
Python Setup and Usage

リリース 3.7.17

Guido van Rossum
and the Python development team

6 月 28, 2023

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	10
第 2 章	Unix プラットフォームで Python を使う	19
2.1	最新バージョンの Python の取得とインストール	19
2.2	Python のビルド	20
2.3	Python に関するパスとファイル	20
2.4	その他	21
第 3 章	Windows で Python を使う	23
3.1	完全版インストーラ	24
3.2	Microsoft ストアパッケージ	29
3.3	nuget.org パッケージ	30
3.4	埋め込み可能なパッケージ	31
3.5	別のバンドル	32
3.6	Python を構成する	33
3.7	UTF-8 mode	34
3.8	Windows の Python ランチャ	35
3.9	モジュールの検索	40
3.10	追加のモジュール	42
3.11	Windows 上で Python をコンパイルする	43
3.12	ほかのプラットフォーム	44
第 4 章	Macintosh で Python を使う	45
4.1	MacPython の入手とインストール	45
4.2	IDE	47
4.3	追加の Python パッケージのインストール	47
4.4	Mac での GUI プログラミング	47
4.5	Mac 上の Python アプリケーションの配布	47
4.6	他のリソース	48
第 5 章	エディタと IDE	49

付録 A 章 用語集	51
付録 B 章 このドキュメントについて	69
B.1 Python ドキュメント 貢献者	69
付録 C 章 歴史とライセンス	71
C.1 Python の歴史	71
C.2 Terms and conditions for accessing or otherwise using Python	72
C.3 Licenses and Acknowledgements for Incorporated Software	76
付録 D 章 Copyright	89
索引	91
索引	91

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 での作業を楽しむ方法を説明します。

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 implementation detail: 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 [-bBdEhiIOqsSuvVWx?] [-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:

- `tty` デバイスに接続された標準入力とともに起動された場合、EOF (end-of-file 文字。UNIX では `Ctrl-D` で、Windows では `Ctrl-Z`, `Enter` で入力可能) を受け取るまで、コマンドを受け取り、それを実行します。
- 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。通常のも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)。

-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。通常のも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
```

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。

<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.

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

全コマンドラインオプションの短い説明を出力します。

-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 3.7.0b2+
```

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

バージョン 3.6 で追加: -VV オプション。

1.1.3 その他のオプション

-b

bytes または bytearray を str と比較した場合、または、bytes を int と比較した場合に警告を発生させます。このオプションを 2 度指定した場合 (-bb) は、エラーを発生させます。

バージョン 3.5 で変更: bytes と int の比較に影響します。

-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

パーサーのデバッグ出力を有効にします。(専門家専用です。コンパイルオプションに依存します)。
[PYTHONDEBUG](#) も参照してください。

-E

全ての PYTHON* 環境変数を見捨てます。例えば、[PYTHONPATH](#) や [PYTHONHOME](#) などです。

-i

最初の引数にスクリプトが指定された場合や **-c** オプションが利用された場合、`sys.stdin` がターミナルに出力されない場合も含めて、スクリプトかコマンドを実行した後にインタラクティブモードに入ります。*PYTHONSTARTUP* ファイルは読み込みません。

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** と **-s** も暗黙的に指定されます。隔離モードでは `sys.path` はスクリプトのディレクトリやユーザのサイトパッケージのディレクトリを含みません。全 *PYTHON** 環境変数も無視されます。ユーザが悪意のあるコードを注入するのを防ぐために更なる制限が課されるかもしれません。

バージョン 3.4 で追加。

-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。

-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。

バージョン 3.2 で追加。

-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.

以前のバージョンの Python では、このオプションはハッシュのランダム化を有効にします。これにより、`str`, `bytes`, `datetime` 型の `__hash__()` 値が予測不可能な乱数で "ソルト" されます。ハッシュ値は各 Python プロセスでは固定ですが、Python を繰り返し再実行した場合は別の予測不能な値になります。

ハッシュのランダム化は、`dict` の生成コストが最悪の $O(n^2)$ になるように注意深く選ばれた入力値を与えることによる DoS 攻撃への防御策として提供されています。詳細は <http://www.ocert.org/advisories/ocert-2011-003.html> を参照してください。

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。

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。

バージョン 3.2.3 で追加.

-s

user site-packages directory 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

-S

site モジュールの import と、そのモジュールが行っていた site ごとの sys.path への操作を無効にします。後に 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、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示すメッセージを出力します。二重に指定された場合 (-vv) は、モジュールを検索するときにチェックされた各ファイルに対して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。*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。典型的な警告メッセージは次の書式で表示されます。

`file:line: category: message`

デフォルトでは、各警告は発生したソース行ごとに一度だけ表示されます。このオプションは、警告をどれくらいの頻度で表示するかを制御します。

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)。

警告は、*PYTHONWARNINGS* 環境変数を使い、そして Python プログラムの中から warnings モジュールを利用して制御できます。

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`

(次のページに続く)

(前のページからの続き)

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

The action names can be abbreviated as desired (e.g. `-Wi`, `-Wd`, `-Wa`, `-We`) and the interpreter will resolve them to the appropriate action name.

See `warning-filter` and `describing-warning-filters` for more details.

-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` は `faulthandler` を有効化します;
- `-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.
- `-X tracemalloc` は `tracemalloc` モジュールを用いて Python のメモリ割り当てのトレースを開始します。デフォルトでは最新のフレームのみがトレースのトレースバックに格納されます。最大 `NFRAME` フレームのトレースバックでトレースを開始するには `-X tracemalloc=NFRAME` を使用してください。詳細は `tracemalloc.start()` を参照してください。
- `-X showalloccount` to output the total count of allocated objects for each type when the program finishes. This only works when Python was built with `COUNT_ALLOCS` defined.
- `-X int_max_str_digits` configures the integer string conversion length limitation. See also [PYTHONINTMAXSTRDIGITS](#).
- `-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](#).
- `-X dev`: enable CPython's "development mode", introducing additional runtime checks which are too expensive to be enabled by default. It should not be more verbose than the default if the code is correct: new warnings are only emitted when an issue is detected. Effect of the developer mode:
 - Add default warning filter, as `-W` default.
 - Install debug hooks on memory allocators: see the `PyMem_SetupDebugHooks()` C function.
 - Enable the `faulthandler` module to dump the Python traceback on a crash.

- Enable asyncio debug mode.
- Set the `dev_mode` attribute of `sys.flags` to `True`.
- `-X utf8` enables UTF-8 mode for operating system interfaces, overriding the default locale-aware mode. `-X utf8=0` explicitly disables UTF-8 mode (even when it would otherwise activate automatically). See [PYTHONUTF8](#) for more details.

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

バージョン 3.2 で変更: `-X` オプションが追加されました。

バージョン 3.3 で追加: `-X faulthandler` オプション。

バージョン 3.4 で追加: `-X showrefcount` および `-X tracemalloc` オプション。

バージョン 3.6 で追加: `-X showalloccount` オプション。

バージョン 3.7 で追加: `-X importtime`, `-X dev`, `-X utf8` オプション。

バージョン 3.7.14 で追加: The `-X int_max_str_digits` option.

