Python Setup and Usage

發🄵 *3.11.11*

Guido van Rossum and the Python development team

12 月 07, 2024

Python Software Foundation
Email: docs@python.org

Contents

這部分的🄴明文件是關於在不同平台上設定 Python 環境的綜合資訊、直譯器的呼叫，以及讓 Python 更容易使用的一些方法。

CHAPTER 1

命令列與環境

为获取各种设置信息，CPython 解析器会扫描命令行与环境。

CPython 實作細節： 其他实现的命令行方案可能会有所不同。详见 implementations。

1.1 命令列

调用 Python 时，可以指定下列任意选项：

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

最常见的用例是启动时执行脚本：

```
python myscript.py
```

1.1.1 介面選項

解释器接口类似于 UNIX shell，但提供了额外的调用方法:

- 用连接到 tty 设备的标准输入调用时，会提示输入并执行命令，输入 EOF （文件结束符，UNIX 中按 `Ctrl-D`，Windows 中按 `Ctrl-Z, Enter`）时终止。
- 用文件名参数或以标准输入文件调用时，读取，并执行该脚本文件。
- 用目录名参数调用时，从该目录读取、执行适当名称的脚本。
- 用 `-c command` 调用时，执行 *command* 表示的 Python 语句。*command* 可以包含用换行符分隔的多条语句。注意，前导空白字符在 Python 语句中非常重要！
- 用 `-m module-name` 调用时，在 Python 模块路径中查找指定的模块，并将其作为脚本执行。

非交互模式下，先解析全部输入，再执行。

接口选项会终结解释器读入的选项列表，所有后续参数都在 `sys.argv` 里 -- 注意，首个元素，即下标为零的元素（`sys.argv[0]`）是表示程序来源的字符串。

-c <command>

執行 *command* 中的 Python 代码。*command* 可以是一条语句，也可以是用换行符分隔的多条语句，其中，前导空白字符与普通模块代码中的作用一样。

使用此选项时，sys.argv 的首个元素为 "-c"，并会把当前目录加入至 sys.path 开头（让该目录中的模块作为顶层模块导入）。

引發一個附帶引數 command 的稽核事件 cpython.run_command。

-m <module-name>

在 sys.path 中搜索指定模块，并以 __main__ 模块执行其内容。

该参数是 模块名，请勿输入文件扩展名（.py）。模块名应为有效的绝对 Python 模块名，但本实现对此不作强制要求（例如，允许使用含连字符 - 的名称）。

包名称（包括命名空间包）也允许使用。使用包名称而不是普通模块名时，解释器把 <pkg>.__main__ 作为主模块执行。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* 环境变量也都会被忽略。

许多标准库模块都包含在执行时，以脚本方式调用的代码。例如 timeit 模块：

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

引發一個附帶引數 module-name 的稽核事件 cpython.run_module。

也參考：

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

在 3.1 版的變更: 提供包名称来运行 __main__ 子模块。

在 3.4 版的變更: 同样支持命名空间包

-

从标准输入 (sys.stdin) 读取命令。标准输入为终端时，使用 *-i*。

使用此选项时，sys.argv 的第一个元素是 "-"，同时，把当前目录加入 sys.path 开头。

引發一個不附帶引數的稽核事件 cpython.run_stdin。

<script>

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

给出此选项时，sys.argv 的第一个元素就是在命令行中指定的脚本名称。

如果脚本名称直接指向 Python 文件，则把该文件所在目录加入 `sys.path` 的开头，并且把该文件当作 `__main__` 模块来执行。

如果脚本名称指向目录或 zip 文件，则把脚本名加入 `sys.path` 的开头，并把该位置中的 `__main__.py` 文件当作 `__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` 既不包含当前目录也不包含用户的 site-packages 目录。所有 `PYTHON*` 环境变量也都会被忽略。

引发一个 审计事件 `cpython.run_file` 并附带参数 `filename`。

也参考:

runpy.run_path()
Python 代码可以直接使用的等效功能

未给出接口选项时，使用*-i*，`sys.argv[0]` 为空字符串 (`""`)，并把当前目录加至 `sys.path` 的开头。此外，如果系统支持，还能自动启用 tab 补全和历史编辑（参见 rlcompleter-config）。

也参考:

tut-invoking

在 3.4 版的變更: 自动启用 tab 补全和历史编辑。

1.1.2 通用选项

-?

-h

--help

打印所有命令行选项及对应环境变量的简短描述然后退出。

--help-env

打印 Python 专属环境变量的简短描述然后退出。

在 3.11 版新加入.

--help-xoptions

打印实现专属*-X* 选项的简短描述然后退出。

在 3.11 版新加入.

--help-all

印出完整使用資訊🄴離開。

在 3.11 版新加入.

-V

--version

输出 Python 版本号并退出。示例如下:

```
Python 3.8.0b2+
```

输入两次 V 选项时，输出更多构建信息，例如:

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

在 3.6 版新加入: `-VV` 選項

1.1.3 其他选项

-b

在将 bytes 或 bytearray 转换为 str 时未指定编码格式或在将 bytes 或 bytearray 与 str 或者在将 bytes 与 int 进行比较时将发出警告。当选项被给出两次 (-bb) 时则会报错。

在 3.5 版的變更: 也会影响 bytes 与 int 的比较。

-B

给出此选项时，Python 不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

Turn on parser debugging output (for expert only, depending on compilation options). See also *PYTHONDEBUG*.

-E

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

另请参阅 *-P* 和 *-I* (隔离) 选项。

-i

脚本是第一个参数，或使用 *-c* 时，即便 sys.stdin 不是终端，执行脚本或命令后，也会进入交互模式。不读取 *PYTHONSTARTUP* 文件。

本选项用于，脚本触发异常时，检查全局变量或堆栈回溯。详见 *PYTHONINSPECT*。

-I

以隔离模式运行 Python。这还将应用 *-E*, *-P* 和 *-s* 选项。

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* 环境变量也都会被忽略。还可以施加更进一步的限制以防止用户注入恶意代码。

在 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*。

在 3.5 版的變更: 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**)。

在 3.5 版的變更: 依据 **PEP 488** 修改 .pyc 文件名。

-P

不要将具有潜在不安全性的路径附加到 sys.path:

- python -m module 命令行: 不要附加当前工作目录。
- python script.py 命令行: 不要附加脚本所在目录。如果是一个符号链接，则会解析符号链接。

- `python -c code` 和 `python` (REPL) 命令行: 不要附加空字符串，这表示当前工作目录。

另请参阅 *PYTHONSAFEPATH* 环境变量，以及 *-E* 和 *-I* (隔离) 选项。

在 3.11 版新加入.

-q

即使在交互模式下也不显示版权和版本信息。

在 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。

在之前版本的 Python 中，此选项会启用哈希随机化，以将字符串和字节串对象的 `__hash__()` 值用不可预测的随机值“加盐”。虽然它们在单个 Python 进程内将保持恒定，但是在重复发起调用的 Python 进程间它们将是不可预测的。

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。请参阅 http://ocert.org/advisories/ocert-2011-003.html 了解详情。

PYTHONHASHSEED 允许你为哈希种子密码设置一个固定值。

在 3.2.3 版新加入.

在 3.7 版的變更: 此选项不会再被忽略。

-s

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

另请参阅 *PYTHONNOUSERSITE*。

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。

另請參🄵 *PYTHONUNBUFFERED*。

在 3.7 版的變更: stdout 和 stderr 流在文本层现在不使用缓冲。

-v

每次在初始化模块时会打印一条信息，显示被加载的地方（文件名或内置模块名）。当给出两个 v（`-vv`）时，搜索模块时会为每个文件打印一条信息。退出时模块清理的信息也会给出来。

在 3.10 版的變更: 由 `site` 模块可以得到将要处理的站点路径和 `.pth` 文件。

另請參🄵 *PYTHONVERBOSE*。

-W `arg`

警告信息的控制。Python 的警告机制默认将警告信息打印到 `sys.stderr`。

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
```

action 名可以根据需要进行缩写，解释器将会解析为合适的名称。例如，`-Wi` 与 `-Wignore` 相同。

完整的参数如下：

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

空字段匹配所有值；尾部的空字段可以省略。例如，`-W ignore::DeprecationWarning` 将忽略所有的 DeprecationWarning 警告。

action 字段如上所述，但只适用于匹配其余字段的警告。

message 字段必须与整个警告信息相匹配；不区分大小写。

category 字段与警告类别相匹配（`DeprecationWarning` 等）。必须是个类名；检测消息的实际警告类别是否为指定类别的子类。

module 字段匹配的是（完整限定）模块名称；这种匹配是大小写敏感的。

lineno 字段匹配行号，其中 0 匹配所有行号，相当于省略了行号。

可以给出多个 *-W* 选项；当某条警告信息匹配上多个选项时，将执行最后一个匹配项的操作。非法 *-W* 选项将被忽略（不过，在触发第一条警告时，会打印出一条无效选项的警告信息）。

警告信息还可以用 *PYTHONWARNINGS* 环境变量来控制，也可以在 Python 程序中用 `warnings` 模块进行控制。例如，`warnings.filterwarnings()` 函数可对警告信息使用正则表达式。

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` 将启用 `faulthandler`。另请参阅 *PYTHONFAULTHANDLER*。
- `-X showrefcount` 可在程序结束时或在交互式解释器每条语句后，输出总的引用计数和使用的内存块数。这只适用于调试版本。
- `-X tracemalloc` 使用 `tracemalloc` 模块启动对 Python 内存分配的跟踪。在默认情况下，只有最近的帧会保存在跟踪的回溯信息中。使用 `-X tracemalloc=NFRAME` 来启动限定回溯 *NFRAME* 帧的跟踪。请参阅 `tracemalloc.start()` 和 *PYTHONTRACEMALLOC* 了解详情。
- `-X int_max_str_digits` 将配置整数字符串转换长度限制。另请参阅 *PYTHONINTMAXSTRDIGITS*。
- `-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 Python 开发模式，引入在默认情况下启用会导致过大开销的运行时检查。另请参阅 *PYTHONDEVMODE*。
- `-X utf8` 启用 Python UTF-8 模式。`-X utf8=0` 将显式地禁用 Python UTF-8 模式 (即使在该模式应该会自动激活时也是如此)。另请参阅 *PYTHONUTF8*。

- `-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* 。
- `-X warn_default_encoding` issues a `EncodingWarning` when the locale-specific default encoding is used for opening files. See also *PYTHONWARNDEFAULTENCODING*.
- `-X no_debug_ranges` 会禁用在代码对象中包括将额外位置信息（结束行、开始列偏移量和结束列偏移量）映射到每条指令的映射表。这在需要较小的代码对象和 pyc 文件时很有用处并可在解释器显示回溯时屏蔽额外的视觉位置提示。另请参阅 *PYTHONNODEBUGRANGES*。
- `-X frozen_modules` 会决定被冻结的模块是否要被导入机制所忽略。值为"on" 表示它们将被导入而"off" 表示它们将被忽略。如果这是已安装的 Python (正常情况) 则默认为"on"。如果是尚在开发中 (从源代码树运行) 则默认为"off"。请注意已冻结的"importlib_bootstrap" 和"importlib_bootstrap_external" 模块将总是会被使用，即使该旗标被设为"off"。

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

在 3.2 版新加入.

在 3.3 版的變更: 增加了 `-X faulthandler` 选项。

在 3.4 版的變更: 增加了 `-X showrefcount` 和 `-X tracemalloc` 选项。

在 3.6 版的變更: 增加了 `-X showalloccount` 选项。

在 3.7 版的變更: 增加了 `-X importtime`, `-X dev` 和 `-X utf8` 选项。

在 3.8 版的變更: 增加了 `-X pycache_prefix` 选项。现在 `-X dev` 选项在 `io.IOBase` 析构器中会记录 `close()` 异常。

在 3.9 版的變更: 使用 `-X dev` 选项，在字符串编码和解码操作时检查 *encoding* 和 *errors* 参数。

`-X showalloccount` 選項已被移除。

在 3.10 版的變更: 增加了 `-X warn_default_encoding` 选项。移除了 `-X oldparser` 选项。

在 3.11 版的變更: 增加了 `-X no_debug_ranges`, `-X frozen_modules` 和 `-X int_max_str_digits` 选项。

1.1.4 你不該使用的選項

-J

保留给 Jython 使用。

1.2 環境變數

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/python*version* 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，正如上文中介面選項 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。

PYTHONSAFEPATH

如果这被设为一个非空字符串，请不要将具有潜在不安全性的路径附加到 sys.path: 参见 *-P* 选项了解详情。

在 3.11 版新加入.

PYTHONPLATLIBDIR

如果它被设为非空字符串，则会覆盖 sys.platlibdir 值。

在 3.9 版新加入.

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__。

引發一個附帶引數 filename 的稽核事件 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。

在 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。

引發一個不附帶引數的稽核事件 cpython.run_stdin。

在 3.11.10 版的變更: (also 3.10.15, 3.9.20, and 3.8.20) Emits audit events.

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

如果设置了此变量，Python 将忽略 `import` 语句中的大小写。这仅在 Windows 和 macOS 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。

在 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。

在 3.2.3 版新加入.

PYTHONINTMAXSTRDIGITS

如果将此变量设为一个整数，它会被用来配置解释器的全局 整数字符串转换长度限制。

在 3.11 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'`。

在 3.4 版的變更: “encodingname” 部分现在是可选的。

在 3.6 版的變更: 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*，它将被用来计算 `user site-packages` 目录以及 `python -m pip install --user` 的 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。这仅在 macOS 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器以转储 Python 回溯信息。此环境变量等价于 *-X* `faulthandler` 选项。

在 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。该变量的值是保存在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()` 函数了解更多信息。这等价于设置 *-X* `tracemalloc` 选项。

在 3.4 版新加入.

PYTHONPROFILEIMPORTTIME

如果此环境变量被设为一个非空字符串，Python 将会显示每次导入耗费了多长时间。这等价于设置 *-X* `importtime` 选项。

在 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。

在 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。

在 3.6 版新加入.

在 3.7 版的變更: 增加了 "default" 分配器。

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。

在 3.6 版的變更: 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

如果设为非空字符串，默认的*filesystem encoding and error handler* 模式将恢复到 3.6 版本之前的值 “mbcs” 和 “replace”。否则，将采用新的默认值 “utf-8” 和 “surrogatepass”。

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

適用：Windows。

在 3.6 版新加入: 更多細節請見 **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。

適用：Windows。

在 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 错误处理器 (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。

適用：Unix。

在 3.7 版新加入: 更多細節請見 **PEP 538**。

PYTHONDEVMODE

如果此环境变量被设为一个非空字符串，则会启用 Python 开发模式，引入在默认情况下启用扩展会导致开销过大的额外运行时检查。这等价于设置 *`-X`* `dev` 选项。

在 3.7 版新加入.

PYTHONUTF8

如果設🄵 `1`，則🄵用 Python UTF-8 Mode。

如果設🄵 `0`，則停用 Python UTF-8 Mode。

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。

在 3.7 版新加入.

PYTHONWARNDEFAULTENCODING

如果该环境变量设为一个非空字符串，则在采用某地区默认编码时，将会引发一条 `EncodingWarning`。

細節請見 io-encoding-warning。

在 3.10 版新加入.

PYTHONNODEBUGRANGES

如果设置了此变量，它会禁用在代码对象中包括将额外位置信息（结束行、开始列偏移量和结束列偏移量）映射到每条指令的映射表。这在需要较小的代码对象和 pyc 文件时很有用处并可在解释器显示回溯时屏蔽额外的视觉位置提示。

在 3.11 版新加入.

1.2.1 除錯模式變數

PYTHONTHREADDEBUG

If set, Python will print threading debug info into stdout.

Need a *debug build of Python*.

自從版本 3.10 後不推薦使用，將會自版本 3.12 中移除。.

PYTHONDUMPREFS

如果设置，Python 将在关闭解释器后转储仍存活的对象和引用计数。

需用 *`--with-trace-refs`* 编译选项来配置 Python。

PYTHONDUMPREFSFILE=FILENAME

如果设置，Python 将在关闭解释器后将仍然存活的对象和引用计数转储至此环境变量给出的路径所对应的文件中。

需用 *`--with-trace-refs`* 编译选项来配置 Python。

在 3.11 版新加入.

CHAPTER 2

在 Unix 平臺上使用 Python

2.1 獲得🄵安裝 Python 的最新版本

2.1.1 在 Linux 上

在大多數 Linux 發行版上會預先安裝 Python，🄵作🄵一個套件提供給所有其他使用者。但是發行版提供的套件可能🄵有你想要使用的某些功能，這時你可以選擇從原始碼編譯最新版本的 Python，做法相當容易。

如果 Python 🄵有預先安裝，🄵且不在發行版提供的儲存庫 (repository) 中，你可以輕松地🄵自己使用的發行版建立套件。參🄵以下連結：

也參考：

https://www.debian.org/doc/manuals/maint-guide/first.en.html
對於 Debian 用🄵

https://en.opensuse.org/Portal:Packaging
對於 OpenSuse 用🄵

https://docs.fedoraproject.org/en-US/package-maintainers/Packaging_Tutorial_GNU_Hello/
對於 Fedora 用🄵

https://slackbook.org/html/package-management-making-packages.html
對於 Slackware 用🄵

2.1.2 在 FreeBSD 和 OpenBSD 上

- FreeBSD 用🄴應使用以下命令增加套件：

