# Python Setup and Usage

發🄵 *3.8.8rc1*

**Guido van Rossum**
**and the Python development team**

**2 月 19, 2021**

**Python Software Foundation**
**Email: docs@python.org**

# Contents

这一部分文档专门介绍关于在不同平台上设置 Python 环境、调用解释器以及让使用 Python 更容易的一些事情的有用信息。

CHAPTER 1

# 命令行与环境

CPython 解析器会扫描命令行与环境用于获取各种设置信息。

**CPython implementation detail:** 其他实现的命令行方案可能有所不同。更多相关资源请参阅 implementations。

## 1.1 命令行

对 Python 发起调用时，你可以指定以下的任意选项:

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

当然最常见的用例就是简单地启动执行一个脚本:

```
python myscript.py
```

### 1.1.1 接口选项

解释器接口类似于 UNIX shell，但提供了一些额外的发起调用方法:

- 当调用时附带连接到某个 tty 设备的标准输入时，它会提示输入命令并执行它们，直到读入一个 EOF（文件结束字符，其产生方式是在 UNIX 中按 `Ctrl-D` 或在 Windows 中按 `Ctrl-Z, Enter`。）
- 当调用时附带一个文件名参数或以一个文件作为标准输入时，它会从该文件读取并执行脚本程序。
- 当调用时附带一个目录名参数时，它会从该目录读取并执行具有适当名称的脚本程序。
- 当调用时附带 `-c command` 时，它会执行 *command* 所给出的 Python 语句。在这里 *command* 可以包含以换行符分隔的多条语句。请注意前导空格在 Python 语句中是有重要作用的！
- 当调用时附带 `-m module-name` 时，会在 Python 模块路径中查找指定的模块，并将其作为脚本程序执行。

在非交互模式下，会对全部输入先解析再执行。

一个接口选项会终结解释器所读入的选项列表，后续的所有参数将被放入 sys.argv -- 请注意其中首个元素即第零项 (sys.argv[0]) 会是一个表示程序源的字符串。

**-c** <command>
执行 *command* 中的 Python 代码。*command* 可以为一条或以换行符分隔的多条语句，其中前导空格像在普通模块代码中一样具有作用。

如果给出此选项，sys.argv 的首个元素将为 "-c" 并且当前目录将被加入 sys.path 的开头（以允许该目录中的模块作为最高层级模块被导入）。

使用 command 参数会引发 审计事件 cpython.run_command 。

**-m** <module-name>
在 sys.path 中搜索指定名称的模块并将其内容作为 __main__ 模块来执行。

由于该参数为 *module* 名称，你不应给出文件扩展名 (.py)。模块名称应为绝对有效的 Python 模块名称，但具体实现可能并不总是强制要求这一点（例如它可能允许你使用包含连字符的名称）。

包名称（包括命名空间包）也允许使用。当所提供的是包名称而非普通模块名称时，解释器将把 <pkg>.__main__ 作为主模块来执行。此行为特意被设计为与作为脚本参数传递给解释器的目录和 zip 文件的处理方式类似。

---

**備🄴:** 此选项不适用于内置模块和以 C 编写的扩展模块，因为它们并没有对应的 Python 模块文件。但是它仍然适用于预编译的模块，即使没有可用的初始源文件。

---

如果给出此选项，sys.argv 的首个元素将为模块文件的完整路径 (在定位模块文件期间，首个元素将设为 "-m")。与 *-c* 选项一样，当前目录将被加入 sys.path 的开头。

*-I* 选项可用来在隔离模式下运行脚本，此模式中 sys.path 既不包含当前目录也不包含用户的 site-packages 目录。所有 PYTHON* 环境变量也会被忽略。

许多标准库模块都包含作为脚本执行时会被发起调用的代码。其中的一个例子是 timeit 模块:

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

使用 module-name 参数会引发 审计事件 cpython.run_module 。

**也参考:**

**runpy.run_module()** Python 代码可以直接使用的等效功能

**PEP 338** -- 将模块作为脚本执行

3.1 版更變: 提供包名称来运行 __main__ 子模块。

3.4 版更變: 同样支持命名空间包

**-**
从标准输入 (sys.stdin) 读取命令。如果标准输入为一个终端，则使用 *-i*。

如果给出此选项，sys.argv 的首个元素将为 "-" 并且当前目录将被加入 sys.path 的开头。

没有参数会引发 审计事件 cpython.run_stdin 。

**<script>**
执行 *script* 中的 Python 代码，该参数应为一个（绝对或相对）文件系统路径，指向某个 Python 文件、包含 __main__.py 文件的目录，或包含 __main__.py 文件的 zip 文件。

如果给出此选项，sys.argv 的首个元素将为在命令行中指定的脚本名称。

如果脚本名称直接指向一个 Python 文件，则包含该文件的目录将被加入 sys.path 的开头，并且该文件会被作为 __main__ 模块来执行。

如果脚本名称指向一个目录或 zip 文件，则脚本名称将被加入 sys.path 的开头，并且该位置中的 __main__.py 文件会被作为 __main__ 模块来执行。

*-I* 选项可用来在隔离模式下运行脚本，此模式中 sys.path 既不包含脚本所在目录也不包含用户的 site-packages 目录。所有 PYTHON* 环境变量也会被忽略。

使用 filename 参数会引发 审计事件 cpython.run_file 。

**也参考:**

**runpy.run_path()** Python 代码可以直接使用的等效功能

如果没有给出接口选项，则使用*-i*，sys.argv[0] 将为空字符串 (""), 并且当前目录会被加入 sys.path 的开头。此外，tab 补全和历史编辑会自动启用，如果你的系统平台支持此功能的话 (参见 rlcompleter-config)。

**也参考:**

tut-invoking

3.4 版更變: 自动启用 tab 补全和历史编辑。

### 1.1.2 通用选项

**-?**
**-h**
**--help**
打印全部命令行选项的简短描述。

**-V**
**--version**
打印 Python 版本号并退出。示例输出信息如下:

```
Python 3.8.0b2+
```

如果重复给出，则打印有关构建的更多信息，例如:

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

3.6 版新加入: -VV 选项。

### 1.1.3 其他选项

**-b**
在将 bytes 或 bytearray 与 str 或是将 bytes 与 int 比较时发出警告。如果重复给出该选项 (-bb) 则会引发错误。

3.5 版更變: 影响 bytes 与 int 的比较。

**-B**
如果给出此选项，Python 将不会试图在导入源模块时写入 .pyc 文件。另请参阅 *PYTHONDONTWRITEBYTECODE*。

**--check-hash-based-pycs** `default|always|never`
控制基于哈希值的 `.pyc` 文件的验证行为。参见 pyc-invalidation。当设为 `default` 时，已选定和未选定的基于哈希值的字节码缓存文件将根据其默认语义进行验证。当设为 `always` 时，所有基于哈希值的 `.pyc` 文件，不论是已选定还是未选定的都将根据其对应的源文件进行验证。当设为 `never` 时，基于哈希值的 `.pyc` 文件将不会根据其对应的源文件进行验证。

基于时间戳的 `.pyc` 文件的语义不会受此选项影响。

**-d**
开启解析器调试输出（限专家使用，依赖于编译选项）。另请参阅 *PYTHONDEBUG*。

**-E**
忽略所有 `PYTHON*` 环境变量，例如可能已设置的 *PYTHONPATH* 和 *PYTHONHOME*。

**-i**
当有脚本被作为首个参数传入或使用了 *-c* 选项时，在执行脚本或命令之后进入交互模式，即使是在 `sys.stdin` 并不是一个终端的时候。*PYTHONSTARTUP* 文件不会被读取。

这一选项的用处是在脚本引发异常时检查全局变量或者栈跟踪。另请参阅 *PYTHONINSPECT*。

**-I**
在隔离模式下运行 Python。这将同时应用 -E 和 -s。在隔离模式下 `sys.path` 既不包含脚本所在目录也不包含用户的 site-packages 目录。所有 `PYTHON*` 环境变量也会被忽略。还可以施加更进一步的限制以防止用户注入恶意代码。

3.4 版新加入.

**-O**
移除 assert 语句以及任何以 `__debug__` 的值作为条件的代码。通过在 `.pyc` 扩展名之前添加 `.opt-1` 来扩充已编译文件 (*bytecode*) 的文件名 (参见 **PEP 488**)。另请参阅 *PYTHONOPTIMIZE*。

3.5 版更變: 依据 **PEP 488** 修改 `.pyc` 文件名。

**-OO**
在启用 *-O* 的同时丢弃文档字符串。通过在 `.pyc` 扩展名之前添加 `.opt-2` 来扩展已编译文件 (*bytecode*) 的文件名 (参见 **PEP 488**)。

3.5 版更變: 依据 **PEP 488** 修改 `.pyc` 文件名。

**-q**
即使在交互模式下也不显示版权和版本信息。

3.2 版新加入.

**-R**
开启哈希随机化。此选项权 *PYTHONHASHSEED* 环境变量设置为 `0` 时起作用，因为哈希随机化是默认启用的。

在 Python 的早期版本中，此选项启用哈希随机化，将 str 和 bytes 的对象 `__hash__()` 的值" 加盐" 为不可预测的随机值。虽然它们在单个 Python 进程中保持不变，但是在重复调用的 Python 进程之间它们是不可预测的。

哈希随机化旨在针对由精心选择的输入引起的拒绝服务攻击提供防护，这种输入利用了构造 dict 在最坏情况下的性能即 O(n^2) 复杂度。详情请参阅 http://www.ocert.org/advisories/ocert-2011-003.html。

*PYTHONHASHSEED* 允许你为哈希种子密码设置一个固定值。

3.7 版更變: 此选项不会再被忽略。

3.2.3 版新加入.

**-s**
不要将 用户 `site-packages` 目录添加到 `sys.path`。

**也参考:**

**PEP 370** -- 分用户的 site-packages 目录

**-S**

禁用 site 的导入及其所附带的基于站点对 sys.path 的操作。如果 site 会在稍后被显式地导入也会禁用这些操作 (如果你希望触发它们则应调用 site.main())。

**-u**

强制 stdout 和 stderr 流不使用缓冲。此选项对 stdin 流无影响。

另请参阅 *PYTHONUNBUFFERED*。

3.7 版更變: stdout 和 stderr 流在文本层现在不使用缓冲。

**-v**

每当一个模块被初始化时打印一条信息，显示其加载位置（文件名或内置模块）。当重复给出时 (-vv)，为搜索模块时所检查的每个文件都打印一条消息。此外还提供退出时有关模块清理的信息 A。另请参阅 *PYTHONVERBOSE*。

**-W** arg

警告控制。Python 的警告机制在默认情况下会向 sys.stderr 打印警告消息。典型的警告消息具有如下形式:

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

默认情况下，每个警告都对于其发生所在的每个源行都会打印一次。此选项可控制警告打印的频繁程度。

可以给出多个 *-W* 选项；当某个警告能与多个选项匹配时，将执行最后一个匹配选项的操作。无效的 *-W* 选项将被忽略（但是，在发出第一个警告时会打印有关无效选项的警告消息）。

警告也可以使用 *PYTHONWARNINGS* 环境变量以及使用 warnings 模块在 Python 程序内部进行控制。

最简单的设置是将某个特定操作无条件地应用于进程所发出所有警告 (即使是在默认情况下会忽略的那些警告):

```
-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
```

操作名称可以根据需要进行缩写 (例如 -Wi, -Wd, -Wa, -We)，解释器将会把它们解析为适当的操作名称。

请参阅 warning-filter 和 describing-warning-filters 了解更多细节。

**-x**

跳过源中第一行，以允许使用非 Unix 形式的 #!cmd。这适用于 DOS 专属的破解操作。

**-X**

保留用于各种具体实现专属的选项。CPython 目前定义了下列可用的值:

- -X faulthandler 启用 faulthandler;
- -X showrefcount 当程序结束或在交互解释器中的每条语句之后输出总引用计数和已使用内存块计数。此选项仅在调试版本中有效。
- -X tracemalloc 使用 tracemalloc 模块启动对 Python 内存分配的跟踪。默认情况下，只有最近的帧会保存在跟踪的回溯信息中。使用 -X tracemalloc=NFRAME 以启动限定回溯 *NFRAME* 帧的跟踪。请参阅 tracemalloc.start() 了解详情。

- `-X showalloccount` 当程序结束时输出每种类型的已分配对象的总数。此选项仅当 Python 在定义了 `COUNT_ALLOCS` 后构建时才会生效。
- `-X importtime` 显示每次导入耗费的时间。它会显示模块名称，累计时间（包括嵌套的导入）和自身时间（排除嵌套的导入）。请注意它的输出在多线程应用程序中可能会出错。典型用法如 `python3 -X importtime -c 'import asyncio'`。另请参阅 *PYTHONPROFILEIMPORTTIME*。
- `-X dev`: 启用 CPython 的 “开发模式”，引入额外的运行时检测，这些检测因开销过大而无法默认启用。如果代码是正确的则它不会比默认输出更详细：新增警告只会在发现问题时才会发出。开发模式的作用效果：
  - 添加 `default` 警告过滤器，即 *-W* `default`。
  - 在内存分配器上安装调试钩子：参见 `PyMem_SetupDebugHooks()` C 函数。
  - 启用 `faulthandler` 模块以在发生崩溃时转储 Python 回溯信息。
  - 启用 asyncio 调试模式。
  - 将 `sys.flags` 的 `dev_mode` 属性设为 `True`。
  - `io.IOBase` 析构函数记录 `close()` 异常。
- `-X utf8` 为操作系统接口启用 UTF-8 模式，覆盖默认的区域感知模式。`-X utf8=0` 显式地禁用 UTF-8 模式（即使在它应当被自动激活的时候）。请参阅 *PYTHONUTF8* 了解详情。
- `-X pycache_prefix=PATH` 允许将 `.pyc` 文件写入以给定目录为根的并行树，而不是代码树。另见 *PYTHONPYCACHEPREFIX* 。

它还允许传入任意值并通过 `sys._xoptions` 字典来提取这些值。

3.2 版更變: 增加了 *-X* 选项。

3.3 版新加入: `-X faulthandler` 选项。

3.4 版新加入: `-X showrefcount` 与 `-X tracemalloc` 选项。

3.6 版新加入: `-X showalloccount` 选项。

3.7 版新加入: `-X importtime`, `-X dev` 与 `-X utf8` 选项。

3.8 版新加入: `-X pycache_prefix` 选项。`-X dev` 选项现在在 `io.IOBase` 析构函数中记录 `close()` 异常。

### 1.1.4 不应当使用的选项

**-J**

保留给 Jython 使用。

## 1.2 环境变量

这些环境变量会影响 Python 的行为，它们是在命令行开关之前被处理的，但 -E 或 -I 除外。根据约定，当存在冲突时命令行开关会覆盖环境变量的设置。

**PYTHONHOME**

更改标准 Python 库的位置。默认情况下库是在 *prefix*`/lib/python`*version* 和 *exec_prefix*`/lib/python`*version* 中搜索，其中 *prefix* 和 *exec_prefix* 是由安装位置确定的目录，默认都位于 `/usr/local`。

当 *PYTHONHOME* 被设为单个目录时，它的值会同时替代 *prefix* 和 *exec_prefix*。要为两者指定不同的值，请将 *PYTHONHOME* 设为 *prefix*`:`*exec_prefix*。

**PYTHONPATH**
增加模块文件默认搜索路径。所用格式与终端的 PATH 相同：一个或多个由 os.pathsep 分隔的目录路径名称（例如 Unix 上用冒号而在 Windows 上用分号）。默认忽略不存在的目录。

除了普通目录之外，单个 *PYTHONPATH* 条目可以引用包含纯 Python 模块的 zip 文件（源代码或编译形式）。无法从 zip 文件导入扩展模块。

默认索引路径依赖于安装路径，但通常都是以 *prefix*/lib/pythonversion 开始 (参见上文中的 *PYTHONHOME*)。它 总是会被添加到 *PYTHONPATH*。

有一个附加目录将被插入到索引路径的 *PYTHONPATH* 之前，正如上文中接口选项 所描述的。搜索路径可以在 Python 程序内作为变量 sys.path 来进行操作。

**PYTHONSTARTUP**
这如果是一个可读文件的名称，该文件中的 Python 命令会在交互模式的首个提示符显示之前被执行。该文件会在与交互式命令执行所在的同一命名空间中被执行，因此其中所定义或导入的对象可以在交互式会话中无限制地使用。你还可以在这个文件中修改提示符 sys.ps1 和 sys.ps2 以及钩子 sys.__interactivehook__。

使用 filename 参数会引发 审计事件 cpython.run_startup 。

**PYTHONOPTIMIZE**
这如果被设为一个非空字符串，它就相当于指定 *-O* 选项。如果设为一个整数，则它就相当于多次指定 *-O*。

**PYTHONBREAKPOINT**
此变量如果被设定，它会使用加点号的路径标记一个可调用对象。包含该可调用对象的模块将被导入，随后该可调用对象将由 sys.breakpointhook() 的默认实现来运行，后者自身将由内置的 breakpoint() 来调用。如果未设定，或设定为空字符串，则它相当于值”pdb.set_trace”。将此变量设为字符串”0” 会导致 sys.breakpointhook() 的默认实现不做任何事而直接返回。

3.7 版新加入.

