
Python Setup and Usage

출시 버전 3.7.17

Guido van Rossum
and the Python development team

6월 28, 2023

1	명령 줄과 환경	3
1.1	명령 줄	3
1.2	환경 변수	9
2	유닉스 플랫폼에서 파이썬 사용하기	15
2.1	최신 버전의 파이썬 내려받기와 설치	15
2.2	파이썬 빌드하기	16
2.3	파이썬 관련 경로와 파일	16
2.4	잠동사니	17
3	Using Python on Windows	19
3.1	The full installer	20
3.2	The Microsoft Store package	24
3.3	The nuget.org packages	24
3.4	The embeddable package	25
3.5	Alternative bundles	26
3.6	Configuring Python	27
3.7	UTF-8 mode	28
3.8	Python Launcher for Windows	29
3.9	Finding modules	33
3.10	Additional modules	34
3.11	Compiling Python on Windows	35
3.12	Other Platforms	35
4	매킨토시에서 파이썬 사용하기	37
4.1	MacPython을 구하고 설치하기	37
4.2	IDE	38
4.3	추가 파이썬 패키지 설치하기	39
4.4	Mac에서의 GUI 프로그래밍	39
4.5	Mac에서 파이썬 응용 프로그램 배포하기	39
4.6	기타 자원	39
5	Editors and IDEs	41
A	용어집	43
B	이 설명서에 관하여	57

B.1	파이썬 설명서의 공헌자들	57
C	역사와 라이선스	59
C.1	소프트웨어의 역사	59
C.2	파이썬에 액세스하거나 사용하기 위한 이용 약관	60
C.3	포함된 소프트웨어에 대한 라이선스 및 승인	63
D	저작권	77
	색인	79

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

CPython 인터프리터는 명령 줄과 환경에서 다양한 설정을 찾습니다.

CPython implementation detail: 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 [-bBdEhiIOqsSuvVWx?] [-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:

- `tty` 장치에 연결된 표준 입력으로 호출하면, 명령을 입력하라는 프롬프트를 준 후 EOF(파일 끝 문자, 유닉스에서는 `Ctrl-D`, 윈도우에서는 `Ctrl-Z`, `Enter`로 만들 수 있습니다)가 읽힐 때까지 실행합니다.
- 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).

-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
```

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, -i가 묵시적으로 적용됩니다.

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

<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 디렉터리나 `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.

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-config를 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`

모든 명령 줄 옵션에 대한 간단한 설명을 인쇄합니다.

`-V`

`--version`

파이썬 버전 번호를 출력하고 종료합니다. 출력 예는 다음과 같습니다:

```
Python 3.7.0b2+
```

두 번 지정하면, 다음과 같이 빌드에 관한 추가 정보를 인쇄합니다:

```
Python 3.7.0b2+ (3.7:0c076caaa8, Sep 22 2018, 12:04:24)
[GCC 6.2.0 20161005]
```

버전 3.6에 추가: `-VV` 옵션.

1.1.3 기타 옵션

`-b`

`bytes`나 `bytearray`를 `str`과, `bytes`를 `int`와 비교할 때 경고를 합니다. 옵션이 두 번 주어지면 (`-bb`) 에러를 줍니다.

버전 3.5에서 변경: `bytes`와 `int` 비교에도 적용됩니다.

`-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-invalidation`를 참조하세요. default로 설정하면, 검사형과 비검사형 해시 기반 바이트 코드 캐시 파일은 기본 의미에 따라 유효성이 검사됩니다. always로 설정하면, 모든 해시 기반 .pyc 파일들은, 검사형과 비검사형을 가리지 않고, 해당 소스 파일에 대해 유효성이 검사됩니다. never로 설정되면, 해시 기반 .pyc 파일은 해당 소스 파일에 대해 유효성이 검사되지 않습니다.

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

-d

파서 디버깅 출력을 켭니다 (컴파일 옵션에 따라 전문가만을 위한 기능입니다). `PYTHONDEBUG`도 참조하십시오.

-E

설정되었을 수 있는 모든 PYTHON* 환경 변수를 무시합니다, 예를 들어 `PYTHONPATH`와 `PYTHONHOME`.

-i

스크립트가 첫 번째 인자로 전달되거나 `-c` 옵션이 사용되면, `sys.stdin`가 터미널로 보이지 않을 때도, 스크립트나 명령을 실행한 후에 대화형 모드에 진입합니다. `PYTHONSTARTUP` 파일은 읽지 않습니다.

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

-I

격리된 모드로 파이썬을 실행합니다. 이것은 또한 `-E`와 `-s`를 묵시적으로 적용합니다. 격리 모드에서 `sys.path`는 스크립트 디렉터리나 사용자의 `site-packages` 디렉터리를 포함하지 않습니다. 모든 PYTHON* 환경 변수도 무시됩니다. 사용자가 악성 코드를 주입하는 것을 방지하기 위해 추가 제한이 부과될 수 있습니다.

버전 3.4에 추가.

-O

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

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

-OO

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

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

-q

대화형 모드에서도 저작권과 버전 메시지를 표시하지 않습니다.

버전 3.2에 추가.

-R

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

이전 버전의 파이썬에서는, 이 옵션이 해시 무작위화를 켜서, `str`, `bytes`, `datetime`의 `__hash__()` 값이 예측할 수 없는 난수로 “솔트(salt)” 됩니다. 개별 파이썬 프로세스 내에서 상수로 유지되지만, 반복되는 파이썬 실행 간에는 예측할 수 없습니다.

해시 무작위화는 신중하게 선택된 입력으로 디저너리 구성의 성능을 최악의 $O(n^2)$ 복잡도를 갖도록 만드는 서비스 거부에 대한 보호를 제공하기 위한 것입니다. 자세한 내용은 <http://www.ocert.org/advisories/ocert-2011-003.html>을 참조하십시오.

`PYTHONHASHSEED`는 해시 시드 시크릿에 고정값을 설정할 수 있게 합니다.

버전 3.7에서 변경: 이 옵션은 더는 무시되지 않습니다.

버전 3.2.3에 추가.

-s

사용자 `site-packages` 디렉터리 를 `sys.path` 에 추가하지 않습니다.

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

모듈이 초기화될 때마다 메시지를 인쇄하여, 로드된 위치(파일명이나 내장 모듈)를 표시합니다. 두 번 주어지면 (`-vv`), 모듈을 검색할 때 검사되는 각 파일에 대한 메시지를 인쇄합니다. 종료 시 모듈 정리에 대한 정보도 제공합니다. `PYTHONVERBOSE` 도 참조하십시오.

-W arg

경고 제어. 파이썬의 경고 장치는 기본적으로 `sys.stderr` 로 경고 메시지를 인쇄합니다. 일반적인 경고 메시지의 형식은 다음과 같습니다:

```
file:line: category: message
```

기본적으로, 각 경고는 발생하는 각 소스 행마다 한 번씩 인쇄됩니다. 이 옵션은 경고 인쇄 빈도를 제어합니다.

다중 `-W` 옵션이 주어질 수 있습니다; 경고가 두 개 이상의 옵션과 일치하면 마지막으로 일치하는 옵션의 액션이 수행됩니다. 유효하지 않은 `-W` 옵션은 무시됩니다 (하지만, 최초의 경고가 발생할 때 유효하지 않은 옵션에 관한 경고 메시지가 출력됩니다).

경고는 `PYTHONWARNINGS` 환경 변수와 파이썬 프로그램 내에서 `warnings` 모듈을 사용해서 제어할 수도 있습니다.

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
-Wmodule   # Warn once per calling module
-Wonce     # Warn once per Python process
-Wignore   # Never warn
```

액션 이름은 원하면 줄일 수 있고 (예, `-Wi`, `-Wd`, `-Wa`, `-We`), 인터프리터는 이를 적절한 액션 이름으로 해석합니다.

자세한 내용은 `warning-filter` 와 `describing-warning-filters` 를 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` 는 `faulthandler` 를 활성화합니다;
- `-X showrefcount` 는 프로그램이 끝나거나 대화형 인터프리터에서 각각의 문장 뒤에서, 총 참조 카운트와 사용된 메모리 블록의 수를 출력합니다. 이것은 디버그 빌드에서만 작동합니다.
- `-X tracemalloc` 은 `tracemalloc` 모듈을 사용하여 파이썬 메모리 할당 추적을 시작합니다. 기본적으로, 가장 최근 프레임만 추적의 트레이스백에 저장됩니다. `NFRAME` 프레임의 트레이스백 한도로 추적을 시작하려면 `-X tracemalloc=NFRAME` 을 사용하십시오. 자세한 정보는 `tracemalloc.start()` 를 참조하십시오.
- `-X showalloccount` 은 프로그램이 끝날 때 형별로 할당된 객체의 총수를 출력합니다. 이것은 파이썬이 `COUNT_ALLOCS` 이 정의된 상태로 빌드되었을 때만 작동합니다.
- `-X int_max_str_digits` configures the integer string conversion length limitation. See also [PYTHONINTMAXSTRDIGITS](#).
- `-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.
- `-X dev`: CPython의 “개발 모드”를 활성화해서, 기본적으로 활성화하기에는 너무 비싼 추가적인 실행시간 검사를 도입합니다. 코드가 올바르면 기본값보다 더 상세하지는 않아야 합니다: 문제가 감지되었을 때만 새로운 경고가 발생합니다. 개발자 모드의 효과는 다음과 같습니다:
 - `-W default` 처럼, `default` 경고 필터를 추가합니다.
 - 메모리 할당자에 디버그 훅을 설치합니다: `PyMem_SetupDebugHooks()` C 함수를 참조하십시오.
 - 충돌 시에 파이썬 트레이스백을 덤프하도록 `faulthandler` 모듈을 활성화합니다.
 - `asyncio` 디버그 모드 를 활성화합니다.
 - Set the `dev_mode` attribute of `sys.flags` to `True`.
- `-X utf8` 은 운영 체제 인터페이스를 위한 UTF-8 모드를 활성화하여, 기본 로케일 인식 모드를 대체합니다. `-X utf8=0` 은 명시적으로 UTF-8 모드를 비활성화합니다(그렇지 않으면 자동으로 활성화될 때조차). 자세한 내용은 [PYTHONUTF8](#) 을 참조하십시오.

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

버전 3.2에서 변경: `-X` 옵션이 추가되었습니다.

버전 3.3에 추가: `-X faulthandler` 옵션.

버전 3.4에 추가: `-X showrefcount` 와 `-X tracemalloc` 옵션.

버전 3.6에 추가: `-X showalloccount` 옵션.

버전 3.7에 추가: `-X importtime`, `-X dev`, `-X utf8` 옵션.

버전 3.7.14에 추가: The `-X int_max_str_digits` option.

1.1.4 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 PATH와 같습니다: 하나 이상의 디렉터리 경로 명이 `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.

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 혹은 `sys.__interactivehook__`도 바꿀 수 있습니다.

PYTHONOPTIMIZE

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

PYTHONBREAKPOINT

설정되면, 점으로 구분된 경로 표기법을 사용하여 콜러블의 이름을 지정합니다. 콜러블을 포함하는 모듈이 임포트된 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.

버전 3.7에 추가.

PYTHONDEBUG

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

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.

PYTHONUNBUFFERED

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

PYTHONVERBOSE

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

PYTHONCASEOK

설정되면, 파이썬은 `import` 문에서 대소 문자를 무시합니다. 이것은 윈도우와 OS X에서만 작동합니다.

PYTHONDONTWRITEBYTECODE

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

PYTHONHASHSEED

이 변수가 설정되어 있지 않거나 `random` 으로 설정되면, `str`, `bytes`, `datetime` 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.

버전 3.2.3에 추가.

PYTHONINTMAXSTRDIGITS

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

버전 3.7.14에 추가.

PYTHONIOENCODING

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

`stderr`의 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` 디렉터리 를 `sys.path` 에 추가하지 않습니다.

더 보기:

PEP 370 – 사용자별 `site-packages` 디렉터리

PYTHONUSERBASE

`python setup.py install --user` 에서 사용자 `site-packages` 디렉터리 의 경로와 `Distutils` 설치 경로를 계산하기 위해 사용되는 사용자 베이스 디렉터리 를 정의합니다.

더 보기:

PEP 370 – 사용자별 `site-packages` 디렉터리

PYTHONEXECUTABLE

이 환경 변수가 설정되면, `sys.argv[0]` 는 C 런타임을 통해 얻은 값 대신에 이 값으로 설정됩니다. 맥 OS X에서만 작동합니다.

