Using Python

リリース *2.6.2*

Guido van Rossum
Fred L. Drake, Jr., editor

2011 年 01 月 23 日

Python Software Foundation
Email: docs@python.org

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 13
2.1 最新バージョンの Python の取得とインストール . . . 13
2.2 Python のビルド . . . 14
2.3 Python に関係するパスとファイル . . . 14
2.4 その他 . . . 15
2.5 エディタ . . . 15

第 3 章 Windows で Python を使う 17
3.1 Python のインストール . . . 17
3.2 別のバンドル . . . 18
3.3 Python を構成する . . . 18
3.4 Windows 上で Python をコンパイルする . . . 21
3.5 その他のリソース . . . 22

第 4 章 Macintosh で Python を使う 23
4.1 MacPython の入手とインストール . . . 23
4.2 IDE . . . 25
4.3 追加の Python パッケージのインストール . . . 25
4.4 Mac での GUI プログラミング . . . 26
4.5 Mac 上の Python アプリケーションの配布 . . . 26
4.6 アプリケーションスクリプティング . . . 26
4.7 他のリソース . . . 27

第 5 章 このドキュメントについて 29

第 6 章　翻訳者一覧 (敬称略) 31

付 録 A 章用語集 33

付 録 B 章このドキュメントについて 43
B.1 Python ドキュメント 貢献者 43

付 録 C 章History and License 45
C.1 Python の歴史 45
C.2 Terms and conditions for accessing or otherwise using Python 47
C.3 Licenses and Acknowledgements for Incorporated Software 50

付 録 D 章Copyright 61

索引 63

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

第 1

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 [-BdEiOQsStuUvVWxX3?] [-c command | -m module-name | script | - ] [args]
```

もちろん、もっとも一般的な利用方法は、シンプルにスクリプトを実行するものです。

```
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 シェルに似ていますが、より多くのの実行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。
- ファイル名引数を指定されるか、ファイルを標準入力に渡された場合は、そのファイルから読み込んだスクリプトを実行します。

- ディレクトリ名を引数に受け取ったときは、そのディレクトリから適切な名前のスクリプトファイルを読み込んで実行します。
- `-c` コマンド オプションを利用して起動された場合、 コマンド として渡された Python の文を実行します。 コマンド の部分には改行で区切られた複数行を指定することもできます。行の先頭の空白文字は Python 文の重要要素です！
- `-m` モジュール名 として Python モジュールパスにあるモジュールを指定された場合、そのモジュールをスクリプトとして実行します。

非インタラクティブモードでは、入力の全体が実行前にパースされます。

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

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

  このオプションが指定された場合、 `sys.argv` の先頭要素は `"-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 することが可能になります。)

-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。`module-name` は有効な Python のモジュール名であるべきですが、実装がそれを矯正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。)

  ノート: このオプションはビルトインモジュールや 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` の先頭要素はモジュールファイルのフルパスになります。*`-c`* オプションのように、カレントディレクトリが `sys.path` の先頭に追加されます。

  Many standard library modules contain code that is invoked on their execution as a script. An example is the `timeit` module:: 多くの標準ライブラリモジュールが、スクリプトとして実行された時のコードを持っています。例えば、 `timeit` モジュールは次のように実行可能です。

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

参考:

`runpy.run_module()` この機能の実際の実装

PEP 338 – Executing modules as scripts

バージョン 2.4 で追加. バージョン 2.5 で変更: パッケージ内のモジュールを指定できるようになりました。

-

標準入力 (`sys.stdin`) からコマンドを読み込みます。標準入力がターミナルだった場合、 *-i* オプションを含みます。

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` ファイルを含むディレクトリ、`__main__.py` ファイルを含む zip ファイルのいづれかの、ファイルシステム上の (絶対あるいは相対) 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` の先頭要素は、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。バージョン 2.5 で変更: トップレベルに `__main__.py` ファイルを持つディレクトリや zip ファイルが有効な Python スクリプトとなりました。

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]` は空白文字列 (`""`) で、カレントディレクトリが `sys.path` の先頭に追加されます。

参考:

tut-invoking

1.1.2 一般オプション

-?