**PYTHONDEBUG**
此变量如果被设为一个非空字符串，它就相当于指定 *-d* 选项。如果设为一个整数，则它就相当于多次指定 *-d*。

**PYTHONINSPECT**
此变量如果被设为一个非空字符串，它就相当于指定 *-i* 选项。

此变量也可由 Python 代码使用 os.environ 来修改以在程序终结时强制检查模式。

**PYTHONUNBUFFERED**
此变量如果被设为一个非空字符串，它就相当于指定 *-u* 选项。

**PYTHONVERBOSE**
此变量如果被设为一个非空字符串，它就相当于指定 *-v* 选项。如果设为一个整数，则它就相当于多次指定 *-v*。

**PYTHONCASEOK**
如果设置此变量，Python 将忽略 import 语句中的大小写。这仅在 Windows 和 OS X 上有效。

**PYTHONDONTWRITEBYTECODE**
此变量如果被设为一个非空字符串，Python 将不会尝试在导入源模块时写入 .pyc 文件。这相当于指定 *-B* 选项。

**PYTHONPYCACHEPREFIX**
如果设置了此选项，Python 将在镜像目录树中的此路径中写入 .pyc 文件，而不是源树中的 __pycache__ 目录中。这相当于指定 *-X* pycache_prefix=PATH 选项。

3.8 版新加入.

**PYTHONHASHSEED**

如果此变量未设置或设为 `random`，将使用一个随机值作为 str 和 bytes 对象哈希运算的种子。

如果 *PYTHONHASHSEED* 被设为一个整数值，它将被作为固定的种子数用来生成哈希随机化所涵盖的类型的 hash() 结果。

它的目的是允许可复现的哈希运算，例如用于解释器本身的自我检测，或允许一组 python 进程共享哈希值。

该整数必须为一个 [0,4294967295] 范围内的十进制数。指定数值 0 将禁用哈希随机化。

3.2.3 版新加入.

**PYTHONIOENCODING**

如果此变量在运行解释器之前被设置，它会覆盖通过 `encodingname:errorhandler` 语法设置的 stdin/stdout/stderr 所用编码。`encodingname` 和 `:errorhandler` 部分都是可选项，与在 `str.encode()` 中的含义相同。

对于 stderr，`:errorhandler` 部分会被忽略；处理程序将总是为 `'backslashreplace'`。

3.4 版更變: “encodingname” 部分现在是可选的。

3.6 版更變: 在 Windows 上，对于交互式控制台缓冲区会忽略此变量所指定的编码，除非还指定了 *PYTHONLEGACYWINDOWSSTDIO*。通过标准流重定向的文件和管道则不受其影响。

**PYTHONNOUSERSITE**

如果设置了此变量，Python 将不会把 `用户 site-packages 目录`添加到 `sys.path`。

**也参考:**

**PEP 370** -- 分用户的 site-packages 目录

**PYTHONUSERBASE**

`定义 用户基准目录，它会在执行 python setup.py install --user 时被用来计算 用户 site-packages 目录`的路径以及 Distutils 安装路径。

**也参考:**

**PEP 370** -- 分用户的 site-packages 目录

**PYTHONEXECUTABLE**

如果设置了此环境变量，则 `sys.argv[0]` 将被设为此变量的值而不是通过 C 运行时所获得的值。仅在 Mac OS X 上起作用。

**PYTHONWARNINGS**

此变量等价于 *-W* 选项。如果被设为一个以逗号分隔的字符串，它就相当于多次指定 *-W*，列表中后出现的过滤器优先级会高于列表中先出现的。

最简单的设置是将某个特定操作无条件地应用于进程所发出所有警告 (即使是在默认情况下会忽略的那些警告):

```
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
```

请参阅 warning-filter 和 describing-warning-filters 了解更多细节。

**PYTHONFAULTHANDLER**

如果此环境变量被设为一个非空字符串，`faulthandler.enable()` 会在启动时被调用：为 `SIGSEGV`, `SIGFPE`, `SIGABRT`, `SIGBUS` 和 `SIGILL` 等信号安装一个处理句柄以转储 Python 回溯信息。此变量等价于 *-X* `faulthandler` 选项。

3.3 版新加入.

**PYTHONTRACEMALLOC**
如果此环境变量被设为一个非空字符串，则会使用 `tracemalloc` 模块启动对 Python 内存分配的跟踪。该变量的值是保存于跟踪的回溯信息中的最大帧数。例如，`PYTHONTRACEMALLOC=1` 只保存最近的帧。请参阅 `tracemalloc.start()` 了解详情。

3.4 版新加入.

**PYTHONPROFILEIMPORTTIME**
如果此变量被设为一个非空字符串，Python 将显示每次导入花费了多长时间。此变量完全等价于在命令行为设置 `-X importtime`。

3.7 版新加入.

**PYTHONASYNCIODEBUG**
如果此变量被设为一个非空字符串，则会启用 `asyncio` 模块的 调试模式。

3.4 版新加入.

**PYTHONMALLOC**
设置 Python 内存分配器和/或安装调试钩子。

设置 Python 所使用的内存分配器族群：

- `default`: 使用 默认内存分配器。
- `malloc`: 对所有域 (`PYMEM_DOMAIN_RAW`, `PYMEM_DOMAIN_MEM`, `PYMEM_DOMAIN_OBJ`) 使用 C 库的 `malloc()` 函数。
- `pymalloc`: 对 `PYMEM_DOMAIN_MEM` 和 `PYMEM_DOMAIN_OBJ` 域使用 pymalloc 分配器而对 `PYMEM_DOMAIN_RAW` 域使用 `malloc()` 函数。

安装调试钩子：

- `debug`: 在 默认内存分配器之上安装调试钩子。
- `malloc_debug`: 与 `malloc` 相同但还会安装调试钩子。
- `pymalloc_debug`: 与 `pymalloc` 相同但还会安装调试钩子。

请参阅 默认内存分配器以及 `PyMem_SetupDebugHooks()` 函数（在 Python 内存分配器之上安装调试钩子）。

3.7 版更變: 增加了 `"default"` 分配器。

3.6 版新加入.

**PYTHONMALLOCSTATS**
如果设为一个非空字符串，Python 将在每次创建新的 pymalloc 对象区域以及在关闭时打印 pymalloc 内存分配器的统计信息。

如果 *`PYTHONMALLOC`* 环境变量被用来强制开启 C 库的 `malloc()` 分配器，或者如果 Python 的配置不支持 `pymalloc`，则此变量将被忽略。

3.6 版更變: 此变量现在也可以被用于在发布模式下编译的 Python。如果它被设置为一个空字符串则将没有任何效果。

**PYTHONLEGACYWINDOWSFSENCODING**
如果设为一个非空字符串，则默认的文件系统编码和错误模式将分别恢复为它们在 3.6 版之前的值'mbcs' 和'replace'。否则会使用新的默认值'utf-8' 和'surrogatepass'。

这也可以在运行时通过 `sys._enablelegacywindowsfsencoding()` 来启用。

可用性: Windows。

3.6 版新加入: 有关更多详细信息，请参阅 **PEP 529**。

**PYTHONLEGACYWINDOWSSTDIO**

如果设为一个非空字符串，则不使用新的控制台读取器和写入器。这意味着 Unicode 字符将根据活动控制台的代码页进行编码，而不是使用 utf-8。

如果标准流被重定向（到文件或管道）而不是指向控制台缓冲区则该变量会被忽略。

可用性: Windows。

3.6 版新加入.

**PYTHONCOERCECLOCALE**

如果值设为 `0`，将导致主 Python 命令行应用跳过将传统的基于 ASCII 的 C 与 POSIX 区域设置强制转换为更强大的基于 UTF-8 的替代方案。

如果此变量 *未被设置*（或被设为 `0` 以外的值），则覆盖环境变量的 `LC_ALL` 区域选项也不会被设置，并且报告给 `LC_CTYPE` 类别的当前区域选项或者为默认的 `C` 区域，或者为显式指明的基于 ASCII 的 `POSIX` 区域，然后 Python CLI 将在加载解释器运行时之前尝试为 `LC_CTYPE` 类别按指定的顺序配置下列区域选项:

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

如果成功设置了以上区域类别中的一个，则初始化 Python 运行时之前也将在当前进程环境中相应地设置 `LC_CTYPE` 环境变量。这会确保除了解释器本身和运行于同一进程中的其他可感知区域选项的组件 (例如 GNU `readline` 库) 之外，还能在子进程 (无论这些进程是否在运行 Python 解释器) 以及在查询环境而非当前 C 区域的操作 (例如 Python 自己的 `locale.getdefaultlocale()`) 中看到更新的设置。

(显式地或通过上述的隐式区域强制转换) 配置其中一个区域选项将自动为 `sys.stdin` 和 `sys.stdout` 启用 `surrogateescape` 错误处理句柄 (`sys.stderr` 会继续使用 `backslashreplace` 如同在任何其他区域选项中一样)。这种流处理行为可以按通常方式使用 *PYTHONIOENCODING* 来覆盖。

出于调试目的，如果激活了区域强制转换，或者如果当 Python 运行时被初始化时某个 *应该*触发强制转换的区域选项仍处于激活状态则设置 `PYTHONCOERCECLOCALE=warn` 将导致 Python 在 `stderr` 上发出警告消息。

还要注意，即使在区域转换转换被禁用，或者在其无法找到合适的目标区域时，默认 *PYTHONUTF8* 仍将在传统的基于 ASCII 的区域中被激活。必须同时禁用这两项特性以强制解释器使用 `ASCII` 而不是 `UTF-8` 作为系统接口。

可用性: *nix。

3.7 版新加入: 请参阅 **PEP 538** 了解详情。

**PYTHONDEVMODE**

如果此环境变量被设为一个非空字符串，则会启用 CPython “开发模式”。参见 *-X* `dev` 选项。

3.7 版新加入.

**PYTHONUTF8**

如果设为 `1`，则会启用解释器的 UTF-8 模式，将 `UTF-8` 用作系统接口的文本编码，无论当前区域选项如何设置。

这意味着:

- `sys.getfilesystemencoding()` 将返回 `'UTF-8'` (本地编码会被忽略)。
- `locale.getpreferredencoding()` 将返回 `'UTF-8'` (本地编码会被忽略，并且该函数的 `do_setlocale` 参数不起作用)。

- sys.stdin, sys.stdout 和 sys.stderr 都将 UTF-8 用作它们的文本编码，并且为 sys.stdin 和 sys.stdout 启用 surrogateescape 错误处理句柄 (sys.stderr 会继续使用 backslashreplace 如同在默认的局部感知模式下一样)

作为低层级 API 发生改变的结果，其他高层级 API 也会表现出不同的默认行为:

- 命令行参数，环境变量和文件名会使用 UTF-8 编码来解码为文本。
- os.fsdecode() 和 os.fsencode() 会使用 UTF-8 编码。
- open(), io.open() 和 codecs.open() 默认会使用 UTF-8 编码。但是，它们默认仍将使用严格错误处理句柄，因此试图在文本模式下打开二进制文件将可能引发异常，而不是生成无意义的数据。

请注意 UTF-8 模式下的标准流设置可以被 *PYTHONIOENCODING* 所覆盖（在默认的区域感知模式下也同样如此）。

如果设置为 “0”，则解释器以其默认的区域识别模式运行。

设置任何其他非空字符串会在解释器初始化期间导致错误。

如果根本未设置此环境变量，则解释器默认使用当前区域设置，除非当前区域被标识为基于 ASCII 的旧式区域设置（如 *PYTHONCOERCECLOCALE* 所述），并且区域强制转换被禁用或失败。在此类旧式区域设置中，解释器将默认启用 UTF-8 模式，除非显式地指定不这样做。

也可以使用 *-X* utf8 选项。

3.7 版新加入: 有关更多详细信息，请参阅 **PEP 540** 。

### 1.2.1 调试模式变量

设置这些变量只会在 Python 的调试版本中产生影响。

**PYTHONTHREADDEBUG**
如果设置，Python 将打印线程调试信息。

需要使用 --with-pydebug 构建选项配置 Python。

**PYTHONDUMPREFS**
如果设置，Python 在关闭解释器，及转储对象和引用计数后仍将保持活动。

需要使用 --with-trace-refs 构建选项配置 Python。

CHAPTER 2

# 在类 Unix 环境下使用 Python

## 2.1 获得并安装 Python 的最新版本

### 2.1.1 在 Linux 中

Python 预装在大多数 Linux 发行版上，并作为一个包提供给所有其他用户。但是，您可能想要使用的某些功能在发行版提供的软件包中不可用。这时您可以从源代码轻松编译最新版本的 Python。

如果 Python 没有预先安装并且不在发行版提供的库中，您可以轻松地为自己使用的发行版创建包。参阅以下链接：

**也参考：**

**https://www.debian.org/doc/manuals/maint-guide/first.en.html** 对于 Debian 用户

**https://en.opensuse.org/Portal:Packaging** 对于 OpenSuse 用户

**https://docs-old.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/ch-creating-rpms.html** 对于 Fedora 用户

**http://www.slackbook.org/html/package-management-making-packages.html** 对于 Slackware 用户

### 2.1.2 在 FreeBSD 和 OpenBSD 上

- FreeBSD 用户，使用以下命令添加包:

```
pkg install python3
```

- OpenBSD 用户，使用以下命令添加包:

```
pkg_add -r python

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

例如：i386 用户获取 Python 2.5.1 的可用版本:

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

### 2.1.3 在 OpenSolaris 系统上

你可以从 OpenCSW 获取、安装及使用各种版本的 Python。比如 `pkgutil -i python27` 。

## 2.2 构建 Python

如果你想自己编译 CPython，首先要做的是获取 source 。您可以下载最新版本的源代码，也可以直接提取最新的 clone 。(如果你想要制作补丁，则需要克隆代码。)

构建过程由常用命令组成:

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

特定 Unix 平台的配置选项和注意事项通常记录在 Python 源代码的根目录下的 README.rst 文件中。

**警告:** `make install` 可以覆盖或伪装 `python3` 二进制文件。因此，建议使用 `make altinstall` 而不是 `make install` ，因为后者只安装了 *`exec_prefix`*`/bin/python`*`version`* 。

## 2.3 与 Python 相关的路径和文件

这取决于本地安装惯例；`prefix` (`${prefix}`) 和 `exec_prefix` (`${exec_prefix}`) 取决于安装，应解释为 GNU 软件；它们可能相同。

例如，在大多数 Linux 系统上，两者的默认值是 `/usr` 。

| 文件/目录 | 含义 |
|---|---|
| *`exec_prefix`*`/bin/python3` | 解释器的推荐位置 |
| *`prefix`*`/lib/python`*`version`*, *`exec_prefix`*`/lib/python`*`version`* | 包含标准模块的目录的推荐位置 |
| *`prefix`*`/include/python`*`version`*, *`exec_prefix`*`/include/python`*`version`* | 包含开发 Python 扩展和嵌入解释器所需的 include 文件的目录的推荐位置 |

## 2.4 杂项

要在 Unix 上使用 Python 脚本，需要添加可执行权限，例如：

