지스트리 값이 없으면 (고정된 (frozen) .exe, 아주 이상한 설치 설정), 일부 기본 (하지만 상대) 경로를 얻게 됩니다.

파이썬을 응용 프로그램이나 배포에 번들로 포함하려는 사용자를 위해, 다음 조언은 다른 설치와의 충돌을 방지합니다:

- 포함할 디렉터리가 포함된 실행 파일과 함께 `._pth` 파일을 포함합니다. 이것은 레지스트리와 환경 변수에 나열된 경로를 무시하고, `import site`가 나열되지 않는 한 `site`도 무시합니다.
- If you are loading `python3.dll` or `python37.dll` in your own executable, explicitly set `PyConfig.module_search_paths` before `Py_InitializeFromConfig()`.
- 응용 프로그램에서 `python.exe`를 시작하기 전에 *PYTHONPATH*를 지우거나 덮어쓰고 *PYTHONHOME*를 설정하십시오.
- 이전 제안을 사용할 수 없으면 (예를 들어, 사용자가 `python.exe`를 직접 실행할 수 있는 배포판이면), 랜드마크 파일(`Lib\os.py`)이 설치 디렉터리에 있도록 하십시오. (ZIP 파일 내에서는 감지되지 않지만, 대신 올바른 이름의 ZIP 파일은 감지됨에 유의하십시오.)

이렇게 하면 시스템 전체 설치의 파일이 응용 프로그램과 함께 번들로 제공되는 표준 라이브러리의 복사본보다 우선하지 않습니다. 그렇지 않으면, 사용자가 여러분의 응용 프로그램을 사용하는 데 문제가 발생할 수 있습니다. 다른 제안은 레지스트리의 비표준 경로와 사용자 site-packages에 여전히 취약할 수 있어서, 첫 번째 제안이 가장 좋음에 유의하십시오.

버전 3.6에서 변경: Add `._pth` file support and removes `applocal` option from `pyvenv.cfg`.

버전 3.6에서 변경: Add `pythonXX.zip` as a potential landmark when directly adjacent to the executable.

버전 3.6부터 폐지됨: Modules specified in the registry under `Modules` (not `PythonPath`) may be imported by `importlib.machinery.WindowsRegistryFinder`. This finder is enabled on Windows in 3.6.0 and earlier, but may need to be explicitly added to `sys.meta_path` in the future.

4.10 추가 모듈

파이썬이 모든 플랫폼 간에 이식성 있는 것을 목표로 하지만, 윈도우에만 고유한 기능이 있습니다. 이러한 기능을 사용하기 위한 표준 라이브러리와 외부에 있는 두 개의 모듈과 스니펫(snippets)이 존재합니다.

윈도우 특정 표준 모듈은 mswin-specific-services에 설명되어 있습니다.

4.10.1 PyWin32

The PyWin32 module by Mark Hammond is a collection of modules for advanced Windows-specific support. This includes utilities for:

- Component Object Model (COM)
- Win32 API 호출
- 레지스트리
- 이벤트 로그
- Microsoft Foundation Classes (MFC) user interfaces

PythonWin은 PyWin32와 함께 제공되는 샘플 MFC 응용 프로그램입니다. 디버거가 내장된 내장할 수 있는 IDE입니다.

더 보기

Win32 How Do I…?
저자: Tim Golden

Python and COM
저자: David와 Paul Boddie

4.10.2 cx_Freeze

cx_Freeze wraps Python scripts into executable Windows programs (`*.exe` files). When you have done this, you can distribute your application without requiring your users to install Python.

4.11 윈도우에서 파이썬 컴파일하기

If you want to compile CPython yourself, first thing you should do is get the source. You can download either the latest release' s source or just grab a fresh checkout.

The source tree contains a build solution and project files for Microsoft Visual Studio, which is the compiler used to build the official Python releases. These files are in the `PCbuild` directory.

빌드 프로세스에 대한 일반 정보는 `PCbuild/readme.txt`를 확인하십시오.

확장 모듈에 대해서는, building-on-windows를 참조하십시오.

4.12 기타 플랫폼

파이썬의 지속적인 개발로 인해, 이전에 지원되던 일부 플랫폼은 더는 지원되지 않습니다 (사용자나 개발자 부족으로 인해). 지원되지 않는 모든 플랫폼에 대한 자세한 내용은 **PEP 11**을 확인하십시오.

- Windows CE is no longer supported since Python 3 (if it ever was).
- The Cygwin installer offers to install the Python interpreter as well

사전 컴파일된 설치 프로그램이 있는 플랫폼에 대한 자세한 정보는 Python for Windows를 참조하십시오.

CHAPTER 5

Using Python on macOS

This document aims to give an overview of macOS-specific behavior you should know about to get started with Python on Mac computers. Python on a Mac running macOS is very similar to Python on other Unix-derived platforms, but there are some differences in installation and some features.

There are various ways to obtain and install Python for macOS. Pre-built versions of the most recent versions of Python are available from a number of distributors. Much of this document describes use of the Pythons provided by the CPython release team for download from the python.org website. See *Alternative Distributions* for some other options.

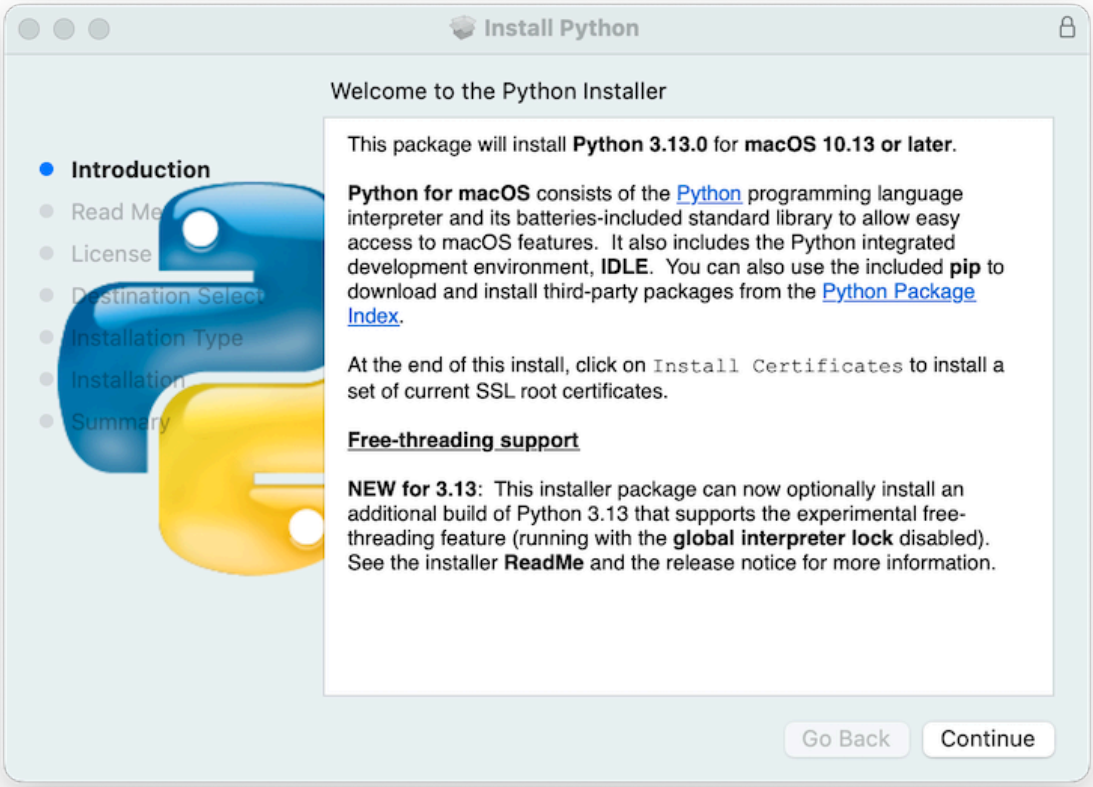
5.1 Using Python for macOS from `python.org`

5.1.1 Installation steps

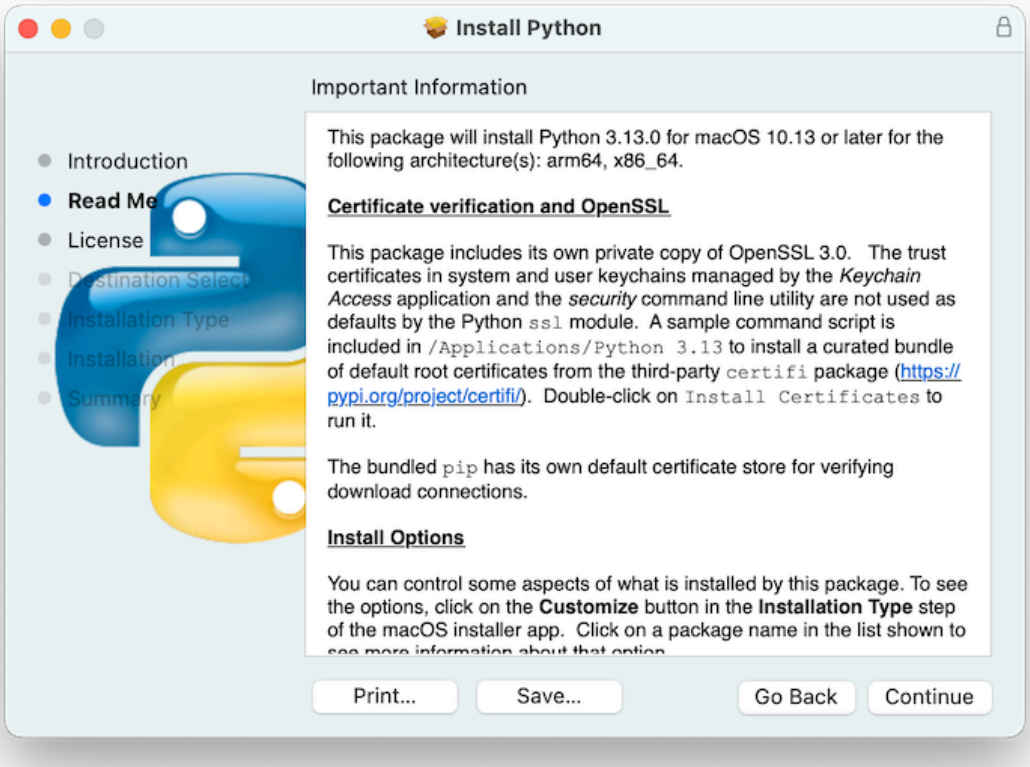
For current Python versions (other than those in `security` status), the release team produces a **Python for macOS** installer package for each new release. A list of available installers is available here. We recommend using the most recent supported Python version where possible. Current installers provide a universal2 binary build of Python which runs natively on all Macs (Apple Silicon and Intel) that are supported by a wide range of macOS versions, currently typically from at least **macOS 10.13 High Sierra** on.