```
pkg install python3
```

- OpenBSD 用🄴應使用以下命令增加套件：

```
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 使用者要獲取 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 系統上

你可以從 OpenCSW 獲取、安裝及使用各種版本的 Python。比如 `pkgutil -i python27`。

2.2 建置 Python

如果你想自己編譯 CPython，首先要做的是獲取原始碼。你可以下載最新版本的原始碼，也可以直接提取最新的 clone（克隆）。（如果你想要貢獻修補程式碼，也會需要一份 clone。）

建置過程由幾個常用命令組成：

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

特定 Unix 平臺的配置選項和注意事項通常會詳細地記🄵在 Python 原始碼樹 (source tree) 根目🄵下的 README.rst 檔案中。

警告： `make install` 可以覆蓋或🄵裝 `python3` 二進位制檔案。因此，建議使用 `make altinstall` 而不是 `make install`，因🄵它只安裝 *`exec_prefix`*`/bin/python`*`version`*。

2.3 與 Python 相關的路徑和檔案

這取🄵於本地安裝慣例；*`prefix`* 和 *`exec_prefix`* 相依於安裝方式，應被直譯來讓 GNU 軟體使用；它們也可能相同。

例如，在大多數 Linux 系統上，兩者的預設值皆是 `/usr`。

檔案/目🄵	含意
exec_prefix/bin/python3	直譯器的推薦位置。
prefix/lib/python*version*、*exec_prefix*/lib/python*version*	包含標准模組目🄵的推薦位置。
prefix/include/python*version*、*exec_prefix*/include/python*version*	包含開發 Python 擴充套件和嵌入直譯器所需 include 檔案之目🄵的推薦位置。

2.4 雜項

要在 Unix 上使用 Python 🄵本，你需要讓他們是可執行的 (executable)，例如用

```
$ chmod +x script
```

🄵在🄵本的頂部放一個合適的 Shebang。以下通常是個好選擇：

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

將在整個 PATH 中搜索 Python 直譯器。然而某些 Unix 系統可能🄵有 **env** 命令，因此你可能需要將 /usr/bin/python3 寫死 (hardcode) 成直譯器路徑。

要在 Python 🄵本中使用 shell 命令，請見 subprocess 模組。

2.5 客🄵化 OpenSSL

1. 要使用你所選擇發行商 (vendor) 的 OpenSSL 配置和系統信任儲存區 (system trust store)，請找到包含 openssl.cnf 檔案的目🄵或位於 /etc 的符號連結 (symlink)。在大多數發行版上，該檔案會是在 /etc/ssl 或者 /etc/pki/tls 中。該目🄵亦應包含一個 cert.pem 檔案和/或一個 certs 目🄵。

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

2. 下載、建置🄵安裝 OpenSSL。請確保你使用 install_sw 而不是 install。install_sw 的目標不會覆蓋 openssl.cnf。

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

3. 使用客🄵化 OpenSSL 建置 Python（參見配置 --with-openssl 和 --with-openssl-rpath 選項）

   ```
   $ pushd python-3.x.x
   $ ./configure -C \
   ```

(繼續下一頁)

(繼續上一頁)

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

備🄴: OpenSSL 的修補釋出版 (patch releases) 具有向後相容的 ABI。你不需要重新編譯 Python 來更新 OpenSSL。使用一個新的版本來替代客🄵化 OpenSSL 安裝版就可以了。

CHAPTER 3

配置 Python

3.1 配置选项

用以下方式列出 `./configure` 脚本的所有选项:

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

参阅 Python 源代码中的 `Misc/SpecialBuilds.txt` 。

3.1.1 通用选项

--enable-loadable-sqlite-extensions

在 `sqlite3` 模块的 `_sqlite` 扩展模块中是否支持可加载扩展（默认为否）。

参见 `sqlite3.Connection.enable_load_extension()` 方法的 `sqlite3` 模块。

在 3.6 版新加入.

--disable-ipv6

禁用 IPv6 支持（若开启支持则默认启用），见 `socket` 模块。

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

定义 Python `int` 数字的比特大小：15 或 30 比特

在默认情况下，数位大小为 30。

将 `PYLONG_BITS_IN_DIGIT` 定義🄵 `15` 或 `30`。

參🄵 `sys.int_info.bits_per_digit`。

--with-cxx-main

--with-cxx-main=COMPILER

Compile the Python `main()` function and link Python executable with C++ compiler: `$CXX`, or *COMPILER* if specified.

--with-suffix=SUFFIX

将 Python 的可执行文件后缀设置为 *SUFFIX*。

在 Windows 和 macOS 上默认后缀为 `.exe` (可执行文件为 `python.exe`)，在 Emscripten 节点上为 `.js`，在 Emscripten 浏览器上为 `.html`，在 WASI 上为 `.wasm`，而在其他平台上为一个空字符串 (可执行文件为 `python`)。

在 3.11 版的變更: 在 WASM 平台上默认后缀为 `.js`, `.html` 或 `.wasm` 之一。

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

Select the default time zone search path for 为 `zoneinfo.TZPATH` 选择默认的时区搜索路径。参见 `zoneinfo` 模块的 编译时配置。

默　认: `/usr/share/zoneinfo:/usr/lib/zoneinfo:/usr/share/lib/zoneinfo:/etc/zoneinfo`

参阅 `os.pathsep` 路径分隔符。

在 3.9 版新加入.

--without-decimal-contextvar

编译 `_decimal` 扩展模块时使用线程本地上下文，而不是协程本地上下文（默认），参见 `decimal` 模块。

参见 `decimal.HAVE_CONTEXTVAR` 和 `contextvars` 模块。

在 3.9 版新加入.

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

覆盖 `dbm` 模块的 db 后端检查顺序。

合法值是用冒号（`:`）分隔的字符串，包含后端名称。

- `ndbm`；
- `gdbm`；
- `bdb`。

--without-c-locale-coercion

禁用 C 语言对 UTF-8 的强制要求（默认为启用）。

不定义 `PY_COERCE_C_LOCALE` 宏。

請見 *`PYTHONCOERCECLOCALE`* 與 **PEP 538**。

--with-platlibdir=DIRNAME

Python 库目录名（默认为 `lib`）。

Fedora 和 SuSE 在 64 位平台用 `lib64`。

參🄵 `sys.platlibdir`。

在 3.9 版新加入.

--with-wheel-pkg-dir=PATH

`ensurepip` 模块用到的 wheel 包目录（默认为无）。

某些 Linux 发行版的打包策略建议不要捆绑依赖关系。如 Fedora 在 `/usr/share/python-wheels/` 目录下安装 wheel 包，而不安装 `ensurepip._bundled` 包。

在 3.10 版新加入.

--with-pkg-config=[check|yes|no]
配置是否应当使用 **pkg-config** 来检测构建依赖。

- check (默认值): **pkg-config** 为可选项
- yes: **pkg-config** 为必选项。
- no: 配置不使用 **pkg-config** 即使其存在

在 3.11 版新加入.

--enable-pystats
启用内部统计数据收集。

The statistics will be dumped to a arbitrary (probably unique) file in /tmp/py_stats/, or C:\temp\py_stats\ on Windows.

使用 Tools/scripts/summarize_stats.py 来读取统计数据。

在 3.11 版新加入.

3.1.2 WebAssembly 选项。

--with-emscripten-target=[browser|node]
为 wasm32-emscripten 设置生成风格。

- browser (默认值): 预加载最小 stdlib，默认 MEMFS。
- node: NODERAWFS 和 pthread 支持。

在 3.11 版新加入.

--enable-wasm-dynamic-linking
为 WASM 启用动态链接支持。

动态链接启用 dlopen。可执行文件的大小将由于限制死代码清理和附加特性而增加。

在 3.11 版新加入.

--enable-wasm-pthreads
为 WASM 启用 pthreads 支持。

在 3.11 版新加入.

3.1.3 安装时的选项

--prefix=PREFIX
在 PREFIX 中安装架构无关的文件。在 Unix 上，它默认为 /usr/local。

该值可在运行时使用 sys.prefix 获取。

作为示例，用户可以使用 --prefix="$HOME/.local/" 在其家目录中安装 Python。

--exec-prefix=EPREFIX
在 EPREFIX 中安装架构无关的文件，默认为*--prefix*。

该值可在运行时使用 sys.exec_prefix 获取。

--disable-test-modules

不编译和安装 test 模块，如 `test` 包或 `_testcapi` 扩展模块（默认会编译并安装）。

在 3.10 版新加入.

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

选择 Python 安装时运行的 `ensurepip` 命令。

- `upgrade` （默认）：运行 `python -m ensurepip --altinstall --upgrade` 命令。
- `install`： 运行 `python -m ensurepip --altinstall` 命令。
- `no`： 不运行 ensurepip。

在 3.6 版新加入.

3.1.4 性能选项

Configuring Python using `--enable-optimizations --with-lto` (PGO + LTO) is recommended for best performance.

--enable-optimizations

用 *PROFILE_TASK* 启用以配置文件主导的优化（PGO）（默认为禁用）。

C 编译器 Clang 需要用到 `llvm-profdata` 程序进行 PGO。在 macOS 上，GCC 也需要用到它：在 macOS 上 GCC 只是 Clang 的别名而已。

如果使用 `--enable-shared` 和 GCC，还可以禁用 libpython 中的语义插值：在编译器和链接器的标志中加入 `-fno-semantic-interposition`。

在 3.6 版新加入.

在 3.10 版的變更: 在 GCC 上使用 `-fno-semantic-interposition`。

PROFILE_TASK

Makefile 用到的环境变量：PGO 用到的 Python 命令行参数。

默认为：`-m test --pgo --timeout=$(TESTTIMEOUT)`。

在 3.8 版新加入.

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

在编译过程中启用链接时间优化（LTO）（默认为禁用）。

LTO 时 C 编译器 Clang 需要用到 `llvm-ar` 参数（在 macOS 则为 `ar`），以及支持 LTO 的链接器（`ld.gold` 或 `lld`）。

在 3.6 版新加入.

在 3.11 版新加入: 要使用 ThinLTO 特性，请在 Clang 上使用 `--with-lto=thin`。

--with-computed-gotos

在求值环节启用 goto 计数（在支持的编译器上默认启用）。

--without-pymalloc

禁用特定的 Python 内存分配器 pymalloc （默认为启用）。

另請參🄵 *PYTHONMALLOC* 環境變數。

--without-doc-strings

禁用静态文档字符串以减少内存占用（默认启用）。Python 中定义的文档字符串不受影响。

不定义 PY_COERCE_C_LOCALE 宏。

参阅宏 PyDoc_STRVAR() 。

--enable-profiling

用 gprof 启用 C 语言级的代码评估（默认为禁用）。

3.1.5 Python 调试级编译

调试版本 Python 是指带有*--with-pydebug* 参数的编译。

调试版本的效果：

- 默认显示所有警告：在 warnings 模块中，默认警告过滤器的列表是空的。
- 在 sys.abiflags 中加入 d 标记。
- 加入 sys.gettotalrefcount() 函数。
- 命令行参数加入*-X showrefcount* 。
- Add *PYTHONTHREADDEBUG* environment variable.
- 添加对 __lltrace__ 变量的支持：如果定义了该变量则会在字节码求值循环中启用低层级追踪。
- 安装 内存分配调试钩子，以便检测缓冲区溢出和其他内存错误。
- 定义宏 Py_DEBUG 和 Py_REF_DEBUG 。
- 增加运行时检查：针对由 #ifdef Py_DEBUG 和 #endif 所包裹的代码。启用 assert(...) 和 _PyObject_ASSERT(...) 断言：不设置 NDEBUG 宏（另请参阅*--with-assertions* 配置选项）。主要的运行时检查有：
  - 增加了对函数参数的合理性检查。
  - 创建 Unicode 和 int 对象时，内存按某种模式进行了填充，用于检测是否使用了未初始化的对象。
  - 确保有能力清除或替换当前异常的函数在调用时不会引发异常。
  - 检查内存释放器函数是否不改变当前异常。
  - 垃圾收集器（gc.collect() 函数）对对象的一致性进行一些基本检查。
  - 从较宽类型转换到较窄类型时，Py_SAFE_DOWNCAST() 宏会检查整数下溢和上溢的情况。

参见 Python 开发模式和配置参数*--with-trace-refs* 。

在 3.8 版的變更: 发布版和调试版的编译现在是 ABI 兼容的：定义了 Py_DEBUG 宏不再意味着同时定义了 Py_TRACE_REFS 宏（参见*--with-trace-refs* 参数），这引入了唯一一处不是 ABI 兼容的地方。

3.1.6 调试选项

--with-pydebug

在调试模式下编译 Python: 定义宏 `Py_DEBUG` (默认为禁用)。

--with-trace-refs

为了调试而启用引用的跟踪（默认为禁用）。

效果如下：

- 定义 `Py_TRACE_REFS` 宏。
- 加入 `sys.getobjects()` 函数。
- 环境变量加入 *PYTHONDUMPREFS* 。

此版本与发布模式（默认编译模式）或调试模式（`Py_DEBUG` 和 `Py_REF_DEBUG` 宏）不具备 ABI 兼容性。

在 3.8 版新加入.

--with-assertions

编译时启用 C 断言：`assert(...);` 和 `_PyObject_ASSERT(...);` （默认不启用）。

如果设置此参数，则在 *OPT* 编译器变量中不定义 `NDEBUG` 宏。

参阅*--with-pydebug* 选项（*调试编译模式*），它也可以启用断言。

在 3.6 版新加入.

--with-valgrind

启用 Valgrind （默认禁用）。

--with-dtrace

启用 DTrace（默认禁用）。

参阅 用 DTrace 和 SystemTap 测试 CPython。

在 3.6 版新加入.

--with-address-sanitizer

启用 AddressSanitizer 内存错误检测 `asan`，（默认为禁用）。

在 3.6 版新加入.

--with-memory-sanitizer

启用 MemorySanitizer 内存错误检测 `msan`，（默认为禁用）。

在 3.6 版新加入.

--with-undefined-behavior-sanitizer

启用 undefinedBehaviorSanitizer 未定义行为检测 `ubsan`，（默认为禁用）。

在 3.6 版新加入.

3.1.7 链接器选项

--enable-shared
启用共享 Python 库 `libpython` 的编译（默认为禁用）。

--without-static-libpython
不编译 `libpythonMAJOR.MINOR.a`，也不安装 `python.o` (默认会编译并安装)。

在 3.10 版新加入.

