Python Setup and Usage

リリース *3.14.0a3*

Guido van Rossum and the Python development team

12 月 21, 2024

目次

第 1 章　コマンドラインと環境　3
- 1.1 コマンドライン . . . 3
- 1.2 環境変数 . . . 13

第 2 章　Unix プラットフォームで Python を使う　23
- 2.1 最新バージョンの Python の取得とインストール . . . 23
- 2.2 Python のビルド . . . 24
- 2.3 Python に関係するパスとファイル . . . 24
- 2.4 その他 . . . 25
- 2.5 Custom OpenSSL . . . 25

第 3 章　Python を構成する　27
- 3.1 ビルド要件 . . . 27
- 3.2 生成されるファイル . . . 28
- 3.3 Configure オプション . . . 28
- 3.4 Python ビルドシステム . . . 45
- 3.5 Compiler and linker flags . . . 48

第 4 章　Windows で Python を使う　53
- 4.1 完全版インストーラ . . . 54
- 4.2 Microsoft ストアパッケージ . . . 60
- 4.3 nuget.org パッケージ . . . 62
- 4.4 埋め込み可能なパッケージ . . . 63
- 4.5 別のバンドル . . . 65
- 4.6 Python を構成する . . . 65
- 4.7 UTF-8 モード . . . 67
- 4.8 Windows の Python ランチャ . . . 67
- 4.9 モジュールの検索 . . . 74
- 4.10 追加のモジュール . . . 76
- 4.11 Windows 上で Python をコンパイルする . . . 77
- 4.12 ほかのプラットフォーム . . . 77

第 5 章　Using Python on macOS　79
- 5.1 Using Python for macOS from `python.org` . . . 79

5.2 Alternative Distributions 87
5.3 追加の Python パッケージのインストール 88
5.4 GUI プログラミング 88
5.5 高度なトピック 88
5.6 他のリソース 93

第 6 章 Using Python on Android 95
6.1 Adding Python to an Android app 95

第 7 章 Using Python on iOS 97
7.1 Python at runtime on iOS 97
7.2 Installing Python on iOS 99
7.3 App Store Compliance 104

第 8 章 エディタと IDE 105

付録 A 章 用語集 107

付録 B 章 このドキュメントについて 131
B.1 Python ドキュメント 貢献者 131

付録 C 章 歴史とライセンス 133
C.1 Python の歴史 133
C.2 Terms and conditions for accessing or otherwise using Python 134
C.3 Licenses and Acknowledgements for Incorporated Software 138

付録 D 章 Copyright 159

索引 161

索引 161

このドキュメントでは 異なるプラットフォームでの Python 環境のセットアップの一般的な方法、インタプリタの起動と Python での作業を楽にする方法を説明します。

第

ONE

コマンドラインと環境

CPython インタプリタはコマンドラインと環境を読み取って様々な設定を行ないます。

CPython 実装の詳細: 他の実装のコマンドラインスキームは CPython とは異なります。さらなる情報は implementations を参照してください。

1.1 コマンドライン

Python を起動するとき、以下のうち任意のオプションを指定できます:

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

もちろん、もっとも一般的な利用方法は、単純にスクリプトを起動することです:

```
python myscript.py
```

1.1.1 インターフェイスオプション

インタプリタのインターフェイスは UNIX シェルのものに似ていますが、より多くの起動方法を提供しています:

- When called with standard input connected to a tty device, it prompts for commands and executes them until an EOF (an end-of-file character, you can produce that with `Ctrl-D` on UNIX or `Ctrl-Z, Enter` on Windows) is read. For more on interactive mode, see tut-interac.
- ファイル名引数か、標準入力としてファイルを渡された場合、そのファイルからスクリプトを読み込んで実行します。
- ディレクトリ名を引数に受け取った場合、そのディレクトリから適切な名前のスクリプトファイルを読み込んで実行します。
- `-c` **コマンド** オプションを利用して起動された場合、**コマンド** として渡された Python の文を実行します。**コマンド** の部分には改行で区切られた複数行を指定することもできます。行の先頭の空白文字は Python 文の重要要素です！
- `-m` **モジュール名** として Python モジュールパスにあるモジュールを指定された場合、そのモジュールをスクリプトとして実行します。

非インタラクティブモードでは、入力の全体が実行前にパースされます。

インタプリタによって消費されるオプションリストが終了したあと、継続する全ての引数は sys.argv に渡ります。-- ただし、添字 0 の先頭要素 (sys.argv[0]) はプログラムのソース自体を示す文字列です。

-c <command>

command 内の Python コードを実行します。*command* は改行によって区切られた 1 行以上の文です。通常のモジュールのコードと同じく、行頭の空白文字は意味を持ちます。

このオプションが指定された場合、sys.argv の最初の要素は "-c" になり、カレントディレクトリが sys.path の先頭に追加されます (そのディレクトリにあるモジュールをトップレベルモジュールとして import 出来るようになります)。

引数 command を指定して 監査イベント cpython.run_command を送出します。

-m <module-name>

sys.path から指定されたモジュール名のモジュールを探し、その内容を __main__ モジュールとして実行します。

引数は *module* 名なので、拡張子 (.py) を含めてはいけません。モジュール名は有効な Python の絶対モジュール名 (absolute module name) であるべきですが、実装がそれを強制しているとは限りません (例えば、ハイフンを名前に含める事を許可するかもしれません)。

パッケージ名 (名前空間パッケージも含む) でも構いません。通常のモジュールの代わりにパッケージ名が与えられた場合、インタプリタは <pkg>.__main__ を main モジュールとして実行します。この挙動はスクリプト引数として渡されたディレクトリや zip ファイルをインタプリタが処理するのと意図的に同じにしています。

注釈

このオプションは組み込みモジュールや C で書かれた拡張モジュールには利用できません。Python モジュールファイルを持っていないからです。しかし、コンパイル済みのモジュールは、たとえ元のソースファイルがなくても利用可能です。

このオプションが指定された場合、sys.argv の最初の要素はモジュールファイルのフルパスになります (モジュールファイルを検索している間、最初の要素は "-m" に設定されます)。-c オプションと同様に、カレントディレクトリが sys.path の先頭に追加されます。

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

多くの標準ライブラリモジュールにはスクリプトとして実行された時のためのコードがあります。例えば、timeit モジュールは次のように実行可能です:

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

引数 module-name を指定して 監査イベント cpython.run_module を送出します。

> **参考**
>
> runpy.run_module()
> Python コードで直接使える等価な機能
>
> **PEP 338** - モジュールをスクリプトとして実行する

バージョン 3.1 で変更: __main__ サブモジュールを実行するパッケージ名が提供されました。

バージョン 3.4 で変更: 名前空間パッケージもサポートされました

-

標準入力 (sys.stdin) からコマンドを読み込みます。標準入力がターミナルだった場合、暗黙的に *-i* オプションが指定されます。

このオプションが指定された場合、sys.argv の最初の要素は "-" で、カレントディレクトリが sys.path の先頭に追加されます。

引数無しで 監査イベント cpython.run_stdin を送出します。

<script>

script 内の Python コードを実行します。*script* は、Python ファイル、__main__.py ファイルがあるディレクトリ、__main__.py ファイルがある zip ファイルのいずれかの、ファイルシステム上の (絶対または相対) パスでなければなりません。

このオプションが指定された場合、sys.argv の最初の要素はコマンドラインで指定されたスクリプト名になります。

スクリプト名が Python ファイルを直接指定していた場合、そのファイルを含むディレクトリが sys.path の先頭に追加され、そのファイルは __main__ モジュールとして実行されます。

スクリプト名がディレクトリか zip ファイルを指定していた場合、スクリプト名が sys.path に追加され、その中の __main__.py ファイルが __main__ モジュールとして実行されます。

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

引数 filename を指定して 監査イベント cpython.run_file を送出します。

> **参考**
>
> runpy.run_path()
> Python コードで直接使える等価な機能

インターフェイスオプションが与えられなかった場合、*-i* が暗黙的に指定され、`sys.argv[0]` が空の文字列 (`""`) になり、現在のディレクトリが `sys.path` の先頭に追加されます。また、利用可能であればタブ補完と履歴編集が自動的に有効かされます (rlcompleter-config を参照してください)。

参考

tut-invoking

バージョン 3.4 で変更: タブ補完と履歴の編集が自動的に有効化されます。

1.1.2 一般オプション

-?

-h

--help

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

--help-env

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

Added in version 3.11.

--help-xoptions

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

Added in version 3.11.

--help-all

Print complete usage information and exit.

Added in version 3.11.

-V

--version

Python のバージョン番号を表示して終了します。出力の例:

```
Python 3.8.0b2+
```

2 つ指定すると、次のようにより多くのビルドの情報を表示します:

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

Added in version 3.6: `-VV` オプション。

1.1.3 その他のオプション

-b

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

バージョン 3.5 で変更: Affects also comparisons of `bytes` with `int`.

-B

与えられた場合、Python はソースモジュールのインポート時に `.pyc` ファイルの作成を試みません。*PYTHONDONTWRITEBYTECODE* 環境変数も参照してください。

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

Control the validation behavior of hash-based `.pyc` files. See pyc-invalidation. When set to `default`, checked and unchecked hash-based bytecode cache files are validated according to their default semantics. When set to `always`, all hash-based `.pyc` files, whether checked or unchecked, are validated against their corresponding source file. When set to `never`, hash-based `.pyc` files are not validated against their corresponding source files.

The semantics of timestamp-based `.pyc` files are unaffected by this option.

-d

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

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

-E

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

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

-i

Enter interactive mode after execution.

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

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

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

このオプションはグローバル変数や、スクリプトが例外を発生させるときにそのスタックトレースを調べるのに便利です。*PYTHONINSPECT* も参照してください。

`-I`

Python を隔離モードで実行します。*-E*、*-P*、および *-s* オプションも暗黙的に指定されます。

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

Added in version 3.4.

`-O`

Remove assert statements and any code conditional on the value of `__debug__`. Augment the filename for compiled (*bytecode*) files by adding `.opt-1` before the `.pyc` extension (see **PEP 488**). See also *PYTHONOPTIMIZE*.

バージョン 3.5 で変更: **PEP 488** に従って `.pyc` ファイル名を変更します。

`-OO`

Do *-O* and also discard docstrings. Augment the filename for compiled (*bytecode*) files by adding `.opt-2` before the `.pyc` extension (see **PEP 488**).

バージョン 3.5 で変更: **PEP 488** に従って `.pyc` ファイル名を変更します。

`-P`

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

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

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

Added in version 3.11.

`-q`

インタラクティブモードでも copyright とバージョンのメッセージを表示しません。

Added in version 3.2.

`-R`

Turn on hash randomization. This option only has an effect if the *PYTHONHASHSEED* environment variable is set to `0`, since hash randomization is enabled by default.

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

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

PYTHONHASHSEED によってハッシュシードの固定値を秘密にすることが出来ます。

Added in version 3.2.3.

バージョン 3.7 で変更: このオプションが無視されなくなりました.

`-s`

ユーザのサイトパッケージのディレクトリ を `sys.path` に追加しません。

See also *PYTHONNOUSERSITE*.

> **参考**
>
> **PEP 370** -- ユーザごとの `site-packages` ディレクトリ

`-S`

`site` モジュールの import と、そのモジュールが行なっていた site ごとの `sys.path` への操作を無効にします。後で `site` を明示的に import しても、これらの操作は実行されません (実行したい場合は、`site.main()` を呼び出してください)。

`-u`

Force the stdout and stderr streams to be unbuffered. This option has no effect on the stdin stream.

PYTHONUNBUFFERED も参照してください。

バージョン 3.7 で変更: The text layer of the stdout and stderr streams now is unbuffered.

`-v`

モジュールが初期化されるたびにメッセージを出力し、それがどこ (ファイル名やビルトインモジュール) からロードされたのかを表示します。二回与えられた場合 (`-vv`) は、モジュールを検索するときにチェックしたファイルごとにメッセージを出力します。また、終了時のモジュールクリーンアップに関する情報も提供します。

バージョン 3.10 で変更: The `site` module reports the site-specific paths and `.pth` files being processed.

PYTHONVERBOSE も参照してください。

`-W arg`

警告制御。Python の警告機構はデフォルトでは警告メッセージを `sys.stderr` に表示します。

The simplest settings apply a particular action unconditionally to all warnings emitted by a process (even those that are otherwise ignored by default):

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

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

引数の完全形は次のようになります:

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

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

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

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

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

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

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

複数の *-W* オプションを指定することができます。警告が 1 つ以上のオプションとマッチしたときは、最後にマッチしたオプションのアクションが有効になります。不正な *-W* オプションは無視されます (最初の警告が発生したときに、不正なオプションに対する警告メッセージが表示されます)。

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

詳しくは warning-filter と describing-warning-filters を参照してください。

-x

Unix 以外の形式の `#!cmd` を使うために、ソースの最初の行をスキップします。これは、DOS 専用のハックのみを目的としています。

-X

様々な実装固有のオプションのために予約されています。現在のところ CPython は以下の値を定義しています:

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

  Added in version 3.3.

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

  Added in version 3.4.

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

  Added in version 3.4.

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

  Added in version 3.11.

- `-X importtime` to show how long each import takes. It shows module name, cumulative time (including nested imports) and self time (excluding nested imports). Note that its output may be broken in multi-threaded application. Typical usage is `python3 -X importtime -c 'import asyncio'`. See also *PYTHONPROFILEIMPORTTIME*.

  Added in version 3.7.

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

  Added in version 3.7.

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

  Added in version 3.7.

- `-X pycache_prefix=PATH` enables writing `.pyc` files to a parallel tree rooted at the given directory instead of to the code tree. See also *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` disables the inclusion of the tables mapping extra location information (end line, start column offset and end column offset) to every instruction in code objects. This is useful when smaller code objects and pyc files are desired as well as suppress-

ing the extra visual location indicators when the interpreter displays tracebacks. See also *PYTHONNODEBUGRANGES*.

Added in version 3.11.

- `-X frozen_modules` determines whether or not frozen modules are ignored by the import machinery. A value of `on` means they get imported and `off` means they are ignored. The default is `on` if this is an installed Python (the normal case). If it's under development (running from the source tree) then the default is `off`. Note that the `importlib_bootstrap` and `importlib_bootstrap_external` frozen modules are always used, even if this flag is set to `off`. See also *PYTHON_FROZEN_MODULES*.

  Added in version 3.11.

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

  Added in version 3.12.

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

  Added in version 3.13.

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

  Added in version 3.13.

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

  Added in version 3.13.

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

  Added in version 3.13.

任意の値を渡し、`sys._xoptions` 辞書から取り出すことも出来ます。

Added in version 3.2.

バージョン 3.9 で変更: Removed the `-X showalloccount` option.

バージョン 3.10 で変更: Removed the `-X oldparser` option.

1.1.4 色の制御

デフォルトでは、Python インタープリターは、トレースバックの表示などの特定の状況で、出力を色でハイライトするように構成されています。この振る舞いは、異なる環境変数を設定することで制御できます。

環境変数 `TERM` を `dumb` に設定することで、色付けを無効にできます。

`FORCE_COLOR` 環境変数が設定されている場合、TERM の値にかかわらず、色付けは有効になります。これは、ターミナルではないが ANSI エスケープシーケンスを表示できる CI システムで有効です。

`NO_COLOR` 環境変数が設定されている場合、Python は出力のすべての色を無効にします。これは `FORCE_COLOR` よりも優先されます。

これらのすべての環境変数は、色の出力の制御のために他のツールにも使用されます。Python インタープリターでのみ色の出力を制御するには、*PYTHON_COLORS* 環境変数を使用できます。この変数は、`FORCE_COLOR` よりも優先される `NO_COLOR` よりも、さらに優先されます。

1.1.5 使うべきでないオプション

`-J`

Jython のために予約されています。

1.2 環境変数

以下の環境変数は Python の挙動に影響します。環境変数は -E や -I 以外のコマンドラインスイッチの前に処理されます。衝突したときにコマンドラインスイッチが環境変数をオーバーライドするのは慣例です。

`PYTHONHOME`

標準 Python ライブラリの場所を変更します。デフォルトでは、ライブラリは *prefix*/lib/python*version* と *exec_prefix*/lib/python*version* から検索されます。ここで、*prefix* と *exec_prefix* はインストール依存のディレクトリで、両方共デフォルトでは /usr/local です。

PYTHONHOME が 1 つのディレクトリに設定されている場合、その値は *prefix* と *exec_prefix* の両方を置き換えます。それらに別々の値を指定したい場合は、*PYTHONHOME* を *prefix*:*exec_prefix* のように指定します。

`PYTHONPATH`

モジュールファイルのデフォルトの検索パスを追加します。この環境変数のフォーマットはシェルの `PATH` と同じで、`os.pathsep` (Unix ならコロン、Windows ならセミコロン) で区切られた 1 つ以上のディレクトリパスです。存在しないディレクトリは警告なしに無視されます。

通常のディレクトリに加えて、*PYTHONPATH* のエントリはピュア Python モジュール (ソース形式でもコンパイルされた形式でも) を含む zip ファイルを参照することもできます。拡張モジュールは zip

ファイルの中から import することはできません。

デフォルトの検索パスはインストール依存ですが、通常は *prefix*/lib/python*version* で始まります。(上の *PYTHONHOME* を参照してください。) これは **常に** *PYTHONPATH* に追加されます。

上の **インターフェイスオプション** で説明されているように、追加の検索パスディレクトリが *PYTHONPATH* の手前に追加されます。検索パスは Python プログラムから `sys.path` 変数として操作することができます。

`PYTHONSAFEPATH`

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

Added in version 3.11.

`PYTHONPLATLIBDIR`

If this is set to a non-empty string, it overrides the `sys.platlibdir` value.

Added in version 3.9.

`PYTHONSTARTUP`

この変数が読み込み可能なファイル名の場合、対話モードで最初のプロンプトが表示される前にそのファイルの Python コマンドが実行されます。ファイル内で定義されているオブジェクトやインポートされたオブジェクトを対話セッションで修飾せずに使用するために、ファイルは対話的なコマンドと同じ名前空間で実行されます。このファイルの中で、プロンプト `sys.ps1` や `sys.ps2`、ならびにフック `sys.__interactivehook__` も変更できます。

Raises an auditing event `cpython.run_startup` with the filename as the argument when called on startup.

`PYTHONOPTIMIZE`

この変数に空でない文字列を設定するのは *-O* オプションを指定するのと等価です。整数を設定した場合、*-O* を複数回指定したのと同じになります。

`PYTHONBREAKPOINT`

If this is set, it names a callable using dotted-path notation. The module containing the callable will be imported and then the callable will be run by the default implementation of `sys.breakpointhook()` which itself is called by built-in `breakpoint()`. If not set, or set to the empty string, it is equivalent to the value "pdb.set_trace". Setting this to the string "0" causes the default implementation of `sys.breakpointhook()` to do nothing but return immediately.

Added in version 3.7.

`PYTHONDEBUG`

この変数に空でない文字列を設定するのは *-d* オプションを指定するのと等価です。整数を指定した場合、*-d* を複数回指定したのと同じになります。

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

PYTHONINSPECT

この変数に空でない文字列を設定するのは *-i* オプションを指定するのと等価です。

この変数は Python コードから os.environ を使って変更して、プログラム終了時のインスペクトモードを強制することができます。

引数無しで 監査イベント cpython.run_stdin を送出します。

バージョン 3.12.5 で変更: (also 3.11.10, 3.10.15, 3.9.20, and 3.8.20) Emits audit events.

バージョン 3.13 で変更: Uses PyREPL if possible, in which case *PYTHONSTARTUP* is also executed. Emits audit events.

PYTHONUNBUFFERED

この変数に空でない文字列を設定するのは *-u* オプションを指定するのと等価です。

PYTHONVERBOSE

この変数に空でない文字列を設定するのは *-v* オプションを指定するのと等価です。整数を設定した場合、*-v* を複数回指定したのと同じになります。

PYTHONCASEOK

この環境変数が設定されている場合、Python は import 文で大文字/小文字を区別しません。これは Windows と macOS でのみ動作します。

PYTHONDONTWRITEBYTECODE

この変数に空でない文字列を設定した場合、Python はソースモジュールのインポート時に .pyc ファイルを作成しようとはしなくなります。*-B* オプションを指定するのと等価です。

PYTHONPYCACHEPREFIX

If this is set, Python will write .pyc files in a mirror directory tree at this path, instead of in __pycache__ directories within the source tree. This is equivalent to specifying the *-X* pycache_prefix=PATH option.

Added in version 3.8.

PYTHONHASHSEED

この変数が設定されていない場合や random に設定された場合、乱数値が str、bytes オブジェクトのハッシュのシードに使われます。

PYTHONHASHSEED が整数値に設定された場合、その値はハッシュランダム化が扱う型の hash() 生成の固定シードに使われます。

その目的は再現性のあるハッシュを可能にすることです。例えばインタープリタ自身の自己テストや Python プロセスのクラスタでハッシュ値を共有するのに用います。

整数は [0,4294967295] の十進数でなければなりません。0 を指定するとハッシュランダム化は無効化されます。

Added in version 3.2.3.

PYTHONINTMAXSTRDIGITS

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

Added in version 3.11.

PYTHONIOENCODING

この変数がインタープリタ実行前に設定されていた場合、encodingname:errorhandler という文法で標準入力/標準出力/標準エラー出力のエンコードを上書きします。encodingname と :errorhandler の部分はどちらも任意で、str.encode() と同じ意味を持ちます。

標準エラー出力の場合、:errorhandler の部分は無視されます; ハンドラは常に 'backslashreplace' です。

バージョン 3.4 で変更: encodingname の部分が任意になりました。

バージョン 3.6 で変更: On Windows, the encoding specified by this variable is ignored for interactive console buffers unless *PYTHONLEGACYWINDOWSSTDIO* is also specified. Files and pipes redirected through the standard streams are not affected.

PYTHONNOUSERSITE

この環境変数が設定されている場合、Python は **ユーザ** site-packages **ディレクトリ** を sys.path に追加しません。

> **参考**
>
> **PEP 370** -- ユーザごとの site-packages ディレクトリ

PYTHONUSERBASE

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

> **参考**
>
> **PEP 370** -- ユーザごとの site-packages ディレクトリ

PYTHONEXECUTABLE

この環境変数が設定された場合、sys.argv[0] に、C ランタイムから取得した値の代わりにこの環境変数の値が設定されます。macOS でのみ動作します。

PYTHONWARNINGS

This is equivalent to the *-W* option. If set to a comma separated string, it is equivalent to specifying *-W* multiple times, with filters later in the list taking precedence over those earlier in the list.

The simplest settings apply a particular action unconditionally to all warnings emitted by a process (even those that are otherwise ignored by default):

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

詳しくは warning-filter と describing-warning-filters を参照してください。

PYTHONFAULTHANDLER

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

Added in version 3.3.

PYTHONTRACEMALLOC

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

Added in version 3.4.

PYTHONPROFILEIMPORTTIME

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

Added in version 3.7.

PYTHONASYNCIODEBUG

この環境変数が空でない文字列に設定された場合、`asyncio` モジュールの デバッグモード が有効化されます。

Added in version 3.4.

PYTHONMALLOC

Set the Python memory allocators and/or install debug hooks.

Set the family of memory allocators used by Python:

- `default`: use the default memory allocators.
- `malloc`: use the `malloc()` function of the C library for all domains (`PYMEM_DOMAIN_RAW`, `PYMEM_DOMAIN_MEM`, `PYMEM_DOMAIN_OBJ`).

- `pymalloc`: use the pymalloc allocator for `PYMEM_DOMAIN_MEM` and `PYMEM_DOMAIN_OBJ` domains and use the `malloc()` function for the `PYMEM_DOMAIN_RAW` domain.
- `mimalloc`: use the mimalloc allocator for `PYMEM_DOMAIN_MEM` and `PYMEM_DOMAIN_OBJ` domains and use the `malloc()` function for the `PYMEM_DOMAIN_RAW` domain.

Install debug hooks:

- `debug`: install debug hooks on top of the default memory allocators.
- `malloc_debug`: same as `malloc` but also install debug hooks.
- `pymalloc_debug`: same as `pymalloc` but also install debug hooks.
- `mimalloc_debug`: same as `mimalloc` but also install debug hooks.

Added in version 3.6.

バージョン 3.7 で変更: Added the `"default"` allocator.

`PYTHONMALLOCSTATS`

空でない文字列に設定されると、Python は新たなオブジェクトアリーナが生成される時と、シャットダウン時に pymalloc メモリアロケータ の統計情報を表示します。

This variable is ignored if the *`PYTHONMALLOC`* environment variable is used to force the `malloc()` allocator of the C library, or if Python is configured without `pymalloc` support.

バージョン 3.6 で変更: This variable can now also be used on Python compiled in release mode. It now has no effect if set to an empty string.

`PYTHONLEGACYWINDOWSFSENCODING`

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

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

Availability: Windows.

Added in version 3.6: より詳しくは **PEP 529** を参照してください。

`PYTHONLEGACYWINDOWSSTDIO`

If set to a non-empty string, does not use the new console reader and writer. This means that Unicode characters will be encoded according to the active console code page, rather than using utf-8.

This variable is ignored if the standard streams are redirected (to files or pipes) rather than referring to console buffers.

Availability: Windows.

Added in version 3.6.

PYTHONCOERCECLOCALE

If set to the value 0, causes the main Python command line application to skip coercing the legacy ASCII-based C and POSIX locales to a more capable UTF-8 based alternative.

If this variable is *not* set (or is set to a value other than 0), the LC_ALL locale override environment variable is also not set, and the current locale reported for the LC_CTYPE category is either the default C locale, or else the explicitly ASCII-based POSIX locale, then the Python CLI will attempt to configure the following locales for the LC_CTYPE category in the order listed before loading the interpreter runtime:

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

If setting one of these locale categories succeeds, then the LC_CTYPE environment variable will also be set accordingly in the current process environment before the Python runtime is initialized. This ensures that in addition to being seen by both the interpreter itself and other locale-aware components running in the same process (such as the GNU readline library), the updated setting is also seen in subprocesses (regardless of whether or not those processes are running a Python interpreter), as well as in operations that query the environment rather than the current C locale (such as Python's own locale.getdefaultlocale()).