```
$ chmod +x script
```

并在脚本的顶部放置一个合适的 Shebang 线。一个很好的选择通常是：

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

将在整个 PATH 中搜索 Python 解释器。但是，某些 Unix 系统可能没有 **env** 命令，因此可能需要将 /usr/bin/python3 硬编码为解释器路径。

要在 Python 脚本中使用 shell 命令，请查看 subprocess 模块。

CHAPTER 3

# 在 Windows 上使用 Python

本文档旨在概述在 Microsoft Windows 上使用 Python 时应了解的特定于 Windows 的行为。

与大多数 UNIX 系统和服务不同，Windows 系统没有预安装 Python。多年来 CPython 团队已经编译了每一个发行版 的 Windows 安装程序（MSI 包），已便 Windows 用户下载和安装。这些安装程序主要用于每个用户单独安装 Python 时，添加核心解释器和库。安装程序还可以为一台机器的所有用户安装，并且可以为应用程序本地分发提供单独的 zip 文件。

如 **PEP 11** 中所述，Python 仅支持微软产品支持生命周期内的 Windows 版本。这意味着 Python 3.8 支持 Windows Vista 和更新版本。如果需要 Windows XP 支持，请安装 Python 3.4。

Windows 提供了许多不同的安装程序，每个安装程序都有一定的优点和缺点。

完整安装程序 内含所有组件，对于使用 Python 进行任何类型项目的开发人员而言，它是最佳选择。

*Microsoft Store* 包 是一个简单的 Python 安装，适用于运行脚本和包，以及使用 IDLE 或其他开发环境。它需要 Windows 10，但可以安全地安装而不会破坏其他程序。它还提供了许多方便的命令来启动 Python 及其工具。

*nuget.org* 安装包 是用于持续集成系统的轻量级安装。它可用于构建 Python 包或运行脚本，但不可更新且没有用户界面工具。

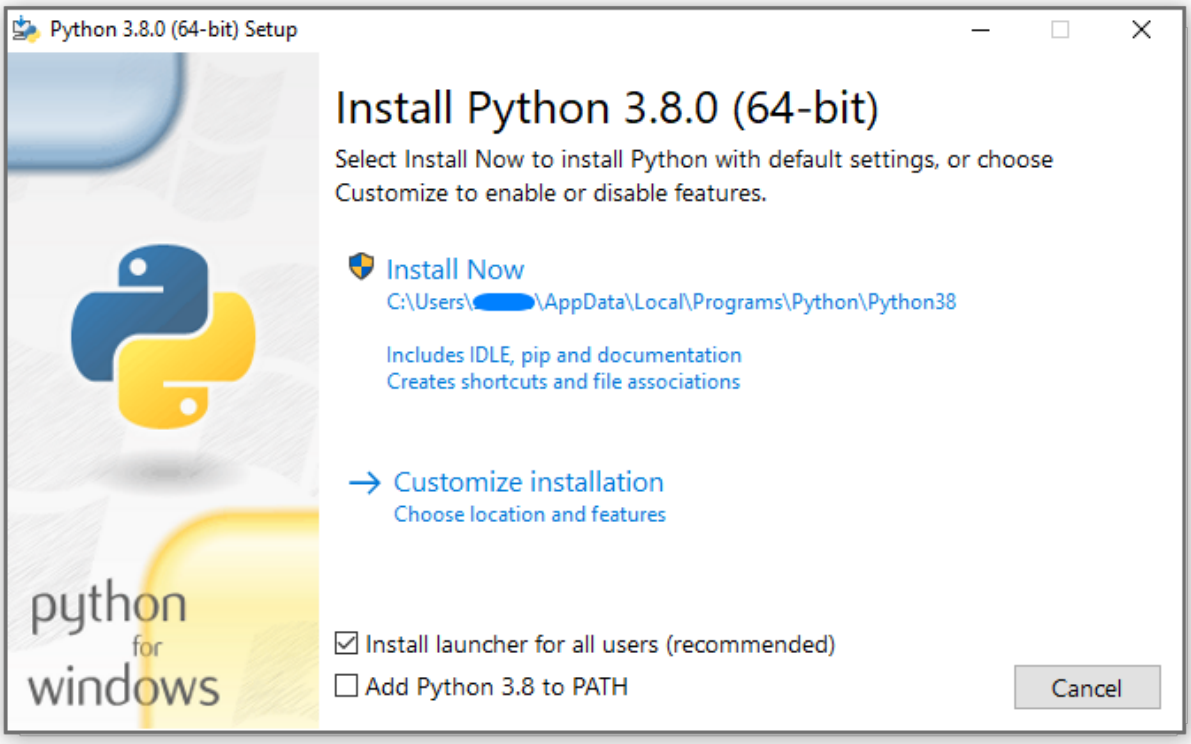
可嵌入的包 是 Python 的最小安装包，适合嵌入到更大的应用程序中。

## 3.1 完整安装程序

### 3.1.1 安装步骤

四个 Python 3.8 安装程序可供下载 - 32 位和 64 位版本的各有两个。*web installer* （网络安装包）是一个小的初始化工具，它将在安装过程中，根据需要自动下载所需的组件。*offline installer* （离线安装包）内含默认安装所需的组件，可选择功能仍需要 Internet 连接下载。请参阅當安裝時不下載 以了解在安装过程中避免下载的其他方法。

启动安装程序后，可以选择以下两个选项之一：




如果你選擇「馬上安裝」:

- 您 不需要成为管理员（除非需要对 C 运行库进行系统更新，或者为所有用户安装*适用于 Windows 的 Python 启动器*）
- Python 将安装到您的用户目录中
- *适用于 Windows 的 Python 启动器* 将根据第一页底部的选项安装
- 将安装标准库，测试套件，启动器和 pip
- 如果選擇，安装目🄵将被加入到你的 `PATH`
- 安装捷徑将只能被目前使用者所看見

選擇「客🄵化安装」將允許你選擇所需的項目進行安装，安装位置與其他選擇或安装後的所需進行的動作。你將需要使用此選項「除錯特徵」或「二進位方式」進行安装。

如要为全部用户安装，应选择“自定义安装”。在这种情况下:

- 您可能需要提供管理凭据或批准
- Python 将安装到 Program Files 目录中
- *适用于 Windows 的 Python 启动器* 将安装到 Windows 目录中
- 安装期间可以选择可选功能
- 标准库可以预编译为字节码
- 如果选中，安装目录将添加到系统 `PATH`
- 捷徑將被所有使用者所見

### 3.1.2 删除 MAX_PATH 限制

历史上 Windows 的路径长度限制为 260 个字符。这意味着长于此的路径将无法解决并导致错误。

在最新版本的 Windows 中，此限制可被扩展到大约 32,000 个字符。但需要让管理员激活 “启用 Win32 长路径” 组策略，或在注册表键 `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem` 中设置 `LongPathsEnabled` 为 `1`。

这允许 `open()` 函数，`os` 模块和大多数其他路径功能接受并返回长度超过 260 个字符的路径。

更改上述选项后，无需进一步配置。

3.6 版更變: Python 中启用了对长路径的支持。

### 3.1.3 安装排除使用者介面

安装程序 UI 中的所有选项也可以从命令行指定，允许脚本安装程序在许多机器上复制安装，而无需用户交互。还可以在不禁用 UI 的情况下设置这些选项，以更改一些默认值。

要完全隐藏安装程序 UI 并静默安装 Python，请使用 `/quiet` 选项。要跳过用户交互但仍然显示进度和错误，请使用 `/passive` 选项。可以通过 `/uninstall` 选项立即开始删除 Python - - 不会显示任何提示。

所有其他选项都传递为 `name=value`，其中值通常是 `0` 来禁用某个特性，`1` 来启用某个特性或路径。可用选项的完整列表如下所示。

| 名称 | 描述 | 預設 |
|---|---|---|
| InstallAllUsers | 为所有用户安装。 | 0 |
| TargetDir | 安装目🄵 | 基于 InstallAllUsers 选择 |
| DefaultAllUsersTargetDir | 为所有用户安装时的默认安装路径 | `%ProgramFiles%\Python X.Y` 或 `%ProgramFiles(x86)%\Python X.Y` |
| DefaultJustForMeTargetDir | 預設安装目🄵給只有給我安装方式 | `%LocalAppData%\Programs\PythonXY` 或 `%LocalAppData%\Programs\PythonXY-32` 或 `%LocalAppData%\Programs\PythonXY-64` |
| DefaultCustomTargetDir | UI 中显示的默认自定义安装目录 | （空） |
| AssociateFiles | 當執行程序也被安裝時創造檔案關聯 | 1 |
| CompileAll | 編譯所有 `.py` 檔案🄵 `.pyc`。 | 0 |
| PrependPath | 将 install 和 Scripts 目录添加到 `PATH` 以及将 `.PY` 添加到 `PATHEXT` | 0 |
| Shortcuts | 如果已安装，为解释器，文档和 IDLE 创建快捷方式 | 1 |
| Include_doc | 安装 Python 文件 | 1 |
| Include_debug | 安装调试二进制文件 | 0 |
| Include_dev | 安装开发人员头文件和库 | 1 |
| Include_exe | 安装 `python.exe` 及相关文件 | 1 |
| Include_launcher | 安装*适用于 Windows 的 Python 启动器*. | 1 |
| InstallLauncherAllUsers | 为所有用户安装*适用于 Windows 的 Python 启动器*。 | 1 |
| Include_lib | 安装标准库和扩展模块 | 1 |
| Include_pip | 安装捆绑的 pip 和 setuptools | 1 |
| Include_symbols | 安装调试符号 (*.pdb) | 0 |
| Include_tcltk | 安装 Tcl/Tk 支持和 IDLE | 1 |
| Include_test | 安装标准库测试套件 | 1 |
| Include_tools | 安装实用程序脚本 | 1 |
| LauncherOnly | 仅安装启动器。这将覆盖大多数其他选项。 | 0 |
| SimpleInstall | 禁用大多数安装 UI | 0 |
| SimpleInstallDescription | 使用简化安装 UI 时显示的自定义消息。 | （空） |

例如，要以静默方式全局安装默认的 Python，您可以（在命令提示符 >）使用以下命令:

```
python-3.8.0.exe /quiet InstallAllUsers=1 PrependPath=1 Include_test=0
```

要允许用户在没有测试套件的情况下轻松安装 Python 的个人副本，可以使用以下命令提供快捷方式。这将显示一个简化的初始页面，不允许自定义:

```
python-3.8.0.exe InstallAllUsers=0 Include_launcher=0 Include_test=0
    SimpleInstall=1 SimpleInstallDescription="Just for me, no test suite."
```

（请注意，省略启动器也会省略文件关联，并且仅在全局安装包含启动器时才建议用于每用户安装。）

上面列出的选项也可以在一个名为 `unattend.xml` 的文件中与可执行文件一起提供。此文件指定选项和值的列表。作为属性提供的值，（如果可能）它将转换为数字。作为文本提供的值，始终保留为字符串。此示例文件设置与上一示例采用相同的选项:

```
<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 當安裝時不下載

由于下载的初始安装包中未包含 Python 的某些可选功能，如果选择安装这些功能可能需要 Internet 连接。为了避免这种需要，可以按需下载所有可能的组件，以创建一个完整的布局，该布局将不再需要 internet 连接，而不管所选择的特性是什么。请注意，此下载可能比要求的要大，但是如果要执行大量安装，则拥有本地缓存 的副本非常有用。

从命令提示符执行以下命令以下载所有可能的必需文件。请记住将 `python-3.8.0.exe` 替换为安装程序的实际名称，并在自己的目录中创建布局，以避免同名的文件之间发生冲突。

```
python-3.8.0.exe /layout [optional target directory]
```

您也可以指定 `/quiet` 选项来隐藏进度显示。

### 3.1.5 修改安装

安装 Python 后，您可以通过 Windows 中的“程序和功能”工具添加或删除功能。选择 Python 条目并选择“卸载/更改”以在维护模式下打开安装程序。

“修改”允许您通过修改复选框来添加或删除功能 - 未更改的复选框将不会安装或删除任何内容。在此模式下无法更改某些选项，例如安装目录；要修改这些，您需要完全删除然后重新安装 Python。

“修复”将使用当前设置验证应安装的所有文件，并替换已删除或修改的任何文件

“卸载”将完全删除 Python，但*适用于 Windows 的 Python 启动器* 除外，它在“程序和功能”中有自己的条目。

## 3.2 Microsoft Store 包

3.7.2 版新加入.

Microsoft Store 包是一个易于安装的 Python 解释器，主要用于交互式使用，例如，学生。

要安装软件包，请确保您拥有最新的 Windows 10 更新，并在 Microsoft Store 应用程序中搜索"Python 3.8" 。确保您选择的应用程序由 Python Software Foundation 发布并安装。

> **警告：** Python 将始终在 Microsoft Store 上免费提供。如果要求您付款，则表示您没有选择正确的包。

安装完成后，可以在开始菜单中找到它来启动 Python。或者可以在命令提示符或 PowerShell 会话中输入 `python` 来启动。此外可以输入 `pip` 或 `idle` 来使用 pip 和 IDLE。IDLE 也在开始菜单中。

所有这三个命令也可以使用版本号后缀，例如，`python3.exe` 和 `python3.x.exe` 以及 `python.exe`（其中 `3.x` 是您要启动的特定版本，例如 3.8 ）。在 `设置--> 主页--> 应用和功能`页面中，点选 `管理可选功能`，选择与每个命令关联的 python 版本。建议确保 `pip` 和 `idle` 与选择的 `python` 版本一致。

可以使用 `python -m venv` 创建虚拟环境并激活并正常使用。

如果你已经安装了另一个版本的 Python 并将它添加到你的 `PATH` 变量中，那么它将作为 `python.exe` 而不是来自 Microsoft Store 的那个。要访问新安装，请使用 `python3.exe` 或 `python3.x.exe` 。

`py.exe` 启动器将检测此 Python 安装版，但会优先使用来自传统安装器的安装版。

要删除 Python，请打开“设置”并使用“应用程序和功能”，或者在“开始”中找到 Python，然后右键单击以选择“卸载”。卸载将删除该已安装 Python 程序中的所有软件包，但不会删除任何虚拟环境

### 3.2.1 已知的问题

由于 Microsoft Store 应用程序的限制，Python 脚本可能无法对共享位置（如 `TEMP` ）和注册表进行完全写入访问。相反，它将写入私人副本。如果脚本必须修改共享位置，则需要安装完整安装程序。

有关此限制的技术原理的更多细节，请查询 Microsoft 已打包完全可信应用的文档，当前位于 docs.microsoft.com/en-us/windows/msix/desktop/desktop-to-uwp-behind-the-scenes

## 3.3 nuget.org 安装包

3.5.2 版新加入.

nuget.org 是一个精简的 Python 环境，用于在没有全局安装 Python 的系统的持续集成和构建。虽然 Nuget 是“.NET 的包管理器”，但是对于包含构建时工具的包来说，它也可以很好地工作。

访问 nuget.org 获取有关使用 nuget 的最新信息。下面的摘要对 Python 开发人员来说已经足够了。

`nuget.exe` 命令行工具可以直接从 `https://aka.ms/nugetclidl` 下载，例如，使用 curl 或 PowerShell。使用该工具安装 64 位或 32 位最新版本的 Python:

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

要选择特定版本，请添加 `-Version 3.x.y` 。输出目录可以从 `.` 更改，包将安装到子目录中。默认情况下，子目录的名称与包的名称相同，如果没有 `-ExcludeVersion` 选项，则此名称将包含已安装的特定版本。子目录里面是一个包含 Python 安装的 `tools` 目录:

```
# 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
```

通常，nuget 包不可升级，应该平行安装较新版本并使用完整路径引用。或者，手动删除程序包目录并再次安装。如果在构建之间不保留文件，许多 CI 系统将自动执行此操作。

除了 `tools` 目录外，还有一个 `build\native` 目录。它包含一个 MSBuild 属性文件 `python.props`，可以在 C++ 项目中使用该文件来引用 Python 安装。包含这些设置将自动在生成中使用标头和导入库。

nuget.org 上的包信息页是 www.nuget.org/packages/python 对于 64 位版本和 www.nuget.org/packages/pythonx86 表示 32 位版本。

## 3.4 可嵌入的包

3.5 版新加入.

嵌入式发行版是一个包含最小 Python 环境的 ZIP 文件。它旨在作为另一个应用程序的一部分，而不是由最终用户直接访问。

解压缩后，嵌入式发行版（几乎）与用户系统完全隔离，包括环境变量、系统注册表设置和已安装的软件包。标准库作为预先编译和优化的 `.pyc` 文件包含在 ZIP 中，并提供了 `python3.dll`，`python37.dll`，`python.exe` 和 `pythonw.exe` 文件。不包括 Tcl/tk（包括所有依赖项，如 Idle），pip 和 Python 文档。

---

**備🄴:** 嵌入式发行版不包括 Microsoft C Runtime，应用程序安装程序负责提供此功能。运行时可能已经预先安装在用户的系统上或通过 Windows Update 自动安装，并且可以通过在系统目录中找到 `ucrtbase.dll` 来检测

---

---

**備🄵:** When running on Windows 7, Python 3.8 requires the KB2533623 update to be installed. The embeddable distribution does not detect this update, and may fail at runtime. Later versions of Windows include this update.