The downloaded file is a standard macOS installer package file (`.pkg`). File integrity information (checksum, size, sigstore signature, etc) for each file is included on the release download page. Installer packages and their contents are signed and notarized with `Python Software Foundation` Apple Developer ID certificates to meet macOS Gatekeeper requirements.

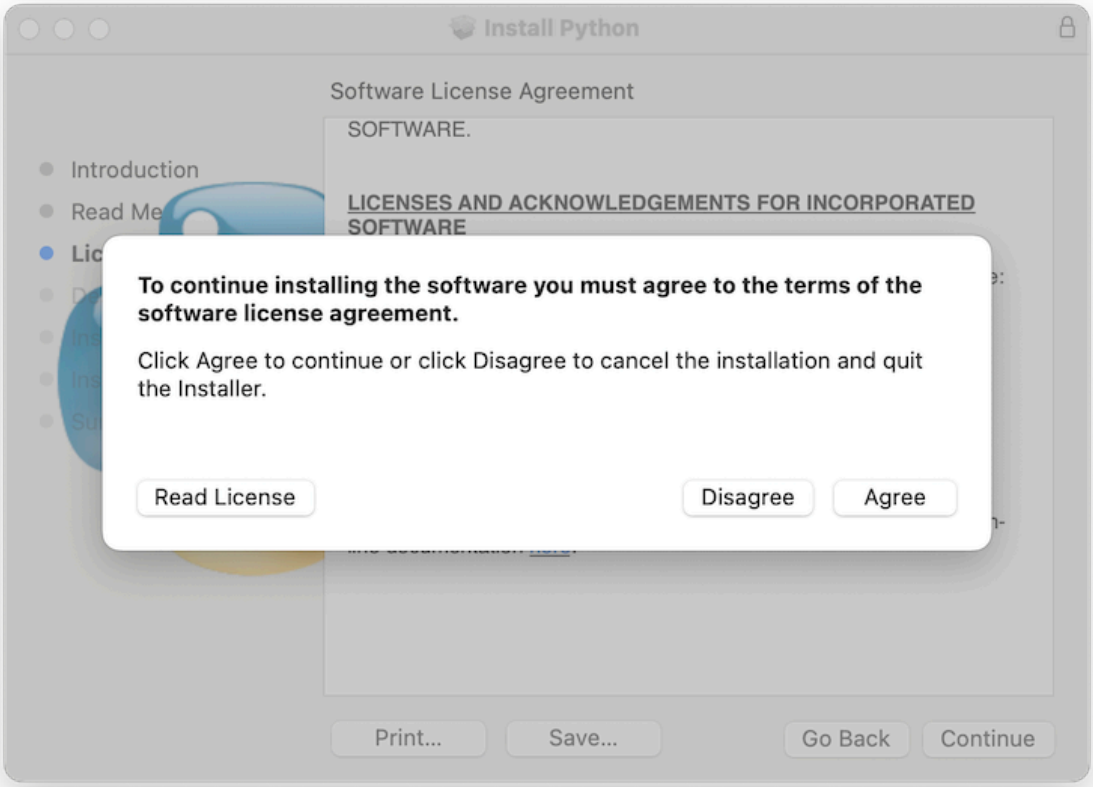
For a default installation, double-click on the downloaded installer package file. This should launch the standard macOS Installer app and display the first of several installer windows steps.



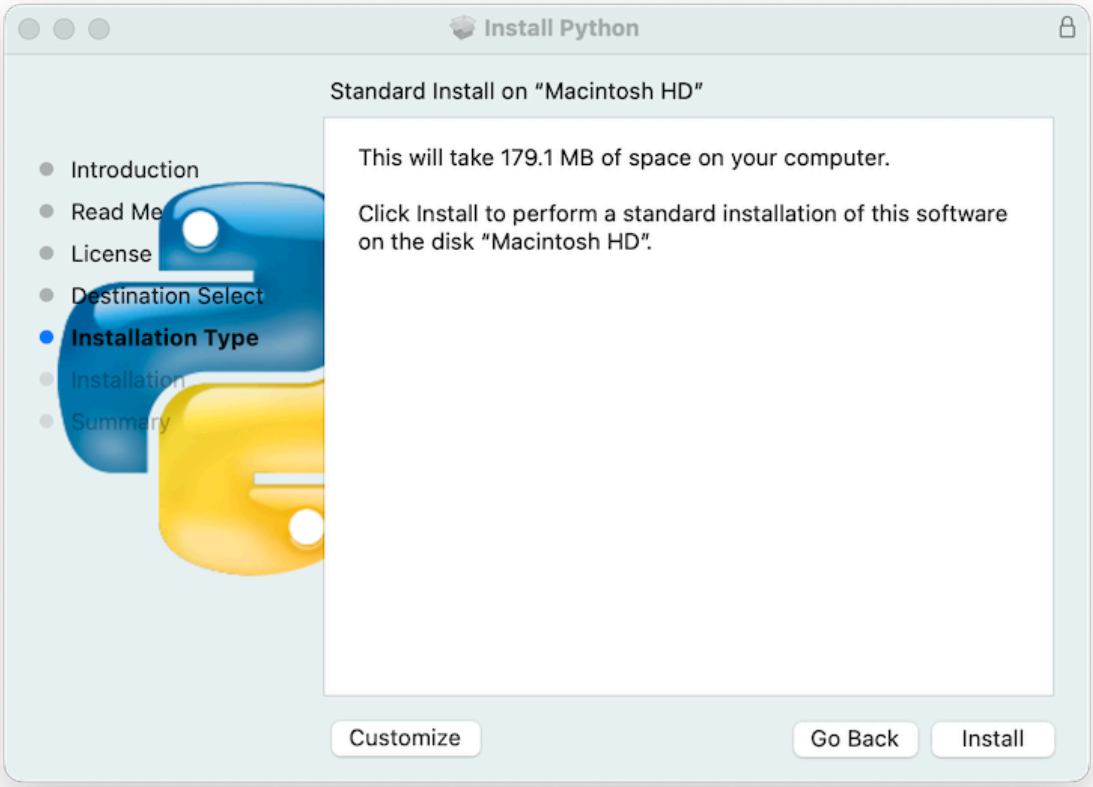
Clicking on the **Continue** button brings up the **Read Me** for this installer. Besides other important information, the **Read Me** documents which Python version is going to be installed and on what versions of macOS it is supported. You may need to scroll through to read the whole file. By default, this **Read Me** will also be installed in `/Applications/Python 3.13/` and available to read anytime.



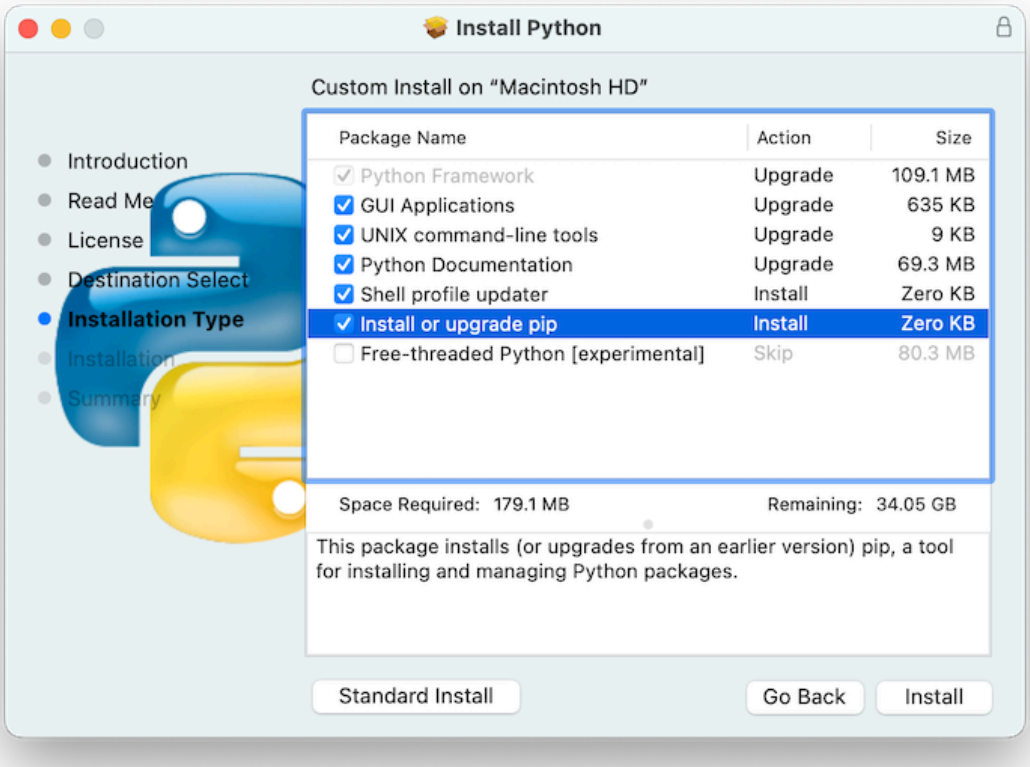
Clicking on **Continue** proceeds to display the license for Python and for other included software. You will then need to **Agree** to the license terms before proceeding to the next step. This license file will also be installed and available to be read later.



After the license terms are accepted, the next step is the **Installation Type** display. For most uses, the standard set of installation operations is appropriate.