Configuring one of these locales (either explicitly or via the above implicit locale coercion) automatically enables the surrogateescape error handler for sys.stdin and sys.stdout (sys.stderr continues to use backslashreplace as it does in any other locale). This stream handling behavior can be overridden using *PYTHONIOENCODING* as usual.

For debugging purposes, setting PYTHONCOERCECLOCALE=warn will cause Python to emit warning messages on stderr if either the locale coercion activates, or else if a locale that *would* have triggered coercion is still active when the Python runtime is initialized.

Also note that even when locale coercion is disabled, or when it fails to find a suitable target locale, *PYTHONUTF8* will still activate by default in legacy ASCII-based locales. Both features must be disabled in order to force the interpreter to use ASCII instead of UTF-8 for system interfaces.

Availability: Unix.

Added in version 3.7: より詳しくは **PEP 538** を参照をしてください。

PYTHONDEVMODE

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

Added in version 3.7.

PYTHONUTF8

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

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

Setting any other non-empty string causes an error during interpreter initialisation.

Added in version 3.7.

`PYTHONWARNDEFAULTENCODING`

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

詳細は io-encoding-warning を参照してください。

Added in version 3.10.

`PYTHONNODEBUGRANGES`

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

Added in version 3.11.

`PYTHONPERFSUPPORT`

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

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

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

Added in version 3.12.

`PYTHON_PERF_JIT_SUPPORT`

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

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

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

Added in version 3.13.

`PYTHON_CPU_COUNT`

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

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

Added in version 3.13.

PYTHON_FROZEN_MODULES

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

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

Added in version 3.13.

PYTHON_COLORS

If this variable is set to 1, the interpreter will colorize various kinds of output. Setting it to 0 deactivates this behavior. See also 色の制御.

Added in version 3.13.

PYTHON_BASIC_REPL

If this variable is set to 1, the interpreter will not attempt to load the Python-based *REPL* that requires curses and readline, and will instead use the traditional parser-based *REPL*.

Added in version 3.13.

PYTHON_HISTORY

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

Added in version 3.13.

PYTHON_GIL

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

See also the *-X gil* command-line option, which takes precedence over this variable, and what-snew313-free-threaded-cpython.

Added in version 3.13.

1.2.1 デバッグモード変数

PYTHONDUMPREFS

設定された場合、Python はインタプリタのシャットダウン後に残っているオブジェクトと参照カウントをダンプします。

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

PYTHONDUMPREFSFILE

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

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

Added in version 3.11.

`PYTHON_PRESITE`

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

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

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

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

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

Added in version 3.13.

第

TWO

UNIX プラットフォームで PYTHON を使う

2.1 最新バージョンの Python の取得とインストール

2.1.1 Linux

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

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

参考

https://www.debian.org/doc/manuals/maint-guide/first.en.html
: Debian ユーザー向け

https://en.opensuse.org/Portal:Packaging
: OpenSuse ユーザー向け

https://docs.fedoraproject.org/en-US/package-maintainers/Packaging_Tutorial_GNU_Hello/
: Fedora ユーザー向け

https://slackbook.org/html/package-management-making-packages.html
: Slackware ユーザー向け

2.1.2 FreeBSD と OpenBSD

- FreeBSD ユーザーが Python パッケージを追加するには次のようにしてください:

```
pkg install python3
```

- OpenBSD ユーザーが Python パッケージを追加するには次のようにしてください:

```
pkg_add -r python
```

(次のページに続く)

(前のページからの続き)

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

例えば、i386 ユーザーが Python 2.5.1 を取得するには次のようにします:

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

2.2 Python のビルド

CPython を自分でコンパイルしたい場合は、まず ソース を入手します。最新リリース版のソースをダウンロード、あるいはソースリポジトリから新しく クローン を作成してください。(パッチの作成に貢献したい場合はクローンが必要になるでしょう。)

ビルド手順は通常のコマンドで行います

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

Configure **のオプション** や特定の Unix プラットフォームにおける注意点は Python ソースツリーのルートにある README.rst に細かく記載されています。

警告

`make install` は `python3` バイナリを上書きまたはリンクを破壊してしまうかもしれません。そのため、`make install` の代わりに *exec_prefix*/bin/python*version* のみインストールする `make altinstall` が推奨されています。

2.3 Python に関係するパスとファイル

これらはローカルインストールの慣例に応じて変化します; *prefix* と *exec_prefix* はインストール状況に依存していて、GNU ソフトウェアによって解釈されます; この二つは同じである場合があります。

例えば、ほとんどの Linux システムでは、両方のデフォルトが `/usr` です。

ファイル/ディレクトリ	意味
exec_prefix/bin/python3	インタプリタの推奨される場所
prefix/lib/python*version*, *exec_prefix*/lib/python*version*	標準モジュールを格納するディレクトリの、推奨される場所。
prefix/include/python*version*, *exec_prefix*/include/python*version*	Python 拡張や Python の埋込みに必要となる include ファイルを格納するディレクトリの推奨される場所。

2.4 その他

Python スクリプトを Unix で簡単に使うためには、例えば次のようにしてスクリプトを実行可能ファイルにし、

```
$ chmod +x script
```

適切な shebang 行をスクリプトの先頭に置きます。たいていの場合良い方法は

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

で、PATH 全体から Python インタープリターを探します。しかし、いくつかの Unix は **env** コマンドを持たないので、インタープリターのパスを /usr/bin/python3 のようにハードコードしなければならないかもしれません。

シェルコマンドを Python スクリプトから使うには、subprocess モジュールを参照してください。

2.5 Custom OpenSSL

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

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

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

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

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

```
$ pushd python-3.x.x
```

(次のページに続く)

(前のページからの続き)

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

注釈

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

第

THREE

PYTHON を構成する

3.1 ビルド要件

CPython のビルドに必要な機能と最小バージョン:

- C11 コンパイラ。C11 オプション機能 は不要です。
- Windows では、Microsoft Visual Studio 2017 以降が必要です。
- Support for IEEE 754 floating-point numbers and floating-point Not-a-Number (NaN).
- Support for threads.
- OpenSSL 1.1.1 is the minimum version and OpenSSL 3.0.9 is the recommended minimum version for the `ssl` and `hashlib` extension modules.
- SQLite 3.15.2 for the `sqlite3` extension module.
- Tcl/Tk 8.5.12 for the `tkinter` module.
- Autoconf 2.71 and aclocal 1.16.5 are required to regenerate the `configure` script.

バージョン 3.1 で変更: Tcl/Tk version 8.3.1 is now required.

バージョン 3.5 で変更: On Windows, Visual Studio 2015 or later is now required. Tcl/Tk version 8.4 is now required.

バージョン 3.6 で変更: Selected C99 features are now required, like `<stdint.h>` and `static inline` functions.

バージョン 3.7 で変更: Thread support and OpenSSL 1.0.2 are now required.

バージョン 3.10 で変更: OpenSSL 1.1.1 is now required. Require SQLite 3.7.15.

バージョン 3.11 で変更: C11 compiler, IEEE 754 and NaN support are now required. On Windows, Visual Studio 2017 or later is required. Tcl/Tk version 8.5.12 is now required for the `tkinter` module.

バージョン 3.13 で変更: Autoconf 2.71, aclocal 1.16.5 and SQLite 3.15.2 are now required.

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

3.2 生成されるファイル

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

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

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

3.2.1 構成スクリプト

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

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

```
autoreconf -ivf -Werror
```

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

3.3 Configure オプション

List all `configure` script options using:

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

Python のソース配布の中の `Misc/SpecialBuilds.txt` も参照してください。

3.3.1 一般的なオプション

`--enable-loadable-sqlite-extensions`

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

`sqlite3` モジュールの `sqlite3.Connection.enable_load_extension()` メソッドを参照してください。

Added in version 3.6.

`--disable-ipv6`

IPv6 サポートを無効にします (サポートされている場合はデフォルトで有効)、`socket` モジュールを

参照してください。

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

Python `int` の桁の大きさをビット単位で定義します: 15 ビットまたは 30 ビットです。

デフォルトでは、桁の大きさは 30 です。

`PYLONG_BITS_IN_DIGIT` を `15` または `30` に定義します。

`sys.int_info.bits_per_digit` を参照してください。

--with-suffix=SUFFIX

Python の実行ファイルの接尾辞を *SUFFIX* に設定します。

デフォルトの接尾辞は、Windows と macOS では `.exe` (`python.exe` 実行ファイル)、Emscripten node では `.js` 、Emscripten browser では `.html` 、WASI では `.wasm` 、その他のプラットフォームでは空文字列になります (`python` 実行ファイル)。

バージョン 3.11 で変更: WASM プラットフォームのデフォルトの接頭辞は、`.js` 、`.html` 、`.wasm` のうちの 1 つです。

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

デフォルトのタイムゾーン検索パスを `zoneinfo.TZPATH` に設定します。`zoneinfo` モジュールの Compile-time configuration を参照してください。

デフォルト: `/usr/share/zoneinfo:/usr/lib/zoneinfo:/usr/share/lib/zoneinfo:/etc/zoneinfo`

`os.pathsep` パスセパレータを参照してください。

Added in version 3.9.

--without-decimal-contextvar

コルーチンローカルコンテキスト (デフォルト) ではなく、スレッドローカルコンテキストを使用して `_decimal` 拡張モジュールをビルドします。`decimal` モジュールを参照してください。

`decimal.HAVE_CONTEXTVAR` および `contextvars` モジュールを参照してください。

Added in version 3.9.

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

`dbm` モジュールの DB バックエンドをチェックする順序をオーバーライドします。

有効な値は、バックエンド名をコロン (`:`) で区切った文字列です:

- `ndbm`;
- `gdbm`;
- `bdb`。

--without-c-locale-coercion

UTF-8 ベースのロケールへの C ロケールの強制を無効にします (デフォルトで有効)。

PY_COERCE_C_LOCALE マクロは定義しないでください。

PYTHONCOERCECLOCALE および **PEP 538** を参照してください。

--with-platlibdir=DIRNAME

Python のライブラリディレクトリ名 (デフォルトは lib)。

Fedora と SuSE は 64 ビットプラットフォームで lib64 を使用します。

sys.platlibdir を参照してください。

Added in version 3.9.

--with-wheel-pkg-dir=PATH

ensurepip モジュールが使用する wheel パッケージのディレクトリです (デフォルトはなし)。

Linux ディストリビューションのパッケージングポリシーの中には、依存関係をバンドルすることを推奨しているものがあります。例えば、Fedora は wheel パッケージを /usr/share/python-wheels/ ディレクトリにインストールし、ensurepip._bundled パッケージはインストールしません。

Added in version 3.10.

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

configure がビルドの依存関係を検出するために **pkg-config** を使用するかどうかを設定します。

- check (デフォルト): **pkg-config** はオプションです。
- yes : **pkg-config** は必須です。
- no : **pkg-config** が存在しても、configure は使用しません。

Added in version 3.11.

--enable-pystats

Turn on internal Python performance statistics gathering.

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

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

Effects:

- Add *-X pystats* command line option.
- Add PYTHONSTATS environment variable.
- Define the Py_STATS macro.
- Add functions to the sys module:

  - `sys._stats_on()`: Turns on statistics gathering.
  - `sys._stats_off()`: Turns off statistics gathering.
  - `sys._stats_clear()`: Clears the statistics.
  - `sys._stats_dump()`: Dump statistics to file, and clears the statistics.

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

統計情報を読むには `Tools/scripts/summarize_stats.py` を使用してください。

Statistics:

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

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

Added in version 3.11.

`--disable-gil`

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

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

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

Added in version 3.13.

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

Indicate how to integrate the JIT compiler.

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

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

Added in version 3.13.

`PKG_CONFIG`

Path to `pkg-config` utility.

`PKG_CONFIG_LIBDIR`

`PKG_CONFIG_PATH`

`pkg-config` options.

3.3.2 C コンパイラのオプション

`CC`

C コンパイラのコマンド。

`CFLAGS`

C コンパイラのフラグ。

`CPP`

C プリプロセッサのコマンド。

`CPPFLAGS`

C プリプロセッサのフラグ。(`-I`*`include_dir`* など)

3.3.3 リンカのオプション

`LDFLAGS`

リンカのフラグ。(`-L`*`library_directory`* など)

`LIBS`

リンカに渡すライブラリ。(`-l`*`library`* など)

`MACHDEP`

マシン依存のライブラリファイルの名前。

3.3.4 サードパーティ依存関係のオプション

Added in version 3.11.

`BZIP2_CFLAGS`

`BZIP2_LIBS`

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

`CURSES_CFLAGS`

`CURSES_LIBS`

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

`GDBM_CFLAGS`

`GDBM_LIBS`

C compiler and linker flags for `gdbm`.

`LIBB2_CFLAGS`

`LIBB2_LIBS`

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

`LIBEDIT_CFLAGS`

`LIBEDIT_LIBS`

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

`LIBFFI_CFLAGS`

`LIBFFI_LIBS`

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

`LIBMPDEC_CFLAGS`

`LIBMPDEC_LIBS`

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

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

`LIBLZMA_CFLAGS`

`LIBLZMA_LIBS`

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

`LIBREADLINE_CFLAGS`

`LIBREADLINE_LIBS`

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

`LIBSQLITE3_CFLAGS`

`LIBSQLITE3_LIBS`

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

`LIBUUID_CFLAGS`

`LIBUUID_LIBS`

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

`PANEL_CFLAGS`

`PANEL_LIBS`

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

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

`TCLTK_CFLAGS`

`TCLTK_LIBS`

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

`ZLIB_CFLAGS`

`ZLIB_LIBS`

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

3.3.5 WebAssembly オプション

`--enable-wasm-dynamic-linking`

WASM のダイナミックリンクサポートをオンにします。

ダイナミックリンクにより `dlopen` が可能になります。デッドコードの排除や機能追加に制限があるため、実行ファイルのファイルサイズが大きくなります。

Added in version 3.11.

`--enable-wasm-pthreads`

WASM の pthreads サポートをオンにします。

Added in version 3.11.

3.3.6 インストールオプション

`--prefix=PREFIX`

アーキテクチャに依存しないファイルを PREFIX にインストールします。Unix の場合、デフォルトは `/usr/local` です。

この値は、実行時に `sys.prefix` を使って取得することができます。

例として、`--prefix="$HOME/.local/"` を使用すると、Python をそのホームディレクトリにインストールすることができます。

`--exec-prefix=EPREFIX`

アーキテクチャ依存のファイルを EPREFIX にインストールします。デフォルトは *`--prefix`* です。

この値は、実行時に `sys.exec_prefix` を使って取得することができます。

`--disable-test-modules`

`test` パッケージや `_testcapi` 拡張モジュール (デフォルトでビルド、インストールされます) のようなテストモジュールをビルド、インストールしないようにします。

Added in version 3.10.

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

Python のインストール時に実行される `ensurepip` コマンドを選択します:

- `upgrade` (デフォルト): `python -m ensurepip --altinstall --upgrade` コマンドを実行します。
- `install`: `python -m ensurepip --altinstall` コマンドを実行します。

- `no`: ensurepip を実行しない;

Added in version 3.6.

3.3.7 パフォーマンスに関するオプション

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

`--enable-optimizations`

PROFILE_TASK を使用して Profile Guided Optimization (PGO) を有効にします (デフォルトでは無効です)。

C コンパイラの Clang では、PGO のために `llvm-profdata` プログラムが必要です。macOS では、GCC もこれを必要とします: GCC は macOS の Clang のエイリアスにすぎません。

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

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

Added in version 3.6.

バージョン 3.10 で変更: GCC で `-fno-semantic-interposition` を使用する。

`PROFILE_TASK`

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

デフォルト: `-m test --pgo --timeout=$(TESTTIMEOUT)`

Added in version 3.8.

バージョン 3.13 で変更: Task failure is no longer ignored silently.

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

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

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

Added in version 3.6.

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

バージョン 3.12 で変更: Use ThinLTO as the default optimization policy on Clang if the compiler accepts the flag.

--enable-bolt

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

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

BOLT is still a fairly new project so this flag should be considered experimental for now. Because this tool operates on machine code its success is dependent on a combination of the build environment + the other optimization configure args + the CPU architecture, and not all combinations are supported. BOLT versions before LLVM 16 are known to crash BOLT under some scenarios. Use of LLVM 16 or newer for BOLT optimization is strongly encouraged.

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

Added in version 3.12.

BOLT_APPLY_FLAGS

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

Added in version 3.12.

BOLT_INSTRUMENT_FLAGS

Arguments to `llvm-bolt` when instrumenting binaries.

Added in version 3.12.

--with-computed-gotos

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

--without-mimalloc

Disable the fast mimalloc allocator (enabled by default).

See also *PYTHONMALLOC* environment variable.

--without-pymalloc

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

See also *PYTHONMALLOC* environment variable.

--without-doc-strings

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

Don't define the `WITH_DOC_STRINGS` macro.

See the `PyDoc_STRVAR()` macro.

`--enable-profiling`

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

`--with-strict-overflow`

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

3.3.8 Python Debug Build

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

Effects of a debug build:

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

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

バージョン 3.8 で変更: Release builds and debug builds are now ABI compatible: defining the `Py_DEBUG` macro no longer implies the `Py_TRACE_REFS` macro (see the *--with-trace-refs* option).

3.3.9 Debug options

--with-pydebug

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

--with-trace-refs

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

Effects:

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

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

Statically allocated objects are not traced.

Added in version 3.8.

バージョン 3.13 で変更: This build is now ABI compatible with release build and *debug build*.

--with-assertions

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

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

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

Added in version 3.6.

--with-valgrind

Enable Valgrind support (default is no).

--with-dtrace

Enable DTrace support (default is no).

See Instrumenting CPython with DTrace and SystemTap.

Added in version 3.6.

--with-address-sanitizer

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

Added in version 3.6.

--with-memory-sanitizer

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

Added in version 3.6.

--with-undefined-behavior-sanitizer

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

Added in version 3.6.

--with-thread-sanitizer

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

Added in version 3.13.

3.3.10 リンカのオプション

--enable-shared

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

--without-static-libpython

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

Added in version 3.10.

3.3.11 Libraries options

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

Link against additional libraries (default is no).

--with-system-expat

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

--with-system-libmpdec

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

Added in version 3.3.

バージョン 3.13 で変更: Default to using the installed `mpdecimal` library.

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

> **参考**
>
> *LIBMPDEC_CFLAGS* and *LIBMPDEC_LIBS*.

--with-readline=readline|editline

Designate a backend library for the `readline` module.

- readline: Use readline as the backend.

- editline: Use editline as the backend.

Added in version 3.10.

--without-readline

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

Don't define the `HAVE_LIBREADLINE` macro.

Added in version 3.10.

--with-libm=STRING

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

--with-libc=STRING

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

--with-openssl=DIR

Root of the OpenSSL directory.

Added in version 3.7.

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

Set runtime library directory (rpath) for OpenSSL libraries:

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

Added in version 3.10.

3.3.12 Security Options

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

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

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

Added in version 3.4.

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

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

Built-in hash modules:

- `md5`;

- `sha1`;
- `sha256`;
- `sha512`;
- `sha3` (with shake);
- `blake2`.

Added in version 3.9.

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

Override the OpenSSL default cipher suites string:

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

See the `ssl` module.

Added in version 3.7.

バージョン 3.10 で変更: The settings `python` and *STRING* also set TLS 1.2 as minimum protocol version.

`--disable-safety`

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

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

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

Added in version 3.14.

`--enable-slower-safety`

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

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

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

Added in version 3.14.

3.3.13 macOS のオプション

Mac/README.rst を参照。

`--enable-universalsdk`

`--enable-universalsdk=SDKDIR`

ユニバーサルバイナリビルドを作成します。*SDKDIR* はビルドの実行にどの macOS SDK が使用されるべきかを指定します (デフォルトでは指定しません) 。

`--enable-framework`

`--enable-framework=INSTALLDIR`

従来の Unix インストールではなく、Python.framework を作成します。オプションの *INSTALLDIR* はインストール先のパスを指定します (デフォルトでは指定しません) 。

`--with-universal-archs=ARCH`

作成するユニバーサルバイナリの種類を指定します。このオプションは、*--enable-universalsdk* が指定された場合のみ有効です。

オプション:

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

`--with-framework-name=FRAMEWORK`

macOS の Python フレームワークの名前を指定します。*--enable-framework* が指定された場合のみ有効です (デフォルトでは `Python`) 。

`--with-app-store-compliance`

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

Python 標準ライブラリは、macOS と iOS の App Store による配布用に送信された場合に、自動検査ツールのエラーを発生させることが知られている文字列を含んでいます。このオプションを有効にした場合、App Store コンプライアンスに合わせて修正することが知られているパッチのリストを適用し

ます。カスタムのパッチファイルを指定することもできます。このオプションはデフォルトでは無効になっています。

Added in version 3.13.

3.3.14 iOS のオプション

iOS/README.rst を参照。

`--enable-framework=INSTALLDIR`

Python.framework を作成します。macOS とは違い、インストールパスを指定する *INSTALLDIR* 引数は必須です。

`--with-framework-name=FRAMEWORK`

フレームワークの名前を指定します (デフォルト: `Python`) 。

3.3.15 クロスコンパイルのオプション

クロスコンパイル、またはクロスビルドは、異なる CPU アーキテクチャやプラットフォーム用に Python をビルドするために使用できます。クロスコンパイルには、ビルドプラットフォーム用の Python インタープリターが必要です。ビルドする Python のバージョンは、クロスコンパイルされたホスト Python のバージョンと一致する必要があります。

`--build=BUILD`

BUILD でビルドするための設定です。通常は、**`config.guess`** により推測されます。

`--host=HOST`

HOST (ターゲットプラットフォーム) で動作するプログラムをビルドするためのクロスコンパイル。

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

クロスコンパイル用のビルド `python` バイナリへのパス。

Added in version 3.11.

`CONFIG_SITE=file`

構成をオーバーライドするファイルを指す環境変数。

config.site ファイルの例:

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

`HOSTRUNNER`

クロスコンパイル用にホストプラットフォームの CPython を実行するプログラム。

Added in version 3.11.