---

第三方软件包应该由应用程序与嵌入式发行版一起安装。这个发行版不支持像常规 Python 安装那样使用 pip 来管理依赖关系，不过可以小心地将 pip 包含进来并使用它进行自动更新。通常，第三方包应该作为应用程序的一部分 ( “打包” ) 处理，以便开发人员在向用户提供更新之前能够确保与新版本兼容。

下面描述了这个发行版的两个推荐用例。

### 3.4.1 Python 应用程序

用 Python 编写的应用程序并不一定要求用户了解这一事实。在这种情况下，可以使用嵌入式发行版在安装包中包含 Python 的私有版本。根据它应该有多透明 (或者相反，它应该看起来有多专业)，有两个选项。

使用专门的可执行文件作为启动程序需要一些编码，但为用户提供了最透明的体验。使用定制的启动器，没有明显的迹象表明程序是在 Python 上运行的：图标可以定制，公司和版本信息可以指定，文件关联可以正常运行。在大多数情况下，自定义启动程序应该只需使用硬编码的命令行就能调用 `Py_Main`

更简单的方法是提供批处理文件或生成的快捷方式，使用所需的命令行参数直接调用 `python.exe` 或 `pythonw.exe` 。在这种情况下，应用程序将显示为 Python 而不是其实际名称，并且用户可能无法将其与其他正在运行的 Python 进程或文件关联区分开来。

对于后一种方法，包应该与 Python 可执行文件一起作为目录安装，以确保它们在路径上可用。使用专用的启动器，包可以位于其他位置，因为在启动应用程序之前有机会指定搜索路径。

### 3.4.2 嵌入 Python

用本地代码编写的应用程序通常需要某种形式的脚本语言，嵌入式 Python 发行版可以用于此目的。通常，应用程序的大部分都是本机代码，某些部分将调用 `python.exe` 或直接使用 `python3.dll` 。无论是哪种情况，将嵌入的发行版解压缩到应用程序安装的子目录中就足以提供可加载的 Python 解释器。

与应用程序使用一样，包可以安装到任何位置，因为在初始化解释器之前有机会指定搜索路径。否则，使用嵌入式发行版和常规安装之间没有根本区别。

## 3.5 替代捆绑包

除了标准的 CPython 发行版之外，还有一些包含附加功能的修改包。以下是热门版本及其主要功能的列表：

**ActivePython** 具有多平台兼容性的安装程序，文档，PyWin32

**Anaconda** 流行的科学模块（如 numpy，scipy 和 pandas）和 `conda` 包管理器。

**Canopy** 具有编辑器和其他开发工具的“全面的 Python 分析环境”。

**WinPython** 特定于 Windows 的发行版，包含用于构建包的预构建科学包和工具。

请注意，这些软件包可能不包含最新版本的 Python 或其他库，并且不由核心 Python 团队维护或支持。

## 3.6 設定 Python

要从命令提示符方便地运行 Python，您可以考虑在 Windows 中更改一些默认环境变量。虽然安装程序提供了为您配置 PATH 和 PATHEXT 变量的选项，但这仅适用于单版本、全局安装。如果您经常使用多个版本的 Python，请考虑使用*适用于 Windows 的 Python 启动器* 。

### 3.6.1 附录：设置环境变量

Windows 允许在用户级别和系统级别永久配置环境变量，或临时在命令提示符中配置环境变量。

要临时设置环境变量，请打开命令提示符并使用 **set** 命令：

```
C:\>set PATH=C:\Program Files\Python 3.8;%PATH%
C:\>set PYTHONPATH=%PYTHONPATH%;C:\My_python_lib
C:\>python
```

这些环境变量的更改将应用 于在该控制台中执行的任何其他命令，并且，由该控制台启动的任何应用程序都继承设这些设置。

在百分号中包含的变量名将被现有值替换，允许在开始或结束时添加新值。通过将包含 **python.exe** 的目录添加到开头来修改 `PATH` 是确保启动正确版本的 Python 的常用方法。

要永久修改默认环境变量，请单击“开始”并搜索“编辑环境变量”，或打开“系统属性” *Advanced system settings* ，然后单击 *Environment Variables* 按钮。在此对话框中，您可以添加或修改用户和系统变量。要更改系统变量，您需要对计算机进行无限制访问（即管理员权限）。

---

**備🄵:** Windows 会将用户变量串联的系统变量 之后，这可能会在修改 `PATH` 时导致意外结果。

*`PYTHONPATH`* 变量被 Python 2 和 Python 3 的所有版本使用，因此除非它只包含与所有已安装的 Python 版本兼容的代码，否则不要永久配置此变量。

---

**也參考:**

**https://www.microsoft.com/en-us/wdsi/help/folder-variables** 環境變數於 Windows NT

**https://technet.microsoft.com/en-us/library/cc754250.aspx** 用于临时修改环境变量的 SET 命令

**https://technet.microsoft.com/en-us/library/cc755104.aspx** 用于永久修改环境变量的 SETX 命令

**https://support.microsoft.com/en-us/help/310519/how-to-manage-environment-variables-in-windows-xp** 如何管理環境變數於 Windows XP

**https://www.chem.gla.ac.uk/~louis/software/faq/q1.html** 設定環境變數 - Louis J. Farrugia

### 3.6.2 查找 Python 可执行文件

3.5 版更變.

除了使用自动创建的 Python 解释器的开始菜单项之外，您可能还想在命令提示符下启动 Python。安装程序有一个选项可以为您设置。

在安装程序的第一页上，可以选择标记为“将 Python 添加到环境变量”的选项，以使安装程序将安装位置添加到 `PATH` 。还添加了 `Scripts\` 文件夹的位置。这允许你输入 **`python`** 来运行解释器，并且 **`pip`** 用于包安装程序。因此，您还可以使用命令行选项执行脚本，请参阅命令行 文档。

如果在安装时未启用此选项，则始终可以重新运行安装程序，选择“修改”并启用它。或者，您可以使用附录：设置环境变量 的方法手动修改 `PATH` 。您需要将 Python 安装目录添加到 `PATH` 环境变量中，该内容与其他条目用分号分隔。示例变量可能如下所示（假设前两个条目已经存在）：

```
C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files\Python 3.8
```

## 3.7 UTF-8 模式

3.7 版新加入.

Windows 仍然使用传统编码格式作为系统的编码格式（ANSI 代码页）。Python 使用它作为文本文件默认的编码格式 (即 `locale.getpreferredencoding()`)。

这可能会造成问题，因为因特网和大多数 Unix 系统包括 WSL (Windows Subsystem for Linux) 广泛使用 UTF-8。

你可以使用 UTF-8 模式将默认的文本编码格式改为 UTF-8。要启用 UTF-8 模式可以通过 `-X utf8` 命令行选项，或者 `PYTHONUTF8=1` 环境变量。请参见 *`PYTHONUTF8`* 了解如何启用 UTF-8 模式，并参见附录：设置环境变量 了解如何修改环境变量。

当 UTF-8 模式被启用时：

- `locale.getpreferredencoding()` 将返回 `'UTF-8'` 而不是系统的编码格式。此函数可用来在许多场合下获取默认的文本编码格式，包括 `open()`, `Popen`, `Path.read_text()` 等等。

- `sys.stdin`, `sys.stdout` 和 `sys.stderr` 都将使用 UTF-8 作为其文本编码格式。
- 你仍然可以通过"mbcs" 编解码器来使用系统的编码格式。

请注意添加 `PYTHONUTF8=1` 到默认环境变量将会影响你的系统中的所有 Python 3.7+ 应用。如果你有任何 Python 3.7+ 应用仍然依赖于传统的系统编码格式，则推荐设置临时环境变量或使用 `-X utf8` 命令行选项。

---

**備🄴:** 即使在不启用 UTF-8 模式时，Windows 版的 Python 也会在以下情况中默认使用 UTF-8：

- 控制台 I/O 包括标准 I/O (详情见 **PEP 528**)。
- 文件系统编码格式 (详情见 **PEP 529**)。

---

## 3.8 适用于 Windows 的 Python 启动器

3.3 版新加入.

用于 Windows 的 Python 启动器是一个实用程序，可帮助定位和执行不同的 Python 版本。它允许脚本（或命令行）指示特定 Python 版本的首选项，并将定位并执行该版本。

与 `PATH` 变量不同，启动器将正确选择最合适的 Python 版本。它更倾向于按用户安装而不是系统安装，并按语言版本排序，而不是使用最新安装的版本。

启动器最初是在 **PEP 397** 中指定的。

### 3.8.1 開始

#### 从命令行

3.6 版更變.

全局安装 Python 3.3 及更高版本将把启动器放在你的 `PATH` 上。启动程序与所有可用的 Python 版本兼容，因此安装哪个版本无关紧要。要检查启动程序是否可用，请在命令提示符中执行以下命令：

```
py
```

您应该会发现已安装的最新版本的 Python 已启动 - 它可以正常退出，并且将指定的任何其他命令行参数直接发送到 Python。

如果您安装了多个版本的 Python（例如，2.7 和 3.8 ），您会注意到 Python 3.8 启动 - 如果要启动 Python 2.7，尝试命令：

```
py -2.7
```

如果您想使用 Python 2.x 的最新版本，请尝试以下命令：

```
py -2
```

你会发现 Python 2.x 的最新版本已启动。

如果您看到以下错误，则表明您没有安装启动器：

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

除非在安装时选择了该选项，单个用户安装的 Python 不会将启动程序添加到 `PATH` 。

#### 🄵擬環境（Virtual environment）

3.5 版新加入.

如果启动程序运行时没有明确的 Python 版本，并且虚拟环境（使用标准库创建 `venv` 模块或外部 `virtualenv` 工具）处于活动状态，则启动程序将运行虚拟环境的解释器而不是全局的。要运行全局解释器，请停用虚拟环境，或显式指定全局 Python 版本。

#### 从脚本

让我们创建一个测试 Python 脚本 - 创建一个名为 “hello.py“的文件，其中包含以下内容

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

从 hello.py 所在的目录中，执行以下命令：

```
py hello.py
```

您应该注意到最新的 Python 2.x 安装的版本号已打印出来。现在尝试将第一行更改为：

```
#! python3
```

现在，重新执行该命令应该打印最新的 Python 3.x 信息。与上面的命令行示例一样，你可以更明确的指定版本限定符。假设您安装了 Python 2.6，请尝试将第一行更改为 `#! python2.6`，你会发现打印的 2.6 版本信息。

请注意，与交互式使用不同，裸 “python” 将使用您已安装的 Python 2.x 的最新版本。这是为了向后兼容及兼容 Unix，其中命令 `python` 通常是指 Python 2。

#### 從檔案關聯

安装时应该将启动器与 Python 文件（即 `.py`, `.pyw`, `.pyc` 文件）相关联。这意味着当您从 Windows 资源管理器中双击其中一个文件时，将使用启动程序，因此您可以使用上述相同的工具让脚本指定应使用的版本。

这样做的主要好处是，单个启动程序可以同时支持多个 Python 版本，具体取决于第一行的内容。

### 3.8.2 Shebang Lines

如果脚本文件的第一行以 `#!` 开头，则称为”shebang” 行。Linux 和其他类 Unix 操作系统都有对这些行的本机支持，它们通常在此类系统上用来指示应该如何执行脚本。这个启动器允许在 Windows 上对 Python 脚本使用相同的工具，上面的示例演示了它们的使用。

为了允许 Python 脚本中的 shebang 行在 Unix 和 Windows 之间移植，该启动器支持许多 “虚拟” 命令来指定要使用的解释器。支持的虚拟命令是：

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

例如，如果脚本开始的第一行

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

将找到并使用默认的 Python。因为在 Unix 上编写的许多 Python 脚本已经有了这一行，你应该发现这些脚本可以由启动器使用而无需修改。如果您在 Windows 上编写一个新脚本，希望在 Unix 上有用，那么您应该使用以 `/usr` 开头的一个 shebang 行。

任何上述虚拟命令都可以显式指定版本（可以仅为主要版本，也可以为主要版本加次要版本）作为后缀。此外，可以通过在次要版本之后添加“-32”来请求 32 位版本。例如 `/usr/bin/python2.7-32` 将请求使用 32 位 python 2.7。

3.7 版新加入: 从 python 启动程序 3.7 开始，可以通过“-64”后缀调用 64 位版本。此外，可以指定没有次要的主要和架构（即 `/usr/bin/python3-64` ）。

shebang line 的 `/usr/bin/env` 形式还有一个特殊属性。在寻找已安装的 Python 解释器之前，此表单将搜索可执行文件 `PATH` 以获取 Python 可执行文件。这对应于 Unix 中 `env` 程序的行为，该程序将在 `PATH` 执行搜索。

### 3.8.3 shebang lines 的参数

shebang lines 还可以指定要传递给 Python 解释器的其他选项。例如，如果你有一个 shebang lines：

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

然后 Python 将以 `-v` 选项启动

### 3.8.4 自定义

#### 通过 INI 文件自定义

启动程序将搜索两个.ini 文件 - 在当前用户的”application data” 目录中搜索 `py.ini` （即通过使用 `CSIDL_LOCAL_APPDATA` 调用 Windows 函数 `SHGetFolderPath` 返回的目录）以及与启动器位于同一目录中的 `py.ini` 。相同的.ini 文件既用于启动器的“控制台”版本（即 py.exe），也用于“windows”版本（即 pyw.exe）

“应用程序目录”中指定的自定义优先于可执行文件旁边.ini 文件的自定义，因此对启动程序旁边的.ini 文件不具有写访问权限的用户可以覆盖该全局.ini 文件中的命令。

#### 自定义默认的 Python 版本

在某些情况下，可以在命令中包含版本限定符，以指定命令将使用哪个 Python 版本。版本限定符以主版本号开头，可以选择后跟 (’.’) 和次版本说明符。此外，可以通过添加”-32” 或“-64”来指定是请求 32 位还是 64 位实现。

例如，一个 shebang line 的 `#!python` 行没有版本限定符，而 `#!python3` 有一个版本限定符，它只指定一个主要版本。

如果在命令中找不到版本限定符，则可以设置环境变量 `PY_PYTHON` 以指定默认版本限定符。如果未设置，则默认为”3”。该变量可以指定能通过命令行传递的任何值，比如”3”, ”3.7”, ”3.7-32” 或”3.7-64”。（请注意”-64” 选项仅适用于 Python 3.7 或更高版本中包含的启动器。）

如果没有找到次要版本限定符，则可以设置环境变量 `PY_PYTHON{major}` （其中 `{major}` 是上面确定的当前主要版本限定符）以指定完整版本。如果没有找到这样的选项，启动器将枚举已安装的 Python 版本并使用为主要版本找到的最新次要版本，尽管不能保证，但该版本可能是该系列中最新安装的版本。

在安装了相同（major.minor）Python 版本的 32 位和 64 位的 64 位 Windows 上，64 位版本将始终是首选。对于启动程序的 32 位和 64 位实现都是如此 -- 这对于启动程序 32 位和 64 位都是正确的 -- 如果可用，32 位启动程序将倾向于执行指定版本的 64 位 Python 安装。这样就可以预测启动器的行为，只知道 PC 上安装了哪些版本，而不考虑它们的安装顺序（即，不知道 32 位或 64 位版本的 Python 和相应的启动器是否是最后安装）。如上所述，可以在版本说明符上使用可选的“-32”或“-64”后缀来更改此行为。

示例：

- 如果没有设置相关选项，命令 `python` 和 `python2` 将使用安装的最新 Python 2.x 版本，命令 `python3` 将使用最新安装的 Python 3.x.
- 命令 `python3.1` 和 `python2.7` 根本不会查阅任何选项，因为版本已完全指定。
- 如果 `PY_PYTHON=3` ，命令 “python“ 和 `python3` 都将使用最新安装的 Python 3 版本。
- 如果 `PY_PYTHON=3.1-32` ，命令 `python` 将使用 3.1 的 32 位实现，而命令 `python3` 将使用最新安装的 Python（PY_PYTHON 根本没有被视为指定了主要版本。）
- 如果 `PY_PYTHON=3` 且 `PY_PYTHON3=3.1`，命令 `python` 和 `python3` 都将特别使用 3.1

除环境变量外，还可以在启动程序使用的.INI 文件中配置相同的设置。INI 文件中的部分称为 `[defaults]` ，键名称将与没有前导 `PY_` 前缀的环境变量相同（并注意 INI 文件中的键名不区分大小写）。）环境变量的内容将覆盖 INI 文件中指定的内容。

例如:

- 设置 `PY_PYTHON=3.1` 等同于包含以下内容的 INI 文件：

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

- 设置 `PY_PYTHON=3` 和 `PY_PYTHON3=3.1` 相当于包含以下内容的 INI 文件：

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

### 3.8.5 诊断

如果设置了环境变量 `PYLAUNCH_DEBUG` （任何值），启动器将诊断信息打印到 stderr（即：控制台）。虽然这些信息同时具有冗长 和简洁性，但它应该允许您查看 Python 的版本、选择特定版本的原因以及用于执行目标 Python 的确切命令行。

## 3.9 查找模块

Python 通常将其库（以及您的 site-packages 文件夹）存储在安装目录中。因此，如果您已将 Python 安装到 `C:\Python\` ，则默认库将驻留在 `C:\Python\Lib\` 中，第三方模块存储在 `C:\Python\Lib\site-packages\` 。

若要完全覆盖 `sys.path` ，请创建与 DLL(`python37._pth`) 或可执行文件 ( “python._pth“ ) 同名的 `._pth` 文件，并为要添加的每个路径指定一行 `sys.path` 。基于 DLL 名称的文件覆盖基于可执行文件的文件，如果需要，可以为加载运行时的任何程序限制路径。

当文件存在时，将忽略所有注册表和环境变量，启用隔离模式，并且：除非文件中的一行指定 `import site` ，否则不会导入 `site` 。以 #开头的空白路径和行将被忽略。每个路径可以是绝对的或相对于文件的位置。不允许使用除 `site` 以外的导入语句，并且不能指定任意代码。

