
Python Setup and Usage

发行版本 3.13.1

Guido van Rossum and the Python development team

一月 11, 2025

Python Software Foundation
Email: docs@python.org

1	命令行与环境	3
1.1	命令行	3
1.1.1	接口选项	3
1.1.2	通用选项	5
1.1.3	其他选项	6
1.1.4	控制颜色	9
1.1.5	不应当使用的选项	10
1.2	环境变量	10
1.2.1	调试模式变量	16
2	在类 Unix 环境下使用 Python	17
2.1	获得并安装 Python 的最新版本	17
2.1.1	在 Linux 中	17
2.1.2	在 FreeBSD 和 OpenBSD 上	17
2.2	构建 Python	18
2.3	与 Python 相关的路径和文件	18
2.4	杂项	18
2.5	自定义 OpenSSL	19
3	配置 Python	21
3.1	构建要求	21
3.2	已生成的文件	21
3.2.1	配置脚本	22
3.3	配置选项	22
3.3.1	通用选项	22
3.3.2	C 编译器选项	25
3.3.3	链接器选项	25
3.3.4	用于第三方依赖的选项	25
3.3.5	WebAssembly 选项。	27
3.3.6	安装时的选项	27
3.3.7	性能选项	28
3.3.8	Python 调试级编译	29
3.3.9	调试选项	30
3.3.10	链接器选项	31
3.3.11	库选项	31
3.3.12	安全性选项	32
3.3.13	macOS 选项	32
3.3.14	iOS 选项	33
3.3.15	交叉编译选项	33
3.4	Python 构建系统	34
3.4.1	构建系统的主要文件	34

3.4.2	主要构建步骤	34
3.4.3	主要 Makefile 目标	34
3.4.4	C 扩展	35
3.5	编译器和链接器的标志	36
3.5.1	预处理器的标志	36
3.5.2	编译器标志	36
3.5.3	链接器标志位	38
4	在 Windows 上使用 Python	41
4.1	完整安装程序	41
4.1.1	安装步骤	41
4.1.2	删除 MAX_PATH 限制	42
4.1.3	无 UI 安装	43
4.1.4	免下载安装	45
4.1.5	修改安装	45
4.1.6	安装自由线程二进制文件	45
4.2	Microsoft Store 包	46
4.2.1	已知的问题	47
4.3	nuget.org 安装包	47
4.3.1	自由线程版软件包	48
4.4	可嵌入的包	48
4.4.1	Python 应用程序	49
4.4.2	嵌入 Python	49
4.5	替代捆绑包	49
4.6	配置 Python	49
4.6.1	附录：设置环境变量	49
4.6.2	查找 Python 可执行文件	50
4.7	UTF-8 模式	50
4.8	适用于 Windows 的 Python 启动器	51
4.8.1	入门	51
4.8.2	Shebang 行	52
4.8.3	shebang 行的参数	53
4.8.4	自定义	53
4.8.5	诊断	54
4.8.6	试运行	55
4.8.7	按需安装	55
4.8.8	返回码	55
4.9	查找模块	55
4.10	附加模块	56
4.10.1	PyWin32	56
4.10.2	cx_Freeze	57
4.11	在 Windows 上编译 Python	57
4.12	其他平台	57
5	在 macOS 上使用 Python	59
5.1	使用来自 python.org 的 macOS 版 Python	59
5.1.1	安装步骤	59
5.1.2	如何运行 Python 脚本	67
5.2	Alternative Distributions	67
5.3	安装额外的 Python 包	67
5.4	GUI 编程	67
5.5	进阶	68
5.5.1	安装自由线程二进制文件	68
5.5.2	使用命令行安装	69
5.5.3	分发 Python 应用程序	70
5.5.4	App Store 合规性	71
5.6	其他资源	71

6	在 Android 上使用 Python	73
6.1	添加 Python 到 Android app	73
7	在 iOS 上使用 Python	75
7.1	iOS 上的 Python 运行时	75
7.1.1	iOS 版本兼容性	75
7.1.2	平台识别	75
7.1.3	标准库可用性	76
7.1.4	二进制扩展模块	76
7.1.5	编译器存根二进制文件	76
7.2	在 iOS 上安装 Python	76
7.2.1	构建 iOS 应用程序的工具	76
7.2.2	在 iOS 项目中添加 Python	77
7.2.3	测试 Python 软件包	79
7.3	App Store 合规性	80
8	编辑器和集成开发环境	81
A	术语对照表	83
B	关于本文档	99
B.1	Python 文档的贡献者	99
C	历史和许可证	101
C.1	该软件的历史	101
C.2	获取或以其他方式使用 Python 的条款和条件	102
C.2.1	用于 PYTHON 3.13.1 的 PSF 许可协议	102
C.2.2	用于 PYTHON 2.0 的 BEOPEN.COM 许可协议	103
C.2.3	用于 PYTHON 1.6.1 的 CNRI 许可协议	103
C.2.4	用于 PYTHON 0.9.0 至 1.2 的 CWI 许可协议	104
C.2.5	ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON 3.13.1 DOCUMENTATION	105
C.3	收录软件的许可与鸣谢	105
C.3.1	Mersenne Twister	105
C.3.2	套接字	106
C.3.3	异步套接字服务	106
C.3.4	Cookie 管理	107
C.3.5	执行追踪	107
C.3.6	UUencode 与 UUdecode 函数	108
C.3.7	XML 远程过程调用	108
C.3.8	test_epoll	109
C.3.9	Select kqueue	109
C.3.10	SipHash24	110
C.3.11	strtod 和 dtoa	110
C.3.12	OpenSSL	110
C.3.13	expat	113
C.3.14	libffi	114
C.3.15	zlib	114
C.3.16	cfuhash	115
C.3.17	libmpdec	115
C.3.18	W3C C14N 测试套件	116
C.3.19	mimalloc	116
C.3.20	asyncio	117
C.3.21	Global Unbounded Sequences (GUS)	117
D	版权所有	119
	索引	121

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

为获取各种设置信息，CPython 解析器会扫描命令行与环境。

CPython 实现细节：其他实现的命令行方案可能会有所不同。详见 [implementations](#)。

1.1 命令行

调用 Python 时，可以指定下列任意选项：

```
python [-bBdEhiIOPqRsSuvVWx?] [-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 接口选项

解释器接口类似于 UNIX shell，但提供了额外的调用方法：

- 当调用时附带连接到某个 tty 设备的标准输入时，它会提示输入命令并执行它们直至读到一个 EOF（文件结束字符，你可以在 UNIX 上按 Ctrl-D 或在 Windows 上按 Ctrl-Z, Enter 来产生此字符）。有关交互模式的更多信息，请参阅 [tut-interac](#)。
- 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>`

执行 *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)。

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

备注

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 # 获取详情
```

引发一个 审计事件 `cpython.run_module` 并附带参数 `module-name`。

参见

`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 版本发生变更: 提供包名称来运行 `__main__` 子模块。

在 3.4 版本发生变更: 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。

引发一个不带参数的 审计事件 `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`。

参见

`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`）。

参见

tut-invoking

在 3.4 版本发生变更：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。

Added in version 3.11.

`--help-xoptions`

打印实现专属 `-x` 选项的简短描述然后退出。

Added in version 3.11.

`--help-all`

打印完整使用信息然后退出。

Added in version 3.11.

`-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]
```

Added in version 3.6: `-vv` 选项。

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 版本发生变更: 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

启用解析器调试输出 (仅供专家查看)。另请参见 `PYTHONDEBUG` 环境变量。

此选项需要 Python 的调试构建版, 否则它将被忽略。

-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。

Added in version 3.4.

-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 版本发生变更: 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 版本发生变更: 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 `PYTHONSAFE_PATH` 环境变量, 以及 `-E` 和 `-I` (隔离) 选项。

Added in version 3.11.

-q

即使在交互模式下也不显示版权和版本信息。

Added in version 3.2.

-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。

Added in version 3.2.3.

在 3.7 版本发生变更：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`。

 参见

PEP 370 -- 分用户的 `site-packages` 目录

-S

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

-u

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

另请参阅 `PYTHONUNBUFFERED`。

在 3.7 版本发生变更：`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 版本发生变更：由 `site` 模块可以得到将要处理的站点路径和 `.pth` 文件。

参阅 `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 # 每个调用位置警告一次
-Werror  # 转换为异常
-Walways # 每次都警告
-Wall    # 与 -Walways 相同
-Wmodule # 每个调用模块警告一次
-Wonce   # 每个 Python 进程警告一次
-Wignore # 从不警告
```

`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`。
Added in version 3.3.
- `-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。
Added in version 3.4.
- `-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。
Added in version 3.4.
- `-X int_max_str_digits` 将配置整数字符串转换长度限制。另请参阅 `PYTHONINTMAXSTRDIGITS`。
Added in version 3.11.
- `-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`。
Added in version 3.7.
- `-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`。
Added in version 3.7.
- `-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`。
Added in version 3.7.

- `-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](#)。

Added in version 3.8.

- `-X warn_default_encoding` issues a `EncodingWarning` when the locale-specific default encoding is used for opening files. See also [PYTHONWARNDEFAULTENCODING](#).

Added in version 3.10.

- `-X no_debug_ranges` 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](#)。

Added in version 3.11.

- `-X frozen_modules` 确定已冻结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。如果是安装版 Python（正常情况）则默认为 `on`。如果是在开发中（基于源代码树运行）则默认为 `off`。请注意 `importlib_bootstrap` 和 `importlib_bootstrap_external` 冻结模块总是会被使用，即使该旗标被设为 `off`。另请参阅 [PYTHON_FROZEN_MODULES](#)。

Added in version 3.11.

- `-X perf` 会启用对 Linux `perf` 性能分析器的支持。当提供了此选项时，`perf` 性能分析器将能够报告 Python 调用。此选项仅在某些平台上可用而在当前系统不支持的情况下将不做任何事。默认值为“off”。另请参阅 [PYTHONPERFSUPPORT](#) 和 `perf_profiling`。

Added in version 3.12.

- `-X perf_jit` 将启用对 Linux `perf` 性能分析器的支持并附带 DWARF 支持。当提供了此选项时，`perf` 性能分析器将能够使用 DWARF 信息来报告 Python 调用。此选项仅在某些平台上可用而在当前系统不支持的情况下将不做任何事。默认值为“off”。另请参阅 [PYTHON_PERF_JIT_SUPPORT](#) 和 `perf_profiling`。

Added in version 3.13.

- `-X cpu_count=n` 将覆盖 `os.cpu_count()`，`os.process_cpu_count()` 和 `multiprocessing.cpu_count()`。`n` 必须大于等于 1。此选项对于需要限制某个容器系统的 CPU 资源的用户来说会很有用处。另请参阅 [PYTHON_CPU_COUNT](#)。如果 `n` 为 `default`，则不会覆盖任何值。

Added in version 3.13.

- `-X presite=package.module` 指明一个模块应当 `site` 模块执行之前以及 `__main__` 模块存在之前被导入。因此，这个被导入的模块不是 `__main__`。此选项适用于要早在 Python 初始化期间就执行的代码。Python 需要以调试模式构建此选项才能存在。另请参阅 [PYTHON_PRESITE](#)。

Added in version 3.13.

- `-X gil=0,1` 强制分别禁用或启用 GIL。设为 0 仅在配置了 `--disable-gil` 的构建版上可用。另请参阅 [PYTHON_GIL](#) 和 [whatsnew313-free-threaded-cpython](#)。

Added in version 3.13.

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

Added in version 3.2.

在 3.9 版本发生变更: 移除了 `-X showalloccount` 选项。

在 3.10 版本发生变更: 移除了 `-X oldparser` 选项。

1.1.4 控制颜色

Python 解释器默认被配置为在特定场景例如当显示回溯信息时使用颜色高亮输出。此行为可通过设置不同的环境变量来控制。

将环境变量 `TERM` 设为 `dumb` 将禁用颜色。

如果设置了 `FORCE_COLOR` 环境变量, 则无论 `TERM` 的值为何都将启用彩色。这适用于不属于终端但仍然会显示 ANSI 转义序列的 CI 系统。

如果设置了 `NO_COLOR` 环境变量, 则 Python 将在输出中禁用所有彩色。此变量的优先级高于 `FORCE_COLOR`。

所有这些环境变量也被其他工具用来控制颜色输出。要仅在 Python 解释器中控制颜色输出, 可以使用 `PYTHON_COLORS` 环境变量。此变量的优先级高于 `NO_COLOR`, 后者的优先级又高于 `FORCE_COLOR`。

1.1.5 不应当使用的选项

`-J`

保留给 Jython 使用。

1.2 环境变量

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

PYTHONHOME

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

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

PYTHONPATH

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

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

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

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

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。

Added in version 3.11.

PYTHONPLATLIBDIR

如果它被设为非空字符串, 则会覆盖 `sys.platlibdir` 值。

Added in version 3.9.

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__`。

在启动时调用文件名作为参数会引发审计事件 `cpython.run_startup`。

PYTHONOPTIMIZE

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

PYTHONBREAKPOINT

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

Added in version 3.7.

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`。

此环境变量需要 *Python* 的调试构建版，否则它将被忽略。

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。

引发一个不带参数的审计事件 `cpython.run_stdin`。

在 3.12.5 版本发生变更：（还有 3.11.10, 3.10.15, 3.9.20 和 3.8.20）发出审计事件。

在 3.13 版本发生变更：如果无法做到则使用 PyREPL，在此情况下 `PYTHONSTARTUP` 也会被执行。将发出审计事件。

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。

Added in version 3.8.

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。

Added in version 3.2.3.

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。

Added in version 3.11.

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 版本发生变更：“`encodingname`”部分现在是可选的。

在 3.6 版本发生变更：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`。

➡ 参见

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。

➡ 参见

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    # 每个调用位置警告一次
PYTHONWARNINGS=error      # 转换为异常
PYTHONWARNINGS=always     # 每次都警告
PYTHONWARNINGS=all        # 与 PYTHONWARNINGS=always 相同
PYTHONWARNINGS=module     # 每个调用模块警告一次
PYTHONWARNINGS=once       # 每个 Python 进程警告一次
PYTHONWARNINGS=ignore     # 从不警告
```

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。

Added in version 3.3.

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。

Added in version 3.4.

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。

Added in version 3.7.

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。

Added in version 3.4.

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。
- `mimalloc`: 对 `PYMEM_DOMAIN_MEM` 和 `PYMEM_DOMAIN_OBJ` 域使用 `mimalloc` 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。
- `mimalloc_debug`: 与 `mimalloc` 相同但还会安装调试钩子。

Added in version 3.6.

在 3.7 版本发生变更: 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 版本发生变更: 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。

Availability: Windows.

Added in version 3.6: 更多详情请参阅 [PEP 529](#)。

PYTHONLEGACYWINDOWSSTDIO

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

如果标准流被重定向（到文件或管道）而不是指向控制台缓冲区则该变量会被忽略。

Availability: Windows.

Added in version 3.6.

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。

Availability: Unix.

Added in version 3.7: 请参阅 [PEP 538](#) 了解详情。

PYTHONDEVMODE

如果此环境变量被设为一个非空字符串，则会启用 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。

Added in version 3.7.

PYTHONUTF8

如果设为 1，将会启用 Python UTF-8 模式。

若设为 0，则会禁用 Python UTF-8 模式。

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。

Added in version 3.7.

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`。

请参阅 [io-encoding-warning](#) 了解详情。

Added in version 3.10.

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。

Added in version 3.11.

PYTHONPERFSUPPORT

如果此变量被设为非零值，它将启用对 Linux perf 分析器的支持以便 Python 调用能被它检测到。

如果设为 0，则禁用 Linux perf 性能分析器支持。

另请参阅 `-X perf` 命令行选项和 `perf_profiling`。

Added in version 3.12.

PYTHON_PERF_JIT_SUPPORT

如果此变量被设为非零值，它将启用对 Linux perf 分析器的支持以便 Python 调用能被它使用 DWARF 信息来检测。

如果设为 0，则禁用 Linux perf 性能分析器支持。

另请参阅 `-X perf_jit` 命令行选项和 `perf_profiling`。

Added in version 3.13.

PYTHON_CPU_COUNT

如果此变量被设为正整数值，它将覆盖 `os.cpu_count()` 和 `os.process_cpu_count()` 的返回值。

另请参阅 `-X cpu_count` 命令行选项。

Added in version 3.13.

PYTHON_FROZEN_MODULES

如果此变量被设为 `on` 或 `off`，它将确定已冻结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。对于非调试构建版（正常情况）默认为 `on` 而对调试构建版则为 `off`。请注意 `importlib_bootstrap` 和 `importlib_bootstrap_external` 冻结模块总是会被使用，即使该旗标被设为 `off`。

另请参阅 `-X frozen_modules` 命令行选项。

Added in version 3.13.

PYTHON_COLORS

如果此变量被设为 1，解释器将对各种输出添加彩色。将其设为 0 将禁用此行为。另请参阅[控制颜色](#)。

Added in version 3.13.

PYTHON_BASIC_REPL

如果此变量被设为 1，解释器将不再尝试加载需要 `curses` 和 `readline` 的基于 Python 的 [REPL](#)，而将改用传统的基于解析器的 [REPL](#)。

Added in version 3.13.

PYTHON_HISTORY

此环境变量可被用来设置 `.python_history` 文件的位置（在默认情况下，它将为用户主目录下的 `.python_history` 文件）。

Added in version 3.13.

PYTHON_GIL

如果将此变量设为 1，则将强制启用全局解释器锁 (GIL)。将其设为 0 将强制禁用 GIL (需要使用 `--disable-gil` 构建选项来配置 Python)。

另请参阅 `-X gil` 命令行选项，该选项的优先级高于此变量，并请参阅 [whatsnew313-free-threaded-cpython](#)。

Added in version 3.13.

1.2.1 调试模式变量

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。

需要使用 `--with-trace-refs` 构建选项来配置 Python。

PYTHONDUMPREFSFILE

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。

需要使用 `--with-trace-refs` 构建选项来配置 Python。

Added in version 3.11.

PYTHON_PRESITE

如果此变量被设为一个模块，则该模块将在解释器生命周期的较早阶段被导入，即在 `site` 模块被执行之前，并在 `__main__` 模块被创建之前。因此，这个被导入的模块不会被作为 `__main__`。

这适用于要早在 Python 初始化期间就执行的代码。

要导入一个子模块，请使用 `package.module` 作为值，就像在 `import` 语句中那样。

另请参阅 `-X presite` 命令行选项，该选项的优先级高于此变量。

需要使用 `--with-pydebug` 构建选项来配置 Python。

Added in version 3.13.

在类 Unix 环境下使用 Python

2.1 获得并安装 Python 的最新版本

2.1.1 在 Linux 中

Python 预装在大多数 Linux 发行版上，并在它们以外的发行版上以软件包的形式提供。不过在你所用的发行版的软件包中可能没有某些你需要使用的功能。这时你可以用源代码来编译最新版本的 Python。

对于最新版本的 Python 未预装也不在软件仓库中的情况，你可以为你的发行版制作软件包。请参阅下面的链接：

参见

<https://www.debian.org/doc/manuals/maint-guide/first.en.html>

对于 Debian 用户

<https://en.opensuse.org/Portal:Packaging>

对于 OpenSuse 用户

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

对于 Fedora 用户

<https://slackbook.org/html/package-management-making-packages.html>

对于 Slackware 用户

2.1.2 在 FreeBSD 和 OpenBSD 上

- FreeBSD 用户，使用以下命令添加包：