3.1.8 库选项

--with-libs='lib1 ...'
链接附加库（默认不会）。

--with-system-expat
用已安装的 `expat` 库编译 `pyexpat` 模块（默认为否）。

--with-system-ffi
Build the `_ctypes` extension module using an installed `ffi` library, see the `ctypes` module (default is system-dependent).

--with-system-libmpdec
用已安装的 `mpdec` 库编译 `_decimal` 扩展模块，参见 `decimal` 模块（默认为否）。

在 3.3 版新加入.

--with-readline=editline
用 `editline` 库作为 `readline` 模块的后端。

定义 `WITH_EDITLINE` 宏。

在 3.10 版新加入.

--without-readline
不编译 `readline` 模块（默认会）。

不定义 `HAVE_LIBREADLINE` 宏。

在 3.10 版新加入.

--with-libm=STRING
将 `libm` 数学库覆盖为 *STRING* (默认情况视系统而定)。

--with-libc=STRING
将 `libc` C 库覆盖为 *STRING* (默认情况视系统而定)。

--with-openssl=DIR
OpenSSL 的根目录。

在 3.7 版新加入.

--with-openssl-rpath=[no|auto|DIR]
设置 OpenSSL 库的运行时库目录（rpath）。

- `no` (默认): 不设置 rpath。
- `auto`: 根据*--with-openssl* 和 `pkg-config` 自动检测 rpath。
- *DIR*： 直接设置 rpath。

在 3.10 版新加入.

3.1.9 安全性选项

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

选择 Python/pyhash.c 采用的哈希算法。

- siphash13 (默认值);
- siphash24;
- fnv.

在 3.4 版新加入.

在 3.11 版新加入: 增加了 siphash13 并且是新的默认值。

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

内置哈希模块:

- md5;
- sha1;
- sha256;
- sha512;
- sha3 (带 shake)。
- blake2。

在 3.9 版新加入.

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

覆盖 OpenSSL 默认的密码套件字符串。

- python (默认值): 采用 Python 推荐选择。
- openssl: 保留 OpenSSL 默认值不动。
- *STRING*: 采用自定义字符串。

参见 ssl 模块。

在 3.7 版新加入.

在 3.10 版的變更: 设置 python 和 *STRING* 也会把 TLS 1.2 设为最低版本的协议。

3.1.10 macOS 选项

参🄵 Mac/README.rst。

--enable-universalsdk

--enable-universalsdk=SDKDIR

创建通用的二进制版本。*SDKDIR* 指定应采用的 macOS SDK （默认为否）。

--enable-framework

--enable-framework=INSTALLDIR

创建 Python.framework，而不是传统的 Unix 安装版。可选参数 *INSTALLDIR* 指定了安装路径（(默认为否)。

--with-universal-archs=ARCH

指定应创建何种通用二进制文件。该选项仅当设置了*--enable-universalsdk* 时才有效。

可选项：

- universal2;
- 32-bit;
- 64-bit;
- 3-way;
- intel;
- intel-32;
- intel-64;
- all。

--with-framework-name=FRAMEWORK

为 macOS 中的 python 框架指定名称，仅当设置了*--enable-framework* 时有效（默认：Python)。

3.1.11 交叉编译选项

交叉编译，或称交叉构建，可被用于为不同的 CPU 架构或平台构建 Python。交叉编译需要一个针对构建平台的 Python 解释器。构建的 Python 版本必须与交叉编译的主机 Python 版本相匹配。

--build=BUILD

用于在 BUILD 上执行构建的配置，通常由 **config.guess** 通过推测得到。

--host=HOST

交叉编译以构建在 HOST (目标平台) 上运行的程序

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

针对交叉编译构建 python 二进制文件的路径

在 3.11 版新加入.

CONFIG_SITE=file

指向一个带有配置重载的的文件的环境变量。

示例 *config.site* 文件：

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

交叉编译示例:

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

3.2 Python 构建系统

3.2.1 构建系统的主要文件

- `configure.ac` => `configure`;
- `Makefile.pre.in` => `Makefile` (由 `configure` 创建);
- `pyconfig.h` (created by `configure`);
- `Modules/Setup`: 由 Makefile 使用 `Module/makesetup` shell 脚本构建的 C 扩展;
- `setup.py`: C extensions built using the `distutils` module.

3.2.2 主要构建步骤

- C 文件（`.c`）是作为对象文件（`.o`）构建的。
- 一个静态库 `libpython`（`.a`）是由对象文件创建的。
- `python.o` 和静态库 `libpython` 被链接到最终程序 `python` 中。
- C 扩展由 Makefile （参见 `Modules/Setup`）和 `python setup.py build` 构建。

3.2.3 主要 Makefile 目标

- `make`： 用标准库构建 Python。
- `make platform:`： 构建 `python` 程序，但不构建标准库扩展模块。
- `make profile-opt`： 使用 Profile Guided Optimization (PGO) 构建 Python 。你可以使用 configure 的*`--enable-optimizations`* 选项来使其成为 `make` 命令的默认目标（`make all` 或只是 `make`）。
- `make buildbottest`： 构建 Python 并运行 Python 测试套件，与 buildbots 测试 Python 的方式相同。设置变量 `TESTTIMEOUT`（单位：秒）来改变测试超时时间（默认为 1200 即 20 分钟）。
- `make install`: 构建并安装 Python 。
- `make regen-all`：重新生成（几乎）所有生成的文件；`make regen-stdlib-module-names` 和 `autoconf` 必须对其余生成的文件单独运行。
- `make clean`： 移除构建的文件。
- `make distclean`： 与 `make clean` 相同，但也删除由配置脚本创建的文件。

3.2.4 C 扩展

有些 C 扩展是作为内置模块构建的，比如 `sys` 模块。它们在定义了 `Py_BUILD_CORE_BUILTIN` 宏的情况下构建。内置模块没有 `__file__` 属性:

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

其他 C 扩展是作为动态库来构建的，比如 _asyncio 模块。它们在定义了 Py_BUILD_CORE_MODULE 宏的情况下构建。在 Linux x86-64 上的例子：

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

Modules/Setup 用于生成 Makefile 目标，以构建 C 扩展。在文件的开头，C 被构建为内置模块。在标记 *shared* 之后定义的扩展被构建为动态库。

setup.py 脚本只使用 distutils 模块将 C 构建为共享库。

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

- 如果 Py_BUILD_CORE_MODULE 定义了，使用 Py_EXPORTED_SYMBOL 。
- 否则使用 Py_IMPORTED_SYMBOL 。

如果宏 Py_BUILD_CORE_BUILTIN 被错误地用在作为共享库构建的 C 扩展上，它的 PyInit_*xxx*() 函数就不会被导出，导致导入时出现 ImportError 。

3.3 编译器和链接器的标志

脚本 ./configure 和环境变量设置的选项，并被 Makefile 使用。

3.3.1 预处理器的标志

CONFIGURE_CPPFLAGS

变量 *CPPFLAGS* 的值被传递给 ./configure 脚本。

在 3.6 版新加入.

CPPFLAGS

(Objective) C/C++ 预处理器标志，例如，如果头文件位于非标准的目录 *include_dir* 中，请使用 -I*include_dir* 。

CPPFLAGS 和 *LDFLAGS* 都需要包含 shell 的值，以便 setup.py 能够使用环境变量中指定的目录构建扩展模块。

BASECPPFLAGS

在 3.4 版新加入.

PY_CPPFLAGS

为构建解释器对象文件增加了额外的预处理器标志。

默认为：$(BASECPPFLAGS) -I. -I$(srcdir)/Include $(CONFIGURE_CPPFLAGS) $(CPPFLAGS) 。

在 3.2 版新加入.

3.3.2 编译器标志

CC

C 编译器指令。

例如：`gcc -pthread`。

MAINCC

C compiler command used to build the `main()` function of programs like `python`.

Variable set by the *`--with-cxx-main`* option of the configure script.

Default: `$(CC)`.

CXX

C++ 编译器指令。

Used if the *`--with-cxx-main`* option is used.

例如：`g++ -pthread`。

CFLAGS

C 编译器标志。

CFLAGS_NODIST

`CFLAGS_NODIST` 用于构建解释器和 stdlib C 扩展。当 Python 安装后，编译器标志 不应该成为 distutils *`CFLAGS`* 的一部分时，可以使用它（bpo-21121）。

特别地，*`CFLAGS`* 不应当包含:

- 编译器旗标 `-I` (用于为包括文件设置搜索路径)。`-I` 旗标将按从左到右的顺序处理，并且 *`CFLAGS`* 中的任何旗标都将优先于 user- 和 package- 层级所提供的 `-I` 旗标。
- 加固旗标如 `-Werror` 因为分发版无法控制由用户安装的包是否符合这样的高标准。

在 3.5 版新加入.

EXTRA_CFLAGS

而外的 C 编译器指令。

CONFIGURE_CFLAGS

变量 *`CFLAGS`* 的值传递给 `./configure` 脚本。

在 3.2 版新加入.

CONFIGURE_CFLAGS_NODIST

变量 *`CFLAGS_NODIST`* 的值传递给 `./configure` 脚本。

在 3.5 版新加入.

BASECFLAGS

基础编译器标志。

OPT

优化标志。

CFLAGS_ALIASING

严格或不严格的别名标志，用于编译 `Python/dtoa.c` 、

在 3.7 版新加入.

CCSHARED

用于构建共享库的编译器标志。

例如🄵 `-fPIC` 被使用於 Linux 與 BSD 上。

CFLAGSFORSHARED

为构建解释器对象文件增加了额外的 C 标志。

，默认为：`$(CCSHARED)`，当*`--enable-shared`* 被使用时，则为空字符串

PY_CFLAGS

默认为：`$(BASECFLAGS) $(OPT) $(CONFIGURE_CFLAGS) $(CFLAGS) $(EXTRA_CFLAGS)`。

PY_CFLAGS_NODIST

默认为：`$(CONFIGURE_CFLAGS_NODIST) $(CFLAGS_NODIST) -I$(srcdir)/Include/internal`。

在 3.5 版新加入.

PY_STDMODULE_CFLAGS

用于构建解释器对象文件的 C 标志。

默认为：`$(PY_CFLAGS) $(PY_CFLAGS_NODIST) $(PY_CPPFLAGS) $(CFLAGSFORSHARED)`。

在 3.7 版新加入.

PY_CORE_CFLAGS

默认为 `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE`。

在 3.2 版新加入.

PY_BUILTIN_MODULE_CFLAGS

编译器标志，将标准库的扩展模块作为内置模块来构建，如 `posix` 模块

默认为：`$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE_BUILTIN`。

在 3.8 版新加入.

PURIFY

Purify 命令。Purify 是一个内存调试程序。

默认为：空字符串（不使用）。

3.3.3 链接器标志位

LINKCC

用于构建如 `python` 和 `_testembed` 的程序的链接器命令。

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

CONFIGURE_LDFLAGS

变量 *`LDFLAGS`* 的值被传递给 `./configure` 脚本。

避免指定 *`CFLAGS`*，*`LDFLAGS`* 等，这样用户就可以在命令行上使用它们来追加这些值，而不用触碰到预设的值。

在 3.2 版新加入.

LDFLAGS_NODIST

LDFLAGS_NODIST 的使用方式与 *CFLAGS_NODIST* 相同。当 Python 安装后，链接器标志 不应该成为 distutils *LDFLAGS* 的一部分时，可以使用它（bpo-35257 ）。

特别地，*LDFLAGS* 不应当包含:

- 编译器旗标 `-L` (用于为库设置搜索路径)。`-L` 旗标将按从左到右的顺序处理，并且 *LDFLAGS* 中的任何旗标都将优先于 user- 和 package 层级所提供的 `-L` 旗标。

CONFIGURE_LDFLAGS_NODIST

变量 *LDFLAGS_NODIST* 的值传递给 `./configure` 脚本。

在 3.8 版新加入.

LDFLAGS

链接器标志，例如，如果库位于非标准的目录 *lib_dir* 中，请使用 `-L`*lib_dir* 。

CPPFLAGS 和 *LDFLAGS* 都需要包含 shell 的值，以便 setup.py 能够使用环境变量中指定的目录构建扩展模块。

LIBS

链接器标志，在链接 Python 可执行文件时将库传递给链接器。

例如: `-lrt` 。

LDSHARED

构建一个共享库的命令。

默认为: `@LDSHARED@ $(PY_LDFLAGS)` 。

BLDSHARED

构建共享库 `libpython` 的命令。

默认为: `@BLDSHARED@ $(PY_CORE_LDFLAGS)` 。

PY_LDFLAGS

默认为: `$(CONFIGURE_LDFLAGS) $(LDFLAGS)` 。

PY_LDFLAGS_NODIST

默认为: `$(CONFIGURE_LDFLAGS_NODIST) $(LDFLAGS_NODIST)` 。

在 3.8 版新加入.

PY_CORE_LDFLAGS

用于构建解释器对象文件的链接器标志。

在 3.8 版新加入.

CHAPTER 4

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。

不同于大多数 Unix 系统和服务，Windows 未包括任何受系统支持的 Python 预安装版。为了让 Python 可用，多年以来 CPython 团队为每个 发布版 编译了 Windows 安装程序。这些安装程序主要被用来安装用户级 Python 安装版，包含供单独用户使用的核心解释器和库。安装程序也能够为单台机器上的所有用户进行安装，还提供了针对应用程序本地分发版的单独 ZIP 文件。

如 **PEP 11** 所述，Python 发布版对某个 Windows 平台的支持仅限于被 Microsoft 视为处于延长支持周期内的版本。这意味着 Python 3.11 支持 Windows 8.1 及其后的版本。如果你需要 Windows 7 支持，请安装 Python 3.8。

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 是一个适用于运行脚本和包，并使用 IDLE 或其他开发环境的简易 Python 安装版。它需要 Windows 10 或更新的系统，但可以安全地安装而不会破坏其他程序。它还提供了许多便捷命令用来启动 Python 及其工具。

nuget.org 套件 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。

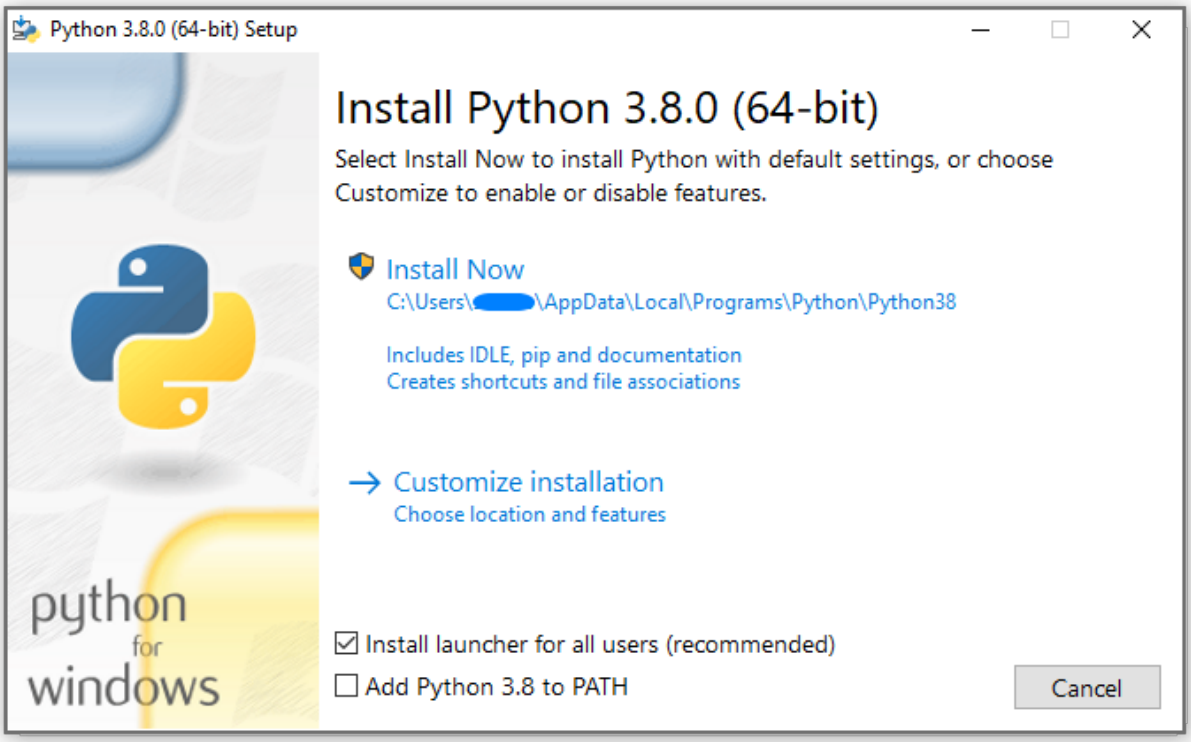
可嵌入的包 是 Python 的最小安装包，适合嵌入到更大的应用程序中。

4.1 完整安装程序

4.1.1 安装步骤

四个 Python 3.11 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🄵將被加入到你的 PATH
- 安装捷徑將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