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=module  # Warn once per calling module
PYTHONWARNINGS=once    # Warn once per Python process
PYTHONWARNINGS=ignore  # Never warn
```

자세한 내용은 `warning-filter`와 `describing-warning-filters`를 참조하십시오.

PYTHONFAULTHANDLER

이 환경 변수가 비어 있지 않은 문자열로 설정되면, `faulthandler.enable()` 이 시작 시에 호출됩니다: 파이썬 트레이스백을 덤프하는 `SIGSEGV`, `SIGFPE`, `SIGABRT`, `SIGBUS` 그리고 `SIGILL` 시그널 처리기를 설치합니다. 이는 `-X faulthandler` 옵션과 동등합니다.

버전 3.3에 추가.

PYTHONTRACEMALLOC

이 환경 변수가 비어 있지 않은 문자열로 설정되면, `tracemalloc` 모듈을 사용하여 파이썬 메모리 할당 추적을 시작합니다. 변수의 값은 추적의 트레이스백에 저장되는 최대 프레임 수입니다. 예를 들어, `PYTHONTRACEMALLOC=1` 은 가장 최근의 프레임만을 저장합니다. 자세한 정보는 `tracemalloc.start()` 를 참조하십시오.

버전 3.4에 추가.

PYTHONPROFILEIMPORTTIME

이 환경 변수가 비어 있지 않은 문자열로 설정되면, 파이썬은 각 임포트에 걸리는 시간을 보여줍니다. 이는 명령 줄에서 `-X importtime` 을 설정하는 것과 정확히 같습니다.

버전 3.7에 추가.

PYTHONASYNCIODEBUG

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

버전 3.4에 추가.

PYTHONMALLOC

파이썬 메모리 할당자를 설정하거나 디버그 훅을 설치합니다.

파이썬이 사용하는 메모리 할당자를 설정합니다:

- `default`: 기본 메모리 할당자를 사용합니다.
- `malloc`: 모든 영역(`PYMEM_DOMAIN_RAW`, `PYMEM_DOMAIN_MEM`, `PYMEM_DOMAIN_OBJ`)에서 C 라이브러리의 `malloc()` 함수를 사용합니다.
- `pymalloc`: `PYMEM_DOMAIN_MEM`과 `PYMEM_DOMAIN_OBJ` 영역에서 `pymalloc` 할당자를 사용하고, `PYMEM_DOMAIN_RAW` 영역에서 `malloc()` 함수를 사용합니다.

디버그 훅을 설치합니다:

- `debug`: 기본 메모리 할당자 위에 디버그 훅을 설치합니다.
- `malloc_debug`: same as `malloc` but also install debug hooks.

- `pymalloc_debug`: same as `pymalloc` but also install debug hooks.

기본 메모리 할당자와 `PyMem_SetupDebugHooks()` 함수(파이썬 메모리 할당자에 디버그 hooks 설치합니다)도 참조하세요.

버전 3.7에서 변경: "default" 할당자를 추가했습니다.

버전 3.6에 추가.

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

비어 있지 않은 문자열로 설정하면, 기본 파일 시스템 인코딩과 에러 모드를 3.6 이전의 값인 'mbcs'와 'replace'로 각각 되돌립니다. 그렇지 않으면, 새 기본값 'utf-8'과 'surrogatepass'가 사용됩니다.

이것은 또한 실행 시간에 `sys._enablelegacywindowsfsencoding()` 으로 활성화될 수 있습니다.

가용성: 윈도우.

버전 3.6에 추가: 자세한 내용은 [PEP 529](#)를 참조하십시오.

PYTHONLEGACYWINDOWSSSTDIO

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 파이프) 경우 무시됩니다.

가용성: 윈도우.

버전 3.6에 추가.

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 활성화 상태면 파이썬은 `stderr` 로 경고 메시지를 보냅니다.

또한, 로케일 강제 변경이 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: 유닉스.

버전 3.7에 추가: 자세한 내용은 [PEP 538](#)을 참조하십시오.

PYTHONDEVMODE

이 환경 변수가 비어 있지 않은 문자열로 설정되면, CPython “개발 모드”를 활성화합니다. `-Xdev` 옵션을 보십시오.

버전 3.7에 추가.

PYTHONUTF8

1 로 설정하면, 인터프리터의 UTF-8 모드가 활성화됩니다. 이 모드에서는 현재 로케일 설정에 상관없이, 시스템 인터페이스의 텍스트 인코딩으로 UTF-8 이 사용됩니다.

이는 다음을 의미합니다:

- `sys.getfilesystemencoding()` 은 'UTF-8' 을 반환합니다(로케일 인코딩은 무시됩니다).
- `locale.getpreferredencoding()` 은 'UTF-8' 을 반환합니다(로케일 인코딩은 무시되고, 함수의 `do_setlocale` 매개 변수는 아무런 효과가 없습니다).
- `sys.stdin`, `sys.stdout`, `sys.stderr` 모두 텍스트 인코딩으로 UTF-8을 사용하고, `sys.stdin` 과 `sys.stdout` 은 `surrogateescape` 에러 처리기 가 활성화됩니다(`sys.stderr` 은 기본 로케일 인식 모드와 같이 `backslashreplace` 를 계속 사용합니다)

이러한 하위 수준 API의 변경으로 인해, 다른 상위 수준 API도 다른 기본 동작을 수행합니다:

- 명령 줄 인자, 환경 변수 및 파일명은 UTF-8 인코딩을 사용하여 텍스트로 디코딩됩니다.
- `os.fsdecode()` 와 `os.fsencode()` 는 UTF-8 인코딩을 사용합니다.
- `open()`, `io.open()`, `codecs.open()` 은 기본적으로 UTF-8 인코딩을 사용합니다. 그러나, 이들은 기본적으로 여전히 `strict` 에러 처리기를 사용하므로, 바이너리 파일을 텍스트 모드로 열려고 하면 말도 안 되는 데이터가 생성되는 대신 예외가 발생합니다.

UTF-8 모드에서 표준 스트림 설정은 `PYTHONIOENCODING` 의해 대체될 수 있습니다(기본 로케일 인식 모드에서와 마찬가지로).

0 으로 설정하면, 인터프리터는 기본 로케일 인식 모드로 실행됩니다.

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

이 환경 변수가 전혀 설정되지 않으면, 인터프리터는 현재 로케일이 레거시 ASCII 기반 로케일(`PYTHONCOERCECLOCALE` 에서 설명하는 것처럼)로 식별되고, 로케일 강제 변경이 비활성화되거나 실패하지 않는 한, 인터프리터는 현재 로케일 설정을 사용합니다. 이러한 레거시 로케일에서, 명시적으로 그렇게 하지 말라고 지시하지 않는 한, 인터프리터는 기본적으로 UTF-8 모드를 활성화합니다.

`-Xutf8` 옵션으로도 사용 가능합니다.

버전 3.7에 추가: 자세한 내용은 [PEP 540](#)을 참조하십시오.

1.2.1 디버그 모드 변수

이 변수들을 설정하는 것은 파이썬의 디버그 빌드에서만 효과가 있습니다. 즉, 파이썬이 `--with-pydebug` 빌드 옵션으로 구성되었어야 합니다.

PYTHONTHREADDEBUG

설정되면, 파이썬은 스레딩 디버그 정보를 인쇄합니다.

PYTHONDUMPPREFS

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

유닉스 플랫폼에서 파이썬 사용하기

2.1 최신 버전의 파이썬 내려받기와 설치

2.1.1 리눅스

파이썬은 대부분 리눅스 배포판에 사전 설치되어 있으며, 다른 모든 곳에서 패키지로 사용할 수 있습니다. 그러나 배포판 패키지에 없는 어떤 기능을 사용하고 싶을 수 있습니다. 소스에서 최신 버전의 파이썬을 쉽게 컴파일할 수 있습니다.

파이썬이 미리 설치되어 있지 않고 저장소에도 없으면, 여러분 자신의 배포를 위한 패키지를 쉽게 만들 수 있습니다. 다음 링크를 살펴보십시오:

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-old.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/ch-creating-rpms.html
Fedora 사용자용

<http://www.slackbook.org/html/package-management-making-packages.html> Slackware 사용자용

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
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

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
```

2.1.3 OpenSolaris

OpenCSW에서 파이썬을 얻을 수 있습니다. 다양한 버전의 파이썬이 있으며, 예를 들어 `pkgutil -i python27`로 설치할 수 있습니다.

2.2 파이썬 빌드하기

CPython을 직접 컴파일하려면, 먼저 소스를 얻습니다. 최신 버전의 소스를 내려받거나 새로 clone 할 수 있습니다. (패치에 이바지하려면 clone이 필요합니다.)

빌드 프로세스는 일반적으로 다음과 같은 호출로 구성됩니다

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

특정 유닉스 플랫폼에 대한 구성 옵션과 주의 사항은 파이썬 소스 트리의 루트에 있는 `README.rst` 파일에 광범위하게 설명되어 있습니다.

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

2.3 파이썬 관련 경로와 파일

이는 지역 설치 규칙에 따라 달라질 수 있습니다; `prefix` (`${prefix}`) 와 `exec_prefix` (`${exec_prefix}`)는 설치에 따라 다르며 GNU 소프트웨어처럼 해석되어야 합니다; 이것들은 같을 수도 있습니다.

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

파일/디렉터리	의미
<code>exec_prefix/bin/python3</code>	인터프리터의 권장 위치.
<code>prefix/lib/pythonversion,</code> <code>exec_prefix/lib/pythonversion</code>	표준 모듈을 포함하는 디렉터리의 권장 위치.
<code>prefix/include/pythonversion,</code> <code>exec_prefix/include/</code> <code>pythonversion</code>	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/python3`를 하드 코딩해야 할 수 있습니다.

파이썬 스크립트에서 셸 명령을 사용하려면, `subprocess` 모듈을 보십시오.

Using Python on Windows

This document aims to give an overview of Windows-specific behaviour you should know about when using Python on Microsoft Windows.

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 (MSI packages) 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 3.7 supports Windows Vista and newer. If you require Windows XP support then please install Python 3.4.

There are a number of different installers available for Windows, each with certain benefits and downsides.

The full installer contains all components and is the best option for developers using Python for any kind of project.

The Microsoft Store package 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, but can be safely installed without corrupting other programs. It also provides many convenient commands for launching Python and its tools.

The nuget.org packages are lightweight installations intended for continuous integration systems. It can be used to build Python packages or run scripts, but is not updateable and has no user interface tools.

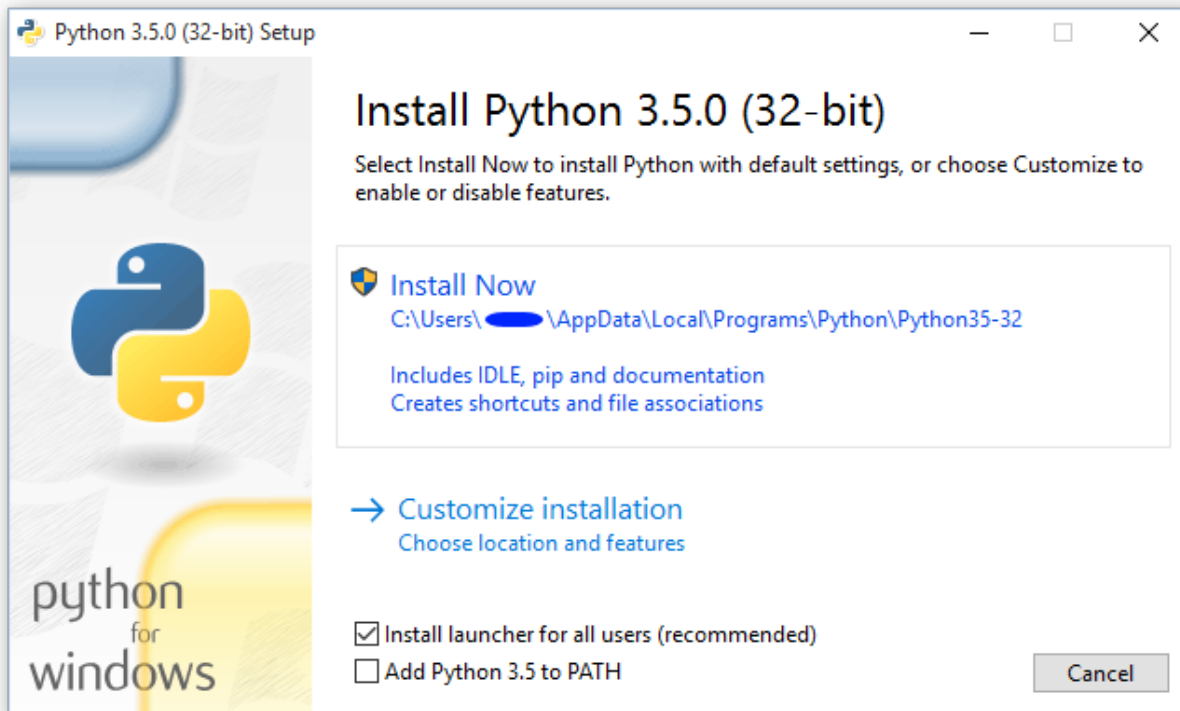
The embeddable package is a minimal package of Python suitable for embedding into a larger application.

3.1 The full installer

3.1.1 Installation steps

Four Python 3.7 installers are available for download - two each for the 32-bit and 64-bit versions of the interpreter. The *web installer* is a small initial download, and it will automatically download the required components as necessary. The *offline installer* includes the components necessary for a default installation and only requires an internet connection for optional features. See *Installing Without Downloading* for other ways to avoid downloading during installation.