```
pkg install python3
```

- OpenBSD 用户，使用以下命令添加包：

```
pkg_add -r python
```

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

例如：i386 用户获取 Python 2.5.1 的可用版本：

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

2.2 构建 Python

如果你想自己编译 CPython，首先要做的是获取 [源代码](#)。你可以下载最新发布版的 [source](#) 或是直接抓取最新的 [clone](#)。（如果你想要贡献补丁，那么你就必须先 [clone](#)。）

构建过程由常用命令组成：

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

特定 Unix 平台的[配置选项](#)和注意事项通常会详细地记录在 Python 源代码树的根目录下的 [README.rst](#) 文件中。

警告

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

2.3 与 Python 相关的路径和文件

这些取决于本机安装惯例的不同；`prefix` 和 `exec_prefix` 依赖于具体安装并且应当被解读为针对 GNU 软件；它们可能具有相同的含义。

例如，在大多数 Linux 系统上，两者的默认值是 `/usr`。

文件/目录	含意
<code>exec_prefix/bin/python3</code>	解释器的推荐位置
<code>prefix/lib/pythonversion,</code> <code>exec_prefix/lib/pythonversion</code>	包含标准模块的目录的推荐位置
<code>prefix/include/pythonversion,</code> <code>exec_prefix/include/pythonversion</code>	包含开发 Python 扩展和嵌入解释器所需的 <code>include</code> 文件的目录的推荐位置

2.4 杂项

要在 Unix 上使用 Python 脚本，需要添加可执行权限，例如：

```
$ chmod +x script
```

并在脚本的顶部放置一个合适的 Shebang 线。一个很好的选择通常是：

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

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

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

2.5 自定义 OpenSSL

1. 要使用发行商的 OpenSSL 配置和系统信任存储库，请找到包含 openssl.cnf 文件或符号链接的目录，它位于 /etc 中。在大多数发行版上该文件是在 /etc/ssl 或者 /etc/pki/tls 中。该目录还应当包含一个 cert.pem 文件和/或一个 certs 目录。

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

2. 下载、编译并安装 OpenSSL。请确保你使用 install_sw 而不是 install。install_sw 的目标不会覆盖 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. 使用自定义的 OpenSSL 编译 Python（参考配置 --with-openssl 和 --with-openssl-rpath 选项）

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

备注

OpenSSL 的补丁发布版具有向下兼容的 ABI。你不需要重新编译 Python 来更新 OpenSSL。使用一个新的版本来替代自定义 OpenSSL 安装版就可以了。

3.1 构建要求

编译 CPython 所需要的特性和最低版本:

- C11 编译器。可选的 C11 特性 不是必须的。
- 在 Windows 上, 需要 Microsoft Visual Studio 2017 或更新版本。
- 对 IEEE 754 浮点数和 浮点 Not-a-Number (NaN) 的支持。
- 对线程的支持。
- 对于 ssl 和 hashlib 扩展模块 OpenSSL 1.1.1 为最低版本而 OpenSSL 3.0.9 为推荐的最低版本。
- SQLite 3.15.2 用于 sqlite3 扩展模块。
- Tcl/Tk 8.5.12 用于 tkinter 模块。
- 需要 autoconf 2.71 和 aclocal 1.16.5 来重新生成 configure 脚本。

在 3.1 版本发生变更: 现在需要 Tcl/Tk 8.3.1 版本。

在 3.5 版本发生变更: 在 Windows 上, 现在需要 Visual Studio 2015 或更新的版本。现在需要 Tcl/Tk 8.4 版本。

在 3.6 版本发生变更: 现在要求选定的 C99 特性, 如 `<stdint.h>` 和 `static inline` 函数。

在 3.7 版本发生变更: 现在要求线程支持和 OpenSSL 1.0.2。

在 3.10 版本发生变更: 现在需要 OpenSSL 1.1.1。需要 SQLite 3.7.15。

在 3.11 版本发生变更: C11 编译器, 现在需要 IEEE 754 和 NaN 支持。在 Windows 上, 需要 Visual Studio 2017 或更新的版本。现在需要 Tcl/Tk version 8.5.12 用于 tkinter 模块。

在 3.13 版本发生变更: 现在需要 autoconf 2.71, aclocal 1.16.5 和 SQLite 3.15.2。

另请参阅 [PEP 7](#) "Style Guide for C Code" 和 [PEP 11](#) "CPython platform support"。

3.2 已生成的文件

为了减少构建依赖性, Python 源代码包含多个已生成的文件。重新生成所有已生成文件的命令如下:

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

Makefile.pre.in 文件记录了已生成的文件、它们的输入以及用于重新生成它们的工具。搜索 `regen-*` `make target`。

3.2.1 配置脚本

`make regen-configure` 命令将使用 `Tools/build/regen-configure.sh` shell 脚本生成 `aclocal.m4` 文件和 `configure` 脚本，它通过使用一个 **Ubuntu** 容器来获取同样的工具版本并具有可复现的输出。

容器是可选的，以下命令可以在本地运行：

```
autoreconf -ivf -Werror
```

生成的文件可根据实际的 `autoconf-archive`、`aclocal` 和 `pkg-config` 版本进行改变。

3.3 配置选项

用以下方式列出所有 `configure` 脚本选项：

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

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

3.3.1 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。

Added in version 3.6.

--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。

定义 `PYLONG_BITS_IN_DIGIT` 为 15 或 30。

参见 `sys.int_info.bits_per_digit`。

--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 版本发生变更：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。

Added in version 3.9.

--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。

Added in version 3.9.

--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。

参阅 [PYTHONCOERCECLOCALE](#) 和 [PEP 538](#)。

--without-freelists

禁用除空元组单例以外的所有自由列表。Disable all freelists except the empty tuple singleton.

Added in version 3.11.

--with-platlibdir=DIRNAME

Python 库目录名（默认为 `lib`）。

Fedora 和 SuSE 在 64 位平台用 `lib64` 。

参阅 `sys.platlibdir` 。

Added in version 3.9.

--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。

Added in version 3.10.

--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

Added in version 3.11.

--enable-pystats

启用内部 Python 性能统计数据收集。

在默认情况下，将关闭统计数据收集。使用 `python3 -X pystats` 命令或设置 `PYTHONSTATS=1` 环境变量在 Python 启动时启用统计数据收集。

在 Python 退出时，如果统计数据收集已启用且未清除则会转储统计数据。

效果如下：

- 增加了 `-X pystats` 命令行选项。
- 增加了 `PYTHONSTATS` 环境变量。
- 定义 `Py_STATS` 宏。
- 为 `sys` 模块增加函数：
 - `sys._stats_on()`: 启用统计数据收集。
 - `sys._stats_off()`: 关闭统计数据收集。
 - `sys._stats_clear()`: 清除统计数据。
 - `sys._stats_dump()`: 将统计数据转储到文件, 并清除统计数据。

统计数据将被转储至 `/tmp/py_stats/` (Unix) 或 `C:\temp\py_stats\` (Windows) 中的任意 (可能唯一) 文件。如果该目录不存在, 结果将被打印到 `stderr`。

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

统计数据:

- 操作码:
 - 专门类别: `success`, `failure`, `hit`, `deferred`, `miss`, `deopt`, `failures`;
 - 执行计数;
 - 对应计数。
- 调用:
 - 内联 Python 调用;
 - `PyEval` 调用;
 - 推入的帧;
 - 创建的帧对象;
 - Eval 调用: `vector`, `generator`, `legacy`, `function VECTORCALL`, `build class`, `slot`, `function "ex"`, `API`, `method`。
- 对象:
 - `incred` 和 `decred`;
 - 解释器 `incred` 和 `decred`;
 - 分配: `all`, 512 bytes, 4 kiB, `big`;
 - 空闲;
 - 去向/来自空闲列表;
 - 实体化/非实体化的字典;
 - 类型缓存;
 - 优化尝试;
 - 已创建/已执行的优化跟踪;
 - 已执行的 `uop`。
- 垃圾回收器:
 - 垃圾回收;
 - 已访问的对象;
 - 已收集的对象。

Added in version 3.11.

--disable-gil

启用对不带 *global interpreter lock* (GIL) 运行 Python 的 **实验性** 支持：自由线程构建版。

定义 `Py_GIL_DISABLED` 宏并向 `sys.abiflags` 添加 "t"。

请参阅 `whatsnew313-free-threaded-cpython` 了解详情。

Added in version 3.13.

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

指明如何集成 JIT 编译器。

- `no` - 构建不带 JIT 的解释器。
- `yes` - 构建带有 JIT 的解释器。
- `yes-off` - 构建带 JIT 的解释器但默认禁用它。
- `interpreter` - 构建不带 JIT 的解释器，但针对启用第 2 层级的解释器。

根据惯例，`--enable-experimental-jit` 是 `--enable-experimental-jit=yes` 的简写形式。

Added in version 3.13.

PKG_CONFIG

指向 `pkg-config` 工具的路径。

PKG_CONFIG_LIBDIR**PKG_CONFIG_PATH**

`pkg-config` 选项。

3.3.2 C 编译器选项

CC

C 编译器指令。

CFLAGS

C 编译器标志。

CPP

C 预处理器命令。

CPPFLAGS

C 预处理器旗标，例如 `-Iinclude_dir`。

3.3.3 链接器选项

LDFLAGS

链接器旗标，例如 `-Llibrary_directory`。

LIBS

要传给链接器的库，例如 `-llibrary`。

MACHDEP

依赖具体机器的库文件名称。

3.3.4 用于第三方依赖的选项

Added in version 3.11.

BZIP2_CFLAGS

BZIP2_LIBS

将 Python 链接到 libbz2 的 C 编译器和链接器旗标，由 bz2 模块使用，覆盖 pkg-config。

CURSES_CFLAGS

CURSES_LIBS

针对 libncurses 或 libncursesw 的 C 编译器和链接器旗标，由 curses 模块使用，覆盖 pkg-config。

GDBM_CFLAGS

GDBM_LIBS

针对 gdbm 的 C 编译器和链接器旗标。

LIBB2_CFLAGS

LIBB2_LIBS

针对 libb2 (BLAKE2) 的 C 编译器和链接器旗标，由 hashlib 模块使用，覆盖 pkg-config。

LIBEDIT_CFLAGS

LIBEDIT_LIBS

针对 libedit 的 C 编译器和链接器旗标，由 readline 模块使用，覆盖 pkg-config。

LIBFFI_CFLAGS

LIBFFI_LIBS

针对 libffi 的 C 编译器和链接器旗标，由 ctypes 模块使用，覆盖 pkg-config。

LIBMPDEC_CFLAGS

LIBMPDEC_LIBS

针对 libmpdec 的 C 编译器和链接器旗标，由 decimal 模块使用，覆盖 pkg-config。

i 备注

除非指定了 `--with-system-libmpdec` 否则这些环境变量将没有效果。

LIBLZMA_CFLAGS

LIBLZMA_LIBS

针对 liblzma 的 C 编译器和链接器旗标，由 lzma 模块使用，覆盖 pkg-config。

LIBREADLINE_CFLAGS

LIBREADLINE_LIBS

针对 libreadline 的 C 编译器和链接器旗标，由 readline 模块使用，覆盖 pkg-config。

LIBSQLITE3_CFLAGS

LIBSQLITE3_LIBS

针对 libsqlite3 的 C 编译器和链接器旗标，由 sqlite3 模块使用，覆盖 pkg-config。

LIBUUID_CFLAGS

LIBUUID_LIBS

针对 libuuid 的 C 编译器和链接器旗标，由 uuid 模块使用，覆盖 pkg-config。

PANEL_CFLAGS

PANEL_LIBS

针对 PANEL 的 C 编译器和链接器旗标，覆盖 pkg-config。

针对 libpanel 或 libpanelw 的 C 编译器和链接器旗标，由 curses.panel 模块使用，覆盖 pkg-config。

TCLTK_CFLAGS**TCLTK_LIBS**

针对 TCLTK 的 C 编译器和链接器旗标，覆盖 pkg-config。

ZLIB_CFLAGS**ZLIB_LIBS**

针对 libzlib 的 C 编译器和链接器旗标，由 gzip 模块使用，覆盖 pkg-config。

3.3.5 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。

Added in version 3.11.

--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。

Added in version 3.11.

--enable-wasm-pthreads

为 WASM 启用 pthreads 支持。

Added in version 3.11.

3.3.6 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）。

Added in version 3.10.

--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。

Added in version 3.6.

3.3.7 性能选项

为获得最佳性能推荐使用 `--enable-optimizations --with-lto` (PGO + LTO) 来配置 Python。试验性的 `--enable-bolt` 旗标也可被用来提升性能。

`--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`。

备注

在构建期间, 你可能会遇到编译器警告提示某些源文件的配置数据不可用。这些警告是无害的, 因为在获取配置数据时只有一部分代码会被使用。要在 Clang 上禁用这些警告, 可通过在 `CFLAGS` 中添加 `-Wno-profile-instr-unprofiled` 来手动抑制它们。

Added in version 3.6.

在 3.10 版本发生变更: 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}`。

Added in version 3.8.

在 3.13 版本发生变更: 任务失败将不会再被静默地忽略。

`--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`)。

Added in version 3.6.

Added in version 3.11: 要使用 ThinLTO 特性, 请在 Clang 上使用 `--with-lto=thin`。

在 3.12 版本发生变更: 如果编译器支持将使用 ThinLTO 旗标作为 Clang 上的默认优化策略。

`--enable-bolt`

允许启用 BOLT 链接后二进制优化器 (默认为禁用)。

BOLT 是 LLVM 项目的一部分但并不总是包括在其二进制分发包中。该旗标要求 `llvm-bolt` 和 `merge-fdata` 可用。

BOLT 仍然是一个相当新的项目因此目前该旗标应当被视为是试验性的。因为此工具是作用于机器码所以其成功依赖于构建环境 + 其他优化配置参数 + CPU 架构的组合, 并且并非所有组合都受到支持。已知 LLVM 16 之前的 BOLT 版本在某些场景下会使得 BOLT 发生崩溃。强烈建议使用 LLVM 16 或更新版本进行 BOLT 优化。

`BOLT_INSTRUMENT_FLAGS` 和 `BOLT_APPLY_FLAGS` **configure** 变量可被定义为覆盖 `llvm-bolt` 的默认参数集合来分别指示和将 BOLT 数据应用于二进制代码中。

Added in version 3.12.

`BOLT_APPLY_FLAGS`

当创建 BOLT 优化的二进制文件时传给 `llvm-bolt` 的参数。

Added in version 3.12.

BOLT_INSTRUMENT_FLAGS

当构建二进制文件时传给 `llvm-bolt` 的参数。

Added in version 3.12.

--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-mimalloc

禁用快速的 `mimalloc` 分配器（默认为启用）。

参见环境变量 `PYTHONMALLOC`。

--without-pymalloc

禁用特定的 Python 内存分配器 `pymalloc`（默认为启用）。

参见环境变量 `PYTHONMALLOC`。

--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）。

--with-strict-overflow

将 `-fstrict-overflow` 添加到 C 编译器旗标（在默认情况下我们将添加 `-fno-strict-overflow` 来代替）。

3.3.8 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`。
- 添加 `-d` 命令行选项和 `PYTHONDEBUG` 环境变量用于调试解析器。
- 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 版本发生变更: 发布构建版和调试构建版现在是 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` 选项)。

3.3.9 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`。

`PYTHONDUMPREFS` 环境变量可被用来转储在 Python 退出时仍然存活的对象和引用计数。

静态分配的对象将不会被追踪。

Added in version 3.8.

在 3.13 版本发生变更: 此构建版现在与发布构建版和调试构建版是 ABI 兼容的。

--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。

Added in version 3.6.

--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。

Added in version 3.6.

--with-address-sanitizer

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

Added in version 3.6.

--with-memory-sanitizer

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

Added in version 3.6.

--with-undefined-behavior-sanitizer

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

Added in version 3.6.

--with-thread-sanitizer

启用 ThreadSanitizer 数据竞争检测器, `tsan` (默认为否)。

Added in version 3.13.

3.3.10 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）。

Added in version 3.10.

3.3.11 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-libmpdec

使用已安装的 mpdecimal 库来构建 _decimal 扩展模块，参见 decimal 模块（默认是）。

Added in version 3.3.

在 3.13 版本发生变更：默认为使用已安装的 mpdecimal 库。

Deprecated since version 3.13, will be removed in version 3.15: mpdecimal 库源代码的副本将不再随 Python 3.15 一起分发。

参见

`LIBMPDEC_CFLAGS` 和 `LIBMPDEC_LIBS`。

--with-readline=readline|editline

为 readline 模块指定一个后端库。

- readline: 使用 readline 作为后端。
- editline: 使用 editline 作为后端。

Added in version 3.10.

--without-readline

不编译 readline 模块（默认会）。

不定义 HAVE_LIBREADLINE 宏。

Added in version 3.10.

--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。

Added in version 3.7.

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

设置 OpenSSL 库的运行时代目录（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*。

Added in version 3.10.

3.3.12 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.

Added in version 3.4.

Added in version 3.11: 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。

Added in version 3.9.

--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。

Added in version 3.7.

在 3.10 版本发生变更: 设置 `python` 和 *STRING* 也会把 TLS 1.2 设为最低版本的协议。

3.3.13 macOS 选项

参见 [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）。

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

Python 标准库包含已知的当提交给 macOS 和 iOS 应用商店进行发布时会触发自动检查工具错误的字符串。如果启用，该选项将应用已知可纠正应用商店合规性的补丁列表。也可以指定自定义补丁文件。在默认情况下将禁用此选项。

Added in version 3.13.

3.3.14 iOS 选项

参见 `iOS/README.rst`。

--enable-framework=INSTALLDIR

创建一个 Python 框架。不同于 macOS，指定安装路径的 `INSTALLDIR` 参数是强制性的。

--with-framework-name=FRAMEWORK

指定框架的名称（默认名称：Python）。

3.3.15 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

Added in version 3.11.

CONFIG_SITE=file

指向一个带有配置重载的的环境变量。

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
```

HOSTRUNNER

用于针对交叉编译主机平台的运行 CPython 的程序。

Added in version 3.11.

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
```

3.4 Python 构建系统

3.4.1 构建系统的主要文件

- `configure.ac` => `configure`;
- `Makefile.pre.in` => `Makefile` (由 `configure` 创建);
- `pyconfig.h` (由 `configure` 创建);
- `Modules/Setup`: 由 `Makefile` 使用 `Module/makesetup shell` 脚本构建的 C 扩展;

3.4.2 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 扩展是由 `Makefile` 构建的 (参见 `Modules/Setup`)。

3.4.3 主要 Makefile 目标

make

对于大部分情况来说, 当编译某段代码或从上游刷新你的签出内容之后重新构建时, 你需要做的就是执行 `make`, 它 (按照 `Make` 的语义) 将构建默认目标, 即在 `Makefile` 中定义的第一个目标。在传统上 (包括在 CPython 项目中) 这通常为 `all` 目标。`configure` 脚本将扩展一个 `autoconf` 变量 `@DEF_MAKE_ALL_RULE@` 来准确地描述 `make all` 将构建哪个目标。有如下三个选择:

- `profile-opt` (使用 `--enable-optimizations` 配置)
- `build_wasm` (使用 `--with-emscripten-target` 配置)
- `build_all` (不显式地使用任何其他配置)

根据最近的源文件更改, `Make` 将重新构建任何尚未被更新的目标 (对象文件和可执行文件), 包括在必要时再次运行 `configure`。不过源/目标的依赖项数量很多并且是手动维护的, 因此 `Make` 有时并没有所需的全部信息来正确地检测所有需要重新构建的目标。根据尚未被重新构建的目标的具体情况, 你可能会遇到许多问题。如果你有无法确定原理的构建或测试问题, `make clean` 和 `make` 应该能够解决大多数依赖问题, 代价则是会耗费更多的时间来构建。

make platform

构建 python 程序，但不构造标准库扩展模块。这将生成一个名为 platform 的文件，其中只包含一行描述构建平台详细信息的文本，例如 macosx-14.3-arm64-3.12 或 linux-x86_64-3.13。

make profile-opt

使用 profile-guided optimization (PGO) 构建 Python。你可以使用 `--enable-optimizations` 配置选项来使其成为 make 命令的默认目标（即对应 make all 或更简洁的 make 命令）。

make clean

移除已构建文件。

make distclean

在 make clean 所做的工作之外，还移除由配置脚本所创建的文件。再次构建之前将需要运行 configure。¹

make install

构建 all 目标并安装 Python。

make test

构建 all 目标并附带 `--fast-ci` 选项运行 Python 测试套件。相关变量：

- TESTOPTS: 额外的回归测试命令行选项。
- TESTPYTHONOPTS: 额外的 Python 命令行选项。
- TESTTIMEOUT: 超时限制（默认值：10 分钟）。

make buildbottest

这与 make test 类似，但会使用默认超时限制为 20 分钟的 `--slow-ci` 选项，而不是 `--fast-ci` 选项。

make regen-all

重新生成（几乎）所有的已生成文件。这包括（但不限于）字节码用例，以及解析器生成器文件。对于其余的已生成文件必须分别运行 make regen-stdlib-module-names 和 autoconf。

3.4.4 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：

¹ git clean -fdx 是“清理”你的签出内容的更激进方式。它将移除所有对 Git 来说未知的文件。当使用 git bisect 查找程序错误时，推荐在多次探查之间执行此命令来确保完整的全新构建。请谨慎使用，因为它将删除所有未签入 Git 的文件，包括你最新的、尚未提交的工作。

```
>>> 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。

宏 `PyAPI_FUNC()`、`PyAPI_DATA()` 和 `PyMODINIT_FUNC` 在 `Include/exports.h` 中的定义将因是否定义了 `Py_BUILD_CORE_MODULE` 宏而不同：

- 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`。

3.5 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。

3.5.1 预处理器的标志

CONFIGURE_CPPFLAGS

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

Added in version 3.6.