4.1.2 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。

在 3.6 版的變更: Python 中启用了对长路径的支持。

4.1.3 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。

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

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\Python\PythonXY` 或 `%LocalAppData%\Programs\Python\PythonXY-32` 或 `%LocalAppData%\Programs\Python\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	将安装和脚本目录添加到 `PATH` 并将 `.PY` 添加到 `PATHEXT`	0
AppendPath	将安装和脚本目录添加到 `PATH` 并将 `.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_debu	安装调试二进制文件	0
Include_dev	安装开发者头文件和库文件。省略这一步可能导致安装不可用。	1
Include_exe	安装 `python.exe` 以及相关文件。忽略此项可能会导致安装不可用。	1
Include_launc	安装*适用于 Windows 的 Python 启动器*。	1
InstallLauncherAllUsers	为所有用户安装启动器。还需要 `Include_launcher` 被设定为 1	1
Include_lib	安装标准库和扩展模块。省略这一步可能导致安装不可用。	1
Include_pip	安装捆绑的 pip 和 setuptools	1
Include_symb	安装调试符号集 (`*.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
LauncherOn	仅安装启动器。这将覆盖大多数其他选项。	0

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-3.9.0.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-3.9.0.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>
```

4.1.4 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。请记得要将 `python-3.9.0.exe` 替换为安装程序的实际名称，并在单独的目录中创建子目录以避免同名文件间的冲突。

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

您也可以指定 `/quiet` 选项来隐藏进度显示。

4.1.5 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。

4.2 Microsoft Store 包

在 3.7.2 版新加入.

Microsoft Store 包是一个易于安装的 Python 解释器，主要针对在交互模式下使用，例如用于教学。

要安装此软件包，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 3.11"。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 3.11 ）。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

4.2.1 已知的问题

本地数据、注册表项和临时路径的重定向

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。相反同，它将写入到一个私有副本。如果你的脚本必须修改共享位置，则需要安装完整的安装器。

在运行时，Python 将使用知名 Windows 文件夹和注册表项的一个私有副本。例如，如果环境变量 `%APPDATA%` 为 `c:\Users\<user>\AppData\`，则当写入 `C:\Users\<user>\AppData\Local` 时将会写入到 `C:\Users\<user>\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.8_qbz5n2kfra8p0\LocalCache\Local\`。

当读取文件时，Windows 将返回来自私有文件夹的文件，或者如果文件不存在，则返回来自知名 Windows 目录的文件。例如读取 `C:\Windows\System32` 将返回 `C:\Windows\System32` 的内容加上 `C:\Program Files\WindowsApps\package_name\VFS\SystemX86` 的内容。

你可以使用 `os.path.realpath()` 找到任何现有文件的真实路径:

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

当写入到 Windows 注册表时，会存在以下行为:

- 从 `HKLM\\Software` 读取是被允许的并且其结果将与包中的 `registry.dat` 文件合并。
- 当相应的键/值存在时向 `HKLM\\Software` 写入，即修改现有键的值是不被允许的。

- 当包中相应的键/值不存在并且用户具有正确的访问权限时向 HKLM\\Software 写入是被允许的。

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

4.3 nuget.org 套件

在 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。

4.4 可嵌入的包

在 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。

在解压缩后，嵌入的分发包（几乎）与用户的系统完全隔离，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 文件被包括在一个 ZIP 文件中，并提供了 python3.dll, python37.dll, python.exe 和 pythonw.exe。其中将不包括 Tcl/tk（包括所有依赖它的包，如 Idle 等）、pip 和 Python 文档。

備🄵: 嵌入式发行版不包括 Microsoft C 运行时，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。

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。

4.4.1 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。

4.4.2 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。

4.5 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。

Enthought Deployment Manager
: “下一代的 Python 环境和包管理器”

  之前 Enthought 提供了 Canopy，但已经 于 2016 年结束生命期。

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。

4.6 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* 。

4.6.1 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 3.9;%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”，或打开系统属性的 高级系统设置，然后单击 环境变量按钮。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 会将用户变量串联在系统变量 之后，这可能会在修改 PATH 时导致意外结果。

PYTHONPATH 变量被 Python 的所有版本使用，因此除非它列出的路径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://docs.microsoft.com/en-us/windows/win32/procthread/environment-variables
Windows 上的環境變數概要

https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/set_1
用于临时修改环境变量的 set 命令

https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/setx
用于永久修改环境变量的 setx 命令

4.6.2 查找 Python 可执行文件

在 3.5 版的變更.

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，请参阅命令列 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 3.9
```

4.7 UTF-8 模式

在 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.getencoding()`)。

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

当 Python UTF-8 模式启用时，你仍然可以通过"mbcs" 编解码器使用系统编码格式（ANSI 代码页）。

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 (参见 **PEP 529** 了解详情)。

4.8 适用于 Windows 的 Python 启动器

在 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。

4.8.1 開始

从命令行

在 3.6 版的變更.

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，3.7 和 3.11），您会注意到 Python 3.11 启动 - 如果要启动 Python 3.7，尝试命令：

```
py -3.7
```

如果您想使用已安装的 Python 2 的最新版本，请尝试以下命令：

```
py -2
```

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.
```

指令：

```
py --list
```

显示当前已安装的 Python 版本。

`-x.y` 参数是 `-V:Company/Tag` 参数的简短形式，它允许选择一个特定的 Python 运行时，包括可能来自于 python.org 以外地方的版本。任何遵循 **PEP 514** 进行注册的运行时都将是可被发现的。`--list` 命令将列出所有使用 `-V:` 格式的可用运行时。

当使用 `-V:` 参数时，指定 Company 将把选择限制到来自该提供方的运行时，而仅指定 Tag 将选择来自所有提供方的运行时。请注意省略斜杠将会视作是一个 Tag:

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

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

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

该参数的简短形式 (`-3`) 将只选择来自核心 Python 发布版的运行时，而不选择其他分发版。但是，完整形式 (`-V:3`) 则将选择来自任何版本的运行时。

Company 是在完整字符串上以大小写不敏感的方式进行匹配。Tag 则是在完整字符串或前缀上进行匹配，具体取决于下一个字符是点号还是横杠。这将允许 `-V:3.1` 匹配 `3.1-32`，但不匹配 `3.10`。Tag 是按数字值进行排序 (`3.10` 比 `3.1` 新)，但是按文本进行比较 (`-V:3.01` 将不匹配 `3.1`)。

🄵擬環境（Virtual environment）

在 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将打印最新的 Python 3.x 信息。如上面的命令行示例一样，你可以更明确地指定版本限定符。假设你已安装了 Python 3.7，请尝试将第一行改为 `#! python3.7` 那么你应当看到打印出了 3.7 的版本信息。

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。

4.8.2 Shebang 行

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

例如，如果脚本开始的第一行为

```
#! /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/python3.7-32` 将请求使用 32 位 python 3.7。

在 3.7 版新加入: 从 python 启动器 3.7 开始，可以通过"-64" 后缀调用 64 位版本。此外还可以指定一个主版本号加架构而不带次版本号 (即 `/usr/bin/python3-64`)。

在 3.11 版的變更: "-64" 后缀已被弃用，现在会被视为 "任何不被确定为 i386/32 位的架构"。要请求一个特定的环境，请使用新的 `-V:TAG` 参数并附带完整的标签。

shebang 行的 `/usr/bin/env` 形式具有一个额外的特别属性。在查找已安装的 Python 解释器时，此形式将在可执行程序目录 `PATH` 中搜索与作为第一个参数传入的名称相匹配的 Python 可执行程序。这对应于 Unix 中 `PATH` 执行搜索的 `env` 程序的行为。如果无法找到与 `env` 命令之后的第一个参数相匹配的可执行程序，但该参数是以 `python` 开头的，它将按针对其他虚拟命令的描述来处理。可以设置环境变量 `PYLAUNCHER_NO_SEARCH_PATH` (为任意值) 来跳过对 `PATH` 的搜索。

无法匹配这些模式中任何一个的井号叹号行将在启动器的*INI 文件* 的 `[commands]` 一节中查找。这可被用来以对你的系统来说有意义的方式处理某些命令。命名的名称必须是一个单独的参数（在井号叹号行的可执行程序中不可有空格），而被替代的值则是该可执行程序的完整路径（在.INI 中指定的附加参数将作为文件名的一部分被引用）。

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

任何未出现在.INI 文件中的命令都会被当作 **Windows** 可执行程序的绝对或相对于包含脚本文件的目录的路径。这对于 Windows 专属的脚本来说很方便，例如由安装器所生成的脚本，因为此行为与 Unix 风格的 shell 是不兼容的。这些路径可以加上引号，并可以包含多个参数，在它之后将会加上脚本路径以及任何附加参数。

4.8.3 shebang 行的参数

shebang 行还可以指定要传递给 Python 解释器的其他选项。举例来说，如果你有这样的 shebang 行:

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

那么 Python 将以 `-v` 选项启动

4.8.4 自定义

通过 INI 文件自定义

启动器将搜索两个.ini 文件——当前用户应用程序数据目录中的 `py.ini` (`%LOCALAPPDATA%` 或 `$env:LocalAppData`) 以及启动器所在目录中的 `py.ini`。同样的.ini 文件还会被用于启动器的 '控制台' 版本 (即 py.exe) 和 '窗口' 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 行的 `#!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。

範例：

- 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 python3.7 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=3.7-32 ，命令 python 将使用 3.7 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=3.7 ，命令 python 和 python3 都将特别使用 3.7

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=3.7 等同于包含以下内容的 INI 文件：

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

- 设置 PY_PYTHON=3 和 PY_PYTHON3=3.7 相当于包含以下内容的 INI 文件：

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

4.8.5 诊断

如果环境变量 PYLAUNCHER_DEBUG 已设置（为任何值），启动器将把诊断信息打印到 stderr（即控制台）。此信息会尽量做到既详细 又简洁，它应当允许你查看已被定位的 Python 的版本，特定版本为何被选择以及被用于执行目标 Python 的实际命令行。它的主要目标是用于测试和调试。

4.8.6 试运行

如果环境变量 PYLAUNCHER_DRYRUN 已设置（为任意值），启动器将输出它将要运行的命令，但不会实际启动 Python。这对于想要使用启动器执行检测然后再直接启动 Python 的工具来说很有用处。请注意写入到标准输出的命令总是会使用 UTF-8 来编码，因而在控制台中可能无法正确渲染。

4.8.7 安裝隨選

如果环境变量 PYLAUNCHER_ALLOW_INSTALL 已经设置（为任何值），而所请求的 Python 版本没有安装但可以在 Microsoft Store 获得，启动器将尝试安装它。这可能需要用户进行交互来完成，你可能需要再次运行此命令。

额外的 PYLAUNCHER_ALWAYS_INSTALL 变量将导致启动器总是尝试安装 Python，即使它已经被检测到。这主要是出于测试目的（并且应当与 PYLAUNCHER_DRYRUN 一起使用）。

4.8.8 返回码

Python 启动器可能返回以下的退出码。不幸的是，没有任何办法可以将这些退出码与 Python 本身的退出码区分开来。

退出码的名称将在源代码中使用，并且仅供参考。除了阅读本页面以外没有其他办法可以获取或解读它们。这些条目是以名称的字母顺序列出的。

名称	值	描述
RC_BAD_VENV_CFG	107	找到了 pyvenv.cfg 但文件已损坏。
RC_CREATE_PROCESS	101	启动 Python 失败。
RC_INSTALLING	111	安装已启动，但命令需要在其完成后重新运行。
RC_INTERNAL_ERROR	109	未预期的错误。请报告程序错误。
RC_NO_COMMANDLINE	108	无法从操作系统获取命令行。
RC_NO_PYTHON	103	无法定位所请求的版本。
RC_NO_VENV_CFG	106	需要 pyvenv.cfg 但没有找到。

4.9 查找模块

这些注释以详细的 Windows 注释对 sys-path-init 中的描述进行了补充。

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，如環境變數 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）。
- 额外的”应用程序路径”可以作为子键被同时添加到注册表 HKEY_CURRENT_USER 和 HKEY_LOCAL_MACHINE 分支下的 \SOFTWARE\Python\PythonCore{version}\PythonPath 中。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都被添加到 sys.path 中。（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 `python.exe` 的分发版），请确保安装目录中存在 landmark 文件 (`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。

在 3.6 版的變更: 添加 `._pth` 文件支持并从 `pyvenv.cfg` 中移除了 `applocal` 选项。

在 3.6 版的變更: 当与可执行文件直接相邻时将添加 `pythonXX.zip` 作为潜在的标志物。

在 3.6 版之後被🄵用: 在 `Modules` (不是 `PythonPath`) 下的注册表中指定的模块可以通过 `importlib.machinery.WindowsRegistryFinder` 导入。在 Windows 上此查找器在 3.6.0 及更早版本中被启用，但在将来可能需要显式地添加到 `sys.meta_path`。

4.10 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。

4.10.1 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 呼叫
- 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。

也參考:

Win32 How Do I...?
由 Tim Golden 所著

Python and COM
由 David 與 Paul Boddie 所著

4.10.2 cx_Freeze

cx_Freeze 将 Python 脚本包装成可执行的 Windows 程序 (`*.exe` 文件)。当你完成此操作后，你就可以分发你的应用程序而无需用户安装 Python。

4.11 編譯 Python 在 Windows

如果你想要自己编译 CPython，首先要做的是获取 源代码。你可以下载最新发行版的源代码或是执行最新的签出。

源代码树包含 Microsoft Visual Studio 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 。

4.12 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 自 Python 3 起 不再受支持 (如果曾经受支持的话)。
- Cygwin <https://cygwin.com/>`_ 安装程序也提供了安装 'Python 解释器 的功能。

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 5

在 Mac 系統使用 Python

作者
Bob Savage <bobsavage@mac.com>

运行 macOS 的 Mac 上的 Python 在原则上与其他 Unix 平台上的 Python 非常相似，但有一些额外的特性如集成开发环境（IDE）和包管理器值得指出。

5.1 获取和安装 Python

macOS 曾经在 10.8 至 12.3 版中预装了 Python 2.7。建议你从 Python 网站 安装最新版本的 Python 3。那里也提供了当前的 Python "universal2 binary" 编译版，它可以在 Mac 的新式 Apple Silicon 和旧式 Intel 处理器上原生运行。

在安装後你必須要做幾件事：

- 你的 `Applications` 文件夹中会有一个 `Python 3.11` 文件夹。你将在这里找到 IDLE，它是作为官方 Python 发行版标准组成部分的开发环境；以及 **`Python Launcher`**，它负责处理在 Finder 中双击 Python 脚本的操作。
- 一个框架 `/Library/Frameworks/Python.framework`，它包括 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 Python，你可以移除这三个项目。将有一个 Python 可执行文件的符号链接放在 `/usr/local/bin/` 中。