请注意，当指定 `import site` 时，`.pth` 文件（没有前导下划线）将由 `site` 模块正常处理。

当找不到 `._pth` 文件时，`sys.path` 是如何在 Windows 上填充的：

- 在开始时，添加一个空条目，该条目对应于当前目录。
- 如果环境变量 *PYTHONPATH* 存在，如环境变量 中所述，则接下来添加其条目。请注意，在 Windows 上，此变量中的路径必须用分号分隔，以区别于驱动器标识符中使用的冒号（`C:\` 等）。
- 附加的"application paths" 可以同时添加到注册表 ' *HKEY_CURRENT_USER*' 和 `HKEY_LOCAL_MACHINE` 分支下的:samp:*\SOFTWARE\Python\PythonCore\{version}\PythonPath* 中作为子键。以分号分隔的路径字符串作为默认值的子键将导致每个路径添加到 `sys.path` 。（请注意，所有已知的安装程序都只使用 HKLM，因此 HKCU 通常为空。）
- 如果设置了环境变量 *PYTHONHOME* ，则将其假定为 "Python 主目录"。否则，主 Python 可执行文件的路径用于定位 "landmark 文件"（`Lib\os.py` 或 `pythonXY.zip` ）以推断" Python 主目录 "。如果找到了 Python 主目录，则基于该文件夹将相关的子目录添加到 `sys.path` （`Lib`，`plat-win` 等）。否则，核心 Python 路径是从存储在注册表中的 PythonPath 构造的。
- 如果找不到 Python Home，也没有指定 *PYTHONPATH* 环境变量，并且找不到注册表项，则使用具有相对条目的默认路径（例如 `.\Lib; .\plat-win` 等等）。

如果在主可执行文件旁边或在可执行文件上一级的目录中找到 `pyvenv.cfg` 文件，则以下变体适用：

- 如果 "home" 是一个绝对路径，并且 *PYTHONHOME* 未设置，则在推断起始位置时使用此路径而不是主可执行文件的路径

最終這所有的結果🄴：

- 运行 `python.exe` ，或主 Python 目录中的任何其他.exe（安装版本，或直接来自 PCbuild 目录）时，推导出核心路径，并忽略注册表中的核心路径。始终读取注册表中的其他 "应用程序路径"。
- 当 Python 托管在另一个.exe（不同的目录，通过 COM 嵌入等）时，将不会推断出 "Python Home"，因此使用了来自注册表的核心路径。始终读取注册表中的其他 "应用程序路径"。
- 如果 Python 找不到它的主目录并且没有注册表值（冻结的.exe，一些非常奇怪的安装设置），那么你会得到一条带有一些默认但相对的路径的路径。

对于那些想要将 Python 捆绑到其应用程序或发行版中的人，以下建议将防止与其他安装冲突：

- 在您的可执行文件中包含一个 `._pth` 文件，其中包含目录。这将忽略注册表和环境变量中列出的路径，并忽略 `site` ，除非列出 `import site` 。
- 如果你在自己的可执行文件中加载 `python3.dll` 或 `python37.dll` ，在 `Py_Initialize()` 之前，要显式调用 `Py_SetPath()` 或（至少）`Py_SetProgramName()`
- 清除和/或覆盖 *PYTHONPATH* 并在启动来自应用程序的 `python.exe` 之前设置 *PYTHONHOME* 。
- 如果您不能使用前面的建议（例如，您是一个允许人们直接运行：file:*python.exe*'的分发版），请确保安装目录中存在 *landmark* 文件（*:file:'Lib\os.py*）。（请注意，在 zip 文件中不会检测到该文件，但会检测到正确命名的 zip 文件。）

这些将确保系统范围安装中的文件不会优先于与应用程序捆绑在一起的标准库的副本。否则，用户可能会在使用您的应用程序时遇到问题请注意，第一个建议是最好的，因为其他建议可能仍然容易受到注册表和用户站点包中的非标准路径的影响。

3.6 版更變:

- 添加 `._pth` 文件支持并从 `pyvenv.cfg` 中删除 `applocal` 选项
- 当直接与可执行文件相邻时，添加 `pythonXX.zip` 作为潜在的 landmark 。

3.6 版後已🄵用：在 `Modules` （不是 `PythonPath` ）下的注册表中指定的模块可以通过以下方式导入 `importlib.machinery.WindowsRegistryFinder` 。在 Windows 上，此查找程序在 3.6.0 及更早版本的可用，但可能需要在将来显式添加到 `sys.meta_path`

## 3.10 附加模块

尽管 Python 的目标是在所有平台中都可移植，但是 Windows 有一些独特的特性。在标准库和外部都有一些模块和代码片段在使用这些特性。

特定于 Windows 的标准模块记录在 mswin-specific-services 中。

### 3.10.1 PyWin32

Mark Hammond 的 PyWin32 模块是一组用于高级 Windows 特定支持的模块。这包括以下实用程序：

- 组件对象模型 (COM)
- Win32 API 调用
- 登🄵檔（Registry）
- 事件日🄵（Event log）
- Microsoft Foundation Classes (MFC) 用户界面

PythonWin 是 PyWin32 附带的一个示例 MFC 应用程序。它是一个内置调试器的可嵌入 IDE。

**也参考：**

**Win32 How Do I...?** Tim Golden 著

**Python and COM** David 和 Paul Boddie 著

### 3.10.2 cx_Freeze

cx_Freeze 是一个 `distutils` 的扩展（参见 extending-distutils ），它将 Python 脚本包装成可执行的 Windows 程序（`*.exe` 文件）。完成此操作后，您可以分发应用程序，而无需用户安装 Python。

### 3.10.3 WConio

由于 Python 的高级终端处理层 `curses` 仅限于类 Unix 系统，因此 Windows 还有一个独立的库：用于 Python 的 Windows 控制台 I/O .

WConio 是 Turbo-C 的 `CONIO.H` 装饰器, 用于创建文本用户界面。

## 3.11 編譯 Python 在 Windows

如果你想自己编译 CPython，首先要做的是获取 source. 。您可以下载最新版本的源代码，也可以重新签出 checkout. 。

源代码树包含 Microsoft Visual Studio 2015 的构建解决方案和项目文件，它是用于构建官方 Python 版本的编译器。这些文件位于 `PCbuild` 目录中。

检查 `PCbuild/readme.txt` 以获取有关构建过程的一般信息。

有关扩展模块，请参阅 building-on-windows 。

**也参考：**

**Python + Windows + distutils + SWIG + gcc MinGW** 或"Creating Python extensions in C/C++ with SWIG and compiling them with MinGW gcc under Windows" 或"Installing Python extension with distutils and without Microsoft Visual C++" by Sébastien Sauvage, 2003

## 3.12 其他平台

随着 Python 的不断发展，不再支持以前曾经支持的一些平台（由于缺少用户或开发人员）。检查 **PEP 11** 了解所有不支持的平台的详细信息。

- Windows CE 仍然受支持。
- Cygwin 安装包也提供安装 Python 解释器 (cf. Cygwin package source, Maintainer releases)

有关具有预编译安装程序平台的详细信息，请参阅 Python for Windows

CHAPTER 4

# 在麥金塔系統使用 Python

**作者** Bob Savage <bobsavage@mac.com>

Python 執行在麥金塔系統的 Mac OS X 和執行在其他 Unix 平台上原理非常相似，但有一些值得提出的是在 Mac OS X 上增加其他額外的功能例如 IDE 與套件管理。

## 4.1 取得和安裝 MacPython

Apple 在 Mac OS X 10.8 預設安裝 Python 2.7。但你也可以到 Python website （https://www.python.org ）更新至最新的 Python 3。Python 建立在" 通用二進位" 上，使 Python 能以本地程序的形式運行在使用英特爾微處理器與 PowerPC 麥金塔電腦上。

在安裝後你必須要做幾件事：

- 会有一个 `Python 3.8` 文件夹在你的 `Applications` 文件夹中。在这里你可以找到 IDLE，它是作为官方 Python 发行版标准组成部分的开发环境；以及 PythonLauncher，它负责处理在 Finder 中双击 Python 脚本的操作。
- 框架 `/Library/Frameworks/Python.framework` ，包括 Python 可执行文件和库。安装程序将此位置添加到 shell 路径。要卸载 MacPython ，你可以简单地移除这三个项目。Python 可执行文件的符号链接放在 /usr/local/bin/ 中。

Apple 提供的 Python 版本分别安装在 `/System/Library/Frameworks/Python.framework` 和 `/usr/bin/python` 中。你永远不应修改或删除这些内容，因为它们由 Apple 控制并由 Apple 或第三方软件使用。请记住，如果你选择从 python.org 安装较新的 Python 版本，那么你的计算机上将安装两个不同但都有用的 Python ，因此你的路径和用法与你想要执行的操作一致非常重要。

IDLE 包含一个帮助菜单，允许你访问 Python 文档。如果您是 Python 的新手，你应该开始阅读该文档中的教程介绍。

如果你熟悉其他 Unix 平台上的 Python ，那么你应该阅读有关从 Unix shell 运行 Python 脚本的部分。

### 4.1.1 如何執行 Python 🄵本

在 Mac OS X 上开始使用 Python 的最佳方法是通过 IDLE 集成开发环境，参见整合化開發工具 部分，并在 IDE 运行时使用“帮助”菜单。

如果要从终端窗口命令行或 Finder 运行 Python 脚本，首先需要一个编辑器来创建脚本。Mac OS X 附带了许多标准的 Unix 命令行编辑器，如 **vim** 和 **emacs** 。如果你想要一个更 Mac 化的编辑器，那么:program:*BBEdit* 或来自 Bare Bones Software 的 **TextWrangler**（参见 http://www.barebones.com/products/bbedit/index.html ）是不错的选择，就如 **TextMate**`（参见 **https://macromates.com/** ）。其他编辑器包括**:program:`Gvim`** (http://macvim-dev.github.io/macvim/ ）和 **Aquamacs** （http://aquamacs.org/ ）。

要从终端窗口运行脚本，必须确保:file:*/usr/local/bin* 位于 shell 搜索路径中。

從 Finder 執行你的🄵本時，你有兩個選項:

- 把脚本拖拽到 **PythonLauncher**
- 选择 **PythonLauncher** 作为通过 finder Info 窗口打开脚本（或任何.py 脚本）的默认应用程序，然后双击脚本。**PythonLauncher** 有各种首选项来控制脚本的启动方式。拖拽方式允许你为一次调用更改这些选项，或使用其“首选项”菜单全局更改内容。

### 4.1.2 透過使用者圖形介面執行🄵本

对于旧版本的 Python ，你需要注意一个 Mac OS X 的怪异之处：与 Aqua 窗口管理器通信的程序（换而言之，任何具有图形界面的程序）需要以特殊方式运行。使用 **pythonw** 而不是 **python** 来启动这样的脚本。

对于 Python 3.8，可以使用 **python** 或 **pythonw**

### 4.1.3 設定

OS X 上的 Python 遵循所有标准的 Unix 环境变量，例如 *PYTHONPATH* ，但是为 Finder 启动的程序设置这些变量是非标准的，因为 Finder 在启动时不读取你的 `.profile` 或 `.cshrc` 。你需要创建一个文件 `~/.MacOSX/environment.plist` 。有关详细信息，请参阅 Apple 的技术文档 QA1067 。

更多关于在 MacPython 中安装 Python 包的信息，参阅安装額外的 *Python* 包 部分。

## 4.2 整合化開發工具

MacPython 附带标准的 IDLE 开发环境。有关使用 IDLE 的详细介绍，请访问 http://www.hashcollision.org/hkn/python/idle_intro/index.html 。

## 4.3 安装額外的 Python 包

有几个方法可以安装额外的 Python 包:

- 可以通过标准的 Python distutils 模式（`python setup.py install`）安装软件包。
- 许多包也可以通过 **setuptools** 扩展或 **pip** 包装器安装，请参阅 https://pip.pypa.io/ 。

## 4.4 圖形化使用者介面 (GUI) 程式開發於 Mac

有許多建立圖形化使用者介面 (GUI) 應用程式選項使用 Python 於 Mac 上

*PyObjC* 是一个 Python 到 Apple 的 Objective-C/Cocoa 框架的绑定，这是大多数现代 Mac 开发的基础。有关 PyObjC 的信息，请访问 https://pypi.org/project/pyobjc/。

标准的 Python GUI 工具包是 `tkinter`，基于跨平台的 Tk 工具包（https://www.tcl.tk）。Apple 的 OS X 捆绑了 Aqua 原生版本的 Tk，最新版本可以从 https://www.activestate.com 下载和安装；它也可以从源代码构建。

*wxPython* 是另一种流行的跨平台 GUI 工具包，可在 Mac OS X 上本机运行。软件包和文档可从 https://www.wxpython.org 获得。

*PyQt* 是另一种流行的跨平台 GUI 工具包，可在 Mac OS X 上本机运行。更多信息可在 https://riverbankcomputing.com/software/pyqt/intro 上找到。

## 4.5 貢獻 Python 應用程式於 Mac

在 Mac 上部署独立 Python 应用程序的标准工具是 **`py2app`**。有关安装和使用 py2app 的更多信息，请访问 http://undefined.org/python/#py2app。

## 4.6 其他資源

MacPython 郵件清單對於 Python 使用者和開發者於 Mac 上是一個極佳的支援資源

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

其他好用的資源是 MacPython wiki:

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

CHAPTER 5

# 编辑器和集成开发环境

有很多支持 Python 编程语言的集成开发环境。大多数编辑器和集成开发环境支持语法高亮，调试工具和 **PEP 8** 检查。

请访问 Python Editors 和 Integrated Development Environments 以获取完整列表。

APPENDIX A

# 术语对照表

**>>>** 交互式终端中默认的 Python 提示符。往往会显示于能以交互方式在解释器里执行的样例代码之前。

**...** 可以是指：

- 交互式终端中输入特殊代码行时默认的 Python 提示符，包括：缩进的代码块，成对的分隔符之内（圆括号、方括号、花括号或三重引号），或是指定一个装饰器之后。
- `Ellipsis` 内置常量。

**2to3** 一个将 Python 2.x 代码转换为 Python 3.x 代码的工具，能够处理大部分通过解析源码并遍历解析树可检测到的不兼容问题。

2to3 包含在标准库中，模块名为 `lib2to3`；并提供一个独立入口点 `Tools/scripts/2to3`。参见 2to3-reference。

**abstract base class -- 抽象基类** 抽象基类简称 ABC，是对*duck-typing* 的补充，它提供了一种定义接口的新方式，相比之下其他技巧例如 `hasattr()` 显得过于笨拙或有微妙错误（例如使用 魔术方法）。ABC 引入了虚拟子类，这种类并非继承自其他类，但却仍能被 `isinstance()` 和 `issubclass()` 所认可；详见 `abc` 模块文档。Python 自带许多内置的 ABC 用于实现数据结构（在 `collections.abc` 模块中）、数字（在 `numbers` 模块中）、流（在 `io` 模块中）、导入查找器和加载器（在 `importlib.abc` 模块中）。你可以使用 `abc` 模块来创建自己的 ABC。

**annotation -- 标注** 关联到某个变量、类属性、函数形参或返回值的标签，被约定作为*type hint* 来使用。

局部变量的标注在运行时不可访问，但全局变量、类属性和函数的标注会分别存放模块、类和函数的 `__annotations__` 特殊属性中。

参见*variable annotation*、*function annotation*、**PEP 484** 和 **PEP 526**，对此功能均有介绍。

**argument -- 参数** 在调用函数时传给*function* （或*method* ）的值。参数分为两种：

- *关键字参数*: 在函数调用中前面带有标识符（例如 `name=`）或者作为包含在前面带有 `**` 的字典里的值传入。举例来说，`3` 和 `5` 在以下对 `complex()` 的调用中均属于关键字参数:

```
complex(real=3, imag=5)
complex(**{'real': 3, 'imag': 5})
```

- *位置参数*: 不属于关键字参数的参数。位置参数可出现于参数列表的开头以及/或者作为前面带有 * 的*iterable* 里的元素被传入。举例来说，3 和 5 在以下调用中均属于位置参数:

```
complex(3, 5)
complex(*(3, 5))
```

参数会被赋值给函数体中对应的局部变量。有关赋值规则参见 calls 一节。根据语法，任何表达式都可用来表示一个参数；最终算出的值会被赋给对应的局部变量。

另参见*parameter* 术语表条目，常见问题中 参数与形参的区别以及 **PEP 362**。

**asynchronous context manager -- 异步上下文管理器** 此种对象通过定义 `__aenter__()` 和 `__aexit__()` 方法来对 `async with` 语句中的环境进行控制。由 **PEP 492** 引入。

**asynchronous generator -- 异步生成器** 返回值为*asynchronous generator iterator* 的函数。它与使用 `async def` 定义的协程函数很相似，不同之处在于它包含 `yield` 表达式以产生一系列可在 `async for` 循环中使用的值。

此术语通常是指异步生成器函数，但在某些情况下则可能是指 异步生成器迭代器。如果需要清楚表达具体含义，请使用全称以避免歧义。

一个异步生成器函数可能包含 `await` 表达式或者 `async for` 以及 `async with` 语句。

**asynchronous generator iterator -- 异步生成器迭代器** *asynchronous generator* 函数所创建的对象。

此对象属于*asynchronous iterator*，当使用 `__anext__()` 方法调用时会返回一个可等待对象来执行异步生成器函数的代码直到下一个 `yield` 表达式。