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 `-Iinclude_dir`。

`CPPFLAGS` 和 `LDFLAGS` 都需要包含 `shell` 的值以便能够使用环境变量中指定的目录构建扩展模块。

BASECPPFLAGS

Added in version 3.4.

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)`。

Added in version 3.2.

3.5.2 编译器标志

CC

C 编译器指令。

例如：`gcc -pthread`。

CXX

C++ 编译器指令。

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。一旦装好 Python 则当某个编译器旗标 不应成为 `CFLAGS` 的一部分时将可使用它 ([gh-65320](#))。

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。

Added in version 3.5.

COMPILEALL_OPTS

当在 `make install` 中构建 `PYC` 文件时传给 `compileall` 命令行的选项。默认值: `-j0`。

Added in version 3.12.

EXTRA_CFLAGS

而外的 `C` 编译器指令。

CONFIGURE_CFLAGS

变量 `CFLAGS` 的值传递给 `./configure` 脚本。

Added in version 3.2.

CONFIGURE_CFLAGS_NODIST

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

Added in version 3.5.

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`、

Added in version 3.7.

CCSHARED

用于构建共享库的编译器标志。

例如, `-fPIC` 在 `Linux` 和 `BSD` 上使用。

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`。

Added in version 3.5.

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)`。

Added in version 3.7.

PY_CORE_CFLAGS

默认为 `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE`。

Added in version 3.2.

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`。

Added in version 3.8.

PURIFY

Purify 命令。Purify 是一个内存调试程序。

默认为: 空字符串（不使用）。

3.5.3 链接器标志位

LINKCC

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

默认值: `$(PURIFY) $(CC)`。

CONFIGURE_LDFLAGS

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

避免指定 `C_FLAGS`，`LD_FLAGS` 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。

Added in version 3.2.

LD_FLAGS_NODIST

`LD_FLAGS_NODIST` 的使用方式与 `C_FLAGS_NODIST` 相同。一旦装好 Python 则当某个链接器旗标 不应成为 `LD_FLAGS` 的一部分时将可使用它 (gh-65320)。

特别地，`LD_FLAGS` 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 `LD_FLAGS` 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 `LD_FLAGS_NODIST` 的值传递给 `./configure` 脚本。

Added in version 3.8.

LD_FLAGS

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

`CPP_FLAGS` 和 `LD_FLAGS` 都需要包含 `shell` 的值以便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`。

LD_SHARED

构建一个共享库的命令。

默认为: `@LD_SHARED@ $(PY_LD_FLAGS)`。

BLD_SHARED

构建共享库 `libpython` 的命令。

默认为: `@BLD_SHARED@ $(PY_CORE_LD_FLAGS)`。

PY_LDFLAGS

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

PY_LDFLAGS_NODIST

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

Added in version 3.8.

PY_CORE_LDFLAGS

用于构建解释器对象文件的链接器标志。

Added in version 3.8.

备注

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

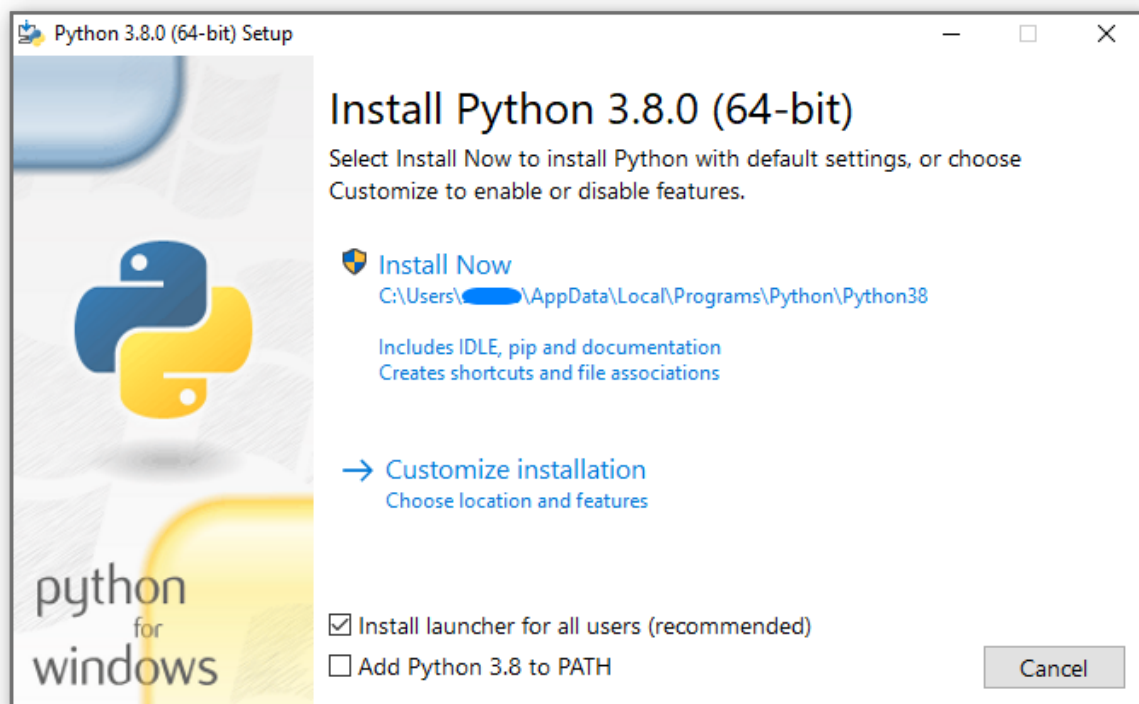
[可嵌入的包](#) 是 Python 的最小安装包，适合嵌入到更大的应用程序中。

4.1 完整安装程序

4.1.1 安装步骤

四个 Python 3.13 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[免下载安装](#)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：



如果选择 “Install Now（立即安装）”：

- 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
- 如果选择将安装目录将添加到 PATH
- 快捷方式仅对当前用户可见

选择“自定义安装”将允许您选择：要安装的功能、安装位置、其他选项或安装后的操作。如果要安装调试符号或二进制文件，您需要使用此选项。

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
- 快捷方式所有用户可用

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 版本发生变更: Python 中启用了对长路径的支持。

4.1.3 无 UI 安装

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。

下列选项（通过附带 `/?` 执行安装器来查看）可被传给安装器：

名称	描述
<code>/passive</code>	显示进度而无需用户交互
<code>/quiet</code>	安装/卸载时不显示任何 UI
<code>/simple</code>	防止用户定制
<code>/uninstall</code>	移除 Python（无需确认）
<code>/layout [directory]</code>	预下载所有组件
<code>/log [filename]</code>	指定日志记录文件位置

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	默认值
InstallAllUsers	为所有用户安装。	0
TargetDir	安装目录	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	仅为当前用户安装时的默认安装路径	%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	如果还安装了启动器, 则创建文件关联。	1
CompileAll	将所有 .py 文件编译为 .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 手册	1
Include_debug	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_launcher	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 setup-tools	1
Include_symbols	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
LauncherOrSimpleInstall	仅安装启动器。这将覆盖大多数其他选项。	0
SimpleInstall	禁用大多数安装 UI	0
SimpleInstallDescription	使用简化安装 UI 时显示的自定义消息。	(空)

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

```
python-3.9.0.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 免下载安装

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 [可选的目标目录]
```

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。

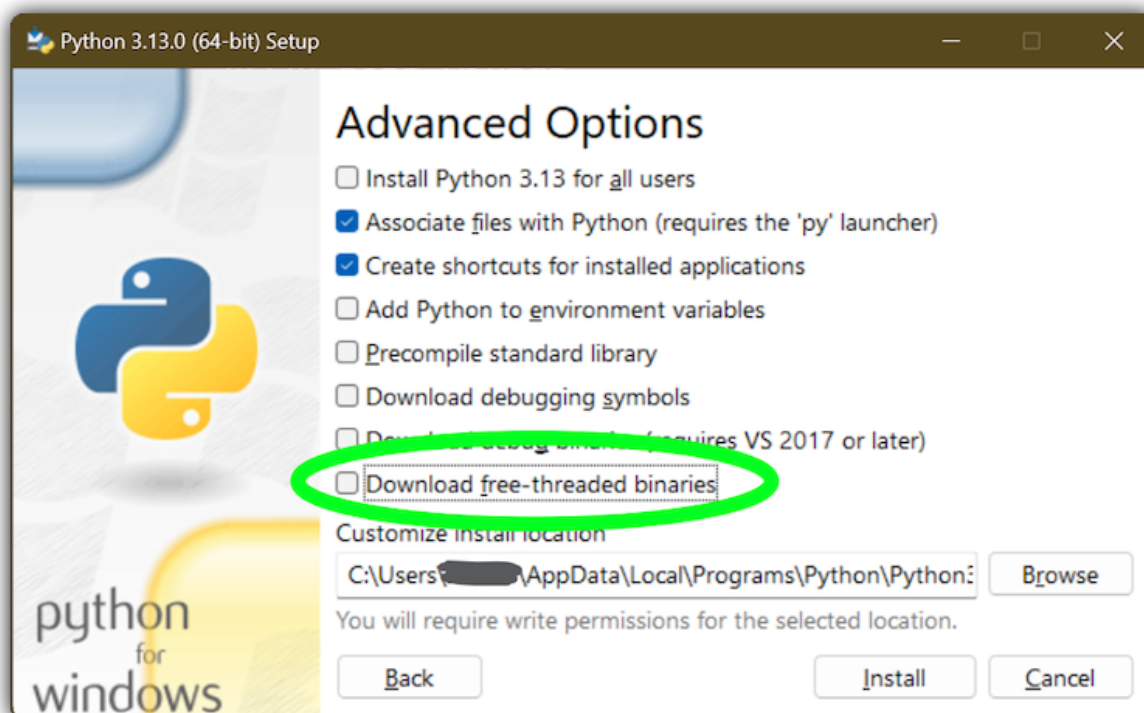
4.1.6 安装自由线程二进制文件

Added in version 3.13: （试验性功能）

备注

本节中描述的所有内容都是试验性的，它们预计会在未来的发布版中发生改变。

要安装启用了自由线程的预编译版二进制文件（参见 [PEP 703](#)），你应当选择“Customize installation”。在第二个选项页中包括了“Download free-threaded binaries”复选框。



选择此选项将下载并将额外的二进制文件安装到与 Python 主安装版本相同的目录下。主可执行文件的名称为 `python3.13t.exe`，而其他二进制文件将带有 `t` 前缀或完整的 ABI 前缀。Python 源文件和捆绑的第三方依赖将与主安装版本共享。

自由线程版将被注册为具有 `3.13t` 标签的常规 Python 安装版（并会按相应系统平台的惯例附带 `-32` 或 `-arm64` 后缀）。这使得各种工具能够找到它，并使得适用于 Windows 的 Python 启动器能够支持 `py.exe -3.13t`。请注意 `launcher` 会将 `py.exe -3`（或 `python3 shebang` 行）解读为“最新的 3.x 安装版”，这将使得自由线程版二进制文件优先于常规版，而 `py.exe -3.13` 则会使用常规版。如果你要使用简短风格的选项，那么目前你应该选择不安装自由线程版二进制文件。

要在命令行中指定安装选项，请使用 `Include_freethreaded=1`。请参阅[免下载安装](#)获取有关预先下载额外二进制文件供离线安装的指导。包括调试符号和二进制文件的选项也同样适用于自由线程构建版。

自由线程版二进制文件也可在 nuget.org 获取。

4.2 Microsoft Store 包

Added in version 3.7.2.

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 3.13”。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 3.13）。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，那么它将被作为 `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，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_8qbz5n2kfra8p0\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_8qbz5n2kfra8p0\\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。

有关这些限制的技术原理的更多细节，请查询 Microsoft 对已打包完全可信应用的文档，目前可在 docs.microsoft.com/en-us/windows/msix/desktop/desktop-to-uwp-behind-the-scenes 获取。

4.3 nuget.org 安装包

Added in version 3.5.2.

`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。

在 nuget.org 上的软件包信息页 www.nuget.org/packages/python 对应 64 位版本，www.nuget.org/packages/pythonx86 对应 32 位版本，www.nuget.org/packages/pythonarm64 对应 ARM64 版本

4.3.1 自由线程版软件包

Added in version 3.13: (试验性功能)

备注

本节中描述的所有内容都是试验性的，它们预计会在未来的发布版中发生改变。

包含自由线程版二进制文件的包名称 `python-freethreaded` 对应 64 位版本，`pythonx86-freethreaded` 对应 32 位版本，`pythonarm64-freethreaded` 对应 ARM64 版本。这些包同时包含 `python3.13t.exe` 和 `python.exe` 入口点，两者均在自由线程模式下运行。

4.4 可嵌入的包

Added in version 3.5.

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。

备注

嵌入式分发版不包括 **Microsoft C 运行时** 并由应用程序安装器负责提供此功能。运行时可能已经预先安装在用户的系统上或是通过 **Windows Update** 自动安装，并且可以通过在系统目录中查找 `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 部署管理器

“下一代的 Python 环境和包管理器”

之前 Enthought 提供了 Canopy，但已于 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 配置 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）。

备注

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。

参见

<https://learn.microsoft.com/windows/win32/procthread/environment-variables>

Windows 中的环境变量概述

https://learn.microsoft.com/windows-server/administration/windows-commands/set_1

用于临时修改环境变量的 `set` 命令

<https://learn.microsoft.com/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 版本发生变更。

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[命令行](#)文档。

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

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

4.7 UTF-8 模式

Added in version 3.7.

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。

您可以使用 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。

备注

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

Added in version 3.3.

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 入门

从命令行

在 3.6 版本发生变更。

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 3.13），您会注意到 Python 3.13 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' 不是内部或外部命令，也不是可运行的程序或批处理文件。
```

Tix 命令：