クロスコンパイルの例:

```
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 ビルドシステム

3.4.1 ビルドシステムの主要なファイル

- `configure.ac` => `configure`;
- `Makefile.pre.in` => `Makefile` (`configure` により作成されます);
- `pyconfig.h` (`configure` により作成されます);
- `Modules/Setup`: `Module/makesetup` シェルスクリプトを使用して Makefile がビルドする C 拡張。

3.4.2 主要なビルドステップ

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

3.4.3 Main Makefile targets

make

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

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

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

explain, `make clean && make` should work around most dependency problems, at the expense of longer build times.

make platform

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

make profile-opt

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

make clean

Remove built files.

make distclean

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

make install

Build the `all` target and install Python.

make test

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

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

make ci

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

Added in version 3.14.

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

make buildbottest

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

make regen-all

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

3.4.4 C extensions

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

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

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

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

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

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

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

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

3.5 Compiler and linker flags

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

3.5.1 Preprocessor flags

`CONFIGURE_CPPFLAGS`

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

Added in version 3.6.

`CPPFLAGS`

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

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

`BASECPPFLAGS`

Added in version 3.4.

`PY_CPPFLAGS`

Extra preprocessor flags added for building the interpreter object files.

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

Added in version 3.2.

3.5.2 Compiler flags

`CC`

C コンパイラのコマンド。

Example: `gcc -pthread`.

`CXX`

C++ compiler command.

Example: `g++ -pthread`.

`CFLAGS`

C コンパイラのフラグ。

`CFLAGS_NODIST`

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

In particular, *CFLAGS* should not contain:

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

Added in version 3.5.

`COMPILEALL_OPTS`

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

Added in version 3.12.

`EXTRA_CFLAGS`

Extra C compiler flags.

`CONFIGURE_CFLAGS`

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

Added in version 3.2.

`CONFIGURE_CFLAGS_NODIST`

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

Added in version 3.5.

`BASECFLAGS`

Base compiler flags.

`OPT`

Optimization flags.

`CFLAGS_ALIASING`

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

Added in version 3.7.

`CCSHARED`

Compiler flags used to build a shared library.

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

`CFLAGSFORSHARED`

Extra C flags added for building the interpreter object files.

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

`PY_CFLAGS`

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

`PY_CFLAGS_NODIST`

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

Added in version 3.5.

`PY_STDMODULE_CFLAGS`

C flags used for building the interpreter object files.

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

Added in version 3.7.

`PY_CORE_CFLAGS`

Default: `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE`.

Added in version 3.2.

`PY_BUILTIN_MODULE_CFLAGS`

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

Default: `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE_BUILTIN`.

Added in version 3.8.

`PURIFY`

Purify command. Purify is a memory debugger program.

Default: empty string (not used).

3.5.3 Linker flags

`LINKCC`

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

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

`CONFIGURE_LDFLAGS`

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

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

Added in version 3.2.

`LDFLAGS_NODIST`

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

In particular, *LDFLAGS* should not contain:

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

`CONFIGURE_LDFLAGS_NODIST`

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

Added in version 3.8.

`LDFLAGS`

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

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

`LIBS`

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

Example: `-lrt`.

`LDSHARED`

Command to build a shared library.

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

`BLDSHARED`

Command to build `libpython` shared library.

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

`PY_LDFLAGS`

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

`PY_LDFLAGS_NODIST`

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

Added in version 3.8.

`PY_CORE_LDFLAGS`

Linker flags used for building the interpreter object files.

Added in version 3.8.

脚注

第

FOUR

WINDOWS で PYTHON を使う

このドキュメントは、Python を Microsoft Windows で使うときに知っておくべき、Windows 固有の動作についての概要を伝えることを目的としています。

ほとんどの Unix システムとサービスとは違って、Windows には、システムがサポートする Python インストールが含まれていません。Python を利用可能にするために、CPython チームは長年の間、すべての リリース で Windows インストーラをコンパイルしてきました。単独のユーザで使われるコアインタープリターとライブラリをユーザーごとに追加する Python インストールを、これらインストーラーは主として意図しています。インストーラでは単一マシンのすべてのユーザ用にインストールすることもでき、また、これとは分離されたアプリケーションローカルな配布物の ZIP ファイルも入手可能です。

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

Windows で使えるインストーラには多くの様々なものがあり、それぞれが利点と欠点を持っています。

完全版インストーラ には全てのコンポーネントが含まれており、Python を使う開発者がどんな種類のプロジェクトでも最適な選択肢です。

Microsoft **ストアパッケージ** は、スクリプトやパッケージの実行、IDLE の使用やその他開発環境に適したシンプルな構成の Python です。Windows 10 以上が求められはしますが、他のプログラムを壊すことなく安全にインストールできます。Python やそのツールを起動する多くの便利なコマンドも提供しています。

nuget.org **パッケージ** は、継続的インテグレーションのための軽量なインストール構成です。これは Python パッケージのビルドやスクリプトの実行にも使えますが、アップデート可能ではなく、ユーザーインターフェイスツールもありません。

埋め込み可能なパッケージ は、他の大きなアプリケーションに埋め込むのに適した、Python の最小パッケージです。

4.1 完全版インストーラ

4.1.1 インストール手順

ダウンロードできる Python 3.14 のインストーラは 4 つあります。インタプリタの 32 ビット版、64 ビット版がそれぞれ 2 つずつあります。*WEB* **インストーラ** は最初のダウンロードサイズは小さく、必要なコンポーネントはインストーラ実行時に必要に応じて自動的にダウンロードします。**オフラインインストーラ** にはデフォルトインストールに必要なコンポーネントが含まれていて、インターネット接続はオプショナルな機能のためにだけに必要となります。インストール時にダウンロードを避けるほかの方法については **ダウンロード不要なインストール** を参照して下さい。

インストーラを開始すると、2 つの選択肢からひとつを選べます:

"Install Now" を選択した場合:

- 管理者権限は **不要です** (ただし C ランタイムライブラリのシステム更新が必要であったり、*Windows の Python* **ランチャ** をすべてのユーザ向けにインストールする場合は必要です)。
- Python はあなたのユーザディレクトリにインストールされます。
- *Windows の Python* **ランチャ** はこのインストールウィザード最初のページの下部のチェックボックス指定に従ってインストールされます。
- 標準ライブラリ、テストスイート、ランチャ、pip がインストールされます。
- このインストールウィザード最初の下部のチェックボックスをチェックすれば、環境変数 `PATH` にインストールディレクトリが追加されます。
- ショートカットはカレントユーザだけに可視になります。

”Customize installation” を選択すると、インストール場所、その他オプションやインストール後のアクションの変更などのインストールの有りようを選べます。デバッグシンボルやデバッグバイナリをインストールするならこちらを選択する必要があるでしょう。

すべてのユーザのためのインストールのためには ”Customize installation” を選んでください。この場合:

- 管理者資格か承認が必要かもしれません。
- Python は Program Files ディレクトリにインストールされます。
- *Windows* の *Python* ランチャ は Windows ディレクトリにインストールされます。
- オプショナルな機能はインストール中に選択できます。
- 標準ライブラリをバイトコードにプリコンパイルできます。
- そう選択すれば、インストールディレクトリはシステム環境変数 `PATH` に追加されます。
- ショートカットがすべてのユーザで利用できるようになります。

4.1.2 MAX_PATH の制限を除去する

Windows は歴史的にパスの長さが 260 文字に制限されています。つまり、これより長いパスは解決できず結果としてエラーになるということです。

Windows の最新版では、この制限は約 32,000 文字まで拡張できます。管理者が、グループポリシーの ”Win32 の長いパスを有効にする (Enable Win32 long paths)” を有効にするか、レジストリキー `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem` の `LongPathsEnabled` の値を `1` に設定する必要があります。

これにより、`open()` 関数や `os` モジュール、他のほとんどのパスの機能が 260 文字より長いパスを受け入れ、返すことができるようになります。

これらのオプションを変更したら、それ以上の設定は必要ありません。

バージョン 3.6 で変更: Python で長いパスのサポートが可能になりました。

4.1.3 インストーラの GUI なしでインストールする

インストーラの GUI で利用できるすべてのオプションは、コマンドラインからも指定できます。これによりユーザとの対話なしで数多くの機器に同じインストールを行うような、スクリプト化されたインストールを行うことができます。ちょっとしたデフォルトの変更のために、GUI を抑制することなしにこれらコマンドラインオプションをセットすることもできます。

インストーラーには、以下のオプション (`/?` でインストーラを実行することで確認できます) を渡すことができます:

名前	説明
/passive	ユーザーの操作なしでも進捗を表示する
/quiet	UI を表示せずにインストール・アンインストールする
/simple	ユーザーによるカスタマイズができないようにする
/uninstall	(確認無しで) Python を削除する
/layout [ディレクトリ]	すべてのコンポーネントを事前にダウンロードする
/log [ファイル名]	ログファイルの場所を指定する

ほかのすべてのオプションは `name=value` の形で渡します。value は大抵 `0` で機能を無効化、`1` で機能を有効化、であるとかパスの指定です。利用可能なオプションの完全なリストは以下の通りです。

名前	説明	デフォルト
InstallAllUsers	システムワイドなインストールを実行する。	0
TargetDir	インストール先ディレクトリ。	InstallAllUsers に基いて選択されます。
DefaultAllUsersTargetDir	すべてのユーザ向けインストールのためのデフォルトインストール先ディレクトリ。	`%ProgramFiles%\Python X.Y` または `%ProgramFiles(x86)%\Python X.Y`
DefaultJustForMeTargetDir	自分一人用インストールのためのデフォルトインストール先ディレクトリ。	`%LocalAppData%\Programs\Python\PythonXY` または `%LocalAppData%\Programs\Python\PythonXY-32` または `%LocalAppData%\Programs\Python\PythonXY-64`
DefaultCustomTargetDir	カスタムインストールディレクトリとしてデフォルトで GUI に表示される値。	(空)
AssociateFiles	ランチャもインストールする場合に、ファイルの関連付けを行う。	1
CompileAll	すべての `.py` ファイルをバイトコンパイルして `.pyc` を作る。	0
PrependPath	`PATH` にインストールディレクトリと Scripts ディレクトリを先頭に追加し、`PATHEXT` に `.PY` を追加する。	0
AppendPath	`PATH` にインストールディレクトリと Scripts ディレクトリを追加し、`PATHEXT` に `.PY` を追加する。	0
Shortcuts	インストールするインタプリタ、ドキュメント、IDLE へのショートカットを作る。	1
Include_doc	Python マニュアルをインストールする。	1
Include_debug	デバッグバイナリをインストールする。	0
Include_dev	開発者用ヘッダーとライブラリをインストールする。これを省略すると、使用不可能なインストールになる可能性があります。	1
Include_exe	`python.exe` と関連するファイルをインストールする。これを省略すると、使用不可能なインストールになる可能性があります。	1
Include_launcher	*Windows の Python ランチャ* をインストールする。	1

例えばデフォルトでシステムワイドな Python インストレーションを静かに行うには、以下コマンドを使えます (コマンドプロンプトより):

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

テストスイートなしの Python のパーソナルなコピーのインストールをユーザに簡単に行わせるには、以下コマンドのショートカットを作れば良いです。これはインストーラの最初のページを単純化して表示し、また、カスタマイズできないようにします:

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

(ランチャのインストールを省略するとファイルの関連付けも省略されるので、これはランチャインストールを含めたシステムワイドなインストールをした場合のユーザごとインストールに限った場合のお勧めです。)

上でリストしたオプションは、実行ファイルと同じ場所の `unattend.xml` と名付けられたファイルで与えることもできます。このファイルはオプションとその値のリストを指定します。値がアトリビュートとして与えられた場合、それは数値であれば数値に変換されます。エレメントテキストで与える場合は常に文字列のままです。以下は、先の例と同じオプションをセットするファイルの実例です:

```
<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 ダウンロード不要なインストール

Python のいくつかの機能は最初にダウンロードしたインストーラには含まれていないため、それらの機能をインストールしようと選択するとインターネット接続が必要になります。インターネット接続が必要にならないように、全てのコンポーネントをすぐにできる限りダウンロードして、完全な **配置構成** *(layout)* を作成し、どんな機能が選択されたかに関わらず、それ以上インターネット接続を必要がないようにします。この方法のダウンロードサイズは必要以上に大きくなるかもしれませんが、たくさんの回数インストールしようとする場合には、ローカルにキャッシュされたコピーを持つことはとても有用です。

コマンドプロンプトから以下のコマンドを実行して、必要なファイルをできる限り全てダウンロードします。`python-3.9.0.exe` 部分は実際のインストーラの名前に置き換え、同名のファイルどうしの衝突が起こらないように、個別のディレクトリ内に配置構成を作るのを忘れないようにしてください。

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

進捗表示を隠すのに `/quiet` オプションを指定することもできます。

4.1.5 インストール後の変更

いったん Python がインストールされたら、Windows のシステム機能の「プログラムと機能」ツールから機能の追加や削除ができます。Python のエントリを選択して「アンインストールと変更」を選ぶことで、インストーラをメンテナンスモードで開きます。

インストーラ GUI で "Modify" を選ぶと、チェックボックスの選択を変えることで機能の追加削除ができます - チェックボックスの選択を変えなければ、何かがインストールされたり削除されたりはしません。いくつかのオプションはこのモードでは変更することはできません。インストールディレクトリなどです。それらを変えたいのであれば、完全に削除してから再インストールする必要があります。

"Repair" では、現在の設定で本来インストールされるべきすべてのファイルを検証し、削除されていたり更新されていたりするファイルを修正します。

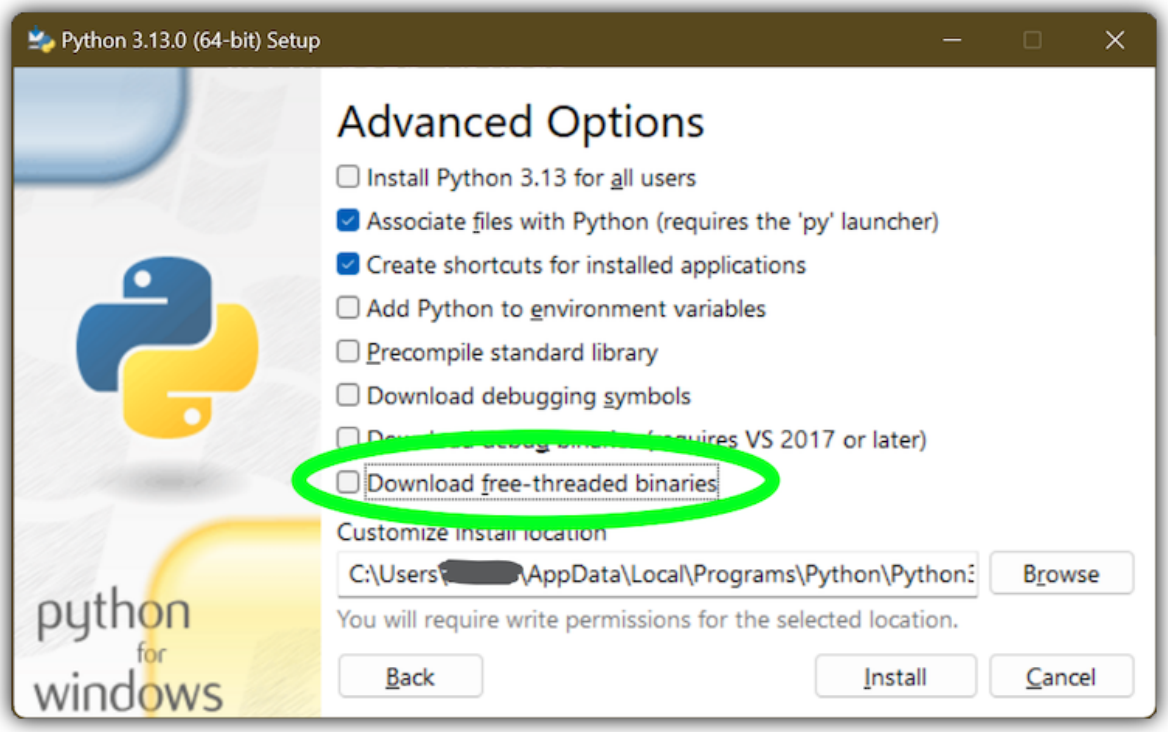
"Uninstall" は Python を完全に削除します。「プログラムと機能」内の自身のエントリを持つ *Windows の Python ランチャ* の例外が起こります。

4.1.6 フリースレッドバイナリ (Free-threaded Binaries) のインストール

Added in version 3.13: (実験的)

> **注釈**
>
> このセクションで説明されているすべての項目は実験的とみなされており、将来のリリースで変更される可能性があります。

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



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

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

To specify the install option at the command line, use `Include_freethreaded=1`. See **ダウンロード不要なインストール** for instructions on pre-emptively downloading the additional binaries for offline install. The options to include debug symbols and binaries also apply to the free-threaded builds.

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

4.2 Microsoft ストアパッケージ

Added in version 3.7.2.

Microsoft ストアパッケージは、例えば生徒が主に対話型で使うことを意図した簡単にインストールできる Python インタプリタです。

このパッケージをインストールするには、最新の Windows 10 のアップデートになっていることを確認し、Microsoft ストアアプリで "Python 3.14" と検索します。選んだアプリが Python Software Foundation が

公開したものであることを確認して、インストールします。

> **警告**
>
> Python は常に Microsoft ストアで無料で利用できます。もしお金を払うように要求されたなら、正しいパッケージを選んでいません。

インストールした後は、スタートメニューから Python を見付けて起動するでしょう。あるいは、`python` とタイプしてコマンドプロンプトや PowerShell のセッションから使えるでしょう。さらに、`pip` や `idle` とタイプして pip あるいは IDLE を利用できます。IDLE はスタートメニューからも見付けられます。

All three commands are also available with version number suffixes, for example, as `python3.exe` and `python3.x.exe` as well as `python.exe` (where `3.x` is the specific version you want to launch, such as 3.14). Open "Manage App Execution Aliases" through Start to select which version of Python is associated with each command. It is recommended to make sure that `pip` and `idle` are consistent with whichever version of `python` is selected.

仮想環境は `python -m venv` で作成し、有効化して普通に使えます。

既に別のバージョンの Python をインストールして `PATH` 変数に追加してある場合は、Microsoft ストアのものではない `python.exe` として使えます。新しくインストールした Python にアクセスするには、`python3.exe` あるいは `python3.x.exe` として使えます。

`py.exe` ランチャーはこの Python のインストールを見つけますが、従来のインストーラーによるインストールを優先します。

Python を除去するには、「設定」を開き「アプリと機能」を使うか、「スタート」にある Python を右クリックしてアンインストールします。アンインストールでは、この Python に直接インストールした全てのパッケージが除去されますが、仮想環境はどれも除去されません。

4.2.1 既知の問題

Redirection of local data, registry, and temporary paths

Microsoft ストアアプリの制限により、Python スクリプトには `TEMP` やレジストリのような共有の場所への完全な書き込み権限は無いでしょう。その代わり、個人用のところへ書き込みます。スクリプトで共有の場所で変更しなければならない場合は、完全版のインストーラでインストールする必要があります。

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

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

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

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

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

- Reading from `HKLM\\Software` is allowed and results are merged with the `registry.dat` file in the package.
- Writing to `HKLM\\Software` is not allowed if the corresponding key/value exists, i.e. modifying existing keys.
- Writing to `HKLM\\Software` is allowed as long as a corresponding key/value does not exist in the package and the user has the correct access permissions.

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

4.3 nuget.org パッケージ

Added in version 3.5.2.

nuget.org パッケージはサイズを縮小した Python 環境で、システム全体で使える Python が無い継続的インテグレーションやビルドシステムで使うことを意図しています。nuget は ".NET のためのパッケージマネージャ" ですが、ビルド時に使うツールを含んだパッケージに対しても非常に上手く動作します。

nuget の使用方法についての最新の情報を得るには nuget.org に行ってください。ここから先は Python 開発者にとって十分な要約です。

`nuget.exe` コマンドラインツールは、例えば curl や PowerShell を使って `https://aka.ms/nugetclidl` から直接ダウンロードできるでしょう。このツールを次のように使って、64 bit あるいは 32 bit のマシン向けの最新バージョンの Python がインストールできます:

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

特定のバージョンを選択するには、`-Version 3.x.y` を追加してください。出力ディレクトリは `.` から変更されることがあり、パッケージはサブディレクトリにインストールされます。デフォルトではサブディレクトリはパッケージと同じ名前になり、`-ExcludeVersion` オプションを付けないとこの名前はインストールされたバージョンを含みます。サブディレクトリの中にはインストールされた Python を含んでいる `tools` ディレクトリがあります:

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

(次のページに続く)

(前のページからの続き)

```
Python 3.5.2

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

一般的には、nuget パッケージはアップグレードできず、より新しいバージョンは横並びにインストールされ、フルパスで参照されます。そうする代わりに、手動で直接パッケージを削除し、再度インストールすることもできます。多くの CI システムは、ビルド間でファイルを保存しておかない場合、この作業を自動的に行います。

`tools` ディレクトリと同じ場所に `build\native` ディレクトリがあります。このディレクトリは、インストールされた Python を参照する C++ プロジェクトで使える MSBuild プロパティファイル `python.props` を含みます。ここに設定を入れると自動的にヘッダを使い、ビルド時にライブラリをインポートします。

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

4.3.1 フリースレッドパッケージ (Free-threaded packages)

Added in version 3.13: (実験的)

> **注釈**
>
> このセクションで説明されているすべての項目は実験的とみなされており、将来のリリースで変更される可能性があります。

フリースレッドバイナリ (free-threaded binaries) を含むパッケージは、64-bit 版では python-freethreaded 、32-bit 版では pythonx86-freethreaded 、ARM64 版では pythonarm64-freethreaded と命名されます。これらのパッケージはともに `python3.13t.exe` と `python.exe` エントリーポイントを含み、どちらもフリースレッドで実行されます。

4.4 埋め込み可能なパッケージ

Added in version 3.5.

埋め込み用の配布 (embedded distribution) は、最小限の Python 環境を含んだ ZIP ファイルです。これは、エンドユーザから直接的にアクセスされるのではなく何かアプリケーションの一部として動作することを意図したものです。

展開されると、埋め込み用の配布は、環境変数、システムレジストリの設定、インストールされているパッケージといったユーザのシステムから (ほぼ) 完全に独立しています。ZIP 内には標準ライブラリがプリコンパイルにより最適化された `.pyc` として含まれ、また、`python3.dll`, `python37.dll`, `python.exe`, `pythonw.exe` のすべてが入っています。(IDLE のようなすべての依存物を含む) Tcl/tk、pip、Python ドキュメントは含

まれていません。

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

サードパーティのパッケージはアプリケーションのインストーラによって、埋め込み用配布と同じ場所にインストールされるべきです。通常の Python インストレーションのように依存性管理に pip を使うことは、この配布ではサポートされません。ですが、ちょっとした注意を払えば、自動更新のために pip を含めて利用することはできるかもしれません。一般的には、ユーザに更新を提供する前に開発者が新しいバージョンとの互換性を保証できるよう、サードパーティーのパッケージはアプリケーションの一部として扱われるべきです (”vendoring”)。

この配布の 2 つのお勧めできるユースケースを、以下で説明します。

4.4.1 Python アプリケーション

Python で記述された、必ずしもユーザにその事実を意識させる必要のないアプリケーションです。埋め込み用配布はこのケースで、インストールパッケージ内に Python のプライベートバージョンを含めるのに使えるでしょう。その事実がどのように透過的であるべきかに依存して (あるいは逆に、どのようにプロフェッショナルにみえるべきか)、2 つの選択肢があります。

ランチャとなる特別な実行ファイルを使うことはちょっとしたコーディングを必要としますが、ユーザにとっては最も透過的なユーザ体験となります。カスタマイズされたランチャでは、何もしなければ Python で実行されるプログラムの明白な目印はありません; アイコンはカスタマイズし、会社名やバージョン情報を指定し、ファイルの関連付けがそれに相応しく振舞うようにできます。ほとんどのケースではカスタムランチャは、ハードコードされたコマンドライン文字列で単純に `Py_Main` を呼び出すので済むはずです。

より簡単なアプローチは、`python.exe` または `pythonw.exe` を必要なコマンドライン引数とともに直接呼び出すバッチファイルかショートカットを提供することです。この場合、そのアプリケーションは実際の名前ではなく Python であるようにみえるので、ほかに動作している Python プロセスやファイルの関連付けと区別するのにユーザが困るかもしれません。

後者のアプローチではパッケージは、パス上で利用可能であることを保証するために、Python 実行ファイルと同じディレクトリにインストールされるべきです。特別なランチャの場合はアプリケーション起動前に検索パスを指定する機会があるので、パッケージはほかの場所に配置できます。

4.4.2 Python の埋め込み

ネイティブコードで書かれ、時々スクリプト言語のようなものを必要とするようなアプリケーションです。Python 埋め込み用の配布はこの目的に使えます。一般的に、アプリケーションの大半がネイティブコード内にあり、一部が `python.exe` を呼び出すか、直接的に `python3.dll` を使います。どちらのケースでも、ロード可能な Python インタプリタを提供するのには、埋め込み用の配布を展開してアプリケーションのインスト

レーションのサブディレクトリに置くことで十分です。

アプリケーションが使うパッケージは、インタプリタ初期化前に検索パスを指定する機会があるので、任意の場所にインストールできます。また、埋め込み用配布を使うのと通常の Python インストレーションを使うのとでの根本的な違いはありません。

4.5 別のバンドル

標準の CPython の配布物の他に、追加の機能を持っている修正されたパッケージがあります。以下は人気のあるバージョンとそのキーとなる機能です:

ActivePython
: マルチプラットフォーム互換のインストーラー、ドキュメント、PyWin32

Anaconda
: 人気のある (numpy, scipy や pandas のような) 科学系モジュールと、パッケージマネージャ `conda` 。

Enthought Deployment Manager
: "次世代の Python 環境とパッケージマネージャー" ("The Next Generation Python Environment and Package Manager")。

  以前は Enthought が Canopy を提供していましたが、これは 2016 年にサポートが終了しました。

WinPython
: ビルド済みの科学系パッケージと、パッケージのビルドのためのツールを含む、Windows 固有のディストリビューション。

これらパッケージは Python や他のライブラリの最新バージョンが含まれるとは限りませんし、コア Python チームはこれらを保守もしませんしサポートもしませんのでご理解ください。

4.6 Python を構成する

コマンドプロンプトより便利に Python を実行するために、Windows のデフォルトの環境変数をいくつか変えたいと思うかもしれません。インストーラは `PATH` と `PATHEXT` 変数を構成させるオプションを提供してはいますが、これは単独のシステムワイドなインストレーションの場合にだけ頼りになるものです。もしもあなたが定常的に複数バージョンの Python を使うのであれば、*Windows の Python ランチャ* の利用を検討してください。

4.6.1 補足: 環境変数の設定

Windows では、環境変数を恒久的にユーザレベルとシステムレベルの両方で設定でき、あるいはコマンドプロンプトから一時的にも設定できます。

一時的に環境変数を設定するには、コマンドプロンプトを開き `set` コマンドを使います:

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

これらの変更は、以降に実行される同じコンソール内で実行される任意のコマンドに適用され、また、そのコンソールから開始するすべてのアプリケーションに引き継がれます。

パーセント記号で変数名を囲んだものは既存の変数の値で展開されるので、新しい値を最初にも最後にも追加することができます。`python.exe` が入っているディレクトリを `PATH` に追加することは、Python の適切なバージョンが起動するように保証するための一般的な方法です。

デフォルトの環境変数を恒久的に変更するには、「スタート」をクリックして検索ボックスで「環境変数を編集」を検索するか、(コンピュータのプロパティなどから) **システムの詳細設定** を開いて **環境変数の設定** ボタンをクリックしてください。これで立ち上がるダイアログで、ユーザ環境変数とシステム環境変数を追加したり修正したりできます。システム変数を変更するにはあなたのマシンへの制限のないアクセス (つまり管理者権限) が必要です。

注釈

Windows はシステム変数の **後ろに** ユーザ変数を結合します。この振る舞いにより `PATH` の修正時に期待とは異なる結果になることがあります。

`PYTHONPATH` 変数は Python のすべてのバージョンで使われるので、インストールされているすべての Python バージョンに互換性のあるコードだけがパスの一覧に含まれているのでない限り、これは恒久的な設定をすべきではありません。

参考

https://learn.microsoft.com/windows/win32/procthread/environment-variables
Windows の環境変数の概要

https://learn.microsoft.com/windows-server/administration/windows-commands/set_1 一時的に環境変数を変更するための `set` コマンドについて

https://learn.microsoft.com/windows-server/administration/windows-commands/setx 恒久的に環境変数を変更するための `setx` コマンドについて。

4.6.2 Python 実行ファイルを見つける

バージョン 3.5 で変更.

自動的に作成される Python インタープリタのスタートメニュー項目を使うだけでなく、Python をコマンドプロンプトから起動したいと思うかもしれません。インストーラにはそのための設定を行うオプションがあります。

インストーラの最初のページに "Add Python to PATH" というラベルのオプションがあり、これを選択するとインストーラはインストール場所を環境変数 `PATH` に追加します。`Scripts\` フォルダの場所も追加されます。これによりコマンドプロンプトから `python` とタイプしてインタプリタを起動したり、`pip` とタイプしてパッケージインストーラを起動したりできます。コマンドラインからの起動なので、スクリプトをコマンドライン引数付きで起動することもできます。**コマンドライン** の文章を参照して下さい。

インストール時にこのオプションを有効にしていなかったとしても、インストーラを再度実行して「Modify」を選んで、それを有効にし直せます。あるいはそうせずとも、PATH 変数は手動で修正できます。**補足: 環境変数の設定** を参照してください。環境変数 PATH には Python インストールディレクトリを含む必要があります。ほかのエントリとはセミコロンで区切って繋いでください。これの実例は以下のようになります (以下最初の 2 つのエントリは既に存在しているものと仮定しています):

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

4.7 UTF-8 モード

Added in version 3.7.

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

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

You can use the Python UTF-8 Mode to change the default text encoding to UTF-8. You can enable the Python UTF-8 Mode via the `-X utf8` command line option, or the `PYTHONUTF8=1` environment variable. See *PYTHONUTF8* for enabling UTF-8 mode, and **補足: 環境変数の設定** for how to modify environment variables.

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

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

注釈

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

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

4.8 Windows の Python ランチャ

Added in version 3.3.

Windows の Python ランチャは、異なる Python のバージョンの位置の特定と実行を助けるユーティリティです。スクリプト (またはコマンドライン) で特定の Python のバージョンの設定を与えられると、位置を特定し、そのバージョンを実行します。

環境変数 PATH による方法と違って、このランチャは Python の一番適切なバージョンを、正しく選択しま

す。このランチャはシステムワイドなものよりもユーザごとのインストレーションの方を優先し、また、新しくインストールされた順よりも言語のバージョンを優先します。

ランチャのオリジナルの仕様は **PEP 397** にあります。

4.8.1 最初に

コマンドラインから起動する

バージョン 3.6 で変更.

Python 3.3 とそれ以降のシステムワイドなインストールでは、ランチャーが `PATH` に追加されます。ランチャーは、入手可能なあらゆる Python のバージョンに互換性があるため、実際にどのバージョンの Python がインストールされているのかは重要ではありません。ランチャーが使えるかを確認するには以下のコマンドをコマンドプロンプトで実行してください:

```
py
```

インストールされている最新バージョンの Python が起動するはずです。通常どおりに終了することもできますし、追加のコマンドライン引数を指定して直接 Python に渡すこともできます。

複数のバージョンの Python (たとえば 3.7 と 3.14) がインストールされている場合は、Python 3.14 が起動することになります。Python 3.7 を起動したいなら、次のコマンドを実行してみてください:

```
py -3.7
```

インストールしてある Python 2 の最新バージョンを起動したい場合は、次のコマンドを実行してみてください:

```
py -2
```

以下のようなエラーが出るようであれば、ランチャはインストールされていません:

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

このコマンド:

```
py --list
```

これは、現在インストールされている Python のバージョンを表示します。

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

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

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

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

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

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

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

仮想環境 (Virtual environments)

Added in version 3.5.

(標準ライブラリの `venv` モジュールか外部ツール `virtualenv` で作った) 仮想環境がアクティブな状態で Python の明示的なバージョンを指定せずにランチャを起動すると、ランチャはグローバルなインタプリタではなくその仮想環境のものを実行します。グローバルなほうのインタプリタを実行するには、仮想環境の動作を停止するか、または明示的にグローバルな Python バージョンを指定してください。

スクリプトから起動する

テスト用の Python スクリプトを作成しましょう。`hello.py` という名前で以下の内容のファイルを作成してください

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

hello.py が存在するディレクトリで、下記コマンドを実行してください:

```
py hello.py
```

インストールされている最新の Python 2.x のバージョン番号が表示されるはずです。では、1 行目を以下のように変更してみてください:

```
#! python3
```

コマンドを再実行すると、今度は最新の Python 3.x の情報が表示されるはずです。これまでのコマンドラインの例と同様に、より細かいバージョン修飾子を指定することもできます。Python 3.7 がインストールされている場合、最初の行を `#! python3.7` に変更すると、3.7 のバージョン情報が表示されるはずです。

コマンドからの呼び出しとは異なり、後ろに何もつかない "python" はインストールされている Python2.x の最新バージョンを利用することに注意してください。これは後方互換性と、python が一般的に Python 2 を指す Unix との互換性のためです。

ファイルの関連付けから起動する

インストール時に、ランチャは Python ファイル (すなわち .py, .pyw, .pyc ファイル) に関連付けられたはずです。そのため、これらのファイルを Windows のエクスプローラーでダブルクリックした際はランチャが使われ、上で述べたのと同じ機能を使ってスクリプトが使われるべきバージョンを指定できるようになります。

このことによる重要な利点は、単一のランチャが先頭行の内容によって複数の Python バージョンを同時にサポートできることです。

4.8.2 シェバン (shebang) 行

スクリプトファイルの先頭の行が #! で始まっている場合は、その行はシェバン (shebang) 行として知られています。Linux や他の Unix 系 OS はこうした行をもともとサポートしているため、それらのシステムでは、スクリプトがどのように実行されるかを示すために広く使われます。Windows の Python ランチャは、Windows 上の Python スクリプトが同じ機能を使用できるようにし、上の例ではそれらの機能の使用法を示しています。

Python スクリプトのシェバン行を Unix-Windows 間で移植可能にするため、このランチャは、どのインタプリタが使われるかを指定するための大量の ' 仮想' コマンドをサポートしています。サポートされる仮想コマンドには以下のものがあります:

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

具体的に、もしスクリプトの 1 行目が

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

で始まっていたら、デフォルトの Python またはアクティブな仮想環境の位置が特定され、使用されます。多くの Unix 上で動作する Python スクリプトにはすでにこの行が存在する傾向がありますので、ランチャによりそれらのスクリプトを修正なしで使うことができるはずです。あなたが新しいスクリプトを Windows 上で書いていて、Unix 上でも有用であってほしいと思うなら、シェバン行のうち /usr で始まるものを使用すべきです。

上記のどの仮想コマンドでも、(メジャーバージョンだけや、メジャー・マイナーバージョンの両方で) 明示的にバージョンを指定できます。さらに、"-32" をマイナーバージョンの後ろに追加して 32-bit 版を要求できます。例えば、/usr/bin/python3.7-32 は 32-bit の Python 3.7 を使うよう要求します。仮想環境がアクティブになっている場合は、バージョンは無視され、その環境が使用されます。

Added in version 3.7: python ランチャの 3.7 からは、末尾に "-64" を付けて 64-bit 版を要求できます。さら

に、マイナーバージョン無しのメジャーバージョンとアーキテクチャだけ (例えば、`/usr/bin/python3-64`) で指定できます。

バージョン 3.11 で変更: The "-64" suffix is deprecated, and now implies "any architecture that is not provably i386/32-bit". To request a specific environment, use the new `-V:TAG` argument with the complete tag.

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

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

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

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

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

4.8.3 シェバン行の引数

シェバン行では Python インタプリタに渡される追加の引数を指定することもできます。たとえば、シェバン行に以下のように書かれているとしましょう:

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

この場合、Python は `-v` オプション付きで起動するでしょう

4.8.4 カスタマイズ

INI ファイルによるカスタマイズ

ランチャは 2 つの .ini ファイルを探しに行きます。具体的には、現在のユーザーのアプリケーションデータディレクトリ (`%LOCALAPPDATA%` または `$env:LocalAppData`) の `py.ini` と、ランチャと同じディレクトリにある `py.ini` です。' コンソール' 版のランチャ (つまり py.exe) と 'Windows' 版のランチャ (つまり pyw.exe) は同一の .ini ファイルを使用します。

"application data" ディレクトリで指定された設定は、実行ファイルの隣にあるものより優先されます。そのため、ランチャの隣にある .ini ファイルへの書き込みアクセスができないユーザは、グローバルな .ini ファイル内のコマンドを上書き (override) できます。

デフォルトの Python バージョンのカスタマイズ

どのバージョンの Python をコマンドで使用するかを定めるため、バージョン修飾子がコマンドに含められることがあります。バージョン修飾子はメジャーバージョン番号で始まり、オプションのピリオド (".") とマイナーバージョン指定子がそれに続きます。さらに、"-32" や "-64" を追記して 32-bit あるいは 64-bit のどちらの実装が要求されるかを指示できます。

たとえば、`#!python` というシェバン行はバージョン修飾子を含みませんが、`#!python3` はメジャーバージョンを指定するバージョン修飾子を含みます。

コマンドにバージョン修飾子が見つからない場合、環境変数 `PY_PYTHON` を設定して、デフォルトのバージョン修飾子を指定できます。設定されていない場合、デフォルト値は "3" です。この変数には "3", "3.7", "3.7-32", "3.7-64" のような任意の値をコマンドラインから指定できます。("-64" オプションは Python 3.7 以降のランチャでしか使えないことに注意してください。)

マイナーバージョン修飾子が見つからない場合、環境変数 `PY_PYTHON{major}` (ここで `{major}` は、上記で決定された現在のメジャーバージョン修飾子) を設定して完全なバージョンを指定することができます。そういったオプションが見つからなければ、ランチャはインストール済みの Python バージョンを列挙して、見つかったそのメジャーバージョン向けマイナーリリースのうち最新のものを使用します。保証されているわけではありませんが、通常はそのメジャーバージョン系で最も後にインストールしたバージョンになります。

64-bit Windows で、同一の (major.minor) Python バージョンの 32-bit と 64-bit の両方の実装がインストールされていた場合、64-bit バージョンのほうが常に優先されます。これはランチャが 32-bit と 64-bit のどちらでも言えることで、32-bit のランチャは、指定されたバージョンが使用可能であれば、64-bit の Python を優先して実行します。これは、どのバージョンが PC にインストールされているかのみでランチャの挙動を予見でき、それらがインストールされた順番に関係なくなる (つまり最後にインストールされた Python とランチャが 32-bit か 64-bit かを知らなくともよい) ようにするためです。上に記したとおり、オプションの "-32", "-64" サフィックスでこの挙動を変更できます。

例:

- 関連するオプションが設定されていない場合、`python` および `python2` コマンドはインストールされている最新の Python 2.x バージョンを使用し、`python3` コマンドはインストールされている最新の Python 3.x を使用します。
- `python3.7` コマンドは、バージョンが完全に指定されているため、全くオプションを参照しません。

- PY_PYTHON=3 の場合、python および python3 コマンドはともにインストールされている最新の Python 3 を使用します。
- PY_PYTHON=3.7-32 の場合、python コマンドは 32-bit 版の 3.7 を使用しますが、python3 コマンドはインストールされている最新の Python を使用します (メジャーバージョンが指定されているため、PY_PYTHON は全く考慮されません。)
- PY_PYTHON=3 で PY_PYTHON3=3.7 の場合、python および python3 はどちらも 3.7 を使用します

環境変数に加え、同じ設定をランチャが使う INI ファイルで構成することができます。INI ファイルの該当するセクションは [defaults] と呼ばれ、キー名は環境変数のキー名から PY_ という接頭辞を取ったものと同じです (INI ファイルのキー名は大文字小文字を区別しないことにご注意ください)。環境変数の内容は INI ファイルでの指定を上書きします。

例えば:

- PY_PYTHON=3.7 と設定することは、INI ファイルに下記が含まれることと等価です:

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

- PY_PYTHON=3 と PY_PYTHON3=3.7 を設定することは、INI ファイルに下記が含まれることと等価です:

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

4.8.5 診断

環境変数 PYLAUNCHER_DEBUG が設定されていたら (設定値が何であっても)、ランチャは診断情報を stderr (つまりコンソール) に出力します。この情報のメッセージは詳細で **しかも** きついものですが、どういったバージョンの Python が検知されたか、なぜ特定のバージョンが選択されたか、そして、対象の Python を実行するのに使われた正確なコマンドラインを教えてくれます。これは主にテストやデバッグのためのものです。

4.8.6 Dry Run

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

4.8.7 Install on demand

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

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

4.8.8 Return codes

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

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

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

4.9 モジュールの検索

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

._pth ファイルが見付からなかったときは、Windows では sys.path は次のように設定されます:

- 最初に空のエントリが追加されます。これはカレントディレクトリを指しています。
- その次に、*PYTHONPATH* 環境変数が存在するとき、**環境変数** で解説されているように追加されます。Windows ではドライブ識別子 (C:\ など) と区別するために、この環境変数に含まれるパスの区切り文字はセミコロンでなければならない事に注意してください。
- 追加で ” アプリケーションのパス” を HKEY_CURRENT_USER か HKEY_LOCAL_MACHINE の中の \SOFTWARE\Python\PythonCore{version}\PythonPath のサブキーとして登録することができます。サブキーはデフォルト値としてセミコロンで区切られたパス文字列を持つことができ、書くパスが sys.path に追加されます。(既存のインストーラーはすべて HKLM しか利用しないので、HKCU は

通常空です)

- *PYTHONHOME* が設定されている場合、それが "Python Home" として扱われます。それ以外の場合、"Python Home" を推定するために Python の実行ファイルのパスから " 目印ファイル" (`Lib\os.py` または `pythonXY.zip`) が探されます。Python home が見つかった場合、そこからいくつかのサブディレクトリ (`Lib`, `plat-win` など) が `sys.path` に追加されます。見つからなかった場合、コアとなる Python path はレジストリに登録された PythonPath から構築されます。
- Python Home が見つからず、環境変数 *PYTHONPATH* が指定されず、レジストリエントリが見つからなかった場合、関連するデフォルトのパスが利用されます (例: `.\Lib;.\plat-win` など)。

メインの実行ファイルと同じ場所か一つ上のディレクトリに `pyvenv.cfg` がある場合、以下の異なった規則が適用されます:

- *PYTHONHOME* が設定されておらず、`home` が絶対パスの場合、home 推定の際メインの実行ファイルから推定するのではなくこのパスを使います。

結果としてこうなります:

- `python.exe` かそれ以外の Python ディレクトリにある .exe ファイルを実行したとき (インストールされている場合でも PCbuild から直接実行されている場合でも) core path が利用され、レジストリ内の core path は無視されます。それ以外のレジストリの "application paths" は常に読み込まれます。
- Python が他の .exe ファイル (他のディレクトリに存在する場合や、COM 経由で組み込まれる場合など) にホストされている場合は、"Python Home" は推定されず、レジストリにある core path が利用されます。それ以外のレジストリの "application paths" は常に読み込まれます。
- Python がその home を見つけられず、レジストリの値もない場合 (これはいくつかのとてもおかしなインストレーションセットアップの凍結された .exe)、パスは最小限のデフォルトとして相対パスが使われます。

自身のアプリケーションや配布物に Python をバンドルしたい場合には、以下の助言 (のいずれかまたは組合せ) によりほかのインストレーションとの衝突を避けることができます:

- Include a `._pth` file alongside your executable containing the directories to include. This will ignore paths listed in the registry and environment variables, and also ignore `site` unless `import site` is listed.
- If you are loading `python3.dll` or `python37.dll` in your own executable, explicitly set `PyConfig.module_search_paths` before `Py_InitializeFromConfig()`.
- 自身のアプリケーションから `python.exe` を起動する前に、*PYTHONPATH* をクリアしたり上書きし、*PYTHONHOME* をセットしてください。
- If you cannot use the previous suggestions (for example, you are a distribution that allows people to run `python.exe` directly), ensure that the landmark file (`Lib\os.py`) exists in your install directory. (Note that it will not be detected inside a ZIP file, but a correctly named ZIP file will be detected instead.)

これらはシステムワイドにインストールされたファイルが、あなたのアプリケーションにバンドルされた標準

ライブラリのコピーに優先しないようにします。これをしなければあなたのアプリケーションのユーザは、何かしら問題を抱えるかもしれません。上で列挙した最初の提案が最善です。ほかのものはレジストリ内の非標準のパスやユーザの site-packages の影響を少し受けやすいからです。

バージョン 3.6 で変更: Add `._pth` file support and removes `applocal` option from `pyvenv.cfg`.

バージョン 3.6 で変更: Add `python*XX*.zip` as a potential landmark when directly adjacent to the executable.

バージョン 3.6 で非推奨: Modules specified in the registry under `Modules` (not `PythonPath`) may be imported by `importlib.machinery.WindowsRegistryFinder`. This finder is enabled on Windows in 3.6.0 and earlier, but may need to be explicitly added to `sys.meta_path` in the future.

4.10 追加のモジュール

Python は全プラットフォーム互換を目指していますが、Windows にしかないユニークな機能もあります。標準ライブラリと外部のライブラリの両方で、幾つかのモジュールと、そういった機能を使うためのスニペットがあります。

Windows 固有の標準モジュールは、mswin-specific-services に書かれています。

4.10.1 PyWin32

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

- Component Object Model (COM)
- Win32 API 呼び出し
- レジストリ
- イベントログ
- Microsoft Foundation Classes (MFC) user interfaces

PythonWin は PyWin32 に付属している、サンプルの MFC アプリケーションです。これはビルトインのデバッガを含む、組み込み可能な IDE です。

参考

Win32 How Do I...?
by Tim Golden

Python and COM
by David and Paul Boddie

4.10.2 cx_Freeze

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

4.11 Windows 上で Python をコンパイルする

CPython を自分でコンパイルしたい場合、最初にすべきことは ソース を取得することです。最新リリース版のソースか、新しい チェックアウト をダウンロードできます。

ソースツリーには Microsoft Visual Studio でのビルドのソリューションファイルとプロジェクトファイルが含まれていて、これが公式の Python リリースに使われているコンパイラです。これらファイルは `PCbuild` ディレクトリ内にあります。

ビルドプロセスについての一般的な情報は、`PCbuild/readme.txt` にあります。

拡張モジュールについては、building-on-windows を参照してください。

4.12 ほかのプラットフォーム

Python の継続的な開発の中で、過去にサポートされていた幾つかのプラットフォームが (ユーザーや開発者の不足のために) サポートされなくなっています。すべてのサポートされないプラットフォームについての詳細は **PEP 11** をチェックしてください。

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

コンパイル済みインストーラが提供されているプラットフォームについての詳細な情報は Python for Windows を参照してください。

第

FIVE

USING PYTHON ON MACOS

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

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

5.1 Using Python for macOS from `python.org`

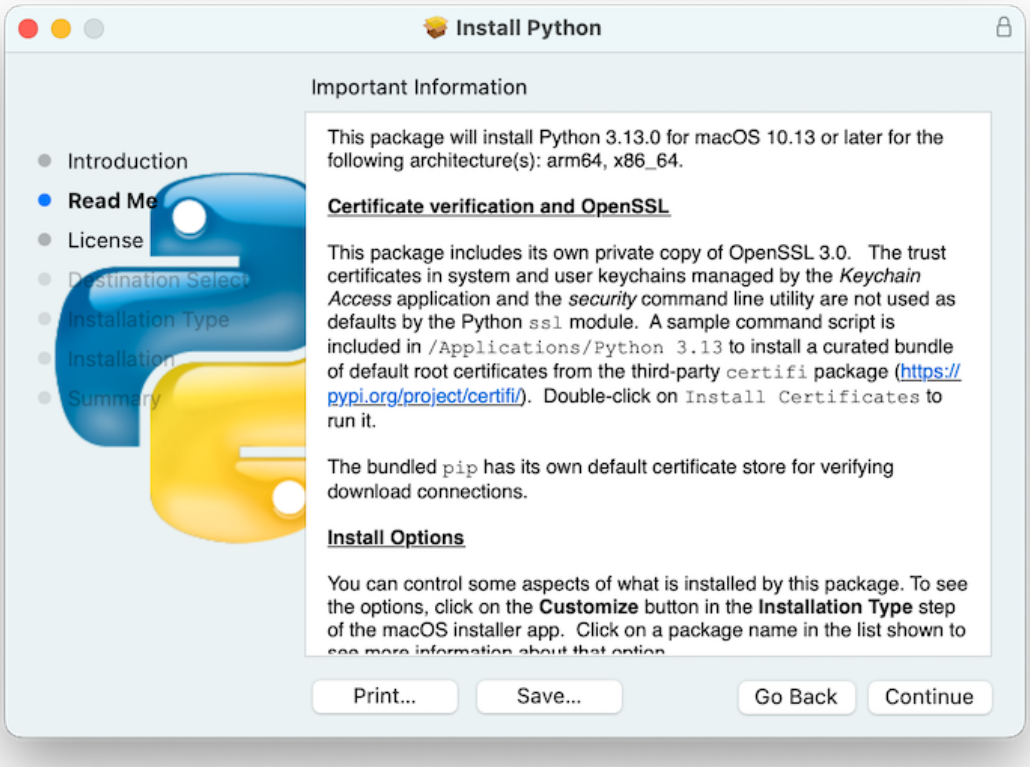
5.1.1 インストール手順

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

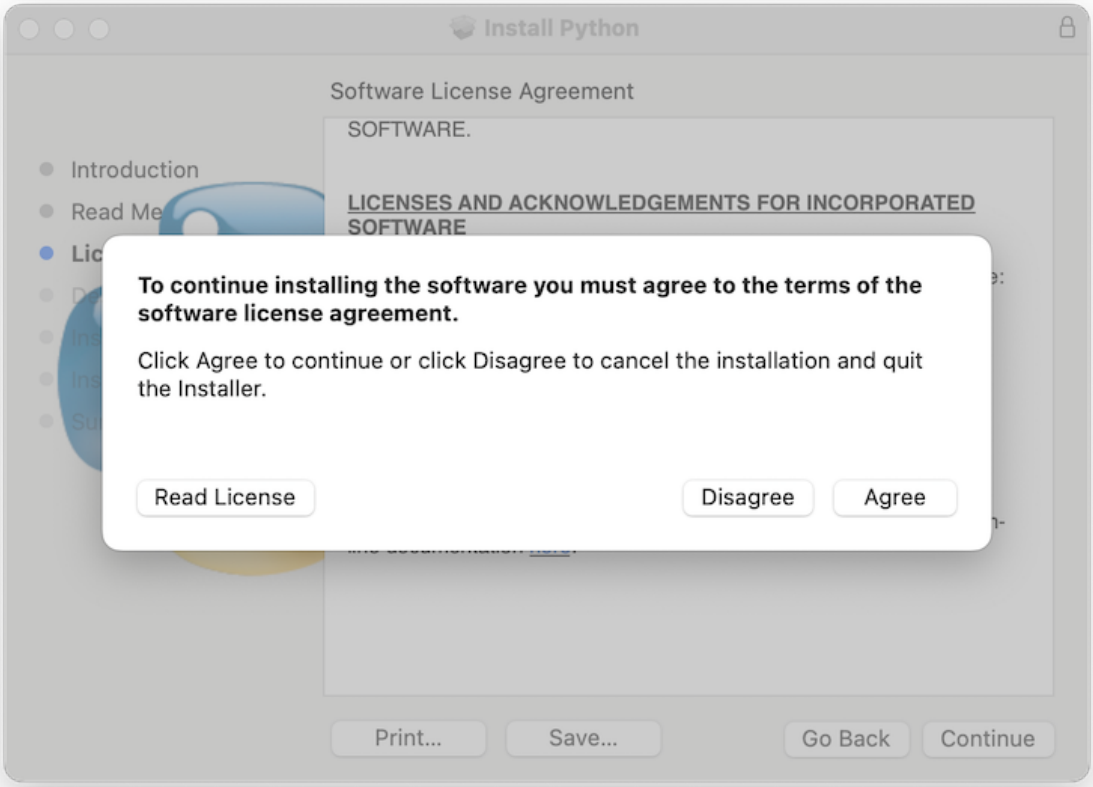
The downloaded file is a standard macOS installer package file (`.pkg`). File integrity information (checksum, size, sigstore signature, etc) for each file is included on the release download page. Installer packages and their contents are signed and notarized with `Python Software Foundation` Apple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

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

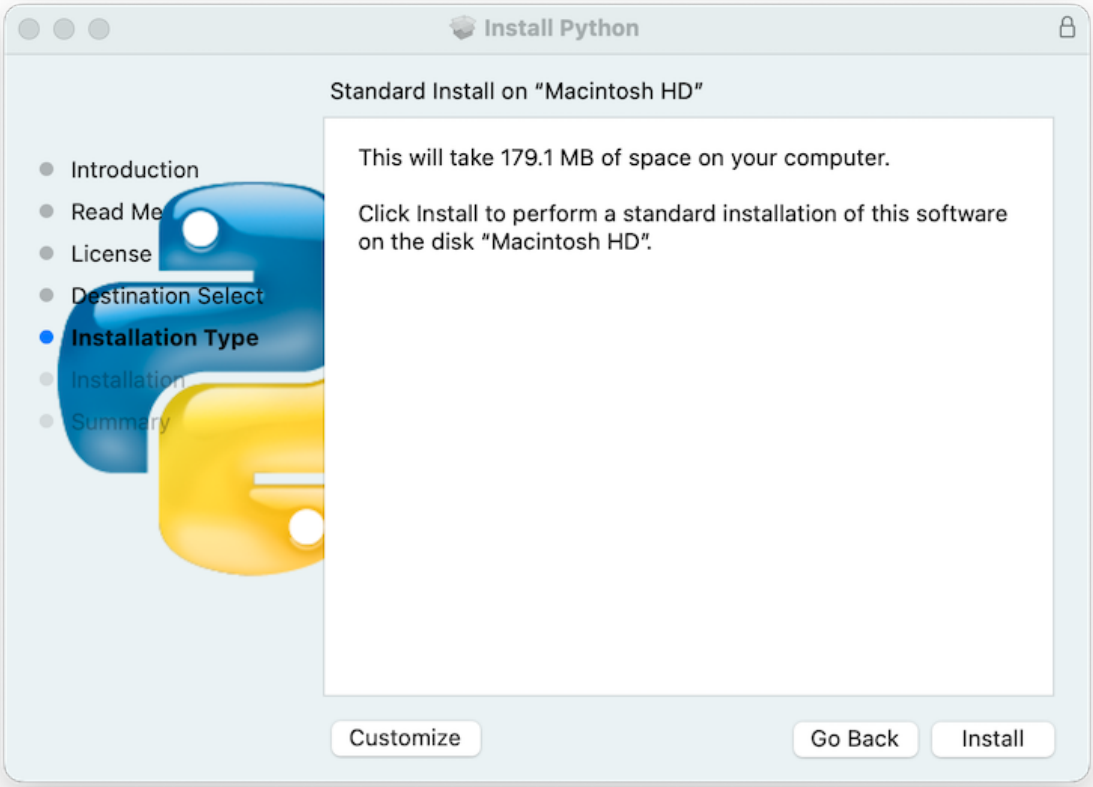
Clicking on the **Continue** button brings up the **Read Me** for this installer. Besides other important information, the **Read Me** documents which Python version is going to be installed and on what versions of macOS it is supported. You may need to scroll through to read the whole file. By default,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



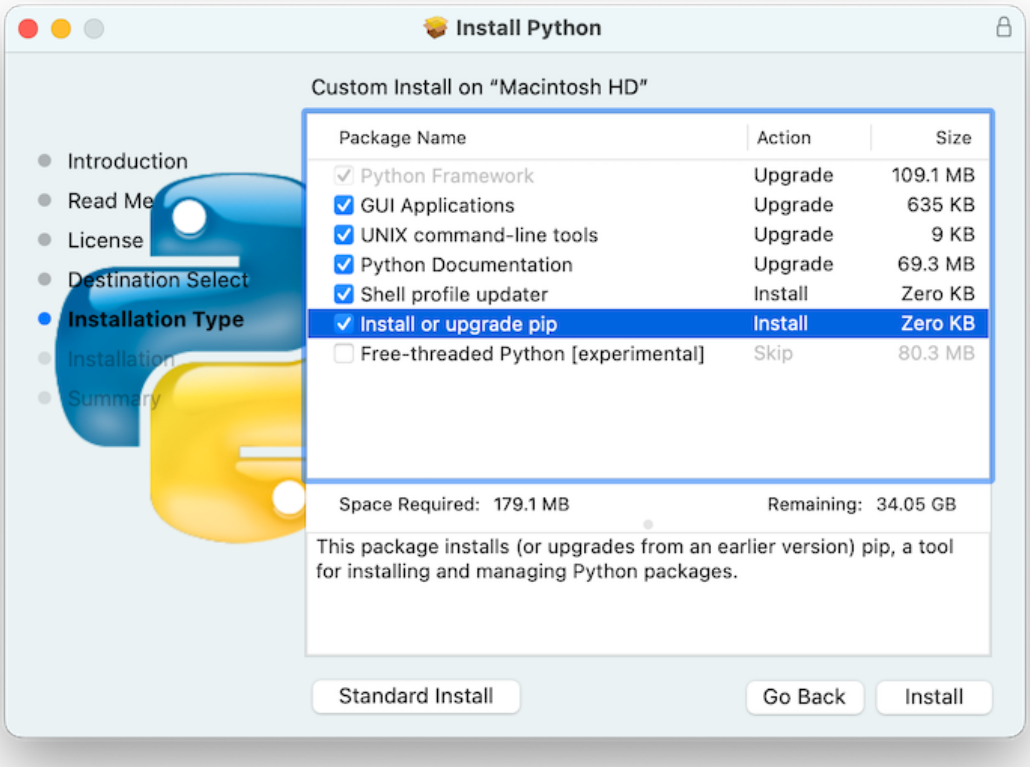
Clicking on **Continue** proceeds to display the license for Python and for other included software. You will then need to **Agree** to the license terms before proceeding to the next step. This license file will also be installed and available to be read later.



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



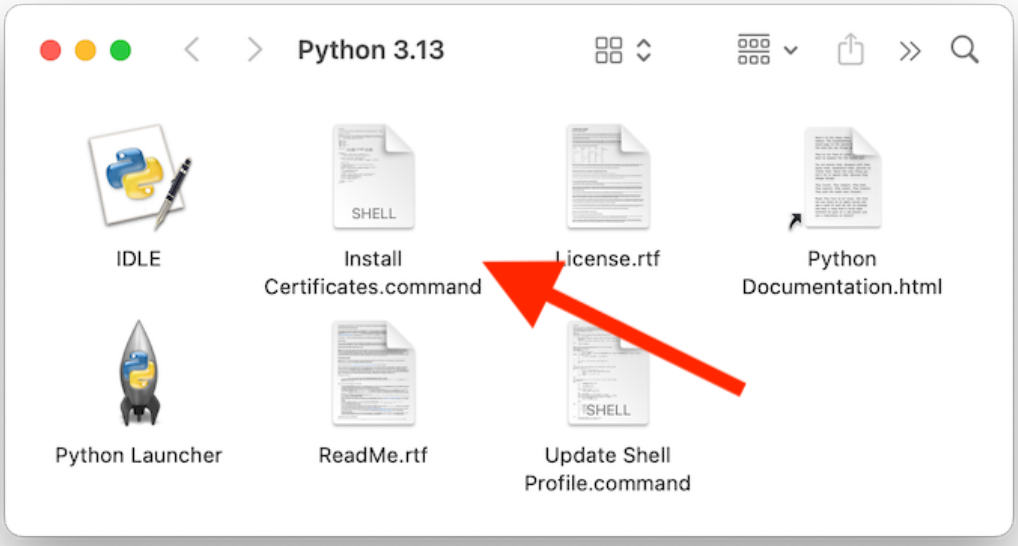
By pressing the **Customize** button, you can choose to omit or select certain package components of the installer. Click on each package name to see a description of what it installs. To also install support for the optional experimental free-threaded feature, see フリースレッドバイナリ *(Free-threaded Binaries)* のインストール.



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

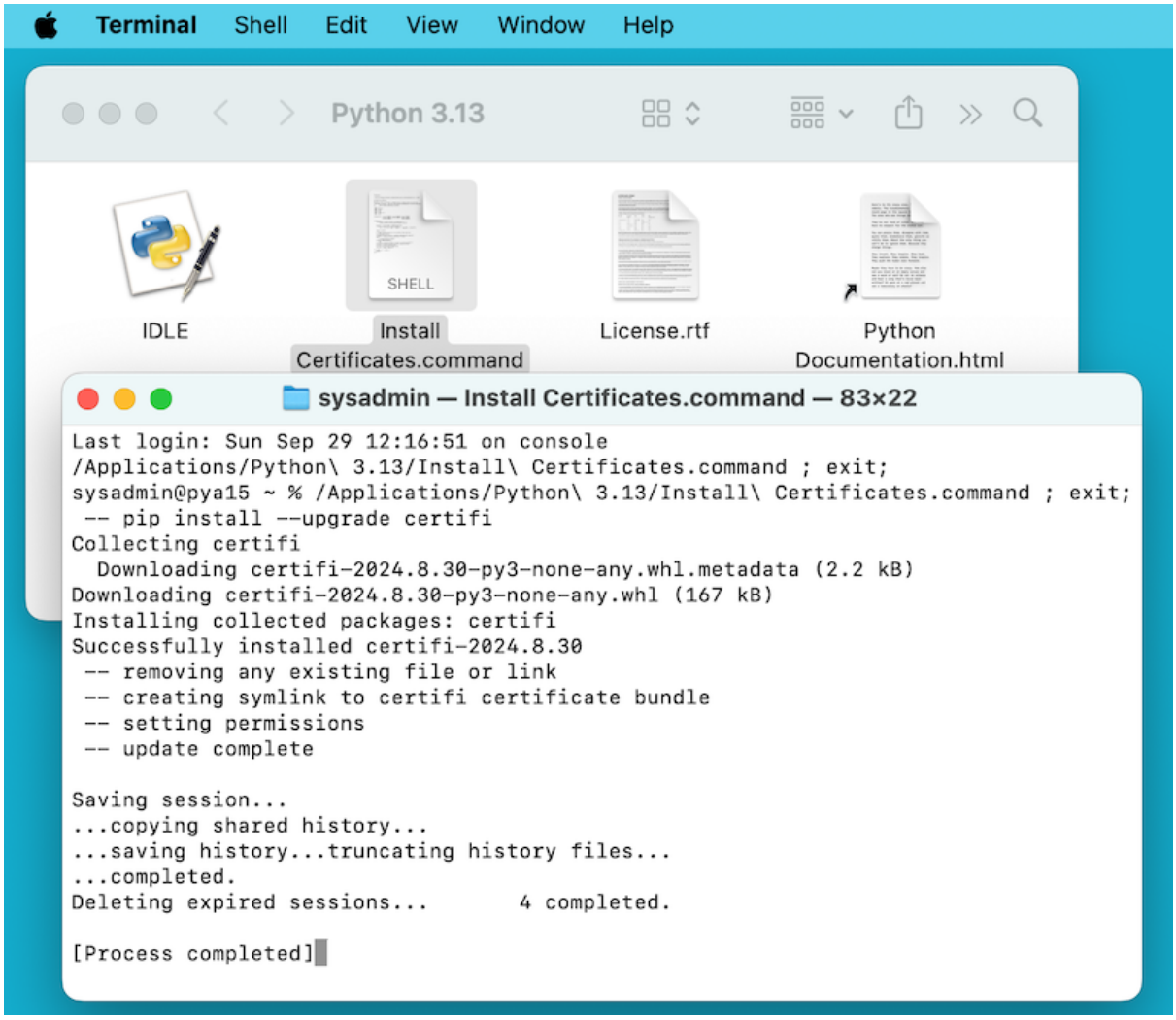
When the installation is complete, the **Summary** window will appear.

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



This will open a temporary `Terminal` shell window that will use the new Python to download and install

SSL root certificates for its use.



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

A default install will include:

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

注釈

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 Python スクリプトの実行方法

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 コマンドラインと環境). 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 エディタと *IDE* for more information.

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

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

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

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

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

5.2 Alternative Distributions

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

ActivePython
: Installer with multi-platform compatibility, documentation

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

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

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

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

5.3 追加の Python パッケージのインストール

Refer to the Python Packaging User Guide for more information.

5.4 GUI プログラミング

Python で Mac 上の GUI アプリケーションをビルドする方法がいくつかあります。

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

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

A number of alternative macOS GUI toolkits are available including:

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

5.5 高度なトピック

5.5.1 フリースレッドバイナリ (Free-threaded Binaries) のインストール

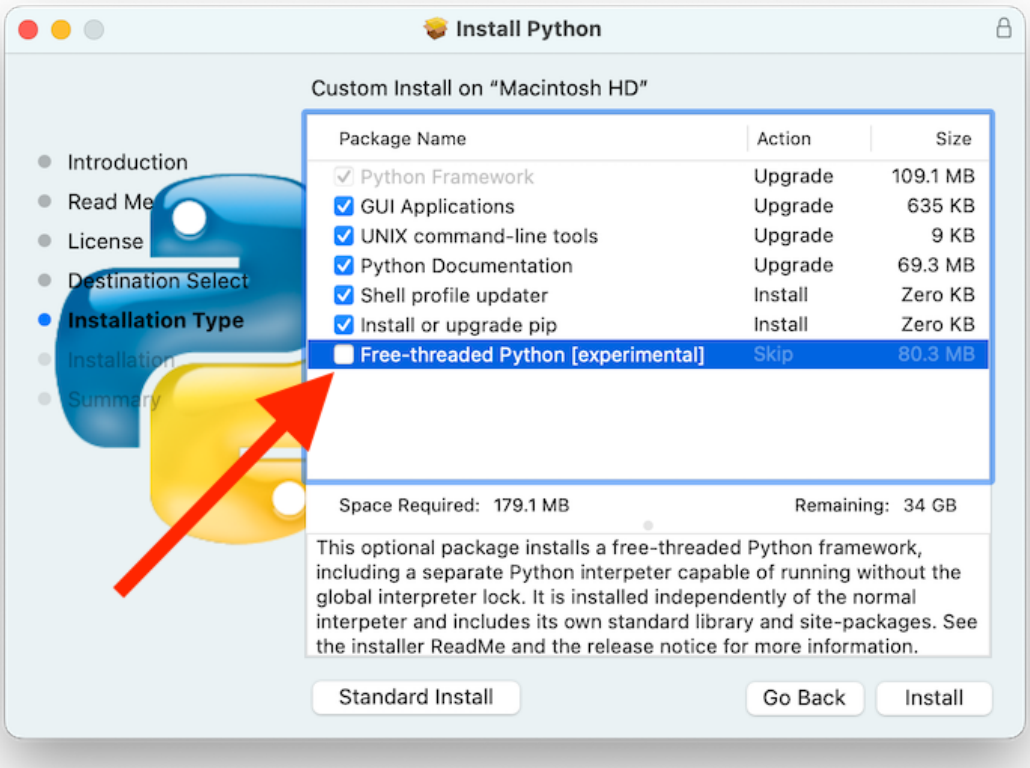
Added in version 3.13: (実験的)

注釈

このセクションで説明されているすべての項目は実験的とみなされており、将来のリリースで変更される可能性があります。

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

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



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

Known cautions and limitations:

- The **UNIX command-line tools** package, which is selected by default, will install links in `/usr/local/bin` for `python3.13t`, the free-threaded interpreter, and `python3.13t-config`, a configuration utility which may be useful for package builders. Since `/usr/local/bin` is typically included in your shell `PATH`, in most cases no changes to your `PATH` environment variables should be needed to use `python3.13t`.
- For this release, the **Shell profile updater** package and the `Update Shell Profile.command` in `/Applications/Python 3.13/` do not support the free-threaded package.

- The free-threaded build and the traditional build have separate search paths and separate `site-packages` directories so, by default, if you need a package available in both builds, it may need to be installed in both. The free-threaded package will install a separate instance of **pip** for use with `python3.13t`.
  - To install a package using **pip** without a **venv**:

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

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

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

  then **activate**.

- To run a free-threaded version of IDLE:

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

- The interpreters in both builds respond to the same *PYTHON environment variables* which may have unexpected results, for example, if you have `PYTHONPATH` set in a shell profile. If necessary, there are *command line options* like `-E` to ignore these environment variables.

- The free-threaded build links to the third-party shared libraries, such as `OpenSSL` and `Tk`, installed in the traditional framework. This means that both builds also share one set of trust certificates as installed by the **Install Certificates.command** script, thus it only needs to be run once.

- If you cannot depend on the link in `/usr/local/bin` pointing to the `python.org` free-threaded `python3.13t` (for example, if you want to install your own version there or some other distribution does), you can explicitly set your shell `PATH` environment variable to include the `PythonT` framework `bin` directory:

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

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

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

5.5.2 Installing using the command line

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

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

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

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

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

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