1.1.4 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/pythonversion` と `exec_prefix/lib/pythonversion` 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/pythonversion` 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。

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。このファイル内のプロンプト `sys.ps1` や `sys.ps2`、ならびにフック
`sys.__interactivehook__` を変更することも出来ます。

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.

バージョン 3.7 で追加.

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。

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。

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 OS X 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。

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 ならびに datetime 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。

バージョン 3.2.3 で追加。

PYTHONINTMAXSTRDIGITS

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

バージョン 3.7.14 で追加。

PYTHONIOENCODING

この変数がインタ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

`user base directory` を設定します。これは `python setup.py install --user` 時に `user site-packages directory` と Distutils installation paths のパスを計算するのに使われます。

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。Mac OS X 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=module   # Warn once per calling module
PYTHONWARNINGS=once     # Warn once per Python process
PYTHONWARNINGS=ignore   # Never warn
```

See warning-filter and describing-warning-filters for more details.

PYTHONFAULTHANDLER

この環境変数が空でない文字列に設定された場合、起動時に `faulthandler.enable()` が呼び出されます。Python のトレースバックをダンプするために SIGSEGV、SIGFPE、SIGABRT、SIGBUS および SIGILL シグナルのハンドラを導入します。`-X faulthandler` オプションと等価です。

バージョン 3.3 で追加.

PYTHONTRACEMALLOC

この環境変数が空でない文字列に設定された場合、`tracemalloc` モジュールを利用して Python のメモリ割り当てのトレースを開始します。変数の値はトレース時のトレースバックで保持されるフレームの最大数です。例えば `PYTHONTRACEMALLOC=1` の場合、最新のフレームのみを保持します。詳細は `tracemalloc.start()` を参照してください、

バージョン 3.4 で追加.

PYTHONPROFILEIMPORTTIME

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

バージョン 3.7 で追加.

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。

バージョン 3.4 で追加.

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.

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.

See the default memory allocators and the `PyMem_SetupDebugHooks()` function (install debug hooks on Python memory allocators).

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

バージョン 3.6 で追加.

PYTHONMALLOCSTATS

If set to a non-empty string, Python will print statistics of the `pymalloc` memory allocator every time a new `pymalloc` object arena is created, and on shutdown.

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 errors 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()`.

利用可能な環境: Windows。

バージョン 3.6 で追加: より詳しくは [PEP 529](#) を参照をしてください。

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.

利用可能な環境: Windows。

バージョン 3.6 で追加.

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: *nix.

バージョン 3.7 で追加: 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 the CPython "development mode". See the `-X dev` option.

バージョン 3.7 で追加.

PYTHONUTF8

If set to 1, enables the interpreter's UTF-8 mode, where UTF-8 is used as the text encoding for system interfaces, regardless of the current locale setting.

This means that:

- `sys.getfilesystemencoding()` returns 'UTF-8' (the locale encoding is ignored).
- `locale.getpreferredencoding()` returns 'UTF-8' (the locale encoding is ignored, and the function's `do_setlocale` parameter has no effect).
- `sys.stdin`, `sys.stdout`, and `sys.stderr` all use UTF-8 as their text encoding, with the `surrogateescape` error handler being enabled for `sys.stdin` and `sys.stdout` (`sys.stderr` continues to use `backslashreplace` as it does in the default locale-aware mode)

As a consequence of the changes in those lower level APIs, other higher level APIs also exhibit different default behaviours:

- Command line arguments, environment variables and filenames are decoded to text using the UTF-8 encoding.
- `os.fsdecode()` and `os.fsencode()` use the UTF-8 encoding.
- `open()`, `io.open()`, and `codecs.open()` use the UTF-8 encoding by default. However, they still use the strict error handler by default so that attempting to open a binary file in text mode is likely to raise an exception rather than producing nonsense data.

Note that the standard stream settings in UTF-8 mode can be overridden by `PYTHONIOENCODING` (just as they can be in the default locale-aware mode).

If set to 0, the interpreter runs in its default locale-aware mode.

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

If this environment variable is not set at all, then the interpreter defaults to using the current locale settings, *unless* the current locale is identified as a legacy ASCII-based locale (as described for `PYTHONCOERCECLOCALE`), and locale coercion is either disabled or fails. In such legacy locales, the interpreter will default to enabling UTF-8 mode unless explicitly instructed not to do so.

Also available as the `-X utf8` option.

バージョン 3.7 で追加: より詳しくは [PEP 540](#) を参照してください。

1.2.1 デバッグモード変数

以下の環境変数は、`--with-pydebug` ビルドオプションを指定して構成されたデバッグビルド版の Python でのみ効果があります。

PYTHONTHREADDEBUG

設定された場合、Python はスレッドデバッグ情報を表示します。

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。

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

ほとんどの Linux ディストリビューションでは Python はプリインストールされており、それ以外でもパッケージとして利用可能です。しかし、ディストリビューションのパッケージでは利用したい機能が使えない場合があります。最新版の Python をソースから簡単にコンパイルすることができます。

Python がプリインストールされておらず、リポジトリにも無い場合、ディストリビューション用のパッケージを簡単につくることができます。以下のリンクを参照してください:

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-old.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/ch-creating-rpms.html
Fedora ユーザー向け

<http://www.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

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
```

2.1.3 OpenSolaris

OpenCSW から Python を入手することができます。Python の様々なバージョンが利用可能でインストールすることができます。e.g. `pkgutil -i python27`.

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。)