每个 `yield` 会临时暂停处理，记住当前位置执行状态 (包括局部变量和挂起的 try 语句)。当该 异步生成器迭代器与其他 `__anext__()` 返回的可等待对象有效恢复时，它会从离开位置继续执行。参见 **PEP 492** 和 **PEP 525**。

**asynchronous iterable -- 异步可迭代对象** 可在 `async for` 语句中被使用的对象。必须通过它的 `__aiter__()` 方法返回一个*asynchronous iterator*。由 **PEP 492** 引入。

**asynchronous iterator -- 异步迭代器** 实现了 `__aiter__()` 和 `__anext__()` 方法的对象。`__anext__` 必须返回一个*awaitable* 对象。`async for` 会处理异步迭代器的 `__anext__()` 方法所返回的可等待对象，直到其引发一个 `StopAsyncIteration` 异常。由 **PEP 492** 引入。

**attribute -- 属性** 关联到一个对象的值，可以使用点号表达式通过其名称来引用。例如，如果一个对象 *o* 具有一个属性 *a*，就可以用 *o.a* 来引用它。

**awaitable -- 可等待对象** 能在 `await` 表达式中使用的对象。可以是*coroutine* 或是具有 `__await__()` 方法的对象。参见 **PEP 492**。

**BDFL** “终身仁慈独裁者”的英文缩写，即 Guido van Rossum，Python 的创造者。

**binary file -- 二进制文件** *file object* 能够读写字节类对象。二进制文件的例子包括以二进制模式（`'rb'`, `'wb'` or `'rb+'`）打开的文件、`sys.stdin.buffer`、`sys.stdout.buffer` 以及 `io.BytesIO` 和 `gzip.GzipFile` 的实例。

另请参见*text file* 了解能够读写 `str` 对象的文件对象。

**bytes-like object -- 字节类对象** 支持 bufferobjects 并且能导出 C-*contiguous* 缓冲的对象。这包括所有 `bytes`、`bytearray` 和 `array.array` 对象，以及许多普通 `memoryview` 对象。字节类对象可在多种二进制数据操作中使用；这些操作包括压缩、保存为二进制文件以及通过套接字发送等。

某些操作需要可变的二进制数据。这种对象在文档中常被称为 “可读写字节类对象”。可变缓冲对象的例子包括 `bytearray` 以及 `bytearray` 的 `memoryview`。其他操作要求二进制数据存放于不可变对象 (” 只读字节类对象”)；这种对象的例子包括 `bytes` 以及 `bytes` 对象的 `memoryview`。

**bytecode -- 字节码** Python 源代码会被编译为字节码，即 CPython 解释器中表示 Python 程序的内部代码。字节码还会缓存在 `.pyc` 文件中，这样第二次执行同一文件时速度更快（可以免去将源码重新编译为字

节码)。这种" 中间语言" 运行在根据字节码执行相应机器码的*virtual machine* 之上。请注意不同 Python 虚拟机上的字节码不一定通用，也不一定能在不同 Python 版本上兼容。

字节码指令列表可以在 dis 模块的文档中查看。

**callback -- 回调** 一个作为参数被传入以用以在未来的某个时刻被调用的子例程函数。

**class -- 类** 用来创建用户定义对象的模板。类定义通常包含对该类的实例进行操作的方法定义。

**class variable -- 类变量** 在类中定义的变量，并且仅限在类的层级上修改 (而不是在类的实例中修改)。

**coercion -- 强制类型转换** 在包含两个相同类型参数的操作中，一种类型的实例隐式地转换为另一种类型。例如，`int(3.15)` 是将原浮点数转换为整型数 `3`，但在 `3+4.5` 中，参数的类型不一致（一个是 int, 一个是 float)，两者必须转换为相同类型才能相加，否则将引发 `TypeError`。如果没有强制类型转换机制，程序员必须将所有可兼容参数归一化为相同类型，例如要写成 `float(3)+4.5` 而不是 `3+4.5`。

**complex number -- 复数** 对普通实数系统的扩展，其中所有数字都被表示为一个实部和一个虚部的和。虚数是虚数单位（`-1` 的平方根）的实倍数，通常在数学中写为 `i`，在工程学中写为 `j`。Python 内置了对复数的支持，采用工程学标记方式；虚部带有一个 `j` 后缀，例如 `3+1j`。如果需要 `math` 模块内对象的对应复数版本，请使用 `cmath`，复数的使用是一个比较高级的数学特性。如果你感觉没有必要，忽略它们也几乎不会有任何问题。

**context manager -- 上下文管理器** 在 `with` 语句中使用，通过定义 `__enter__()` 和 `__exit__()` 方法来控制环境状态的对象。参见 **PEP 343**。

**context variable -- 上下文变量** 一种根据其所属的上下文可以具有不同的值的变量。这类似于在线程局部存储中每个执行线程可以具有不同的变量值。不过，对于上下文变量来说，一个执行线程中可能会有多个上下文，而上下文变量的主要用途是对并发异步任务中变量进行追踪。参见 `contextvars`。

**contiguous -- 连续** 一个缓冲如果是 *C 连续*或 *Fortran 连续*就会被认为是连续的。零维缓冲是 C 和 Fortran 连续的。在一维数组中，所有条目必须在内存中彼此相邻地排列，采用从零开始的递增索引顺序。在多维 C-连续数组中，当按内存地址排列时用最后一个索引访问条目时速度最快。但是在 Fortran 连续数组中则是用第一个索引最快。

**coroutine -- 协程** 协程是子例程的更一般形式。子例程可以在某一点进入并在另一点退出。协程则可以在许多不同的点上进入、退出和恢复。它们可通过 `async def` 语句来实现。参见 **PEP 492**。

**coroutine function -- 协程函数** 返回一个*coroutine* 对象的函数。协程函数可通过 `async def` 语句来定义，并可能包含 `await`、`async for` 和 `async with` 关键字。这些特性是由 **PEP 492** 引入的。

**CPython** Python 编程语言的规范实现，在 python.org 上发布。"CPython" 一词用于在必要时将此实现与其他实现例如 Jython 或 IronPython 相区别。

**decorator -- 装饰器** 返回值为另一个函数的函数，通常使用 `@wrapper` 语法形式来进行函数变换。装饰器的常见例子包括 `classmethod()` 和 `staticmethod()`。

装饰器语法只是一种语法糖，以下两个函数定义在语义上完全等价:

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

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

同的样概念也适用于类，但通常较少这样使用。有关装饰器的详情可参见 函数定义和 类定义的文档。

**descriptor -- 描述器** 任何定义了 `__get__()`, `__set__()` 或 `__delete__()` 方法的对象。当一个类属性为描述器时，它的特殊绑定行为就会在属性查找时被触发。通常情况下，使用 *a.b* 来获取、设置或删除一个属性时会在 *a* 的类字典中查找名称为 *b* 的对象，但如果 *b* 是一个描述器，则会调用对应的描述器

方法。理解描述器的概念是更深层次理解 Python 的关键，因为这是许多重要特性的基础，包括函数、方法、属性、类方法、静态方法以及对超类的引用等等。

有关描述符的方法的详情可参看 descriptors。

**dictionary -- 字典** 一个关联数组，其中的任意键都映射到相应的值。键可以是任何具有 `__hash__()` 和 `__eq__()` 方法的对象。在 Perl 语言中称为 hash。

**dictionary comprehension -- 字典推导式** 处理一个可迭代对象中的所有或部分元素并返回结果字典的一种紧凑写法。`results = {n: n ** 2 for n in range(10)}` 将生成一个由键 `n` 到值 `n ** 2` 的映射构成的字典。参见 comprehensions。

**dictionary view -- 字典视图** 从 `dict.keys()`, `dict.values()` 和 `dict.items()` 返回的对象被称为字典视图。它们提供了字典条目的一个动态视图，这意味着当字典改变时，视图也会相应改变。要将字典视图强制转换为真正的列表，可使用 `list(dictview)`。参见 dict-views。

**docstring -- 文档字符串** 作为类、函数或模块之内的第一个表达式出现的字符串字面值。它在代码执行时会被忽略，但会被解释器识别并放入所在类、函数或模块的 `__doc__` 属性中。由于它可用于代码内省，因此是对象存放文档的规范位置。

**duck-typing -- 鸭子类型** 指一种编程风格，它并不依靠查找对象类型来确定其是否具有正确的接口，而是直接调用或使用其方法或属性（“看起来像鸭子，叫起来也像鸭子，那么肯定就是鸭子。”）由于强调接口而非特定类型，设计良好的代码可通过允许多态替代来提升灵活性。鸭子类型避免使用 `type()` 或 `isinstance()` 检测。(但要注意鸭子类型可以使用抽象基类 作为补充。) 而往往会采用 `hasattr()` 检测或是*EAFP* 编程。

**EAFP** “求原谅比求许可更容易”的英文缩写。这种 Python 常用代码编写风格会假定所需的键或属性存在，并在假定错误时捕获异常。这种简洁快速风格的特点就是大量运用 `try` 和 `except` 语句。于其相对的则是所谓*LBYL* 风格，常见于 C 等许多其他语言。

**expression -- 表达式** 可以求出某个值的语法单元。换句话说，一个表达式就是表达元素例如字面值、名称、属性访问、运算符或函数调用的汇总，它们最终都会返回一个值。与许多其他语言不同，并非所有语言构件都是表达式。还存在不能被用作表达式的*statement*，例如 `while`。赋值也是属于语句而非表达式。

**extension module -- 扩展模块** 以 C 或 C++ 编写的模块，使用 Python 的 C API 来与语言核心以及用户代码进行交互。

**f-string -- f-字符串** 带有 `'f'` 或 `'F'` 前缀的字符串字面值通常被称为“f-字符串”即 格式化字符串字面值的简写。参见 **PEP 498**。

**file object -- 文件对象** 对外提供面向文件 API 以使用下层资源的对象（带有 `read()` 或 `write()` 这样的方法）。根据其创建方式的不同，文件对象可以处理对真实磁盘文件，对其他类型存储，或是对通讯设备的访问（例如标准输入/输出、内存缓冲区、套接字、管道等等）。文件对象也被称为 *文件类对象* 或 *流*。

实际上共有三种类别的文件对象: 原始二进制文件, 缓冲二进制文件 以及文本文件。它们的接口定义均在 `io` 模块中。创建文件对象的规范方式是使用 `open()` 函数。

**file-like object -- 文件类对象** *file object* 的同义词。

**finder -- 查找器** 一种会尝试查找被导入模块的*loader* 的对象。

从 Python 3.3 起存在两种类型的查找器: 元路径查找器 配合 `sys.meta_path` 使用，以及*path entry finders* 配合 `sys.path_hooks` 使用。

更多详情可参见 **PEP 302**, **PEP 420** 和 **PEP 451**。

**floor division -- 向下取整除法** 向下舍入到最接近的整数的数学除法。向下取整除法的运算符是 `//`。例如，表达式 `11 // 4` 的计算结果是 `2`，而与之相反的是浮点数的真正除法返回 `2.75`。注意 `(-11) // 4` 会返回 `-3` 因为这是 `-2.75` 向下舍入得到的结果。见 **PEP 238**。

**function -- 函数** 可以向调用者返回某个值的一组语句。还可以向其传入零个或多个参数 并在函数体执行中被使用。另见*parameter*, *method* 和 function 等节。

**function annotation -- 函数标注** 即针对函数形参或返回值的*annotation* 。

函数标注通常用于类型提示：例如以下函数预期接受两个 `int` 参数并预期返回一个 `int` 值:

```
def sum_two_numbers(a: int, b: int) -> int:
   return a + b
```

函数标注语法的详解见 function 一节。

请参看*variable annotation* 和 **PEP 484** 对此功能的描述。

**__future__** 一种伪模块，可被程序员用来启用与当前解释器不兼容的新语言特性。

通过导入 `__future__` 模块并对其中的变量求值，你可以查看新特性何时首次加入语言以及何时成为默认:

```
>>> import __future__
>>> __future__.division
_Feature((2, 2, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0), 8192)
```

**garbage collection -- 垃圾回收** 释放不再被使用的内存空间的过程。Python 是通过引用计数和一个能够检测和打破循环引用的循环垃圾回收器来执行垃圾回收的。可以使用 `gc` 模块来控制垃圾回收器。

**generator -- 生成器** 返回一个*generator iterator* 的函数。它看起来很像普通函数，不同点在于其包含 `yield` 表达式以便产生一系列值供给 for-循环使用或是通过 `next()` 函数逐一获取。

通常是指生成器函数，但在某些情况下也可能是指 生成器迭代器。如果需要清楚表达具体含义，请使用全称以避免歧义。

**generator iterator -- 生成器迭代器** *generator* 函数所创建的对象。

每个 `yield` 会临时暂停处理，记住当前位置执行状态（包括局部变量和挂起的 try 语句）。当该 生成器迭代器恢复时，它会从离开位置继续执行（这与每次调用都从新开始的普通函数差别很大）。

**generator expression -- 生成器表达式** 返回一个迭代器的表达式。它看起来很像普通表达式后面带有定义了一个循环变量、范围的 `for` 子句，以及一个可选的 `if` 子句。以下复合表达式会为外层函数生成一系列值:

```
>>> sum(i*i for i in range(10))         # sum of squares 0, 1, 4, ... 81
285
```

**generic function -- 泛型函数** 为不同的类型实现相同操作的多个函数所组成的函数。在调用时会由调度算法来确定应该使用哪个实现。

另请参见*single dispatch* 术语表条目、`functools.singledispatch()` 装饰器以及 **PEP 443**。

**GIL** 参见*global interpreter lock*。

**global interpreter lock -- 全局解释器锁** *CPython* 解释器所采用的一种机制，它确保同一时刻只有一个线程在执行 Python *bytecode*。此机制通过设置对象模型（包括 `dict` 等重要内置类型）针对并发访问的隐式安全简化了 CPython 实现。给整个解释器加锁使得解释器多线程运行更方便，其代价则是牺牲了在多处理器上的并行性。

不过，某些标准库或第三方库的扩展模块被设计为在执行计算密集型任务如压缩或哈希时释放 GIL。此外，在执行 I/O 操作时也总是会释放 GIL。

创建一个（以更精细粒度来锁定共享数据的）“自由线程”解释器的努力从未获得成功，因为这会牺牲在普通单处理器情况下的性能。据信克服这种性能问题的措施将导致实现变得更复杂，从而更难以维护。

**hash-based pyc -- 基于哈希的 pyc** 使用对应源文件的哈希值而非最后修改时间来确定其有效性的字节码缓存文件。参见 pyc-invalidation。

**hashable -- 可哈希** 一个对象的哈希值如果在其生命周期内绝不改变，就被称为 可哈希（它需要具有 __hash__() 方法），并可以同其他对象进行比较（它需要具有 __eq__() 方法）。可哈希对象必须具有相同的哈希值比较结果才会相同。

可哈希性使得对象能够作为字典键或集合成员使用，因为这些数据结构要在内部使用哈希值。

大多数 Python 中的不可变内置对象都是可哈希的；可变容器（例如列表或字典）都不可哈希；不可变容器（例如元组和 frozenset）仅当它们的元素均为可哈希时才是可哈希的。用户定义类的实例对象默认是可哈希的。它们在比较时一定不相同（除非是与自己比较），它们的哈希值的生成是基于它们的 id()。

**IDLE** Python 的 IDE，“集成开发与学习环境”的英文缩写。是 Python 标准发行版附带的基本编辑器和解释器环境。

**immutable -- 不可变** 具有固定值的对象。不可变对象包括数字、字符串和元组。这样的对象不能被改变。如果必须存储一个不同的值，则必须创建新的对象。它们在需要常量哈希值的地方起着重要作用，例如作为字典中的键。

**import path -- 导入路径** 由多个位置（或路径条目）组成的列表，会被模块的*path based finder* 用来查找导入目标。在导入时，此位置列表通常来自 sys.path，但对次级包来说也可能来自上级包的 __path__ 属性。

**importing -- 导入** 令一个模块中的 Python 代码能为另一个模块中的 Python 代码所使用的过程。

**importer -- 导入器** 查找并加载模块的对象；此对象既属于*finder* 又属于*loader*。

**interactive -- 交互** Python 带有一个交互式解释器，即你可以在解释器提示符后输入语句和表达式，立即执行并查看其结果。只需不带参数地启动 python 命令（也可以在你的计算机开始菜单中选择相应菜单项）。在测试新想法或检验模块和包的时候用这种方式会非常方便（请记得使用 help(x)）。

**interpreted -- 解释型** Python 一是种解释型语言，与之相对的是编译型语言，虽然两者的区别由于字节码编译器的存在而会有所模糊。这意味着源文件可以直接运行而不必显式地创建可执行文件再运行。解释型语言通常具有比编译型语言更短的开发/调试周期，但是其程序往往运行得更慢。参见*interactive*。