```
$ # test that the free-threaded interpreter was installed if the Unix Command Tools␣
↪package was enabled
$ /usr/local/bin/python3.13t -VV
Python 3.13.0b2 experimental free-threading build (v3.13.0b2:3a83b172af, Jun  5 2024,␣
↪12:57:31) [Clang 15.0.0 (clang-1500.3.9.4)]
```

(次のページに続く)

(前のページからの続き)

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

注釈

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

5.5.3 Python アプリケーションの配布

Python コードをスタンドアローンな配布アプリケーションに変換するツールには様々なものがあります:

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

5.5.4 App Store コンプライアンス

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

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

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

This patch is not normally required to use CPython on a Mac; nor is it required if you are distributing

an app *outside* the macOS App Store. It is *only* required if you are using the macOS App Store as a distribution channel.

5.6 他のリソース

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.

第

SIX

USING PYTHON ON ANDROID

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

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

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

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

6.1 Adding Python to an Android app

These instructions are only needed if you're planning to compile Python for Android yourself. Most users should *not* need to do this. Instead, use one of the following tools, which will provide a much easier experience:

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

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

- Build Python by following the instructions in Android/README.md.

- Add code to your build.gradle file to copy the following items into your project. All except your own Python code can be copied from `cross-build/HOST/prefix/lib`:
  - In your JNI libraries:
    - `libpython*.*.so`
    - `lib*_python.so` (external libraries such as OpenSSL)
  - In your assets:
    - `python*.*` (the Python standard library)
    - `python*.*/site-packages` (your own Python code)
- Add code to your app to extract the assets to the filesystem.
- Add code to your app to start Python in embedded mode. This will need to be C code called via JNI.

第

SEVEN

USING PYTHON ON IOS

Authors

Russell Keith-Magee (2024-03)

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

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

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

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

7.1 Python at runtime on iOS

7.1.1 iOS version compatibility

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

7.1.2 Platform identification

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

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

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

7.1.3 Standard library availability

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

7.1.4 Binary extension modules

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

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

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

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

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

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

7.1.5 Compiler stub binaries

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

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

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

7.2 Installing Python on iOS

7.2.1 Tools for building iOS apps

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

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

7.2.2 Adding Python to an iOS project

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

To add Python to an iOS Xcode project:

1. Build or obtain a Python `XCFramework`. See the instructions in iOS/README.rst (in the CPython source distribution) for details on how to build a Python `XCFramework`. At a minimum, you will need a build that supports `arm64-apple-ios`, plus one of either `arm64-apple-ios-simulator` or

x86_64-apple-ios-simulator.

2. Drag the `XCframework` into your iOS project. In the following instructions, we'll assume you've dropped the `XCframework` into the root of your project; however, you can use any other location that you want by adjusting paths as needed.

3. Drag the `iOS/Resources/dylib-Info-template.plist` file into your project, and ensure it is associated with the app target.

4. Add your application code as a folder in your Xcode project. In the following instructions, we'll assume that your user code is in a folder named `app` in the root of your project; you can use any other location by adjusting paths as needed. Ensure that this folder is associated with your app target.

5. Select the app target by selecting the root node of your Xcode project, then the target name in the sidebar that appears.

6. In the "General" settings, under "Frameworks, Libraries and Embedded Content", add `Python.xcframework`, with "Embed & Sign" selected.

7. In the "Build Settings" tab, modify the following:

   - Build Options
     - User Script Sandboxing: No
     - Enable Testability: Yes
   - Search Paths
     - Framework Search Paths: `$(PROJECT_DIR)`
     - Header Search Paths: `"$(BUILT_PRODUCTS_DIR)/Python.framework/Headers"`
   - Apple Clang - Warnings - All languages
     - Quoted Include In Framework Header: No

8. Add a build step that copies the Python standard library into your app. In the "Build Phases" tab, add a new "Run Script" build step *before* the "Embed Frameworks" step, but *after* the "Copy Bundle Resources" step. Name the step "Install Target Specific Python Standard Library", disable the "Based on dependency analysis" checkbox, and set the script content to:

```
set -e

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

(次のページに続く)

(前のページからの続き)

```
    echo "Installing Python modules for iOS Device"
    rsync -au --delete "$PROJECT_DIR/Python.xcframework/ios-arm64/lib/" "
↪$CODESIGNING_FOLDER_PATH/python/lib/"
fi
```