備🄵： 在 macOS 10.8-12.3 上，Apple 提供的 Python 版本将分别安装在 `/System/Library/Frameworks/Python.framework` 和 `/usr/bin/python`。你绝不应当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有两个不同但都可用的 Python 安装版，因此你的路径和用法与你想要执行的操作保持一致是非常重要的。

IDLE 包括一个 Help 菜单，允许你访问 Python 文件。如果你是完全的 Python 新手那么你应当开始阅读该文档中的教程部分。

如果你熟悉其他 Unix 平臺上的 Python，那🄵你應該🄵讀有關從 Unix shell 執行 Python 🄵本的部分。

5.1.1 如何執行 Python 🄵本

你在 macOS 上入门 Python 的最好方式是通过 IDLE 集成开发环境；请参阅整合化開發工具 *(IDE)* 章节并在 IDE 运行时使用 Help 菜单。

如果你想从 Terminal 窗口命令行或 Finder 中运行 Python 脚本那么你首先需要用一个编辑器来创建你的脚本。macOS 自带了多个标准的 Unix 命令行编辑器，其中包括 **vim** **nano**。如果你想要一个更具 Mac 风格的编辑器，那么来自 Bare Bones Software 的 **BBEdit** (参见 https://www.barebones.com/products/bbedit/index.html) 是一个很好的选择，还有 **TextMate** (参见 https://macromates.com)。其他的编辑器包括 **MacVim** (https://macvim.org) 和 **Aquamacs** (https://aquamacs.org)。

要從終端機視窗執行🄵本，你必須確保 `/usr/local/bin` 位於 shell 搜尋路徑中。

從 Finder 執行你的🄵本時，你有兩個選項：

- 将其拖拽到 **Python Launcher**。
- 选择 **Python Launcher** 作为通过 finder Info 窗口打开脚本（或任何 `.py` 脚本）和双击它时的默认应用程序。**Python Launcher** 有各种首选项来控制脚本的启动方式。选项拖拽允许你为单次发起调用更改这些首选项，或使用其 Preferences 菜单进行全局性的更改。

5.1.2 透過 GUI 執行🄵本

對於舊版本的 Python，你需要注意一個 macOS 的怪🄵之處：與 Aqua 視窗管理器溝通的程式（🄵而言之，任何具有 GUI（圖形化使用者介面）的程式）需要以特殊方式執行。使用 **pythonw** 而不是 **python** 來🄵動這樣的🄵本。

Python 3.9 上，你可以使用 **python** 或者 **pythonw**。

5.1.3 設定

macOS 上的 Python 遵循所有标准 Unix 环境变量例如 *PYTHONPATH*，但是为从 Finder 启动的程序设置这些变量是非标准的因为 Finder 在启动时不会读取你的 `.profile` 或 `.cshrc`。你需要创建一个文件 `~/.MacOSX/environment.plist`。详情参见 Apple 的 Technical Q&A QA1067。

有关安装 Python 包的更多信息，请参阅安裝額外的 *Python* 套件 一节。

5.2 整合化開發工具 (IDE)

Python 附带了标准的 IDLE 开发环境。可以在 https://www.hashcollision.org/hkn/python/idle_intro/index.html 找到一篇有关使用 IDLE 的介绍。

5.3 安裝額外的 Python 套件

本小节已移至 Python Packaging User Guide。

5.4 GUI 编程

於 Mac 上使用 Python 來建立 GUI 應用程式有許多選項。

PyObjC 是一個 Apple Objective-C/Cocoa 框架的 Python 🄵結 (binding)，這是大多數現代 Mac 開發的基礎。有關 PyObjC 的資訊，請見 https://pypi.org/project/pyobjc/。

標准的 Python GUI 工具套件是 `tkinter`，基於跨平臺的 Tk 工具套件 (https://www.tcl.tk)。Apple 的 OS X 包含了 Aqua 原生版本的 Tk，最新版本可以從 https://www.activestate.com 下載和安装；它也可以從原始碼開始建置。

有许多其他 macOS 的 GUI 工具包可供选择：

- PySide: Qt GUI 工具包 的官方 Python 绑定包。
- PyQt: Qt 的替代性 Python 绑定包。
- Kivy: 一款支持桌面和移动设备的跨平台 GUI 工具包。
- Toga: BeeWare 项目 的一部分；支持桌面、移动设备、Web 和控制台应用。
- wxPython: 一款支持桌面操作系统的跨平台工具包。

5.5 分发 Python 应用程序

有一系列工具可将你的 Python 代码转换为独立发布的应用程序：

- py2app: 支持基于 Python 项目创建 macOS `.app` 软件包。
- Briefcase: BeeWare 项目 的一部分；一款支持在 macOS 上创建 `.app` 软件包，并能管理签名和公证的跨平台打包工具。
- PyInstaller: 一款通过创建单独文件或目录作为可发布形式的跨平台打包工具。

5.6 其他資源

Pythonmac-SIG 邮件列表是 Mac 上 Python 用户和开发人员的优秀支持资源：

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

另一個好用資源是 MacPython wiki：

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

CHAPTER 6

編輯器與 IDE

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

術語表

>>>
互動式 shell 的預設 Python 提示字元。常見於能在直譯器中以互動方式被執行的程式碼範例。

...
可以表示：

- 在一個被縮排的程式碼區塊、在一對匹配的左右定界符（delimiter，例如括號、方括號、花括號或三引號）🄵部，或是在指定一個裝飾器 (decorator) 之後，要輸入程式碼時，互動式 shell 顯示的預設 Python 提示字元。
- 🄵建常數 `Ellipsis`。

2to3
一個試著將 Python 2.x 程式碼轉🄵🄵 Python 3.x 程式碼的工具，它是透過處理大部分的不相容性來達成此目的，而這些不相容性能🄵透過剖析原始碼和遍歷剖析樹而被檢測出來。

2to3 在可以標准函式庫中以 `lib2to3` 被使用；它提供了一個獨立的入口點，在 `Tools/scripts/2to3`。請參🄵 2to3-reference。

abstract base class（抽象基底類🄵）
抽象基底類🄵（又稱🄵 ABC）提供了一種定義介面的方法，作🄵*duck-typing*（鴨子型🄵）的補充。其他類似的技術，像是 `hasattr()`，則顯得笨拙或是帶有細微的錯誤（例如使用魔術方法 (magic method)）。ABC 🄵用🄵擬的 subclass（子類🄵），它們🄵不繼承自另一個 class（類🄵），但仍可被 `isinstance()` 及 `issubclass()` 辨識；請參🄵 `abc` 模組的🄵明文件。Python 有許多🄵建的 ABC，用於資料結構（在 `collections.abc` 模組）、數字（在 `numbers` 模組）、串流（在 `io` 模組）及 import 尋檢器和載入器（在 `importlib.abc` 模組）。你可以使用 `abc` 模組建立自己的 ABC。

annotation（🄵釋）
一個與變數、class 屬性、函式的參數或回傳值相關聯的標🄵。照慣例，它被用來作🄵*type hint*（型🄵提示）。

在執行環境 (runtime)，區域變數的🄵釋無法被存取，但全域變數、class 屬性和函式的🄵解，會分🄵被儲存在模組、class 和函式的 `__annotations__` 特殊屬性中。

請參🄵*variable annotation*、*function annotation*、**PEP 484** 和 **PEP 526**，這些章節皆有此功能的🄵明。關於🄵釋的最佳實踐方法也請參🄵 annotations-howto。

argument（引數）
呼叫函式時被傳遞給*function*（或*method*）的值。引數有兩種：

- 關鍵字引數 *(keyword argument)*：在函式呼叫中，以識🄵字（identifier，例如 `name=`）開頭的引數，或是以 `**` 後面 dictionary（字典）🄵的值被傳遞的引數。例如，`3` 和 `5` 都是以下 `complex()` 呼叫中的關鍵字引數：

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

- 位置引數 *(positional argument)*：不是關鍵字引數的引數。位置引數可在一個引數列表的起始處出現，和（或）作🄵 `*` 之後的*iterable*（可🄵代物件）中的元素被傳遞。例如，`3` 和 `5` 都是以下呼叫中的位置引數：

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

引數會被指定給函式主體中的附名區域變數。關於支配這個指定過程的規則，請參🄵 calls 章節。在語法上，任何運算式都可以被用來表示一個引數；其評估值會被指定給區域變數。

另請參🄵術語表的*parameter*（參數）條目、常見問題中的引數和參數之間的差🄵，以及 **PEP 362**。

asynchronous context manager（非同步情境管理器）
一個可以控制 `async with` 陳述式中所見環境的物件，而它是透過定義 `__aenter__()` 和 `__aexit__()` method（方法）來控制的。由 **PEP 492** 引入。

asynchronous generator（非同步🄵生器）
一個會回傳*asynchronous generator iterator*（非同步🄵生器🄵代器）的函式。它看起來像一個以 `async def` 定義的協程函式 (coroutine function)，但不同的是它包含了 `yield` 運算式，能生成一系列可用於 `async for` 🄵圈的值。

這個術語通常用來表示一個非同步🄵生器函式，但在某些情境中，也可能是表示非同步🄵生器🄵代器 *(asynchronous generator iterator)*。萬一想表達的意思不🄵清楚，那就使用完整的術語，以避免歧義。

一個非同步🄵生器函式可能包含 `await` 運算式，以及 `async for` 和 `async with` 陳述式。

asynchronous generator iterator（非同步🄵生器🄵代器）
一個由*asynchronous generator*（非同步🄵生器）函式所建立的物件。

這是一個*asynchronous iterator*（非同步🄵代器），當它以 `__anext__()` method 被呼叫時，會回傳一個可等待物件 (awaitable object)，該物件將執行非同步🄵生器的函式主體，直到遇到下一個 `yield` 運算式。

每個 `yield` 會暫停處理程序，🄵記住位置執行狀態（包括區域變數及擱置中的 try 陳述式）。當非同步🄵生器🄵代器以另一個被 `__anext__()` 回傳的可等待物件有效地回復時，它會從停止的地方繼續執行。請參🄵 **PEP 492** 和 **PEP 525**。

asynchronous iterable（非同步可🄵代物件）
一個物件，它可以在 `async for` 陳述式中被使用。必須從它的 `__aiter__()` method 回傳一個*asynchronous iterator*（非同步🄵代器）。由 **PEP 492** 引入。

asynchronous iterator（非同步🄵代器）
一個實作 `__aiter__()` 和 `__anext__()` method 的物件。`__anext__()` 必須回傳一個*awaitable*（可等待物件）。`async for` 會解析非同步🄵代器的 `__anext__()` method 所回傳的可等待物件，直到它引發 `StopAsyncIteration` 例外。由 **PEP 492** 引入。

attribute（屬性）
一個與某物件相關聯的值，該值大多能透過使用點分隔運算式 (dotted expression) 的名稱被參照。例如，如果物件 *o* 有一個屬性 *a*，則該屬性能以 *o.a* 被參照。

如果一個物件允許，給予該物件一個名稱不是由 identifiers 所定義之識🄴符 (identifier) 的屬性是有可能的，例如使用 `setattr()`。像這樣的屬性將無法使用點分隔運算式來存取，而是需要使用 `getattr()` 來取得它。

awaitable（可等待物件）
一個可以在 `await` 運算式中被使用的物件。它可以是一個*coroutine*（協程），或是一個有 `__await__()` method 的物件。另請參🄴 **PEP 492**。

BDFL
Benevolent Dictator For Life（終身仁慈獨裁者），又名 Guido van Rossum，Python 的創造者。

binary file（二進制檔案）
file object 能够读写字节型对象。二进制文件的例子包括以二进制模式 (`'rb'`, `'wb'` 或 `'rb+'`) 打开的文件、`sys.stdin.buffer`、`sys.stdout.buffer` 以及 `io.BytesIO` 和 `gzip.GzipFile` 的实例。

另請參🄵*text file*（文字檔案），它是一個能🄵讀取和寫入 `str` 物件的檔案物件。

borrowed reference（借用參照）
在 Python 的 C API 中，借用引用是指一种对象引用，使用该对象的代码并不持有该引用。如果对象被销毁则它就会变成一个悬空指针。例如，垃圾回收器可以移除对象的最后一个*strong reference* 来销毁它。

對*borrowed reference* 呼叫 `Py_INCREF()` 以將它原地 (in-place) 轉🄵🄵*strong reference* 是被建議的做法，除非該物件不能在最後一次使用借用參照之前被銷🄵。`Py_NewRef()` 函式可用於建立一個新的*strong reference*。

bytes-like object（類位元組串物件）
一個支援 bufferobjects 且能🄵匯出 C-*contiguous* 緩衝區的物件。這包括所有的 `bytes`、`bytearray` 和 `array.array` 物件，以及許多常見的 `memoryview` 物件。類位元組串物件可用於處理二進制資料的各種運算；這些運算包括壓縮、儲存至二進制檔案和透過 socket（插座）發送。

有些運算需要二進制資料是可變的。🄵明文件通常會將這些物件稱🄵「可讀寫的類位元組串物件」。可變緩衝區的物件包括 `bytearray`，以及 `bytearray` 的 `memoryview`。其他的運算需要讓二進制資料被儲存在不可變物件（「唯讀的類位元組串物件」）中；這些物件包括 `bytes`，以及 `bytes` 物件的 `memoryview`。

bytecode（位元組碼）
Python 的原始碼會被編譯成位元組碼，它是 Python 程式在 CPython 直譯器中的🄴部表示法。該位元組碼也會被暫存在 `.pyc` 檔案中，以便第二次執行同一個檔案時能🄵更快速（可以不用從原始碼重新編譯🄵位元組碼）。這種「中間語言 (intermediate language)」據🄵是運行在一個*virtual machine*（🄵擬機器）上，該🄵擬機器會執行與每個位元組碼對應的機器碼 (machine code)。要注意的是，位元組碼理論上是無法在不同的 Python 🄵擬機器之間運作的，也不能在不同版本的 Python 之間保持穩定。

位元組碼的指令列表可以在 dis 模組的🄵明文件中找到。

callable（可呼叫物件）
一個 callable 是可以被呼叫的物件，呼叫時可能以下列形式帶有一組引數（請見*argument*）：

```
callable(argument1, argument2, argumentN)
```

一個*function* 與其延伸的*method* 都是 callable。一個有實作 `__call__()` 方法的 class 之實例也是個 callable。

callback（回呼）
作🄵引數被傳遞的一個副程式 (subroutine) 函式，會在未來的某個時間點被執行。

class（類🄵）
一個用於建立使用者定義物件的模板。Class 的定義通常會包含 method 的定義，這些 method 可以在 class 的實例上進行操作。

class variable（類🄵變數）
一個在 class 中被定義，且應該只能在 class 層次（意即不是在 class 的實例中）被修改的變數。

complex number（🄵數）
一個我們熟悉的實數系統的擴充，在此所有數字都會被表示🄵一個實部和一個🄵部之和。🄵數就是🄵數單位（-1 的平方根）的實數倍，此單位通常在數學中被寫🄵 i，在工程學中被寫🄵 j。Python 🄵建了對🄵數的支援，它是用後者的記法來表示🄵數；🄵部會帶著一個後綴的 j 被編寫，例如 3+1j。若要將 math 模組🄵的工具等效地用於🄵數，請使用 cmath 模組。🄵數的使用是一個相當進階的數學功能。如果你🄵有察覺到對它們的需求，那🄵幾乎能確定你可以安全地忽略它們。

context manager（情境管理器）
在 with 语句中通过定义 __enter__() 和 __exit__() 方法来控制环境状态的对象。参见 **PEP 343**。

context variable（情境變數）
一個變數，其值可以根據上下文的情境而有所不同。這類似執行緒區域儲存區 (Thread-Local Storage)，在其中，一個變數在每個執行緒可能具有不同的值。然而，關於情境變數，在一個執行緒中可能會有多個情境，而情境變數的主要用途，是在🄵行的非同步任務 (concurrent asynchronous task) 中，對於變數狀態的追🄵。請參🄵 contextvars。

contiguous（連續的）
如果一個緩衝區是 *C-contiguous* 或是 *Fortran contiguous*，則它會確切地被視🄵是連續的。零維 (zero-dimensional) 的緩衝區都是 C 及 Fortran contiguous。在一維 (one-dimensional) 陣列中，各項目必須在記憶體中彼此相鄰地排列，而其索引順序是從零開始遞增。在多維的 (multidimensional) C-contiguous 陣列中，按記憶體位址的順序訪問各個項目時，最後一個索引的變化最快。然而，在 Fortran contiguous 陣列中，第一個索引的變化最快。

coroutine（協程）
協程是副程式 (subroutine) 的一種更🄵廣義的形式。副程式是在某個時間點被進入🄵在另一個時間點被退出。協程可以在許多不同的時間點被進入、退出和回復。它們能🄵以 async def 陳述式被實作。另請參🄵 **PEP 492**。

coroutine function（協程函式）
一個回傳*coroutine*（協程）物件的函式。一個協程函式能以 async def 陳述式被定義，🄵可能會包含 await、async for 和 async with 關鍵字。這些關鍵字由 **PEP 492** 引入。