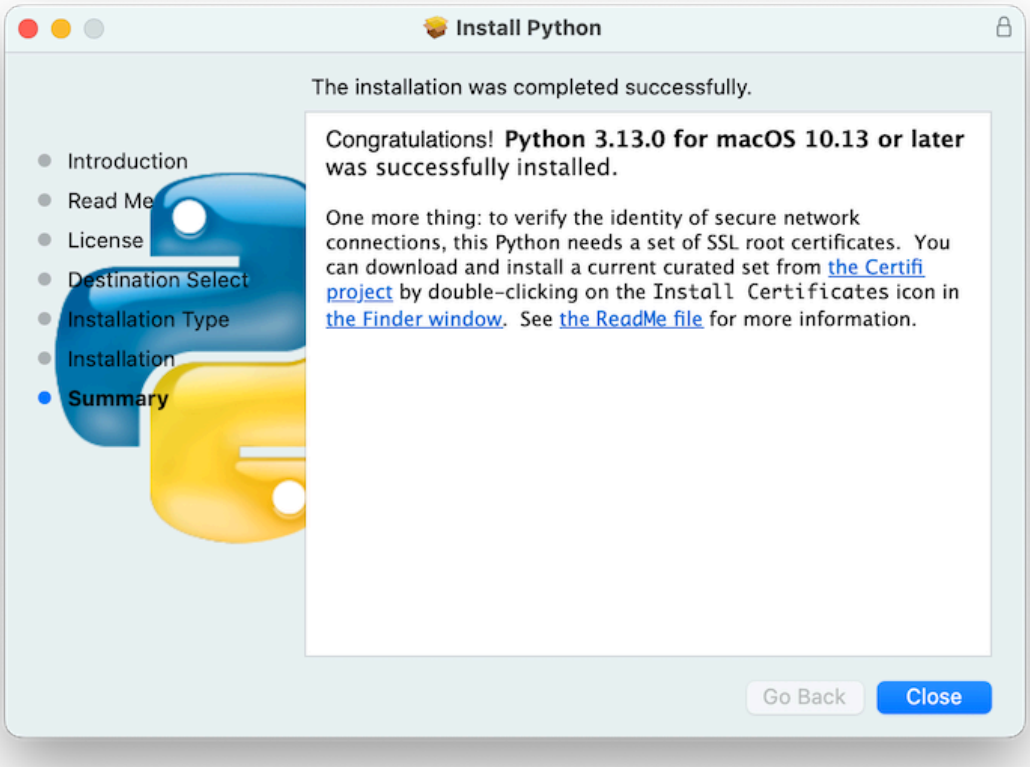


By pressing the **Customize** button, you can choose to omit or select certain package components of the installer. Click on each package name to see a description of what it installs. To also install support for the optional experimental free-threaded feature, see *Installing Free-threaded Binaries*.

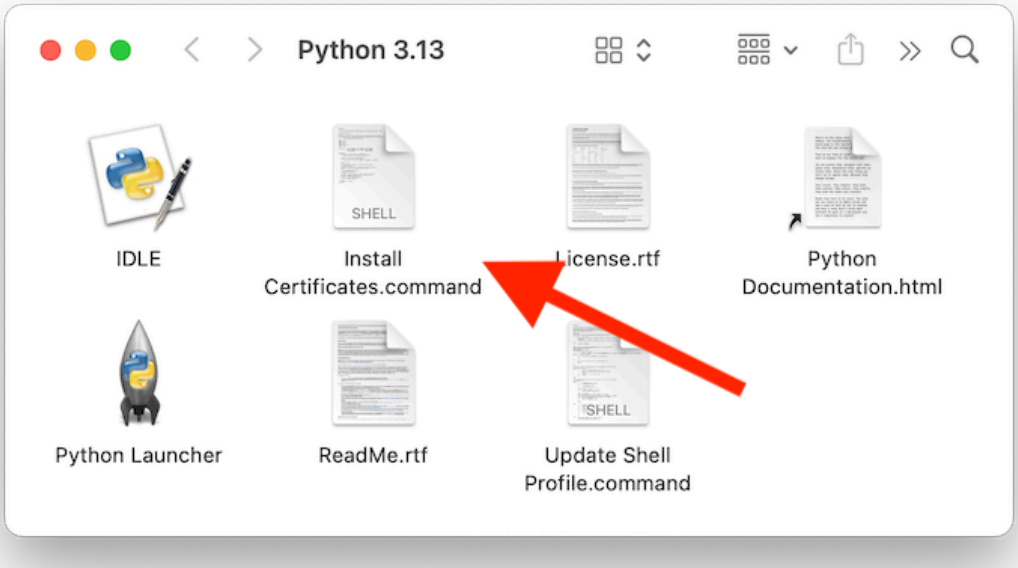


In either case, clicking **Install** will begin the install process by asking permission to install new software. A macOS user name with `Administrator` privilege is needed as the installed Python will be available to all users of the Mac.

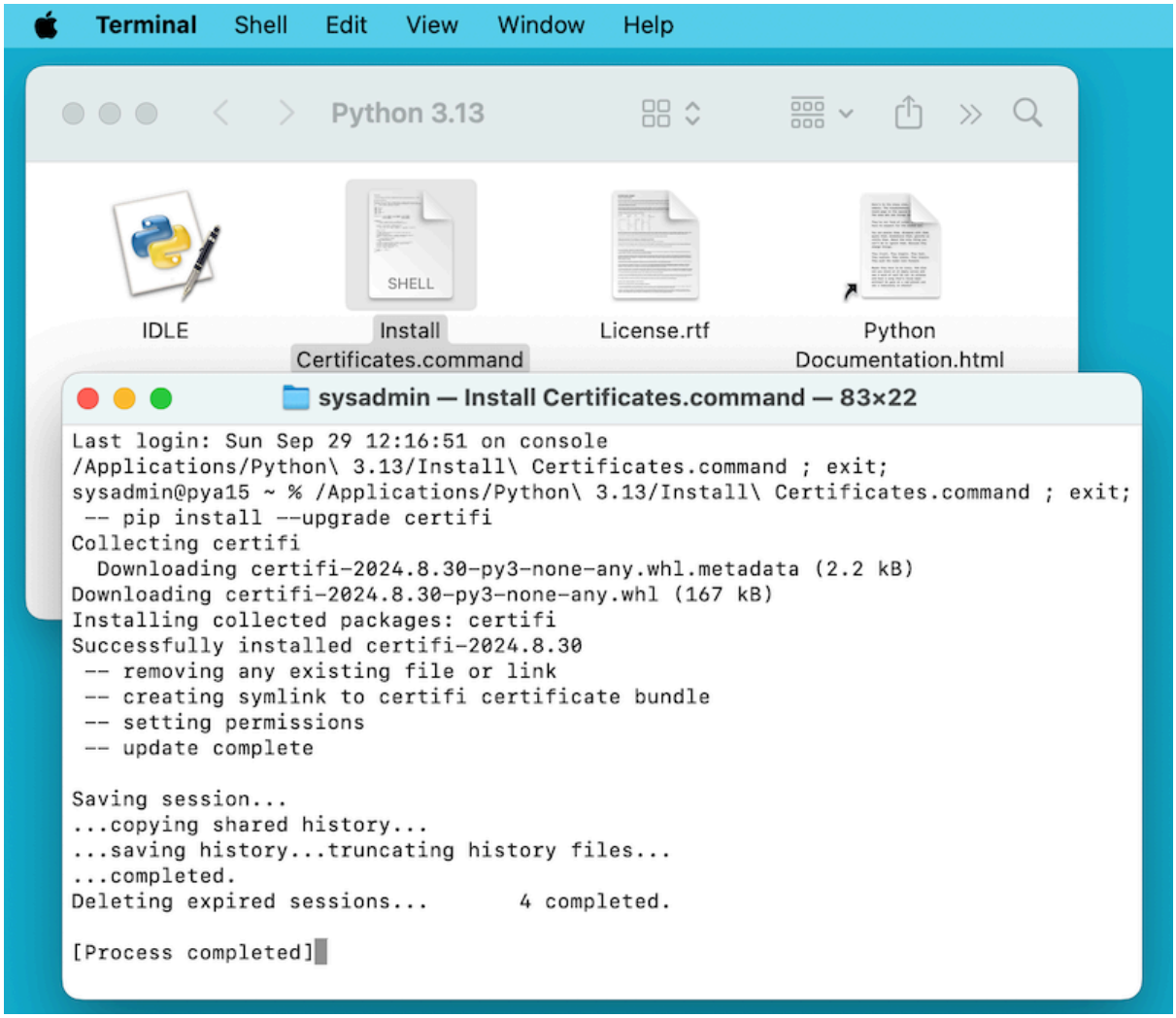
When the installation is complete, the **Summary** window will appear.



Double-click on the **`Install Certificates.command`** icon or file in the `/Applications/Python 3.13/` window to complete the installation.



This will open a temporary **`Terminal`** shell window that will use the new Python to download and install SSL root certificates for its use.



If `Successfully installed certifi` and `update complete` appears in the terminal window, the installation is complete. Close this terminal window and the installer window.

A default install will include:

- A `Python 3.13` folder in your `Applications` folder. In here you find **`IDLE`**, the development environment that is a standard part of official Python distributions; and **`Python Launcher`**, which handles double-clicking Python scripts from the macOS Finder.
- A framework `/Library/Frameworks/Python.framework`, which includes the Python executable and libraries. The installer adds this location to your shell path. To uninstall Python, you can remove these three things. Symlinks to the Python executable are placed in `/usr/local/bin/`.

> **참고**
>
> Recent versions of macOS include a **`python3`** command in `/usr/bin/python3` that links to a usually older and incomplete version of Python provided by and for use by the Apple development tools, **`Xcode`** or the **`Command Line Tools for Xcode`**. You should never modify or attempt to delete this installation, as it is Apple-controlled and is used by Apple-provided or third-party software. If you choose to install a newer Python version from `python.org`, you will have two different but functional Python installations on your computer that can co-exist. The default installer options should ensure that its **`python3`** will be used instead of the system **`python3`**.

5.1.2 파이썬 스크립트를 실행하는 방법

There are two ways to invoke the Python interpreter. If you are familiar with using a Unix shell in a terminal window, you can invoke `python3.13` or `python3` optionally followed by one or more command line options (described in 명령 줄과 환경). The Python tutorial also has a useful section on using Python interactively from a shell.

You can also invoke the interpreter through an integrated development environment. idle is a basic editor and interpreter environment which is included with the standard distribution of Python. **`IDLE`** includes a Help menu that allows you to access Python documentation. If you are completely new to Python, you can read the tutorial introduction in that document.

There are many other editors and IDEs available, see 편집기와 *IDE* for more information.

To run a Python script file from the terminal window, you can invoke the interpreter with the name of the script file:

```
python3.13 myscript.py
```

To run your script from the Finder, you can either:

- Drag it to **`Python Launcher`**.
- Select **`Python Launcher`** as the default application to open your script (or any `.py` script) through the Finder Info window and double-click it. **`Python Launcher`** has various preferences to control how your script is launched. Option-dragging allows you to change these for one invocation, or use its `Preferences` menu to change things globally.

Be aware that running the script directly from the macOS Finder might produce different results than when running from a terminal window as the script will not be run in the usual shell environment including any setting of environment variables in shell profiles. And, as with any other script or program, be certain of what you are about to run.

5.2 Alternative Distributions

Besides the standard `python.org` for macOS installer, there are third-party distributions for macOS that may include additional functionality. Some popular distributions and their key features:

ActivePython
: Installer with multi-platform compatibility, documentation

Anaconda
: Popular scientific modules (such as numpy, scipy, and pandas) and the `conda` package manager.