After starting the installer, one of two options may be selected:



If you select “Install Now”:

- You will *not* need to be an administrator (unless a system update for the C Runtime Library is required or you install the *Python Launcher for Windows* for all users)
- Python will be installed into your user directory
- The *Python Launcher for Windows* will be installed according to the option at the bottom of the first page
- The standard library, test suite, launcher and pip will be installed
- If selected, the install directory will be added to your PATH
- Shortcuts will only be visible for the current user

Selecting “Customize installation” will allow you to select the features to install, the installation location and other options or post-install actions. To install debugging symbols or binaries, you will need to use this option.

To perform an all-users installation, you should select “Customize installation”. In this case:

- You may be required to provide administrative credentials or approval
- Python will be installed into the Program Files directory

- The *Python Launcher for Windows* will be installed into the Windows directory
- Optional features may be selected during installation
- The standard library can be pre-compiled to bytecode
- If selected, the install directory will be added to the system PATH
- Shortcuts are available for all users

3.1.2 Removing the MAX_PATH Limitation

Windows historically has limited path lengths to 260 characters. This meant that paths longer than this would not resolve and errors would result.

In the latest versions of Windows, this limitation can be expanded to approximately 32,000 characters. Your administrator will need to activate the “Enable Win32 long paths” group policy, or set the registry value `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem@LongPathsEnabled` to 1.

This allows the `open()` function, the `os` module and most other path functionality to accept and return paths longer than 260 characters.

After changing the above option, no further configuration is required.

버전 3.6에서 변경: Support for long paths was enabled in Python.

3.1.3 Installing Without UI

All of the options available in the installer UI can also be specified from the command line, allowing scripted installers to replicate an installation on many machines without user interaction. These options may also be set without suppressing the UI in order to change some of the defaults.

To completely hide the installer UI and install Python silently, pass the `/quiet` option. To skip past the user interaction but still display progress and errors, pass the `/passive` option. The `/uninstall` option may be passed to immediately begin removing Python - no prompt will be displayed.

All other options are passed as `name=value`, where the value is usually 0 to disable a feature, 1 to enable a feature, or a path. The full list of available options is shown below.

Name	Description	Default
InstallAllUsers	Perform a system-wide installation.	0
TargetDir	The installation directory	Selected based on InstallAllUsers
DefaultAllUsersTargetDir	The default installation directory for all-user installs	%ProgramFiles%\Python X.Y or %ProgramFiles(x86)%\Python X.Y
Default-Just-ForMeTargetDir	The default install directory for just-for-me installs	%LocalAppData%\Programs\PythonXY or %LocalAppData%\Programs\PythonXY-32 or %LocalAppData%\Programs\PythonXY-64
Default-Custom-TargetDir	The default custom install directory displayed in the UI	(empty)
Associate-Files	Create file associations if the launcher is also installed.	1
CompileAll	Compile all .py files to .pyc.	0
Prepend-Path	Add install and Scripts directories to PATH and .PY to PATHEXT	0
Shortcuts	Create shortcuts for the interpreter, documentation and IDLE if installed.	1
Include_doc	Install Python manual	1
Include_debug	Install debug binaries	0
Include_dev	Install developer headers and libraries	1
Include_exe	Install python.exe and related files	1
Include_launcher	Install <i>Python Launcher for Windows</i> .	1
Install-Launcher-AllUsers	Installs <i>Python Launcher for Windows</i> for all users.	1
Include_lib	Install standard library and extension modules	1
Include_pip	Install bundled pip and setup-tools	1
Include_symbols	Install debugging symbols (*.pdb)	0
Include_tcltk	Install Tcl/Tk support and IDLE	1
Include_test	Install standard library test suite	1
Include_tools	Install utility scripts	1
LauncherOnly	Only installs the launcher. This will override most other options.	0
SimpleInstall	Disable most install UI	0
SimpleInstallDescription	A custom message to display when the simplified install UI is used.	(empty)

For example, to silently install a default, system-wide Python installation, you could use the following command (from an elevated command prompt):

```
python-3.7.0.exe /quiet InstallAllUsers=1 PrependPath=1 Include_test=0
```

To allow users to easily install a personal copy of Python without the test suite, you could provide a shortcut with the following command. This will display a simplified initial page and disallow customization:

```
python-3.7.0.exe InstallAllUsers=0 Include_launcher=0 Include_test=0
SimpleInstall=1 SimpleInstallDescription="Just for me, no test suite."
```

(Note that omitting the launcher also omits file associations, and is only recommended for per-user installs when there is also a system-wide installation that included the launcher.)

The options listed above can also be provided in a file named `unattend.xml` alongside the executable. This file specifies a list of options and values. When a value is provided as an attribute, it will be converted to a number if possible. Values provided as element text are always left as strings. This example file sets the same options as the previous example:

```
<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>
```

3.1.4 Installing Without Downloading

As some features of Python are not included in the initial installer download, selecting those features may require an internet connection. To avoid this need, all possible components may be downloaded on-demand to create a complete *layout* that will no longer require an internet connection regardless of the selected features. Note that this download may be bigger than required, but where a large number of installations are going to be performed it is very useful to have a locally cached copy.

Execute the following command from Command Prompt to download all possible required files. Remember to substitute `python-3.7.0.exe` for the actual name of your installer, and to create layouts in their own directories to avoid collisions between files with the same name.

```
python-3.7.0.exe /layout [optional target directory]
```

You may also specify the `/quiet` option to hide the progress display.

3.1.5 Modifying an install

Once Python has been installed, you can add or remove features through the Programs and Features tool that is part of Windows. Select the Python entry and choose “Uninstall/Change” to open the installer in maintenance mode.

“Modify” allows you to add or remove features by modifying the checkboxes - unchanged checkboxes will not install or remove anything. Some options cannot be changed in this mode, such as the install directory; to modify these, you will need to remove and then reinstall Python completely.

“Repair” will verify all the files that should be installed using the current settings and replace any that have been removed or modified.

“Uninstall” will remove Python entirely, with the exception of the *Python Launcher for Windows*, which has its own entry in Programs and Features.

3.2 The Microsoft Store package

버전 3.7.2에 추가.

참고: The Microsoft Store package is currently considered unstable while its interactions with other tools and other copies of Python are evaluated. While Python itself is stable, this installation method may change its behavior and capabilities during Python 3.7 releases.

The Microsoft Store package is an easily installable Python interpreter that is intended mainly for interactive use, for example, by students.

To install the package, ensure you have the latest Windows 10 updates and search the Microsoft Store app for “Python 3.7”. Ensure that the app you select is published by the Python Software Foundation, and install it.

경고: Python will always be available for free on the Microsoft Store. If you are asked to pay for it, you have not selected the correct package.

After installation, Python may be launched by finding it in Start. Alternatively, it will be available from any Command Prompt or PowerShell session by typing `python`. Further, `pip` and `IDLE` may be used by typing `pip` or `idle`. `IDLE` can also be found in Start.

All three commands are also available with version number suffixes, for example, as `python3.exe` and `python3.x.exe` as well as `python.exe` (where `3.x` is the specific version you want to launch, such as 3.7).

Virtual environments can be created with `python -m venv` and activated and used as normal.

If you have installed another version of Python and added it to your `PATH` variable, it will be available as `python.exe` rather than the one from the Microsoft Store. To access the new installation, use `python3.exe` or `python3.x.exe`.

To remove Python, open Settings and use Apps and Features, or else find Python in Start and right-click to select Uninstall. Uninstalling will remove all packages you installed directly into this Python installation, but will not remove any virtual environments

3.2.1 Known Issues

Currently, the `py.exe` launcher cannot be used to start Python when it has been installed from the Microsoft Store.

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.

3.3 The nuget.org packages

버전 3.5.2에 추가.

The `nuget.org` package is a reduced size Python environment intended for use on continuous integration and build systems that do not have a system-wide install of Python. While `nuget` is “the package manager for .NET”, it also works perfectly fine for packages containing build-time tools.

Visit nuget.org for the most up-to-date information on using `nuget`. What follows is a summary that is sufficient for Python developers.

The `nuget.exe` command line tool may be downloaded directly from <https://aka.ms/nugetclidl>, for example, using `curl` or `PowerShell`. With the tool, the latest version of Python for 64-bit or 32-bit machines is installed using:

```
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
```

In general, `nuget` packages are not upgradeable, and newer versions should be installed side-by-side and referenced using the full path. Alternatively, delete the package directory manually and install it again. Many CI systems will do this automatically if they do not preserve files between builds.

Alongside the `tools` directory is a `build\native` directory. This contains a MSBuild properties file `python.props` that can be used in a C++ project to reference the Python install. Including the settings will automatically use the headers and import libraries in your build.

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

3.4 The embeddable package

버전 3.5에 추가.

The embedded distribution is a ZIP file containing a minimal Python environment. It is intended for acting as part of another application, rather than being directly accessed by end-users.

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 dependants, 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.

Third-party packages should be installed by the application installer alongside the embedded distribution. Using `pip` to manage dependencies as for a regular Python installation is not supported with this distribution, though with some care it may be possible to include and use `pip` for automatic updates. In general, third-party packages should be treated as part of the application ("vendoring") so that the developer can ensure compatibility with newer versions before providing updates to users.

The two recommended use cases for this distribution are described below.

3.4.1 Python Application

An application written in Python does not necessarily require users to be aware of that fact. The embedded distribution may be used in this case to include a private version of Python in an install package. Depending on how transparent it should be (or conversely, how professional it should appear), there are two options.

Using a specialized executable as a launcher requires some coding, but provides the most transparent experience for users. With a customized launcher, there are no obvious indications that the program is running on Python: icons can be customized, company and version information can be specified, and file associations behave properly. In most cases, a custom launcher should simply be able to call `Py_Main` with a hard-coded command line.

The simpler approach is to provide a batch file or generated shortcut that directly calls the `python.exe` or `pythonw.exe` with the required command-line arguments. In this case, the application will appear to be Python and not its actual name, and users may have trouble distinguishing it from other running Python processes or file associations.

With the latter approach, packages should be installed as directories alongside the Python executable to ensure they are available on the path. With the specialized launcher, packages can be located in other locations as there is an opportunity to specify the search path before launching the application.

3.4.2 Embedding Python

Applications written in native code often require some form of scripting language, and the embedded Python distribution can be used for this purpose. In general, the majority of the application is in native code, and some part will either invoke `python.exe` or directly use `python3.dll`. For either case, extracting the embedded distribution to a subdirectory of the application installation is sufficient to provide a loadable Python interpreter.

As with the application use, packages can be installed to any location as there is an opportunity to specify search paths before initializing the interpreter. Otherwise, there is no fundamental differences between using the embedded distribution and a regular installation.

3.5 Alternative bundles

Besides the standard CPython distribution, there are modified packages including additional functionality. The following is a list of popular versions and their key features:

ActivePython Installer with multi-platform compatibility, documentation, PyWin32

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

Canopy A “comprehensive Python analysis environment” with editors and other development tools.

WinPython Windows-specific distribution with prebuilt scientific packages and tools for building packages.

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

3.6 Configuring Python

To run Python conveniently from a command prompt, you might consider changing some default environment variables in Windows. While the installer provides an option to configure the `PATH` and `PATHEXT` variables for you, this is only reliable for a single, system-wide installation. If you regularly use multiple versions of Python, consider using the *Python Launcher for Windows*.

3.6.1 Excursus: Setting environment variables

Windows allows environment variables to be configured permanently at both the User level and the System level, or temporarily in a command prompt.

To temporarily set environment variables, open Command Prompt and use the `set` command:

```
C:\>set PATH=C:\Program Files\Python 3.7;%PATH%
C:\>set PYTHONPATH=%PYTHONPATH%;C:\My_python_lib
C:\>python
```

These changes will apply to any further commands executed in that console, and will be inherited by any applications started from the console.

Including the variable name within percent signs will expand to the existing value, allowing you to add your new value at either the start or the end. Modifying `PATH` by adding the directory containing `python.exe` to the start is a common way to ensure the correct version of Python is launched.

To permanently modify the default environment variables, click Start and search for ‘edit environment variables’, or open System properties, *Advanced system settings* and click the *Environment Variables* button. In this dialog, you can add or modify User and System variables. To change System variables, you need non-restricted access to your machine (i.e. Administrator rights).

참고: Windows will concatenate User variables *after* System variables, which may cause unexpected results when modifying `PATH`.

The `PYTHONPATH` variable is used by all versions of Python 2 and Python 3, so you should not permanently configure this variable unless it only includes code that is compatible with all of your installed Python versions.

더 보기:

<https://www.microsoft.com/en-us/wdsi/help/folder-variables> Environment variables in Windows NT

<https://technet.microsoft.com/en-us/library/cc754250.aspx> The SET command, for temporarily modifying environment variables

<https://technet.microsoft.com/en-us/library/cc755104.aspx> The SETX command, for permanently modifying environment variables

<https://support.microsoft.com/en-us/help/310519/how-to-manage-environment-variables-in-windows-xp> How To Manage Environment Variables in Windows XP

<https://www.chem.gla.ac.uk/~louis/software/faq/q1.html> Setting Environment variables, Louis J. Farrugia

3.6.2 Finding the Python executable

버전 3.5에서 변경.

Besides using the automatically created start menu entry for the Python interpreter, you might want to start Python in the command prompt. The installer has an option to set that up for you.