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

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

```
set -e

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

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

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

(次のページに続く)

(前のページからの続き)

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

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

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

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

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

    - UTF-8 mode (`PyPreConfig.utf8_mode`) is *enabled*;
    - Buffered stdio (`PyConfig.buffered_stdio`) is *disabled*;
    - Writing bytecode (`PyConfig.write_bytecode`) is *disabled*;
    - Signal handlers (`PyConfig.install_signal_handlers`) are *enabled*;
    - System logging (`PyConfig.use_system_logger`) is *enabled* (optional, but strongly recommended);

- PYTHONHOME for the interpreter is configured to point at the python subfolder of your app's bundle; and
- The PYTHONPATH for the interpreter includes:
  - the python/lib/python3.X subfolder of your app's bundle,
  - the python/lib/python3.X/lib-dynload subfolder of your app's bundle, and
  - the app subfolder of your app's bundle

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

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

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

7.2.3 Testing a Python package

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

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

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

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

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

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

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

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

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

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

7.3 App Store Compliance

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

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

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

第 EIGHT

エディタと IDE

Python プログラミング言語をサポートする IDE はたくさんあります。多くのエディタや IDE にはシンタックスハイライト機能、デバッグツール、**PEP 8** チェック機能があります。

包括的な一覧を見るには、Python Editors や Integrated Development Environments を訪れてください。

付録

A

用語集

>>> 対話型 (*interactive*) シェルにおけるデフォルトの Python プロンプトです。インタープリターで対話的に実行されるコード例でよく見られます。

... 次のものが考えられます:

- 対話型 (*interactive*) シェルにおいて、インデントされたコードブロック、対応する左右の区切り文字の組 (丸括弧、角括弧、波括弧、三重引用符) の内側、デコレーターの後に、コードを入力する際に表示されるデフォルトの Python プロンプトです。
- 組み込みの定数 `Ellipsis` 。

abstract base class

(抽象基底クラス) 抽象基底クラスは *duck-typing* を補完するもので、`hasattr()` などの別のテクニックでは不恰好であったり微妙に誤る (例えば magic methods の場合) 場合にインターフェースを定義する方法を提供します。ABC は仮想 (virtual) サブクラスを導入します。これは親クラスから継承しませんが、それでも `isinstance()` や `issubclass()` に認識されます; `abc` モジュールのドキュメントを参照してください。Python には、多くの組み込み ABC が同梱されています。その対象は、(`collections.abc` モジュールで) データ構造、(`numbers` モジュールで) 数、(`io` モジュールで) ストリーム、(`importlib.abc` モジュールで) インポートファインダ及びローダーです。`abc` モジュールを利用して独自の ABC を作成できます。

annotate function

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

annotation

(アノテーション) 変数、クラス属性、関数のパラメータや返り値に関係するラベルです。慣例により *type hint* として使われています。

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

See *variable annotation*, *function annotation*, **PEP 484**, **PEP 526**, and **PEP 649**, which describe this functionality. Also see annotations-howto for best practices on working with annotations.

引数 (argument)

(実引数) 関数を呼び出す際に、**関数** (または **メソッド**) に渡す値です。実引数には 2 種類あります:

- **キーワード引数**: 関数呼び出しの際に引数の前に識別子がついたもの (例: `name=`) や、`**` に続けた辞書の中の値として渡された引数。例えば、次の `complex()` の呼び出しでは、`3` と `5` がキーワード引数です:

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

- **位置引数**: キーワード引数以外の引数。位置引数は引数リストの先頭に書くことができ、また `*` に続けた *iterable* の要素として渡すことができます。例えば、次の例では `3` と `5` は両方共位置引数です:

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

実引数は関数の実体において名前付きのローカル変数に割り当てられます。割り当てを行う規則については calls を参照してください。シンタックスにおいて実引数を表すためにあらゆる式を使うことが出来ます。評価された値はローカル変数に割り当てられます。

仮引数 、FAQ の 実引数と仮引数の違いは何ですか? 、**PEP 362** を参照してください。

asynchronous context manager

(非同期コンテキストマネージャ) `__aenter__()` と `__aexit__()` メソッドを定義することで `async with` 文内の環境を管理するオブジェクトです。**PEP 492** で導入されました。

asynchronous generator

(非同期ジェネレータ) *asynchronous generator iterator* を返す関数です。`async def` で定義されたコルーチン関数に似ていますが、`yield` 式を持つ点で異なります。`yield` 式は `async for` ループで使用できる値の並びを生成するのに使用されます。

通常は非同期ジェネレータ関数を指しますが、文脈によっては **非同期ジェネレータイテレータ** を指す場合があります。意図された意味が明らかでない場合、明瞭化のために完全な単語を使用します。

非同期ジェネレータ関数には、`async for` 文や `async with` 文だけでなく `await` 式もあることがあります。

asynchronous generator iterator

(非同期ジェネレータイテレータ) *asynchronous generator* 関数で生成されるオブジェクトです。

これは *asynchronous iterator* で、`__anext__()` メソッドを使って呼ばれると awaitable オブジェクトを返します。この awaitable オブジェクトは、次の `yield` 式まで非同期ジェネレータ関数の本体を実行します。

各 `yield` では一時的に処理を中断し、その場の実行状態 (ローカル変数や保留中の try 文を含む) を

記憶します。**非同期ジェネレータイテレータ** が `__anext__()` で返された他の awaitable で実際に再開する時には、その中断箇所が選ばれます。**PEP 492** および **PEP 525** を参照してください。

asynchronous iterable

(非同期イテラブル) `async for` 文の中で使用できるオブジェクトです。自身の `__aiter__()` メソッドから *asynchronous iterator* を返さなければなりません。**PEP 492** で導入されました。

asynchronous iterator

(非同期イテレータ) `__aiter__()` と `__anext__()` メソッドを実装したオブジェクトです。`__anext__()` は *awaitable* オブジェクトを返さなければなりません。`async for` は `StopAsyncIteration` 例外を送出するまで、非同期イテレータの `__anext__()` メソッドが返す awaitable を解決します。**PEP 492** で導入されました。

属性

(属性) オブジェクトに関連付けられ、ドット表記式によって名前で通常参照される値です。例えば、オブジェクト *o* が属性 *a* を持っているとき、その属性は *o.a* で参照されます。

オブジェクトには、identifiers で定義される識別子ではない名前の属性を与えることができます。たとえば `setattr()` を使い、オブジェクトがそれを許可している場合に行えます。このような属性はドット表記式ではアクセスできず、代わりに `getattr()` を使って取る必要があります。

awaitable

(待機可能) `await` 式で使用することが出来るオブジェクトです。*coroutine* か、`__await__()` メソッドがあるオブジェクトです。**PEP 492** を参照してください。

BDFL 慈

悲深き終身独裁者 (Benevolent Dictator For Life) の略です。Python の作者、Guido van Rossum のことです。

binary file

(バイナリファイル) *bytes-like* **オブジェクト** の読み込みおよび書き込みができる **ファイルオブジェクト** です。バイナリファイルの例は、バイナリモード (`'rb'`, `'wb'` or `'rb+'`) で開かれたファイル、`sys.stdin.buffer`、`sys.stdout.buffer`、`io.BytesIO` や `gzip.GzipFile` のインスタンスです。

`str` オブジェクトの読み書きができるファイルオブジェクトについては、*text file* も参照してください。

borrowed reference In

Python's C API, a borrowed reference is a reference to an object, where the code using the object does not own the reference. It becomes a dangling pointer if the object is destroyed. For example, a garbage collection can remove the last *strong reference* to the object and so destroy it.

Calling `Py_INCREF()` on the *borrowed reference* is recommended to convert it to a *strong reference* in-place, except when the object cannot be destroyed before the last usage of the borrowed reference. The `Py_NewRef()` function can be used to create a new *strong reference*.

bytes-like object

bufferobjects をサポートしていて、C 言語の意味で **連続した** バッファーを提供可能なオブジェクト。`bytes`, `bytearray`, `array.array` や、多くの一般的な `memoryview` オブジェクトがこれに当たりま

す。bytes-like オブジェクトは、データ圧縮、バイナリファイルへの保存、ソケットを経由した送信など、バイナリデータを要求するいろいろな操作に利用することができます。

幾つかの操作ではバイナリデータを変更する必要があります。その操作のドキュメントではよく "読み書き可能な bytes-like オブジェクト" に言及しています。変更可能なバッファーオブジェクトには、`bytearray` と `bytearray` の `memoryview` などが含まれます。また、他の幾つかの操作では不変なオブジェクト内のバイナリデータ ("読み出し専用の bytes-like オブジェクト") を必要します。それには `bytes` と `bytes` の `memoryview` オブジェクトが含まれます。

bytecode

(バイトコード) Python のソースコードは、Python プログラムの CPython インタプリタの内部表現であるバイトコードへとコンパイルされます。バイトコードは `.pyc` ファイルにキャッシュされ、同じファイルが二度目に実行されるときはより高速になります (ソースコードからバイトコードへの再度のコンパイルは回避されます)。この "中間言語 (intermediate language)" は、各々のバイトコードに対応する機械語を実行する **仮想マシン** で動作するといえます。重要な注意として、バイトコードは異なる Python 仮想マシン間で動作することや、Python リリース間で安定であることは期待されていません。

バイトコードの命令一覧は dis モジュール にあります。

callable A

callable is an object that can be called, possibly with a set of arguments (see *argument*), with the following syntax:

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

A *function*, and by extension a *method*, is a callable. An instance of a class that implements the `__call__()` method is also a callable.

callback

(コールバック) 将来のある時点で実行されるために引数として渡される関数

クラス

(クラス) ユーザー定義オブジェクトを作成するためのテンプレートです。クラス定義は普通、そのクラスのインスタンス上の操作をするメソッドの定義を含みます。

class variable

(クラス変数) クラス上に定義され、クラスレベルで (つまり、クラスのインスタンス上ではなしに) 変更されることを目的としている変数です。

closure variable A

free variable referenced from a *nested scope* that is defined in an outer scope rather than being resolved at runtime from the globals or builtin namespaces. May be explicitly defined with the `nonlocal` keyword to allow write access, or implicitly defined if the variable is only being read.

For example, in the `inner` function in the following code, both `x` and `print` are *free variables*, but only `x` is a *closure variable*:

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

Due to the `codeobject.co_freevars` attribute (which, despite its name, only includes the names of closure variables rather than listing all referenced free variables), the more general *free variable* term is sometimes used even when the intended meaning is to refer specifically to closure variables.

complex number

(複素数) よく知られている実数系を拡張したもので、すべての数は実部と虚部の和として表されます。虚数は虚数単位 (`-1` の平方根) に実数を掛けたもので、一般に数学では `i` と書かれ、工学では `j` と書かれます。Python は複素数に組み込みで対応し、後者の表記を取っています。虚部は末尾に `j` をつけて書きます。例えば `3+1j` です。`math` モジュールの複素数版を利用するには、`cmath` を使います。複素数の使用はかなり高度な数学の機能です。必要性を感じなければ、ほぼ間違いなく無視してしまってよいでしょう。

context

This term has different meanings depending on where and how it is used. Some common meanings:

- The temporary state or environment established by a *context manager* via a `with` statement.
- The collection of keyvalue bindings associated with a particular `contextvars.Context` object and accessed via `ContextVar` objects. Also see *context variable*.
- A `contextvars.Context` object. Also see *current context*.

コンテキスト管理プロトコル

The `__enter__()` and `__exit__()` methods called by the `with` statement. See **PEP 343**.

context manager

An object which implements the *context management protocol* and controls the environment seen in a `with` statement. See **PEP 343**.

context variable

A variable whose value depends on which context is the *current context*. Values are accessed via `contextvars.ContextVar` objects. Context variables are primarily used to isolate state between concurrent asynchronous tasks.

contiguous

(隣接、連続) バッファが厳密に *C*-**連続** または *Fortran* **連続** である場合に、そのバッファは連続しているとみなせます。ゼロ次元バッファは C 連続であり Fortran 連続です。一次元の配列では、その要素は必ずメモリ上で隣接するように配置され、添字がゼロから始まり増えていく順序で並びます。多次

元の C-連続な配列では、メモリアドレス順に要素を巡る際には最後の添え字が最初に変わるのに対し、Fortran 連続な配列では最初の添え字が最初に動きます。

コルーチン

(コルーチン) コルーチンはサブルーチンのより一般的な形式です。サブルーチンには決められた地点から入り、別の決められた地点から出ます。コルーチンには多くの様々な地点から入る、出る、再開することができます。コルーチンは `async def` 文で実装できます。**PEP 492** を参照してください。

coroutine function

(コルーチン関数) *coroutine* オブジェクトを返す関数です。コルーチン関数は `async def` 文で実装され、`await`、`async for`、および `async with` キーワードを持つことが出来ます。これらは **PEP 492** で導入されました。

CPython

python.org で配布されている、Python プログラミング言語の標準的な実装です。"CPython" という単語は、この実装を Jython や IronPython といった他の実装と区別する必要が有る場合に利用されます。

current context

The *context* (`contextvars.Context` object) that is currently used by `ContextVar` objects to access (get or set) the values of *context variables*. Each thread has its own current context. Frameworks for executing asynchronous tasks (see `asyncio`) associate each task with a context which becomes the current context whenever the task starts or resumes execution.

decorator

(デコレータ) 別の関数を返す関数で、通常、`@wrapper` 構文で関数変換として適用されます。デコレータの一般的な利用例は、`classmethod()` と `staticmethod()` です。

デコレータの文法はシンタックスシュガーです。次の 2 つの関数定義は意味的に同じものです:

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

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

同じ概念がクラスにも存在しますが、あまり使われません。デコレータについて詳しくは、関数定義 および クラス定義 のドキュメントを参照してください。

descriptor

Any object which defines the methods `__get__()`, `__set__()`, or `__delete__()`. When a class attribute is a descriptor, its special binding behavior is triggered upon attribute lookup. Normally, using *a.b* to get, set or delete an attribute looks up the object named *b* in the class dictionary for *a*, but if *b* is a descriptor, the respective descriptor method gets called. Understanding descriptors is a key to a deep understanding of Python because they are the basis for many features including

functions, methods, properties, class methods, static methods, and reference to super classes.

デスクリプタのメソッドに関しての詳細は、descriptors や Descriptor How To Guide を参照してください。

dictionary

An associative array, where arbitrary keys are mapped to values. The keys can be any object with `__hash__()` and `__eq__()` methods. Called a hash in Perl.

dictionary comprehension

(辞書内包表記) iterable 内の全てあるいは一部の要素を処理して、その結果からなる辞書を返すコンパクトな書き方です。`results = {n: n ** 2 for n in range(10)}` とすると、キー `n` を値 `n ** 2` に対応付ける辞書を生成します。comprehensions を参照してください。

dictionary view

(辞書ビュー) `dict.keys()`、`dict.values()`、`dict.items()` が返すオブジェクトです。辞書の項目の動的なビューを提供します。すなわち、辞書が変更されるとビューはそれを反映します。辞書ビューを強制的に完全なリストにするには `list(dictview)` を使用してください。dict-views を参照してください。

docstring A

string literal which appears as the first expression in a class, function or module. While ignored when the suite is executed, it is recognized by the compiler and put into the `__doc__` attribute of the enclosing class, function or module. Since it is available via introspection, it is the canonical place for documentation of the object.

duck-typing あ

るオブジェクトが正しいインターフェースを持っているかを決定するのにオブジェクトの型を見ないプログラミングスタイルです。代わりに、単純にオブジェクトのメソッドや属性が呼ばれたり使われたりします。(「アヒルのように見えて、アヒルのように鳴けば、それはアヒルである。」) インターフェースを型より重視することで、上手くデザインされたコードは、ポリモーフィックな代替を許して柔軟性を向上させます。ダックタイピングは `type()` や `isinstance()` による判定を避けます。(ただし、ダックタイピングを **抽象基底クラス** で補完することもできます。) その代わり、典型的に `hasattr()` 判定や *EAFP* プログラミングを利用します。

EAFP

「認可をとるより許しを請う方が容易 (easier to ask for forgiveness than permission、マーフィーの法則)」の略です。この Python で広く使われているコーディングスタイルでは、通常は有効なキーや属性が存在するものと仮定し、その仮定が誤っていた場合に例外を捕捉します。この簡潔で手早く書けるコーディングスタイルには、`try` 文および `except` 文がたくさんあるのが特徴です。このテクニックは、C のような言語でよく使われている *LBYL* スタイルと対照的なものです。

evaluate function A

function that can be called to evaluate a lazily evaluated attribute of an object, such as the value of type aliases created with the `type` statement.

expression

(式) 何かの値と評価される、一まとまりの構文 (a piece of syntax) です。言い換えると、式とはリテ

ラル、名前、属性アクセス、演算子や関数呼び出しなど、値を返す式の要素の積み重ねです。他の多くの言語と違い、Python では言語の全ての構成要素が式というわけではありません。`while` のように、式としては使えない **文** もあります。代入も式ではなく文です。

extension module

(拡張モジュール) C や C++ で書かれたモジュールで、Python の C API を利用して Python コアやユーザーコードとやりとりします。

f-string

`'f'` や `'F'` が先頭に付いた文字列リテラルは "f-string" と呼ばれ、これは フォーマット済み文字列リテラル の短縮形の名称です。**PEP 498** も参照してください。

file object

An object exposing a file-oriented API (with methods such as `read()` or `write()`) to an underlying resource. Depending on the way it was created, a file object can mediate access to a real on-disk file or to another type of storage or communication device (for example standard input/output, in-memory buffers, sockets, pipes, etc.). File objects are also called *file-like objects* or *streams*.

ファイルオブジェクトには実際には 3 種類あります: 生の **バイナリーファイル**、バッファされた **バイナリーファイル**、そして **テキストファイル** です。インターフェイスは `io` モジュールで定義されています。ファイルオブジェクトを作る標準的な方法は `open()` 関数を使うことです。

file-like object

file object と同義です。

filesystem encoding and error handler

Encoding and error handler used by Python to decode bytes from the operating system and encode Unicode to the operating system.

ファイルシステムのエンコーディングでは、すべてが 128 バイト以下に正常にデコードされることが保証されなくてはなりません。ファイルシステムのエンコーディングでこれが保証されなかった場合は、API 関数が `UnicodeError` を送出することがあります。

The `sys.getfilesystemencoding()` and `sys.getfilesystemencodeerrors()` functions can be used to get the filesystem encoding and error handler.

The *filesystem encoding and error handler* are configured at Python startup by the `PyConfig_Read()` function: see `filesystem_encoding` and `filesystem_errors` members of `PyConfig`.

See also the *locale encoding*.

finder

(ファインダ) インポートされているモジュールの *loader* の発見を試行するオブジェクトです。

There are two types of finder: *meta path finders* for use with `sys.meta_path`, and *path entry finders* for use with `sys.path_hooks`.

See finders-and-loaders and `importlib` for much more detail.

floor division

(切り捨て除算) 一番近い整数に切り捨てる数学的除算。切り捨て除算演算子は `//` です。例えば、`11 // 4` は 2 になり、それとは対称に浮動小数点数の真の除算では `2.75` が 返ってきます。`(-11) // 4` は `-2.75` を **小さい方に** 丸める (訳注: 負の無限大への丸めを行う) ので `-3` になることに注意してください。**PEP 238** を参照してください。

free threading

A threading model where multiple threads can run Python bytecode simultaneously within the same interpreter. This is in contrast to the *global interpreter lock* which allows only one thread to execute Python bytecode at a time. See **PEP 703**.

free variable

Formally, as defined in the language execution model, a free variable is any variable used in a namespace which is not a local variable in that namespace. See *closure variable* for an example. Pragmatically, due to the name of the `codeobject.co_freevars` attribute, the term is also sometimes used as a synonym for *closure variable*.

関数

(関数) 呼び出し側に値を返す一連の文のことです。関数には 0 以上の **実引数** を渡すことが出来ます。実体の実行時に引数を使用することが出来ます。**仮引数**、**メソッド**、function を参照してください。

function annotation

(関数アノテーション) 関数のパラメータや返り値の *annotation* です。

関数アノテーションは、通常は **型ヒント** のために使われます: 例えば、この関数は 2 つの `int` 型の引数を取ると期待され、また `int` 型の返り値を持つと期待されています。

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

関数アノテーションの文法は function の節で解説されています。

機能の説明がある *variable annotation*, **PEP 484**, を参照してください。また、アノテーションを利用するベストプラクティスとして annotations-howto も参照してください。

__future__

`from __future__ import <feature>` という future 文 は、コンパイラーに将来の Python リリースで標準となる構文や意味を使用して現在のモジュールをコンパイルするよう指示します。`__future__` モジュールでは、*feature* のとりうる値をドキュメント化しています。このモジュールをインポートし、その変数を評価することで、新機能が最初に言語に追加されたのはいつかや、いつデフォルトになるか (またはなったか) を見ることができます:

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

garbage collection

(ガベージコレクション) これ以降使われることのないメモリを解放する処理です。Python は、参照カ

ウントと、循環参照を検出し破壊する循環ガベージコレクタを使ってガベージコレクションを行います。ガベージコレクタは gc モジュールを使って操作できます。

ジェネレータ

(ジェネレータ) *generator iterator* を返す関数です。通常の関数に似ていますが、yield 式を持つ点で異なります。yield 式は、for ループで使用できたり、next() 関数で値を 1 つずつ取り出したりできる、値の並びを生成するのに使用されます。

通常はジェネレータ関数を指しますが、文脈によっては **ジェネレータイテレータ** を指す場合があります。意図された意味が明らかでない場合、明瞭化のために完全な単語を使用します。

generator iterator

(ジェネレータイテレータ) *generator* 関数で生成されるオブジェクトです。

yield のたびに局所実行状態 (局所変数や未処理の try 文などを含む) を記憶して、処理は一時的に中断されます。**ジェネレータイテレータ** が再開されると、中断した位置を取得します (通常の関数が実行のたびに新しい状態から開始するのと対照的です)。

generator expression

An *expression* that returns an *iterator*. It looks like a normal expression followed by a for clause defining a loop variable, range, and an optional if clause. The combined expression generates values for an enclosing function:

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

generic function

(ジェネリック関数) 異なる型に対し同じ操作をする関数群から構成される関数です。呼び出し時にどの実装を用いるかはディスパッチアルゴリズムにより決定されます。

single dispatch、functools.singledispatch() デコレータ、**PEP 443** を参照してください。

generic type A

type that can be parameterized; typically a container class such as list or dict. Used for *type hints* and *annotations*.

For more details, see generic alias types, **PEP 483**, **PEP 484**, **PEP 585**, and the typing module.

GIL

global interpreter lock を参照してください。

global interpreter lock

(グローバルインタプリタロック) *CPython* インタプリタが利用している、一度に Python の **バイトコード** を実行するスレッドは一つだけであることを保証する仕組みです。これにより (dict などの重要な組み込み型を含む) オブジェクトモデルが同時アクセスに対して暗黙的に安全になるので、CPython の実装がシンプルになります。インタプリタ全体をロックすることで、マルチプロセッサマシンが生じる並列化のコストと引き換えに、インタプリタを簡単にマルチスレッド化できるようになります。

ただし、標準あるいは外部のいくつかの拡張モジュールは、圧縮やハッシュ計算などの計算の重い処理をするときに GIL を解除するように設計されています。また、I/O 処理をする場合 GIL は常に解除されます。

As of Python 3.13, the GIL can be disabled using the `--disable-gil` build configuration. After building Python with this option, code must be run with `-X gil=0` or after setting the `PYTHON_GIL=0` environment variable. This feature enables improved performance for multi-threaded applications and makes it easier to use multi-core CPUs efficiently. For more details, see **PEP 703**.

hash-based pyc

(ハッシュベース pyc ファイル) 正当性を判別するために、対応するソースファイルの最終更新時刻ではなくハッシュ値を使用するバイトコードのキャッシュファイルです。pyc-invalidation を参照してください。

hashable

An object is *hashable* if it has a hash value which never changes during its lifetime (it needs a `__hash__()` method), and can be compared to other objects (it needs an `__eq__()` method). Hashable objects which compare equal must have the same hash value.

ハッシュ可能なオブジェクトは辞書のキーや集合のメンバーとして使えます。辞書や集合のデータ構造は内部でハッシュ値を使っているからです。

Python のイミュータブルな組み込みオブジェクトは、ほとんどがハッシュ可能です。(リストや辞書のような) ミュータブルなコンテナはハッシュ不可能です。(タプルや frozenset のような) イミュータブルなコンテナは、要素がハッシュ可能であるときのみハッシュ可能です。ユーザー定義のクラスのインスタンスであるようなオブジェクトはデフォルトでハッシュ可能です。それらは全て (自身を除いて) 比較結果は非等価であり、ハッシュ値は `id()` より得られます。

IDLE

Python の統合開発環境 (Integrated DeveLopment Environment) 及び学習環境 (Learning Environment) です。idle は Python の標準的な配布に同梱されている基本的な機能のエディタとインタプリタ環境です。

永続オブジェクト (immortal)

Immortal objects are a CPython implementation detail introduced in **PEP 683**.

If an object is immortal, its *reference count* is never modified, and therefore it is never deallocated while the interpreter is running. For example, `True` and `None` are immortal in CPython.

immutable

(イミュータブル) 固定の値を持ったオブジェクトです。イミュータブルなオブジェクトには、数値、文字列、およびタプルなどがあります。これらのオブジェクトは値を変えられません。別の値を記憶させる際には、新たなオブジェクトを作成しなければなりません。イミュータブルなオブジェクトは、固定のハッシュ値が必要となる状況で重要な役割を果たします。辞書のキーがその例です。

import path

path based finder が import するモジュールを検索する場所 (または *path entry*) のリスト。import

中、このリストは通常 `sys.path` から来ますが、サブパッケージの場合は親パッケージの `__path__` 属性からも来ます。

importing
あるモジュールの Python コードが別のモジュールの Python コードで使えるようにする処理です。

importer
モジュールを探してロードするオブジェクト。*finder* と *loader* のどちらでもあるオブジェクト。

interactive
Python has an interactive interpreter which means you can enter statements and expressions at the interpreter prompt, immediately execute them and see their results. Just launch `python` with no arguments (possibly by selecting it from your computer's main menu). It is a very powerful way to test out new ideas or inspect modules and packages (remember `help(x)`). For more on interactive mode, see tut-interac.

interpreted
Python はインタプリタ形式の言語であり、コンパイラ言語の対極に位置します。(バイトコードコンパイラがあるために、この区別は曖昧ですが。) ここでのインタプリタ言語とは、ソースコードのファイルを、まず実行可能形式にしてから実行させるといった操作なしに、直接実行できることを意味します。インタプリタ形式の言語は通常、コンパイラ形式の言語よりも開発／デバッグのサイクルは短いものの、プログラムの実行は一般に遅いです。**対話的** も参照してください。

interpreter shutdown
Python インタープリターはシャットダウンを要請された時に、モジュールやすべてのクリティカルな内部構造をなどの、すべての確保したリソースを段階的に開放する、特別なフェーズに入ります。このフェーズは **ガベージコレクタ** を複数回呼び出します。これによりユーザー定義のデストラクターや weakref コールバックが呼び出されることがあります。シャットダウンフェーズ中に実行されるコードは、それが依存するリソースがすでに機能しない (よくある例はライブラリーモジュールや warning 機構です) ために様々な例外に直面します。

インタープリタがシャットダウンする主な理由は `__main__` モジュールや実行されていたスクリプトの実行が終了したことです。

iterable
An object capable of returning its members one at a time. Examples of iterables include all sequence types (such as `list`, `str`, and `tuple`) and some non-sequence types like `dict`, *file objects*, and objects of any classes you define with an `__iter__()` method or with a `__getitem__()` method that implements *sequence* semantics.

Iterables can be used in a `for` loop and in many other places where a sequence is needed (`zip()`, `map()`, ...). When an iterable object is passed as an argument to the built-in function `iter()`, it returns an iterator for the object. This iterator is good for one pass over the set of values. When using iterables, it is usually not necessary to call `iter()` or deal with iterator objects yourself. The `for` statement does that automatically for you, creating a temporary unnamed variable to hold the iterator for the duration of the loop. See also *iterator*, *sequence*, and *generator*.

iterator

データの流れを表現するオブジェクトです。イテレータの `__next__()` メソッドを繰り返し呼び出す (または組み込み関数 `next()` に渡す) と、流れの中の要素を一つずつ返します。データがなくなると、代わりに `StopIteration` 例外を送出します。その時点で、イテレータオブジェクトは尽きており、それ以降は `__next__()` を何度呼んでも `StopIteration` を送出します。イテレータは、そのイテレータオブジェクト自体を返す `__iter__()` メソッドを実装しなければならないので、イテレータは他の iterable を受理するほとんどの場所で利用できます。はっきりとした例外は複数の反復を行うようなコードです。(`list` のような) コンテナオブジェクトは、自身を `iter()` 関数にオブジェクトに渡したり `for` ループ内で使うたびに、新たな未使用のイテレータを生成します。これをイテレータで行おうとすると、前回のイテレーションで使用済みの同じイテレータオブジェクトを単純に返すため、空のコンテナのようになってしまします。

詳細な情報は typeiter にあります。

CPython 実装の詳細: CPython does not consistently apply the requirement that an iterator define `__iter__()`. And also please note that the free-threading CPython does not guarantee the thread-safety of iterator operations.

key function

(キー関数) キー関数、あるいは照合関数とは、ソートや順序比較のための値を返す呼び出し可能オブジェクト (callable) です。例えば、`locale.strxfrm()` をキー関数に使えば、ロケール依存のソートの慣習にのっとったソートキーを返します。

Python の多くのツールはキー関数を受け取り要素の並び順やグループ化を管理します。`min()`, `max()`, `sorted()`, `list.sort()`, `heapq.merge()`, `heapq.nsmallest()`, `heapq.nlargest()`, `itertools.groupby()` 等があります。

キー関数を作る方法はいくつかあります。例えば `str.lower()` メソッドを大文字小文字を区別しないソートを行うキー関数として使うことが出来ます。あるいは、`lambda r: (r[0], r[2])` のような `lambda` 式からキー関数を作ることができます。また、`operator.attrgetter()`, `operator.itemgetter()`, `operator.methodcaller()` の 3 つのキー関数コンストラクタがあります。キー関数の作り方と使い方の例は Sorting HOW TO を参照してください。

keyword argument 実

引数 を参照してください。

lambda

(ラムダ) 無名のインライン関数で、関数が呼び出されたときに評価される 1 つの 式 を含みます。ラムダ関数を作る構文は `lambda [parameters]: expression` です。

LBYL

「ころばぬ先の杖 (look before you leap)」の略です。このコーディングスタイルでは、呼び出しや検索を行う前に、明示的に前提条件 (pre-condition) 判定を行います。*EAFP* アプローチと対照的で、`if` 文がたくさん使われるのが特徴的です。

マルチスレッド化された環境では、LBYL アプローチは ”見る” 過程と ”飛ぶ” 過程の競合状態を引き起こすリスクがあります。例えば、`if key in mapping: return mapping[key]` というコードは、判定の後、別のスレッドが探索の前に *mapping* から *key* を取り除くと失敗します。この問題は、ロックするか EAFP アプローチを使うことで解決できます。

list

A built-in Python *sequence*. Despite its name it is more akin to an array in other languages than to a linked list since access to elements is $O(1)$.

list comprehension

(リスト内包表記) シーケンス中の全てあるいは一部の要素を処理して、その結果からなるリストを返す、コンパクトな方法です。`result = ['{:#04x}'.format(x) for x in range(256) if x % 2 == 0]` とすると、0 から 255 までの偶数を 16 進数表記 (0x..) した文字列からなるリストを生成します。`if` 節はオプションです。`if` 節がない場合、`range(256)` の全ての要素が処理されます。

loader

An object that loads a module. It must define a method named `load_module()`. A loader is typically returned by a *finder*. See also:

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

ロケールエンコーディング

On Unix, it is the encoding of the LC_CTYPE locale. It can be set with `locale.setlocale(locale.LC_CTYPE, new_locale)`.

On Windows, it is the ANSI code page (ex: `"cp1252"`).

On Android and VxWorks, Python uses `"utf-8"` as the locale encoding.

`locale.getencoding()` can be used to get the locale encoding.

See also the *filesystem encoding and error handler*.

magic method

special method のくだけた同義語です。

mapping

(マッピング) 任意のキー探索をサポートしていて、`collections.abc.Mapping` か `collections.abc.MutableMapping` の 抽象基底クラス で指定されたメソッドを実装しているコンテナオブジェクトです。例えば、`dict`, `collections.defaultdict`, `collections.OrderedDict`, `collections.Counter` などです。

meta path finder

`sys.meta_path` を検索して得られた *finder*. meta path finder は *path entry finder* と関係はありますが、別物です。

meta path finder が実装するメソッドについては `importlib.abc.MetaPathFinder` を参照してください。

metaclass

(メタクラス) クラスのクラスです。クラス定義は、クラス名、クラスの辞書と、基底クラスのリストを作ります。メタクラスは、それら 3 つを引数として受け取り、クラスを作る責任を負います。ほとんど

のオブジェクト指向言語は (訳注: メタクラスの) デフォルトの実装を提供しています。Python が特別なのはカスタムのメタクラスを作成できる点です。ほとんどのユーザーに取って、メタクラスは全く必要のないものです。しかし、一部の場面では、メタクラスは強力でエレガントな方法を提供します。たとえば属性アクセスのログを取ったり、スレッドセーフ性を追加したり、オブジェクトの生成を追跡したり、シングルトンを実装するなど、多くの場面で利用されます。

詳細は metaclasses を参照してください。

メソッド

(メソッド) クラス本体の中で定義された関数。そのクラスのインスタンスの属性として呼び出された場合、メソッドはインスタンスオブジェクトを第一 **引数** として受け取ります (この第一引数は通常 `self` と呼ばれます)。**関数** と **ネストされたスコープ** も参照してください。

method resolution order

Method Resolution Order is the order in which base classes are searched for a member during lookup. See python_2.3_mro for details of the algorithm used by the Python interpreter since the 2.3 release.

module

(モジュール) Python コードの組織単位としてはたらくオブジェクトです。モジュールは任意の Python オブジェクトを含む名前空間を持ちます。モジュールは *importing* の処理によって Python に読み込まれます。

パッケージ を参照してください。

module spec

モジュールをロードするのに使われるインポート関連の情報を含む名前空間です。`importlib.machinery.ModuleSpec` のインスタンスです。

See also module-specs.

MRO

method resolution order を参照してください。

mutable

(ミュータブル) ミュータブルなオブジェクトは、`id()` を変えることなく値を変更できます。**イミュータブル**) も参照してください。

named tuple

”名前付きタプル” という用語は、タプルを継承していて、インデックスが付く要素に対し属性を使ってのアクセスもできる任意の型やクラスに応用されています。その型やクラスは他の機能も持っていることもあります。

`time.localtime()` や `os.stat()` の返り値を含むいくつかの組み込み型は名前付きタプルです。他の例は `sys.float_info` です:

```
>>> sys.float_info[1]                    # indexed access
1024
>>> sys.float_info.max_exp               # named field access
```

(次のページに続く)

(前のページからの続き)

```
1024
>>> isinstance(sys.float_info, tuple)   # kind of tuple
True
```

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

namespace

(名前空間) 変数が格納される場所です。名前空間は辞書として実装されます。名前空間にはオブジェクトの (メソッドの) 入れ子になったものだけでなく、局所的なもの、大域的なもの、そして組み込みのものがあります。名前空間は名前の衝突を防ぐことによってモジュール性をサポートする。例えば関数 `builtins.open` と `os.open()` は名前空間で区別されています。また、どのモジュールが関数を実装しているか明示することによって名前空間は可読性と保守性を支援します。例えば、`random.seed()` や `itertools.islice()` と書くと、それぞれモジュール `random` や `itertools` で実装されていることが明らかです。

namespace package

(名前空間パッケージ) サブパッケージのコンテナとしてのみ提供される **PEP 420** で定義された *package* です。名前空間パッケージは物理的な表現を持たないことができ、`__init__.py` ファイルを持たないため、*regular package* とは異なります。

module を参照してください。

nested scope

(ネストされたスコープ) 外側で定義されている変数を参照する機能です。例えば、ある関数が別の関数の中で定義されている場合、内側の関数は外側の関数中の変数を参照できます。ネストされたスコープはデフォルトでは変数の参照だけができ、変数の代入はできないので注意してください。ローカル変数は、最も内側のスコープで変数を読み書きします。同様に、グローバル変数を使うとグローバル名前空間の値を読み書きします。`nonlocal` で外側の変数に書き込めます。

new-style class

Old name for the flavor of classes now used for all class objects. In earlier Python versions, only new-style classes could use Python's newer, versatile features like `__slots__`, descriptors, properties, `__getattribute__()`, class methods, and static methods.

object

(オブジェクト) 状態 (属性や値) と定義された振る舞い (メソッド) をもつ全てのデータ。もしくは、全ての **新スタイルクラス** の究極の基底クラスのこと。

optimized scope

A scope where target local variable names are reliably known to the compiler when the code is compiled, allowing optimization of read and write access to these names. The local namespaces for functions, generators, coroutines, comprehensions, and generator expressions are optimized in

this fashion. Note: most interpreter optimizations are applied to all scopes, only those relying on a known set of local and nonlocal variable names are restricted to optimized scopes.

package

(パッケージ) サブモジュールや再帰的にサブパッケージを含むことの出来る *module* のことです。専門的には、パッケージは `__path__` 属性を持つ Python オブジェクトです。

regular package と *namespace package* を参照してください。

parameter

(仮引数) 名前付の実体で **関数** (や **メソッド**) の定義において関数が受ける **実引数** を指定します。仮引数には 5 種類あります:

- **位置またはキーワード**: **位置** であるいは **キーワード引数** として渡すことができる引数を指定します。これはたとえば以下の *foo* や *bar* のように、デフォルトの仮引数の種類です:

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

- 位置専用: 位置によってのみ与えられる引数を指定します。位置専用の引数は 関数定義の引数のリストの中でそれらの後ろに `/` を含めることで定義できます。例えば下記の *posonly1* と *posonly2* は位置専用引数になります:

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

- **キーワード専用**: キーワードによってのみ与えられる引数を指定します。キーワード専用の引数を定義できる場所は、例えば以下の *kw_only1* や *kw_only2* のように、関数定義の仮引数リストに含めた可変長位置引数または裸の `*` の後です:

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

- **可変長位置**: (他の仮引数で既に受けられた任意の位置引数に加えて) 任意の個数の位置引数が与えられることを指定します。このような仮引数は、以下の *args* のように仮引数名の前に `*` をつけることで定義できます:

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

- **可変長キーワード**: (他の仮引数で既に受けられた任意のキーワード引数に加えて) 任意の個数のキーワード引数が与えられることを指定します。このような仮引数は、上の例の *kwargs* のように仮引数名の前に `**` をつけることで定義できます。

仮引数はオプションと必須の引数のどちらも指定でき、オプションの引数にはデフォルト値も指定できます。

仮引数 、FAQ の 実引数と仮引数の違いは何ですか? 、`inspect.Parameter` クラス、function セクション、**PEP 362** を参照してください。

path entry

path based finder が import するモジュールを探す *import path* 上の 1 つの場所です。

path entry finder

sys.path_hooks にある callable (つまり *path entry hook*) が返した *finder* です。与えられた *path entry* にあるモジュールを見つける方法を知っています。

パスエントリーファインダが実装するメソッドについては importlib.abc.PathEntryFinder を参照してください。

path entry hook

A callable on the sys.path_hooks list which returns a *path entry finder* if it knows how to find modules on a specific *path entry*.

path based finder

デフォルトの *meta path finder* の 1 つは、モジュールの *import path* を検索します。

path-like object

(path-like オブジェクト) ファイルシステムパスを表します。path-like オブジェクトは、パスを表す str オブジェクトや bytes オブジェクト、または os.PathLike プロトコルを実装したオブジェクトのどれかです。os.PathLike プロトコルをサポートしているオブジェクトは os.fspath() を呼び出すことで str または bytes のファイルシステムパスに変換できます。os.fsdecode() と os.fsencode() はそれぞれ str あるいは bytes になるのを保証するのに使えます。**PEP 519** で導入されました。

PEP

Python Enhancement Proposal。PEP は、Python コミュニティに対して情報を提供する、あるいは Python の新機能やその過程や環境について記述する設計文書です。PEP は、機能についての簡潔な技術的仕様と提案する機能の論拠 (理論) を伝えるべきです。

PEP は、新機能の提案にかかる、コミュニティによる問題提起の集積と Python になされる設計決断の文書化のための最上位の機構となることを意図しています。PEP の著者にはコミュニティ内の合意形成を行うこと、反対意見を文書化することの責務があります。

PEP 1 を参照してください。

portion

PEP 420 で定義されている、namespace package に属する、複数のファイルが (zip ファイルに格納されている場合もある) 1 つのディレクトリに格納されたもの。

位置引数 (positional argument)

実引数 を参照してください。

provisional API

(暫定 API) 標準ライブラリの後方互換性保証から計画的に除外されたものです。そのようなインターフェースへの大きな変更は、暫定であるとされている間は期待されていませんが、コア開発者によって必要とみなされれば、後方非互換な変更 (インターフェースの削除まで含まれる) が行われえます。このような変更はむやみに行われるものではありません -- これは API を組み込む前には見落とされていた重大な欠陥が露呈したときにのみ行われます。

暫定 API についても、後方互換性のない変更は「最終手段」とみなされています。問題点が判明した場合でも後方互換な解決策を探すべきです。

このプロセスにより、標準ライブラリは問題となるデザインエラーに長い間閉じ込められることなく、時代を超えて進化を続けられます。詳細は **PEP 411** を参照してください。

provisional package

provisional API を参照してください。

Python 3000

Python 3.x リリースラインのニックネームです。(Python 3 が遠い将来の話だった頃に作られた言葉です。) "Py3k" と略されることもあります。

Pythonic

他の言語で一般的な考え方で書かれたコードではなく、Python の特に一般的なイディオムに従った考え方やコード片。例えば、Python の一般的なイディオムでは `for` 文を使ってイテラブルのすべての要素に渡ってループします。他の多くの言語にはこの仕組みはないので、Python に慣れていない人は代わりに数値のカウンターを使うかもしれません:

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

これに対し、きれいな Pythonic な方法は:

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

qualified name

(修飾名) モジュールのグローバルスコープから、そのモジュールで定義されたクラス、関数、メソッドへの、" パス" を表すドット名表記です。**PEP 3155** で定義されています。トップレベルの関数やクラスでは、修飾名はオブジェクトの名前と同じです:

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

モジュールへの参照で使われると、**完全修飾名** *(fully qualified name)* はすべての親パッケージを含む全体のドット名表記、例えば `email.mime.text` を意味します:

```
>>> import email.mime.text
```

(次のページに続く)

(前のページからの続き)

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

reference count

(参照カウント) あるオブジェクトに対する参照の数。参照カウントが 0 になったとき、そのオブジェクトは破棄されます。**永続** であり、参照カウントが決して変更されないために割り当てが解除されないオブジェクトもあります。参照カウントは通常は Python のコード上には現れませんが、*CPython* 実装の重要な要素です。プログラマーは、任意のオブジェクトの参照カウントを知るために `sys.getrefcount()` 関数を呼び出すことが出来ます。

regular package 伝

統的な、`__init__.py` ファイルを含むディレクトリとしての *package*。

namespace package を参照してください。

REPL

"read – eval – print loop" の頭字語で、**対話型** インタープリターシェルの別名。

__slots__ ク

ラス内での宣言で、インスタンス属性の領域をあらかじめ定義しておき、インスタンス辞書を排除することで、メモリを節約します。これはよく使われるテクニックですが、正しく扱うには少しトリッキーなので、稀なケース、例えばメモリが死活問題となるアプリケーションでインスタンスが大量に存在する、といったときを除き、使わないのがベストです。

sequence

An *iterable* which supports efficient element access using integer indices via the `__getitem__()` special method and defines a `__len__()` method that returns the length of the sequence. Some built-in sequence types are `list`, `str`, `tuple`, and `bytes`. Note that `dict` also supports `__getitem__()` and `__len__()`, but is considered a mapping rather than a sequence because the lookups use arbitrary *hashable* keys rather than integers.

The `collections.abc.Sequence` abstract base class defines a much richer interface that goes beyond just `__getitem__()` and `__len__()`, adding `count()`, `index()`, `__contains__()`, and `__reversed__()`. Types that implement this expanded interface can be registered explicitly using `register()`. For more documentation on sequence methods generally, see Common Sequence Operations.

set comprehension

(集合内包表記) iterable 内の全てあるいは一部の要素を処理して、その結果からなる集合を返すコンパクトな書き方です。`results = {c for c in 'abracadabra' if c not in 'abc'}` とすると、`{'r', 'd'}` という文字列の辞書を生成します。comprehensions を参照してください。

single dispatch

generic function の一種で実装は一つの引数の型により選択されます。

slice

(スライス) 一般に **シーケンス** の一部を含むオブジェクト。スライスは、添字表記 `[]` で与えられた複

数の数の間にコロンを書くことで作られます。例えば、`variable_name[1:3:5]` です。角括弧 (添字) 記号は `slice` オブジェクトを内部で利用しています。

soft deprecated A

soft deprecated API should not be used in new code, but it is safe for already existing code to use it. The API remains documented and tested, but will not be enhanced further.

Soft deprecation, unlike normal deprecation, does not plan on removing the API and will not emit warnings.

See PEP 387: Soft Deprecation.

special method

(特殊メソッド) ある型に特定の操作、例えば加算をするために Python から暗黙に呼び出されるメソッド。この種類のメソッドは、メソッド名の最初と最後にアンダースコア 2 つがついています。特殊メソッドについては specialnames で解説されています。

statement

(文) 文はスイート (コードの" ブロック") に不可欠な要素です。文は **式** かキーワードから構成されるもののどちらかです。後者には `if`、`while`、`for` があります。

static type checker

An external tool that reads Python code and analyzes it, looking for issues such as incorrect types. See also *type hints* and the `typing` module.

strong reference In

Python's C API, a strong reference is a reference to an object which is owned by the code holding the reference. The strong reference is taken by calling `Py_INCREF()` when the reference is created and released with `Py_DECREF()` when the reference is deleted.

The `Py_NewRef()` function can be used to create a strong reference to an object. Usually, the `Py_DECREF()` function must be called on the strong reference before exiting the scope of the strong reference, to avoid leaking one reference.

See also *borrowed reference*.

text encoding A

string in Python is a sequence of Unicode code points (in range `U+0000--U+10FFFF`). To store or transfer a string, it needs to be serialized as a sequence of bytes.

Serializing a string into a sequence of bytes is known as "encoding", and recreating the string from the sequence of bytes is known as "decoding".

There are a variety of different text serialization codecs, which are collectively referred to as "text encodings".

text file

(テキストファイル) `str` オブジェクトを読み書きできる *file object* です。しばしば、テキストファイルは実際にバイト指向のデータストリームにアクセスし、**テキストエンコーディング** を自動的に行い

ます。テキストファイルの例は、sys.stdin, sys.stdout, io.StringIO インスタンスなどをテキストモード ('r' or 'w') で開いたファイルです。

bytes-like **オブジェクト** を読み書きできるファイルオブジェクトについては、**バイナリファイル** も参照してください。

triple-quoted string

(三重クォート文字列) 3 つの連続したクォート記号 (") かアポストロフィー (') で囲まれた文字列。通常の (一重) クォート文字列に比べて表現できる文字列に違いはありませんが、幾つかの理由で有用です。1 つか 2 つの連続したクォート記号をエスケープ無しに書くことができますし、行継続文字 (\) を使わなくても複数行にまたがることができるので、ドキュメンテーション文字列を書く時に特に便利です。

type

The type of a Python object determines what kind of object it is; every object has a type. An object's type is accessible as its `__class__` attribute or can be retrieved with `type(obj)`.

type alias

(型エイリアス) 型の別名で、型を識別子に代入して作成します。

型エイリアスは **型ヒント** を単純化するのに有用です。例えば:

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

これは次のようにより読みやすくできます:

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

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

機能の説明がある typing と **PEP 484** を参照してください。

type hint

(型ヒント) 変数、クラス属性、関数のパラメータや返り値の期待される型を指定する *annotation* です。

Type hints are optional and are not enforced by Python but they are useful to *static type checkers*. They can also aid IDEs with code completion and refactoring.

グローバル変数、クラス属性、関数で、ローカル変数でないものの型ヒントは typing.get_type_hints() で取得できます。

機能の説明がある typing と **PEP 484** を参照してください。

universal newlines

テキストストリームの解釈法の一つで、以下のすべてを行末と認識します: Unix の行末規定 '\n'、

Windows の規定 `'\r\n'`、古い Macintosh の規定 `'\r'`。利用法について詳しくは、**PEP 278** と **PEP 3116** 、さらに `bytes.splitlines()` も参照してください。

variable annotation

(変数アノテーション) 変数あるいはクラス属性の *annotation* 。

変数あるいはクラス属性に注釈を付けたときは、代入部分は任意です:

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

変数アノテーションは通常は **型ヒント** のために使われます: 例えば、この変数は `int` の値を取ることを期待されています:

```
count: int = 0
```

変数アノテーションの構文については annassign 節で解説しています。

機能の説明がある *function annotation*, **PEP 484** , **PEP 526** を参照してください。また、アノテーションを利用するベストプラクティスとして annotations-howto も参照してください。

virtual environment

(仮想環境) 協調的に切り離された実行環境です。これにより Python ユーザとアプリケーションは同じシステム上で動いている他の Python アプリケーションの挙動に干渉することなく Python パッケージのインストールと更新を行うことができます。

`venv` を参照してください。

virtual machine

(仮想マシン) 完全にソフトウェアにより定義されたコンピュータ。Python の仮想マシンは、バイトコードコンパイラが出力した **バイトコード** を実行します。

Zen of Python

(Python の悟り) Python を理解し利用する上での導きとなる、Python の設計原則と哲学をリストにしたものです。対話プロンプトで "`import this`" とするとこのリストを読めます。

付録

B

このドキュメントについて

このドキュメントは、Python のドキュメントを主要な目的として作られた ドキュメントプロセッサの Sphinx を利用して、reStructuredText 形式のソースから生成されました。

ドキュメントとそのツール群の開発は、Python 自身と同様に完全にボランティアの努力です。もしあなたが貢献したいなら、どのようにすればよいかについて reporting-bugs ページをご覧下さい。新しいボランティアはいつでも歓迎です!（訳注: 日本語訳の問題については、GitHub 上の Issue Tracker で報告をお願いします。）

多大な感謝を:

- Fred L. Drake, Jr., オリジナルの Python ドキュメントツールセットの作成者で、ドキュメントの多くを書きました。
- Docutils プロジェクト reStructuredText と Docutils ツールセットを作成しました。
- Fredrik Lundh の Alternative Python Reference プロジェクトから Sphinx は多くのアイデアを得ました。

B.1 Python ドキュメント 貢献者

多くの方々が Python 言語、Python 標準ライブラリ、そして Python ドキュメンテーションに貢献してくれています。ソース配布物の Misc/ACKS に、それら貢献してくれた人々を部分的にではありますがリストアップしてあります。

Python コミュニティからの情報提供と貢献がなければこの素晴らしいドキュメンテーションは生まれませんでした -- ありがとう!

付録

C

歴史とライセンス

C.1 Python の歴史

Python は 1990 年代の始め、オランダにある Stichting Mathematisch Centrum (CWI, https://www.cwi.nl/ 参照) で Guido van Rossum によって ABC と呼ばれる言語の後継言語として生み出されました。その後多くの人々が Python に貢献していますが、Guido は今日でも Python 製作者の先頭に立っています。

1995 年、Guido は米国ヴァージニア州レストンにある Corporation for National Reserch Initiatives (CNRI, https://www.cnri.reston.va.us/ 参照) で Python の開発に携わり、いくつかのバージョンをリリースしました。

2000 年 3 月、Guido と Python のコア開発チームは BeOpen.com に移り、BeOpen PythonLabs チームを結成しました。同年 10 月、PythonLabs チームは Digital Creations (現在の Zope Corporation, https://www.zope.org/ 参照) に移りました。そして 2001 年、Python に関する知的財産を保有するための非営利組織 Python Software Foundation (PSF、https://www.python.org/psf/ 参照) を立ち上げました。このとき Zope Corporation は PSF の賛助会員になりました。

Python のリリースは全てオープンソース (オープンソースの定義は https://opensource.org/ を参照してください) です。歴史的にみて、ごく一部を除くほとんどの Python リリースは GPL 互換になっています; 各リリースについては下表にまとめてあります。

リリース	ベース	西暦年	権利	GPL 互換
0.9.0 - 1.2	n/a	1991-1995	CWI	yes
1.3 - 1.5.2	1.2	1995-1999	CNRI	yes
1.6	1.5.2	2000	CNRI	no
2.0	1.6	2000	BeOpen.com	no
1.6.1	1.6	2001	CNRI	no
2.1	2.0+1.6.1	2001	PSF	no
2.0.1	2.0+1.6.1	2001	PSF	yes
2.1.1	2.1+2.0.1	2001	PSF	yes
2.1.2	2.1.1	2002	PSF	yes
2.1.3	2.1.2	2002	PSF	yes
2.2 以降	2.1.1	2001-現在	PSF	yes

注釈

「GPL 互換」という表現は、Python が GPL で配布されているという意味ではありません。Python のライセンスは全て、GPL と違い、変更したバージョンを配布する際に変更をオープンソースにしなくてもかまいません。GPL 互換のライセンスの下では、GPL でリリースされている他のソフトウェアと Python を組み合わせられますが、それ以外のライセンスではそうではありません。

Guido の指示の下、これらのリリースを可能にしてくださった多くのボランティアのみなさんに感謝します。

C.2 Terms and conditions for accessing or otherwise using Python

Python software and documentation are licensed under the *PSF License Agreement*.

Starting with Python 3.8.6, examples, recipes, and other code in the documentation are dual licensed under the PSF License Agreement and the *Zero-Clause BSD license*.

Some software incorporated into Python is under different licenses. The licenses are listed with code falling under that license. See *Licenses and Acknowledgements for Incorporated Software* for an incomplete list of these licenses.

C.2.1 PSF LICENSE AGREEMENT FOR PYTHON 3.14.0a3

```
1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and
   the Individual or Organization ("Licensee") accessing and otherwise using Python
   3.14.0a3 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 3.14.0a3 alone or in any derivative
   version, provided, however, that PSF's License Agreement and PSF's notice of
   copyright, i.e., "Copyright © 2001 Python Software Foundation; All Rights
   Reserved" are retained in Python 3.14.0a3 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 3.14.0a3 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
   3.14.0a3.

4. PSF is making Python 3.14.0a3 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 3.14.0a3 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.14.0a3
   FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF
   MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.14.0a3, 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 3.14.0a3, Licensee agrees
   to be bound by the terms and conditions of this License Agreement.
```

C.2.2 BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

```
1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at
   160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization
   ("Licensee") accessing and otherwise using this software in source or binary
   form and its associated documentation ("the Software").

2. Subject to the terms and conditions of this BeOpen Python License Agreement,
   BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license
   to reproduce, analyze, test, perform and/or display publicly, prepare derivative
   works, distribute, and otherwise use the Software alone or in any derivative
   version, provided, however, that the BeOpen Python License is retained in the
   Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an "AS IS" basis.
   BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED.  BY WAY OF
   EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR
   WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE
   USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR
   ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING,
   MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF
```

(次のページに続く)

(前のページからの続き)

```
   ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate upon a material breach of
   its terms and conditions.

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

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

C.2.3 CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

```
1. This LICENSE AGREEMENT is between the Corporation for National Research
   Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191
   ("CNRI"), and the Individual or Organization ("Licensee") accessing and
   otherwise using Python 1.6.1 software in source or binary form and its
   associated documentation.

2. Subject to the terms and conditions of this License Agreement, CNRI hereby
   grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce,
   analyze, test, perform and/or display publicly, prepare derivative works,
   distribute, and otherwise use Python 1.6.1 alone or in any derivative version,
   provided, however, that CNRI's License Agreement and CNRI's notice of copyright,
   i.e., "Copyright © 1995-2001 Corporation for National Research Initiatives; All
   Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version
   prepared by Licensee.  Alternately, in lieu of CNRI's License Agreement,
   Licensee may substitute the following text (omitting the quotes): "Python 1.6.1
   is made available subject to the terms and conditions in CNRI's License
   Agreement.  This Agreement together with Python 1.6.1 may be located on the
   internet using the following unique, persistent identifier (known as a handle):
   1895.22/1013.  This Agreement may also be obtained from a proxy server on the
   internet using the following URL: http://hdl.handle.net/1895.22/1013."
```

(次のページに続く)

(前のページからの続き)

```
3. In the event Licensee prepares a derivative work that is based on or
   incorporates Python 1.6.1 or any part thereof, and wants to make the derivative
   work available to others as provided herein, then Licensee hereby agrees to
   include in any such work a brief summary of the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis.  CNRI
   MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED.  BY WAY OF EXAMPLE,
   BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY
   OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF
   PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR
   ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF
   MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE
   THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of
   its terms and conditions.

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

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

C.2.4 CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

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

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

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

C.2.5 ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON 3.14.0a3 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 Licenses and Acknowledgements for Incorporated Software

This section is an incomplete, but growing list of licenses and acknowledgements for third-party software incorporated in the Python distribution.

C.3.1 Mersenne Twister

The `_random` C extension underlying the `random` module includes code based on a download from http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html. The following are the verbatim comments from the original code:

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

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

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

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

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

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

 3. The names of its contributors may not be used to endorse or promote
    products derived from this software without specific prior written
    permission.

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



Any feedback is very welcome.
```

(次のページに続く)

(前のページからの続き)

```
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 ソケット

The `socket` module uses the functions, `getaddrinfo()`, and `getnameinfo()`, which are coded in separate source files from the WIDE Project, https://www.wide.ad.jp/.

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

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

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

The `test.support.asynchat` and `test.support.asyncore` modules contain the following notice:

```
Copyright 1996 by Sam Rushing

                         All Rights Reserved
```

(次のページに続く)

(前のページからの続き)

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

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

C.3.4 Cookie management

The `http.cookies` module contains the following notice:

```
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
```

(次のページに続く)

(前のページからの続き)

```
PERFORMANCE OF THIS SOFTWARE.
```

C.3.5 Execution tracing

The `trace` module contains the following notice:

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

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

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

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

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



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

C.3.6 UUencode and UUdecode functions

The `uu` codec contains the following notice:

```
Copyright 1994 by Lance Ellinghouse
Cathedral City, California Republic, United States of America.
                        All Rights Reserved
Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
```

(次のページに続く)

(前のページからの続き)

```
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 Remote Procedure Calls

The `xmlrpc.client` module contains the following notice:

```
    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
```

(次のページに続く)

(前のページからの続き)

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

C.3.8 test_epoll

The `test.test_epoll` module contains the following notice:

```
Copyright (c) 2001-2006 Twisted Matrix Laboratories.

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

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

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
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` モジュールは kqueue インターフェースについての次の告知を含んでいます:

```
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
```

(次のページに続く)

(前のページからの続き)

```
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

The file `Python/pyhash.c` contains Marek Majkowski' implementation of Dan Bernstein's SipHash24 algorithm. It contains the following note:

```
<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:
```

(次のページに続く)

(前のページからの続き)

```
  Samuel Neves (supercop/crypto_auth/siphash24/little)
  djb (supercop/crypto_auth/siphash24/little2)
  Jean-Philippe Aumasson (https://131002.net/siphash/siphash24.c)
```

C.3.11 strtod と dtoa

The file `Python/dtoa.c`, which supplies C functions dtoa and strtod for conversion of C doubles to and from strings, is derived from the file of the same name by David M. Gay, currently available from https://web.archive.org/web/20220517033456/http://www.netlib.org/fp/dtoa.c. The original file, as retrieved on March 16, 2009, contains the following copyright and licensing notice:

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

C.3.12 OpenSSL

The modules `hashlib`, `posix` and `ssl` use the OpenSSL library for added performance if made available by the operating system. Additionally, the Windows and macOS installers for Python may include a copy of the OpenSSL libraries, so we include a copy of the OpenSSL license here. For the OpenSSL 3.0 release, and later releases derived from that, the Apache License v2 applies:

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

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
```

(次のページに続く)

(前のページからの続き)

```
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.
```

(次のページに続く)

(前のページからの続き)

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

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

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

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

4. Redistribution. You may reproduce and distribute copies of the
```

(次のページに続く)

(前のページからの続き)

```
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

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

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

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

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

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

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
```

(次のページに続く)

(前のページからの続き)

```
   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
```

(次のページに続く)

(前のページからの続き)

```
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
```

C.3.13 expat

The `pyexpat` extension is built using an included copy of the expat sources unless the build is configured `--with-system-expat`:

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

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

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

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

C.3.14 libffi

The `_ctypes` C extension underlying the `ctypes` module is built using an included copy of the libffi sources unless the build is configured `--with-system-libffi`:

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

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
``Software''), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
```

(次のページに続く)

(前のページからの続き)

```
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

The `zlib` extension is built using an included copy of the zlib sources if the zlib version found on the system is too old to be used for the build:

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

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

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

1. The origin of this software must not be misrepresented; you must not
   claim that you wrote the original software. If you use this software
   in a product, an acknowledgment in the product documentation would be
   appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be
   misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly        Mark Adler
```

(次のページに続く)

(前のページからの続き)

```
jloup@gzip.org          madler@alumni.caltech.edu
```

C.3.16 cfuhash

The implementation of the hash table used by the `tracemalloc` で使用しているハッシュテーブルの実装は、cfuhash プロジェクトのものに基づきます:

```
Copyright (c) 2005 Don Owens
All rights reserved.