Homebrew
: Package manager for macOS including multiple versions of Python and many third-party Python-based packages (including numpy, scipy, and pandas).

MacPorts
: Another package manager for macOS including multiple versions of Python and many third-party Python-based packages. May include pre-built versions of Python and many packages for older versions of macOS.

Note that distributions might not include the latest versions of Python or other libraries, and are not maintained or supported by the core Python team.

5.3 추가 파이썬 패키지 설치하기

Refer to the Python Packaging User Guide for more information.

5.4 GUI Programming

Mac에서 파이썬으로 GUI 응용 프로그램을 작성하기 위한 몇 가지 옵션이 있습니다.

The standard Python GUI toolkit is `tkinter`, based on the cross-platform Tk toolkit (https://www.tcl.tk). A macOS-native version of Tk is included with the installer.

PyObjC is a Python binding to Apple' s Objective-C/Cocoa framework. Information on PyObjC is available from pyobjc.

A number of alternative macOS GUI toolkits are available including:

- PySide: Official Python bindings to the Qt GUI toolkit.
- PyQt: Alternative Python bindings to Qt.
- Kivy: A cross-platform GUI toolkit that supports desktop and mobile platforms.
- Toga: Part of the BeeWare Project; supports desktop, mobile, web and console apps.
- wxPython: A cross-platform toolkit that supports desktop operating systems.

5.5 Advanced Topics

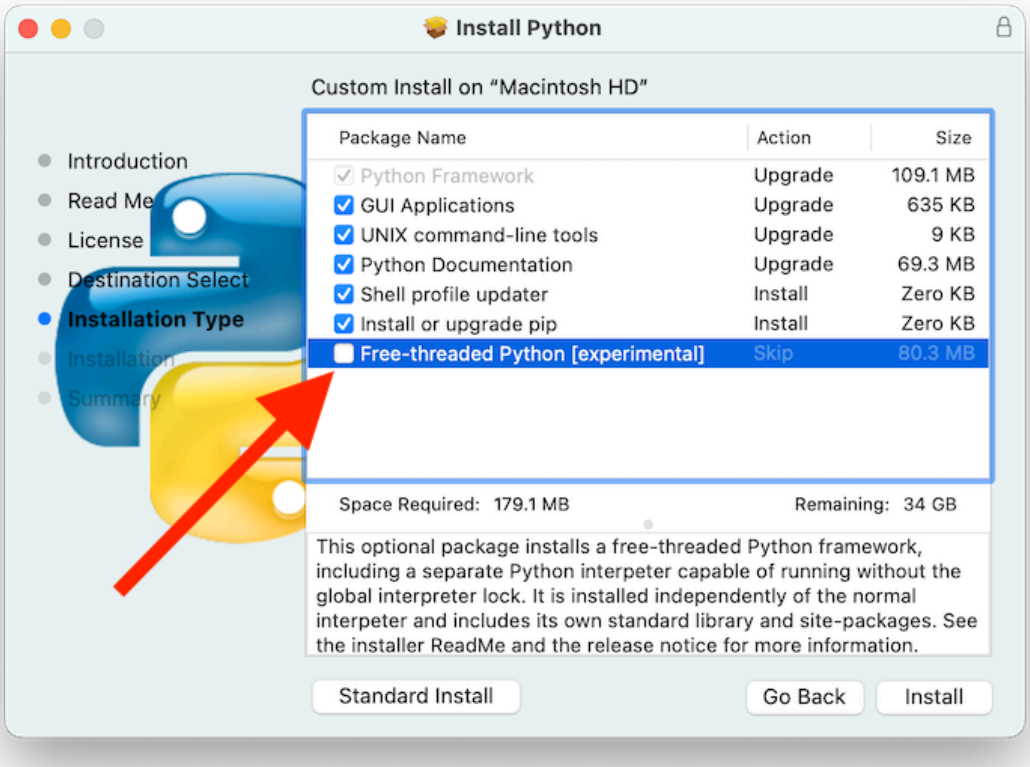
5.5.1 Installing Free-threaded Binaries

Added in version 3.13: (Experimental)

> **참고**
>
> Everything described in this section is considered experimental, and should be expected to change in future releases.

The `python.org` *Python for macOS* installer package can optionally install an additional build of Python 3.13 that supports **PEP 703**, the experimental free-threading feature (running with the *global interpreter lock* disabled). Check the release page on `python.org` for possible updated information.

Because this feature is still considered experimental, the support for it is not installed by default. It is packaged as a separate install option, available by clicking the **Customize** button on the **Installation Type** step of the installer as described above.



If the box next to the **Free-threaded Python** package name is checked, a separate `PythonT.framework` will also be installed alongside the normal `Python.framework` in `/Library/Frameworks`. This configuration allows a free-threaded Python 3.13 build to co-exist on your system with a traditional (GIL only) Python 3.13 build with minimal risk while installing or testing. This installation layout is itself experimental and is subject to change in future releases.

Known cautions and limitations:

- The **UNIX command-line tools** package, which is selected by default, will install links in `/usr/local/bin` for `python3.13t`, the free-threaded interpreter, and `python3.13t-config`, a configuration utility which may be useful for package builders. Since `/usr/local/bin` is typically included in your shell `PATH`, in most cases no changes to your `PATH` environment variables should be needed to use `python3.13t`.
- For this release, the **Shell profile updater** package and the `Update Shell Profile.command` in `/Applications/Python 3.13/` do not support the free-threaded package.
- The free-threaded build and the traditional build have separate search paths and separate `site-packages` directories so, by default, if you need a package available in both builds, it may need to be installed in both. The free-threaded package will install a separate instance of **`pip`** for use with `python3.13t`.
  - To install a package using **`pip`** without a **`venv`**:

    ```
    python3.13t -m pip install <package_name>
    ```

- When working with multiple Python environments, it is usually safest and easiest to create and use virtual environments. This can avoid possible command name conflicts and confusion about which Python is in use:

  ```
  python3.13t -m venv <venv_name>
  ```

  then **`activate`**.

- To run a free-threaded version of IDLE:

  ```
  python3.13t -m idlelib
  ```

- The interpreters in both builds respond to the same *PYTHON environment variables* which may have unexpected results, for example, if you have `PYTHONPATH` set in a shell profile. If necessary, there are *command line options* like `-E` to ignore these environment variables.
- The free-threaded build links to the third-party shared libraries, such as `OpenSSL` and `Tk`, installed in the traditional framework. This means that both builds also share one set of trust certificates as installed by the **`Install Certificates.command`** script, thus it only needs to be run once.
- If you cannot depend on the link in `/usr/local/bin` pointing to the `python.org` free-threaded `python3.13t` (for example, if you want to install your own version there or some other distribution does), you can explicitly set your shell `PATH` environment variable to include the `PythonT` framework `bin` directory:

```
export PATH="/Library/Frameworks/PythonT.framework/Versions/3.13/bin":"$PATH"
```

  The traditional framework installation by default does something similar, except for `Python.framework`. Be aware that having both framework `bin` directories in `PATH` can lead to confusion if there are duplicate names like `python3.13` in both; which one is actually used depends on the order they appear in `PATH`. The `which python3.x` or `which python3.xt` commands can show which path is being used. Using virtual environments can help avoid such ambiguities. Another option might be to create a shell **`alias`** to the desired interpreter, like:

```
alias py3.13="/Library/Frameworks/Python.framework/Versions/3.13/bin/python3.13
↪"
alias py3.13t="/Library/Frameworks/PythonT.framework/Versions/3.13/bin/python3.
↪13t"
```

5.5.2 Installing using the command line

If you want to use automation to install the `python.org` installer package (rather than by using the familiar macOS **`Installer`** GUI app), the macOS command line **`installer`** utility lets you select non-default options, too. If you are not familiar with **`installer`**, it can be somewhat cryptic (see **`man installer`** for more information). As an example, the following shell snippet shows one way to do it, using the `3.13.0b2` release and selecting the free-threaded interpreter option:

```
RELEASE="python-3.13.0b2-macos11.pkg"

# download installer pkg
curl -O https://www.python.org/ftp/python/3.13.0/${RELEASE}

# create installer choicechanges to customize the install:
#    enable the PythonTFramework-3.13 package
#    while accepting the other defaults (install all other packages)
cat > ./choicechanges.plist <<EOF
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
↪PropertyList-1.0.dtd">
<plist version="1.0">
<array>
        <dict>
                <key>attributeSetting</key>
                <integer>1</integer>
                <key>choiceAttribute</key>
                <string>selected</string>
                <key>choiceIdentifier</key>
                <string>org.python.Python.PythonTFramework-3.13</string>
        </dict>
</array>
</plist>
EOF
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
sudo installer -pkg ./${RELEASE} -applyChoiceChangesXML ./choicechanges.plist -
↪target /
```

You can then test that both installer builds are now available with something like:

```
$ # test that the free-threaded interpreter was installed if the Unix Command␣
↪Tools package was enabled
$ /usr/local/bin/python3.13t -VV
Python 3.13.0b2 experimental free-threading build (v3.13.0b2:3a83b172af, Jun  5␣
↪2024, 12:57:31) [Clang 15.0.0 (clang-1500.3.9.4)]
$ #    and the traditional interpreter
$ /usr/local/bin/python3.13 -VV
Python 3.13.0b2 (v3.13.0b2:3a83b172af, Jun  5 2024, 12:50:24) [Clang 15.0.0 (clang-
↪1500.3.9.4)]
$ # test that they are also available without the prefix if /usr/local/bin is on
↪$PATH
$ python3.13t -VV
Python 3.13.0b2 experimental free-threading build (v3.13.0b2:3a83b172af, Jun  5␣
↪2024, 12:57:31) [Clang 15.0.0 (clang-1500.3.9.4)]
$ python3.13 -VV
Python 3.13.0b2 (v3.13.0b2:3a83b172af, Jun  5 2024, 12:50:24) [Clang 15.0.0 (clang-
↪1500.3.9.4)]
```

참고

Current `python.org` installers only install to fixed locations like `/Library/Frameworks/`, `/Applications`, and `/usr/local/bin`. You cannot use the **`installer`** `-domain` option to install to other locations.

5.5.3 Distributing Python Applications

A range of tools exist for converting your Python code into a standalone distributable application:

- py2app: Supports creating macOS `.app` bundles from a Python project.
- Briefcase: Part of the BeeWare Project; a cross-platform packaging tool that supports creation of `.app` bundles on macOS, as well as managing signing and notarization.
- PyInstaller: A cross-platform packaging tool that creates a single file or folder as a distributable artifact.