```
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`：

```
# 选择任意带 '3.*' 标签的运行时
py -V:3

# 选择任何 'PythonCore' 发行的运行时
py -V:PythonCore/

# 选择 PythonCore 的最新 Python 3 运行时
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 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连字符。这将允许 -v:3.1 匹配 3.1-32，但不匹配 3.10。Tag 是使用数字顺序进行排序的 (3.10 比 3.1 新)，但会按文本进行比较 (-v:3.01 将不匹配 3.1)。

从虚拟环境

Added in version 3.5.

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。

从文件关联

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 或激活的虚拟环境。因为在 Unix 上编写的许多 Python 脚本都已经有了这一行，你应该会发现这些脚本可以由启动器使用而无需修改。如果你在 Windows 上编写一个新脚本并希望其在 Unix 上可用，你应当使用某个以 /usr 开头的 shebang 行。

任何上述虚拟命令都可以附带一个显式版本号的后缀（可以是只有主版本号，也可以是有主版本号和次版本号）。此外还可以在次版本号之后添加“-32”来请求 32 位版本。即 /usr/bin/python3.7-32 将请求使用 32 位的 Python 3.7。如果激活了一个虚拟环境，则将忽略版本号并使用激活的环境。

Added in version 3.7: 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 版本发生变更：“-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。

在 3.13 版本发生变更：引用了 python 的虚拟命令现在会优先使用激活的虚拟环境再去搜索 PATH。这是为了处理 shebang 指定了 /usr/bin/env python3 但激活的环境中没有 python3.exe 的情况。

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。

示例：

- 如果没有设置相关选项，命令 `python` 和 `python2` 将使用安装的最新 Python 2.x 版本，命令 `python3` 将使用最新安装的 Python 3.x。
- 命令 `python3.7` 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 按需安装

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
<code>RC_BAD_VENV_CFG</code>	107	找到了 <code>pyvenv.cfg</code> 但文件已损坏。
<code>RC_CREATE_PROCESS</code>	101	启动 Python 失败。
<code>RC_INSTALLING</code>	111	安装已启动，但命令需要在其完成后重新运行。
<code>RC_INTERNAL_ERROR</code>	109	未预期的错误。请报告程序错误。
<code>RC_NO_COMMANDLINE</code>	108	无法从操作系统获取命令行。
<code>RC_NO_PYTHON</code>	103	无法定位所请求的版本。
<code>RC_NO_VENV_CFG</code>	106	需要 <code>pyvenv.cfg</code> 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，如环境变量中所述，则接下来添加其条目。请注意，在 Windows 上，此变量中的路径必须用分号分隔，以区别于驱动器标识符中使用的冒号（`C:\` 等）。
- 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。

这一切的最终结果是：

- 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`。
- 如果你在可执行文件中加载 `python3.dll` 或 `python37.dll`，请在 `Py_InitializeFromConfig()` 之前显式地设置 `PyConfig.module_search_paths`。
- 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 版本发生变更：添加 `._pth` 文件支持并从 `pyvenv.cfg` 中移除了 `applocal` 选项。

在 3.6 版本发生变更：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。

自 3.6 版本弃用：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 编写的 `PyWin32` 模块是一组用于高级 Windows 专属支持的模块。这包括以下实用工具：

- `Component Object Model (COM)`
- Win32 API 调用
- 注册
- 事件日志
- `Microsoft Foundation Classes (MFC)` 用户接口

`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。

➡ 参见

Win32 How Do I...?

Tim Golden 著

Python and COM

David 和 Paul Boddie 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 在 Windows 上编译 Python

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](#)

在 macOS 上使用 Python

本文档旨在为您于 Mac 电脑上开始使用 Python 前提供您必须了解的 macOS 平台之特有行为。在运行了 macOS 的 Mac 电脑上使用 Python 与在其他 Unix 家族平台电脑上面使用 Python 的体验非常相近，但在 Python 安装及部分特性上仍存有些许差异。

有多种途径可为 macOS 获取 Python。多家软件分发源均提供最新版 Python 的预编译版本。本文档的多数内容描述的是由 CPython 发布团队于 python.org 网站中提供下载的 Python 版本。参见 [Alternative Distributions](#) 获取其他选择。

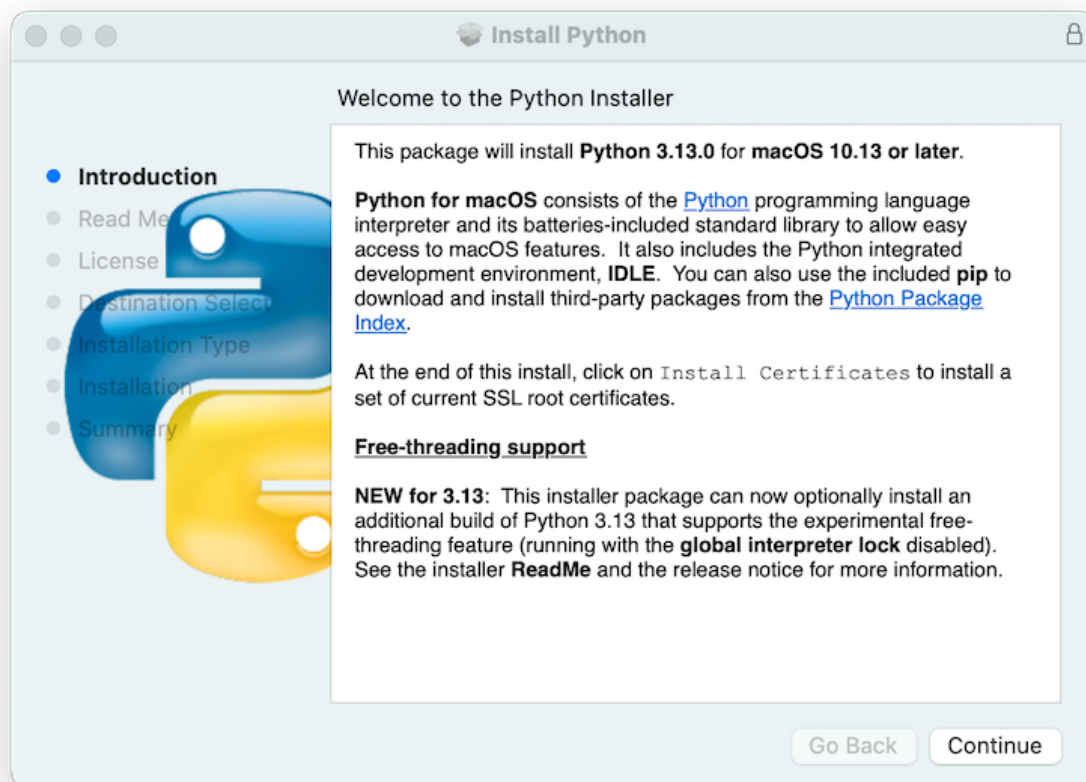
5.1 使用来自 python.org 的 macOS 版 Python

5.1.1 安装步骤

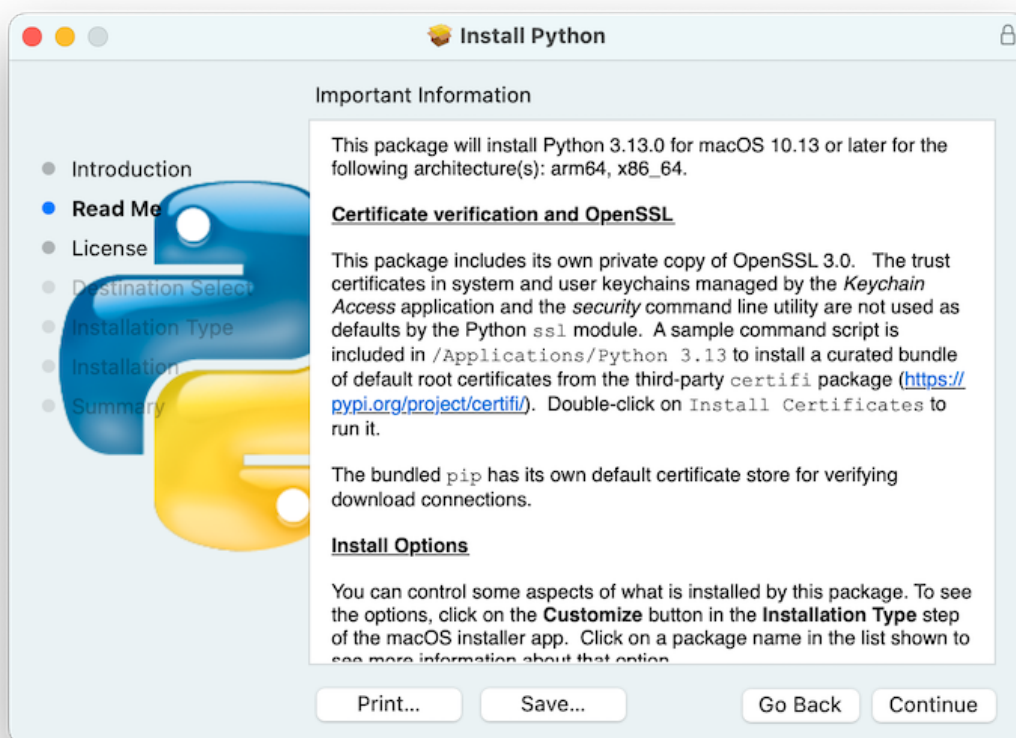
For [current Python versions](#) (other than those in `security status`), the release team produces a **Python for macOS** installer package for each new release. A list of available installers is available [here](#). We recommend using the most recent supported Python version where possible. Current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

您下载的文件是一个标准的 macOS 安装器包文件（.pkg）。我们为每个文件都提供用于校验文件完整性的资料（校验码、文档大小、签名等），随附于下载页面中。安装器包及其内容均使用 Python Software Foundation 的 Apple 开发者 ID 证书签名及公证，符合 macOS 门禁（Gatekeeper）要求。

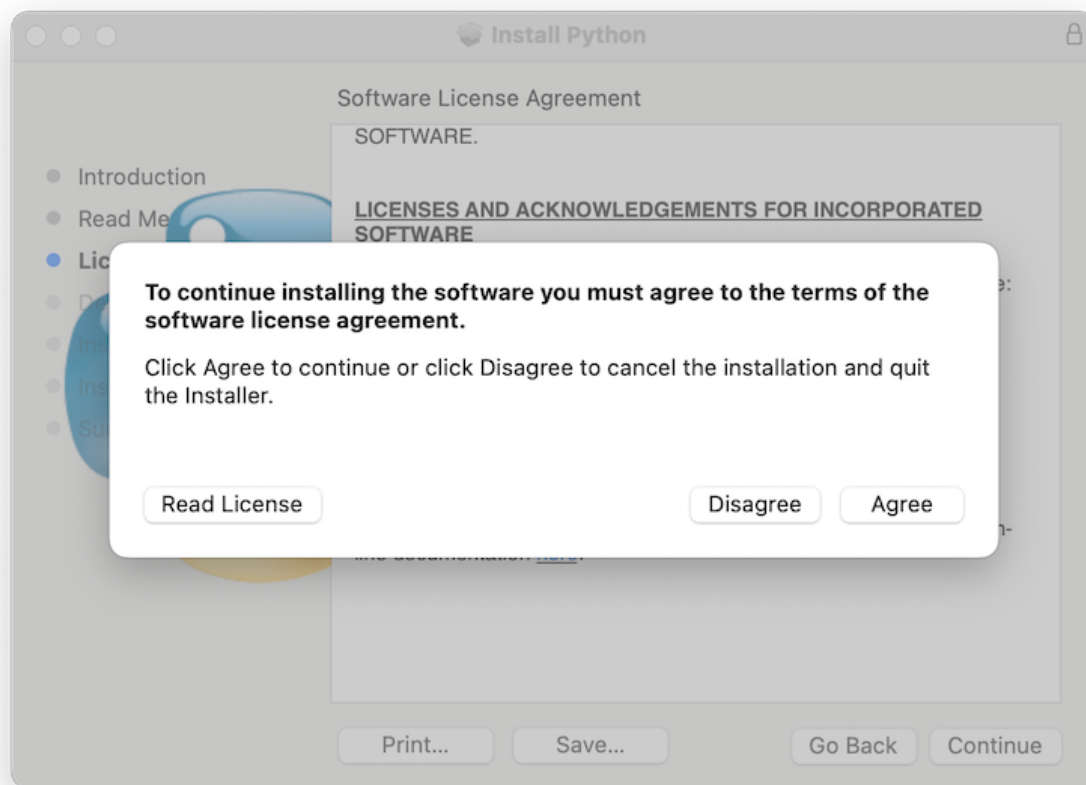
如欲使用默认预设安装配置，请直接双击您所下载的安装包文件。这将开启 macOS 内置的标准安装器软件，窗口中会显示安装步骤的第一步。



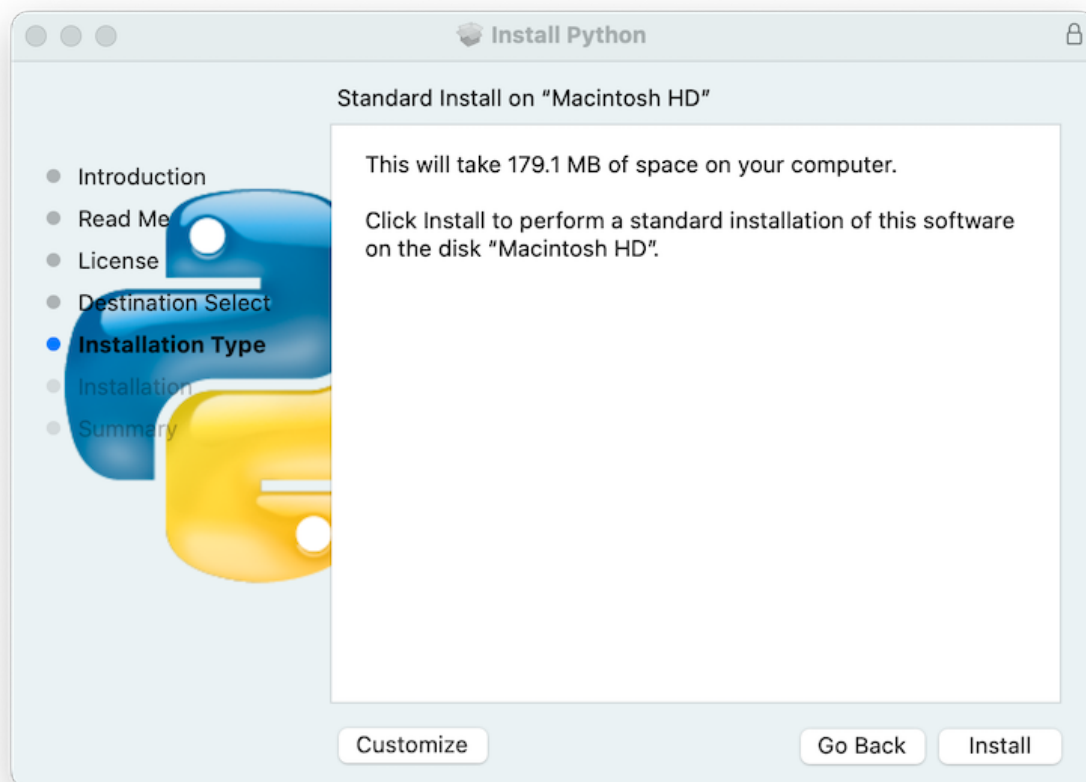
点击 **继续** 按钮后会开启安装包中自带的 **Read Me** 文档。文档里面有一些重要内容，但也记录了您将要安装的 Python 版本，此外也记录了本软件所支持的 macOS 版本。您可能需要滚动窗口才能阅读完整的文件。在默认配置下，该 **Read Me** 文档也会安装在 `/Applications/Python 3.13/` 路径中以供随时阅读。



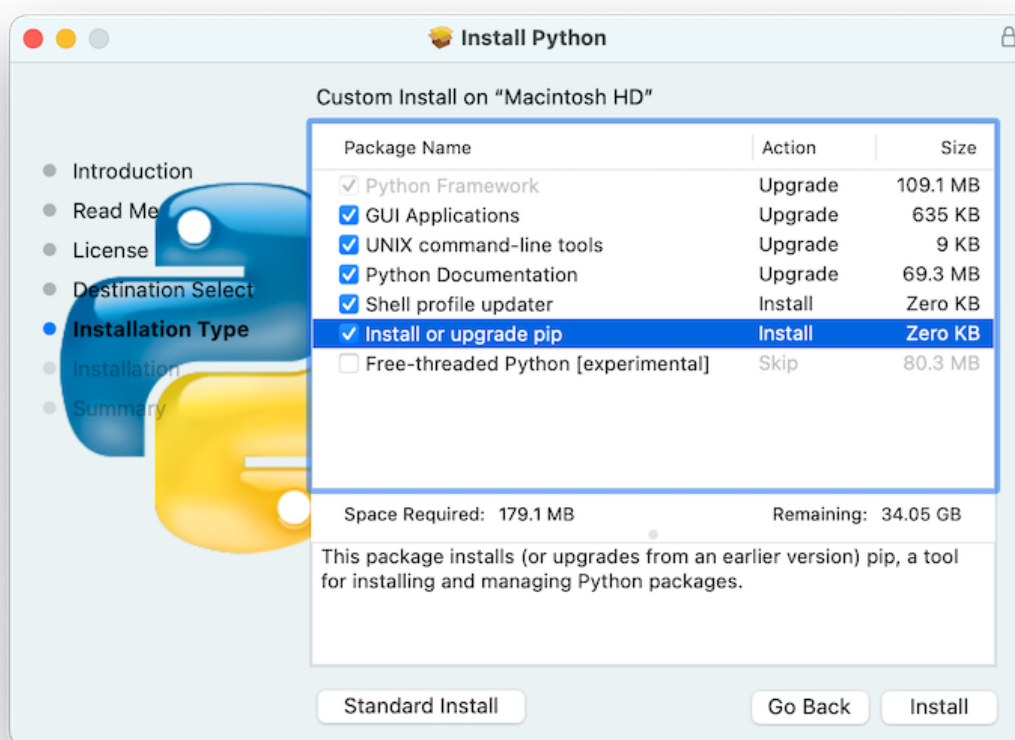
点击 **继续** 按钮将会显示 Python 软件及其他随附软件的授权许可。你需要选择 **同意** 该许可证才能继续下一步安装。同样，许可证日后也会安装在磁盘中以供随时阅读。



接受许可证条款后，下一步是选择您的安装类型。在大多数情况下，默认的标准安装操作就是最适合您的安装类型。



若您点击左下角的 **自定义** 按钮，您可以选择或取消选择安装包当中的部分组件。点击列表中的每一个项目可以了解该组件包会安装什么东西。若您想安装实验性可选的自由线程特性，请参见[安装自由线程二进制文件](#)。

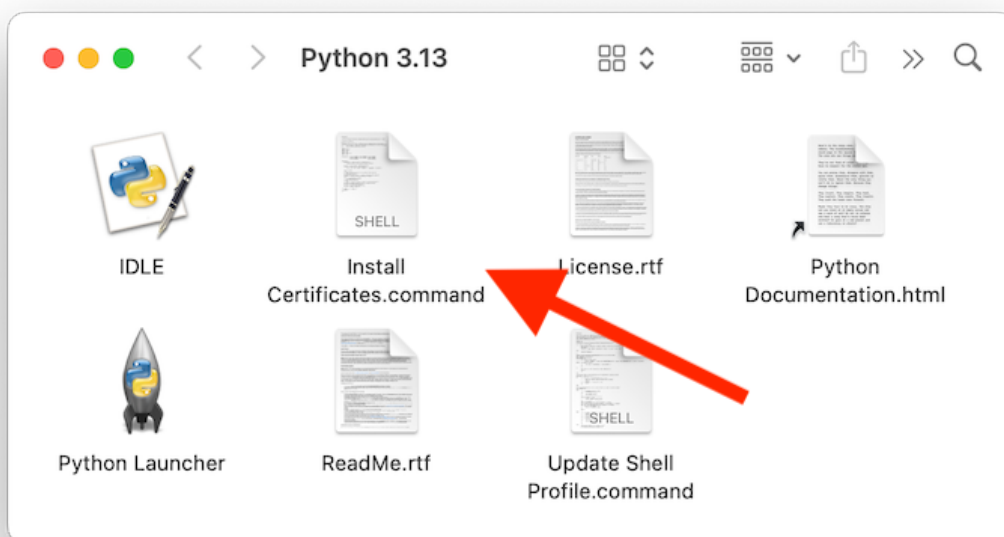


不论您选择接受默认安装配置还是自定义安装配置，点击 **安装** 按钮即可开始安装过程。您可能被要求提供安装软件所需的必要权限。您需要提供拥有 Administrator（管理员）权限的 macOS 用户名和密码，因为 Python 会面向电脑中的所有用户安装。

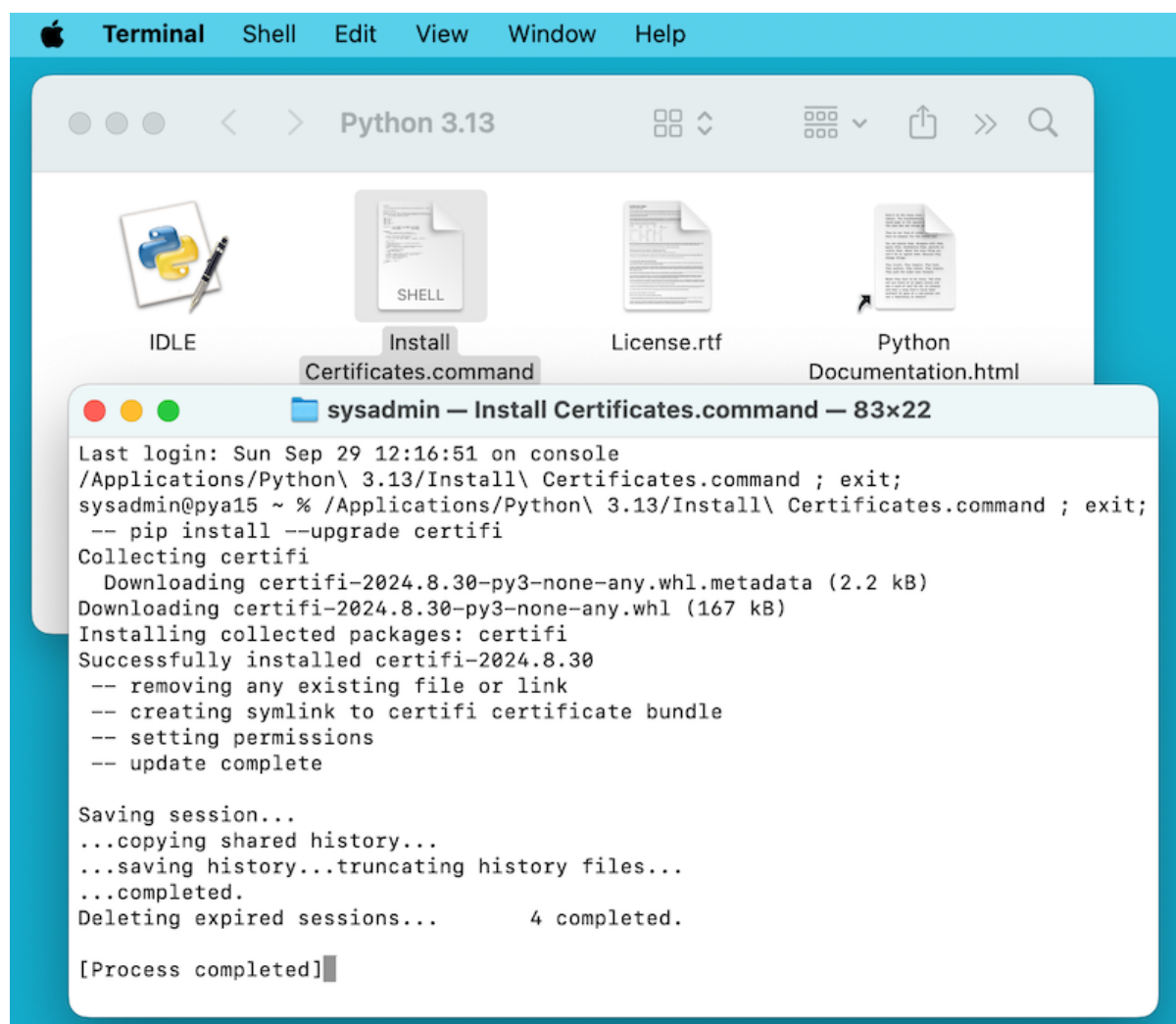
安装完成后会显示 **概览** 窗口。



浏览至 `/Applications/Python 3.13/` 文件夹窗口，双击 `Install Certificates.command` 图标或文件以完成最后的安装步骤。



这会开启一个临时的 `Terminal` 命令行窗口，使用新安装的 Python 下载并安装 SSL 根证书供其使用。



如果 `Successfully installed certifi` 和 `update complete` 字样出现在了终端的窗口中，那么安装就顺利完成。您可以关闭该终端窗口和安装器窗口。

使用默认安装配置会安装下列内容：

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

i 备注

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

5.1.2 如何运行 Python 脚本

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

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

There are many other editors and IDEs available, see [编辑器和集成开发环境](#) for more information.

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

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

要从 Finder（访达）运行你的脚本，你有两种选择：

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

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

5.2 Alternative Distributions

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

ActivePython

具有多平台兼容性的安装器，文档

Anaconda

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

Homebrew

针对 macOS 的包管理器包括多个 Python 版本和许多基于 Python 的第三方包（包括 `numpy`, `scipy` 和 `pandas`）。

MacPorts

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

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

5.3 安装额外的 Python 包

请参阅 [Python Packaging User Guide](#) 了解详情。

5.4 GUI 编程

使用 Python 在 Mac 上构建 GUI 应用程序有多种选择。

标准的 Python GUI 工具包是 `tkinter`，基于跨平台的 Tk 工具包 (<https://www.tcl.tk>)。安装器包括了一个 macOS 原生版本的 Tk。

PyObjC 是一个针对 Apple 的 Objective-C/Cocoa 框架的 Python 绑定。有关 *PyObjC* 的信息可从 [pyobjc](#) 获取。有多个替代性的 macOS GUI 工具包可供使用，包括：

- *PySide*: Qt GUI 工具包的官方 Python 绑定。
- *PyQt*: Qt 的另一款 Python 绑定。
- *Kivy*: 一款支持桌面和移动平台的跨平台 GUI 工具包。
- *Toga*: Part of the [BeeWare Project](#); supports desktop, mobile, web and console apps.
- *wxPython*: A cross-platform toolkit that supports desktop operating systems.

5.5 进阶

5.5.1 安装自由线程二进制文件

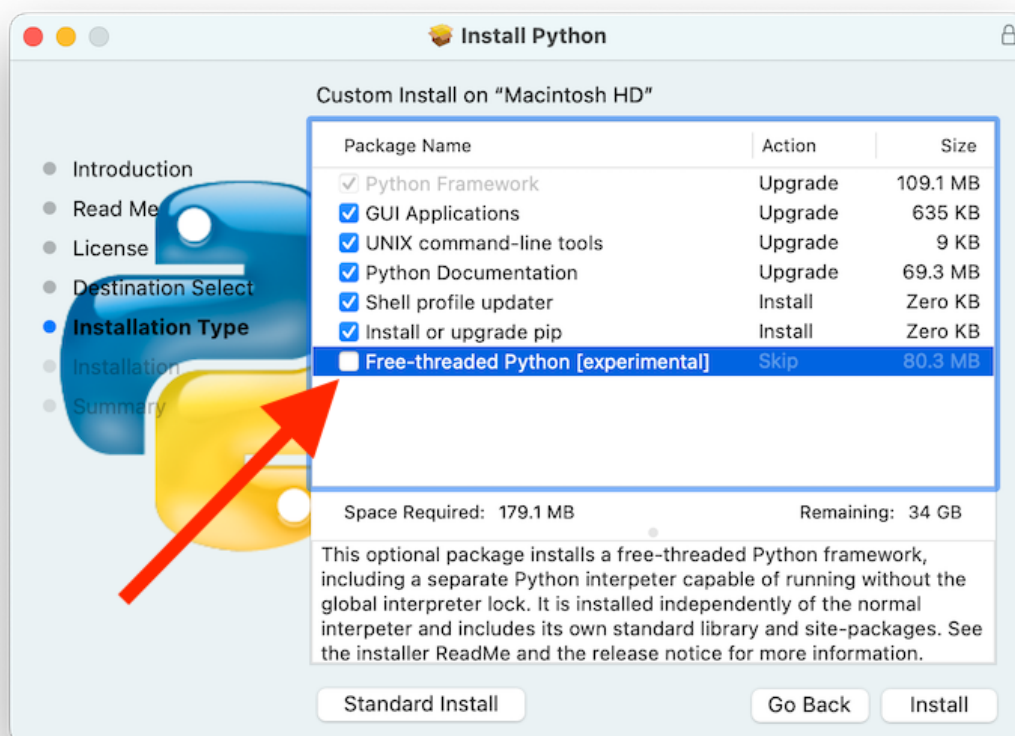
Added in version 3.13: (试验性功能)

备注

本节中描述的所有内容都是试验性的，它们预计会在未来的发布版中发生改变。

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

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



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

已知的注意事项和限制：

- The **UNIX command-line tools** package, which is selected by default, will install links in `/usr/local/bin` for `python3.13t`, the free-threaded interpreter, and `python3.13t-config`, a configuration utility which may be useful for package builders. Since `/usr/local/bin` is typically included in your shell `PATH`, in most cases no changes to your `PATH` environment variables should be needed to use `python3.13t`.
- For this release, the **Shell profile updater** package and the `Update Shell Profile.command` in `/Applications/Python 3.13/` do not support the free-threaded package.
- The free-threaded build and the traditional build have separate search paths and separate `site-packages` directories so, by default, if you need a package available in both builds, it may need to be installed in both. The free-threaded package will install a separate instance of **pip** for use with `python3.13t`.

– 不带 **venv** 地使用 **pip** 来安装软件包：

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

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

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

然后执行 **activate**。

- 要运行 IDLE 的自由线程版本：

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

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

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

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

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

5.5.2 使用命令行安装

If you want to use automation to install the `python.org` installer package (rather than by using the familiar macOS **Installer** GUI app), the macOS command line **installer** utility lets you select non-default options, too. If you are not familiar with **installer**, it can be somewhat cryptic (see **man installer** for more information).

As an example, the following shell snippet shows one way to do it, using the 3.13.0b2 release and selecting the free-threaded interpreter option:

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

# 下载安装器 pkg
curl -O https://www.python.org/ftp/python/3.13.0/${RELEASE}

# 创建安装器 choicechanges 文件来定制安装:
#   启用 PythonTFramework-3.13 包
#   并接受其他默认选项 (安装所有其他包)
cat > ./choicechanges.plist <<EOF
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-
1.0.dtd">
<plist version="1.0">
<array>
  <dict>
    <key>attributeSetting</key>
    <integer>1</integer>
    <key>choiceAttribute</key>
    <string>selected</string>
    <key>choiceIdentifier</key>
    <string>org.python.Python.PythonTFramework-3.13</string>
  </dict>
</array>
</plist>
EOF

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

接下来你可以这样测试两个安装器构建版现在是否可用:

```
$ # 当 Unix Command Tools 包被启用时测试自由线程解释器是否已安装
$ /usr/local/bin/python3.13t -VV
Python 3.13.0b2 experimental free-threading build (v3.13.0b2:3a83b172af, Jun  5 2024, 12:57:31) [Clang 15.0.0 (clang-1500.3.9.4)]
$ # 并测试传统解释器
$ /usr/local/bin/python3.13 -VV
Python 3.13.0b2 (v3.13.0b2:3a83b172af, Jun  5 2024, 12:50:24) [Clang 15.0.0 (clang-1500.3.9.4)]
$ # 当 /usr/local/bin 在 $PATH 中时测试它们在不带前缀的情况下是否可用
$ python3.13t -VV
Python 3.13.0b2 experimental free-threading build (v3.13.0b2:3a83b172af, Jun  5 2024, 12:57:31) [Clang 15.0.0 (clang-1500.3.9.4)]
$ python3.13 -VV
Python 3.13.0b2 (v3.13.0b2:3a83b172af, Jun  5 2024, 12:50:24) [Clang 15.0.0 (clang-1500.3.9.4)]
```

备注

当前的 python.org 安装器只会安装到固定位置如 /Library/Frameworks/, /Applications 和 /usr/local/bin。你不能使用 **installer** -domain 选项来安装到其他位置。

5.5.3 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 捆绑包, 并能管理签名和公证

的跨平台打包工具。

- **PyInstaller**: 一款可创建单独文件或文件夹作为可分发包的跨平台打包工具。

5.5.4 App Store 合规性

在 macOS App Store 中提交发布的 app 必须通过 Apple 的 app 审核进程。此进程包括一组在所提交的应用程序包中自动检查有问题代码的验证规则。

Python 标准库包含了一些已知会违反这些自动规则的代码。虽然这些违规情况看来是属于误报，但 Apple 的审核规则是不可挑战的。因此，有必要修改 Python 标准库以便 app 能够通过 App Store 的审核。

Python 源代码树包含一个补丁文件可移除所有已知的会导致 App Store 审核过程出现问题的代码。这个补丁会在 CPython 配置了 `--with-app-store-compliance` 选项时自动应用。

这个补丁对于在 Mac 上使用 CPython 并不是必需的；如果你是在 macOS App Store 以外的地方发布 app 它也不是必需的。它只有在你使用 macOS App Store 作为发布渠道时才是必需的。

5.6 其他资源

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

在 Android 上使用 Python

Python 在 Android 上与桌面平台上不同。在桌面平台上，通常是作为系统资源安装的，该计算机的任何用户都可以使用。然后，用户通过运行 `python` 可执行文件并交互提示器中输入命令，或运行脚本。

在安卓系统中，没有将安装作为系统资源的概念。软件分发的唯一单位是“应用程序”。也没有可以运行 `python` 可执行文件或 Python REPL 交互的控制台。

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

Python 标准库在 Android 上有一些明显的遗漏和限制。详情参见 API 可用性指南。

6.1 添加 Python 到 Android app

只有当您打算自己编译 Python for Android 时，才需要使用这些指令。大多数用户应该 不需要这样做。相反，使用以下工具之一会带来更轻松的体验：

- 来自 BeeWare 项目的 [Briefcase](#)
- 来自 Kivy 项目的 [Buildozer](#)
- [Chaquopy](#)
- [pyqtdeploy](#)
- [Termux](#)

如果您确定要手动完成所有这些操作，请继续阅读。您可以使用 [testbed app](#) 作为指南；下面的每个步骤都包含相关文件的链接。

- 构建 Python 请按照 [Android/README.md](#) 中的指令进行操作。
- 在您的 `build.gradle` 文件中添加代码，将以下项目复制到您的项目中。除您自己的 Python 代码外，其他代码均可从 `cross-build/HOST/prefix/lib` 复制：
 - 在 JNI 库中：

```
* libpython*.so
```

```
* lib*_python.so (外部库，如 OpenSSL)
```
 - 在您的资源文件中：

* python*. * (Python 标准库)

* python*/site-packages (您自己的 Python 代码)

- 在应用程序中添加代码以提取资源文件到文件系统。
- 在应用程序中添加代码以嵌入模式启动 Python。这需要通过 JNI 调用 C 代码。

在 iOS 上使用 Python

作者

Russell Keith-Magee (2024-03)

在 iOS 上的 Python 不同于桌面平台上的 Python。在桌面平台上，Python 通常是作为系统资源安装并可供该计算机上的任何用户使用。用户将通过运行 `python` 可执行文件并在交互提示符下输入命令，或者通过运行 Python 脚本的方式与 Python 进行交互。

在 iOS，不存在作为系统资源安装的概念。唯一的软件分发单元是“app”。也不存在可以让你运行 `python` 可执行文件，或与 Python REPL 进行交互的控制台。

因此，你在 iOS 上使用 Python 的唯一方式是嵌入模式——也就是说，通过编写原生的 iOS 应用，并使用 `libPython` 嵌入一个 Python 解释器，然后使用 Python 嵌入式 API 来发起调用 Python 代码。完整的 Python 解释器、标准库，以及所有 Python 代码都将被打包为可通过 iOS App Store 发布的独立软件包。

如果你想要上手尝试以 Python 来编写 iOS app，像 [BeeWare](#) 和 [Kivy](#) 这样的项目将提供更方便的用户体验。这些项目能够管理支持 iOS 项目运行的相关复杂问题，这样你只需要处理 Python 代码本身。

7.1 iOS 上的 Python 运行时

7.1.1 iOS 版本兼容性

受支持的最低 iOS 版本是在编译时指定的，对 `configure` 使用 `--host` 选项。在默认情况下，当针对 iOS 编译时，Python 编译将设置受支持的最低 iOS 版本为 13.0。我使用不同的最低 iOS 版本，请将版本号作为 `--host` 参数的一部分提供——例如，`--host=arm64-apple-ios15.4-simulator` 将编译一份部署目标为 15.4 的 ARM64 模拟器构建版。

7.1.2 平台识别

当在 iOS 上执行时，`sys.platform` 将报告为 `ios`。无论 app 是在模拟器还是物理设备上运行，都将在 iPhone 或 iPad 上返回该值。

有关特定运行时环境的信息，包括 iOS 版本、设备型号以及设备是否为模拟器，可使用 `platform.ios_ver()` 来获取。`platform.system()` 将根据具体设备报告为 `iOS` 或 `iPadOS`。

`os.uname()` 报告内核等级的详情；它将报告系统名称 `Darwin`。

7.1.3 标准库可用性

Python 标准库在 standard library has some notable omissions and restrictions on iOS 上有一点需要注意的缺失和限制。请参阅 针对 iOS 的 API 可用性指南 了解详情。

7.1.4 二进制扩展模块

作为一个平台的 iOS 与别家的显著不同之处在于 App Store 分发机制对应用的打包方式设置了硬性要求。其中一项要求规定了应当如何分发二进制扩展模块。

iOS App Store 要求 iOS app 中的 所有二进制模块都必须为动态库，包含在具有适当元数据的框架中，保存于被打包 app 的 Frameworks 文件夹下。每个框架只能有一个二进制模块，而在 Frameworks 文件夹之外不能有可执行的二进制文件。

这与 Python 通常用于发布二进制文件的方式存在冲突，此方式允许从 `sys.path` 上的任何位置加载二进制扩展模块。为确保符合 App Store 的政策，iOS 项目必须对任何 Python 包进行后期处理，将 `.so` 二进制模块转换为具有适当元数据和签名的单个独立框架。有关如何执行后期处理的更多细节，请参阅将 *Python 添加到你的项目* 指南。

为帮助 Python 在新位置中发现二进制文件，在 `sys.path` 中的原始 `.so` 文件将替换为 `.framework` 文件。此文件是一个包含框架二进制文件相对于 app 捆绑包位置的文本文件。为允许框架将其解析回原始位置，框架必须包含一个 `.origin` 文件，其中包含 `.framework` 相对对 app 捆绑包的位置。

例如，考虑导入 `from foo.bar import _whiz` 的情况，其中 `_whiz` 是使用二进制模块 `sources/foo/bar/_whiz.abi3.so` 实现的，这里 `sources` 是在 `sys.path` 中注册的相对于 app 包的位置。此模块必须发布为 `Frameworks/foo.bar._whiz.framework/foo.bar._whiz` (根据模块的完整导入路径创建框架名称)，并通过 `.framework` 目录中的 `Info.plist` 文件将二进制文件标识为一个框架。`foo.bar._whiz` 模块在原始位置中以一个 `sources/foo/bar/_whiz.abi3.framework` 标记文件来代表，其中包含路径 `Frameworks/foo.bar._whiz.framework/foo.bar._whiz`。该框架还要包含 `Frameworks/foo.bar._whiz.framework/foo.bar._whiz.origin`，其中包含 `.framework` 文件的路径。

当在 iOS 上运行时，Python 解释器将安装一个能够读取并导入 `.framework` 文件的 `AppleFrameworkLoader`。一旦被导入，该二进制模块的 `__file__` 属性将报告为 `.framework` 文件的位置。不过，被加载模块的 `ModuleSpec` 则会将 `origin` 报告为框架文件夹下的二进制文件的位置。

7.1.5 编译器存根二进制文件

Xcode 不会暴露针对 iOS 的显式编译器；作为替代，它会使用一个能解析出完整编译器路径的 `xcrun` 脚本 (例如，`xcrun --sdk iphoneos clang` 将获得针对 iPhone 设备的 `clang`)。不过，使用此脚本会导致两个问题：

- `xcrun` 的输出包括特定机器专属的路径，这导致无法在用户之间共享 `sysconfig` 模块；并且
- 它会导致包括空格的 `CC/CPP/LD/AR` 定义。有大量 C 生态系统工具假定你可以在第一个空格处拆分命令行来获取编译器可执行文件的路径；当使用 `xcrun` 这是不行的。

为避免这些问题，Python 提供了针对这些工具的程序段。这些代码段是在下层 `xcrun` 工具之上的 `shell` 脚本包装器，发布在随同已编译的 iOS 框架一起分发的 `bin` 文件夹中。这些脚本可以被重新调整位置，并且总是会解析到正确的本机系统路径。通过在配合框架的 `bin` 文件夹中包括这些脚本，`sysconfig` 模块的内容将可适用于要编译他们自己的模块的最终用户。在为 iOS 编译第三方 Python 模块时，你应当确保这些程序段二进制文件位于你的路径中。

7.2 在 iOS 上安装 Python

7.2.1 构建 iOS 应用程序的工具

针对 iOS 构建需要使用 Apple 的 Xcode 工具集。强烈建议你使用 Xcode 最新稳定发布版。这将需要使用最新（或者次新）发布的 macOS 版本，因为 Apple 不会为更旧的 macOS 版本维护 Xcode。Xcode 命令行工具对于 iOS 开发来说是不够的；你需要完整的 Xcode 安装版。

如果你想要在 iOS 模拟器上运行你的代码，你还需要安装 iOS 模拟器平台。当你首次运行 Xcode 应该会提示你选择一个 iOS 模拟器平台。或者，你也可以通过在 Settings 面板的 Platforms 选项卡中添加一个 iOS 模拟器平台。

7.2.2 在 iOS 项目中添加 Python

Python 可以被添加到任何 iOS 项目，不论它是使用 Swift 还是 Objective C。下面的例子将使用 Objective C；如果你是使用 Swift，你可能会发现 [PythonKit](#) 这样的库非常有用。

要在 iOS Xcode 项目中添加 Python：

1. 构建或是获取一个 Python XCFramework。请参阅 [iOS/README.rst](#) (在 CPython 源代码发布包中) 中的说明了解有关如何构建 Python XCFramework 的细节。要满足最基本要求，你需要一个支持 arm64-apple-ios，再加上 arm64-apple-ios-simulator 或 x86_64-apple-ios-simulator 之一的构建版。
2. 将 XCframework 拖到你的 iOS 项目中。在下面的说明中，我们将假定你已将 XCframework 放到你的项目的根目录下；不过，你也可以通过按需调整路径来使用任何其他你想要的位置。
3. 将 iOS/Resources/dylib-Info-template.plist 文件拖到你的项目中，并确保它已关联到 app 目标。
4. 将你的应用程序代码作为一个文件夹添加到你的 Xcode 项目中。在下面的说明中，我们将假定你的用户代码位于你的项目中名为 app 的文件夹；你也可以通过按需调整路径来使用任何其他位置。请确保该文件夹已关联到你的 app 目标。
5. 通过选择你的 Xcode 项目的根节点来选择 app 目标，然后在边栏中出现的目标名称进行选择。
6. 在“General”设置中，在“Frameworks, Libraries and Embedded Content”之下，增加了 Python.xcframework，并选中“Embed & Sign”。
7. 在“构建设置”选项卡中，修改以下内容：
 - 构建选项
 - 用户脚本沙盒：否
 - 启用可测试性：是
 - 搜索路径
 - 框架搜索路径：\$(PROJECT_DIR)
 - 头文件搜索路径：“\$(BUILT_PRODUCTS_DIR)/Python.framework/Headers”
 - Apple Clang - 警告 - 所有语言
 - 引用包含在框架头文件中：否
8. 添加一个用于将 Python 标准库拷贝到你的 app 中的构建步骤。在“Build Phases”选项卡中，在“Embed Frameworks”步骤之前，“Copy Bundle Resources”步骤之后添加一个新的“Run Script”构建步骤。将该步骤命名为“Install Target Specific Python Standard Library”，取消“Based on dependency analysis”复选框，并将脚本内容设为：