**interpreter shutdown -- 解释器关闭** 当被要求关闭时，Python 解释器将进入一个特殊运行阶段并逐步释放所有已分配资源，例如模块和各种关键内部结构等。它还会多次调用垃圾回收器。这会触发用户定义析构器或弱引用回调中的代码执行。在关闭阶段执行的代码可能会遇到各种异常，因为其所依赖的资源已不再有效（常见的例子有库模块或警告机制等）。

解释器需要关闭的主要原因有 __main__ 模块或所运行的脚本已完成执行。

**iterable -- 可迭代对象** 能够逐一返回其成员项的对象。可迭代对象的例子包括所有序列类型 (例如 list, str 和 tuple) 以及某些非序列类型例如 dict, 文件对象 以及定义了 __iter__() 方法或是实现了序列 语义的 __getitem__() 方法的任意自定义类对象。

可迭代对象被可用于 for 循环以及许多其他需要一个序列的地方（zip()、map() ...）。当一个可迭代对象作为参数传给内置函数 iter() 时，它会返回该对象的迭代器。这种迭代器适用于对值集合的一次性遍历。在使用可迭代对象时，你通常不需要调用 iter() 或者自己处理迭代器对象。for 语句会为你自动处理那些操作，创建一个临时的未命名变量用来在循环期间保存迭代器。参见*iterator*、*sequence* 以及*generator*。

**iterator -- 迭代器** 用来表示一连串数据流的对象。重复调用迭代器的 __next__() 方法（或将其传给内置函数 next()）将逐个返回流中的项。当没有数据可用时则将引发 StopIteration 异常。到这时迭代器对象中的数据项已耗尽，继续调用其 __next__() 方法只会再次引发 StopIteration 异常。迭代器必须具有 __iter__() 方法用来返回该迭代器对象自身，因此迭代器必定也是可迭代对象，可被用于其他可迭代对象适用的大部分场合。一个显著的例外是那些会多次重复访问迭代项的代码。容器对象（例如 list）在你每次向其传入 iter() 函数或是在 for 循环中使用它时都会产生一个新的迭代器。如果在此情况下你尝试用迭代器则会返回在之前迭代过程中被耗尽的同一迭代器对象，使其看起来就像是一个空容器。

更多信息可查看 typeiter。

**key function -- 键函数** 键函数或称整理函数，是能够返回用于排序或排位的值的可调用对象。例如，`locale.strxfrm()` 可用于生成一个符合特定区域排序约定的排序键。

Python 中有许多工具都允许用键函数来控制元素的排位或分组方式。其中包括 `min()`，`max()`，`sorted()`，`list.sort()`，`heapq.merge()`，`heapq.nsmallest()`，`heapq.nlargest()` 以及 `itertools.groupby()`。

要创建一个键函数有多种方式。例如，`str.lower()` 方法可以用作忽略大小写排序的键函数。另外，键函数也可通过 `lambda` 表达式来创建，例如 `lambda r: (r[0], r[2])`。还有 `operator` 模块提供了三个键函数构造器：`attrgetter()`、`itemgetter()` 和 `methodcaller()`。请查看 如何排序 一节以获取创建和使用键函数的示例。

**keyword argument -- 关键字参数** 参见*argument*。

**lambda** 由一个单独*expression* 构成的匿名内联函数，表达式会在调用时被求值。创建 lambda 函数的句法为 `lambda [parameters]: expression`

**LBYL** “先查看后跳跃”的英文缩写。这种代码编写风格会在进行调用或查找之前显式地检查前提条件。此风格与*EAFP* 方式恰成对比，其特点是大量使用 `if` 语句。

在多线程环境中，LBYL 方式会导致“查看”和“跳跃”之间发生条件竞争风险。例如，以下代码 `if key in mapping: return mapping[key]` 可能由于在检查操作之后其他线程从 *mapping* 中移除了 *key* 而出错。这种问题可通过加锁或使用 EAFP 方式来解决。

**list -- 列表** Python 内置的一种*sequence*。虽然名为列表，但更类似于其他语言中的数组而非链接列表，因为访问元素的时间复杂度为 O(1)。

**list comprehension -- 列表推导式** 处理一个序列中的所有或部分元素并返回结果列表的一种紧凑写法。`result = ['{:#04x}'.format(x) for x in range(256) if x % 2 == 0]` 将生成一个 0 到 255 范围内的十六进制偶数对应字符串（0x..）的列表。其中 `if` 子句是可选的，如果省略则 `range(256)` 中的所有元素都会被处理。

**loader -- 加载器** 负责加载模块的对象。它必须定义名为 `load_module()` 的方法。加载器通常由一个*finder* 返回。详情参见 **PEP 302**，对于*abstract base class* 可参见 `importlib.abc.Loader`。

**magic method -- 魔术方法** *special method* 的非正式同义词。

**mapping -- 映射** 一种支持任意键查找并实现了 `Mapping` 或 `MutableMapping` 抽象基类中所规定方法的容器对象。此类对象的例子包括 `dict`，`collections.defaultdict`，`collections.OrderedDict` 以及 `collections.Counter`。

**meta path finder -- 元路径查找器** `sys.meta_path` 的搜索所返回的*finder*。元路径查找器与*path entry finders* 存在关联但并不相同。

请查看 `importlib.abc.MetaPathFinder` 了解元路径查找器所实现的方法。

**metaclass -- 元类** 一种用于创建类的类。类定义包含类名、类字典和基类列表。元类负责接受上述三个参数并创建相应的类。大部分面向对象的编程语言都会提供一个默认实现。Python 的特别之处在于可以创建自定义元类。大部分用户永远不需要这个工具，但当需要出现时，元类可提供强大而优雅的解决方案。它们已被用于记录属性访问日志、添加线程安全性、跟踪对象创建、实现单例，以及其他许多任务。

更多详情参见 metaclasses。

**method -- 方法** 在类内部定义的函数。如果作为该类的实例的一个属性来调用，方法将会获取实例对象作为其第一个*argument* (通常命名为 `self`)。参见*function* 和*nested scope*。

**method resolution order -- 方法解析顺序** 方法解析顺序就是在查找成员时搜索全部基类所用的先后顺序。请查看 Python 2.3 方法解析顺序 了解自 2.3 版起 Python 解析器所用相关算法的详情。

**module -- 模块** 此对象是 Python 代码的一种组织单位。各模块具有独立的命名空间，可包含任意 Python 对象。模块可通过*importing* 操作被加载到 Python 中。

另见*package*。

**module spec -- 模块规格** 一个命名空间，其中包含用于加载模块的相关导入信息。是 `importlib.machinery.ModuleSpec` 的实例。

**MRO** 参见*method resolution order*。

**mutable -- 可变** 可变对象可以在其 `id()` 保持固定的情况下改变其取值。另请参见*immutable*。

**named tuple -- 具名元组** 术语“具名元组”可用于任何继承自元组，并且其中的可索引元素还能使用名称属性来访问的类型或类。这样的类型或类还可能拥有其他特性。

有些内置类型属于具名元组，包括 `time.localtime()` 和 `os.stat()` 的返回值。另一个例子是 `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
```

有些具名元组是内置类型(例如上面的例子)。此外，具名元组还可通过常规类定义从 `tuple` 继承并定义名称字段的方式来创建。这样的类可以手工编写，或者使用工厂函数 `collections.namedtuple()` 创建。后一种方式还会添加一些手工编写或内置具名元组所没有的额外方法。

**namespace -- 命名空间** 命名空间是存放变量的场所。命名空间有局部、全局和内置的，还有对象中的嵌套命名空间（在方法之内）。命名空间通过防止命名冲突来支持模块化。例如，函数 `builtins.open` 与 `os.open()` 可通过各自的命名空间来区分。命名空间还通过明确哪个模块实现那个函数来帮助提高可读性和可维护性。例如，`random.seed()` 或 `itertools.islice()` 这种写法明确了这些函数是由 `random` 与 `itertools` 模块分别实现的。

**namespace package -- 命名空间包** **PEP 420** 所引入的一种仅被用作子包的容器的*package*，命名空间包可以没有实体表示物，其描述方式与*regular package* 不同，因为它们没有 `__init__.py` 文件。

另可参见*module*。

**nested scope -- 嵌套作用域** 在一个定义范围内引用变量的能力。例如，在另一函数之内定义的函数可以引用前者的变量。请注意嵌套作用域默认只对引用有效而对赋值无效。局部变量的读写都受限于最内层作用域。类似的，全局变量的读写则作用于全局命名空间。通过 `nonlocal` 关键字可允许写入外层作用域。

**new-style class -- 新式类** 对于目前已被应于所有类对象的类形式的旧称谓。在早先的 Python 版本中，只有新式类能够使用 Python 新增的更灵活特性，例如 `__slots__`、描述符、特征属性、`__getattribute__()`、类方法和静态方法等。

**object -- 对象** 任何具有状态（属性或值）以及预定义行为（方法）的数据。object 也是任何*new-style class* 的最顶层基类名。

**package -- 包** 一种可包含子模块或递归地包含子包的 Python *module*。从技术上说，包是带有 `__path__` 属性的 Python 模块。

另参见*regular package* 和*namespace package*。

**parameter -- 形参** *function*（或方法）定义中的命名实体，它指定函数可以接受的一个*argument*（或在某些情况下，多个实参）。有五种形参：

- *positional-or-keyword*：位置或关键字，指定一个可以作为位置参数 传入也可以作为关键字参数 传入的实参。这是默认的形参类型，例如下面的 *foo* 和 *bar*:

```
def func(foo, bar=None): ...
```

- *positional-only*：仅限位置，指定一个只能通过位置传入的参数。仅限位置形参可通过在函数定义的形参列表中它们之后包含一个 `/` 字符来定义，例如下面的 *posonly1* 和 *posonly2*:

```
def func(posonly1, posonly2, /, positional_or_keyword): ...
```

- *keyword-only*：仅限关键字，指定一个只能通过关键字传入的参数。仅限关键字形参可通过在函数定义的形参列表中包含单个可变位置形参或者在多个可变位置形参之前放一个 `*` 来定义，例如下面的 *kw_only1* 和 *kw_only2*：

```
def func(arg, *, kw_only1, kw_only2): ...
```

- *var-positional*：可变位置，指定可以提供由一个任意数量的位置参数构成的序列（附加在其他形参已接受的位置参数之后）。这种形参可通过在形参名称前加缀 `*` 来定义，例如下面的 *args*：

```
def func(*args, **kwargs): ...
```

- *var-keyword*：可变关键字，指定可以提供任意数量的关键字参数（附加在其他形参已接受的关键字参数之后）。这种形参可通过在形参名称前加缀 `**` 来定义，例如上面的 *kwargs*。

形参可以同时指定可选和必选参数，也可以为某些可选参数指定默认值。

另参见*argument* 术语表条目、参数与形参的区别中的常见问题、`inspect.Parameter` 类、function 一节以及 **PEP 362**。

**path entry -- 路径入口** *import path* 中的一个单独位置，会被*path based finder* 用来查找要导入的模块。

**path entry finder -- 路径入口查找器** 任一可调用对象使用 `sys.path_hooks`（即*path entry hook*）返回的*finder*，此种对象能通过*path entry* 来定位模块。

请参看 `importlib.abc.PathEntryFinder` 以了解路径入口查找器所实现的各个方法。

**path entry hook -- 路径入口钩子** 一种可调用对象，在知道如何查找特定*path entry* 中的模块的情况下能够使用 `sys.path_hook` 列表返回一个*path entry finder*。

**path based finder -- 基于路径的查找器** 默认的一种元路径查找器，可在一个*import path* 中查找模块。

**path-like object -- 路径类对象** 代表一个文件系统路径的对象。类路径对象可以是一个表示路径的 `str` 或者 `bytes` 对象，还可以是一个实现了 `os.PathLike` 协议的对象。一个支持 `os.PathLike` 协议的对象可通过调用 `os.fspath()` 函数转换为 `str` 或者 `bytes` 类型的文件系统路径；`os.fsdecode()` 和 `os.fsencode()` 可被分别用来确保获得 `str` 或 `bytes` 类型的结果。此对象是由 **PEP 519** 引入的。

**PEP** “Python 增强提议”的英文缩写。一个 PEP 就是一份设计文档，用来向 Python 社区提供信息，或描述一个 Python 的新增特性及其进度或环境。PEP 应当提供精确的技术规格和所提议特性的原理说明。

PEP 应被作为提出主要新特性建议、收集社区对特定问题反馈以及为必须加入 Python 的设计决策编写文档的首选机制。PEP 的作者有责任在社区内部建立共识，并应将不同意见也记入文档。

参见 **PEP 1**。

**portion -- 部分** 构成一个命名空间包的单个目录内文件集合（也可能存放于一个 zip 文件内），具体定义见 **PEP 420**。

**positional argument -- 位置参数** 参见*argument*。

**provisional API -- 暂定 API** 暂定 API 是指被有意排除在标准库的向后兼容性保证之外的应用编程接口。虽然此类接口通常不会再有重大改变，但只要其被标记为暂定，就可能在核心开发者确定有必要的情况下进行向后不兼容的更改（甚至包括移除该接口）。此种更改并不会随意进行 -- 仅在 API 被加入之前未考虑到的严重基础性缺陷被发现时才可能会这样做。

即便是对暂定 API 来说，向后不兼容的更改也会被视为“最后的解决方案”——任何问题被确认时都会尽可能先尝试找到一种向后兼容的解决方案。

这种处理过程允许标准库持续不断地演进，不至于被有问题的长期性设计缺陷所困。详情见 **PEP 411**。

**provisional package -- 暂定包** 参见*provisional API*。

**Python 3000** Python 3.x 发布路线的昵称（这个名字在版本 3 的发布还遥遥无期的时候就已出现了）。有时也被缩写为“Py3k”。

**Pythonic** 指一个思路或一段代码紧密遵循了 Python 语言最常用的风格和理念，而不是使用其他语言中通用的概念来实现代码。例如，Python 的常用风格是使用 `for` 语句循环来遍历一个可迭代对象中的所有元素。许多其他语言没有这样的结构，因此不熟悉 Python 的人有时会选择使用一个数字计数器:

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

而相应的更简洁更 Pythonic 的方法是这样的:

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

**qualified name -- 限定名称** 一个以点号分隔的名称，显示从模块的全局作用域到该模块中定义的某个类、函数或方法的“路径”，相关定义见 **PEP 3155**。对于最高层级的函数和类，限定名称与对象名称一致:

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

当被用于引用模块时，完整限定名称意为标示该模块的以点号分隔的整个路径，其中包含其所有的父包，例如 `email.mime.text`:

```
>>> import email.mime.text
>>> email.mime.text.__name__
'email.mime.text'
```

**reference count -- 引用计数** 对特定对象的引用的数量。当一个对象的引用计数降为零时，所分配资源将被释放。引用计数对 Python 代码来说通常是不可见的，但它是*CPython* 实现的一个关键元素。`sys` 模块定义了一个 `getrefcount()` 函数，程序员可调用它来返回特定对象的引用计数。

**regular package -- 常规包** 传统型的*package*，例如包含有一个 `__init__.py` 文件的目录。

另参见*namespace package*。

**__slots__** 一种写在类内部的声明，通过预先声明实例属性等对象并移除实例字典来节省内存。虽然这种技巧很流行，但想要用好却并不容易，最好是只保留在少数情况下采用，例如极耗内存的应用程序，并且其中包含大量实例。

**sequence -- 序列** 一种*iterable*，它支持通过 `__getitem__()` 特殊方法来使用整数索引进行高效的元素访问，并定义了一个返回序列长度的 `__len__()` 方法。内置的序列类型有 `list`、`str`、`tuple` 和 `bytes`。注意虽然 `dict` 也支持 `__getitem__()` 和 `__len__()`，但它被认为属于映射而非序列，因为它查找时使用任意的*immutable* 键而非整数。

`collections.abc.Sequence` 抽象基类定义了一个更丰富的接口，它在 `__getitem__()` 和 `__len__()` 之外又添加了 `count()`, `index()`, `__contains__()` 和 `__reversed__()`。实现此扩展接口的类型可以使用 `register()` 来显式地注册。

**set comprehension -- 集合推导式** 处理一个可迭代对象中的所有或部分元素并返回结果集合的一种紧凑写法。`results = {c for c in 'abracadabra' if c not in 'abc'}` 将生成字符串集合 `{'r',`

'd'}。参见 comprehensions。

**single dispatch -- 单分派** 一种*generic function* 分派形式，其实现是基于单个参数的类型来选择的。

**slice -- 切片** 通常只包含了特定*sequence* 的一部分的对象。切片是通过使用下标标记来创建的，在 [] 中给出几个以冒号分隔的数字，例如 variable_name[1:3:5]。方括号（下标）标记在内部使用 slice 对象。

**special method -- 特殊方法** 一种由 Python 隐式调用的方法，用来对某个类型执行特定操作例如相加等等。这种方法的名称的首尾都为双下划线。特殊方法的文档参见 specialnames。

**statement -- 语句** 语句是程序段（一个代码“块”）的组成单位。一条语句可以是一个*expression* 或某个带有关键字的结构，例如 if、while 或 for。

**text encoding -- 文本编码** 用于将 Unicode 字符串编码为字节串的编码器。

**text file -- 文本文件** 一种能够读写 str 对象的*file object*。通常一个文本文件实际是访问一个面向字节的数据流并自动处理*text encoding*。文本文件的例子包括以文本模式（'r' 或 'w'）打开的文件、sys.stdin、sys.stdout 以及 io.StringIO 的实例。

另请参看*binary file* 了解能够读写字节类对象 的文件对象。

**triple-quoted string -- 三引号字符串** 首尾各带三个连续双引号（"）或者单引号（'）的字符串。它们在功能上与首尾各用一个引号标注的字符串没有什么不同，但是有多种用处。它们允许你在字符串内包含未经转义的单引号和双引号，并且可以跨越多行而无需使用连接符，在编写文档字符串时特别好用。

**type -- 类型** 类型决定一个 Python 对象属于什么种类；每个对象都具有一种类型。要知道对象的类型，可以访问它的 __class__ 属性，或是通过 type(obj) 来获取。

**type alias -- 类型别名** 一个类型的同义词，创建方式是把类型赋值给特定的标识符。

类型别名的作用是简化类型提示。例如:

```
from typing import List, Tuple

def remove_gray_shades(
        colors: List[Tuple[int, int, int]]) -> List[Tuple[int, int, int]]:
    pass
```

可以这样提高可读性:

```
from typing import List, Tuple

Color = Tuple[int, int, int]

def remove_gray_shades(colors: List[Color]) -> List[Color]:
    pass
```

参见 typing 和 **PEP 484**，其中有对此功能的详细描述。

**type hint -- 类型提示** *annotation* 为变量、类属性、函数的形参或返回值指定预期的类型。

类型提示属于可选项，Python 不要求提供，但其可对静态类型分析工具起作用，并可协助 IDE 实现代码补全与重构。

全局变量、类属性和函数的类型提示可以使用 typing.get_type_hints() 来访问，但局部变量则不可以。

参见 typing 和 **PEP 484**，其中有对此功能的详细描述。

**universal newlines -- 通用换行** 一种解读文本流的方式，将以下所有符号都识别为行结束标志：Unix 的行结束约定 '\n'、Windows 的约定 '\r\n' 以及旧版 Macintosh 的约定 '\r'。参见 **PEP 278** 和 **PEP 3116** 和 bytes.splitlines() 了解更多用法说明。

**variable annotation -- 变量标注** 对变量或类属性的*annotation*。

在标注变量或类属性时，还可选择为其赋值:

```
class C:
    field: 'annotation'
```

变量标注通常被用作类型提示：例如以下变量预期接受 int 类型的值:

```
count: int = 0
```

变量标注语法的详细解释见 annassign 一节。

请参看*function annotation*、**PEP 484** 和 **PEP 526**，其中对此功能有详细描述。

**virtual environment -- 虚拟环境** 一种采用协作式隔离的运行时环境，允许 Python 用户和应用程序在安装和升级 Python 分发包时不会干扰到同一系统上运行的其他 Python 应用程序的行为。

另参见 venv。

**virtual machine -- 虚拟机** 一台完全通过软件定义的计算机。Python 虚拟机可执行字节码编译器所生成的*bytecode*。

**Zen of Python -- Python 之禅** 列出 Python 设计的原则与哲学，有助于理解与使用这种语言。查看其具体内容可在交互模式提示符中输入”import this”。

APPENDIX B

# 關於這些🄵明文件

這些🄵明文件是透過 Sphinx （一個專🄵 Python 🄵明文件所撰寫的文件處理器）將使用 reStructuredText 撰寫的原始檔轉🄵而成。

如同 Python 自身，透過自願者的努力下🄵出文件與封裝後自動化執行工具。若想要回報臭蟲，請見 reporting-bugs 頁面，🄵含相關資訊。我們永遠歡迎新的自願者加入！

致謝：

- Fred L. Drake, Jr.，原始 Python 文件工具集的創造者以及一大部份🄵容的作者。
- 創造 reStructuredText 和 Docutils 工具組的 Docutils 專案；
- Fredrik Lundh 先生，Sphinx 從他的 Alternative Python Reference 計劃中獲得許多的好主意。

## B.1 Python 文件的貢獻者們

許多人都曾🄵 Python 這門語言、Python 標准函式庫和 Python 🄵明文件貢獻過。Python 所發🄵的原始碼中含有部份貢獻者的清單，請見 Misc/ACKS 。

正因🄵 Python 社群的撰寫與貢獻才有這份這🄵棒的🄵明文件 -- 感謝所有貢獻過的人們！

APPENDIX C

# 歷史與授權

## C.1 该软件的历史

Python 由荷兰数学和计算机科学研究学会（CWI，见 https://www.cwi.nl/ ）的 Guido van Rossum 于 1990 年代初设计，作为一门叫做 ABC 的语言的替代品。尽管 Python 包含了许多来自其他人的贡献，Guido 仍是其主要作者。

1995 年，Guido 在弗吉尼亚州的国家创新研究公司（CNRI，见 https://www.cnri.reston.va.us/ ）继续他在 Python 上的工作，并在那里发布了该软件的多个版本。

2000 年五月，Guido 和 Python 核心开发团队转到 BeOpen.com 并组建了 BeOpen PythonLabs 团队。同年十月，PythonLabs 团队转到 Digital Creations (现为 Zope Corporation；见 https://www.zope.org/)。2001 年，Python 软件基金会 (PSF，见 https://www.python.org/psf/) 成立，这是一个专为拥有 Python 相关知识产权而创建的非营利组织。Zope Corporation 现在是 PSF 的赞助成员。

所有的 Python 版本都是开源的（有关开源的定义参阅 https://opensource.org/ ）。历史上，绝大多数 Python 版本是 GPL 兼容的；下表总结了各个版本情况。

| 发布版本 | 源自 | 年份 | 所有者 | GPL 兼容？ |
|---|---|---|---|---|
| 0.9.0 至 1.2 | n/a | 1991-1995 | CWI | 是 |
| 1.3 至 1.5.2 | 1.2 | 1995-1999 | CNRI | 是 |
| 1.6 | 1.5.2 | 2000 | CNRI | 否 |
| 2.0 | 1.6 | 2000 | BeOpen.com | 否 |
| 1.6.1 | 1.6 | 2001 | CNRI | 否 |
| 2.1 | 2.0+1.6.1 | 2001 | PSF | 否 |
| 2.0.1 | 2.0+1.6.1 | 2001 | PSF | 是 |
| 2.1.1 | 2.1+2.0.1 | 2001 | PSF | 是 |
| 2.1.2 | 2.1.1 | 2002 | PSF | 是 |
| 2.1.3 | 2.1.2 | 2002 | PSF | 是 |
| 2.2 及更高 | 2.1.1 | 2001 至今 | PSF | 是 |

**備🄴:** GPL 兼容并不意味着 Python 在 GPL 下发布。与 GPL 不同，所有 Python 许可证都允许您分发修改后

的版本，而无需开源所做的更改。GPL 兼容的许可证使得 Python 可以与其它在 GPL 下发布的软件结合使用；但其它的许可证则不行。

感谢众多在 Guido 指导下工作的外部志愿者，使得这些发布成为可能。

## C.2 获取或以其他方式使用 Python 的条款和条件

Python 软件和文档的使用许可是依据*PSF 许可证协议*。

从 Python 3.8.6 开始，文档中的示例、操作指导和其他代码采用的是 PSF 许可协议和*零条款 BSD 许可* 的双重使用许可。

某些包含在 Python 中的软件是基于不同的许可。这些许可会与相应许可之下的代码一同列出。有关这些许可的不完整列表请参阅*收录软件的许可与鸣谢*。

### C.2.1 用于 PYTHON 3.8.8rc1 的 PSF 许可协议

```
1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"),␣
↪and
   the Individual or Organization ("Licensee") accessing and otherwise using␣
↪Python
   3.8.8rc1 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 3.8.8rc1 alone or in any derivative
   version, provided, however, that PSF's License Agreement and PSF's notice␣
↪of
   copyright, i.e., "Copyright © 2001-2021 Python Software Foundation; All␣
↪Rights
   Reserved" are retained in Python 3.8.8rc1 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 3.8.8rc1 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
   3.8.8rc1.

4. PSF is making Python 3.8.8rc1 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 3.8.8rc1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
```

```
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.8.8rc1
   FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT␣
↪OF
   MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.8.8rc1, 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 3.8.8rc1, Licensee agrees
   to be bound by the terms and conditions of this License Agreement.
```

### C.2.2 用于 PYTHON 2.0 的 BEOPEN.COM 许可协议

BEOPEN PYTHON 开源许可协议第 1 版

```
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.
```

(下页继续)

(繼續上一頁)

```
6. This License Agreement shall be governed by and interpreted in all respects
   by the law of the State of California, excluding conflict of law provisions.
   Nothing in this License Agreement shall be deemed to create any relationship of
   agency, partnership, or joint venture between BeOpen and Licensee.  This License
   Agreement does not grant permission to use BeOpen trademarks or trade names in a
   trademark sense to endorse or promote products or services of Licensee, or any
   third party.  As an exception, the "BeOpen Python" logos available at
   http://www.pythonlabs.com/logos.html may be used according to the permissions
   granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be
   bound by the terms and conditions of this License Agreement.
```

### C.2.3 用于 PYTHON 1.6.1 的 CNRI 许可协议

```
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.
```

(下页继续)

(繼續上一頁)

```
7. This License Agreement shall be governed by the federal intellectual property
   law of the United States, including without limitation the federal copyright
   law, and, to the extent such U.S. federal law does not apply, by the law of the
   Commonwealth of Virginia, excluding Virginia's conflict of law provisions.
   Notwithstanding the foregoing, with regard to derivative works based on Python
   1.6.1 that incorporate non-separable material that was previously distributed
   under the GNU General Public License (GPL), the law of the Commonwealth of
   Virginia shall govern this License Agreement only as to issues arising under or
   with respect to Paragraphs 4, 5, and 7 of this License Agreement.  Nothing in
   this License Agreement shall be deemed to create any relationship of agency,
   partnership, or joint venture between CNRI and Licensee.  This License Agreement
   does not grant permission to use CNRI trademarks or trade name in a trademark
   sense to endorse or promote products or services of Licensee, or any third
   party.

8. By clicking on the "ACCEPT" button where indicated, or by copying, installing
   or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and
   conditions of this License Agreement.
```

### C.2.4 用于 PYTHON 0.9.0 至 1.2 的 CWI 许可协议

```
Copyright © 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The
Netherlands.  All rights reserved.

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted, provided that
the above copyright notice appear in all copies and that both that copyright
notice and this permission notice appear in supporting documentation, and that
the name of Stichting Mathematisch Centrum or CWI not be used in advertising or
publicity pertaining to distribution of the software without specific, written
prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO
EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT
OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE,
DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS
SOFTWARE.
```

### C.2.5 ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON 3.8.8rc1 DOCUMENTATION

```
Permission to use, copy, modify, and/or distribute this software for any
purpose with or without fee is hereby granted.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH
REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT,
INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM
LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
```

(下页继续)

(繼續上一頁)

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

## C.3 收录软件的许可与鸣谢

本节是 Python 发行版中收录的第三方软件的许可和致谢清单，该清单是不完整且不断增长的。

### C.3.1 Mersenne Twister

_random 模块包含基于 http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html 下载的代码。以下是原始代码的完整注释（声明）：

```
A C-program for MT19937, with initialization improved 2002/1/26.
Coded by Takuji Nishimura and Makoto Matsumoto.

Before using, initialize the state by using init_genrand(seed)
or init_by_array(init_key, key_length).

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

Redistribution and use in source and binary forms, with or without
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 套接字

socket 模块使用 getaddrinfo() 和 getnameinfo() 函数，这些函数源代码在 WIDE 项目（http://www.wide.ad.jp/）的单独源文件中。

```
Copyright (C) 1995, 1996, 1997, and 1998 WIDE Project.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. Neither the name of the project nor the names of its contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE PROJECT AND CONTRIBUTORS ``AS IS'' AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE PROJECT OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
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 异步套接字服务

asynchat 和 asyncore 模块包含以下声明:

```
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 Cookie 管理

http.cookies 模块包含以下声明:

```
Copyright 2000 by Timothy O'Malley <timo@alum.mit.edu>

               All Rights Reserved

Permission to use, copy, modify, and distribute this software
and its documentation for any purpose and without fee is hereby
granted, provided that the above copyright notice appear in all
copies and that both that copyright notice and this permission
notice appear in supporting documentation, and that the name of
Timothy O'Malley  not be used in advertising or publicity
pertaining to distribution of the software without specific, written
prior permission.

Timothy O'Malley DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS, IN NO EVENT SHALL Timothy O'Malley BE LIABLE FOR
ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

### C.3.5 执行追踪

trace 模块包含以下声明:

```
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 与 UUdecode 函数

uu 模块包含以下声明:

```
Copyright 1994 by Lance Ellinghouse
Cathedral City, California Republic, United States of America.
                        All Rights Reserved
Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation, and that the name of Lance Ellinghouse
not be used in advertising or publicity pertaining to distribution
of the software without specific, written prior permission.
LANCE ELLINGHOUSE DISCLAIMS ALL WARRANTIES WITH REGARD TO
THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS, IN NO EVENT SHALL LANCE ELLINGHOUSE CENTRUM BE LIABLE
FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT
OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

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 远程过程调用

xmlrpc.client 模块包含以下声明:

```
    The XML-RPC client interface is

Copyright (c) 1999-2002 by Secret Labs AB
Copyright (c) 1999-2002 by Fredrik Lundh

By obtaining, using, and/or copying this software and/or its
associated documentation, you agree that you have read, understood,
and will comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this software and
its associated documentation for any purpose and without fee is
hereby granted, provided that the above copyright notice appears in
all copies, and that both that copyright notice and this permission
notice appear in supporting documentation, and that the name of
Secret Labs AB or the author not be used in advertising or publicity
pertaining to distribution of the software without specific, written
prior permission.

SECRET LABS AB AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD
TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANT-
ABILITY AND FITNESS.  IN NO EVENT SHALL SECRET LABS AB OR THE AUTHOR
BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
```

(下页继续)

(繼續上一頁)

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

### C.3.8 test_epoll

`test_epoll` 模块包含以下声明:

```
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` 模块关于 kqueue 的接口包含以下声明:

```
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
```

(下页继续)

(繼續上一頁)

```
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

### C.3.10 SipHash24

Python/pyhash.c 文件包含 Marek Majkowski' 对 Dan Bernstein 的 SipHash24 算法的实现。它包含以下声明:

```
<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 和 dtoa

Python/dtoa.c 文件提供了 C 语言的 dtoa 和 strtod 函数，用于将 C 语言的双精度型和字符串进行转换，由 David M. Gay 的同名文件派生而来，该文件当前可从 http://www.netlib.org/fp/ 下载。2009 年 3 月 16 日检索到的原始文件包含以下版权和许可声明:

```
/****************************************************************
 *
 * 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

如果操作系统可用，则 hashlib, posix, ssl, crypt 模块使用 OpenSSL 库来提高性能。此外，适用于 Python 的 Windows 和 Mac OS X 安装程序可能包括 OpenSSL 库的拷贝，所以在此处也列出了 OpenSSL 许可证的拷贝:

```
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,
```

(下页继续)

(繼續上一頁)

```
    * 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.
    * ====================================================================
    *
    * 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)"
```

(下页继续)

(繼續上一頁)

```
    *
    * 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

除非使用 `--with-system-expat` 配置了构建，否则 `pyexpat` 扩展都是用包含 expat 源的拷贝构建的:

```
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

除非使用 `--with-system-libffi` 配置了构建，否则 `_ctypes` 扩展都是包含 libffi 源的拷贝构建的:

```
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

如果系统上找到的 zlib 版本太旧而无法用于构建，则使用包含 zlib 源代码的拷贝来构建 `zlib` 扩展:

```
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 使用的哈希表的实现基于 cfuhash 项目:

```
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

除非使用 --with-system-libmpdec 配置了构建，否则 _decimal 模块都是用包含 libmpdec 库的拷贝构建的。

```
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.
```

(下页继续)

(繼續上一頁)

```
THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

### C.3.18 W3C C14N 测试套件

`test` 包（lib/test/xmltestdata/c14n-20/）中的 C14N2.0 测试套件来源于 W3C 网站 https://www.w3.org/TR/xml-c14n2-testcases/ ，并根据 BSD 许可证（三条款版）发行：

Copyright (c) 2013 W3C(R) (MIT, ERCIM, Keio, Beihang), All Rights Reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of works must retain the original copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the original copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the W3C nor the names of its contributors may be used to endorse or promote products derived from this work without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ”AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, 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

# 版權宣告

Python 和這些文件是:



完整的授權條款資訊請參見歷史與授權。

# 索引

## 非字母



## A



## B



## C

## D



## E



## F



## G



## H



## I



## J

## Q



## R



## S



## T



## U



## V



## W



## X



## Z