5.5.4 App Store Compliance

Apps submitted for distribution through the macOS App Store must pass Apple' s app review process. This process includes a set of automated validation rules that inspect the submitted application bundle for problematic code.

The Python standard library contains some code that is known to violate these automated rules. While these violations appear to be false positives, Apple' s review rules cannot be challenged. Therefore, it is necessary to modify the Python standard library for an app to pass App Store review.

The Python source tree contains a patch file that will remove all code that is known to cause issues with the App Store review process. This patch is applied automatically when CPython is configured with the `--with-app-store-compliance` option.

This patch is not normally required to use CPython on a Mac; nor is it required if you are distributing an app *outside* the macOS App Store. It is *only* required if you are using the macOS App Store as a distribution channel.

5.6 기타 자원

The python.org Help page has links to many useful resources. The Pythonmac-SIG mailing list is another support resource specifically for Python users and developers on the Mac.

CHAPTER 6

Using Python on Android

Python on Android is unlike Python on desktop platforms. On a desktop platform, Python is generally installed as a system resource that can be used by any user of that computer. Users then interact with Python by running a **`python`** executable and entering commands at an interactive prompt, or by running a Python script.

On Android, there is no concept of installing as a system resource. The only unit of software distribution is an "app". There is also no console where you could run a **`python`** executable, or interact with a Python REPL.

As a result, the only way you can use Python on Android is in embedded mode – that is, by writing a native Android application, embedding a Python interpreter using `libpython`, and invoking Python code using the Python embedding API. The full Python interpreter, the standard library, and all your Python code is then packaged into your app for its own private use.

The Python standard library has some notable omissions and restrictions on Android. See the API availability guide for details.

6.1 Adding Python to an Android app

Most app developers should use one of the following tools, which will provide a much easier experience:

- Briefcase, from the BeeWare project
- Buildozer, from the Kivy project
- Chaquopy
- pyqtdeploy
- Termux

If you' re sure you want to do all of this manually, read on. You can use the testbed app as a guide; each step below contains a link to the relevant file.

- Build Python by following the instructions in Android/README.md. This will create the directory `cross-build/HOST/prefix`.
- Add code to your build.gradle file to copy the following items into your project. All except your own Python code can be copied from `prefix/lib`:
  - In your JNI libraries:
    * `libpython*.*.so`
    * `lib*_python.so` (external libraries such as OpenSSL)

  - In your assets:
    * `python*.*` (the Python standard library)
    * `python*.*/site-packages` (your own Python code)
- Add code to your app to extract the assets to the filesystem.
- Add code to your app to start Python in embedded mode. This will need to be C code called via JNI.

CHAPTER 7

Using Python on iOS

Authors
Russell Keith-Magee (2024-03)

Python on iOS is unlike Python on desktop platforms. On a desktop platform, Python is generally installed as a system resource that can be used by any user of that computer. Users then interact with Python by running a **`python`** executable and entering commands at an interactive prompt, or by running a Python script.

On iOS, there is no concept of installing as a system resource. The only unit of software distribution is an "app". There is also no console where you could run a **`python`** executable, or interact with a Python REPL.

As a result, the only way you can use Python on iOS is in embedded mode - that is, by writing a native iOS application, and embedding a Python interpreter using `libPython`, and invoking Python code using the Python embedding API. The full Python interpreter, the standard library, and all your Python code is then packaged as a standalone bundle that can be distributed via the iOS App Store.

If you' re looking to experiment for the first time with writing an iOS app in Python, projects such as BeeWare and Kivy will provide a much more approachable user experience. These projects manage the complexities associated with getting an iOS project running, so you only need to deal with the Python code itself.

7.1 Python at runtime on iOS

7.1.1 iOS version compatibility

The minimum supported iOS version is specified at compile time, using the *`--host`* option to `configure`. By default, when compiled for iOS, Python will be compiled with a minimum supported iOS version of 13.0. To use a different minimum iOS version, provide the version number as part of the `--host` argument - for example, `--host=arm64-apple-ios15.4-simulator` would compile an ARM64 simulator build with a deployment target of 15.4.

7.1.2 Platform identification

When executing on iOS, `sys.platform` will report as `ios`. This value will be returned on an iPhone or iPad, regardless of whether the app is running on the simulator or a physical device.

Information about the specific runtime environment, including the iOS version, device model, and whether the device is a simulator, can be obtained using `platform.ios_ver()`. `platform.system()` will report `iOS` or `iPadOS`, depending on the device.

`os.uname()` reports kernel-level details; it will report a name of `Darwin`.

7.1.3 Standard library availability

The Python standard library has some notable omissions and restrictions on iOS. See the API availability guide for iOS for details.

7.1.4 Binary extension modules

One notable difference about iOS as a platform is that App Store distribution imposes hard requirements on the packaging of an application. One of these requirements governs how binary extension modules are distributed.

The iOS App Store requires that *all* binary modules in an iOS app must be dynamic libraries, contained in a framework with appropriate metadata, stored in the `Frameworks` folder of the packaged app. There can be only a single binary per framework, and there can be no executable binary material outside the `Frameworks` folder.

This conflicts with the usual Python approach for distributing binaries, which allows a binary extension module to be loaded from any location on `sys.path`. To ensure compliance with App Store policies, an iOS project must post-process any Python packages, converting `.so` binary modules into individual standalone frameworks with appropriate metadata and signing. For details on how to perform this post-processing, see the guide for *adding Python to your project*.

To help Python discover binaries in their new location, the original `.so` file on `sys.path` is replaced with a `.fwork` file. This file is a text file containing the location of the framework binary, relative to the app bundle. To allow the framework to resolve back to the original location, the framework must contain a `.origin` file that contains the location of the `.fwork` file, relative to the app bundle.

For example, consider the case of an import `from foo.bar import _whiz`, where `_whiz` is implemented with the binary module `sources/foo/bar/_whiz.abi3.so`, with `sources` being the location registered on `sys.path`, relative to the application bundle. This module *must* be distributed as `Frameworks/foo.bar._whiz.framework/foo.bar._whiz` (creating the framework name from the full import path of the module), with an `Info.plist` file in the `.framework` directory identifying the binary as a framework. The `foo.bar._whiz` module would be represented in the original location with a `sources/foo/bar/_whiz.abi3.fwork` marker file, containing the path `Frameworks/foo.bar._whiz/foo.bar._whiz`. The framework would also contain `Frameworks/foo.bar._whiz.framework/foo.bar._whiz.origin`, containing the path to the `.fwork` file.

When running on iOS, the Python interpreter will install an `AppleFrameworkLoader` that is able to read and import `.fwork` files. Once imported, the `__file__` attribute of the binary module will report as the location of the `.fwork` file. However, the `ModuleSpec` for the loaded module will report the `origin` as the location of the binary in the framework folder.

7.1.5 Compiler stub binaries

Xcode doesn' t expose explicit compilers for iOS; instead, it uses an `xcrun` script that resolves to a full compiler path (e.g., `xcrun --sdk iphoneos clang` to get the `clang` for an iPhone device). However, using this script poses two problems:

- The output of `xcrun` includes paths that are machine specific, resulting in a sysconfig module that cannot be shared between users; and
- It results in `CC`/`CPP`/`LD`/`AR` definitions that include spaces. There is a lot of C ecosystem tooling that assumes that you can split a command line at the first space to get the path to the compiler executable; this isn' t the case when using `xcrun`.

To avoid these problems, Python provided stubs for these tools. These stubs are shell script wrappers around the underingly `xcrun` tools, distributed in a `bin` folder distributed alongside the compiled iOS framework. These scripts are relocatable, and will always resolve to the appropriate local system paths. By including these scripts in the bin folder that accompanies a framework, the contents of the `sysconfig` module becomes useful for end-users to compile their own modules. When compiling third-party Python modules for iOS, you should ensure these stub binaries are on your path.

7.2 Installing Python on iOS

7.2.1 Tools for building iOS apps

Building for iOS requires the use of Apple' s Xcode tooling. It is strongly recommended that you use the most recent stable release of Xcode. This will require the use of the most (or second-most) recently released macOS version, as Apple does not maintain Xcode for older macOS versions. The Xcode Command Line Tools are not sufficient for iOS development; you need a *full* Xcode install.

If you want to run your code on the iOS simulator, you' ll also need to install an iOS Simulator Platform. You should be prompted to select an iOS Simulator Platform when you first run Xcode. Alternatively, you can add an iOS Simulator Platform by selecting from the Platforms tab of the Xcode Settings panel.

7.2.2 Adding Python to an iOS project

Python can be added to any iOS project, using either Swift or Objective C. The following examples will use Objective C; if you are using Swift, you may find a library like PythonKit to be helpful.

To add Python to an iOS Xcode project:

1. Build or obtain a Python `XCFramework`. See the instructions in iOS/README.rst (in the CPython source distribution) for details on how to build a Python `XCFramework`. At a minimum, you will need a build that supports `arm64-apple-ios`, plus one of either `arm64-apple-ios-simulator` or `x86_64-apple-ios-simulator`.
2. Drag the `XCframework` into your iOS project. In the following instructions, we' ll assume you' ve dropped the `XCframework` into the root of your project; however, you can use any other location that you want by adjusting paths as needed.
3. Drag the `iOS/Resources/dylib-Info-template.plist` file into your project, and ensure it is associated with the app target.
4. Add your application code as a folder in your Xcode project. In the following instructions, we' ll assume that your user code is in a folder named `app` in the root of your project; you can use any other location by adjusting paths as needed. Ensure that this folder is associated with your app target.
5. Select the app target by selecting the root node of your Xcode project, then the target name in the sidebar that appears.
6. In the "General" settings, under "Frameworks, Libraries and Embedded Content", add `Python.xcframework`, with "Embed & Sign" selected.
7. In the "Build Settings" tab, modify the following:
   - Build Options
     - User Script Sandboxing: No
     - Enable Testability: Yes
   - Search Paths
     - Framework Search Paths: `$(PROJECT_DIR)`
     - Header Search Paths: `"$(BUILT_PRODUCTS_DIR)/Python.framework/Headers"`
   - Apple Clang - Warnings - All languages
     - Quoted Include In Framework Header: No
8. Add a build step that copies the Python standard library into your app. In the "Build Phases" tab, add a new "Run Script" build step *before* the "Embed Frameworks" step, but *after* the "Copy Bundle Resources" step. Name the step "Install Target Specific Python Standard Library", disable the "Based on dependency analysis" checkbox, and set the script content to:

```
set -e

mkdir -p "$CODESIGNING_FOLDER_PATH/python/lib"
if [ "$EFFECTIVE_PLATFORM_NAME" = "-iphonesimulator" ]; then
    echo "Installing Python modules for iOS Simulator"
    rsync -au --delete "$PROJECT_DIR/Python.xcframework/ios-arm64_x86_64-
↪simulator/lib/" "$CODESIGNING_FOLDER_PATH/python/lib/"
else
    echo "Installing Python modules for iOS Device"
    rsync -au --delete "$PROJECT_DIR/Python.xcframework/ios-arm64/lib/" "
↪$CODESIGNING_FOLDER_PATH/python/lib/"
fi
```

Note that the name of the simulator "slice" in the XCframework may be different, depending the CPU architectures your `XCFramework` supports.