ビルドは通常次の手順で行います

```
./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/pythonversion` 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` (`${prefix}`) と `exec_prefix` (`${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
<code>exec_prefix/bin/python3</code>	インタプリタの推奨される場所
<code>prefix/lib/pythonversion</code> , <code>exec_prefix/lib/pythonversion</code>	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。
<code>prefix/include/pythonversion</code> , <code>exec_prefix/include/pythonversion</code>	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** コマンドを持ってないので、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。

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 インストールेशन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 (MSI パッケージ) 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プリタとライブラリをユーザごとに追加する Python インストールेशनを、これらインストーラ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のため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。

PEP 11 で明記しているとおり Python のリリースは、Microsoft が延長サポート期間であるとしている Windows プラットフォームのみをサポートします。つまり Python 3.7 は Vista とそれより新しい Windows をサポートするということです。Windows XP サポートが必要な場合は、Python 3.4 をインストールしてください。

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 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也没有ません。

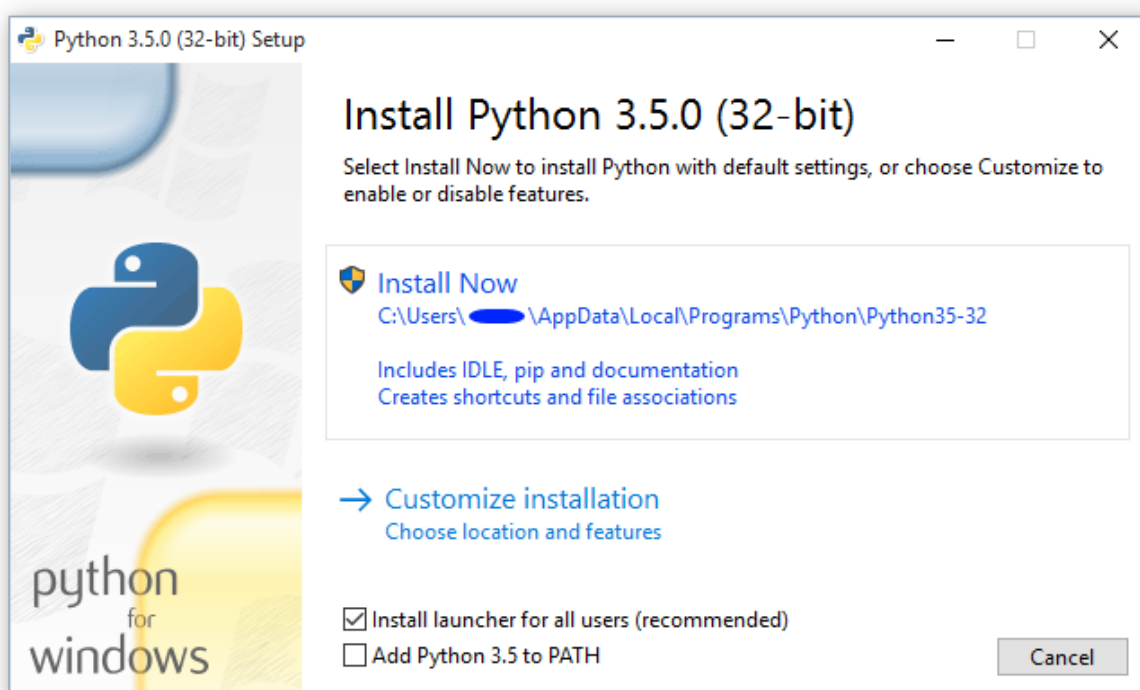
埋め込み可能なパッケージ は、他の大きなアプリケーションに埋め込むのに適した、Python の最小パッケージです。

3.1 完全版インストーラ

3.1.1 インストール手順

ダウンロードできる Python 3.7 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ン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。
- オプション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。

3.1.2 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 HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem\LongPathsEnabled の値を 1 に設定する必要があります。

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

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。

3.1.3 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。

完全にインストーラの GUI を隠して Python を静かにインストールするには、`/quiet` オプションを渡してください。対話的に設定するのはスキップしつつも進捗とエラーの表示は行いたいなら、`/passive` オプションを渡してください。`/uninstall` オプションも渡せます。これはプロンプトの表示なしに、即座に Python の削除を開始します。

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。
Default-AllUsersTargetDir	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\PythonXY, %LocalAppData%\Programs\PythonXY-32, %LocalAppData%\Programs\PythonXY-64 のいずれか
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
IncludeDoc	Python マニュアルをインストールする。	1
IncludeDebug	デバッグバイナリをインストールする。	0
IncludeDev	開発用のヘッダ・ライブラリをインストールする。	1
IncludeExe	python.exe とその関連ファイルをインストールする。	1
IncludeLauncher	Windows の Python ランチャをインストールする。	1
InstallLauncherAllUsers	Windows の Python ランチャをすべてのユーザにインストールする。	1
IncludeLib	標準ライブラリと拡張モジュールをインストールする。	1
IncludePip	バンドル版の pip と setuptools をインストールする。	1
IncludeSymbols	デバッグシンボル (*.pdb) をインストールする。	0
IncludeTclTk	Tcl/Tk サポートと IDLE 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 インストールを静かに行うには、以下コマンドを使えます (コマンドプロンプトより):

```
python-3.7.0.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-3.7.0.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>
```

3.1.4 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-3.7.0.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-3.7.0.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。

3.1.5 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。

3.2 Microsoft ストアパッケージ

バージョン 3.7.2 で追加。

注釈: 他のツールや他から提供されている Python が評価されている一方、Microsoft ストアパッケージは現時点で不安定だと見られています。Python そのものは安定しているのですが、そのインストール方法により Python 3.7 リリースの間、振る舞いや機能に変更されるかもしれません。

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 3.7" 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。

この 3 つのコマンドは全て、末尾にバージョン番号を付けても使えます。例えば、`python.exe` とするだけでなく `python3.exe` や `python3.x.exe` ともできます (3.x は 3.7 のような起動したい特定のバージョンです)。

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。

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。

3.2.1 既知の問題

現時点では、Microsoft ストアからインストールすると `py.exe` ランチャーは Python を起動するのには使えません。

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を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。

3.3 nuget.org パッケージ

バージョン 3.5.2 で追加.

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
Python 3.5.2

# With -ExcludeVersion
```

(次のページに続く)

(前のページからの続き)

```
> .\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。

nuget.org にあるパッケージ情報のページは、64-bit バージョンが www.nuget.org/packages/python、32-bit バージョンが www.nuget.org/packages/pythonx86 です。

3.4 埋め込み可能なパッケージ

バージョン 3.5 で追加。

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: 埋め込み用配布には [Microsoft C Runtime](#) は含まれません。これを提供するのはアプリケーションのインストーラの責務です。そのランタイムは既に以前にユーザのシステムにインストール済みかもしれませんし、Windows Update により自動で更新されているかもしれません。このことはシステムディレクトリに ucrtbase.dll があるか探せばわかります。

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 インストール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。

3.4.1 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が。

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。

3.4.2 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ールディレクトリ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 インストールを使うのとでの根本的な違いはありません。

3.5 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`。

Canopy 「包括的な Python 解析環境 (comprehensive Python analysis environment)」で、エディタとほかの開発ツールを含みます。

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。

3.6 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いますが、これは単独のシステムワイドなインストールの場合にだけ頼りになるものです。もしもあなたが定期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。

3.6.1 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 3.7;%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 2 と Python 3 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 バージョンで互換なコードだけを使うのでない限り、この環境変数は恒久的な設定をすべきではありません。

参考:

<https://www.microsoft.com/en-us/wdsi/help/folder-variables> Windows NT の環境変数

<https://technet.microsoft.com/en-us/library/cc754250.aspx> 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://technet.microsoft.com/en-us/library/cc755104.aspx> 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。

<https://support.microsoft.com/en-us/help/310519/how-to-manage-environment-variables-in-windows-xp>
Windows XP での環境変数の管理方法

<https://www.chem.gla.ac.uk/~louis/software/faq/q1.html> Louis J. Farrugia による、環境変数の設定についての説明。

3.6.2 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 3.7
```

3.7 UTF-8 mode

バージョン 3.7 で追加.

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

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

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

When UTF-8 mode is enabled:

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

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

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).
-

3.8 Windows の Python ランチャ

バージョン 3.3 で追加.

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ールの方を優先し、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。

3.8.1 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、ランチャがあなたの PATH に追加されます。ランチャは、入手可能なあらゆる Python のバージョンに互換であるため、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。ランチャが使えるかを確認するには以下コマンドをコマンドプロンプトより実行してください:

```
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。通常どおりを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 2.7 と 3.7) がインストールされている場合は、Python 3.7 が起動することになります。Python 2.7 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 -2.7
```

インストールしてある Python 2.x 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
```

最新バージョンの Python 2.x が起動するはずです。

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.
```

ユーザごとの Python インストールでは、インストール時にオプションで指定しない限り、ランチャは PATH に追加されません。

仮想環境 (Virtual environments)

バージョン 3.5 で追加.