This code is released under the BSD license:

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

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

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

  * Neither the name of the author nor the names of its
    contributors may be used to endorse or promote products derived
    from this software without specific prior written permission.

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

C.3.17 libmpdec

The `_decimal` C extension underlying the `decimal` module is built using an included copy of the libmpdec library unless the build is configured `--with-system-libmpdec`:

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

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

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

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

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

C.3.18 W3C C14N test suite

The C14N 2.0 test suite in the `test` package (`Lib/test/xmltestdata/c14n-20/`) was retrieved from the W3C website at https://www.w3.org/TR/xml-c14n2-testcases/ and is distributed under the 3-clause BSD license:

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

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

* Redistributions of works must retain the original copyright notice,
```

(次のページに続く)

(前のページからの続き)

```
  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 License:

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

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

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

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

(次のページに続く)

(前のページからの続き)

```
SOFTWARE.
```

C.3.20 asyncio

Parts of the `asyncio` module are incorporated from uvloop 0.16, which is distributed under the MIT license:

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

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

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

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

C.3.21 Global Unbounded Sequences (GUS)

The file `Python/qsbr.c` is adapted from FreeBSD's "Global Unbounded Sequences" safe memory reclamation scheme in subr_smr.c. The file is distributed under the 2-Clause BSD License:

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

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice unmodified, this list of conditions, and the following
   disclaimer.
2. Redistributions in binary form must reproduce the above copyright
```

(次のページに続く)

(前のページからの続き)

```
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

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

付録

D

COPYRIGHT

Python and this documentation is:

Copyright © 2001 Python Software Foundation. All rights reserved.

Copyright © 2000 BeOpen.com. All rights reserved.

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

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

ライセンスおよび許諾に関する完全な情報は、歴史とライセンス を参照してください。

索引

アルファベット以外

. . . , **107**
-?
　コマンドラインオプション, 6
%APPDATA%, 61
>>>, **107**
__future__, **115**
__slots__, **126**
クラス, **110**
コマンドラインオプション
　-?, 6
　-B, 7
　-b, 7
　BOLT_APPLY_FLAGS, 37
　BOLT_INSTRUMENT_FLAGS, 37
　--build, 44
　BZIP2_CFLAGS, 33
　BZIP2_LIBS, 33
　-c, 4
　CC, 32
　CFLAGS, 32
　--check-hash-based-pycs, 7
　CONFIG_SITE, 44
　CPP, 32
　CPPFLAGS, 33
　CURSES_CFLAGS, 33
　CURSES_LIBS, 33
　-d, 7
　--disable-gil, 32
　--disable-ipv6, 28
　--disable-safety, 42
　--disable-test-modules, 35
　-E, 7
　--enable-big-digits, 29
　--enable-bolt, 37
　--enable-experimental-jit, 32
　--enable-framework, 43, 44
　--enable-loadable-sqlite-extensions, 28
　--enable-optimizations, 36
　--enable-profiling, 38
　--enable-pystats, 30
　--enable-shared, 40
　--enable-slower-safety, 42
　--enable-universalsdk, 43
　--enable-wasm-dynamic-linking, 35
　--enable-wasm-pthreads, 35
　--exec-prefix, 35
　GDBM_CFLAGS, 33
　GDBM_LIBS, 33
　-h, 6
　--help, 6
　--help-all, 6
　--help-env, 6
　--help-xoptions, 6
　--host, 44
　HOSTRUNNER, 44
　-I, 7
　-i, 7
　-J, 13
　LDFLAGS, 33
　LIBB2_CFLAGS, 33
　LIBB2_LIBS, 33
　LIBEDIT_CFLAGS, 33
　LIBEDIT_LIBS, 33
　LIBFFI_CFLAGS, 33
　LIBFFI_LIBS, 34
　LIBLZMA_CFLAGS, 34
　LIBLZMA_LIBS, 34
　LIBMPDEC_CFLAGS, 34
　LIBMPDEC_LIBS, 34
　LIBREADLINE_CFLAGS, 34
　LIBREADLINE_LIBS, 34
　LIBS, 33
　LIBSQLITE3_CFLAGS, 34
　LIBSQLITE3_LIBS, 34
　LIBUUID_CFLAGS, 34
　LIBUUID_LIBS, 34
　-m, 4
　MACHDEP, 33
　-O, 8
　-OO, 8
　-P, 8
　PANEL_CFLAGS, 34
　PANEL_LIBS, 34
　PKG_CONFIG, 32
　PKG_CONFIG_LIBDIR, 32
　PKG_CONFIG_PATH, 32
　--prefix, 35
　-q, 8
　-R, 8
　-S, 9
　-s, 9
　TCLTK_CFLAGS, 34
　TCLTK_LIBS, 34
　-u, 9
　-V, 6
　-v, 9
　--version, 6
　-W, 9
　--with-address-sanitizer, 39
　--with-app-store-compliance, 43
　--with-assertions, 39
　--with-build-python, 44
　--with-builtin-hashlib-hashes, 41
　--with-computed-gotos, 37
　--with-dbmliborder, 29
　--with-dtrace, 39
　--with-ensurepip, 35
　--with-framework-name, 43, 44
　--with-hash-algorithm, 41
　--with-libc, 41
　--with-libm, 41
　--with-libs, 40
　--with-lto, 36
　--with-memory-sanitizer, 39
　--with-openssl, 41
　--with-openssl-rpath, 41
　--without-c-locale-coercion, 29
　--without-decimal-contextvar, 29
　--without-doc-strings, 37
　--without-mimalloc, 37
　--without-pymalloc, 37
　--without-readline, 41
　--without-static-libpython, 40
　--with-pkg-config, 30
　--with-platlibdir, 30
　--with-pydebug, 39
　--with-readline, 40
　--with-ssl-default-suites, 42
　--with-strict-overflow, 38
　--with-suffix, 29
　--with-system-expat, 40
　--with-system-libmpdec, 40
　--with-thread-sanitizer, 40
　--with-trace-refs, 39
　--with-tzpath, 29
　--with-undefined-behavior-sanitizer, 39
　--with-universal-archs, 43
　--with-valgrind, 39
　--with-wheel-pkg-dir, 30
　-X, 10
　-x, 10
　ZLIB_CFLAGS, 34
　ZLIB_LIBS, 35
コルーチン, **112**
コンテキスト管理プロトコル, **111**
ジェネレータ, **116**
メソッド, **121**
　magic, 120
　特殊, 127
ロケールエンコーディング, **120**
位置引数 (*positional argument*), **124**
特殊
　メソッド, 127

環境変数
%APPDATA%, 61
BASECFLAGS, 49
BASECPPFLAGS, 48
BLDSHARED, 51
CC, 48
CCSHARED, 49
CFLAGS, 36, 4850
CFLAGS_ALIASING, 49
CFLAGS_NODIST, 48, 49, 51
CFLAGSFORSHARED, 49
COMPILEALL_OPTS, 49
CONFIGURE_CFLAGS, 49
CONFIGURE_CFLAGS_NODIST, 49
CONFIGURE_CPPFLAGS, 48
CONFIGURE_LDFLAGS, 50
CONFIGURE_LDFLAGS_NODIST, 51
CPPFLAGS, 48, 51
CXX, 48
EXTRA_CFLAGS, 49
LDFLAGS, 48, 50, 51
LDFLAGS_NODIST, 50, 51
LDSHARED, 51
LIBS, 51
LINKCC, 50
OPT, 39, 49
PATH, 13, 25, 54, 55, 57, 6568, 71
PATHEXT, 57, 65
PROFILE_TASK, 36
PURIFY, 50
PY_BUILTIN_MODULE_CFLAGS, 50
PY_CFLAGS, 49
PY_CFLAGS_NODIST, 50
PY_CORE_CFLAGS, 50
PY_CORE_LDFLAGS, 51
PY_CPPFLAGS, 48
PY_LDFLAGS, 51
PY_LDFLAGS_NODIST, 51
PY_PYTHON, 72
PY_STDMODULE_CFLAGS, 50
PYLAUNCHER_ALLOW_INSTALL, 74
PYLAUNCHER_ALWAYS_INSTALL, 74
PYLAUNCHER_DEBUG, 73
PYLAUNCHER_DRYRUN, 73, 74
PYLAUNCHER_NO_SEARCH_PATH, 71
PYTHON_BASIC_REPL, 21
PYTHON_COLORS, 13, 21
PYTHON_CPU_COUNT, 12, 20
PYTHON_FROZEN_MODULES, 12, 20
PYTHON_GIL, 12, 21, 117
PYTHON_HISTORY, 21
PYTHON_PERF_JIT_SUPPORT, 12, 20
PYTHON_PRESITE, 12, 22
PYTHONASYNCIODEBUG, 17
PYTHONBREAKPOINT, 14
PYTHONCASEOK, 15
PYTHONCOERCECLOCALE, 18, 30
PYTHONDEBUG, 7, 14, 38
PYTHONDEVMODE, 11, 19
PYTHONDONTWRITEBYTECODE, 7, 15
PYTHONDUMPREFS, 21, 39
PYTHONDUMPREFSFILE, 21
PYTHONEXECUTABLE, 16
PYTHONFAULTHANDLER, 11, 17
PYTHONHASHSEED, 8, 9, 15
PYTHONHOME, 7, 13, 14, 75
PYTHONINSPECT, 7, 14
PYTHONINTMAXSTRDIGITS, 11, 15
PYTHONIOENCODING, 16, 19
PYTHONLEGACYWINDOWSFSENCODING, 18
PYTHONLEGACYWINDOWSSTDIO, 16, 18
PYTHONMALLOC, 17, 18, 37
PYTHONMALLOCSTATS, 18
PYTHONNODEBUGRANGES, 12, 20
PYTHONNOUSERSITE, 9, 16
PYTHONOPTIMIZE, 8, 14
PYTHONPATH, 7, 13, 14, 66, 74, 75
PYTHONPERFSUPPORT, 12, 20
PYTHONPLATLIBDIR, 14
PYTHONPROFILEIMPORTTIME, 11, 17
PYTHONPYCACHEPREFIX, 11, 15
PYTHONSAFEPATH, 8, 14
PYTHONSTARTUP, 7, 14, 15
PYTHONTRACEMALLOC, 11, 17
PYTHONUNBUFFERED, 9, 15
PYTHONUSERBASE, 16
PYTHONUTF8, 11, 19, 67
PYTHONVERBOSE, 9, 15
PYTHONWARNDEFAULTENCODING, 11, 20
PYTHONWARNINGS, 10, 16
TEMP, 61
関数, 115

A

abstract base class, 107
annotate function, 107
annotation, 107
asynchronous context manager, 108
asynchronous generator, 108
asynchronous generator iterator, 108
asynchronous iterable, 109
asynchronous iterator, 109
awaitable, 109

B

-B
コマンドラインオプション, 7
-b
コマンドラインオプション, 7
BDFL, 109
binary file, 109
BOLT_APPLY_FLAGS
コマンドラインオプション, 37
BOLT_INSTRUMENT_FLAGS
コマンドラインオプション, 37
borrowed reference, 109
--build
コマンドラインオプション, 44
bytecode, 110
bytes-like object, 109
BZIP2_CFLAGS
コマンドラインオプション, 33
BZIP2_LIBS
コマンドラインオプション, 33

C

-c
コマンドラインオプション, 4
callable, 110
callback, 110
CC
コマンドラインオプション, 32
C-contiguous, 111
CFLAGS, 36, 4850
コマンドラインオプション, 32
CFLAGS_NODIST, 48, 49, 51
--check-hash-based-pycs
コマンドラインオプション, 7
class variable, 110
closure variable, 110
complex number, 111
CONFIG_SITE
コマンドラインオプション, 44
context, 111
context manager, 111
context variable, 111
contiguous, 111
coroutine function, 112
CPP
コマンドラインオプション, 32
CPPFLAGS, 48, 51
コマンドラインオプション, 33
CPython, 112
current context, 112
CURSES_CFLAGS
コマンドラインオプション, 33
CURSES_LIBS
コマンドラインオプション, 33

D

-d
コマンドラインオプション, 7
decorator, 112
descriptor, 112
dictionary, 113
dictionary comprehension, 113
dictionary view, 113
--disable-gil
コマンドラインオプション, 32
--disable-ipv6
コマンドラインオプション, 28
--disable-safety
コマンドラインオプション, 42
--disable-test-modules
コマンドラインオプション, 35
docstring, 113
duck-typing, 113

E

-E
コマンドラインオプション, 7
EAFP, 113
--enable-big-digits
コマンドラインオプション, 29
--enable-bolt
コマンドラインオプション, 37
--enable-experimental-jit
コマンドラインオプション, 32
--enable-framework
コマンドラインオプション, 43, 44
--enable-loadable-sqlite-extensions
コマンドラインオプション, 28
--enable-optimizations
コマンドラインオプション, 36
--enable-profiling
コマンドラインオプション, 38
--enable-pystats
コマンドラインオプション, 30
--enable-shared
コマンドラインオプション, 40
--enable-slower-safety
コマンドラインオプション, 42
--enable-universalsdk
コマンドラインオプション, 43
--enable-wasm-dynamic-linking
コマンドラインオプション, 35
--enable-wasm-pthreads
コマンドラインオプション, 35
evaluate function, 113
--exec-prefix
コマンドラインオプション, 35
expression, 113
extension module, 114

F

f-string, 114
file object, 114
file-like object, 114
filesystem encoding and error handler, 114
finder, 114
floor division, 115
Fortran contiguous, 111
free threading, 115
free variable, 115
function annotation, 115

G

garbage collection, 115
GDBM_CFLAGS
    コマンドラインオプション, 33
GDBM_LIBS
    コマンドラインオプション, 33
generator expression, 116
generator iterator, 116
generic function, 116
generic type, 116
GIL, 116
global interpreter lock, 116

H

-h
    コマンドラインオプション, 6
hash-based pyc, 117
hashable, 117
--help
    コマンドラインオプション, 6
--help-all
    コマンドラインオプション, 6
--help-env
    コマンドラインオプション, 6
--help-xoptions
    コマンドラインオプション, 6
--host
    コマンドラインオプション, 44
HOSTRUNNER
    コマンドラインオプション, 44

I

-I
    コマンドラインオプション, 7
-i
    コマンドラインオプション, 7
IDLE, 117
immutable, 117
import path, 117
importer, 118
importing, 118
interactive, 118
interpreted, 118
interpreter shutdown, 118
iterable, 118
iterator, 118

J

-J
    コマンドラインオプション, 13

K

key function, 119
keyword argument, 119

L

lambda, 119
LBYL, 119
LDFLAGS, 48, 50, 51
    コマンドラインオプション, 33
LDFLAGS_NODIST, 51
LIBB2_CFLAGS
    コマンドラインオプション, 33
LIBB2_LIBS
    コマンドラインオプション, 33
LIBEDIT_CFLAGS
    コマンドラインオプション, 33
LIBEDIT_LIBS
    コマンドラインオプション, 33
LIBFFI_CFLAGS
    コマンドラインオプション, 33
LIBFFI_LIBS
    コマンドラインオプション, 34
LIBLZMA_CFLAGS
    コマンドラインオプション, 34
LIBLZMA_LIBS
    コマンドラインオプション, 34
LIBMPDEC_CFLAGS
    コマンドラインオプション, 34
LIBMPDEC_LIBS
    コマンドラインオプション, 34
LIBREADLINE_CFLAGS
    コマンドラインオプション, 34
LIBREADLINE_LIBS
    コマンドラインオプション, 34
LIBS
    コマンドラインオプション, 33
LIBSQLITE3_CFLAGS
    コマンドラインオプション, 34
LIBSQLITE3_LIBS
    コマンドラインオプション, 34
LIBUUID_CFLAGS
    コマンドラインオプション, 34
LIBUUID_LIBS
    コマンドラインオプション, 34
list, 120
list comprehension, 120
loader, 120

M

-m
    コマンドラインオプション, 4
MACHDEP
    コマンドラインオプション, 33
magic
    メソッド, 120
magic method, 120
mapping, 120
meta path finder, 120
metaclass, 120
method resolution order, 121
module, 121
module spec, 121
MRO, 121
mutable, 121

N

named tuple, 121
namespace, 122
namespace package, 122
nested scope, 122
new-style class, 122

O

-O
    コマンドラインオプション, 8
object, 122
-OO
    コマンドラインオプション, 8
OPT, 39
optimized scope, 122

P

-P
    コマンドラインオプション, 8
package, 123
PANEL_CFLAGS
    コマンドラインオプション, 34
PANEL_LIBS
    コマンドラインオプション, 34
parameter, 123
PATH, 13, 25, 54, 55, 57, 6568, 71
path based finder, 124
path entry, 123
path entry finder, 124
path entry hook, 124
path-like object, 124
PATHEXT, 57, 65
PEP, 124
PKG_CONFIG
    コマンドラインオプション, 32
PKG_CONFIG_LIBDIR
    コマンドラインオプション, 32
PKG_CONFIG_PATH
    コマンドラインオプション, 32
portion, 124
--prefix
    コマンドラインオプション, 35
PROFILE_TASK, 36
provisional API, 124
provisional package, 125
PY_PYTHON, 72
PYLAUNCHER_ALLOW_INSTALL, 74
PYLAUNCHER_ALWAYS_INSTALL, 74
PYLAUNCHER_DEBUG, 73
PYLAUNCHER_DRYRUN, 73, 74
PYLAUNCHER_NO_SEARCH_PATH, 71
Python 3000, 125
Python Enhancement Proposals
    PEP 1, 124
    PEP 7, 27
    PEP 8, 105
    PEP 11, 27, 53, 77
    PEP 238, 115
    PEP 278, 129
    PEP 302, 120
    PEP 338, 5
    PEP 343, 111
    PEP 362, 108, 123
    PEP 370, 9, 16
    PEP 397, 68
    PEP 411, 125
    PEP 420, 122, 124
    PEP 443, 116
    PEP 483, 116
    PEP 484, 108, 115, 116, 128, 129
    PEP 488, 8
    PEP 492, 108, 109, 112
    PEP 498, 114
    PEP 514, 68
    PEP 519, 124
    PEP 525, 109
    PEP 526, 108, 129
    PEP 528, 67
    PEP 529, 18, 67
    PEP 538, 19, 30
    PEP 585, 116
    PEP 649, 108
    PEP 683, 117
    PEP 703, 59, 89, 115, 117

PEP 3116, 129
PEP 3155, 125
PYTHON_COLORS, 13
PYTHON_CPU_COUNT, 12
PYTHON_FROZEN_MODULES, 12
PYTHON_GIL, 12, 117
PYTHON_PERF_JIT_SUPPORT, 12
PYTHON_PRESITE, 12
PYTHONCOERCECLOCALE, 30
PYTHONDEBUG, 7, 38
PYTHONDEVMODE, 11
PYTHONDONTWRITEBYTECODE, 7
PYTHONDUMPREFS, 39
PYTHONFAULTHANDLER, 11
PYTHONHASHSEED, 8, 9, 15
PYTHONHOME, 7, 13, 14, 75
Pythonic, **125**
PYTHONINSPECT, 7
PYTHONINTMAXSTRDIGITS, 11
PYTHONIOENCODING, 19
PYTHONLEGACYWINDOWSSTDIO, 16
PYTHONMALLOC, 18, 37
PYTHONNODEBUGRANGES, 12
PYTHONNOUSERSITE, 9
PYTHONOPTIMIZE, 8
PYTHONPATH, 7, 13, 14, 66, 74, 75
PYTHONPERFSUPPORT, 12
PYTHONPROFILEIMPORTTIME, 11
PYTHONPYCACHEPREFIX, 11
PYTHONSAFEPATH, 8
PYTHONSTARTUP, 7, 15
PYTHONTRACEMALLOC, 11
PYTHONUNBUFFERED, 9
PYTHONUTF8, 11, 19, 67
PYTHONVERBOSE, 9
PYTHONWARNDEFAULTENCODING, 11
PYTHONWARNINGS, 10

Q

-q
コマンドラインオプション, 8
qualified name, **125**

R

-R
コマンドラインオプション, 8
reference count, **126**
regular package, **126**
REPL, **126**

S

-S
コマンドラインオプション, 9
-s
コマンドラインオプション, 9
sequence, **126**
set comprehension, **126**
single dispatch, **126**
slice, **126**
soft deprecated, **127**
special method, **127**
statement, **127**
static type checker, **127**
strong reference, **127**

T

TCLTK_CFLAGS
コマンドラインオプション, 34
TCLTK_LIBS
コマンドラインオプション, 34
TEMP, 61
text encoding, **127**
text file, **127**
triple-quoted string, **128**
type, **128**
type alias, **128**
type hint, **128**

U

-u
コマンドラインオプション, 9
universal newlines, **128**

V

-V
コマンドラインオプション, 6
-v
コマンドラインオプション, 9
variable annotation, **129**
--version
コマンドラインオプション, 6
virtual environment, **129**
virtual machine, **129**
属性, **109**
引数 (*argument*), **108**

W

-W
コマンドラインオプション, 9
--with-address-sanitizer
コマンドラインオプション, 39
--with-app-store-compliance
コマンドラインオプション, 43
--with-assertions
コマンドラインオプション, 39
--with-build-python
コマンドラインオプション, 44
--with-builtin-hashlib-hashes
コマンドラインオプション, 41
--with-computed-gotos
コマンドラインオプション, 37
--with-dbmliborder
コマンドラインオプション, 29
--with-dtrace
コマンドラインオプション, 39
--with-ensurepip
コマンドラインオプション, 35
--with-framework-name
コマンドラインオプション, 43, 44
--with-hash-algorithm
コマンドラインオプション, 41
--with-libc
コマンドラインオプション, 41
--with-libm
コマンドラインオプション, 41
--with-libs
コマンドラインオプション, 40
--with-lto
コマンドラインオプション, 36
--with-memory-sanitizer
コマンドラインオプション, 39
--with-openssl
コマンドラインオプション, 41
--with-openssl-rpath
コマンドラインオプション, 41
--without-c-locale-coercion
コマンドラインオプション, 29
--without-decimal-contextvar
コマンドラインオプション, 29
--without-doc-strings
コマンドラインオプション, 37
--without-mimalloc
コマンドラインオプション, 37
--without-pymalloc
コマンドラインオプション, 37
--without-readline
コマンドラインオプション, 41
--without-static-libpython
コマンドラインオプション, 40
--with-pkg-config
コマンドラインオプション, 30
--with-platlibdir
コマンドラインオプション, 30
--with-pydebug
コマンドラインオプション, 39
--with-readline
コマンドラインオプション, 40
--with-ssl-default-suites
コマンドラインオプション, 42
--with-strict-overflow
コマンドラインオプション, 38
--with-suffix
コマンドラインオプション, 29
--with-system-expat
コマンドラインオプション, 40
--with-system-libmpdec
コマンドラインオプション, 40
--with-thread-sanitizer
コマンドラインオプション, 40
--with-trace-refs
コマンドラインオプション, 39
--with-tzpath
コマンドラインオプション, 29
--with-undefined-behavior-sanitizer
コマンドラインオプション, 39
--with-universal-archs
コマンドラインオプション, 43
--with-valgrind
コマンドラインオプション, 39
--with-wheel-pkg-dir
コマンドラインオプション, 30
永続オブジェクト (*immortal*), **117**

X

-X
コマンドラインオプション, 10
-x
コマンドラインオプション, 10

Z

Zen of Python, **129**
ZLIB_CFLAGS
コマンドラインオプション, 34
ZLIB_LIBS
コマンドラインオプション, 35