9. Add a second build step that processes the binary extension modules in the standard library into "Framework" format. Add a "Run Script" build step *directly after* the one you added in step 8, named "Prepare Python Binary Modules". It should also have "Based on dependency analysis" unchecked, with the following script content:

```
set -e

install_dylib () {
    INSTALL_BASE=$1
    FULL_EXT=$2

    # The name of the extension file
    EXT=$(basename "$FULL_EXT")
    # The location of the extension file, relative to the bundle
    RELATIVE_EXT=${FULL_EXT#$CODESIGNING_FOLDER_PATH/}
    # The path to the extension file, relative to the install base
    PYTHON_EXT=${RELATIVE_EXT/$INSTALL_BASE/}
    # The full dotted name of the extension module, constructed from the file␣
↪path.
    FULL_MODULE_NAME=$(echo $PYTHON_EXT | cut -d "." -f 1 | tr "/" ".");
    # A bundle identifier; not actually used, but required by Xcode framework␣
↪packaging
    FRAMEWORK_BUNDLE_ID=$(echo $PRODUCT_BUNDLE_IDENTIFIER.$FULL_MODULE_NAME |␣
↪tr "_" "-")
    # The name of the framework folder.
    FRAMEWORK_FOLDER="Frameworks/$FULL_MODULE_NAME.framework"

    # If the framework folder doesn't exist, create it.
    if [ ! -d "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER" ]; then
        echo "Creating framework for $RELATIVE_EXT"
        mkdir -p "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER"
        cp "$CODESIGNING_FOLDER_PATH/dylib-Info-template.plist" "$CODESIGNING_
↪FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
        plutil -replace CFBundleExecutable -string "$FULL_MODULE_NAME" "
↪$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
        plutil -replace CFBundleIdentifier -string "$FRAMEWORK_BUNDLE_ID" "
↪$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
    fi

    echo "Installing binary for $FRAMEWORK_FOLDER/$FULL_MODULE_NAME"
    mv "$FULL_EXT" "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/$FULL_MODULE_
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
↪NAME"
    # Create a placeholder .fwork file where the .so was
    echo "$FRAMEWORK_FOLDER/$FULL_MODULE_NAME" > ${FULL_EXT%.so}.fwork
    # Create a back reference to the .so file location in the framework
    echo "${RELATIVE_EXT%.so}.fwork" > "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_
↪FOLDER/$FULL_MODULE_NAME.origin"
}

PYTHON_VER=$(ls -1 "$CODESIGNING_FOLDER_PATH/python/lib")
echo "Install Python $PYTHON_VER standard library extension modules..."
find "$CODESIGNING_FOLDER_PATH/python/lib/$PYTHON_VER/lib-dynload" -name "*.so
↪" | while read FULL_EXT; do
    install_dylib python/lib/$PYTHON_VER/lib-dynload/ "$FULL_EXT"
done

# Clean up dylib template
rm -f "$CODESIGNING_FOLDER_PATH/dylib-Info-template.plist"

echo "Signing frameworks as $EXPANDED_CODE_SIGN_IDENTITY_NAME ($EXPANDED_CODE_
↪SIGN_IDENTITY)..."
find "$CODESIGNING_FOLDER_PATH/Frameworks" -name "*.framework" -exec /usr/bin/
↪codesign --force --sign "$EXPANDED_CODE_SIGN_IDENTITY" ${OTHER_CODE_SIGN_
↪FLAGS:-} -o runtime --timestamp=none --preserve-metadata=identifier,
↪entitlements,flags --generate-entitlement-der "{}" \;
```

10. Add Objective C code to initialize and use a Python interpreter in embedded mode. You should ensure that:

    - UTF-8 mode (`PyPreConfig.utf8_mode`) is *enabled*;
    - Buffered stdio (`PyConfig.buffered_stdio`) is *disabled*;
    - Writing bytecode (`PyConfig.write_bytecode`) is *disabled*;
    - Signal handlers (`PyConfig.install_signal_handlers`) are *enabled*;
    - System logging (`PyConfig.use_system_logger`) is *enabled* (optional, but strongly recommended; this is enabled by default);
    - `PYTHONHOME` for the interpreter is configured to point at the `python` subfolder of your app' s bundle; and
    - The `PYTHONPATH` for the interpreter includes:
        - the `python/lib/python3.X` subfolder of your app' s bundle,
        - the `python/lib/python3.X/lib-dynload` subfolder of your app' s bundle, and
        - the `app` subfolder of your app' s bundle

    Your app' s bundle location can be determined using `[[NSBundle mainBundle] resourcePath]`.

Steps 8, 9 and 10 of these instructions assume that you have a single folder of pure Python application code, named `app`. If you have third-party binary modules in your app, some additional steps will be required:

- You need to ensure that any folders containing third-party binaries are either associated with the app target, or copied in as part of step 8. Step 8 should also purge any binaries that are not appropriate for the platform a specific build is targeting (i.e., delete any device binaries if you' re building an app targeting the simulator).
- Any folders that contain third-party binaries must be processed into framework form by step 9. The invocation of `install_dylib` that processes the `lib-dynload` folder can be copied and adapted for this purpose.
- If you' re using a separate folder for third-party packages, ensure that folder is included as part of the `PYTHONPATH` configuration in step 10.

7.2.3 Testing a Python package

The CPython source tree contains a testbed project that is used to run the CPython test suite on the iOS simulator. This testbed can also be used as a testbed project for running your Python library' s test suite on iOS.

After building or obtaining an iOS XCFramework (See iOS/README.rst for details), create a clone of the Python iOS testbed project by running:

```
$ python iOS/testbed clone --framework <path/to/Python.xcframework> --app <path/to/
↪module1> --app <path/to/module2> app-testbed
```

You will need to modify the `iOS/testbed` reference to point to that directory in the CPython source tree; any folders specified with the `--app` flag will be copied into the cloned testbed project. The resulting testbed will be created in the `app-testbed` folder. In this example, the `module1` and `module2` would be importable modules at runtime. If your project has additional dependencies, they can be installed into the `app-testbed/iOSTestbed/app_packages` folder (using `pip install --target app-testbed/iOSTestbed/app_packages` or similar).

You can then use the `app-testbed` folder to run the test suite for your app, For example, if `module1.tests` was the entry point to your test suite, you could run:

```
$ python app-testbed run -- module1.tests
```

This is the equivalent of running `python -m module1.tests` on a desktop Python build. Any arguments after the `--` will be passed to the testbed as if they were arguments to `python -m` on a desktop machine.

You can also open the testbed project in Xcode by running:

```
$ open app-testbed/iOSTestbed.xcodeproj
```

This will allow you to use the full Xcode suite of tools for debugging.

7.3 App Store Compliance

The only mechanism for distributing apps to third-party iOS devices is to submit the app to the iOS App Store; apps submitted for distribution must pass Apple' s app review process. This process includes a set of automated validation rules that inspect the submitted application bundle for problematic code.

The Python standard library contains some code that is known to violate these automated rules. While these violations appear to be false positives, Apple' s review rules cannot be challenged; so, it is necessary to modify the Python standard library for an app to pass App Store review.

The Python source tree contains a patch file that will remove all code that is known to cause issues with the App Store review process. This patch is applied automatically when building for iOS.

CHAPTER 8

편집기와 IDE

파이썬 프로그래밍 언어를 지원하는 여러 IDE가 있습니다. 많은 편집기와 IDE는 구문 강조, 디버깅 도구 및 **PEP 8** 검사를 제공합니다.

8.1 IDLE — Python editor and shell

IDLE is Python' s Integrated Development and Learning Environment and is generally bundled with Python installs. If you are on Linux and do not have IDLE installed see *Installing IDLE on Linux*. For more information see the IDLE docs.

8.2 다른 편집기와 IDE

파이썬의 커뮤니티 위키에는 편집기와 IDE에 대해 커뮤니티가 제공한 정보가 있습니다. 포괄적인 목록을 보려면 Python Editors와 Integrated Development Environments로 이동하십시오.

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.

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.

annotate function
A function that can be called to retrieve the *annotations* of an object. This function is accessible as the `__annotate__` attribute of functions, classes, and modules. Annotate functions are a subset of *evaluate functions*.

annotation (어노테이션)
관습에 따라 형 힌트 로 사용되는 변수, 클래스 어트리뷰트 또는 함수 매개변수 나 반환 값과 연결된 레이블입니다.

Annotations of local variables cannot be accessed at runtime, but annotations of global variables, class attributes, and functions can be retrieved by calling `annotationlib.get_annotations()` on modules, classes, and functions, respectively.

See *variable annotation*, *function annotation*, **PEP 484**, **PEP 526**, and **PEP 649**, 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)
`__aenter__()` 와 `__aexit__()` 메서드를 정의함으로써 `async with` 문에서 보이는 환경을 제어하는 객체. **PEP 492**로 도입되었습니다.

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.

비동기 이터레이터 인데 `__anext__()` 를 호출하면 어웨이터블 객체를 돌려주고, 이것은 다음 `yield` 표현식 까지 비동기 제너레이터 함수의 바디를 실행합니다.

각 `yield`는 일시적으로 처리를 중단하고, (지역 변수들과 대기 중인 try-문들을 포함하는) 실행 상태를 기억합니다. 비동기 제너레이터 이터레이터 가 `__anext__()` 가 돌려주는 또 하나의 어웨이터블로 재개되면, 떠난 곳으로 복귀합니다. **PEP 492**와 **PEP 525**를 보세요.

asynchronous iterable (비동기 이터러블)
`async for` 문에서 사용될 수 있는 객체. `__aiter__()` 메서드는 비동기 이터레이터 를 돌려줘야 합니다. **PEP 492** 로 도입되었습니다.

asynchronous iterator (비동기 이터레이터)
`__aiter__()` 와 `__anext__()` 메서드를 구현하는 객체. `__anext__()` 는 어웨이터블 객체를 돌려줘야 합니다. `async for`는 `StopAsyncIteration` 예외가 발생할 때까지 비동기 이터레이터의 `__anext__()` 메서드가 돌려주는 어웨이터블을 풉니다. **PEP 492**로 도입되었습니다.

attached thread state
A *thread state* that is active for the current OS thread.