(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 2.6 がインストールされている場合、最初の行を `#!/python2.6` に変更すると、2.6 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。

3.8.2 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 python
- /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 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/python2.7-32 は 32-bit の python 2.7 を使うよう要求します。

バージョン 3.7 で追加: 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。

/usr/bin/env 形式のシェバン行にはもう一つ更に特別な特性があります。インストールされている Python を探す前に、この形式は Python 実行ファイルを PATH から検索します。これは Unix の env プログラムに対応する振る舞いで、これも PATH からの検索をするものです。

3.8.3 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

3.8.4 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 "application data" ディレクトリ (つまり、Windows の関数 SHGetFolderPath に CSIDL_LOCAL_APPDATA を与えて呼ぶと返ってくるディレクトリ) 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ま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 バージョンのほうに常に優先されます。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。
- `python3.1` および `python2.7` 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=3.1-32` の場合、`python` コマンドは 32-bit 版の 3.1 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=3.1 の場合、python および python3 はどちらも 3.1 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=3.1 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=3.1
```

- PY_PYTHON=3 と PY_PYTHON3=3.1 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=3.1
```

3.8.5 診断

環境変数 PYLAUNCH_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ださい。

3.9 モジュールの検索

Python は通常そのライブラリ (と site-packages フォルダ) をインストールしたディレクトリに格納します。そのため、Python を C:\Python\ ディレクトリにインストールしたとすると、デフォルトのライブラリは C:\Python\Lib\ に存在し、サードパーティーのモジュールは C:\Python\Lib\site-packages\ に格納することになります。

sys.path を完全に上書きするには、DLL と同じ名前 (python37._pth) か実行可能ファイル (python._pth) と同じ名前の ._pth ファイルを作成し、1 行につき 1 つのパスを指定して sys.path に追加されるようにしてください。DLL 名に基づいたファイルは実行可能ファイル名に基づいたファイルを上書きします。これにより、望むならば、ランタイムを読み込むどんなプログラムもパスで制限できます。

ファイルが存在したときは、全てのレジストリと環境変数は無視され、隔離モードになり、そのファイルに import site と指定していない限りは site がインポートできなくなります。空行と # で始まる行は無視されます。それぞれのパスはファイルの場所を指す絶対パスあるいは相対パスです。site 以外のインポート文は許可されておらず、任意のコードも書けません。

import site を指定したときは、(アンダースコアが前に付かない) .pth ファイルは site モジュールにより通常通り処理されることに注意してください。

`._pth` ファイルが見付かったときは、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 が見付かった場合、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ール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ール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 call `Py_SetPath()` or (at least) `Py_SetProgramName()` before `Py_Initialize()`.
- 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.)

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

バージョン 3.6 で変更:

- Adds `._pth` file support and removes `applocal` option from `pyvenv.cfg`.
- Adds `pythonXX.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.

3.10 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。

3.10.1 PyWin32

Mark Hammond によって開発された `PyWin32` モジュールは、進んだ Windows 専用のサポートをするモジュール群です。このモジュールは以下のユーティリティを含んでいます:

- `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)` ユーザーインターフェイス

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

3.10.2 cx_Freeze

`cx_Freeze` is a `distutils` extension (see `extending-distutils`) which wraps Python scripts into executable Windows programs (`*.exe` files). When you have done this, you can distribute your application without requiring your users to install Python.

3.10.3 WConio

Python の進んだターミナル制御レイヤである `curses` は、Unix ライクシステムでしか使うことができません。逆に Windows 専用のライブラリ、Windows Console I/O for Python があります。

`WConio` は Turbo-C の `CONIO.H` のラッパーで、テキストユーザーインターフェースを作成するために利用することができます。

3.11 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 2015 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。

参考:

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

`MingW -- Python extensions`

3.12 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](#) は今でもサポートされています。
- [Cygwin](#) インストーラも Python インタープリタのインストールを提供しています。(cf. [Cygwin package source](#), [Maintainer releases](#))

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。

MACINTOSH で PYTHON を使う

著者 Bob Savage <bobsavage@mac.com>

Mac OS X が動作している Macintosh 上の Python は原則的には他の Unix プラットフォーム上の Python と非常によく似ていますが、IDE やパッケージ・マネージャなどの指摘すべき追加要素があります。

4.1 MacPython の入手とインストール

Mac OS X 10.8 には Apple によって Python 2.7 がプリインストールされています。Python の Web サイト (<https://www.python.org>) から最新バージョンの Python 3 を取得しインストールすることもできます。新しい Intel の CPU でも古い PPC の CPU でもネイティブに動作する ”ユニバーサル・バイナリ” ビルドの最新のものがあります。

インストールを行うといくつかのものが手に入ります:

- A Python 3.7 folder in your Applications folder. In here you find IDLE, the development environment that is a standard part of official Python distributions; and PythonLauncher, which handles double-clicking Python scripts from the Finder.
- Python 実行ファイルやライブラリを含む `/Library/Frameworks/Python.framework` フレームワーク。インストーラはシェルのパスにこの場所を追加します。MacPython をアンインストールするには、これら 3 つを削除すればよいだけです。Python 実行ファイルへのシンボリックリンクは `/usr/local/bin/` に置かれています。

Apple が提供している Python のビルドは `/System/Library/Frameworks/Python.framework` と `/usr/bin/python` にそれぞれインストールされています。これらは Apple が管理しているものであり Apple やサードパーティのソフトウェアが使用するので、編集したり削除したりしてはいけません。python.org から新しいバージョンの Python をインストールする場合には、それぞれ正常に動作する 2 つの異なる Python 環境がコンピュータにインストールされるため、意図する方の Python のパスを設定し、使用することが重要です。

IDLE にはヘルプメニューがあり Python のドキュメントにアクセスすることができます。もし Python が全くの初めての場合にはドキュメントのチュートリアルを最初から読み進めることをおすすめします。

もし他の Unix プラットフォームで Python を使い慣れている場合には Unix シェルからの Python スクリプトの実行についての節を読むことをおすすめします。

4.1.1 Python スクリプトの実行方法

Mac OS X で Python を始める最良の方法は統合開発環境である IDLE を使うことです、*IDE* 節を参照し IDE を実行しているときにヘルプメニューを使ってください。

もし Python スクリプトをターミナルのコマンドラインや Finder から実行したい場合は最初にエディタでスクリプトを作る必要があります。Mac OS X には **vim** や **emacs** などの Unix の標準のラインエディタが備わっています。もしもっと Mac らしいエディタが欲しい場合には、Bare Bones Software (<http://www.barebones.com/products/bbedit/index.html> を参照) の **BBEdition** や **TextWrangler** もしくは **TextMate** (<https://macromates.com/>) は良い選択候補です。他には **Gvim** (<http://macvim-dev.github.io/macvim/>) や **Aquamacs** (<http://aquamacs.org/>) などがあります。

ターミナルからスクリプトを実行するには `/usr/local/bin` がシェルのパスに含まれていることを確認してください。

Finder からスクリプトを実行するには 2 つの方法があります:

- **PythonLauncher** へドラッグする
- Finder の情報ウィンドウから **PythonLauncher** をそのスクリプト (もしくは `.py` スクリプト全て) を開くデフォルトのアプリケーションとして選び、スクリプトファイルをダブルクリックしてください。**PythonLauncher** の環境設定にはどのようにスクリプトを実行するかを管理する様々な設定があります。option キーを押しながらドラッグすることで実行するごとにこれらの設定を変えられますし、環境設定メニューから全ての実行に対して設定変更することもできます。