CPython
Python 程式語言的標準實作 (canonical implementation)，被發布在 python.org 上。「CPython」這個術語在必要時被使用，以區分此實作與其它語言的實作，例如 Jython 或 IronPython。

decorator（裝飾器）
一個函式，它會回傳另一個函式，通常它會使用 @wrapper 語法，被應用🄵一種函式的變🄵 (function transformation)。裝飾器的常見範例是 classmethod() 和 staticmethod()。

裝飾器語法只是語法糖。以下兩個函式定義在語義上是等效的：

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

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

Class 也存在相同的概念，但在那🄵比較不常用。關於裝飾器的更多🄵容，請參🄵函式定義和 class 定義的🄵明文件。

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* 类的字典中查找名称为 *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、特征属性、类方法、静态方法以及对超类的引用等等。

關於描述器 method 的更多資訊，請參🄵 descriptors 或描述器使用指南。

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 中称为 hash。

dictionary comprehension（字典綜合運算）
一種緊密的方法，用來處理一個可🄵代物件中的全部或部分元素，🄵將處理結果以一個字典回傳。`results = {n: n ** 2 for n in range(10)}` 會🄵生一個字典，它包含了鍵 `n` 映射到值 `n ** 2`。請參🄵 comprehensions。

dictionary view（字典檢視）
從 `dict.keys()`、`dict.values()` 及 `dict.items()` 回傳的物件被稱🄵字典檢視。它們提供了字典中項目的動態檢視，這表示當字典有變動時，該檢視會反映這些變動。若要🄵制將字典檢視轉🄵完整的 list（串列），須使用 `list(dictview)`。請參🄵 dict-views。

docstring（🄵明字串）
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。它在代码被执行时会被忽略，但会被编译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，因此是存放对象的文档的规范位置。

duck-typing（鴨子型🄵）
一種程式設計風格，它不是藉由檢查一個物件的型🄵來確定它是否具有正確的介面；取而代之的是，method 或屬性會單純地被呼叫或使用。（「如果它看起來像一🄵鴨子而且叫起來像一🄵鴨子，那🄵它一定是一🄵鴨子。」）因🄵🄵調介面而非特定型🄵，精心設計的程式碼能讓多形替代 (polymorphic substitution) 來增進它的靈活性。鴨子型🄵要避免使用 `type()` 或 `isinstance()` 進行測試。（但是請注意，鴨子型🄵可以用抽象基底類🄵 *(abstract base class)* 來補充。）然而，它通常會🄵用 `hasattr()` 測試，或是*EAFP* 程式設計風格。

EAFP
Easier to ask for forgiveness than permission.（請求寬恕比請求許可更容易。）這種常見的 Python 編碼風格會先假設有效的鍵或屬性的存在，🄵在該假設被推翻時再捕獲例外。這種乾🄵且快速的風格，其特色是存在許多的 `try` 和 `except` 陳述式。該技術與許多其他語言（例如 C）常見的*LBYL* 風格形成了對比。

expression（運算式）
一段可以被評估🄵求值的語法。🄵句話🄵，一個運算式就是文字、名稱、屬性存取、運算子或函式呼叫等運算式元件的累積，而這些元件都能回傳一個值。與許多其他語言不同的是，🄵非所有的 Python 語言構造都是運算式。另外有一些*statement*（陳述式）不能被用作運算式，例如 `while`。賦值 (assignment) 也是陳述式，而不是運算式。

extension module（擴充模組）
一個以 C 或 C++ 編寫的模組，它使用 Python 的 C API 來與核心及使用者程式碼進行互動。

f-string（f 字串）
以 `'f'` 或 `'F'` 🄵前綴的字串文本通常被稱🄵「f 字串」，它是格式化的字串文本的縮寫。另請參🄵 **PEP 498**。

file object（檔案物件）
对外公开面向文件的 API（带有 `read()` 或 `write()` 等方法）以使用下层资源的对象。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、其他类型的存储或通信设备的访问（例如标准输入/输出、内存缓冲区、套接字、管道等）。文件对象也被称为 文件型对象或 流。

實際上，有三種檔案物件：原始的二進制檔案、緩衝的二進制檔案和文字檔案。它們的介面在 `io` 模組中被定義。建立檔案物件的標准方法是使用 `open()` 函式。

file-like object（類檔案物件）
file object（檔案物件）的同義字。

filesystem encoding and error handler（檔案系統編碼和錯誤處理函式）
Python 所使用的一種編碼和錯誤處理函式，用來解碼來自作業系統的位元組，以及將 Unicode 編碼到

作業系統。

檔案系統編碼必須保證能成功解碼所有小於 128 的位元組。如果檔案系統編碼無法提供此保證，則 API 函式會引發 `UnicodeError`。

`sys.getfilesystemencoding()` 和 `sys.getfilesystemencodeerrors()` 函式可用於取得檔案系統編碼和錯誤處理函式。

filesystem encoding and error handler（檔案系統編碼和錯誤處理函式）會在 Python 🄵動時由 `PyConfig_Read()` 函式來配置：請參🄵 `filesystem_encoding`，以及 `PyConfig` 的成員 `filesystem_errors`。

另請參🄵*locale encoding*（區域編碼）。

finder（尋檢器）
一個物件，它會嘗試🄵正在被 import 的模組尋找*loader*（載入器）。

從 Python 3.3 開始，有兩種類型的尋檢器：*元路徑尋檢器 (meta path finder)* 會使用 `sys.meta_path`，而*路徑項目尋檢器 (path entry finder)* 會使用 `sys.path_hooks`。

請參🄵 **PEP 302**、**PEP 420** 和 **PEP 451** 以了解更多細節。

floor division（向下取整除法）
向下無條件舍去到最接近整數的數學除法。向下取整除法的運算子是 `//`。例如，運算式 `11 // 4` 的計算結果🄵 `2`，與 float（浮點數）真除法所回傳的 `2.75` 不同。請注意，`(-11) // 4` 的結果是 `-3`，因🄵是 `-2.75` 被向下無條件舍去。請參🄵 **PEP 238**。

function（函式）
一連串的陳述式，它能🄵向呼叫者回傳一些值。它也可以被傳遞零個或多個*引數*，這些引數可被使用於函式本體的執行。另請參🄵*parameter*（參數）、*method*（方法），以及 function 章節。

function annotation（函式🄵釋）
函式參數或回傳值的一個*annotation*（🄵釋）。

函式🄵釋通常被使用於*型🄵提示*：例如，這個函式預期會得到兩個 `int` 引數，🄵會有一個 `int` 回傳值：

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

函式🄵釋的語法在 function 章節有詳細解釋。

請參🄵*variable annotation* 和 **PEP 484**，皆有此功能的描述。關於🄵釋的最佳實踐方法，另請參🄵 annotations-howto。

__future__
future 陳述式：`from __future__ import <feature>`，會指示編譯器使用那些在 Python 未來的發布版本中將成🄵標准的語法或語義，來編譯當前的模組。而 `__future__` 模組則記🄵了 *feature*（功能）可能的值。透過 import 此模組🄵對其變數求值，你可以看見一個新的功能是何時首次被新增到此語言中，以及它何時將會（或已經）成🄵預設的功能：

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

garbage collection（垃圾回收）
當記憶體不再被使用時，將其釋放的過程。Python 執行垃圾回收，是透過參照計數 (reference counting)，以及一個能🄵檢測和中斷參照循環 (reference cycle) 的循環垃圾回收器 (cyclic garbage collector) 來完成。垃圾回收器可以使用 `gc` 模組對其進行控制。

generator（🄵生器）
一個會回傳*generator iterator*（🄵生器🄵代器）的函式。它看起來像一個正常的函式，但不同的是它包含

了 yield 運算式，能🄵生一系列的值，這些值可用於 for 🄵圈，或是以 next() 函式，每次檢索其中的一個值。

這個術語通常用來表示一個🄵生器函式，但在某些情境中，也可能是表示🄵生器🄵代器。萬一想表達的意思不🄵清楚，那就使用完整的術語，以避免歧義。

generator iterator（🄵生器🄵代器）
一個由*generator*（🄵生器）函式所建立的物件。

每個 yield 會暫停處理程序，🄵記住位置執行狀態（包括區域變數及擱置中的 try 陳述式）。當🄵生器🄵代器回復時，它會從停止的地方繼續執行（與那些每次調用時都要重新開始的函式有所不同）。

generator expression（🄵生器運算式）
一個會回傳🄵代器的運算式。它看起來像一個正常的運算式，後面接著一個 for 子句，該子句定義了🄵圈變數、範圍以及一個選擇性的 if 子句。該組合運算式會🄵外層函式🄵生多個值：

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

generic function（泛型函式）
一個由多個函式組成的函式，該函式會對不同的型🄵實作相同的運算。呼叫期間應該使用哪種實作，是由調度演算法 (dispatch algorithm) 來🄵定。

另請參🄵*single dispatch*（單一調度）術語表條目、functools.singledispatch() 裝飾器和 **PEP 443**。

generic type（泛型型🄵）
一個能🄵被參數化 (parameterized) 的*type*（型🄵）；通常是一個 容器型🄵，像是 list 和 dict。它被用於型🄵提示和🄵釋。

詳情請參🄵泛型🄵名型🄵、**PEP 483**、**PEP 484**、**PEP 585** 和 typing 模組。

GIL
請參🄵*global interpreter lock*（全域直譯器鎖）。

global interpreter lock（全域直譯器鎖）
CPython 直譯器所使用的機制，用以確保每次都只有一個執行緒能執行 Python 的*bytecode*（位元組碼）。透過使物件模型（包括關鍵的🄵建型🄵，如 dict）自動地避免🄵行存取 (concurrent access) 的危險，此機制可以簡化 CPython 的實作。鎖定整個直譯器，會使直譯器更容易成🄵多執行緒 (multi-threaded)，但代價是會犧牲掉多處理器的機器能🄵提供的一大部分平行性 (parallelism)。

然而，有些擴充模組，無論是標準的或是第三方的，它們被設計成在執行壓縮或雜🄵等計算密集 (computationally intensive) 的任務時，可以解除 GIL。另外，在執行 I/O 時，GIL 總是會被解除。

過去對於建立「無限制執行緒」直譯器（以更高的精細度鎖定共享資料的直譯器）的努力🄵未成功，因🄵在一般的單一處理器情🄵下，效能會有所損失。一般認🄵，若要克服這個效能問題，會使實作變得🄵雜許多，進而付出更高的維護成本。

hash-based pyc（雜🄵架構的 pyc）
一個位元組碼 (bytecode) 暫存檔，它使用雜🄵值而不是對應原始檔案的最後修改時間，來確定其有效性。請參🄵 pyc-invalidation。

hashable（可雜🄵的）
一个对象如果具有在其生命期内绝不改变的哈希值（它需要有 __hash__() 方法），并可以同其他对象进行比较（它需要有 __eq__() 方法）就被称为 可哈希对象。可哈希对象必须具有相同的哈希值比较结果才会相等。

可雜🄵性 (hashability) 使一個物件可用作 dictionary（字典）的鍵和 set（集合）的成員，因🄵這些資料結構都在其🄵部使用了雜🄵值。

大多數的 Python 不可變🄵建物件都是可雜🄵的；可變的容器（例如 list 或 dictionary）🄵不是；而不可變的容器（例如 tuple（元組）和 frozenset），只有當它們的元素是可雜🄵的，它們本身才是可雜🄵的。

若物件是使用者自定 class 的實例，則這些物件會被預設🄴可雜🄴的。它們在互相比較時都是不相等的（除非它們與自己比較），而它們的雜🄴值則是衍生自它們的 `id()`。

IDLE
Python 的 Integrated Development and Learning Environment（整合開發與學習環境）。idle 是一個基本的編輯器和直譯器環境，它和 Python 的標准發行版本一起被提供。

immutable（不可變物件）
一個具有固定值的物件。不可變物件包括數字、字串和 tuple（元組）。這類物件是不能被改變的。如果一個不同的值必須被儲存，則必須建立一個新的物件。它們在需要🄴定雜🄴值的地方，扮演重要的角色，例如 dictionary（字典）中的一個鍵。

import path（引入路徑）
一個位置（或路徑項目）的列表，而那些位置就是在 import 模組時，會被*path based finder*（基於路徑的尋檢器）搜尋模組的位置。在 import 期間，此位置列表通常是來自 `sys.path`，但對於子套件 (subpackage) 而言，它也可能是來自父套件的 `__path__` 屬性。

importing（引入）
一個過程。一個模組中的 Python 程式碼可以透過此過程，被另一個模組中的 Python 程式碼使用。

importer（引入器）
一個能🄵尋找及載入模組的物件；它既是*finder*（尋檢器）也是*loader*（載入器）物件。

interactive（互動的）
Python 有一個互動式直譯器，這表示你可以在直譯器的提示字元輸入陳述式和運算式，立即執行它們🄵且看到它們的結果。只要🄴動 `python`，不需要任何引數（可能藉由從你的電腦的主選單選擇它）。這是測試新想法或檢查模塊和包的非常🄵大的方法（請記住 help(x)）。

interpreted（直譯的）
Python 是一種直譯語言，而不是編譯語言，不過這個區分可能有些模糊，因🄵有位元組碼 (bytecode) 編譯器的存在。這表示原始檔案可以直接被運行，而不需明確地建立另一個執行檔，然後再執行它。直譯語言通常比編譯語言有更短的開發 / 除錯🄵期，不過它們的程式通常也運行得較慢。另請參🄵*interactive*（互動的）。

interpreter shutdown（直譯器關閉）
當 Python 直譯器被要求關閉時，它會進入一個特殊階段，在此它逐漸釋放所有被配置的資源，例如模組和各種關鍵🄴部結構。它也會多次呼叫垃圾回收器 *(garbage collector)*。這能🄵觸發使用者自定的解構函式 (destructor) 或弱引用的回呼 (weakref callback)，🄵執行其中的程式碼。在關閉階段被執行的程式碼會遇到各種例外，因🄵它所依賴的資源可能不再有作用了（常見的例子是函式庫模組或是警告機制）。

直譯器關閉的主要原因，是 `__main__` 模組或正被運行的🄵本已經執行完成。

iterable（可🄴代物件）
一种能够逐个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（如 `list`, `str` 和 `tuple` 等）以及某些非序列类型如 `dict`, 文件对象 以及任何定义了 `__iter__()` 方法或实现了*sequence* 语义的 `__getitem__()` 方法的自定义类的对象。