When a *thread state* is attached, the OS thread has access to the full Python C API and can safely invoke the bytecode interpreter.

Unless a function explicitly notes otherwise, attempting to call the C API without an attached thread state will result in a fatal error or undefined behavior. A thread state can be attached and detached explicitly by the user through the C API, or implicitly by the runtime, including during blocking C calls and by the bytecode interpreter in between calls.

On most builds of Python, having an attached thread state implies that the caller holds the *GIL* for the current interpreter, so only one OS thread can have an attached thread state at a given moment. In *free-threaded* builds

of Python, threads can concurrently hold an attached thread state, allowing for true parallelism of the bytecode interpreter.

attribute (어트리뷰트)
흔히 점표현식을 사용하는 이름으로 참조되는 객체와 결합한 값. 예를 들어, 객체 *o*가 어트리뷰트 *a*를 가지면, *o.a*처럼 참조됩니다.

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)
`await` 표현식에 사용할 수 있는 객체. 코루틴 이나 `__await__()` 메서드를 가진 객체가 될 수 있습니다. **PEP 492**를 보세요.

BDFL
자비로운 종신 독재자 (Benevolent Dictator For Life), 즉 Guido van Rossum, 파이썬의 창시자.

binary file (바이너리 파일)
바이트열류 객체들을 읽고 쓸 수 있는 파일 객체. 바이너리 파일의 예로는 바이너리 모드 (`'rb'`, `'wb'` 또는 `'rb+'`) 로 열린 파일, `sys.stdin.buffer`, `sys.stdout.buffer`, `io.BytesIO` 와 `gzip.GzipFile` 의 인스턴스를 들 수 있습니다.

`str` 객체를 읽고 쓸 수 있는 파일 객체에 대해서는 텍스트 파일 도 참조하세요.

borrowed reference (빌린 참조)
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)
bufferobjects 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 (콜러블)
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.

closure variable (클로저 변수)
A *free variable* referenced from a *nested scope* that is defined in an outer scope rather than being resolved at runtime from the globals or builtin namespaces. May be explicitly defined with the `nonlocal` keyword to allow write access, or implicitly defined if the variable is only being read.

For example, in the `inner` function in the following code, both `x` and `print` are *free variables*, but only `x` is a *closure variable*:

```
def outer():
    x = 0
    def inner():
        nonlocal x
        x += 1
        print(x)
    return inner
```

Due to the `codeobject.co_freevars` attribute (which, despite its name, only includes the names of closure variables rather than listing all referenced free variables), the more general *free variable* term is sometimes used even when the intended meaning is to refer specifically to closure variables.

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 (컨텍스트)
This term has different meanings depending on where and how it is used. Some common meanings:

- The temporary state or environment established by a *context manager* via a `with` statement.
- The collection of keyvalue bindings associated with a particular `contextvars.Context` object and accessed via `ContextVar` objects. Also see *context variable*.
- A `contextvars.Context` object. Also see *current context*.

context management protocol (컨텍스트 관리 프로토콜)
The `__enter__()` and `__exit__()` methods called by the `with` statement. See **PEP 343**.

context manager (컨텍스트 관리자)
컨텍스트 관리 프로토콜을 구현하고 `with` 문에서 보이는 환경을 제어하는 객체. **PEP 343**을 참조하십시오.

context variable (컨텍스트 변수)
A variable whose value depends on which context is the *current context*. Values are accessed via `contextvars.ContextVar` objects. Context variables are primarily used to isolate state between concurrent asynchronous tasks.

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.

current context (현재 컨텍스트)
The *context* (`contextvars.Context` object) that is currently used by `ContextVar` objects to access (get or set) the values of *context variables*. Each thread has its own current context. Frameworks for executing asynchronous tasks (see `asyncio`) associate each task with a context which becomes the current context whenever the task starts or resumes execution.

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)
메서드 `__get__()` 이나 `__set__()` 이나 `__delete__()`를 정의하는 객체. 클래스 어트리뷰트가 디스크립터일 때, 어트리뷰트 조회는 특별한 연결 작용을 일으킵니다. 보통, *a.b*를 읽거나, 쓰거나, 삭제하는데 사용할 때, *a*의 클래스 딕셔너리에서 *b* 라고 이름 붙여진 객체를 찾습니다. 하지만 *b*가 디스크립터면, 해당하는 디스크립터 메서드가 호출됩니다. 디스크립터를 이해하는 것은 파이썬에 대한 깊은 이해의 열쇠인데, 함수, 메서드, 프로퍼티, 클래스 메서드, 스태틱 메서드, 슈퍼 클래스 참조 등의 많은 기능의 기초를 이루고 있기 때문입니다.

디스크립터의 메서드들에 대한 자세한 내용은 descriptors나 디스크립터 사용법 안내서에 나옵니다.

dictionary (딕셔너리)
임의의 키를 값에 대응시키는 연관 배열 (associative array). 키는 `__hash__()` 와 `__eq__()` 메서드를 갖는 모든 객체가 될 수 있습니다. 펄에서 해시라고 부릅니다.

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)
클래스, 함수, 모듈에서 첫 번째 표현식으로 나타나는 문자열 리터럴. 스위트가 실행될 때는 무시되

지만, 컴파일러에 의해 인지되어 둘러싼 클래스, 함수, 모듈의 `__doc__` 어트리뷰트로 삽입됩니다. 인트로스펙션을 통해 사용할 수 있으므로, 객체의 설명서를 위한 규범적인 장소입니다.

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.

evaluate function
A function that can be called to evaluate a lazily evaluated attribute of an object, such as the value of type aliases created with the `type` statement.

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)
하부 자원에 대해 파일 지향적 API(`read()` 나 `write()` 같은 메서드들)를 드러내는 객체. 만들어진 방법에 따라, 파일 객체는 실제 디스크 상의 파일이나 다른 저장 장치나 통신 장치 (예를 들어, 표준 입출력, 인-메모리 버퍼, 소켓, 파이프, 등등)에 대한 액세스를 중계할 수 있습니다. 파일 객체는 파일류 객체 (*file-like objects*) 나 스트림 (*streams*) 이라고도 불립니다.

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 (파일시스템 인코딩과 에러 처리기)
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`.

로케일 인코딩 도 보세요.

finder (파인더)
임포트될 모듈을 위한 로더 를 찾으려고 시도하는 객체.

두 종류의 파인더가 있습니다: `sys.meta_path` 와 함께 사용하는 메타 경로 파인더 와 `sys.path_hooks` 과 함께 사용하는 경로 엔트리 파인더.

더 자세한 내용은 finders-and-loaders 와 `importlib`를 참조하십시오.

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.

free threading (자유 스레딩)
A threading model where multiple threads can run Python bytecode simultaneously within the same interpreter. This is in contrast to the *global interpreter lock* which allows only one thread to execute Python bytecode at a time. See **PEP 703**.

free variable (자유 변수)
Formally, as defined in the language execution model, a free variable is any variable used in a namespace which is not a local variable in that namespace. See *closure variable* for an example. Pragmatically, due to the name of the `codeobject.co_freevars` attribute, the term is also sometimes used as a synonym for *closure variable*.

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.

이 기능을 설명하는 변수 어노테이션 과 **PEP 484**를 참조하세요. 또한 어노테이션에 대한 모범 사례는 annotations-howto를 참조하세요.

__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.

각 yield는 일시적으로 처리를 중단하고, (지역 변수들과 대기 중인 try-문들을 포함하는) 실행 상태를 기억합니다. 제너레이터 이터레이터 가 재개되면, 떠난 곳으로 복귀합니다 (호출마다 새로 시작하는 함수와 대비됩니다).

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)
매개 변수화 할 수 있는 형; 일반적으로 list 나 dict와 같은 컨테이너 클래스. 형 힌트와 어노테이션에 사용됩니다.

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. (dict와 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.

하지만, 어떤 확장 모듈들은, 표준이나 제삼자 모두, 압축이나 해싱 같은 계산 집약적인 작업을 수행할 때는 GIL을 반납하도록 설계되었습니다. 또한, I/O를 할 때는 항상 GIL을 반납합니다.

As of Python 3.13, the GIL can be disabled using the *--disable-gil* build configuration. After building Python with this option, code must be run with *-X gil=0* or after setting the *PYTHON_GIL=0* environment variable. This feature enables improved performance for multi-threaded applications and makes it easier to use multi-core CPUs efficiently. For more details, see **PEP 703**.

In prior versions of Python' s C API, a function might declare that it requires the GIL to be held in order to use it. This refers to having an *attached thread state*.

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)
객체가 일생 그 값이 변하지 않는 해시값을 갖고 (__hash__() 메서드가 필요합니다), 다른 객체와 비교될 수 있으면 (__eq__() 메서드가 필요합니다), 해시 가능하다고 합니다. 같다고 비교되는 해시 가능한 객체들의 해시값은 같아야 합니다.

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
파이썬을 위한 통합 개발 및 학습 환경 (Integrated Development and Learning Environment). idle은 파이썬의 표준 배포판에 따라오는 기초적인 편집기와 인터프리터 환경입니다.

immortal (불멸)
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.

Immortal objects can be identified via `sys._is_immortal()`, or via `PyUnstable_IsImmortal()` in the C API.

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). 대화형 모드에 대한 자세한 내용은 tut-interac 를 보세요.

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)
멤버들을 한 번에 하나씩 돌려줄 수 있는 객체. 이터러블의 예로는 모든 (`list`, `str`, `tuple` 같은) 시퀀스 형들, `dict` 같은 몇몇 비 시퀀스 형들, 파일 객체들, `__iter__()` 나 시퀀스 개념을 구현하는 `__getitem__()` 메서드를 써서 정의한 모든 클래스의 객체들이 있습니다.