4.1.2 GUI でスクリプトを実行

古いバージョンの Python には、気を付けておかないといけない Mac OS X の癖があります: Aqua ウィンドウマネージャとやりとりをする (別の言い方をすると GUI を持つ) プログラムは特別な方法で実行する必要があります。そのようなスクリプトを実行するには **python** ではなく **pythonw** を使ってください。

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

4.1.3 Configuration

OS X 上の Python では **PYTHONPATH** のような全ての標準の Unix 環境変数が使えますが、Finder からプログラムを起動する場合このような環境変数を設定する方法は非標準であり Finder は起動時に `.profile` や `.cshrc` を読み込みません。`~/MacOSX/environment.plist` ファイルを作る必要があります。詳細については Apple の Technical Document QA1067 を参照してください。

MacPython の Python パッケージのインストールについてのさらなる情報は、*追加の Python パッケージのインストール* 節を参照してください。

4.2 IDE

MacPython には標準の IDLE 開発環境が付いてきます。http://www.hashcollision.org/hkn/python/idle_intro/index.html に IDLE を使うための良い入門があります。

4.3 追加の Python パッケージのインストール

追加の Python パッケージをインストールする方法がいくつかあります:

- パッケージは Python の標準の `distutils` モードを使ってインストールすることができます (`python setup.py install`)。
- 多くのパッケージは `setuptools` 拡張や `pip` ラッパーを使ってもインストールできます。<https://pip.pypa.io/> を参照してください。

4.4 Mac での 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。

PyObjC は Mac の最新の開発基盤である Apple の Objective-C/Cocoa フレームワークへの Python バインディングです。PyObjC の情報は <https://pypi.org/project/pyobjc/> にあります。

Python 標準の GUI ツールキットは、クロスプラットフォームの Tk ツールキット (<https://www.tcl.tk>) 上に構築された `tkinter` です。Tk の Aqua ネイティブバージョンは Apple が OS X にバンドルしており、最新バージョンは <https://www.activestate.com> からダウンロードおよびインストールできます; またソースからビルドすることもできます。

wxPython は別の人気のあるクロスプラットフォームの GUI ツールキットで Mac OS X 上でネイティブに動作します。パッケージとドキュメントは <https://www.wxpython.org> から利用できます。

PyQt は別の人気のあるクロスプラットフォームの GUI ツールキットで Mac OS X 上でネイティブに動作します。詳しい情報は <https://riverbankcomputing.com/software/pyqt/intro> にあります。

4.5 Mac 上の Python アプリケーションの配布

Mac 上の単独の Python アプリケーションをデプロイする標準のツールは `py2app` です。py2app のインストールと使用方法に関する情報は <http://undefined.org/python/#py2app> にあります。

4.6 他のリソース

MacPython メーリングリストは Mac での Python ユーザや開発者にとって素晴らしいサポートリソースです:

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

他の役に立つリソースは MacPython wiki です:

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

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。

用語集

>>> インタラクティブシェルにおけるデフォルトの Python プロンプトです。インタプリタでインタラクティブに実行されるコード例でよく出てきます。

... インタラクティブシェルにおいて、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。

2to3 Python 2.x のコードを Python 3.x のコードに変換するツールです。ソースコードを解析してその解析木を巡回 (traverse) することで検知できる、非互換性の大部分を処理します。

2to3 は標準ライブラリの `lib2to3` として利用できます。単体のツールとしての使えるスクリプトが `Tools/scripts/2to3` として提供されています。2to3-reference を参照してください。

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。

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。

ローカル変数のアノテーションは実行時にはアクセスできませんが、グローバル変数、クラス属性、関数のアノテーションはそれぞれモジュール、クラス、関数の `__annotations__` 特殊属性に保持されています。

機能の説明がある *variable annotation*, *function annotation*, [PEP 484](#), [PEP 526](#) を参照してください。

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を書くことができ、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できます。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、`__anext__()` メソッドを使って呼び出されたときに awaitable オブジェクトを返す *asynchronous iterator* です。この awaitable オブジェクトは、次の `yield` 式までの非同期ジェネレータ関数の本体を実行します。

`yield` にくるたびに、その位置での実行状態 (ローカル変数と保留状態の `try` 文) 処理は一時停止されます。`__anext__()` で返された他の awaitable によって **非同期ジェネレータイテレータ** が実際に再開されたとき、中断した箇所を取得します。**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。

awaitable (待機可能) `await` 式で使う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。

bytes-like object `bufferobjects` をサポートしていて、C 言語の意味で 連続した contiguous バッファ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。

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くに) 変更されることを目的としている変数です。

coercion (型強制) 同じ型の 2 引数を伴う演算の最中に行われる、ある型のインスタンスの別の型への暗黙の変換です。例えば、`int(3.15)` は浮動小数点数を整数 3 に変換します。しかし `3+4.5` では、各引数は型が異なり (一つは整数、一つは浮動小数点数)、加算をする前に同じ型に変換できなければ `TypeError`

例外が投げられます。型強制がなかったら、すべての引数は、たとえ互換な型であっても、単に `3+4.5` ではなく `float(3)+4.5` というように、プログラマーが同じ型に正規化しなければいけません。

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 manager (コンテキストマネージャ) `__enter__()` と `__exit__()` メソッドを定義することで `with` 文内の環境を管理するオブジェクトです。[PEP 343](#) を参照してください。

context variable (コンテキスト変数) コンテキストに依存して異なる値を持つ変数。これは、ある変数の値が各々の実行スレッドで異なり得るスレッドローカルストレージに似ています。しかしコンテキスト変数では、1 つの実行スレッドにいくつかのコンテキストがあり得、コンテキスト変数の主な用途は並列な非同期タスクの変数の追跡です。`contextvars` を参照してください。

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ある場合に利用されます。

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(...):
    ...
f = staticmethod(f)

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

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 (デスクリプタ) メソッド `__get__()`, `__set__()`, あるいは `__delete__()` を定義しているオブジェクトです。あるクラス属性がデスクリプタであるとき、属性探索によって、束縛されている特別な動作が呼び出されます。通常、`get`, `set`, `delete` のために `a.b` と書くと、`a` のクラス辞書内でオブジェクト `b` を検索しますが、`b` がデスクリプタであればそれぞれのデスクリプタメソッドが呼び出されます。デスクリプタの理解は、Python を深く理解する上で鍵となります。というのは、デスクリプタこそが、関数、メソッド、プロパティ、クラスメソッド、静的メソッド、そしてスーパークラスの参照といった多くの機能の基盤だからです。

デスクリプタのメソッドに関して詳しくは、`descriptors` を参照してください。

dictionary (辞書) 任意のキーを値に対応付ける連想配列です。`__hash__()` メソッドと `__eq__()` メソッドを実装した任意のオブジェクトをキーにできます。Perl ではハッシュ (hash) と呼ばれています。

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 クラス、関数、モジュールの最初の式である文字列リテラルです。そのスイートの実行時には無視されますが、コンパイラによって識別され、そのクラス、関数、モジュールの `__doc__` 属性として保存されます。イントロスペクションできる (訳注: 属性として参照できる) ので、オブジェクトのドキュメントを書く標準的な場所です。

duck-typing あるオブジェクトが正しいインタ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、マーフイーの法則)」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。