```
set -e

mkdir -p "$CODESIGNING_FOLDER_PATH/python/lib"

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

请注意 XCframework 中的模拟器名称“slice”可能有所不同，具体取决于你的 XCFramework 所支持的 CPU 架构。

9. 再添加第二个用于将标准库中的二进制扩展模块处理为“Framework”格式的构建步骤。紧随你在第 8 步添加的步骤之后立即添加一个“Run Script”构建步骤，命名为“Prepare Python Binary Modules”。它应当取消选择“Based on dependency analysis”，并将脚本内容设为：

```
set -e

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

    # 扩展文件的名称
    EXT=$(basename "$FULL_EXT")
    # 扩展文件相对与捆绑包的位置
    RELATIVE_EXT=${FULL_EXT#$CODESIGNING_FOLDER_PATH/}
    # 扩展文件相对于安装目录的路径
    PYTHON_EXT=${RELATIVE_EXT/$INSTALL_BASE/}
    # 扩展模块带点号的完整名称，根据文件目录来构造。
    FULL_MODULE_NAME=$(echo $PYTHON_EXT | cut -d "." -f 1 | tr "/" ".");
    # 捆绑包标识；未实际使用，但为 Xcode 框架打包所必需
    FRAMEWORK_BUNDLE_ID=$(echo $PRODUCT_BUNDLE_IDENTIFIER.$FULL_MODULE_NAME | tr "_" "-")
    # 框架文件夹的名称。
    FRAMEWORK_FOLDER="Frameworks/$FULL_MODULE_NAME.framework"

    # 如果框架文件夹不存在，则创建它。
    if [ ! -d "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER" ]; then
        echo "Creating framework for $RELATIVE_EXT"
        mkdir -p "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER"
        cp "$CODESIGNING_FOLDER_PATH/dylib-Info-template.plist" "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
        plutil -replace CFBundleExecutable -string "$FULL_MODULE_NAME" "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
        plutil -replace CFBundleIdentifier -string "$FRAMEWORK_BUNDLE_ID" "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
    fi

    echo "Installing binary for $FRAMEWORK_FOLDER/$FULL_MODULE_NAME"
    mv "$FULL_EXT" "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/$FULL_MODULE_NAME"
    # 创建占位用 .fwork 文件指明 .so 所在位置
    echo "$FRAMEWORK_FOLDER/$FULL_MODULE_NAME" > ${FULL_EXT%.so}.fwork
    # 在框架中创建指向 .so 文件位置的反向引用
    echo "${RELATIVE_EXT%.so}.fwork" > "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/$FULL_MODULE_NAME.origin"
}

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

# 清理 dylib 模板
rm -f "$CODESIGNING_FOLDER_PATH/dylib-Info-template.plist"

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

(续下页)

(接上页)

```
↩️ entitlements-der "{}" \;
```

10. 添加 Objective C 代码来初始化并使用嵌入模式的 Python 解释器。你应当确保：

- UTF-8 模式 (PyPreConfig.utf8_mode) 已启用；
- 带缓冲的 `stdio` (PyConfig.buffered_stdio) 已禁用；
- 写入字节码 (PyConfig.write_bytecode) 已禁用；
- 信号处理器 (PyConfig.install_signal_handlers) 已启用；
- 系统日志记录 (PyConfig.use_system_logger) 已启用 (可选项，但强烈建议启用)；
- 用于解释器的 `PYTHONHOME` 将被配置为指向你的 app 捆绑包的 `python` 子文件夹；并且
- 用于解释器的 `PYTHONPATH` 包括：
 - 你的 app 的封包的 `python/lib/python3.X` 子文件夹，
 - 你的 app 的封包的 `python/lib/python3.X/lib-dynload` 子文件夹，以及
 - 你的 app 的封包的 `app` 子文件夹

你的 app 捆绑包的位置可使用 `[[NSBundle mainBundle] resourcePath]` 来确定。

这些说明的步骤 8, 9 和 10 均假定你有一个单独的纯 Python 应用程序代码文件夹，其名称为 `app`。如果在你的 `app` 中还存在第三方二进制模块，则还需要一些额外的步骤：

- 你需要确保任何包含第三方二进制文件的文件夹均已关联到 `app` 目标，或已作为第 8 步的一部分被拷贝。第 8 步还应当清理任何对于特定构建目标的平台来说不适用的二进制文件（即当你要构建以该模拟器为目标的 `app` 就应删除任何设备二进制文件）。
- 任何包含第三方二进制文件的文件夹都必须通过第 9 步处理为框架形式。可以拷贝并调整对于处理 `lib-dynload` 文件夹的 `install_dylib` 发起调用的代码以符合这一目的。
- 如果你使用了不同的文件夹来存放第三方包，请确保该文件夹已在第 10 步中被包括为 `PYTHONPATH` 配置的一部分。

7.2.3 测试 Python 软件包

CPython 源码树包含一个 `testbed` 项目 用于在 iOS 模拟器上运行 CPython 测试套件。这个 `testbed` 还可被用作在 iOS 上运行你的 Python 库的测试套件的 `testbed` 项目。

在构建或获取 iOS XCFramework (请参见 [iOS/README.rst](#) 了解详情) 之后，通过运行以下命令创建 Python iOS `testbed` 项目的克隆：

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

你将需要将 `iOS/testbed` 引用修改为指向 CPython 源码树的相应目录；任何使用 `--app` 旗标指定的文件夹都将被拷贝到克隆的 `testbed` 项目中。结果 `testbed` 将在 `app-testbed` 文件夹中被创建。在本示例中，`module1` 和 `module2` 将成为运行时可导入的模块。如果你的项目还有额外的依赖项，它们可以被安装到 `app-testbed/iOSTestbed/app_packages` 文件夹中 (使用类似 `pip install --target app-testbed/iOSTestbed/app_packages` 的命令)。

你可以随后使用 `app-testbed` 文件夹来运行你的 app 的测试套件，举例来说，如果 `module1.tests` 是你的测试套件的入口点，你可以运行：

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

这相当于在桌面 Python 构建版上运行 `python -m module1.tests`。任何在 `--` 之后的参数都将被传给 `testbed` 就像它们是桌面机上 `python -m` 的参数那样。

你还可以通过运行以下命令在 Xcode 中打开这个 `testbed` 项目：

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

这将允许你使用完整的 Xcode 工具套件进行调试。

7.3 App Store 合规性

将 app 发布到第三方 iOS 设备的唯一机制是将 app 提交到 iOS App Store；提交发布的 app 必须通过 Apple 的 app 审核进程。此进程包括一组在所提交的应用程序包中自动检查有问题代码的验证规则。

Python 标准库包含了一些已知会违反这些自动规则的代码。虽然这些违规情况看来是属于误报，但 Apple 的审核规则是不可挑战的；因此，有必要修改 Python 标准库以便 app 能够通过 App Store 的审核。

Python 源代码树包含一个补丁文件可移除所有已知的会导致 App Store 审核过程出现问题的代码。这个补丁会在针对 iOS 进行构建时自动应用。

编辑器和集成开发环境

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

请访问 [Python Editors](#) 和 [Integrated Development Environments](#) 以获取完整列表。

术语对照表

>>>

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

...

具有以下含义：

- *interactive* shell 中输入特殊代码时默认的 Python 提示符，特殊代码包括缩进的代码块，左右成对分隔符（圆括号、方括号、花括号或三重引号等）之内，或是在指定一个装饰器之后。
- Ellipsis 内置常量。

abstract base class -- 抽象基类

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

annotation -- 标注

关联到某个变量、类属性、函数形参或返回值的标签，被约定作为 *类型注解* 来使用。

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

参见 *variable annotation*、*function annotation*、**PEP 484** 和 **PEP 526**，对此功能均有介绍。另请参见 `annotations-howto` 了解使用标注的最佳实践。

argument -- 参数

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

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

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

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

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

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

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

asynchronous context manager -- 异步上下文管理器

此种对象通过定义 `__aenter__()` 和 `__aexit__()` 方法来对 `async with` 语句中的环境进行控制。由 [PEP 492](#) 引入。

asynchronous generator -- 异步生成器

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

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

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

asynchronous generator iterator -- 异步生成器迭代器

[asynchronous generator](#) 函数所创建的对象。

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

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

asynchronous iterable -- 异步可迭代对象

一个可以在 `async for` 语句中使用的对象。必须通过它的 `__aiter__()` 方法返回一个 [asynchronous iterator](#)。由 [PEP 492](#) 引入。

asynchronous iterator -- 异步迭代器

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

attribute -- 属性

关联到一个对象的值，通常使用点号表达式按名称来引用。举例来说，如果对象 *o* 具有属性 *a* 则可以用 *o.a* 来引用它。

如果对象允许，将未被定义为 `identifiers` 的非标识名称用作一个对象的属性也是可以的，例如使用 `setattr()`。这样的属性将无法使用点号表达式来访问，而是必须通过 `getattr()` 来获取。

awaitable -- 可等待对象

一个可在 `await` 表达式中使用的对象。可以是 [coroutine](#) 或是具有 `__await__()` 方法的对象。参见 [PEP 492](#)。

BDFL

“终身仁慈独裁者”的英文缩写，即 [Guido van Rossum](#)，Python 的创造者。

binary file -- 二进制文件

[file object](#) 能够读写字节型对象。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。

另请参见 [text file](#) 了解能够读写 `str` 对象的文件对象。

borrowed reference -- 借入引用

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。

推荐在 *borrowed reference* 上调用 `Py_INCREF()` 以将其原地转换为 *strong reference*，除非是当该对象无法在借入引用的最后一次使用之前被销毁。`Py_NewRef()` 函数可以被用来创建一个新的 *strong reference*。

bytes-like object -- 字节型对象

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

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

bytecode -- 字节码

Python 源代码会被编译为字节码，即 CPython 解释器中表示 Python 程序的内部代码。字节码还会缓存在 `.pyc` 文件中，这样第二次执行同一文件时速度更快（可以免去将源码重新编译为字节码）。这种“中间语言”运行在根据字节码执行相应机器码的 *virtual machine* 之上。请注意不同 Python 虚拟机上的字节码不一定通用，也不一定能在不同 Python 版本上兼容。

字节码指令列表可以在 `dis` 模块的文档中查看。

callable -- 可调用对象

可调用对象就是可以执行调用运算的对象，并可能附带一组参数（参见 *argument*），使用以下语法：

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

function，还可扩展到 *method* 等，就属于可调用对象。实现了 `__call__()` 方法的类的实例也属于可调用对象。

callback -- 回调

一个作为参数被传入以用在未来的某个时刻被调用的子例程函数。

class -- 类

用来创建用户定义对象的模板。类定义通常包含对该类的实例进行操作的方法定义。

class variable -- 类变量

在类中定义的变量，并且仅限在类的层级上修改（而不是在类的实例中修改）。

closure variable -- 闭包变量

引用自 *nested scope* 的 *free variable*，它是在外层作用域中定义而不是在运行时自全局或内置命名空间中取得。可能使用 `nonlocal` 关键字显式地定义以允许写入访问，或者如果变量仅供读取则只需隐式地定义。

例如，在以下代码的 `inner` 函数中，`x` 和 `print` 均为自由变量，但只有 `x` 属于闭包变量：

```
def outer():
    x = 0
    def inner():
        nonlocal x
        x += 1
        print(x)
    return inner
```

由于 `codeobject.co_freevars` 属性的存在（虽然名称如此，但其仅包括闭包变量名称而非列出所有被引用的自由变量），更一般化的术语 *free variable* 有时在即便本意是专指闭包变量时也会被使用。

complex number -- 复数

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

context -- 上下文

此术语根据其所在场合和使用方式的不同而具有不同的含义。一些常见的含义为：

- 通过 `with` 语句由一个 *context manager* 所创建的临时环境或状态。
- 一组关联到特定 `contextvars.Context` 对象并通过 `ContextVar` 对象来访问的键值绑定。另请参见 *context variable*。
- 一个 `contextvars.Context` 对象。另请参见 *current context*。

上下文管理协议

`__enter__()` 和 `__exit__()` 方法将由 `with` 语句来调用。参见 [PEP 343](#)。

context manager -- 上下文管理器

一个实现了 *context management protocol* 并负责控制某个 `with` 语句内的环境的对象。参见 [PEP 343](#)。

context variable -- 上下文变量

一个具体值取决于哪个上下文是 *current context* 的变量。这些值是通过 `contextvars.ContextVar` 对象来访问的。上下文变量主要被用来隔离并发的异步任务之间的状态。

contiguous -- 连续

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

coroutine -- 协程

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

coroutine function -- 协程函数

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

CPython

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

current context -- 当前上下文

在当前被 `ContextVar` 对象用来访问（获取或设置）*上下文变量* 的值的 *context* (`contextvars.Context` 对象)。每个线程都具有它自己的当前上下文。用于执行异步任务（参见 *asyncio*）的框架会在每个任务开始或恢复执行时将任务关联到一个成为当前上下文的上下文。

decorator -- 装饰器

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

装饰器语法只是一种语法糖，以下两个函数定义在语义上完全等价：

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

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

同样的概念也适用于类，但通常较少这样使用。有关装饰器的详情可参见 *函数定义* 和 *类定义* 的文档。

descriptor -- 描述器

任何定义了 `__get__()`、`__set__()` 或 `__delete__()` 方法的对象。当一个类属性为描述器时，它的特殊绑定行为就会在属性查找时被触发。通常情况下，使用 *a.b* 来获取、设置或删除一个属性时会在 *a* 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。

有关描述器的方法的更多信息，请参阅 `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 -- 字典推导式

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

dictionary view -- 字典视图

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

docstring -- 文档字符串

作为类、函数或模块之内的第一个表达式出现的字符串字面值。它在代码块被执行时将被忽略，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 -- 鸭子类型

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

EAFP

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

expression -- 表达式

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

extension module -- 扩展模块

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

f-string -- f-字符串

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

file object -- 文件对象

对外公开面向文件的 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。

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

file-like object -- 文件型对象

[file object](#) 的同义词。

filesystem encoding and error handler -- 文件系统编码格式与错误处理器

Python 用来从操作系统解码字节串和向操作系统编码 Unicode 的编码格式与错误处理器。

文件系统编码格式必须保证能成功解码长度在 128 以下的所有字节串。如果文件系统编码格式无法提供此保证，则 API 函数可能会引发 `UnicodeError`。

`sys.getfilesystemencoding()` 和 `sys.getfilesystemencodeerrors()` 函数可被用来获取文件系统编码格式与错误处理器。

[filesystem encoding and error handler](#) 是在 Python 启动时通过 `PyConfig_Read()` 函数来配置的：请参阅 `PyConfig` 的 `filesystem_encoding` 和 `filesystem_errors` 等成员。

另请参见 *locale encoding*。

finder -- 查找器

一种会尝试查找被导入模块的 *loader* 的对象。

存在两种类型的查找器: *元路径查找器* 配合 `sys.meta_path` 使用, 以及 *路径条目查找器* 配合 `sys.path_hooks` 使用。

请参阅 `finders-and-loaders` 和 `importlib` 以了解更多细节。

floor division -- 向下取整除法

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

free threading -- 自由线程

一种线程模型, 在同一个解释器内部的多个线程可以同时运行 Python 字节码。与此相对的是 *global interpreter lock*, 在同一时刻只允许一个线程执行 Python 字节码。参见 [PEP 703](#)。

free variable -- 自由变量

在正式场合下, 如语言执行模型所定义的, 自由变量是指在某个命名空间中被使用的不属于该命名空间中的局部变量的任何变量。参见 *closure variable* 中的样例。在实际应用中, 由于 `codeobject.co_freevars` 属性被如此命名, 该术语有时也被用作 *closure variable* 的同义词。

function -- 函数

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

function annotation -- 函数标注

即针对函数形参或返回值的 *annotation*。

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

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

函数标注语法的详解见 *function* 一节。

参见 *variable annotation* 和 [PEP 484](#), 其中描述了此功能。另请参阅 `annotations-howto` 以了解使用标的最佳实践。

__future__

`future` 语句, `from __future__ import <feature>` 指示编译器使用将在未来的 Python 发布版中成为标准的语法和语义来编译当前模块。`__future__` 模块文档记录了可能的 *feature* 取值。通过导入此模块并对其变量求值, 你可以看到每项新特性在何时被首次加入到该语言中以及它将 (或已) 在何时成为默认:

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

garbage collection -- 垃圾回收

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

generator -- 生成器

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

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

generator iterator -- 生成器迭代器

generator 函数所创建的对象。

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

generator expression -- 生成器表达式

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

```
>>> sum(i*i for i in range(10))      # 平方值 0, 1, 4, ... 81 之和
285
```

generic function -- 泛型函数

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

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

generic type -- 泛型

可参数化的 *type*；通常为 `list` 或 `dict` 这样的容器类。用于类型提示和注解。

更多细节参见 泛型别名类型、[PEP 483](#)、[PEP 484](#)、[PEP 585](#) 和 `typing` 模块。

GIL

参见 *global interpreter lock*。

global interpreter lock -- 全局解释器锁

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

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

在 Python 3.13 中，GIL 可以使用 `--disable-gil` 构建配置选项来禁用。在使用此选项构建 Python 之后，代码必须附带 `-X gil=0` 或在设置 `PYTHON_GIL=0` 环境变量后运行。此特性将为多线程应用程序启用性能提升并让高效率地使用多核 CPU 更加容易。要了解详情，请参阅 [PEP 703](#)。

hash-based pyc -- 基于哈希的 pyc

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

hashable -- 可哈希

一个对象如果具有在其生命周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。

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

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

IDLE

Python 的集成开发与学习环境。idle 是 Python 标准发行版附带的基本编辑器和解释器环境。

immortal -- 永生对象

永生对象是在 [PEP 683](#) 中引入的 *CPython* 实现细节。

如果对象属于永生对象，则它的 *reference count* 永远不会被修改，因而它在解释器运行期间永远不会被取消分配。例如，`True` 和 `None` 在 *CPython* 中都属于永生对象。

immutable -- 不可变对象

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

import path -- 导入路径

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

importing -- 导入

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

importer -- 导入器

查找并加载模块的对象；此对象既属于[finder](#)又属于[loader](#)。

interactive -- 交互

Python 带有一个交互式解释器，这意味着你可以在解释器提示符后输入语句和表达式，立即执行并查看其结果。只需不带参数地启动 `python` 命令（也可以在你的计算机主菜单中选择相应菜单项）。在测试新想法或检验模块和包的时候这会非常方便（记住 `help(x)` 函数）。有关交互模式的详情，参见 [tut-interac](#)。

interpreted -- 解释型

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

interpreter shutdown -- 解释器关闭

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

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

iterable -- 可迭代对象

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 -- 迭代器

用来表示一连串数据流的对象。重复调用迭代器的 `__next__()` 方法（或将其传给内置函数 `next()`）将逐个返回流中的项。当没有数据可用时则将引发 `StopIteration` 异常。到这时迭代器对象中的数据项已耗尽，继续调用其 `__next__()` 方法只会再次引发 `StopIteration`。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。

更多信息可查看 [typeiter](#)。

CPython 实现细节：CPython 没有强制推行迭代器定义 `__iter__()` 的要求。还要注意的是自由线程 CPython 并不保证迭代器操作的线程安全性。

key function -- 键函数

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

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

有多种方式可以创建一个键函数。例如，`str.lower()` 方法可以用作忽略大小写排序的键函数。或者，键函数也可通过 `lambda` 表达式来创建例如 `lambda r: (r[0], r[2])`。此外，`operator`。

`attrgetter()`, `operator.itemgetter()` 和 `operator.methodcaller()` 是键函数的三个构造器。请查看 排序指引来获取创建和使用键函数的示例。

keyword argument -- 关键字参数

参见 *argument*。

lambda

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

LBYL

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

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

list -- 列表

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 -- 列表推导式

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

loader -- 加载器

负责加载模块的对象。它必须定义名为 `load_module()` 的方法。加载器通常由一个 *finder* 返回。另请参阅：

- `finders-and-loaders`
- `importlib.abc.Loader`
- **PEP 302**

locale encoding -- 语言区域编码格式

在 Unix 上，它是 `LC_CTYPE` 语言区域的编码格式。它可以通过 `locale.setlocale(locale.LC_CTYPE, new_locale)` 来设置。

在 Windows 上，它是 ANSI 代码页 (如: "cp1252")。

在 Android 和 VxWorks 上，Python 使用 "utf-8" 作为语言区域编码格式。

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

另请参阅 *filesystem encoding and error handler*。

magic method -- 魔术方法

special method 的非正式同义词。

mapping -- 映射

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

meta path finder -- 元路径查找器

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

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

metaclass -- 元类

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

更多详情参见 [metaclasses](#)。

method -- 方法

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

method resolution order -- 方法解析顺序

方法解析顺序就是在查找成员时搜索基类的顺序。请参阅 `python_2.3_mro` 了解自 2.3 发布版起 Python 解释器所使用算法的详情。

module -- 模块

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

另见 [package](#)。

module spec -- 模块规格

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

另请参阅 `module-specs`。

MRO

参见 [method resolution order](#)。

mutable -- 可变对象

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

named tuple -- 具名元组

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

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