이터러블은 `for` 루프에 사용될 수 있고, 시퀀스를 필요로 하는 다른 많은 곳 (`zip()`, `map()`, …) 에 사용될 수 있습니다. 이터러블 객체가 내장 함수 `iter()` 에 인자로 전달되면, 그 객체의 이터레이터를 돌려줍니다. 이 이터레이터는 값들의 집합을 한 번 거치는 동안 유효합니다. 이터러블을 사용할 때, 보통은 `iter()` 를 호출하거나, 이터레이터 객체를 직접 다룰 필요는 없습니다. `for` 문은 이것들을 여러분을 대신해서 자동으로 해주는데, 루프를 도는 동안 이터레이터를 잡아둘 이름 없는 변수를 만듭니다. 이터레이터, 시퀀스, 제너레이터 도 보세요.

iterator (이터레이터)
데이터의 스트림을 표현하는 객체. 이터레이터의 `__next__()` 메서드를 반복적으로 호출하면 (또는 내장 함수 `next()` 로 전달하면) 스트림에 있는 항목들을 차례대로 돌려줍니다. 더 이상의 데이터가 없을 때는 대신 `StopIteration` 예외를 일으킵니다. 이 지점에서, 이터레이터 객체는 소진되고, 이후의 모든 `__next__()` 메서드 호출은 `StopIteration` 예외를 다시 일으키기만 합니다. 이터레이터는 이터레이터 객체 자신을 돌려주는 `__iter__()` 메서드를 가질 것이 요구되기 때문에, 이터레이터는

이터러블이기도 하고 다른 이터러블들을 받아들이는 대부분의 곳에서 사용될 수 있습니다. 중요한 예외는 여러 번의 이터레이션을 시도하는 코드입니다. (`list` 같은) 컨테이너 객체는 `iter()` 함수로 전달하거나 `for` 루프에 사용할 때마다 새 이터레이터를 만듭니다. 이런 것을 이터레이터에 대해서 수행하려고 하면, 지난 이터레이션에 사용된 이미 소진된 이터레이터를 돌려줘서, 빈 컨테이너처럼 보이게 만듭니다.

typeiter 에 더 자세한 내용이 있습니다.

CPython 구현 상세: CPython does not consistently apply the requirement that an iterator define `__iter__()`. And also please note that the free-threading CPython does not guarantee the thread-safety of iterator operations.

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.

키 함수를 만드는 데는 여러 방법이 있습니다. 예를 들어, `str.lower()` 메서드는 케이스 구분 없는 정렬을 위한 키 함수로 사용될 수 있습니다. 대안적으로, 키 함수는 `lambda` 표현식으로 만들 수도 있는데, 이런 식입니다: `lambda r: (r[0], r[2])`. 또한, `operator.attrgetter()`, `operator.itemgetter()`, `operator.methodcaller()` 가 세 개의 키 함수 생성자입니다. 키 함수를 만들고 사용하는 법에 대한 예로 Sorting HOW TO 를 보세요.

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 (어휘 분석기)
Formal name for the *tokenizer*; see *token*.

list (리스트)
내장 파이썬 시퀀스. 그 이름에도 불구하고, 원소에 대한 액세스가 $O(1)$이기 때문에, 연결 리스트(linked list)보다는 다른 언어의 배열과 유사합니다.

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 (로케일 인코딩)
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)
임의의 키 조회를 지원하고 `collections.abc.Mapping` 이나 `collections.abc.MutableMapping` 추상 베이스 클래스 에 지정된 메서드들을 구현하는 컨테이너 객체. 예로는 `dict`, `collections.defaultdict`, `collections.OrderedDict`, `collections.Counter` 를 들 수 있습니다.

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)
메서드 결정 순서는 조회하는 동안 멤버를 검색하는 베이스 클래스들의 순서입니다. 2.3 릴리스부터 파이썬 인터프리터에 사용된 알고리즘의 상세한 내용은 python_2.3_mro를 보세요.

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.

module-specs 도 보세요.

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
>>> sys.float_info.max_exp              # named field access
1024
>>> isinstance(sys.float_info, tuple)   # kind of tuple
True
```

일부 네임드 튜플은 내장형(위의 예)입니다. 또는, tuple에서 상속하고 이름 붙은 필드를 정의하는 일반 클래스 정의로 네임드 튜플을 만들 수 있습니다. 이러한 클래스는 직접 작성하거나, typing.NamedTuple를 계승하거나 팩토리 함수 collections.namedtuple()로 만들 수 있습니다. 후자의 기법은 직접 작성하거나 내장 네임드 튜플에서는 찾을 수 없는 몇 가지 추가 메서드를 추가하기도 합니다.

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)
오직 서브 패키지들의 컨테이너로만 기능하는 패키지. 이름 공간 패키지는 물리적인 실체가 없을 수도 있고, 특히 __init__.py 파일이 없으므로 정규 패키지 와는 다릅니다.

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)
지금은 모든 클래스 객체에 사용되고 있는 클래스 버전의 예전 이름. 초기의 파이썬 버전에서는, 오직 뉴스타일 클래스만 __slots__, 디스크립터, 프라퍼티, __getattribute__(), 클래스 메서드, 스태틱 메서드와 같은 파이썬의 새롭고 다양한 기능들을 사용할 수 있었습니다.

object (객체)
상태 (어트리뷰트나 값) 를 갖고 동작 (메서드) 이 정의된 모든 데이터. 또한, 모든 뉴스타일 클래스의 최종적인 베이스 클래스입니다.

optimized scope (최적화된 스코프)
A scope where target local variable names are reliably known to the compiler when the code is compiled, allowing optimization of read and write access to these names. The local namespaces for functions, generators, coroutines, comprehensions, and generator expressions are optimized in this fashion. Note: most interpreter optimizations are applied to all scopes, only those relying on a known set of local and nonlocal variable names are restricted to optimized scopes.

package (패키지)
서브 모듈들이나, 재귀적으로 서브 패키지들을 포함할 수 있는 파이썬 모듈. 기술적으로, 패키지는 __path__ 어트리뷰트가 있는 파이썬 모듈입니다.

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 엔트리 로 모듈을 찾는 방법을 알고 있습니다.

경로 엔트리 파인더들이 구현하는 메서드들은 `importlib.abc.PathEntryFinder` 에 나옵니다.

path entry hook (경로 엔트리 훅)
`sys.path_hooks` 리스트에 있는 콜러블인데, 특정 경로 엔트리 에서 모듈을 찾는 법을 알고 있다면 경로 엔트리 파인더 를 돌려줍니다.

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 411을 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)
객체에 대한 참조의 개수. 객체의 참조 횟수가 0으로 떨어지면, 메모리가 반납됩니다. 일부 객체는 불멸이며 참조 횟수가 수정되지 않아서, 객체가 할당 해제되지 않습니다. 참조 횟수 추적은 일반적으로 파이썬 코드에 노출되지는 않지만, *CPython* 구현의 핵심 요소입니다. 프로그래머는 특정 객체의 참조 횟수를 돌려주는 `sys.getrefcount()` 함수를 호출할 수 있습니다.

regular package (정규 패키지)
`__init__.py` 파일을 포함하는 디렉터리와 같은 전통적인 패키지.

이름 공간 패키지 도 보세요.

REPL
An acronym for the "read–eval–print loop", another name for the *interactive* interpreter shell.

__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)
`__getitem__()` 특수 메서드를 통해 정수 인덱스를 사용한 빠른 요소 액세스를 지원하고, 시퀀스의 길이를 돌려주는 `__len__()` 메서드를 정의하는 이터러블. 몇몇 내장 시퀀스들을 나열해보면, `list`, `str`, `tuple`, `bytes` 가 있습니다. `dict` 또한 `__getitem__()` 과 `__len__()` 을 지원하지만, 조회에 정수 대신 임의의 해시 가능 키를 사용하기 때문에 시퀀스가 아니라 매핑으로 취급된다는 것에 주의해야 합니다.

`collections.abc.Sequence` 추상 베이스 클래스는 `__getitem__()` 과 `__len__()` 을 넘어서 훨씬 풍부한 인터페이스를 정의하는데, `count()`, `index()`, `__contains__()`, `__reversed__()` 를 추가합니다. 이 확장된 인터페이스를 구현한 형을 `register()` 를 사용해서 명시적으로 등록할 수 있습니다. 시퀀스 메서드 일반에 대한 자세한 문서는, 공통 시퀀스 연산을 참조하세요.

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 (약하게 폐지된)
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 (정적 형 검사기)
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 (강한 참조)
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.

빌린 참조 도 보세요.

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.

thread state
The information used by the *CPython* runtime to run in an OS thread. For example, this includes the current exception, if any, and the state of the bytecode interpreter.

Each thread state is bound to a single OS thread, but threads may have many thread states available. At most, one of them may be *attached* at once.

An *attached thread state* is required to call most of Python' s C API, unless a function explicitly documents otherwise. The bytecode interpreter only runs under an attached thread state.

Each thread state belongs to a single interpreter, but each interpreter may have many thread states, including multiple for the same OS thread. Thread states from multiple interpreters may be bound to the same thread, but only one can be *attached* in that thread at any given moment.

See Thread State and the Global Interpreter Lock for more information.

token (토큰)
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)
파이썬 객체의 형은 그것이 어떤 종류의 객체인지를 결정합니다; 모든 객체는 형이 있습니다. 객체의 형은 `__class__` 어트리뷰트로 액세스할 수 있거나 `type(obj)` 로 얻을 수 있습니다.

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.

형 힌트는 선택 사항이고 파이썬에서 강제되지는 않지만, 정적 형 검사기에 유용합니다. 또한 IDE의 코드 완성 및 리팩토링을 돕습니다.

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.

이 기능을 설명하는 함수 어노테이션, **PEP 484** 및 **PEP 526**을 참조하세요. 또한 어노테이션 작업에 대한 모범 사례는 annotations-howto를 참조하세요.

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

이 설명서에 관하여

파이썬 설명서는 reStructuredText 소스에서 만들어진 것으로, 원래 파이썬을 위해 제작되었고 이제는 독립 프로젝트로 유지 관리되는 설명서 생성기인 Sphinx 를 사용했습니다.

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., 원래 파이썬 설명서 도구 집합의 작성자이자 많은 콘텐츠의 저자;
- reStructuredText와 Docutils 스위트를 만드는 Docutils 프로젝트.
- Fredrik Lundh, 그의 대안 파이썬 참조(Alternative Python Reference) 프로젝트에서 Sphinx가 많은 아이디어를 얻었습니다.

B.1 파이썬 설명서의 공헌자들

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 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

The `uu` codec contains the following notice:

```
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` and `ssl` 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 mimalloc

MIT License:

```
Copyright (c) 2018-2021 Microsoft Corporation, Daan Leijen

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.3.20 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.
```

C.3.21 Global Unbounded Sequences (GUS)

The file `Python/qsbr.c` is adapted from FreeBSD' s "Global Unbounded Sequences" safe memory reclamation scheme in subr_smr.c. The file is distributed under the 2-Clause BSD License:

```
Copyright (c) 2019,2020 Jeffrey Roberson <jeff@FreeBSD.org>

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 unmodified, 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 "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 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.
```

APPENDIX D

저작권

파이썬과 이 설명서는:

전체 라이센스 및 사용 권한 정보는 역사와 라이센스 에서 제공합니다.

색인

알파벳 이외

A

B

C

D

E

F

G

H

I

Q

R

S

T

U

V

W

X

Y

환경 변수

Z