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 (ファイルオブジェクト) 下位のリソースへのファイル志向 API (`read()` や `write()` メソッドを持つもの) を公開しているオブジェクトです。ファイルオブジェクトは、作成された手段によって、実際のディスク上のファイルや、その他のタイプのストレージや通信デバイス (例えば、標準入出力、インメモリバッファ、ソケット、パイプ、等) へのアクセスを媒介できます。ファイルオブジェクトは *file-like objects* や *streams* とも呼ばれます。

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。

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。

Python 3.3 以降では 2 種類のファインダがあります。`sys.meta_path` で使用される [meta path finder](#) と、`sys.path_hooks` で使用される [path entry finder](#) です。

詳細については [PEP 302](#)、[PEP 420](#) および [PEP 451](#) を参照してください。

floor division 一番近い小さい整数に丸める数学除算。floor division 演算子は `//` です。例えば、`11 // 4` は 2 になり、`float` の true division の結果 2.75 と異なります。`(-11) // 4` は -2.75 を **小さい方に丸める**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。

function (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](#) と [PEP 484](#) を参照してください。

`__future__` 互換性のない新たな言語機能を現在のインタプリタで有効にするためにプログラマが利用できる擬似モジュールです。

`__future__` モジュールを `import` してその変数を評価すれば、新たな機能が初めて追加されたのがいつで、いつ言語デフォルトの機能になるかわかります:

```
>>> 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 (ジェネレータ式) イテレータを返す式です。普通の式に、ループ変数を定義する `for` 節、範囲、そして省略可能な `if` 節がつづいているように見えます。こうして構成された式は、外側の関数に向けて値を生成します:

```
>>> 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。

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。

過去に ”自由なマルチスレッド化” したインタプリタ (供用されるデータを細かい粒度でロックする) が開発されましたが、一般的なシングルスレッサの場合のパフォーマンスが悪かったので成功しませんでした。このパフォーマンスの問題を克服しようとする、実装がより複雑になり保守コストが増加すると考えられています。

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。

hashable (ハッシュ可能) **ハッシュ可能** なオブジェクトとは、生存期間中変わらないハッシュ値を持ち (`__hash__()` メソッドが必要)、他のオブジェクトと比較ができる (`__eq__()` メソッドが必要) オブジェクトです。同値なハッシュ可能オブジェクトは必ず同じハッシュ値を持つ必要があります。

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) 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。

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 (対話的) Python には対話的インタプリタがあり、文や式をインタプリタのプロンプトに入力すると即座に実行されて結果を見ることができます。`python` と何も引数を与えずに実行してください。(コンピュータのメインメニューから Python の対話的インタプリタを起動できるかもしれません。) 対話的インタプリタは、新しいアイデアを試してみたり、モジュールやパッケージの中を覗いてみる (`help(x)` を覚えておいてください) のに非常に便利なツールです。

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 (反復可能オブジェクト) 要素を一度に 1 つずつ返せるオブジェクトです。反復可能オブジェクトの例には、`(list, str, tuple` といった) 全てのシーケンス型や、`dict` や **ファイルオブジェクト** といった幾つかの非シーケンス型、あるいは *Sequence* 意味論を実装した `__iter__()` メソッドか `__getitem__()` メソッドを持つ任意のクラスのインスタンスが含まれます。

反復可能オブジェクトは `for` ループ内やその他多くのシーケンス (訳注: ここでのシーケンスとは、シーケンス型ではなくただの列という意味) が必要となる状況 (`zip()`, `map()`, ...) で利用できます。反復可能オブジェクトを組み込み関数 `iter()` の引数として渡すと、オブジェクトに対するイテレータを返します。このイテレータは一連の値を引き渡す際に便利です。通常は反復可能オブジェクトを使う際には、`iter()` を呼んだりイテレータオブジェクトを自分で操作する必要はありません。`for` 文ではこの操作を自動的に扱い、一時的な無名の変数を作成してループを回している間イテレータを保持します。**イテレータ**、**シーケンス**、**ジェネレータ** も参照してください。

iterator (イテレータ) 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 out 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います。

詳細な情報は `typeiter` にあります。

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()` をキー関数にを使えば、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` モジュールは `attrgetter()`, `itemgetter()`, `methodcaller()` という 3 つのキー関数コンストラクタを提供しています。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 (リスト) Python の組み込みの [シーケンス](#) です。リストという名前ですが、リンクリストではなく、他の言語で言う配列 (array) と同種のもので、要素へのアクセスは $O(1)$ です。

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 モジュールをロードするオブジェクト。`load_module()` という名前のメソッドを定義していなければなりません。ローダーは一般的に *finder* から返されます。詳細は [PEP 302](#) を、*abstract base class* については `importlib.abc.Loader` を参照してください。

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、`Mapping` か `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とって、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。

method (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 (メソッド解決順序) 探索中に基底クラスが構成要素を検索される順番です。2.3 以降の Python インタープリタが使用するアルゴリズムの詳細については [The Python 2.3 Method](#)

[Resolution Order](#) を参照してください。

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。

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つてのアクセスもできる任意の型やクラスに適用されています。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() の戻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
1024
>>> isinstance(sys.float_info, tuple) # kind of tuple
True
```

(上の例のように) いくつかの名前付きタプルは組み込み型になっています。その他にも名前付きタプルは、通常のクラス定義で tuple を継承し、名前のフィールドを定義して作成できます。そのようなクラスは手動で書いたり、collections.namedtuple() ファクトリ関数で作成したりできます。後者の方法は、手動で書いた名前付きタプルや組み込みの名前付きタプルには無い付加的なメソッドを追加できます。

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提供される [PEP 420 package](#)。Namespace package はおそらく物理表現を持たず、__init__.py ファイルがないため、*regular package* と異なります。

[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 (新スタイルクラス) 今では全てのクラスオブジェクトに使われている味付けの古い名前です。以前の Python のバージョンでは、新スタイルクラスのみが `__slots__`、デスクリプタ、`__getattr__()`、クラスメソッド、そして静的メソッド等の Python の新しい、多様な機能を利用できました。

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。

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** でまたは **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): ...
```

- **位置のみ**: 位置によってのみ与えられる引数を指定します。Python に引数が位置のみであることを定義する文法はありませんが、組み込み関数には位置のみの引数を持つもの (例: `abs()`) があります。
- **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 `sys.path_hook` リストにある callable で、指定された *path entry* にあるモジュールを見つける方法を知っている場合に *path entry finder* を返します。

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ナ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
>>> 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。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。`sys` モジュールは、プログラマーが任意のオブジェクトの参照カウントを知るための `getrefcount()` 関数を提供しています。

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。

__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 (シーケンス) 整数インデクスによる効率的な要素アクセスを `__getitem__()` 特殊メソッドを通じてサポートし、長さを返す `__len__()` メソッドを定義した *iterable* です。組み込みシーケンス型には、`list`, `str`, `tuple`, `bytes` などがあります。`dict` は `__getitem__()` と `__len__()` もサポートしますが、検索の際に整数ではなく任意の *immutable* なキーを使うため、シーケンスではなくマッピング (mapping) とみなされているので注意してください。