```
>>> sys.float_info[1]           # 索引访问
1024
>>> sys.float_info.max_exp      # 命名字段访问
1024
>>> isinstance(sys.float_info, tuple) # 属于元组
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 -- 命名空间

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

namespace package -- 命名空间包

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

另可参见 [module](#)。

nested scope -- 嵌套作用域

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

new-style class -- 新式类

对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、`__getattr__()`、类方法和静态方法等。

object -- 对象

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

optimized scope -- 已优化的作用域

当代码被编译时编译器已充分知晓目标局部变量名称的作用域，这允许对这些名称的读写进行优化。针对函数、生成器、协程、推导式和生成器表达式的局部命名空间都是这样的已优化作用域。注意：大部分解释器优化将应用于所有作用域，只有那些依赖于已知的局部和非局部变量名称集合的优化会仅限于已优化的作用域。

package -- 包

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

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

parameter -- 形参

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

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

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

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

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

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

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

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

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

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

形参可以同时指定可选和必选参数，也可以为某些可选参数指定默认值。

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

path entry -- 路径入口

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

path entry finder -- 路径入口查找器

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

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

path entry hook -- 路径入口钩子

一种可调用对象，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 -- 基于路径的查找器

默认的一种元路径查找器，可在一个 *import path* 中查找模块。

path-like object -- 路径类对象

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

PEP

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

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

参见 **PEP 1**。

portion -- 部分

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

positional argument -- 位置参数

参见 *argument*。

provisional API -- 暂定 API

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

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

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

provisional package -- 暂定包

参见 *provisional API*。

Python 3000

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

Pythonic

指一个思路或一段代码紧密遵循了 Python 语言最常用的风格和理念，而不是使用其他语言中通用的概念来实现代码。例如，Python 的常用风格是使用 `for` 语句循环来遍历一个可迭代对象中的所有元素。许多其他语言没有这样的结构，因此不熟悉 Python 的人有时会选择一个数字计数器：

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

而相应的更简洁更 Pythonic 的方法是这样的：

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

qualified name -- 限定名称

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

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

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

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

reference count -- 引用计数

指向某个对象的引用的数量。当一个对象的引用计数降为零时，它就会被释放。特殊的 *immortal* 对象具有永远不会被修改的引用计数，因此这种对象永远不会被释放。引用计数对 Python 代码来说通常是不可见的，但它是 CPython 实现的一个关键元素。程序员可以调用 `sys.getrefcount()` 函数来返回特定对象的引用计数。

regular package -- 常规包

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

另参见 *namespace package*。

REPL

“读取-求值-打印循环” read-eval-print loop 的缩写，*interactive* 解释器 shell 的另一个名字。

__slots__

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

sequence -- 序列

一种 *iterable*，它支持通过 `__getitem__()` 特殊方法来使用整数索引进行高效的元素访问，并定义了一个返回序列长度的 `__len__()` 方法。内置序列类型有 `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，因为它使用任意的 *hashable* 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 -- 集合推导式

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

single dispatch -- 单分派

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

slice -- 切片

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

soft deprecated -- 软弃用

被软弃用的 API 不应在新编写的代码中使用，但在已有代码中使用仍是安全的。这样的 API 将保留在文档中并被测试，但不会再获得进一步的功能增强。

与普通的弃用不同，软弃用没有 API 移除计划也不会发出弃用警告。

参见 [PEP 387: Soft Deprecation](#)。

special method -- 特殊方法

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

statement -- 语句

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

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 -- 强引用

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()` 函数可被用于创建一个对象的强引用。通常，必须在退出某个强引用的作用域时在该强引用上调用 `Py_DECREF()` 函数，以避免引用的泄漏。

另请参阅 [borrowed reference](#)。

text encoding -- 文本编码格式

在 Python 中，一个字符串是一串 Unicode 代码点（范围为 U+0000--U+10FFFF）。为了存储或传输一个字符串，它需要被序列化为一串字节。

将一个字符串序列化为一个字节序列被称为“编码”，而从字节序列中重新创建字符串被称为“解码”。

有各种不同的文本序列化 编码器，它们被统称为“文本编码格式”。

text file -- 文本文件

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

另请参看 [binary file](#) 了解能够读写 [字节型对象](#) 的文件对象。

triple-quoted string -- 三引号字符串

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

type -- 类型

Python 对象的类型决定它属于什么种类；每个对象都具有特定的类型。对象的类型可通过其 `__class__` 属性来访问或是用 `type(obj)` 来获取。

type alias -- 类型别名

一个类型的同义词，创建方式是把类型赋值给特定的标识符。

类型别名的作用是简化 [类型注解](#)。例如：

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

可以这样提高可读性：

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

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

参见 `typing` 和 [PEP 484](#)，其中有对此功能的详细描述。

type hint -- 类型注解

annotation 为变量、类属性、函数的形参或返回值指定预期的类型。

类型提示是可选的而不是 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。

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

参见 `typing` 和 [PEP 484](#)，其中有对此功能的详细描述。

universal newlines -- 通用换行

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

variable annotation -- 变量标注

对变量或类属性的 *annotation*。

在标注变量或类属性时，还可选择为其赋值：

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

变量标注通常被用作类型提示：例如以下变量预期接受 `int` 类型的值：

```
count: int = 0
```

变量标注语法的详细解释见 `annassign` 一节。

参见 *function annotation*、[PEP 484](#) 和 [PEP 526](#)，其中描述了此功能。另请参阅 `annotations-howto` 以了解使用标注的最佳实践。

virtual environment -- 虚拟环境

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

另参见 `venv`。

virtual machine -- 虚拟机

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

Zen of Python -- Python 之禅

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

关于本文档

Python 的文档是用 [Sphinx](#) 从 [reStructuredText](#) 源生成的，*Sphinx* 是一个专为处理 Python 文档而编写的文档生成器。

本文档及其工具链之开发，皆在于志愿者之努力，亦恰如 Python 本身。如果您想为此作出贡献，请阅读 [reporting-bugs](#) 了解如何参与。我们随时欢迎新的志愿者！

特别鸣谢：

- Fred L. Drake, Jr.，原始 Python 文档工具集之创造者，众多文档之作者；
- 用于创建 [reStructuredText](#) 和 [Docutils](#) 套件的 [Docutils](#) 项目；
- Fredrik Lundh 的 [Alternative Python Reference](#) 项目，为 *Sphinx* 提供许多好的点子。

B.1 Python 文档的贡献者

有很多对 Python 语言，Python 标准库和 Python 文档有贡献的人，随 Python 源代码分发的 [Misc/ACKS](#) 文件列出了部分贡献者。

有了 Python 社区的输入和贡献，Python 才有了如此出色的文档——谢谢你们！

历史和许可证

C.1 该软件的历史

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

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

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

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

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

i 备注

GPL 兼容并不意味着 Python 在 GPL 下发布。与 GPL 不同，所有 Python 许可证都允许您分发修改后的版本，而无需开源所做的更改。GPL 兼容的许可证使得 Python 可以与其它在 GPL 下发布的软件结

合使用；但其它的许可证则不行。

感谢众多在 Guido 指导下工作的外部志愿者，使得这些发布成为可能。

C.2 获取或以其他方式使用 Python 的条款和条件

Python 软件和文档的使用许可均基于 *PSF* 许可协议。

从 Python 3.8.6 开始，文档中的示例、操作指导和其他代码采用的是 PSF 许可协议和零条款 *BSD* 许可的双重使用许可。

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

C.2.1 用于 PYTHON 3.13.1 的 PSF 许可协议

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 3.13.1 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 3.13.1 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001–2024 Python Software Foundation; All Rights Reserved" are retained in Python 3.13.1 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 3.13.1 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 3.13.1.
4. PSF is making Python 3.13.1 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 3.13.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.13.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.13.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. 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 3.13.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

C.2.2 用于 PYTHON 2.0 的 BEOPEN.COM 许可协议

BEOPEN PYTHON 开源许可协议第 1 版

1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").
2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.
3. BeOpen is making the Software available to Licensee on an "AS IS" basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the "BeOpen Python" logos available at <http://www.pythonlabs.com/logos.html> may be used according to the permissions granted on that web page.
7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

C.2.3 用于 PYTHON 1.6.1 的 CNRI 许可协议

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 ("CNRI"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI's License Agreement and CNRI's notice of copyright, i.e., "Copyright © 1995-2001 Corporation for National Research Initiatives; All Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI's License Agreement, Licensee may substitute the following text (omitting the quotes): "Python 1.6.1 is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python 1.6.1 may be located on the

(续下页)

(接上页)

- internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the internet using the following URL: <http://hdl.handle.net/1895.22/1013>."
3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.
 4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
 5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
 6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
 7. This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia's conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License Agreement only as to issues arising under or with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
 8. By clicking on the "ACCEPT" button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

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

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

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

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

(续下页)

(接上页)

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

C.2.5 ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON 3.13.1 DOCUMENTATION

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

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

C.3 收录软件的许可与鸣谢

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

C.3.1 Mersenne Twister

作为 random 模块下层的 _random C 扩展包括基于从 <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html> 下载的代码。以下是原始代码的完整注释：

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

(续下页)

(接上页)

```
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

Any feedback is very welcome.

<http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html>

email: m-mat @ math.sci.hiroshima-u.ac.jp (remove space)

C.3.2 套接字

socket 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 异步套接字服务

`test.support.asyncchat` 和 `test.support.asyncore` 模块包含以下说明。:

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

(续下页)

(接上页)

```
CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS
OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```

C.3.4 Cookie 管理

http.cookies 模块包含以下声明:

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

    All Rights Reserved

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

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

C.3.5 执行追踪

trace 模块包含以下声明:

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

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

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

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

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

Permission to use, copy, modify, and distribute this Python software and
its associated documentation for any purpose without fee is hereby
granted, provided that the above copyright notice appears in all copies,
and that both that copyright notice and this permission notice appear in
supporting documentation, and that the name of neither Automatrix,
```

(续下页)

(接上页)

Bioreason or Mojam Media be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

C.3.6 UUencode 与 UUdecode 函数

uu 编解码器包含以下声明:

```
Copyright 1994 by Lance Ellinghouse
Cathedral City, California Republic, United States of America.

    All Rights Reserved

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

Modified by Jack Jansen, CWI, July 1995:
- Use binascii module to do the actual line-by-line conversion
  between ascii and binary. This results in a 1000-fold speedup. The C
  version is still 5 times faster, though.
- Arguments more compliant with Python standard
```

C.3.7 XML 远程过程调用

xmlrpc.client 模块包含以下声明:

```
The XML-RPC client interface is

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

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

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

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

(续下页)

(接上页)

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 模块关于 kqueue 的接口包含以下声明:

Copyright (c) 2000 Doug White, 2006 James Knight, 2007 Christian Heimes
All rights reserved.

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

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

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

C.3.10 SipHash24

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

```
<MIT License>
Copyright (c) 2013  Marek Majkowski <marek@popcount.org>

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

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

Original location:
    https://github.com/majek/csiphash/

Solution inspired by code from:
    Samuel Neves (supercop/crypto_auth/siphhash24/little)
    djb (supercop/crypto_auth/siphhash24/little2)
    Jean-Philippe Aumasson (https://131002.net/siphhash/siphhash24.c)
```

C.3.11 strtod 和 dtoa

Python/dtoa.c 文件提供了 C 函数 `dtoa` 和 `strtod`，用于 C 双精度数值和字符串之间的转换，它派生自由 David M. Gay 编写的同名文件。目前该文件可在 <https://web.archive.org/web/20220517033456/http://www.netlib.org/fp/dtoa.c> 访问。在 2009 年 3 月 16 日检索到的原始文件包含以下版权和许可声明:

```
/* *****
 *
 * The author of this software is David M. Gay.
 *
 * Copyright (c) 1991, 2000, 2001 by Lucent Technologies.
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose without fee is hereby granted, provided that this entire notice
 * is included in all copies of any software which is or includes a copy
 * or modification of this software and in all copies of the supporting
 * documentation for such software.
 *
 * THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED
 * WARRANTY.  IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY
 * REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY
 * OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.
 *
 * *****/
```

C.3.12 OpenSSL

如果操作系统有支持则 `hashlib`, `posix` 和 `ssl` 会使用 OpenSSL 库来提升性能。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 发布版, 及其后续衍生版本, 均使用 Apache License v2:

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

(续下页)

(接上页)

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.

"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.

(续下页)

(接上页)

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 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

(续下页)

(接上页)

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, 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

(续下页)

(接上页)

```
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

作为 ctypes 模块下层的 _ctypes C 扩展是使用包括了 libffi 源的副本构建的, 除非构建时配置了 --with-system-libffi:

```
Copyright (c) 1996-2008 Red Hat, Inc and others.
```

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

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

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

C.3.15 zlib

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

```
Copyright (C) 1995-2011 Jean-loup Gailly and Mark Adler
```

```
This software is provided 'as-is', without any express or implied
warranty. In no event will the authors be held liable for any damages
arising from the use of this software.
```

```
Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:
```

```
1. The origin of this software must not be misrepresented; you must not
```

(续下页)

(接上页)

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
jloup@gzip.org

Mark Adler
madler@alumni.caltech.edu

C.3.16 cfuhash

tracemalloc 使用的哈希表的实现基于 cfuhash 项目:

Copyright (c) 2005 Don Owens
All rights reserved.

This code is released under the BSD license:

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

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

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

C.3.17 libmpdec

作为 decimal 模块下层的 _decimal C 扩展是使用包括了 libmpdec 库的副本构建的, 除非构建时配置了 --with-system-libmpdec:

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

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

(续下页)

(接上页)

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

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

C.3.18 W3C C14N 测试套件

test 包中的 C14N 2.0 测试集 (Lib/test/xmltestdata/c14n-20/) 提取自 W3C 网站 <https://www.w3.org/TR/xml-c14n2-testcases/> 并根据 3 条款版 BSD 许可证发行:

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

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

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

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

C.3.19 mimalloc

MIT 许可证:

Copyright (c) 2018-2021 Microsoft Corporation, Daan Leijen

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

(续下页)

(接上页)

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

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

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

C.3.20 asyncio

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.
```

C.3.21 Global Unbounded Sequences (GUS)

文件 `Python/qsbr.c` 改编自 `subr_smr.c` 中 FreeBSD 的“Global Unbounded Sequences”安全内存回收方案。该文件是基于 2 条款 BSD 许可证分发的：

```
Copyright (c) 2019,2020 Jeffrey Roberson <jeff@FreeBSD.org>
```

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

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

```
THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
```

(续下页)

(接上页)

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

版权所有

Python 与这份文档：

版权所有 © 2001-2024 Python 软件基金会。保留所有权利。

版权所有 © 2000 BeOpen.com。保留所有权利。

版权所有 © 1995-2000 Corporation for National Research Initiatives。保留所有权利。

版权所有 © 1991-1995 Stichting Mathematisch Centrum。保留所有权利。

有关完整的许可证和许可信息，请参见[历史](#)和[许可证](#)。

非字母

..., 83

-?

命令行选项, 5

%APPDATA%, 47

>>>, 83

__future__, 88

__slots__, 95

上下文管理协议, 86

特殊

method -- 方法, 96

环境变量

%APPDATA%, 47

BASECFLAGS, 37

BASECPPFLAGS, 36

BLD_SHARED, 38

CC, 36

CC_SHARED, 37

CFLAGS, 28, 3638

CFLAGS_ALIASING, 37

CFLAGS_NODIST, 3638

CFLAGS_FOR_SHARED, 37

COMPILEALL_OPTS, 37

CONFIGURE_CFLAGS, 37

CONFIGURE_CFLAGS_NODIST, 37

CONFIGURE_CPPFLAGS, 36

CONFIGURE_LDFLAGS, 38

CONFIGURE_LDFLAGS_NODIST, 38

CPPFLAGS, 36, 38

CXX, 36

EXTRA_CFLAGS, 37

LDFLAGS, 36, 38

LDFLAGS_NODIST, 38

LD_SHARED, 38

LIBS, 38

LINKCC, 38

OPT, 30, 37

PATH, 10, 18, 42, 44, 50, 51, 53

PATHEXT, 44

PROFILE_TASK, 28

PURIFY, 38

PY_BUILTIN_MODULE_CFLAGS, 38

PY_CFLAGS, 37

PY_CFLAGS_NODIST, 37

PY_CORE_CFLAGS, 37

PY_CORE_LDFLAGS, 39

PY_CPPFLAGS, 36

PY_LDFLAGS, 38

PY_LDFLAGS_NODIST, 39

PY_PYTHON, 54

PY_STDMODULE_CFLAGS, 37

PYLAUNCHER_ALLOW_INSTALL, 55

PYLAUNCHER_ALWAYS_INSTALL, 55

PYLAUNCHER_DEBUG, 54

PYLAUNCHER_DRYRUN, 55

PYLAUNCHER_NO_SEARCH_PATH, 53

PYTHON_BASIC_REPL, 15

PYTHON_COLORS, 10, 15

PYTHON_CPU_COUNT, 9, 15

PYTHON_FROZEN_MODULES, 9, 15

PYTHON_GIL, 9, 15, 89

PYTHON_HISTORY, 15

PYTHON_PERF_JIT_SUPPORT, 9, 15

PYTHON_PRESITE, 9, 16

PYTHONASYNCIODEBUG, 13

PYTHONBREAKPOINT, 10

PYTHONCASEOK, 11

PYTHONCOERCECLOCALE, 13, 23

PYTHONDEBUG, 6, 11, 29

PYTHONDEVMODE, 8, 14

PYTHONDONTWRITEBYTECODE, 6, 11

PYTHONDUMPREFS, 16, 30

PYTHONDUMPREFSFILE, 16

PYTHONEXECUTABLE, 12

PYTHONFAULTHANDLER, 8, 12

PYTHONHASHSEED, 7, 11

PYTHONHOME, 6, 10, 55, 56

PYTHONINSPECT, 6, 11

PYTHONINTMAXSTRDIGITS, 8, 11

PYTHONIOENCODING, 11, 14

PYTHONLEGACYWINDOWSFSENCODING, 13

PYTHONLEGACYWINDOWSSTDIO, 12, 13

PYTHONMALLOC, 13, 29

PYTHONMALLOCSTATS, 13

PYTHONNODEBUGRANGES, 9, 14

PYTHONNOUSERSITE, 7, 12

PYTHONOPTIMIZE, 6, 10

PYTHONPATH, 6, 10, 50, 55, 56
 PYTHONPERFSUPPORT, 9, 14
 PYTHONPLATLIBDIR, 10
 PYTHONPROFILEIMPORTTIME, 8, 13
 PYTHONPYCACHEPREFIX, 9, 11
 PYTHONSAFEPATH, 6, 10
 PYTHONSTARTUP, 6, 10, 11
 PYTHONTRACEMALLOC, 8, 12
 PYTHONUNBUFFERED, 7, 11
 PYTHONUSERBASE, 12
 PYTHONUTF8, 8, 14, 50
 PYTHONVERBOSE, 7, 11
 PYTHONWARNDEFAULTENCODING, 9, 14
 PYTHONWARNINGS, 8, 12
 TEMP, 47