On the first page of the installer, an option labelled “Add Python to PATH” may be selected to have the installer add the install location into the `PATH`. The location of the `Scripts\` folder is also added. This allows you to type `python` to run the interpreter, and `pip` for the package installer. Thus, you can also execute your scripts with command line options, see [명령 줄 documentation](#).

If you don’t enable this option at install time, you can always re-run the installer, select Modify, and enable it. Alternatively, you can manually modify the `PATH` using the directions in [Excursus: Setting environment variables](#). You need to set your `PATH` environment variable to include the directory of your Python installation, delimited by a semicolon from other entries. An example variable could look like this (assuming the first two entries already existed):

`C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files\Python 3.7`

3.7 UTF-8 mode

버전 3.7에 추가.

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.getpreferredencoding()`).

This may cause issues because UTF-8 is widely used on the internet and most Unix systems, including WSL (Windows Subsystem for Linux).

You can use UTF-8 mode to change the default text encoding to UTF-8. You can enable UTF-8 mode via the `-X utf8` command line option, or the `PYTHONUTF8=1` environment variable. See [PYTHONUTF8](#) for enabling UTF-8 mode, and [Excursus: Setting environment variables](#) for how to modify environment variables.

When UTF-8 mode is enabled:

- `locale.getpreferredencoding()` returns 'UTF-8' instead of the system encoding. This function is used for the default text encoding in many places, including `open()`, `Popen`, `Path.read_text()`, etc.
- `sys.stdin`, `sys.stdout`, and `sys.stderr` all use UTF-8 as their text encoding.
- You can still use the system encoding via the “mbcs” codec.

Note that adding `PYTHONUTF8=1` to the default environment variables will affect all Python 3.7+ applications on your system. If you have any Python 3.7+ applications which rely on the legacy system encoding, it is recommended to set the environment variable temporarily or use the `-X utf8` command line option.

참고: Even when UTF-8 mode is disabled, Python uses UTF-8 by default on Windows for:

- Console I/O including standard I/O (see [PEP 528](#) for details).
 - The filesystem encoding (see [PEP 529](#) for details).
-

3.8 Python Launcher for Windows

버전 3.3에 추가.

The Python launcher for Windows is a utility which aids in locating and executing of different Python versions. It allows scripts (or the command-line) to indicate a preference for a specific Python version, and will locate and execute that version.

Unlike the `PATH` variable, the launcher will correctly select the most appropriate version of Python. It will prefer per-user installations over system-wide ones, and orders by language version rather than using the most recently installed version.

The launcher was originally specified in [PEP 397](#).

3.8.1 Getting started

From the command-line

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
```

You should find that the latest version of Python you have installed is started - it can be exited as normal, and any additional command-line arguments specified will be sent directly to Python.

If you have multiple versions of Python installed (e.g., 2.7 and 3.7) you will have noticed that Python 3.7 was started - to launch Python 2.7, try the command:

```
py -2.7
```

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

```
py -2
```

You should find the latest version of Python 2.x starts.

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.
```

Per-user installations of Python do not add the launcher to `PATH` unless the option was selected on installation.

Virtual environments

버전 3.5에 추가.

If the launcher is run with no explicit Python version specification, and a virtual environment (created with the standard library `venv` module or the external `virtualenv` tool) active, the launcher will run the virtual environment's interpreter rather than the global one. To run the global interpreter, either deactivate the virtual environment, or explicitly specify the global Python version.

From a script

Let's create a test Python script - create a file called `hello.py` with the following contents

```
#!/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
```

You should notice the version number of your latest Python 2.x installation is printed. Now try changing the first line to be:

```
#!/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 2.6 installed, try changing the first line to `#!/python2.6` and you should find the 2.6 version information printed.

Note that unlike interactive use, a bare “python” will use the latest version of Python 2.x that you have installed. This is for backward compatibility and for compatibility with Unix, where the command `python` typically refers to Python 2.

From file associations

The launcher should have been associated with Python files (i.e. `.py`, `.pyw`, `.pyc` files) when it was installed. This means that when you double-click on one of these files from Windows explorer the launcher will be used, and therefore you can use the same facilities described above to have the script specify the version which should be used.

The key benefit of this is that a single launcher can support multiple Python versions at the same time depending on the contents of the first line.

3.8.2 Shebang Lines

If the first line of a script file starts with `#!`, it is known as a “shebang” line. Linux and other Unix like operating systems have native support for such lines and they are commonly used on such systems to indicate how a script should be executed. This launcher allows the same facilities to be used with Python scripts on Windows and the examples above demonstrate their use.

To allow shebang lines in Python scripts to be portable between Unix and Windows, this launcher supports a number of ‘virtual’ commands to specify which interpreter to use. The supported virtual commands are:

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

- /usr/local/bin/python
- python

For example, if the first line of your script starts with

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

The default Python 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/python2.7-32 will request usage of the 32-bit python 2.7.

버전 3.7에 추가: Beginning with python launcher 3.7 it is possible to request 64-bit version by the “-64” suffix. Furthermore it is possible to specify a major and architecture without minor (i.e. /usr/bin/python3-64).

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. This corresponds to the behaviour of the Unix env program, which performs a PATH search.

3.8.3 Arguments in shebang lines

The shebang lines can also specify additional options to be passed to the Python interpreter. For example, if you have a shebang line:

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

Then Python will be started with the -v option

3.8.4 Customization

Customization via INI files

Two .ini files will be searched by the launcher - py.ini in the current user’s “application data” directory (i.e. the directory returned by calling the Windows function SHGetFolderPath with CSIDL_LOCAL_APPDATA) 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).

Customization specified in the “application directory” will have precedence over the one next to the executable, so a user, who may not have write access to the .ini file next to the launcher, can override commands in that global .ini file.

Customizing default Python versions

In some cases, a version qualifier can be included in a command to dictate which version of Python will be used by the command. A version qualifier starts with a major version number and can optionally be followed by a period (‘.’) and a minor version specifier. Furthermore it is possible to specify if a 32 or 64 bit implementation shall be requested by adding “-32” or “-64”.

For example, a shebang line of #!python has no version qualifier, while #!python3 has a version qualifier which specifies only a major version.

If no version qualifiers are found in a command, the environment variable PY_PYTHON can be set to specify the default version qualifier. If it is not set, the default is “3”. The variable can specify any value that may be passed on the command

line, such as “3”, “3.7”, “3.7-32” or “3.7-64”. (Note that the “-64” option is only available with the launcher included with Python 3.7 or newer.)

If no minor version qualifiers are found, the environment variable `PY_PYTHON{major}` (where `{major}` is the current major version qualifier as determined above) can be set to specify the full version. If no such option is found, the launcher will enumerate the installed Python versions and use the latest minor release found for the major version, which is likely, although not guaranteed, to be the most recently installed version in that family.

On 64-bit Windows with both 32-bit and 64-bit implementations of the same (major.minor) Python version installed, the 64-bit version will always be preferred. This will be true for both 32-bit and 64-bit implementations of the launcher - a 32-bit launcher will prefer to execute a 64-bit Python installation of the specified version if available. This is so the behavior of the launcher can be predicted knowing only what versions are installed on the PC and without regard to the order in which they were installed (i.e., without knowing whether a 32 or 64-bit version of Python and corresponding launcher was installed last). As noted above, an optional “-32” or “-64” suffix can be used on a version specifier to change this behaviour.

Examples:

- If no relevant options are set, the commands `python` and `python2` will use the latest Python 2.x version installed and the command `python3` will use the latest Python 3.x installed.
- The commands `python3.1` and `python2.7` will not consult any options at all as the versions are fully specified.
- If `PY_PYTHON=3`, the commands `python` and `python3` will both use the latest installed Python 3 version.
- If `PY_PYTHON=3.1-32`, the command `python` will use the 32-bit implementation of 3.1 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=3.1`, the commands `python` and `python3` will both use specifically 3.1

In addition to environment variables, the same settings can be configured in the .INI file used by the launcher. The section in the INI file is called `[defaults]` and the key name will be the same as the environment variables without the leading `PY_` prefix (and note that the key names in the INI file are case insensitive.) The contents of an environment variable will override things specified in the INI file.

For example:

- Setting `PY_PYTHON=3.1` is equivalent to the INI file containing:

```
[defaults]
python=3.1
```

- Setting `PY_PYTHON=3` and `PY_PYTHON3=3.1` is equivalent to the INI file containing:

```
[defaults]
python=3
python3=3.1
```

3.8.5 Diagnostics

If an environment variable `PYLAUNCH_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.

3.9 Finding modules

Python usually stores its library (and thereby your site-packages folder) in the installation directory. So, if you had installed Python to `C:\Python\`, the default library would reside in `C:\Python\Lib\` and third-party modules should be stored in `C:\Python\Lib\site-packages\`.

To completely override `sys.path`, create a `._pth` file with the same name as the DLL (`python37._pth`) or the executable (`python._pth`) and specify one line for each path to add to `sys.path`. The file based on the DLL name overrides the one based on the executable, which allows paths to be restricted for any program loading the runtime if desired.

When the file exists, all registry and environment variables are ignored, isolated mode is enabled, and `site` is not imported unless one line in the file specifies `import site`. Blank paths and lines starting with `#` are ignored. Each path may be absolute or relative to the location of the file. Import statements other than to `site` are not permitted, and arbitrary code cannot be specified.

Note that `._pth` files (without leading underscore) will be processed normally by the `site` module when `import site` has been specified.

When no `._pth` file is found, this is how `sys.path` is populated on Windows:

- An empty entry is added at the start, which corresponds to the current directory.
- If the environment variable `PYTHONPATH` exists, as described in [환경 변수](#), its entries are added next. Note that on Windows, paths in this variable must be separated by semicolons, to distinguish them from the colon used in drive identifiers (`C:\` etc.).
- Additional “application paths” can be added in the registry as subkeys of `\SOFTWARE\Python\PythonCore{version}\PythonPath` under both the `HKEY_CURRENT_USER` and `HKEY_LOCAL_MACHINE` hives. Subkeys which have semicolon-delimited path strings as their default value will cause each path to be added to `sys.path`. (Note that all known installers only use `HKLM`, so `HKCU` is typically empty.)
- If the environment variable `PYTHONHOME` is set, it is assumed as “Python Home”. Otherwise, the path of the main Python executable is used to locate a “landmark file” (either `Lib\os.py` or `pythonXY.zip`) to deduce the “Python Home”. If a Python home is found, the relevant sub-directories added to `sys.path` (`Lib`, `plat-win`, etc) are based on that folder. Otherwise, the core Python path is constructed from the `PythonPath` stored in the registry.
- If the Python Home cannot be located, no `PYTHONPATH` is specified in the environment, and no registry entries can be found, a default path with relative entries is used (e.g. `.\Lib`; `.\plat-win`, etc).

If a `pyvenv.cfg` file is found alongside the main executable or in the directory one level above the executable, the following variations apply:

- If `home` is an absolute path and `PYTHONHOME` is not set, this path is used instead of the path to the main executable when deducing the home location.

The end result of all this is:

- When running `python.exe`, or any other `.exe` in the main Python directory (either an installed version, or directly from the PCbuild directory), the core path is deduced, and the core paths in the registry are ignored. Other “application paths” in the registry are always read.
- When Python is hosted in another `.exe` (different directory, embedded via COM, etc), the “Python Home” will not be deduced, so the core path from the registry is used. Other “application paths” in the registry are always read.
- If Python can’t find its home and there are no registry value (frozen `.exe`, some very strange installation setup) you get a path with some default, but relative, paths.

For those who want to bundle Python into their application or distribution, the following advice will prevent conflicts with other installations:

- Include a `._pth` file alongside your executable containing the directories to include. This will ignore paths listed in the registry and environment variables, and also ignore `site` unless `import site` is listed.
- If you are loading `python3.dll` or `python37.dll` in your own executable, explicitly call `Py_SetPath()` or (at least) `Py_SetProgramName()` before `Py_Initialize()`.
- Clear and/or overwrite `PYTHONPATH` and set `PYTHONHOME` before launching `python.exe` from your application.
- If you cannot use the previous suggestions (for example, you are a distribution that allows people to run `python.exe` directly), ensure that the landmark file (`Lib\os.py`) exists in your install directory. (Note that it will not be detected inside a ZIP file, but a correctly named ZIP file will be detected instead.)

These will ensure that the files in a system-wide installation will not take precedence over the copy of the standard library bundled with your application. Otherwise, your users may experience problems using your application. Note that the first suggestion is the best, as the others may still be susceptible to non-standard paths in the registry and user site-packages.

버전 3.6에서 변경:

- Adds `._pth` file support and removes `applocal` option from `pyenv.config`.
- Adds `pythonXX.zip` as a potential landmark when directly adjacent to the executable.

버전 3.6부터 폐지: 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.

3.10 Additional modules

Even though Python aims to be portable among all platforms, there are features that are unique to Windows. A couple of modules, both in the standard library and external, and snippets exist to use these features.

The Windows-specific standard modules are documented in `mswin-specific-services`.

3.10.1 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 calls
- Registry
- Event log

- [Microsoft Foundation Classes \(MFC\)](#) user interfaces

[PythonWin](#) is a sample MFC application shipped with [PyWin32](#). It is an embeddable IDE with a built-in debugger.

더 보기:

[Win32 How Do I...?](#) by Tim Golden

[Python and COM](#) by David and Paul Boddie

3.10.2 `cx_Freeze`

`cx_Freeze` is a `distutils` extension (see [extending-distutils](#)) which 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.

3.10.3 `WConio`

Since Python's advanced terminal handling layer, `curses`, is restricted to Unix-like systems, there is a library exclusive to Windows as well: Windows Console I/O for Python.

`WConio` is a wrapper for Turbo-C's `CONIO.H`, used to create text user interfaces.

3.11 Compiling Python on Windows

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 2015, which is the compiler used to build the official Python releases. These files are in the `PCbuild` directory.

Check `PCbuild/readme.txt` for general information on the build process.

For extension modules, consult [building-on-windows](#).

더 보기:

[Python + Windows + distutils + SWIG + gcc MinGW](#) or “Creating Python extensions in C/C++ with SWIG and compiling them with MinGW gcc under Windows” or “Installing Python extension with distutils and without Microsoft Visual C++” by Sébastien Sauvage, 2003

[MingW – Python extensions](#)

3.12 Other Platforms

With ongoing development of Python, some platforms that used to be supported earlier are no longer supported (due to the lack of users or developers). Check [PEP 11](#) for details on all unsupported platforms.

- [Windows CE](#) is still supported.
- The [Cygwin](#) installer offers to install the Python interpreter as well (cf. [Cygwin package source](#), [Maintainer releases](#))

See [Python for Windows](#) for detailed information about platforms with pre-compiled installers.

맥킨토시에서 파이썬 사용하기

저자 Bob Savage <bobsavage@mac.com>

Mac OS X을 실행하는 맥킨토시 상의 파이썬은 원칙적으로 다른 모든 유닉스 플랫폼에 있는 파이썬과 매우 유사하지만, IDE와 패키지 관리자와 같은 언급할 만한 많은 추가 기능들이 있습니다.

4.1 MacPython을 구하고 설치하기

Mac OS X 10.8은 애플이 사전 설치한 파이썬 2.7을 제공합니다. 원한다면, 파이썬 웹 사이트 (<https://www.python.org>)에서 파이썬 3의 최신 버전을 설치하도록 초대합니다. 맥의 새로운 인텔과 레거시 PPC CPU에서 그대로 실행되는 최신 “유니버설 바이너리” 파이썬 빌드를 사용할 수 있습니다.

설치 후 얻을 수 있는 것은 여러 가지가 있습니다:

- A Python 3.7 folder in your Applications folder. In here you find IDLE, the development environment that is a standard part of official Python distributions; and PythonLauncher, which handles double-clicking Python scripts from the Finder.
- 파이썬 실행 파일과 라이브러리를 포함하는 프레임워크 `/Library/Frameworks/Python.framework`. 설치기는 이 위치를 셸 경로에 추가합니다. MacPython을 제거하려면, 이 세 가지를 지우면 됩니다. 파이썬 실행 파일에 대한 심볼릭 링크는 `/usr/local/bin/`에 있습니다.

애플에서 제공한 파이썬 빌드는 `/System/Library/Frameworks/Python.framework`와 `/usr/bin/python`에 각각 설치됩니다. 애플에서 제어하고 애플이나 제삼자 소프트웨어에서 사용하므로, 절대로 수정하거나 삭제해서는 안 됩니다. Python.org에서 최신 파이썬 버전을 설치했다면, 컴퓨터에 서로 다르지만 동작하는 두 개의 파이썬 설치를 갖게 된다는 것을 기억하십시오, 경로와 사용이 여러분이 원하는 것과 일치하는 것이 중요합니다.

IDLE에는 파이썬 설명서에 액세스할 수 있는 도움말 메뉴가 포함되어 있습니다. 파이썬을 완전히 처음 접한다면, 이 설명서에서 자습서 소개를 읽는 것으로 시작해야 합니다.

다른 유닉스 플랫폼에서 파이썬에 익숙하다면 유닉스 셸에서 파이썬 스크립트를 실행하는 부분을 읽어야 합니다.

4.1.1 파이썬 스크립트를 실행하는 방법

Mac OS X에서 파이썬을 시작하는 가장 좋은 방법은 IDLE 통합 개발 환경을 사용하는 것입니다, *IDE* 절을 참조하고, IDE가 실행 중일 때 도움말 메뉴를 사용하십시오.

터미널 창 명령 줄이나 Finder에서 파이썬 스크립트를 실행하려면 먼저 스크립트를 만들 편집기가 필요합니다. Mac OS X에는 여러 표준 유닉스 명령 줄 편집기, 그중에서도 **vim**과 **emacs**가 함께 제공됩니다. 좀 더 Mac 다운 편집기가 필요하다면, Bare Bones Software의 **BBEdit**나 **TextWrangler**(<http://www.barebones.com/products/bbedit/index.html>) 을 참조하십시오)와 **TextMate**(<https://macromates.com/>) 을 참조하십시오)는 좋은 선택입니다. 다른 편집기로는 **Gvim**(<http://macvim-dev.github.io/macvim/>)과 **Aquamacs**(<http://aquamacs.org/>)가 있습니다.

터미널 창에서 스크립트를 실행하려면 `/usr/local/bin`이 셸 검색 경로에 있도록 해야 합니다.

Finder에서 스크립트를 실행하려면 두 가지 옵션이 있습니다:

- 스크립트를 **PythonLauncher**로 드래그하십시오.
- Finder 정보 창을 통해 여러분의 스크립트(또는 모든 .py 스크립트)를 여는 기본 응용 프로그램으로 **PythonLauncher**를 선택하고 스크립트를 더블 클릭하십시오. **PythonLauncher**에는 스크립트를 시작하는 방법을 제어하는 다양한 설정이 있습니다. Option-드래그하면 하나의 호출에 대해 이를 변경할 수 있으며, 환경 설정 메뉴를 사용하여 전역적으로 변경할 수 있습니다.

4.1.2 GUI로 스크립트 실행하기

이전 버전의 파이썬에는, 여러분이 신경 써야 할 Mac OS X의 단점이 하나 있습니다: Aqua 창 관리자와 대화하는 프로그램(즉, GUI가 있는 프로그램)은 특별한 방법으로 실행해야 합니다. 이러한 스크립트를 시작하려면 **python** 대신 **pythonw**를 사용하십시오.

With Python 3.7, you can use either **python** or **pythonw**.

4.1.3 구성

OS X의 파이썬은 `PYTHONPATH`와 같은 표준 유닉스 환경 변수를 모두 준수하지만, Finder에서 시작된 프로그램에 대해 이러한 변수를 설정하는 것은 비표준인데, 시작 시 Finder가 여러분의 `.profile`이나 `.cshrc`를 읽지 않기 때문입니다. `~/MacOSX/environment.plist` 파일을 만들 필요가 있습니다. 자세한 내용은 애플의 기술 문서 QA1067을 참조하십시오.

MacPython의 파이썬 패키지 설치에 대한 자세한 내용은, 추가 파이썬 패키지 설치하기 절을 참조하십시오.

4.2 IDE

MacPython에는 표준 IDLE 개발 환경이 따라옵니다. IDLE 사용에 대한 좋은 소개는 http://www.hashcollision.org/hkn/python/idle_intro/index.html에서 찾을 수 있습니다.

4.3 추가 파이썬 패키지 설치하기

추가 파이썬 패키지를 설치하는 몇 가지 방법이 있습니다:

- 패키지는 표준 파이썬 `distutils` 모드를 통해 설치할 수 있습니다(`python setup.py install`).
- 많은 패키지는 **setuptools** 확장이나 **pip** 래퍼를 통해 설치할 수도 있습니다, <https://pip.pypa.io/> 를 참조하십시오.

4.4 Mac에서의 GUI 프로그래밍

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

*PyObjC*는 애플의 Objective-C/Cocoa 프레임워크에 대한 파이썬 바인딩입니다. 이 프레임워크는 가장 현대적인 Mac 개발의 기초입니다. PyObjC에 대한 정보는 <https://pypi.org/project/pyobjc/> 에서 얻을 수 있습니다.

표준 파이썬 GUI 킷은 크로스 플랫폼 Tk 킷(<https://www.tcl.tk>)을 기반으로 하는 tkinter입니다. Tk의 Aqua-네이티브 버전은 애플에 의해 OS X에 번들로 제공되며, 최신 버전은 <https://www.activestate.com> 에서 내려받아 설치할 수 있습니다; 소스에서 빌드할 수도 있습니다.

*wxPython*은 Mac OS X에서 네이티브하게 실행되는 다른 인기 있는 크로스 플랫폼 GUI 킷입니다. 패키지와 설명서는 <https://www.wxpython.org> 에서 제공됩니다.

*PyQt*는 Mac OS X에서 네이티브하게 실행되는 다른 인기 있는 크로스 플랫폼 GUI 킷입니다. 자세한 정보는 <https://riverbankcomputing.com/software/pyqt/intro> 에서 찾을 수 있습니다.

4.5 Mac에서 파이썬 응용 프로그램 배포하기

Mac에서 독립형 파이썬 응용 프로그램을 배포하기 위한 표준 도구는 **py2app**입니다. py2app의 설치와 사용에 대한 자세한 정보는 <http://undefined.org/python/#py2app> 에서 확인할 수 있습니다.

4.6 기타 자원

MacPython 메일링리스트는 Mac 파이썬 사용자와 개발자를 위한 훌륭한 지원 자원입니다:

<https://www.python.org/community/sigs/current/pythonmac-sig/>

또 다른 유용한 자원은 MacPython 위키입니다:

<https://wiki.python.org/moin/MacPython>

CHAPTER 5

Editors and IDEs

There are a number of IDEs that support Python programming language. Many editors and IDEs provide syntax highlighting, debugging tools, and **PEP 8** checks.

Please go to [Python Editors](#) and [Integrated Development Environments](#) for a comprehensive list.

>>> 대화형 셸의 기본 파이썬 프롬프트. 인터프리터에서 대화형으로 실행될 수 있는 코드 예에서 자주 볼 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.

2to3 파이썬 2.x 코드를 파이썬 3.x 코드로 변환하려고 시도하는 도구인데, 소스를 구문 분석하고 구문 분석 트리를 탐색해서 감지할 수 있는 대부분의 비호환성을 다룹니다.

2to3 는 표준 라이브러리에서 lib2to3 로 제공됩니다; 독립적으로 실행할 수 있는 스크립트는 Tools/scripts/2to3 로 제공됩니다. 2to3-reference 을 보세요.

abstract base class (추상 베이스 클래스) 추상 베이스 클래스는 `hasattr()` 같은 다른 테크닉들이 불편하거나 미묘하게 잘못된 (예를 들어, 매직 메서드) 경우, 인터페이스를 정의하는 방법을 제공함으로써 **덕 타이핑** 을 보완합니다. ABC는 가상 서브 클래스를 도입하는데, 클래스를 계승하지 않으면서도 `isinstance()` 와 `issubclass()` 에 의해 감지될 수 있는 클래스들입니다; abc 모듈 설명서를 보세요. 파이썬에는 많은 내장 ABC 들이 따라오는데 다음과 같은 것들이 있습니다: 자료 구조 (`collections.abc` 모듈에서), 숫자 (`numbers` 모듈에서), 스트림 (`io` 모듈에서), 임포트 파인더와 로더 (`importlib.abc` 모듈에서). abc 모듈을 사용해서 자신만의 ABC를 만들 수도 있습니다.

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

지역 변수의 어노테이션은 실행 시간에 액세스할 수 없지만, 전역 변수, 클래스 속성 및 함수의 어노테이션은 각각 모듈, 클래스, 함수의 `__annotations__` 특수 어트리뷰트에 저장됩니다.

이 기능을 설명하는 **변수 어노테이션**, **함수 어노테이션**, **PEP 484**, **PEP 526**을 참조하세요.

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 차이와 **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, 그 위치의 (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 *이터레이터*의 `__anext__()` 메서드가 돌려주는 어웨이터블을 팝니다. **PEP 492**로 도입되었습니다.

attribute (어트리뷰트) 점표현식을 사용하는 이름으로 참조되는 객체와 결합한 값. 예를 들어, 객체 *o*가 어트리뷰트 *a*를 가지면, *o.a*처럼 참조됩니다.

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.

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 캐시되어, 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.

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.

coercion (코어션) 같은 형의 두 인자를 수반하는 연산이 일어나는 동안, 한 형의 인스턴스를 다른 형으로 묵시적으로 변환하는 것. 예를 들어, `int(3.15)`는 실수를 정수 3으로 변환합니다. 하지만, `3+4.5`에서, 각 인자는 다른 형이고 (하나는 `int`, 다른 하나는 `float`), 둘을 더하기 전에 같은 형으로 변환해야 합니다. 그렇지 않으면 `TypeError`를 일으킵니다. 코어션 없이는, 호환되는 형들조차도 프로그래머가 같은 형으로 정규화해주어야 합니다, 예를 들어, 그냥 `3+4.5` 하는 대신 `float(3)+4.5`.

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 필요하다면, `cmath`를 사용합니다. 복소수의 활용은 꽤 수준 높은 수학적 기능입니다. 필요하다고 느끼지 못한다면, 거의 확실히 무시해도 좋습니다.

context manager (컨텍스트 관리자) `__enter__()`와 `__exit__()` 메서드를 정의함으로써 `with` 문에서 보이는 환경을 제어하는 객체. **PEP 343**으로 도입되었습니다.

context variable (컨텍스트 변수) 컨텍스트에 따라 다른 값을 가질 수 있는 변수. 이는 각 실행 스레드가 변수에 대해 다른 값을 가질 수 있는 스레드-로컬 저장소와 비슷합니다. 그러나, 컨텍스트 변수를 통해, 하나의 실행 스레드에 여러 컨텍스트가 있을 수 있으며 컨텍스트 변수의 주 용도는 동시성 비동기 태스크에서 변수를 추적하는 것입니다. `contextvars`를 참조하십시오.

contiguous (연속) 버퍼는 정확히 C-연속(*C-contiguous*)이거나 포트란 연속(*Fortran contiguous*)일 때 연속이라고 여겨집니다. 영차원 버퍼는 C-연속이면서 포트란 연속입니다. 일차원 배열에서, 항목들은 서로에 인접하고, 0에서 시작하는 오름차순 인덱스의 순서대로 메모리에 배치되어야 합니다. 다차원 C-연속 배열에서, 메모리 주소의 순서대로 항목들을 방문할 때 마지막 인덱스가 가장 빨리 변합니다. 하지만, 포트란 연속 배열에서는, 첫 번째 인덱스가 가장 빨리 변합니다.

coroutine (코루틴) Coroutines are a more generalized form of subroutines. Subroutines are entered at one point and exited at another point. Coroutines can be entered, exited, and resumed at many different points. They can be implemented with the `async def` statement. See also **PEP 492**.

coroutine function (코루틴 함수) 코루틴 객체를 돌려주는 함수. 코루틴 함수는 `async def` 문으로 정의될 수 있고, `await`와 `async for`와 `async with` 키워드를 포함할 수 있습니다. 이것들은 **PEP 492**에 의해 도입되었습니다.

CPython 파이썬 프로그래밍 언어의 규범적인 구현인데, python.org에서 배포됩니다. 이 구현을 Jython 이나 IronPython 과 같은 다른 것들과 구별할 필요가 있을 때 용어 “CPython”이 사용됩니다.

decorator (데코레이터) 다른 함수를 돌려주는 함수인데, 보통 `@wrapper` 문법을 사용한 함수 변환으로 적용됩니다. 데코레이터의 흔한 예는 `classmethod()`과 `staticmethod()`입니다.

데코레이터 문법은 단지 편의 문법일 뿐입니다. 다음 두 함수 정의는 의미상으로 동등합니다:

```
def f(...):
    ...
f = staticmethod(f)

@staticmethod
def f(...):
    ...
```

같은 개념이 클래스에도 존재하지만, 덜 자주 쓰입니다. 데코레이터에 대한 더 자세한 내용은 함수 정의와 클래스 정의의 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 디렉터리에서 *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, 슈퍼클래스 참조 등의 많은 기능의 기초를 이루고 있기 때문입니다.

디스크립터의 메서드들에 대한 자세한 내용은 `descriptors`에 나옵니다.

dictionary (딕셔너리) 임의의 키를 값에 대응시키는 연관 배열 (associative array). 키는 `__hash__()` 와 `__eq__()` 메서드를 갖는 모든 객체가 될 수 있습니다. 필에서 해시라고 부릅니다.

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-views`를 보세요.

docstring (독스트링) 클래스, 함수, 모듈에서 첫 번째 표현식으로 나타나는 문자열 리터럴. 스위트가 실행될 때는 무시되지만, 컴파일러에 의해 인지되어 돌려쏜 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 꺾꺾댄다면, 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 클래스로 보완될 수 있음에 유의해야 합니다.) 대신에, `hasattr()` 검사나 *EAFP* 프로그래밍을 씁니다.

EAFP 허락보다는 용서를 구하기가 쉽다 (Easier to ask for forgiveness than permission). 이 흔히 볼 수 있는 파이썬 코딩 스타일은, 올바른 키나 어트리뷰트의 존재를 가정하고, 그 가정이 틀리면 예외를 잡습니다. 이 깔끔하고 빠른 스타일은 많은 `try`와 `except` 문의 존재로 특징지어집니다. 이 테크닉은 C와 같은 다른 많은 언어에서 자주 사용되는 *LBYL* 스타일과 대비됩니다.

expression (표현식) 어떤 값으로 구해질 수 있는 문법적인 조각. 다른 말로 표현하면, 표현식은 리터럴, 이름, 어트리뷰트 액세스, 연산자, 함수들과 같은 값을 돌려주는 표현 요소들을 쌓아 올린 것입니다. 다른 많은 언어와 대조적으로, 모든 언어 구성물들이 표현식인 것은 아닙니다. `while`처럼, 표현식으로 사용할 수 없는 문장들이 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 종류의 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 객체의 비슷한 말.

finder (파인더) 임포트될 모듈을 위한 **로더**를 찾으려고 시도하는 객체.

파이썬 3.3. 이후로, 두 종류의 파인더가 있습니다: `sys.meta_path`와 함께 사용하는 **메타 경로 파인더**와 `sys.path_hooks`와 함께 사용하는 **경로 엔트리 파인더**.

더 자세한 내용은 **PEP 302**, **PEP 420**, **PEP 451**에 나옵니다.

floor division (정수 나눗셈) 가장 가까운 정수로 내림하는 수학적 나눗셈. 정수 나눗셈 연산자는 `//`다. 예를 들어, 표현식 `11 // 4`의 값은 2가 되지만, 실수 나눗셈은 2.75를 돌려줍니다. `(-11) // 4`가 -2.75를 내림한 -3이 됨에 유의해야 합니다. **PEP 238**을 보세요.

function (함수) 호출자에게 어떤 값을 돌려주는 일련의 문장들. 없거나 그 이상의 **인자**가 전달될 수 있는데, 바디의 실행에 사용될 수 있습니다. **매개변수**와 **메서드**와 **function** 섹션도 보세요.

function annotation (함수 어노테이션) 함수 매개변수나 반환 값의 **어노테이션**.

함수 어노테이션은 일반적으로 **형 힌트**로 사용됩니다: 예를 들어, 이 함수는 두 개의 `int` 인자를 받아 들일 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.

__future__ 프로그래머가 현재 인터프리터와 호환되지 않는 새 언어 기능들을 활성화할 수 있도록 하는 가상 모듈.

`__future__` 모듈을 임포트하고 그 변수들의 값들을 구해서, 새 기능이 언제 처음으로 언어에 추가되었고, 언제부터 그것이 기본이 되는지 볼 수 있습니다:

```
>>> 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 이터레이터를 가리킵니다. 의도하는 의미가 명확하지 않은 경우는, 완전한 용어를 써서 모호함을 없앱니다.

generator iterator (제너레이터 이터레이터) **제너레이터** 함수가 만드는 객체.

각 `yield`는 일시적으로 처리를 중단하고, 그 위치의 (지역 변수들과 대기 중인 `try`-문들을 포함하는) 실행 상태를 기억합니다. 제너레이터 이터레이터가 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 표현식처럼 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.

GIL 전역 인터프리터 록 을 보세요.

global interpreter lock (전역 인터프리터 록) 한 번에 오직 하나의 스레드가 파이썬 바이트 코드를 실행하도록 보장하기 위해 CPython 인터프리터가 사용하는 메커니즘. (`dict`와 같은 중요한 내장형들을 포함하는) 객체 모델이 묵시적으로 동시 액세스에 대해 안전하도록 만들어서 CPython 구현을 단순하게 만듭니다. 인터프리터 전체를 잠그는 것은 인터프리터를 다중스레드화하기 쉽게 만드는 대신, 다중 프로세서 기계가 제공하는 병렬성의 많은 부분을 희생합니다.

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.

(훨씬 더 미세하게 공유 데이터를 잠그는) “스레드에 자유로운(*free-threaded*)” 인터프리터를 만들고자 하는 과거의 노력은 성공적이지 못했는데, 혼란 단일 프로세서 경우의 성능 저하가 심하기 때문입니다. 이 성능 이슈를 극복하는 것은 구현을 훨씬 복잡하게 만들어서 유지 비용이 더 들어갈 것으로 여겨지고 있습니다.

hash-based pyc (해시 기반 pyc) 유효성을 판별하기 위해 해당 소스 파일의 최종 수정 시간이 아닌 해시를 사용하는 바이트 코드 캐시 파일. `pyc-invalidation`을 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 환경 (Integrated Development Environment). IDLE은 파이썬의 표준 배포판에 따라오는 기초적인 편집기와 인터프리터 환경입니다.

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 (임포트) 한 모듈의 파이썬 코드가 다른 모듈의 파이썬 코드에서 사용될 수 있도록 하는 절차.

importer (임포터) 모듈을 찾기도 하고 로드 하기도 하는 객체; 동시에 **파인더**이자 **로더** 객체입니다.

interactive (대화형) 파이썬은 대화형 인터프리터를 갖고 있는데, 인터프리터 프롬프트에서 문장과 표현식을 입력할 수 있고, 즉각 실행된 결과를 볼 수 있다는 뜻입니다. 인자 없이 단지 `python`을 실행하세요 (컴퓨터의 주메뉴에서 선택하는 것도 가능할 수 있습니다). 새 아이디어를 검사하거나 모듈과 패키지를 들여다보는 매우 강력한 방법입니다 (`help(x)`를 기억하세요).

interpreted (인터프리티드) 바이트 코드 컴파일러의 존재 때문에 그 부분이 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 종료를 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 여러 분을 대신해서 자동으로 해주는데, 루프를 도는 동안 이터레이터를 잡아들 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.

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` 모듈은 세 개의 키 함수 생성자를 제공합니다: `attrgetter()`, `itemgetter()`, `methodcaller()`. 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.

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) 모듈을 로드하는 객체. `load_module()` 이라는 이름의 메서드를 정의해야 합니다. 로더는 보통 *파인더* 가 돌려줍니다. 자세한 내용은 [PEP 302](#) 를, 추상 베이스 클래스는 `importlib.abc.Loader` 를 보세요.

magic method (매직 메서드) 특수 메서드 의 비공식적인 비슷한 말.

mapping (매핑) 임의의 키 조회를 지원하고 Mapping 이나 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 디렉터리, 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 사용되었습니다.

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 [The Python 2.3 Method Resolution Order](#) 를 보면 됩니다.

module (모듈) 파이썬 코드의 조직화 단위를 담당하는 객체. 모듈은 임의의 파이썬 객체들을 담는 이름 공간을 갖습니다. 모듈은 *임포트* 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.

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) The term “named tuple” applies to any type or class that inherits from tuple and whose indexable elements are also accessible using named attributes. The type or class may have other features as well.

Several built-in types are named tuples, including the values returned by `time.localtime()` and `os.stat()`. Another example is `sys.float_info`:

```

>>> 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

```

Some named tuples are built-in types (such as the above examples). Alternatively, a named tuple can be created from a regular class definition that inherits from `tuple` and that defines named fields. Such a class can be written by hand or it can be created with the factory function `collections.namedtuple()`. The latter technique also adds some extra methods that may not be found in hand-written or built-in named tuples.

namespace (이름 공간) 변수가 저장되는 장소. 이름 공간은 딕셔너리로 구현됩니다. 객체에 중첩된 이름 공간(메서드에서) 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 유지보수성에 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 **PEP 420** 패키지. 이름 공간 패키지는 물리적인 실체가 없을 수도 있고, 특히 `__init__.py` 파일이 없으므로 정규 패키지와는 다릅니다.

모듈도 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, `__getattr__()`, 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.

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 인자로 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. 파이썬은 위치-전용 매개변수를 정의하는 문법을 갖고 있지 않습니다. 하지만, 어떤 매개 함수들은 위치-전용 매개변수를 갖습니다(예를 들어, `abs()`).
- 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 파인더가 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](#) [혹](#)) 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 [혹](#)) `sys.path_hook` 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` 객체가거나 `os.PathLike` 프로토콜을 구현하는 객체입니다. `os.PathLike` 프로토콜을 지원하는 객체는 `os.fspath()` 함수를 호출해서 `str` 나 `bytes` 파일 시스템 경로로 변환될 수 있습니다; 대신 `os.fsdecode()` 와 `os.fsencode()` 는 각각 `str` 나 `bytes` 결과를 보장하는데 사용될 수 있습니다. [PEP 519](#)로 도입되었습니다.

PEP 파이썬 개선 제안. PEP는 파이썬 커뮤니티에 정보를 제공하거나 파이썬 또는 그 프로세스 또는 환경에 대한 새로운 기능을 설명하는 설계 문서입니다. PEP는 제안된 기능에 대한 간결한 기술 사양 및 근거를 제공해야 합니다.

PEP는 주요 새로운 기능을 제안하고 문제에 대한 커뮤니티 입력을 수집하며 파이썬에 들어간 설계 결정을 문서로 만들기 위한 기본 메커니즘입니다. PEP 작성자는 커뮤니티 내에서 합의를 구축하고 반대 의견을 문서화 할 책임이 있습니다.