可迭代对象可被用于 `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被传给内置函数 `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` 语句会自动为你处理那些操作，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* 和*generator*。

iterator（🄴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`。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`) 在你每次将其传入 `iter()` 函数或是在 `for` 循环中使用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。

在 typeiter 文中可以找到更多資訊。

CPython 實作細節： CPython 没有统一应用迭代器定义 `__iter__()` 的要求。

key function（鍵函式）
鍵函式或理序函式 (collation function) 是一個可呼叫 (callable) 函式，它會回傳一個用於排序 (sorting) 或定序 (ordering) 的值。例如，`locale.strxfrm()` 被用來🄵生一個了解區域特定排序慣例的排序鍵。

Python 中的許多工具，都接受以鍵函式來控制元素被定序或分組的方式。它們包括 `min()`、`max()`、`sorted()`、`list.sort()`、`heapq.merge()`、`heapq.nsmallest()`、`heapq.nlargest()` 和 `itertools.groupby()`。

有幾種方法可以建立一個鍵函式。例如，`str.lower()` method 可以作🄵不分大小寫排序的鍵函式。或者，一個鍵函式也可以從 `lambda` 運算式被建造，例如 `lambda r: (r[0], r[2])`。另外，`operator.attrgetter()`、`operator.itemgetter()` 和 `operator.methodcaller()` 🄵三個鍵函式的建構函式 (constructor)。關於如何建立和使用鍵函式的範例，請參🄵如何排序。

keyword argument（關鍵字引數）
請參🄵*argument*（引數）。

lambda
由單一*expression*（運算式）所組成的一個匿名行🄵函式 (inline function)，於該函式被呼叫時求值。建立 lambda 函式的語法是 `lambda [parameters]: expression`

LBYL
Look before you leap.（三思而後行。）這種編碼風格會在進行呼叫或查找之前，明確地測試先🄵條件。這種風格與*EAFP* 方式形成對比，且它的特色是會有許多 `if` 陳述式的存在。

在一個多執行緒環境中，LBYL 方式有在「三思」和「後行」之間引入了競🄵條件 (race condition) 的風險。例如以下程式碼 `if key in mapping: return mapping[key]`，如果另一個執行緒在測試之後但在查找之前，從 *mapping* 中移除了 *key*，則該程式碼就會失效。這個問題可以用鎖 (lock) 或使用 EAFP 編碼方式來解🄵。

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

list comprehension（串列綜合運算）
一種用來處理一個序列中的全部或部分元素，🄵將處理結果以一個 list 回傳的簡要方法。`result = ['{:#04x}'.format(x) for x in range(256) if x % 2 == 0]` 會🄵生一個字串 list，其中包含 0 到 255 範圍🄵，所有偶數的十六進位數 (0x..)。`if` 子句是選擇性的。如果省略它，則 `range(256)` 中的所有元素都會被處理。

loader（載入器）
一個能🄵載入模組的物件。它必須定義一個名🄵 `load_module()` 的 method（方法）。載入器通常是被*finder*（尋檢器）回傳。更多細節請參🄵 **PEP 302**，關於*abstract base class*（抽象基底類🄵），請參🄵 `importlib.abc.Loader`。

locale encoding（區域編碼）
在 Unix 上，它是 LC_CTYPE 區域設定的編碼。它可以用 `locale.setlocale(locale.LC_CTYPE, new_locale)` 來設定。

在 Windows 上，它是 ANSI 代碼頁（code page，例如 `"cp1252"`）。

在 Android 和 VxWorks 上，Python 使用 `"utf-8"` 作🄵區域編碼。

`locale.getencoding()` 可被用来获取语言区域编码格式。

也請參考*filesystem encoding and error handler*。

magic method（魔術方法）
special method（特殊方法）的一個非正式同義詞。

mapping（對映）
一個容器物件，它支援任意鍵的查找，且能實作 abstract base classes（抽象基底類🄵）中，`collections.abc.Mapping` 或 `collections.abc.MutableMapping` 所指定的 method。範例包括 `dict`、`collections.defaultdict`、`collections.OrderedDict` 和 `collections.Counter`。

meta path finder（元路徑尋檢器）
一種經由搜尋 `sys.meta_path` 而回傳的*finder*（尋檢器）。元路徑尋檢器與*路徑項目尋檢器 (path entry finder)* 相關但是不同。

關於元路徑尋檢器實作的 method，請參🄵 `importlib.abc.MetaPathFinder`。

metaclass（元類🄵）
一種 class 的 class。Class 定義過程會建立一個 class 名稱、一個 class dictionary（字典），以及一個 base class（基底類🄵）的列表。Metaclass 負責接受這三個引數，🄵建立該 class。大多數的物件導向程式語言會提供一個預設的實作。Python 的特🄵之處在於它能🄵建立自訂的 metaclass。大部分的使用者從未需要此工具，但是當需要時，metaclass 可以提供🄵大且優雅的解🄵方案。它們已被用於記🄵屬性存取、增加執行緒安全性、追🄵物件建立、實作單例模式 (singleton)，以及許多其他的任務。

更多資訊可以在 metaclasses 章節中找到。

method（方法）
一個在 class 本體🄵被定義的函式。如果 method 作🄵其 class 實例的一個屬性被呼叫，則它將會得到該實例物件成🄵它的第一個*argument*（引數）（此引數通常被稱🄵 `self`）。請參🄵*function*（函式）和*nested scope*（巢狀作用域）。

method resolution order（方法解析順序）
方法解析順序是在查找某個成員的過程中，base class（基底類🄵）被搜尋的順序。關於第 2.3 版至今，Python 直譯器所使用的演算法細節，請參🄵 Python 2.3 版方法解析順序。

module（模組）
一個擔任 Python 程式碼的組織單位 (organizational unit) 的物件。模組有一個命名空間，它包含任意的 Python 物件。模組是藉由*importing* 的過程，被載入至 Python。

另請參🄵*package*（套件）。

module spec（模組規格）
一個命名空間，它包含用於載入模組的 import 相關資訊。它是 `importlib.machinery.ModuleSpec` 的一個實例。

MRO
請參🄵*method resolution order*（方法解析順序）。

mutable（可變物件）
可變物件可以改變它們的值，但維持它們的 `id()`。另請參🄵*immutable*（不可變物件）。

named tuple（附名元組）
術語「named tuple（附名元組）」是指從 tuple 繼承的任何型🄵或 class，且它的可索引 (indexable) 元素也可以用附名屬性來存取。這些型🄵或 class 也可以具有其他的特性。

有些🄵建型🄵是 named tuple，包括由 `time.localtime()` 和 `os.stat()` 回傳的值。另一個例子是 `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（比如上面的例子）。此外，具名元组还可通过常规类定义从 `tuple` 继承并定义指定名称的字段的方式来创建。这样的类可以手工编号，或者也可以通过继承 `typing.NamedTuple`，或者使用工厂函数 `collections.namedtuple()` 来创建。后一种方式还会添加一些手工编写或内置的具名元组所没有的额外方法。

namespace（命名空間）
變數被儲存的地方。命名空間是以 dictionary（字典）被實作。有區域的、全域的及🄵建的命名空間，而在物件中（在 method 中）也有巢狀的命名空間。命名空間藉由防止命名衝突，來支援模組化。例如，函式 `builtins.open` 和 `os.open()` 是透過它們的命名空間來區分彼此。命名空間也藉由明確地區分是哪個模組在實作一個函式，來增進可讀性及可維護性。例如，寫出 `random.seed()` 或 `itertools.islice()` 明確地表示，這些函式分🄵是由 `random` 和 `itertools` 模組在實作。

namespace package（命名空間套件）
一個 **PEP 420** *package*（套件），它只能作🄵子套件 (subpackage) 的一個容器。命名空間套件可能🄵有實體的表示法，而且具體來🄵它們不像是一個*regular package*（正規套件），因🄵它們🄵🄵有 `__init__.py` 這個檔案。

另請參🄵*module*（模組）。

nested scope（巢狀作用域）
能🄴參照外層定義 (enclosing definition) 中的變數的能力。舉例來🄵，一個函式如果是在另一個函式中被定義，則它便能🄴參照外層函式中的變數。請注意，在預設情🄵下，巢狀作用域僅適用於參照，而無法用於賦值。區域變數能在最🄵層作用域中讀取及寫入。同樣地，全域變數是在全域命名空間中讀取及寫入。`nonlocal` 容許對外層作用域進行寫入。

new-style class（新式類🄵）
对目前已被应用于所有类对象的类形式的旧称谓。在较早的 Python 版本中，只有新式类能够使用 Python 新增的更灵活我，如 `__slots__`、描述器、特征属性、`__getattribute__()`、类方法和静态方法等。

object（物件）
具有狀態（屬性或值）及被定義的行🄵（method）的任何資料。它也是任何*new-style class*（新式類🄵）的最終 base class（基底類🄵）。

package（套件）
一個 Python 的*module*（模組），它可以包含子模組 (submodule) 或是遞🄴的子套件 (subpackage)。技術上而言，套件就是具有 `__path__` 屬性的一個 Python 模組。

另請參🄵*regular package*（正規套件）和*namespace package*（命名空間套件）。

parameter（參數）
在*function*（函式）或 method 定義中的一個命名實體 (named entity)，它指明該函式能🄴接受的一個*argument*（引數），或在某些情🄵下指示多個引數。共有有五種不同的參數類型：

- *positional-or-keyword*（位置或關鍵字）：指明一個可以按照位置或是作🄵關鍵字引數被傳遞的引數。這是參數的預設類型，例如以下的 *foo* 和 *bar*：

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

- *positional-only*（僅限位置）：指明一個只能按照位置被提供的引數。在函式定義的參數列表中包含一個 `/` 字元，就可以在該字元前面定義僅限位置參數，例如以下的 *posonly1* 和 *posonly2*：

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

- *keyword-only*（僅限關鍵字）：指明一個只能以關鍵字被提供的引數。在函式定義的參數列表中，包含一個任意數量位置參數 (var-positional parameter) 或是單純的 `*` 字元，就可以在其後方定義僅限關鍵字參數，例如以下的 *kw_only1* 和 *kw_only2*：

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

- *var-positional*（任意數量位置）：指明一串能以任意序列被提供的位置引數（在已被其他參數接受的任何位置引數之外）。這類參數是透過在其參數名稱字首加上 * 來定義的，例如以下的 *args*：

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

- *var-keyword*（任意數量關鍵字）：指明可被提供的任意數量關鍵字引數（在已被其他參數接受的任何關鍵字引數之外）。這類參數是透過在其參數名稱字首加上 ** 來定義的，例如上面範例中的 *kwargs*。

參數可以指明引數是選擇性的或必需的，也可以🄵一些選擇性的引數指定預設值。

另請參🄵術語表的*argument*（引數）條目、常見問題中的引數和參數之間的差🄵、`inspect.Parameter` class、function 章節，以及 **PEP 362**。

path entry（路徑項目）
在*import path*（引入路徑）中的一個位置，而*path based finder*（基於路徑的尋檢器）會參考該位置來尋找要 import 的模組。

path entry finder（路徑項目尋檢器）
被 `sys.path_hooks` 中的一個可呼叫物件 (callable)（意即一個*path entry hook*）所回傳的一種*finder*，它知道如何以一個*path entry*定位模組。

關於路徑項目尋檢器實作的 method，請參🄵 `importlib.abc.PathEntryFinder`。

path entry hook（路徑項目🄵）
一种可调用对象，它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_hooks` 列表返回一个*path entry finder*。

path based finder（基於路徑的尋檢器）
預設的*元路徑尋檢器 (meta path finder)* 之一，它會在一個*import path* 中搜尋模組。

path-like object（類路徑物件）
一個表示檔案系統路徑的物件。類路徑物件可以是一個表示路徑的 `str` 或 `bytes` 物件，或是一個實作 `os.PathLike` 協定的物件。透過呼叫 `os.fspath()` 函式，一個支援 `os.PathLike` 協定的物件可以被轉🄵🄵 `str` 或 `bytes` 檔案系統路徑；而 `os.fsdecode()` 及 `os.fsencode()` 則分🄵可用於確保 `str` 及 `bytes` 的結果。由 **PEP 519** 引入。

PEP
Python Enhancement Proposal（Python 增🄴提案）。PEP 是一份設計🄵明文件，它能🄵 Python 社群提供資訊，或是描述 Python 的一個新功能或該功能的程序和環境。PEP 應該要提供簡潔的技術規範以及被提案功能的運作原理。

PEP 的存在目的，是要成🄵重大新功能的提案、社群中關於某個問題的意見收集，以及已納入 Python 的設計🄵策的記🄵，這些過程的主要機制。PEP 的作者要負責在社群🄵建立共識🄵記🄵反對意見。

請參🄵 **PEP 1**。

portion（部分）
在單一目🄵中的一組檔案（也可能儲存在一個 zip 檔中），這些檔案能對一個命名空間套件 (namespace package) 有所貢獻，如同 **PEP 420** 中的定義。

positional argument（位置引數）
請參🄵*argument*（引數）。

provisional API（暫行 API）
暫行 API 是指，從標准函式庫的向後相容性 (backwards compatibility) 保證中，故意被排除的 API。雖然此類介面，只要它們被標示🄵暫行的，理論上🄵不會有重大的變更，但如果核心開發人員認🄵有必要，也可能會出現向後不相容的變更（甚至包括移除該介面）。這種變更🄵不會無端地🄵生——只有 API 被納入之前未察覺的嚴重基本缺陷被揭露時，它們才會發生。

即使對於暫行 API，向後不相容的變更也會被視🄵「最後的解🄵方案」——對於任何被發現的問題，仍然會盡可能找出一個向後相容的解🄵方案。

這個過程使得標準函式庫能隨著時間不斷進化，而避免耗費過長的時間去鎖定有問題的設計錯誤。請參🄵 **PEP 411** 了解更多細節。

provisional package（暫行套件）
請參🄵*provisional API*（暫行 API）。

Python 3000
Python 3.x 系列版本的🄵稱（很久以前創造的，當時第 3 版的發布是在🄵遠的未來。）也可以縮寫🄵「Py3k」。

Pythonic（Python 風格的）
一個想法或一段程式碼，它應用了 Python 語言最常見的慣用語，而不是使用其他語言常見的概念來實作程式碼。例如，Python 中常見的一種習慣用法，是使用一個 `for` 陳述式，對一個可🄵代物件的所有元素進行🄵圈。許多其他語言🄵🄵有這種類型的架構，所以不熟悉 Python 的人有時會使用一個數值計數器來代替：

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

相較之下，以下方法更簡潔、更具有 Python 風格：

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

qualified name（限定名稱）
一個「點分隔名稱」，它顯示從一個模組的全域作用域到該模組中定義的 class、函式或 method 的「路徑」，如 **PEP 3155** 中的定義。對於頂層的函式和 class 而言，限定名稱與其物件名稱相同：

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

當用於引用模組時，*完全限定名稱 (fully qualified name)* 是表示該模組的完整點分隔路徑，包括任何的父套件，例如 `email.mime.text`：

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

reference count（參照計數）
對於一個物件的參照次數。當一個物件的參照計數下降到零時，它會被解除配置 (deallocated)。參照計數通常在 Python 程式碼中看不到，但它🄵是*CPython* 實作的一個關鍵元素。程式設計師可以呼叫 `getrefcount()` 函式來回傳一個特定物件的參照計數。

regular package（正規套件）
一個傳統的*package*（套件），例如一個包含 `__init__.py` 檔案的目🄵。

另請參🄵*namespace package*（命名空間套件）。

__slots__
在 class 🄵部的一個宣告，它藉由預先宣告實例屬性的空間，以及消除實例 dictionary（字典），來節省記憶

體。雖然該技術很普遍，但它有點難以正確地使用，最好保留給那種在一個記憶體關鍵 (memory-critical) 的應用程式中存在大量實例的罕見情🄴。

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` 等。请注意虽然 `dict` 也支持 `__getitem__()` 和 `__len__()`，但它被归类为映射而非序列，因为它使用任意*immutable* 键而不是整数来查找元素。