魔术

method -- 方法, 91

A

abstract base class -- 抽象基类, 83
 annotation -- 标注, 83
 argument -- 参数, 83
 asynchronous context manager -- 异步上下文管理器, 84
 asynchronous generator -- 异步生成器, 84
 asynchronous generator iterator -- 异步生成器迭代器, 84
 asynchronous iterable -- 异步可迭代对象, 84
 asynchronous iterator -- 异步迭代器, 84
 attribute -- 属性, 84
 awaitable -- 可等待对象, 84

B

-B
 命令行选项, 6
 -b
 命令行选项, 6
 BDFL, 84
 binary file -- 二进制文件, 84
 BOLT_APPLY_FLAGS
 命令行选项, 28
 BOLT_INSTRUMENT_FLAGS
 命令行选项, 28
 borrowed reference -- 借入引用, 84
 --build
 命令行选项, 33
 bytecode -- 字节码, 85
 bytes-like object -- 字节型对象, 85
 BZIP2_CFLAGS
 命令行选项, 25
 BZIP2_LIBS
 命令行选项, 25

C

-c
 命令行选项, 3
 C 连续, 86
 callable -- 可调用对象, 85

callback -- 回调, 85
 CC
 命令行选项, 25
 CFLAGS, 28, 37, 38
 命令行选项, 25
 CFLAGS_NODIST, 37, 38
 --check-hash-based-pycs
 命令行选项, 6
 class -- 类, 85
 class variable -- 类变量, 85
 closure variable -- 闭包变量, 85
 complex number -- 复数, 85
 CONFIG_SITE
 命令行选项, 33
 context -- 上下文, 86
 context manager -- 上下文管理器, 86
 context variable -- 上下文变量, 86
 contiguous -- 连续, 86
 coroutine -- 协程, 86
 coroutine function -- 协程函数, 86
 CPP
 命令行选项, 25
 CPPFLAGS, 36, 38
 命令行选项, 25
 CPython, 86
 current context -- 当前上下文, 86
 CURSES_CFLAGS
 命令行选项, 26
 CURSES_LIBS
 命令行选项, 26

D

-d
 命令行选项, 6
 decorator -- 装饰器, 86
 descriptor -- 描述器, 86
 dictionary -- 字典, 87
 dictionary comprehension -- 字典推导式, 87
 dictionary view -- 字典视图, 87
 --disable-gil
 命令行选项, 24
 --disable-ipv6
 命令行选项, 22
 --disable-test-modules
 命令行选项, 27
 docstring -- 文档字符串, 87
 duck-typing -- 鸭子类型, 87

E

-E
 命令行选项, 6
 EAFP, 87
 --enable-big-digits
 命令行选项, 22
 --enable-bolt
 命令行选项, 28
 --enable-experimental-jit
 命令行选项, 25

--enable-framework
 命令行选项, 32, 33

--enable-loadable-sqlite-extensions
 命令行选项, 22

--enable-optimizations
 命令行选项, 28

--enable-profiling
 命令行选项, 29

--enable-pystats
 命令行选项, 23

--enable-shared
 命令行选项, 31

--enable-universalsdk
 命令行选项, 32

--enable-wasm-dynamic-linking
 命令行选项, 27

--enable-wasm-pthreads
 命令行选项, 27

--exec-prefix
 命令行选项, 27

expression -- 表达式, 87

extension module -- 扩展模块, 87

F

f-string -- f-字符串, 87

file object -- 文件对象, 87

file-like object -- 文件型对象, 87

filesystem encoding and error handler --
 文件系统编码格式与错误处理器, 87

finder -- 查找器, 88

floor division -- 向下取整除法, 88

Fortran 连续, 86

free threading -- 自由线程, 88

free variable -- 自由变量, 88

function -- 函数, 88

function annotation -- 函数标注, 88

G

garbage collection -- 垃圾回收, 88

GDBM_CFLAGS
 命令行选项, 26

GDBM_LIBS
 命令行选项, 26

generator -- 生成器, 88

generator -- 生成器, 88

generator expression -- 生成器表达式, 89

generator expression -- 生成器表达式, 89

generator iterator -- 生成器迭代器, 88

generic function -- 泛型函数, 89

generic type -- 泛型, 89

GIL, 89

global interpreter lock -- 全局解释器锁, 89

H

-h
 命令行选项, 5

hash-based pyc -- 基于哈希的 pyc, 89

hashable -- 可哈希, 89

--help
 命令行选项, 5

--help-all
 命令行选项, 5

--help-env
 命令行选项, 5

--help-xoptions
 命令行选项, 5

--host
 命令行选项, 33

HOSTRUNNER
 命令行选项, 34

I

-I
 命令行选项, 6

-i
 命令行选项, 6

IDLE, 89

immortal -- 永生对象, 89

immutable -- 不可变对象, 89

import path -- 导入路径, 90

importer -- 导入器, 90

importing -- 导入, 90

interactive -- 交互, 90

interpreted -- 解释型, 90

interpreter shutdown -- 解释器关闭, 90

iterable -- 可迭代对象, 90

iterator -- 迭代器, 90

J

-J
 命令行选项, 10

K

key function -- 键函数, 90

keyword argument -- 关键字参数, 91

L

lambda, 91

LBYL, 91

LDFLAGS, 36, 38
 命令行选项, 25

LDFLAGS_NODIST, 38

LIBB2_CFLAGS
 命令行选项, 26

LIBB2_LIBS
 命令行选项, 26

LIBEDIT_CFLAGS
 命令行选项, 26

LIBEDIT_LIBS
 命令行选项, 26

LIBFFI_CFLAGS
 命令行选项, 26

LIBFFI_LIBS
 命令行选项, 26

LIBLZMA_CFLAGS
 命令行选项, 26

LIBLZMA_LIBS
 命令行选项, 26

LIBMPDEC_CFLAGS
 命令行选项, 26

LIBMPDEC_LIBS
 命令行选项, 26

LIBREADLINE_CFLAGS
 命令行选项, 26

LIBREADLINE_LIBS
 命令行选项, 26

LIBS
 命令行选项, 25

LIBSQLITE3_CFLAGS
 命令行选项, 26

LIBSQLITE3_LIBS
 命令行选项, 26

LIBUUID_CFLAGS
 命令行选项, 26

LIBUUID_LIBS
 命令行选项, 26

list -- 列表, 91

list comprehension -- 列表推导式, 91

loader -- 加载器, 91

locale encoding -- 语言区域编码格式, 91

M

-m
 命令行选项, 4

MACHDEP
 命令行选项, 25

magic method -- 魔术方法, 91

mapping -- 映射, 91

meta path finder -- 元路径查找器, 91

metaclass -- 元类, 91

method -- 方法
 特殊, 96
 魔术, 91

method -- 方法, 92

method resolution order -- 方法解析顺序, 92

module -- 模块, 92

module spec -- 模块规格, 92

MRO, 92

mutable -- 可变对象, 92

N

named tuple -- 具名元组, 92

namespace -- 命名空间, 92

namespace package -- 命名空间包, 92

nested scope -- 嵌套作用域, 92

new-style class -- 新式类, 93

O

-O
 命令行选项, 6

object -- 对象, 93

-OO
 命令行选项, 6

OPT, 30

optimized scope -- 已优化的作用域, 93

P

-P
 命令行选项, 6

package -- 包, 93

PANEL_CFLAGS
 命令行选项, 26

PANEL_LIBS
 命令行选项, 26

parameter -- 形参, 93

PATH, 10, 18, 42, 44, 50, 51, 53

path based finder -- 基于路径的查找器, 94

path entry -- 路径入口, 93

path entry finder -- 路径入口查找器, 93

path entry hook -- 路径入口钩子, 94

path-like object -- 路径类对象, 94

PATHEXT, 44

PEP, 94

PKG_CONFIG
 命令行选项, 25

PKG_CONFIG_LIBDIR
 命令行选项, 25

PKG_CONFIG_PATH
 命令行选项, 25

portion -- 部分, 94

positional argument -- 位置参数, 94

--prefix
 命令行选项, 27

PROFILE_TASK, 28

provisional API -- 暂定 API, 94

provisional package -- 暂定包, 94

PY_PYTHON, 54

PYLAUNCHER_ALLOW_INSTALL, 55

PYLAUNCHER_ALWAYS_INSTALL, 55

PYLAUNCHER_DEBUG, 54

PYLAUNCHER_DRYRUN, 55

PYLAUNCHER_NO_SEARCH_PATH, 53

Python 3000, 94

Python 增强建议; PEP 1, 94

Python 增强建议; PEP 7, 21

Python 增强建议; PEP 8, 81

Python 增强建议; PEP 11, 21, 41, 57

Python 增强建议; PEP 238, 88

Python 增强建议; PEP 278, 97

Python 增强建议; PEP 302, 91

Python 增强建议; PEP 338, 4

Python 增强建议; PEP 343, 86

Python 增强建议; PEP 362, 84, 93

Python 增强建议; PEP 370, 7, 12

Python 增强建议; PEP 397, 51

Python 增强建议; PEP 411, 94

Python 增强建议; PEP 420, 92, 94

Python 增强建议; PEP 443, 89

Python 增强建议; PEP 483, 89

Python 增强建议; PEP 484, 83, 88, 89, 96, 97

Python 增强建议; PEP 488, 6

Python 增强建议; PEP 492, 84, 86

Python 增强建议; PEP 498, 87
 Python 增强建议; PEP 514, 51
 Python 增强建议; PEP 519, 94
 Python 增强建议; PEP 525, 84
 Python 增强建议; PEP 526, 83, 97
 Python 增强建议; PEP 528, 51
 Python 增强建议; PEP 529, 13, 51
 Python 增强建议; PEP 538, 14, 23
 Python 增强建议; PEP 585, 89
 Python 增强建议; PEP 683, 89
 Python 增强建议; PEP 703, 45, 68, 88, 89
 Python 增强建议; PEP 3116, 97
 Python 增强建议; PEP 3155, 94

PYTHON_COLORS, 10
 PYTHON_CPU_COUNT, 9
 PYTHON_FROZEN_MODULES, 9
 PYTHON_GIL, 9, 89
 PYTHON_PERF_JIT_SUPPORT, 9
 PYTHON_PRESITE, 9
 PYTHONCOERCECLOCALE, 23
 PYTHONDEBUG, 6, 29
 PYTHONDEVMODE, 8
 PYTHONDONTWRITEBYTECODE, 6
 PYTHONDUMPPREFS, 30
 PYTHONFAULTHANDLER, 8
 PYTHONHASHSEED, 7, 11
 PYTHONHOME, 6, 10, 55, 56
 Pythonic, 94
 PYTHONINSPECT, 6
 PYTHONINTMAXSTRDIGITS, 8
 PYTHONIOENCODING, 14
 PYTHONLEGACYWINDOWSSTDIO, 12
 PYTHONMALLOC, 13, 29
 PYTHONNODEBUGRANGES, 9
 PYTHONNOUSERSITE, 7
 PYTHONOPTIMIZE, 6
 PYTHONPATH, 6, 10, 50, 55, 56
 PYTHONPERFSUPPORT, 9
 PYTHONPROFILEIMPORTTIME, 8
 PYTHONPYCACHEPREFIX, 9
 PYTHONSAFEPATH, 6
 PYTHONSTARTUP, 6, 11
 PYTHONTRACEMALLOC, 8
 PYTHONUNBUFFERED, 7
 PYTHONUTF8, 8, 14, 50
 PYTHONVERBOSE, 7
 PYTHONWARNDEFAULTENCODING, 9
 PYTHONWARNINGS, 8

Q

-q 命令行选项, 6
 qualified name -- 限定名称, 94

R

-R 命令行选项, 7
 reference count -- 引用计数, 95

regular package -- 常规包, 95
 REPL, 95

S

-S 命令行选项, 7
 -s 命令行选项, 7
 sequence -- 序列, 95
 set comprehension -- 集合推导式, 95
 single dispatch -- 单分派, 95
 slice -- 切片, 95
 soft deprecated -- 软弃用, 95
 special method -- 特殊方法, 96
 statement -- 语句, 96
 static type checker -- 静态类型检查器, 96
 strong reference -- 强引用, 96

T

TCLTK_CFLAGS 命令行选项, 27
 TCLTK_LIBS 命令行选项, 27
 TEMP, 47
 text encoding -- 文本编码格式, 96
 text file -- 文本文件, 96
 triple-quoted string -- 三引号字符串, 96
 type -- 类型, 96
 type alias -- 类型别名, 96
 type hint -- 类型注解, 97

U

-u 命令行选项, 7
 universal newlines -- 通用换行, 97

V

-V 命令行选项, 5
 -v 命令行选项, 7
 variable annotation -- 变量标注, 97
 命令行选项
 -?, 5
 -B, 6
 -b, 6
 BOLT_APPLY_FLAGS, 28
 BOLT_INSTRUMENT_FLAGS, 28
 --build, 33
 BZIP2_CFLAGS, 25
 BZIP2_LIBS, 25
 -c, 3
 CC, 25
 CFLAGS, 25
 --check-hash-based-pycs, 6
 CONFIG_SITE, 33
 CPP, 25
 CPPFLAGS, 25

CURSES_CFLAGS, 26
 CURSES_LIBS, 26
 -d, 6
 --disable-gil, 24
 --disable-ipv6, 22
 --disable-test-modules, 27
 -E, 6
 --enable-big-digits, 22
 --enable-bolt, 28
 --enable-experimental-jit, 25
 --enable-framework, 32, 33
 --enable-loadable-sqlite-extensions, 22
 --enable-optimizations, 28
 --enable-profiling, 29
 --enable-pystats, 23
 --enable-shared, 31
 --enable-universalsdk, 32
 --enable-wasm-dynamic-linking, 27
 --enable-wasm-pthreads, 27
 --exec-prefix, 27
 GDBM_CFLAGS, 26
 GDBM_LIBS, 26
 -h, 5
 --help, 5
 --help-all, 5
 --help-env, 5
 --help-xoptions, 5
 --host, 33
 HOSTRUNNER, 34
 -I, 6
 -i, 6
 -J, 10
 LDFLAGS, 25
 LIBB2_CFLAGS, 26
 LIBB2_LIBS, 26
 LIBEDIT_CFLAGS, 26
 LIBEDIT_LIBS, 26
 LIBFFI_CFLAGS, 26
 LIBFFI_LIBS, 26
 LIBLZMA_CFLAGS, 26
 LIBLZMA_LIBS, 26
 LIBMPDEC_CFLAGS, 26
 LIBMPDEC_LIBS, 26
 LIBREADLINE_CFLAGS, 26
 LIBREADLINE_LIBS, 26
 LIBS, 25
 LIBSQLITE3_CFLAGS, 26
 LIBSQLITE3_LIBS, 26
 LIBUUID_CFLAGS, 26
 LIBUUID_LIBS, 26
 -m, 4
 MACHDEP, 25
 -O, 6
 -OO, 6
 -P, 6
 PANEL_CFLAGS, 26
 PANEL_LIBS, 26
 PKG_CONFIG, 25
 PKG_CONFIG_LIBDIR, 25
 PKG_CONFIG_PATH, 25
 --prefix, 27
 -q, 6
 -R, 7
 -S, 7
 -s, 7
 TCLTK_CFLAGS, 27
 TCLTK_LIBS, 27
 -u, 7
 -V, 5
 -v, 7
 --version, 5
 -W, 7
 --with-address-sanitizer, 30
 --with-app-store-compliance, 33
 --with-assertions, 30
 --with-build-python, 33
 --with-builtin-hashlib-hashes, 32
 --with-computed-gotos, 29
 --with-dbmliborder, 23
 --with-dtrace, 30
 --with-emsripten-target, 27
 --with-ensurepip, 27
 --with-framework-name, 33
 --with-hash-algorithm, 32
 --with-libc, 31
 --with-libm, 31
 --with-libs, 31
 --with-lto, 28
 --with-memory-sanitizer, 30
 --with-openssl, 31
 --with-openssl-rpath, 31
 --without-c-locale-coercion, 23
 --without-decimal-contextvar, 23
 --without-doc-strings, 29
 --without-freelists, 23
 --without-mimalloc, 29
 --without-pymalloc, 29
 --without-readline, 31
 --without-static-libpython, 31
 --with-pkg-config, 23
 --with-platlibdir, 23
 --with-pydebug, 30
 --with-readline, 31
 --with-ssl-default-suites, 32
 --with-strict-overflow, 29
 --with-suffix, 22
 --with-system-expat, 31
 --with-system-libmpdec, 31
 --with-thread-sanitizer, 30
 --with-trace-refs, 30
 --with-tzpath, 22
 --with-undefined-behavior-sanitizer, 30
 --with-universal-archs, 32
 --with-valgrind, 30

--with-wheel-pkg-dir, 23
 -X, 8
 -x, 8
 ZLIB_CFLAGS, 27
 ZLIB_LIBS, 27
 --version
 命令行选项, 5
 virtual environment -- 虚拟环境, 97
 virtual machine -- 虚拟机, 97

W

-W
 命令行选项, 7
 --with-address-sanitizer
 命令行选项, 30
 --with-app-store-compliance
 命令行选项, 33
 --with-assertions
 命令行选项, 30
 --with-build-python
 命令行选项, 33
 --with-builtin-hashlib-hashes
 命令行选项, 32
 --with-computed-gotos
 命令行选项, 29
 --with-dbmliborder
 命令行选项, 23
 --with-dtrace
 命令行选项, 30
 --with-emsripten-target
 命令行选项, 27
 --with-ensurepip
 命令行选项, 27
 --with-framework-name
 命令行选项, 33
 --with-hash-algorithm
 命令行选项, 32
 --with-libc
 命令行选项, 31
 --with-libm
 命令行选项, 31
 --with-libs
 命令行选项, 31
 --with-lto
 命令行选项, 28
 --with-memory-sanitizer
 命令行选项, 30
 --with-openssl
 命令行选项, 31
 --with-openssl-rpath
 命令行选项, 31
 --without-c-locale-coercion
 命令行选项, 23
 --without-decimal-contextvar
 命令行选项, 23
 --without-doc-strings
 命令行选项, 29
 --without-freelists

 命令行选项, 23
 --without-mimalloc
 命令行选项, 29
 --without-pymalloc
 命令行选项, 29
 --without-readline
 命令行选项, 31
 --without-static-libpython
 命令行选项, 31
 --with-pkg-config
 命令行选项, 23
 --with-platlibdir
 命令行选项, 23
 --with-pydebug
 命令行选项, 30
 --with-readline
 命令行选项, 31
 --with-ssl-default-suites
 命令行选项, 32
 --with-strict-overflow
 命令行选项, 29
 --with-suffix
 命令行选项, 22
 --with-system-expat
 命令行选项, 31
 --with-system-libmpdec
 命令行选项, 31
 --with-thread-sanitizer
 命令行选项, 30
 --with-trace-refs
 命令行选项, 30
 --with-tzpath
 命令行选项, 22
 --with-undefined-behavior-sanitizer
 命令行选项, 30
 --with-universal-archs
 命令行选项, 32
 --with-valgrind
 命令行选项, 30
 --with-wheel-pkg-dir
 命令行选项, 23

X

-X
 命令行选项, 8
 -x
 命令行选项, 8

Z

Zen of Python -- Python 之禅, 97
 ZLIB_CFLAGS
 命令行选项, 27
 ZLIB_LIBS
 命令行选项, 27