[PEP 1](#) 참조하세요.

portion (포션) [PEP 420](#)에서 정의한 것처럼, 이름 공간 패키지에 이바지하는 하나의 디렉터리에 들어있는 파일들의 집합 (zip 파일에 저장되는 것도 가능합니다).

positional argument (위치 인자) [인자](#)를 보세요.

provisional API (잠정 API) 잠정 API는 표준 라이브러리의 과거 호환성 보장으로부터 신중히 제외된 것입니다. 인터페이스의 큰 변화가 예상되지는 않지만, 잠정적이라고 표시되는 한, 코어 개발자들이 필요하다고 생각한다면 과거 호환성이 유지되지 않는 변경이 일어날 수 있습니다. 그런 변경은 불필요한 방식으로 일어나지는 않을 것입니다 — API를 포함하기 전에 놓친 중대하고 근본적인 결함이 발견된 경우에만 일어날 것입니다.

잠정 API에서조차도, 과거 호환성이 유지되지 않는 변경은 “최후의 수단”으로 여겨집니다 - 모든 식별된 문제들에 대해 과거 호환성을 유지하는 해법을 찾으려는 모든 시도가 선행됩니다.

이 절차는 표준 라이브러리가 오랜 시간 동안 잘못된 설계 오류에 발목 잡히지 않고 발전할 수 있도록 만듭니다. 더 자세한 내용은 [PEP 411](#)을 보면 됩니다.

provisional package (잠정 패키지) 잠정 *API* 를 보세요.

Python 3000 (파이썬 3000) 파이썬 3.x 배포 라인의 별명 (버전 3의 배포가 먼 미래의 이야기던 시절에 만들어진 이름이다.) 이것을 “Py3k” 로 줄여 쓰기도 합니다.

Pythonic (파이썬다운) 다른 언어들에서 일반적인 개념들을 사용해서 코드를 구현하는 대신, 파이썬 언어에서 가장 자주 사용되는 이디엄들을 가까이 따르는 아이디어나 코드 조각. 예를 들어, 파이썬에서 자주 쓰는 이디엄은 `for` 문을 사용해서 이터러블의 모든 요소로 루핑하는 것입니다. 다른 많은 언어에는 이런 종류의 구성물이 없으므로, 파이썬에 익숙하지 않은 사람들은 대신에 숫자 카운터를 사용하기도 합니다:

```
for i in range(len(food)):
    print(food[i])
```

더 깔끔한, 파이썬다운 방법은 이렇습니다:

```
for piece in food:
    print(piece)
```

qualified name (정규화된 이름) 모듈의 전역 스코프에서 모듈에 정의된 클래스, 함수, 메서드에 이르는 “경로” 를 보여주는 점으로 구분된 이름. [PEP 3155](#) 에서 정의됩니다. 최상위 함수와 클래스의 경우에, 정규화된 이름은 객체의 이름과 같습니다:

```
>>> class C:
...     class D:
...         def meth(self):
...             pass
...
>>> C.__qualname__
'C'
>>> C.D.__qualname__
'C.D'
>>> C.D.meth.__qualname__
'C.D.meth'
```

모듈을 가리키는데 사용될 때, 완전히 정규화된 이름 (*fully qualified name*)은 모든 부모 패키지들을 포함 해서 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. 참조 횟수 추적은 일반적으로 파이썬 코드에 노출되지는 않지만, *CPython* 구현의 핵심 요소입니다. `sys` 모듈은 특정 객체의 참조 횟수를 돌려주는 `getrefcount()` 을 정의합니다.

regular package (정규 패키지) `__init__.py` 파일을 포함하는 디렉터리와 같은 전통적인 패키지.

이름 공간 패키지 도 보세요.

__slots__ 클래스 내부의 선언인데, 인스턴스 어트리뷰트들을 위한 공간을 미리 선언하고 인스턴스 디렉터리를 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 불변 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.

single dispatch (싱글 디스패치) 구현이 하나의 인자의 형에 기초해서 결정되는 제네릭 함수 디스패치의 한 형태.

slice (슬라이스) 보통 시퀀스의 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.

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`.

text encoding (텍스트 인코딩) 유니코드 문자열을 바이트열로 인코딩하는 코덱.

text file (텍스트 파일) `str` 객체를 읽고 쓸 수 있는 파일 객체. 종종, 텍스트 파일은 실제로는 바이트 지향 데이터 스트림을 액세스하고 텍스트 인코딩을 자동 처리합니다. 텍스트 파일의 예로는 텍스트 모드 ('r' 또는 'w') 로 열린 파일, `sys.stdin`, `sys.stdout`, `io.StringIO` 의 인스턴스를 들 수 있습니다.

바이트열 객체를 읽고 쓸 수 있는 파일 객체에 대해서는 바이너리 파일도 참조하세요.

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 걸쳐 줄 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:

```
from typing import List, Tuple

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:

```
from typing import List, Tuple

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 사항이며 파이썬에서 강제되지는 않습니다. 하지만, 정적 형 분석 도구에 유용하며 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 힌트](#)로 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.

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 코드](#)를 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

이 설명서는 `reStructuredText` 소스에서 만들어진 것으로, 파이썬 설명서를 위해 특별히 제작된 문서 처리기인 `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 작가;
- `reStructuredText`와 `Docutils` 스위트를 만드는 `Docutils` 프로젝트.
- Fredrik Lundh, 그의 `Alternative Python Reference` 프로젝트에서 `Sphinx`가 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!

역사와 라이선스

C.1 소프트웨어의 역사

파이썬은 ABC라는 언어의 후계자로서 네덜란드의 Stichting Mathematisch Centrum (CWI, <https://www.cwi.nl/> 참조)의 Guido van Rossum에 의해 1990년대 초반에 만들어졌습니다. 파이썬에는 다른 사람들의 많은 공헌이 포함되었지만, Guido는 파이썬의 주요 저자로 남아 있습니다.

1995년, Guido는 Virginia의 Reston에 있는 Corporation for National Research Initiatives(CNRI, <https://www.cnri.reston.va.us/> 참조)에서 파이썬 작업을 계속했고, 이곳에서 여러 버전의 소프트웨어를 출시했습니다.

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 (now Zope Corporation; see <https://www.zope.org/>). 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 is a sponsoring member of the PSF.

모든 파이썬 배포판은 공개 소스입니다 (공개 소스 정의에 대해서는 <https://opensource.org/>를 참조하십시오). 역사적으로, 대부분 (하지만 전부는 아닙니다) 파이썬 배포판은 GPL과 호환됩니다; 아래의 표는 다양한 배포판을 요약한 것입니다.

배포판	파생된 곳	해	소유자	GPL 호환?
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	no
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: GPL과 호환된다는 것은 우리가 GPL로 파이썬을 배포한다는 것을 의미하지는 않습니다. 모든 파이썬 라이선스는 GPL과 달리 여러분의 변경을 공개 소스로 만들지 않고 수정된 버전을 배포할 수 있게 합니다. GPL 호환 라이선스는 파이썬과 GPL 하에 발표된 다른 소프트웨어를 결합할 수 있게 합니다; 다른 것들은 그렇지 않습니다.

Guido의 지도하에 이 배포를 가능하게 만든 많은 외부 자원봉사자들에게 감사드립니다.

C.2 파이썬에 액세스하거나 사용하기 위한 이용 약관

C.2.1 PSF LICENSE AGREEMENT FOR PYTHON 3.7.17

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"),
→and
the Individual or Organization ("Licensee") accessing and otherwise using
→Python
3.7.17 software 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 3.7.17 alone or in any derivative
version, provided, however, that PSF's License Agreement and PSF's notice
→of
copyright, i.e., "Copyright © 2001-2023 Python Software Foundation; All
→Rights
Reserved" are retained in Python 3.7.17 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 3.7.17 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
3.7.17.
4. PSF is making Python 3.7.17 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 3.7.17 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.7.17
FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT
→OF
MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.7.17, 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 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 3.7.17, 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

(다음 페이지에 계속)

(이전 페이지에서 계속)

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 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

(다음 페이지에 계속)

(이전 페이지에서 계속)

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.3 포함된 소프트웨어에 대한 라이선스 및 승인

이 섹션은 파이썬 배포판에 포함된 제삼자 소프트웨어에 대한 불완전하지만 늘어나고 있는 라이선스와 승인의 목록입니다.

C.3.1 메르센 트위스터

_random 모듈은 <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html> 에서 내려 받은 코드에 기반한 코드를 포함합니다. 다음은 원래 코드의 주석을 그대로 옮긴 것입니다:

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).

(다음 페이지에 계속)

(이전 페이지에서 계속)

Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura,
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. 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

socket 모듈은 `getaddrinfo()` 와 `getnameinfo()` 함수를 사용합니다. 이들은 WIDE Project, <http://www.wide.ad.jp/>, 에서 온 별도 소스 파일로 코딩되어 있습니다.

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.

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
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
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

asynchat과 asyncore 모듈은 다음과 같은 주의 사항을 포함합니다:

```
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
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

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 ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

C.3.5 실행 추적

trace 모듈은 다음과 같은 주의 사항을 포함합니다:

portions copyright 2001, Autonomous Zones Industries, Inc., all rights...
err... reserved and offered to the public under the terms of the
Python 2.2 license.
Author: Zooko O'Whielacronx
<http://zooko.com/>
<mailto:zooko@zooko.com>

Copyright 2000, Mojam Media, Inc., all rights reserved.
Author: Skip Montanaro

Copyright 1999, Bioreason, Inc., all rights reserved.
Author: Andrew Dalke

Copyright 1995-1997, Automatrix, Inc., all rights reserved.
Author: Skip Montanaro

Copyright 1991-1995, Stichting Mathematisch Centrum, all rights reserved.

Permission to use, copy, modify, and distribute this Python software and its associated documentation for any purpose without fee is hereby granted, provided that the above copyright notice appears in all copies, and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of neither Automatrix, Bioreason or Mojam Media be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

C.3.6 UUencode 및 UUdecode 함수

uu 모듈은 다음과 같은 주의 사항을 포함합니다:

Copyright 1994 by Lance Ellinghouse
Cathedral City, California Republic, United States of America.
All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in

(다음 페이지에 계속)

(이전 페이지에서 계속)

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.

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 MERCHANTABILITY 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

test_epoll 모듈은 다음과 같은 주의 사항을 포함합니다:

```
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
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/

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

C double과 문자열 간의 변환을 위한 C 함수 dtoa 와 strtod 를 제공하는 파일 Python/dtoa.c 는 현재 <http://www.netlib.org/fp/> 에서 얻을 수 있는 David M. Gay의 같은 이름의 파일에서 파생되었습니다. 2009년 3월 16일에 받은 원본 파일에는 다음과 같은 저작권 및 라이선스 공지가 포함되어 있습니다:

```
/*****
 *
 * 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

모듈 `hashlib`, `posix`, `ssl`, `crypt` 는 운영 체제가 사용할 수 있게 하면 추가의 성능을 위해 OpenSSL 라이브러리를 사용합니다. 또한, 윈도우와 맥 OS X 파이썬 설치 프로그램은 OpenSSL 라이브러리 사본을 포함할 수 있으므로, 여기에 OpenSSL 라이선스 사본을 포함합니다:

LICENSE ISSUES

=====

The OpenSSL toolkit stays under a dual license, i.e. both the conditions of the OpenSSL License and the original SSLeay license apply to the toolkit. See below for the actual license texts. Actually both licenses are BSD-style Open Source licenses. In case of any license issues related to OpenSSL please contact openssl-core@openssl.org.

OpenSSL License

```
/* =====
 * Copyright (c) 1998-2008 The OpenSSL 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. All advertising materials mentioning features or use of this
 * software must display the following acknowledgment:
 * "This product includes software developed by the OpenSSL Project
 * for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 * endorse or promote products derived from this software without
 * prior written permission. For written permission, please contact
 * openssl-core@openssl.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 * nor may "OpenSSL" appear in their names without prior written
 * permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 * acknowledgment:
 * "This product includes software developed by the OpenSSL Project
 * for use in the OpenSSL Toolkit (http://www.openssl.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED 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 OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

* 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.
* =====
*
* This product includes cryptographic software written by Eric Young
* (eay@cryptsoft.com). This product includes software written by Tim
* Hudson (tjh@cryptsoft.com).
*
*/

```

Original SSLeay License

```

/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
* All rights reserved.
*
* This package is an SSL implementation written
* by Eric Young (eay@cryptsoft.com).
* The implementation was written so as to conform with Netscapes SSL.
*
* This library is free for commercial and non-commercial use as long as
* the following conditions are aheared to. The following conditions
* apply to all code found in this distribution, be it the RC4, RSA,
* lhash, DES, etc., code; not just the SSL code. The SSL documentation
* included with this distribution is covered by the same copyright terms
* except that the holder is Tim Hudson (tjh@cryptsoft.com).
*
* Copyright remains Eric Young's, and as such any Copyright notices in
* the code are not to be removed.
* If this package is used in a product, Eric Young should be given attribution
* as the author of the parts of the library used.
* This can be in the form of a textual message at program startup or
* in documentation (online or textual) provided with the package.
*
* 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 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. All advertising materials mentioning features or use of this software
* must display the following acknowledgement:
* "This product includes cryptographic software written by
* Eric Young (eay@cryptsoft.com)"
* The word 'cryptographic' can be left out if the rouines from the library
* being used are not cryptographic related :-).
* 4. If you include any Windows specific code (or a derivative thereof) from
* the apps directory (application code) you must include an acknowledgement:
* "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
*

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
* THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``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.
*
* The licence and distribution terms for any publically available version or
* derivative of this code cannot be changed. i.e. this code cannot simply be
* copied and put under another distribution licence
* [including the GNU Public Licence.]
*/
```

C.3.13 expat

pyexpat 확장은 빌드를 `--with-system-expat` 로 구성하지 않는 한, 포함된 expat 소스 사본을 사용하여 빌드됩니다:

```
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
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

`_ctypes` 확장은 빌드를 `--with-system-libffi` 로 구성하지 않는 한, 포함된 `libffi` 소스 사본을 사용하여 빌드됩니다:

```
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.

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

_decimal 모듈은 빌드를 --with-system-libmpdec 로 구성하지 않는 한, 포함된 libmpdec 소스 사본을 사용하여 빌드됩니다:

```
Copyright (c) 2008-2016 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.
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

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.