`collections.abc.Sequence` 抽象基底クラスは `__getitem__()` や `__len__()` だけでなく `count()`、`index()`、`__contains__()`、`__reversed__()` よりも豊富なインターフェイスを定義しています。この拡張されたインターフェイスを実装している型は `register()` を使用することで明示的に登録することが出来ます。

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。

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。

text encoding ユニコード文字列をエンコードするコーデックです。

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のできるので、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 (型) Python オブジェクトの型はオブジェクトがどのようなものかを決めます。あらゆるオブジェクトは型を持っています。オブジェクトの型は `__class__` 属性でアクセスしたり、`type(obj)` で取得したり出来ます。

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:

```
from typing import List, Tuple

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

これは次のように読みやすくなります:

```
from typing import List, Tuple

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戻り値の期待される型を指定する *annotation* です。

型ヒントは必須ではなく Python では強制ではありませんが、静的型解析ツールにとって有用であり、IDE のコード補完とリファクタリングの手助けになります。

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。

この機能について解説している *function annotation*, [PEP 484](#), [PEP 526](#) を参照してください。

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。

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](#) ページをご覧ください。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!

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

C.2.1 PSF LICENSE AGREEMENT FOR PYTHON 3.7.17

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 3.7.17 software in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 3.7.17 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001–2023 Python Software Foundation; All Rights Reserved" are retained in Python 3.7.17 alone or in any derivative version prepared by Licensee.
3. In the event Licensee prepares a derivative work that is based on or incorporates Python 3.7.17 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 3.7.17.
4. PSF is making Python 3.7.17 available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 3.7.17 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.7.17 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.7.17, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
8. By copying, installing or otherwise using Python 3.7.17, Licensee agrees to be bound by the terms and conditions of this License Agreement.

C.2.2 BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

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

(次のページに続く)

(前のページからの続き)

<http://www.pythonlabs.com/logos.html> may be used according to the permissions granted on that web page.

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

C.2.3 CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 ("CNRI"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI's License Agreement and CNRI's notice of copyright, i.e., "Copyright © 1995-2001 Corporation for National Research Initiatives; All Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI's License Agreement, Licensee may substitute the following text (omitting the quotes): "Python 1.6.1 is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: <http://hdl.handle.net/1895.22/1013>."
3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.
4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. This License Agreement shall be governed by the federal intellectual property

(次のページに続く)

(前のページからの続き)

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.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` 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.
```

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

(次のページに続く)

(前のページからの続き)

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, <http://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 `asynchat` and `asyncore` modules contain the following notice:

Copyright 1996 by Sam Rushing

All Rights Reserved

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

(次のページに続く)

(前のページからの続き)

```
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
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/
```

(次のページに続く)

(前のページからの続き)

`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 module 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,
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
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

`test_epoll` モジュールは次の告知を含んでいます:

```
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
```

(次のページに続く)

(前のページからの続き)

```
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
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
```

(次のページに続く)

(前のページからの続き)

```
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/siphash/
```

```
Solution inspired by code from:
```

```
Samuel Neves (supercop/crypto_auth/siphash24/little)
```

```
djb (supercop/crypto_auth/siphash24/little2)
```

```
Jean-Philippe Aumasson (https://131002.net/siphash/siphash24.c)
```

C.3.11 strtod 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 <http://www.netlib.org/fp/>. 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`, `ssl`, `crypt` use the OpenSSL library for added performance if made available by the operating system. Additionally, the Windows and Mac OS X installers for Python may include a copy of the OpenSSL libraries, so we include a copy of the OpenSSL license here:

LICENSE ISSUES

=====

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

OpenSSL License

```
/* =====
 * Copyright (c) 1998-2008 The OpenSSL Project. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in
 * the documentation and/or other materials provided with the
 * distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 * software must display the following acknowledgment:
 * "This product includes software developed by the OpenSSL Project
 * for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 * endorse or promote products derived from this software without
 * prior written permission. For written permission, please contact
 * openssl-core@openssl.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 * nor may "OpenSSL" appear in their names without prior written
 * permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 * acknowledgment:
 * "This product includes software developed by the OpenSSL Project
 * for use in the OpenSSL Toolkit (http://www.openssl.org/)"
 *
```

(次のページに続く)

(前のページからの続き)

```
* THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
* PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
* OF THE POSSIBILITY OF SUCH DAMAGE.
* =====
*
* This product includes cryptographic software written by Eric Young
* (eay@cryptsoft.com).  This product includes software written by Tim
* Hudson (tjh@cryptsoft.com).
*
*/
```

Original SSLeay License

```
/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
* All rights reserved.
*
* This package is an SSL implementation written
* by Eric Young (eay@cryptsoft.com).
* The implementation was written so as to conform with Netscapes SSL.
*
* This library is free for commercial and non-commercial use as long as
* the following conditions are aheared to.  The following conditions
* apply to all code found in this distribution, be it the RC4, RSA,
* lhash, DES, etc., code; not just the SSL code.  The SSL documentation
* included with this distribution is covered by the same copyright terms
* except that the holder is Tim Hudson (tjh@cryptsoft.com).
*
* Copyright remains Eric Young's, and as such any Copyright notices in
* the code are not to be removed.
* If this package is used in a product, Eric Young should be given attribution
* as the author of the parts of the library used.
* This can be in the form of a textual message at program startup or
* in documentation (online or textual) provided with the package.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the copyright
* notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
* notice, this list of conditions and the following disclaimer in the
```

(次のページに続く)

(前のページからの続き)

```

* documentation and/or other materials provided with the distribution.
* 3. All advertising materials mentioning features or use of this software
* must display the following acknowledgement:
* "This product includes cryptographic software written by
* Eric Young (eay@cryptsoft.com)"
* The word 'cryptographic' can be left out if the rouines from the library
* being used are not cryptographic related :-).
* 4. If you include any Windows specific code (or a derivative thereof) from
* the apps directory (application code) you must include an acknowledgement:
* "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
*
* THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*
* The licence and distribution terms for any publically available version or
* derivative of this code cannot be changed. i.e. this code cannot simply be
* copied and put under another distribution licence
* [including the GNU Public Licence.]
*/

```

C.3.13 expat

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,

```

(次のページに続く)

(前のページからの続き)

```
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` extension 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,
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
```

(次のページに続く)

(前のページからの続き)

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
jloup@gzip.org

Mark Adler
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)

(次のページに続く)

(前のページからの続き)

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` module is built using an included copy of the libmpdec library unless the build is configured `--with-system-libmpdec`:

Copyright (c) 2008-2016 Stefan Krah. All rights reserved.

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

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

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.

付録

D

COPYRIGHT

Python and this documentation is:

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

Copyright © 2000 BeOpen.com. All rights reserved.