`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，还添加了 `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。要获取有关通用序列方法的更多文档，请参阅 通用序列操作。

set comprehension（集合綜合運算）
一種緊密的方法，用來處理一個可🄴代物件中的全部或部分元素，🄴將處理結果以一個 set 回傳。`results = {c for c in 'abracadabra' if c not in 'abc'}` 會🄴生一個字串 set：`{'r', 'd'}`。請參🄵 comprehensions。

single dispatch（單一調度）
generic function（泛型函式）調度的一種形式，在此，實作的選擇是基於單一引數的型🄴。

slice（切片）
一個物件，它通常包含一段*sequence*（序列）的某一部分。建立一段切片的方法是使用下標符號 (subscript notation) `[]`，若要給出多個數字，則在數字之間使用冒號，例如 `variable_name[1:3:5]`。在括號（下標）符號的🄴部，會使用 `slice` 物件。

special method（特殊方法）
一種會被 Python 自動呼叫的 method，用於對某種型🄴執行某種運算，例如加法。這種 method 的名稱會在開頭和結尾有兩個下底🄴。Special method 在 specialnames 中有詳細🄴明。

statement（陳述式）
陳述式是一個套組（suite，一個程式碼「區塊」）中的一部分。陳述式可以是一個*expression*（運算式），或是含有關鍵字（例如 `if`、`while` 或 `for`）的多種結構之一。

static type checker -- 静态类型检查器
读取 Python 代码并进行分析，以查找问题例如拼写错误的外部工具。另请参阅类型提示 以及 `typing` 模块。

strong reference（🄴參照）
在 Python 的 C API 中，强引用是指为持有引用的代码所拥有的对象的引用。在创建引用时可通过调用 `Py_INCREF()` 来获取强引用而在删除引用时可通过 `Py_DECREF()` 来释放它。

`Py_NewRef()` 函式可用於建立一個對物件的🄴參照。通常，在退出🄴參照的作用域之前，必須在該🄴參照上呼叫 `Py_DECREF()` 函式，以避免🄴漏一個參照。

另請參🄵*borrowed reference*（借用參照）。

text encoding（文字編碼）
Python 中的字串是一個 Unicode 碼點 (code point) 的序列（範圍在 `U+0000` -- `U+10FFFF` 之間）。若要儲存或傳送一個字串，它必須被序列化🄴一個位元組序列。

將一個字串序列化🄴位元組序列，稱🄴「編碼」，而從位元組序列重新建立該字串則稱🄴「解碼 (decoding)」。

有多種不同的文字序列化編解碼器 (codecs)，它們被統稱🄴「文字編碼」。

text file（文字檔案）
一個能🄴讀取和寫入 `str` 物件的一個*file object*（檔案物件）。通常，文字檔案實際上是存取位元組導向的資料流 (byte-oriented datastream) 🄵會自動處理*text encoding*（文字編碼）。文字檔案的例子有：以文字模式（`'r'` 或 `'w'`）開🄴的檔案、`sys.stdin`、`sys.stdout` 以及 `io.StringIO` 的實例。

另請參🄵*binary file*（二進制檔案），它是一個能🄵讀取和寫入類位元組串物件 *(bytes-like object)* 的檔案物件。

triple-quoted string（三引號🄵字串）
由三個雙引號（"）或單引號（'）的作🄵邊界的一個字串。雖然它們🄵🄵有提供🄵於單引號字串的任何額外功能，但基於許多原因，它們仍是很有用的。它們讓你可以在字串中包含未跳🄵 (unescaped) 的單引號和雙引號，而且它們不需使用連續字元 (continuation character) 就可以跨越多行，這使得它們在編寫🄵明字串時特🄵有用。

type（型🄵）
一個 Python 物件的型🄵🄵定了它是什🄵類型的物件；每個物件都有一個型🄵。一個物件的型🄵可以用它的 `__class__` 屬性來存取，或以 `type(obj)` 來檢索。

type alias（型🄵🄵名）
一個型🄵的同義詞，透過將型🄵指定給一個識🄵符 (identifier) 來建立。

型🄵🄵名對於簡化型🄵提示 *(type hint)* 很有用。例如：

```
def remove_gray_shades(
        colors: list[tuple[int, int, int]]) -> list[tuple[int, int, int]]:
    pass
```

可以寫成這樣，更具有可讀性：

```
Color = tuple[int, int, int]

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

請參🄵 `typing` 和 **PEP 484**，有此功能的描述。

type hint（型🄴提示）
一種*annotation*（🄵釋），它指定一個變數、一個 class 屬性或一個函式的參數或回傳值的預期型🄵。

类型提示是可选的而不是 Python 的强制要求，但它们对静态类型检查器 很有用处。它们还能协助 IDE 实现代码补全与重构。

全域變數、class 屬性和函式（不含區域變數）的型🄵提示，都可以使用 `typing.get_type_hints()` 來存取。

請參🄵 `typing` 和 **PEP 484**，有此功能的描述。

universal newlines（通用🄴行字元）
一種解譯文字流 (text stream) 的方式，會將以下所有的情🄴識🄵🄵一行的結束：Unix 行尾慣例 `'\n'`、Windows 慣例 `'\r\n'` 和舊的 Macintosh 慣例 `'\r'`。請參🄵 **PEP 278** 和 **PEP 3116**，以及用於 `bytes.splitlines()` 的附加用途。

variable annotation（變數🄵釋）
一個變數或 class 屬性的*annotation*（🄵釋）。

🄵釋變數或 class 屬性時，賦值是選擇性的：

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

變數🄵釋通常用於型🄵提示 *(type hint)*：例如，這個變數預期會取得 `int`（整數）值：

```
count: int = 0
```

變數🄵釋的語法在 annassign 章節有詳細的解釋。

請參🄵*function annotation*（函式🄵釋）、**PEP 484** 和 **PEP 526**，皆有此功能的描述。關於🄵釋的最佳實踐方法，另請參🄵 annotations-howto。

virtual environment（🄵擬環境）
一個協作隔離 (cooperatively isolated) 的執行環境，能讓 Python 的使用者和應用程式得以安裝和升級 Python 發🄵套件，而不會對同一個系統上運行的其他 Python 應用程式的行🄵🄵生干擾。

另請參🄵 `venv`。

virtual machine（🄵擬機器）
一部完全由軟體所定義的電腦 (computer)。Python 的🄵擬機器會執行由*bytecode*（位元組碼）編譯器所發出的位元組碼。

Zen of Python（Python 之🄵）
Python 設計原則與哲學的列表，其🄵容有助於理解和使用此語言。此列表可以透過在互動式提式字元後輸入「`import this`」來找到它。

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

沿革與授權

C.1 軟體沿革

Python 是由荷蘭數學和計算機科學研究學會（CWI，見 https://www.cwi.nl/）的 Guido van Rossum 於 1990 年代早期所創造，目的是作🄵一種稱🄵 ABC 語言的後繼者。🄵管 Python 包含了許多來自其他人的貢獻，Guido 仍是其主要作者。

1995 年，Guido 在維吉尼亞州雷斯頓的國家創新研究公司（CNRI，見 https://www.cnri.reston.va.us/）繼續他在 Python 的工作，🄵在那🄵發🄵了該軟體的多個版本。

2000 年五月，Guido 和 Python 核心開發團隊轉移到 BeOpen.com 🄵成立了 BeOpen PythonLabs 團隊。同年十月，PythonLabs 團隊轉移到 Digital Creations（現🄵 Zope Corporation；見 https://www.zope.org/）。2001 年，Python 軟體基金會（PSF，見 https://www.python.org/psf/）成立，這是一個專🄵擁有 Python 相關的智慧財🄵權而創立的非營利組織。Zope Corporation 是 PSF 的一個贊助會員。

所有的 Python 版本都是開源的（有關開源的定義，參🄵 https://opensource.org/）。歷史上，大多數但非全部的 Python 版本，也是 GPL 相容的；以下表格總結各個版本的差🄵。

發🄵版本	源自	年份	擁有者	GPL 相容性？
0.9.0 至 1.2	不適用	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 以上	2.1.1	2001 至今	PSF	是

備🄴： GPL 相容🄵不表示我們是在 GPL 下發🄴 Python。不像 GPL，所有的 Python 授權都可以讓您發🄴修改後的版本，但不一定要使您的變更成🄵開源。GPL 相容的授權使得 Python 可以結合其他在 GPL 下發🄵的軟體一起使用；但其它的授權則不行。

感謝許多的外部志工，在 Guido 指導下的付出，使得這些版本的發🄴成🄵可能。

C.2 關於存取或以其他方式使用 Python 的合約條款

Python 軟體和🄴明文件的授權是基於*PSF 授權合約*。

從 Python 3.8.6 開始，🄴明文件中的範例、程式庫和其他程式碼，是被雙重授權 (dual licensed) 於 PSF 授權合約以及*Zero-Clause BSD 授權*。

有些被納入 Python 中的軟體是基於不同的授權。這些授權將會與其授權之程式碼一起被列出。關於這些授權的不完整清單，請參🄵被收🄵軟體的授權與致謝。

C.2.1 用於 PYTHON 3.11.11 的 PSF 授權合約

```
1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"),␣
↪and
   the Individual or Organization ("Licensee") accessing and otherwise using␣
↪Python
   3.11.11 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 3.11.11 alone or in any derivative
   version, provided, however, that PSF's License Agreement and PSF's notice␣
↪of
   copyright, i.e., "Copyright © 2001-2023 Python Software Foundation; All␣
↪Rights
   Reserved" are retained in Python 3.11.11 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 3.11.11 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
   3.11.11.

4. PSF is making Python 3.11.11 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 3.11.11 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
```

```
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.11.11
   FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT␣
↪OF
   MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.11.11, 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 3.11.11, Licensee agrees
   to be bound by the terms and conditions of this License Agreement.
```

C.2.2 用於 PYTHON 2.0 的 BEOPEN.COM 授權合約

BEOPEN PYTHON 開源授權合約第 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.
```

(繼續下一頁)

(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 用於 PYTHON 1.6.1 的 CNRI 授權合約

```
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.
```

(繼續下一頁)

(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 用於 PYTHON 0.9.0 至 1.2 的 CWI 授權合約

```
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 用於 PYTHON 3.11.11 🄵明文件🄵程式碼的 ZERO-CLAUSE BSD 授權

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

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

C.3 被收🄵軟體的授權與致謝

本節是一個不完整但持續增加的授權與致謝清單，對象是在 Python 發🄵版本中所收🄵的第三方軟體。

C.3.1 Mersenne Twister

_random 模組包含了以 http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html 的下載🄵容🄵基礎的程式碼。以下是原始程式碼的完整聲明：

```
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 Sockets

socket 使用了 getaddrinfo() 和 getnameinfo() WIDE 项目的不同源文件中: https://www.wide.ad.jp/

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

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

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

asynchat 和 asyncore 模組包含以下聲明:

```
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` 模組包含以下聲明：

```
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 執行追🄵

`trace` 模組包含以下聲明：

```
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 與 UUdecode 函式

uu 模組包含以下聲明：

```
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 遠端程序呼叫

xmlrpc.client 模組包含以下聲明：

```
    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
```

(繼續下一頁)

(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.test_epoll` 模块包含以下说明:

```
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` 模組對於 kqueue 介面包含以下聲明:

```
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
```

(繼續下一頁)

(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 檔案包含 Marek Majkowski' 基於 Dan Bernstein 的 SipHash24 演算法的實作。它包含以下聲明:

```
<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 與 dtoa

Python/dtoa.c 檔案提供了 C 的 dtoa 和 strtod 函式，用於將 C 的雙精度浮點數和字串互相轉🄵。該檔案是衍生自 David M. Gay 建立的同名檔案，後者現在可以從 https://web.archive.org/web/20220517033456/http://www.netlib.org/fp/dtoa.c 下載。於 2009 年 3 月 16 日所檢索的原始檔案包含以下版權與授權聲明:

```
/****************************************************************
 *
 * 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.
```

(繼續下一頁)

(繼續上一頁)

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

C.3.12 OpenSSL

如果操作系统提供支持，则 hashlib, posix, ssl, crypt 模块会使用 OpenSSL 库来提高性能。此外，Python 的 Windows 和 macOS 安装程序可能会包括 OpenSSL 库的副本，所以我们也在此包括一份 OpenSSL 许可证的副本。对于 OpenSSL 3.0 版及其后续衍生版本，均使用 Apache 许可证 v2:

```
                                 Apache License
                           Version 2.0, January 2004
                        https://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.
```

(繼續下一頁)

(繼續上一頁)

```
   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
```

(繼續下一頁)

(繼續上一頁)

```
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
```

(繼續下一頁)

(繼續上一頁)

```
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
```

C.3.13 expat

pyexpat 扩展是使用所包括的 expat 源副本来构建的，除非配置了 --with-system-expat:

```
Copyright (c) 1998, 1999, 2000 Thai Open Source Software Center Ltd
                               and Clark Cooper

Permission is hereby granted, free of charge, to any person obtaining
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

除非在建置 _ctypes 擴充時設定🄵 --with-system-libffi，否則該擴充會用一個🄴含 libffi 原始碼的副本來建置:

```
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
```

(繼續下一頁)

(繼續上一頁)

```
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

如果在系統上找到的 zlib 版本太舊以致於無法用於建置 `zlib` 擴充，則該擴充會用一個🄴含 zlib 原始碼的副本來建置:

```
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` 使用的雜🄴表 (hash table) 實作，是以 cfuhash 專案🄴基礎：

```
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

除非在建置 `_decimal` 模組時設定🄵 `--with-system-libmpdec`，否則該模組會用一個🄴含 libmpdec 函式庫的副本來建置：

```
Copyright (c) 2008-2020 Stefan Krah. All rights reserved.

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

1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
```

(繼續下一頁)

(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 測試套件

`test` 程式包中的 C14N 2.0 測試套件 (`Lib/test/xmltestdata/c14n-20/`) 是從 W3C 網站 https://www.w3.org/TR/xml-c14n2-testcases/ 被檢索，且是基於 3-clause BSD 授權被發🄵:

```
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.
```

C.3.19 audioop

audioop 模块使用了 SoX 项目的 g771.c 文件中的基础代码。https://sourceforge.net/projects/sox/files/sox/12.17.7/sox-12.17.7.tar.gz

> 此源代码是 Sun Microsystems, Inc. 的产品并可供无限制地使用。用户可以拷贝或修改此源代码而无须付费。
>
> SUN SOURCE CODE IS PROVIDED AS IS WITH NO WARRANTIES OF ANY KIND INCLUDING THE WARRANTIES OF DESIGN, MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE, OR ARISING FROM A COURSE OF DEALING, USAGE OR TRADE PRACTICE.
>
> 提供的 Sun 源代码不附带技术支持并且 Sun Microsystems, Inc. 也没有义务协助其使用、排错、修改或增强。
>
> SUN MICROSYSTEMS, INC. SHALL HAVE NO LIABILITY WITH RESPECT TO THE INFRINGEMENT OF COPYRIGHTS, TRADE SECRETS OR ANY PATENTS BY THIS SOFTWARE OR ANY PART THEREOF.
>
> 在任何情况下 Sun Microsystems, Inc. 均不对任何收入或利润损失或其他特殊的、间接的和后续的损害负责，即使 Sun 已被告知可能发生此类损害。
>
> Sun Microsystems, Inc. 2550 Garcia Avenue Mountain View, California 94043

C.3.20 asyncio

`asyncio` 模块的某些部分来自 uvloop 0.16，它是基于 MIT 许可证发行的:

```
Copyright (c) 2015-2021 MagicStack Inc.  http://magic.io

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

APPENDIX D

版權宣告

Python 和這份🄴明文件的版權：

完整的授權條款資訊請參見沿革與授權。

索引

非依字母順序

A

B

C

D

E

P

Q

R

S

T

U

V

W

X

Z