APPENDIX D

저작권

파이썬과 이 설명서는:

Copyright © 2001-2023 Python Software Foundation. All rights reserved.

Copyright © 2000 BeOpen.com. All rights reserved.

Copyright © 1995-2000 Corporation for National Research Initiatives. All rights reserved.

Copyright © 1991-1995 Stichting Mathematisch Centrum. All rights reserved.

전체 라이선스 및 사용 권한 정보는 [역사](#)와 [라이선스](#) 에서 제공합니다.

Non-alphabetical

..., 43

-?

command line option, 5

2to3, 43

>>>, 43

__future__, 47

__slots__, 53

A

abstract base class (추상 베이스 클래스), 43

annotation (어노테이션), 43

argument (인자), 43

asynchronous context manager (비동기 컨텍스트 관리자), 44

asynchronous generator (비동기 제너레이터), 44

asynchronous generator iterator (비동기 제너레이터 이터레이터), 44

asynchronous iterable (비동기 이터러블), 44

asynchronous iterator (비동기 이터레이터), 44

attribute (어트리뷰트), 44

awaitable (어웨이터블), 44

B

-B

command line option, 5

-b

command line option, 5

BDFL, 44

binary file (바이너리 파일), 44

bytecode (바이트 코드), 45

bytes-like object (바이트열류 객체), 44

C

-c <command>

command line option, 4

C-contiguous, 45

--check-hash-based-pycs

default | always | never

command line option, 5

class (클래스), 45

class variable (클래스 변수), 45

coercion (코어션), 45

command line option

-, 5

-B, 5

-b, 5

-c <command>, 4

--check-hash-based-pycs

default | always | never, 5

-d, 6

-E, 6

-h, 5

--help, 5

-I, 6

-i, 6

-J, 9

-m <module-name>, 4

-O, 6

-OO, 6

-q, 6

-R, 6

-S, 7

-s, 7

-u, 7

-V, 5

-v, 7

--version, 5

-W arg, 7

-X, 8

-x, 7

complex number (복소수), 45

context manager (컨텍스트 관리자), 45

context variable (컨텍스트 변수), 45

contiguous (연속), 45

coroutine (코루틴), 45

coroutine function (코루틴 함수), 45

CPython, [45](#)

D

-d

command line option, [6](#)

decorator (데코레이터), [45](#)

descriptor (디스크립터), [46](#)

dictionary (딕셔너리), [46](#)

dictionary view (딕셔너리 뷰), [46](#)

docstring (독스트링), [46](#)

duck-typing (덕 타이핑), [46](#)

E

-E

command line option, [6](#)

EAFP, [46](#)

exec_prefix, [16](#)

expression (표현식), [46](#)

extension module (확장 모듈), [46](#)

F

f-string (*f*-문자열), [46](#)

file object (파일 객체), [46](#)

file-like object (파일류 객체), [47](#)

finder (파인더), [47](#)

floor division (정수 나눗셈), [47](#)

Fortran contiguous, [45](#)

function (함수), [47](#)

function annotation (함수 어노테이션), [47](#)

G

garbage collection (가비지 수거), [47](#)

generator, [47](#)

generator (제너레이터), [47](#)

generator expression, [47](#)

generator expression (제너레이터 표현식), [47](#)
generator iterator (제너레이터 이터레이터), [47](#)

generic function (제네릭 함수), [48](#)

GIL, [48](#)

global interpreter lock (전역 인터프리터
록), [48](#)

H

-h

command line option, [5](#)

hash-based pyc (해시 기반 *pyc*), [48](#)

hashable (해시 가능), [48](#)

--help

command line option, [5](#)

I

-I

command line option, [6](#)

-i

command line option, [6](#)

IDLE, [48](#)

immutable (불변), [48](#)

import path (임포트 경로), [48](#)

importer (임포터), [48](#)

importing (임포팅), [48](#)

interactive (대화형), [48](#)

interpreted (인터프리티드), [48](#)

interpreter shutdown (인터프리터 종료), [49](#)

iterable (이터러블), [49](#)

iterator (이터레이터), [49](#)

J

-J

command line option, [9](#)

K

key function (키 함수), [49](#)

keyword argument (키워드 인자), [49](#)

L

lambda (람다), [49](#)

LBYL, [49](#)

list (리스트), [50](#)

list comprehension (리스트 컴프리헨션), [50](#)

loader (로더), [50](#)

M

-m <module-name>

command line option, [4](#)

magic

method, [50](#)

magic method (매직 메서드), [50](#)

mapping (매핑), [50](#)

meta path finder (메타 경로 파인더), [50](#)

metaclass (메타 클래스), [50](#)

method

magic, [50](#)

special, [54](#)

method (메서드), [50](#)

method resolution order (메서드 결정 순서),
[50](#)

module (모듈), [50](#)

module spec (모듈 스펙), [50](#)

MRO, [50](#)

mutable (가변), [50](#)

N

named tuple (네임드 튜플), [50](#)

namespace (이름 공간), [51](#)

namespace package (이름 공간 패키지), [51](#)

nested scope (중첩된 스코프), [51](#)
 new-style class (뉴스타일 클래스), [51](#)

O

-O
 command line option, [6](#)
 object (객체), [51](#)
 -OO
 command line option, [6](#)

P

package (패키지), [51](#)
 parameter (매개변수), [51](#)
 PATH, [9](#), [17](#), [2022](#), [2729](#), [31](#)
 path based finder (경로 기반 파인더), [52](#)
 path entry (경로 엔트리), [52](#)
 path entry finder (경로 엔트리 파인더), [52](#)
 path entry hook (경로 엔트리 훅), [52](#)
 path-like object (경로류 객체), [52](#)
 PATHEXT, [22](#)
 PEP, [52](#)
 portion (포션), [52](#)
 positional argument (위치 인자), [52](#)
 prefix, [16](#)
 provisional API (잠정 API), [52](#)
 provisional package (잠정 패키지), [53](#)
 PY_PYTHON, [31](#)
 Python 3000 (파이썬 3000), [53](#)
 PYTHON*, [46](#)
 PYTHONCOERCECLOCALE, [13](#)
 PYTHONDEBUG, [6](#)
 PYTHONDONTWRITEBYTECODE, [5](#)
 PYTHONHASHSEED, [6](#), [9](#), [33](#), [34](#)
 PYTHONHOME, [6](#), [9](#), [33](#), [34](#)
 Pythonic (파이썬다운), [53](#)
 PYTHONINSPECT, [6](#)
 PYTHONINTMAXSTRDIGITS, [8](#)
 PYTHONIOENCODING, [13](#)
 PYTHONLEGACYWINDOWSSSTDIO, [10](#)
 PYTHONMALLOC, [12](#)
 PYTHONOPTIMIZE, [6](#)
 PYTHONPATH, [6](#), [9](#), [27](#), [33](#), [34](#), [38](#)
 PYTHONPROFILEIMPORTTIME, [8](#)
 PYTHONSTARTUP, [6](#)
 PYTHONUNBUFFERED, [7](#)
 PYTHONUTF8, [8](#), [13](#), [28](#)
 PYTHONVERBOSE, [7](#)
 PYTHONWARNINGS, [7](#)

Q

-q
 command line option, [6](#)
 qualified name (정규화된 이름), [53](#)

R

-R
 command line option, [6](#)
 reference count (참조 횟수), [53](#)
 regular package (정규 패키지), [53](#)

S

-S
 command line option, [7](#)
 -s
 command line option, [7](#)
 sequence (시퀀스), [53](#)
 single dispatch (싱글 디스패치), [54](#)
 slice (슬라이스), [54](#)
 special
 method, [54](#)
 special method (특수 메서드), [54](#)
 statement (문장), [54](#)

T

text encoding (텍스트 인코딩), [54](#)
 text file (텍스트 파일), [54](#)
 triple-quoted string (삼중 따옴표 된 문자열),
 [54](#)
 type (형), [54](#)
 type alias (형 에일리어스), [54](#)
 type hint (형 힌트), [54](#)

U

-u
 command line option, [7](#)
 universal newlines (유니버설 줄 넘김), [55](#)

V

-V
 command line option, [5](#)
 -v
 command line option, [7](#)
 variable annotation (변수 어노테이션), [55](#)
 --version
 command line option, [5](#)
 virtual environment (가상 환경), [55](#)
 virtual machine (가상 기계), [55](#)

W

-W arg
 command line option, [7](#)

X

-X
 command line option, [8](#)
 -x
 command line option, [7](#)

Y

파이썬 향상 제안

[PEP 1](#), [52](#)
[PEP 8](#), [41](#)
[PEP 11](#), [19](#), [35](#)
[PEP 238](#), [47](#)
[PEP 278](#), [55](#)
[PEP 302](#), [47](#), [50](#)
[PEP 338](#), [4](#)
[PEP 343](#), [45](#)
[PEP 362](#), [44](#), [52](#)
[PEP 370](#), [7](#), [10](#)
[PEP 397](#), [29](#)
[PEP 411](#), [53](#)
[PEP 420](#), [47](#), [51](#), [52](#)
[PEP 443](#), [48](#)
[PEP 451](#), [47](#)
[PEP 484](#), [43](#), [47](#), [54](#), [55](#)
[PEP 488](#), [6](#)
[PEP 492](#), [44](#), [45](#)
[PEP 498](#), [46](#)
[PEP 519](#), [52](#)
[PEP 525](#), [44](#)
[PEP 526](#), [43](#), [55](#)
[PEP 528](#), [28](#)
[PEP 529](#), [12](#), [28](#)
[PEP 538](#), [13](#)
[PEP 540](#), [13](#)
[PEP 3116](#), [55](#)
[PEP 3155](#), [53](#)

환경 변수

[exec_prefix](#), [16](#)
[PATH](#), [9](#), [17](#), [2022](#), [2729](#), [31](#)
[PATHEXT](#), [22](#)
[prefix](#), [16](#)
[PY_PYTHON](#), [31](#)
[PYTHON*](#), [46](#)
[PYTHONASYNCIODEBUG](#), [11](#)
[PYTHONBREAKPOINT](#), [9](#)
[PYTHONCASEOK](#), [10](#)
[PYTHONCOERCECLOCALE](#), [12](#), [13](#)
[PYTHONDEBUG](#), [6](#), [9](#)
[PYTHONDEVMODE](#), [13](#)
[PYTHONDONTWRITEBYTECODE](#), [5](#), [10](#)
[PYTHONDUMPPREFS](#), [14](#)
[PYTHONEXECUTABLE](#), [10](#)
[PYTHONFAULTHANDLER](#), [11](#)
[PYTHONHASHSEED](#), [6](#), [10](#)
[PYTHONHOME](#), [6](#), [9](#), [33](#), [34](#)
[PYTHONINSPECT](#), [6](#), [9](#)
[PYTHONINTMAXSTRDIGITS](#), [8](#), [10](#)
[PYTHONIOENCODING](#), [10](#), [13](#)
[PYTHONLEGACYWINDOWSFSENCODING](#), [12](#)
[PYTHONLEGACYWINDOWSSTDIO](#), [10](#), [12](#)

[PYTHONMALLOC](#), [11](#), [12](#)
[PYTHONMALLOCSTATS](#), [12](#)
[PYTHONNOUSERSITE](#), [10](#)
[PYTHONOPTIMIZE](#), [6](#), [9](#)
[PYTHONPATH](#), [6](#), [9](#), [27](#), [33](#), [34](#), [38](#)
[PYTHONPROFILEIMPORTTIME](#), [8](#), [11](#)
[PYTHONSTARTUP](#), [6](#), [9](#)
[PYTHONTHREADDEBUG](#), [14](#)
[PYTHONTRACEMALLOC](#), [11](#)
[PYTHONUNBUFFERED](#), [7](#), [10](#)
[PYTHONUSERBASE](#), [10](#)
[PYTHONUTF8](#), [8](#), [13](#), [28](#)
[PYTHONVERBOSE](#), [7](#), [10](#)
[PYTHONWARNINGS](#), [7](#), [11](#)

Z

[Zen of Python \(파이썬 젠\)](#), [55](#)