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

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

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

..., 51
 -?
 command line option, 6
 2to3, 51
 >>>, 51
 __future__, 56
 __slots__, 64
 クラス, 53
 コルーチン, 54
 ジェネレータ, 57
 位置引数 (*positional argument*), 63
 環境変数
 exec_prefix, 20
 PATH, 10, 21, 24, 25, 27, 3336, 38
 PATHEXT, 27, 33
 prefix, 20
 PY_PYTHON, 39
 PYTHON*, 47
 PYTHONASYNCIODEBUG, 13
 PYTHONBREAKPOINT, 11
 PYTHONCASEOK, 11
 PYTHONCOERCECLOCALE, 15, 16
 PYTHONDEBUG, 6, 11
 PYTHONDEVMODE, 15
 PYTHONDONTWRITEBYTECODE, 6, 11
 PYTHONDUMPPREFS, 17
 PYTHONEXECUTABLE, 13
 PYTHONFAULTHANDLER, 13
 PYTHONHASHSEED, 7, 12
 PYTHONHOME, 6, 10, 11, 41, 42
 PYTHONINSPECT, 7, 11
 PYTHONINTMAXSTRDIGITS, 9, 12
 PYTHONIOENCODING, 12, 15, 16
 PYTHONLEGACYWINDOWSFSENCODING, 14
 PYTHONLEGACYWINDOWSSSTDIO, 12, 14
 PYTHONMALLOC, 13, 14
 PYTHONMALLOCSTATS, 14
 PYTHONNOUSERSITE, 12
 PYTHONOPTIMIZE, 7, 11
 PYTHONPATH, 6, 10, 11, 33, 41, 42, 46
 PYTHONPROFILEIMPORTTIME, 9, 13
 PYTHONSTARTUP, 7, 11
 PYTHONTHREADDEBUG, 17
 PYTHONTRACEMALLOC, 13
 PYTHONUNBUFFERED, 8, 11
 PYTHONUSERBASE, 12
 PYTHONUTF8, 10, 15, 35
 PYTHONVERBOSE, 8, 11
 PYTHONWARNINGS, 8, 13

A

abstract base class, 51
 annotation, 51
 asynchronous context manager, 52
 asynchronous generator, 52
 asynchronous generator iterator, 52

asynchronous iterable, 52
 asynchronous iterator, 52
 awaitable, 53

B

-B
 command line option, 6
 -b
 command line option, 6
 BDFL, 53
 binary file, 53
 bytecode, 53
 bytes-like object, 53

C

-c <command>
 command line option, 4
 C-contiguous, 54
 --check-hash-based-pycs default|always|never
 command line option, 6
 class variable, 53
 coercion, 53
 command line option
 -?, 6
 -B, 6
 -b, 6
 -c <command>, 4
 --check-hash-based-pycs default|always|never, 6
 -d, 6
 -E, 6
 -h, 6
 --help, 6
 -I, 7
 -i, 6
 -J, 10
 -m <module-name>, 4
 -O, 7
 -O0, 7
 -q, 7
 -R, 7
 -S, 8
 -s, 8
 -u, 8
 -V, 6
 -v, 8
 --version, 6
 -W arg, 8
 -X, 9
 -x, 9
 complex number, 54
 context manager, 54
 context variable, 54
 contiguous, 54
 coroutine function, 54
 CPython, 54

D

-d
 command line option, 6
decorator, 54
descriptor, 55
dictionary, 55
dictionary view, 55
docstring, 55
duck-typing, 55

E

-E
 command line option, 6
EAFF, 55
exec_prefix, 20
expression, 55
extension module, 55

F

f-string, 56
file object, 56
file-like object, 56
finder, 56
floor division, 56
Fortran contiguous, 54
function, 56
function annotation, 56

G

garbage collection, 57
generator, 57
generator expression, 57
generator iterator, 57
generic function, 57
GIL, 57
global interpreter lock, 57

H

-h
 command line option, 6
hash-based pyc, 57
hashable, 58
--help
 command line option, 6

I

-I
 command line option, 7
-i
 command line option, 6
IDLE, 58
immutable, 58
import path, 58
importer, 58
importing, 58
interactive, 58
interpreted, 58
interpreter shutdown, 58
iterable, 59
iterator, 59

J

-J
 command line option, 10

K

key function, 59
keyword argument, 59

L

lambda, 59
LBYL, 60
list, 60
list comprehension, 60
loader, 60

M

-m <module-name>
 command line option, 4
magic
 method, 60
magic method, 60
mapping, 60
meta path finder, 60
metaclass, 60
method, 60
 magic, 60
 special, 65
method resolution order, 60
module, 61
module spec, 61
MRO, 61
mutable, 61

N

named tuple, 61
namespace, 61
namespace package, 61
nested scope, 62
new-style class, 62

O

-O
 command line option, 7
object, 62
-OO
 command line option, 7

P

package, 62
parameter, 62
PATH, 10, 21, 24, 25, 27, 3336, 38
path based finder, 63
path entry, 63
path entry finder, 63
path entry hook, 63
path-like object, 63
PATHEXT, 27, 33
PEP, 63
portion, 63
prefix, 20
provisional API, 63
provisional package, 63
PY_PYTHON, 39
Python 3000, 64
Python Enhancement Proposals
 PEP 1, 63
 PEP 8, 49
 PEP 11, 23, 44
 PEP 238, 56
 PEP 278, 66
 PEP 302, 56, 60
 PEP 338, 5
 PEP 343, 54
 PEP 362, 52, 63
 PEP 370, 8, 12
 PEP 397, 35
 PEP 411, 63
 PEP 420, 56, 61, 63

PEP 443, 57
 PEP 451, 56
 PEP 484, 51, 56, 66
 PEP 488, 7
 PEP 492, 5254
 PEP 498, 56
 PEP 519, 63
 PEP 525, 52
 PEP 526, 51, 66
 PEP 528, 35
 PEP 529, 14, 35
 PEP 538, 15
 PEP 540, 16
 PEP 3116, 66
 PEP 3155, 64
 PYTHON*, 47
 PYTHONCOERCECLOCALE, 16
 PYTHONDEBUG, 6
 PYTHONDONTWRITEBYTECODE, 6
 PYTHONHASHSEED, 7, 12
 PYTHONHOME, 6, 10, 11, 41, 42
 Pythonic, 64
 PYTHONINSPECT, 7
 PYTHONINTMAXSTRDIGITS, 9
 PYTHONIOENCODING, 15, 16
 PYTHONLEGACYWINDOWSTDIO, 12
 PYTHONMALLOC, 14
 PYTHONOPTIMIZE, 7
 PYTHONPATH, 6, 10, 11, 33, 41, 42, 46
 PYTHONPROFILEIMPORTTIME, 9
 PYTHONSTARTUP, 7
 PYTHONUNBUFFERED, 8
 PYTHONUTF8, 10, 15, 35
 PYTHONVERBOSE, 8
 PYTHONWARNINGS, 8

Q

-q
 command line option, 7
 qualified name, 64

R

-R
 command line option, 7
 reference count, 64
 regular package, 64

S

-S
 command line option, 8
 -s
 command line option, 8
 sequence, 65
 single dispatch, 65
 slice, 65
 special
 method, 65
 special method, 65
 statement, 65

T

text encoding, 65
 text file, 65
 triple-quoted string, 65
 type, 65
 type alias, 65
 type hint, 66

U

-u

 command line option, 8
 universal newlines, 66

V

-V
 command line option, 6
 -v
 command line option, 8
 variable annotation, 66
 --version
 command line option, 6
 virtual environment, 66
 virtual machine, 67
 属性, 53
 引数 (*argument*), 51

W

-W arg
 command line option, 8

X

-X
 command line option, 9
 -x
 command line option, 9

Z

Zen of Python, 67