-h
-help
全てのコマンドラインオプションの短い説明を表示します。バージョン 2.5 で変更: `--help` 形式

-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 2.5.1
```

バージョン 2.5 で変更: `--version` 形式

1.1.3 その他のオプション

-B
Python は import したソースモジュールの `.pyc` や `.pyo` 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。バージョン 2.6 で追加.

-d
パーサーのデバッグ出力を有効にします。(魔法使い専用です。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*` 環境変数を無視します。例えば、`PYTHONPATH` と `PYTHONHOME` などです。バージョン 2.2 で追加.

-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。

-O
基本的な最適化を有効にします。コンパイル済み (*bytecode*) ファイルの拡張子を `.pyc` から `.pyo` に変更します。 `PYTHONOPTIMIZE` も参照してください。

-OO
-O の最適化に加えて、ドキュメンテーション文字列の除去も行ないます。

-Q `<arg>`
除算制御。引数は以下のうち 1 つでなければなりません:

old int/int と long/long の除算は、 int か long を返します。 (デフォルト)

new 新しい除算方式。 int/int や long/long の除算が float を返します。

warn 古い除算方式で、 int/int や long/long 除算に警告を表示します。

warnall 古い除算方式で、全ての除算演算子に対して警告を表示します。

参考:

`Tools/scripts/fixdiv.py` `warnall` を使っています.

PEP 238 – Changing the division operator

-s

sys.path にユーザー site ディレクトリを追加しません。バージョン 2.6 で追加.

参考:

PEP 370 – Per user site-packages directory

-S

`site` モジュールのインポートを無効にし、そのモジュールで行われている場所独自の `sys.path` 操作を無効にします。

-t

ソースファイルが、タブ幅に依存して意味が変わるような方法でタブ文字とスペースを混ぜて含んでいる場合に警告を発生させます。このオプションを 2 重にする (*-tt*) とエラーになります。

-u

stdin, stdout, stderr のバッファを強制的に無効にします。関係するシステムでは、stdin, stdout, stderr をバイナリモードにします。

`file.readlines()` や *bltin-file-objects* (`for line in sys.stdin`) はこのオプションに影響されない内部バッファリングをしています。これを回避したい場合は、 `while 1:` ループの中で `file.readline()` します。

`PYTHONUNBUFFERED` も参照してください。

-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表示します。2 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表示します。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形をしています:

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

デフォルトでは、各警告は発生したソース業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 Python プログラムの中から `warnings` モジュールを利用して制御することができます。

引数の一番シンプルな形は、以下のアクション文字列 (かそのユニークな短縮形) を単体で利用するものです。

ignore 全ての警告を無視する。

default 明示的にデフォルトの動作 (ソース行ごとに 1 度警告を表示する) を要求する。

all 警告が発生するたびに表示する (これは、ループの中などで同じソース行により繰り返し警告が発生された場合に、大量のメッセージを表示します。)

module 各モジュールで最初に発生した警告を表示する。

once プログラムで最初に発生した警告だけを表示する。

error 警告メッセージを表示する代わりに例外を発生させる。

引数の完全形は次のようになります:

```
action:message:category:module:line
```

ここで、 *action* は上で説明されたものですが、残りのフィールドにマッチしたメッセージにだけ適用されます。空のフィールドは全ての値にマッチします。空のフィールドの後ろは除外されます。 *message* フィールドは表示される警告メッセージの先頭に、大文字小文字を無視してマッチします。 *category* フィールドは警告カテゴリにマッチします。これはクラス名でなければなりません。*category* のマッチは、メッセージの実際の警告カテゴリーが指定された警告カテゴリーのサブクラスかどうかをチェックします。完全なクラス名を指定しなければなりません。*module* フィールドは、(完全正規形 (fully-qualified) の) モジュール名に対してマッチします。このマッチは大文字小文字を区別します。*line* フィールドは行番号にマッチします。 0 は全ての行番号にマッチし、省略した時と同じです。

参考:

`warnings` – the warnings module

PEP 230 – Warning framework

-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。

ノート: エラーメッセージ内の行番号は -1 されます。

-3

Python 3.x との、*2to3* によって簡単に解決できない互換性の問題について警告します。以下のものが該当します。

- `dict.has_key()`
- `apply()`
- `callable()`
- `coerce()`
- `execfile()`
- `reduce()`
- `reload()`

これらを使うと、`DeprecationWarning` を発生させます。バージョン 2.6 で追加.

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。

-U

全ての文字列リテラルを、全部 unicode にします。このオプションはあなたの世界を破壊してしまうかもしれないので、このオプションを使おうとしないでください。これは、通常とは違うマジックナンバーを使って `.pyc` ファイルを生成します。ファイルの先頭に次のように書いて、このオプションの代わりにモジュール単位で unicode リテラルを有効にできます。

```
from __future__ import unicode_literals
```

詳細は `__future__` を参照してください。

-X

別の Python の実装が独自の目的で利用する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 の動作に影響します。

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 から探されます。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
もし読込み可能ファイルの名前であれば、インタラクティブモードで最初のプロンプトを表示する前にそのファイル内の Python コマンドを実行します。このファイルはインタラクティブコマンドが実行されるのと同じ名前空間の中で実行されるので、このファイル内で定義されたり import されたオブジェクトはインタラクティブセッションから制限無しに利用することができます。このファイルで sys.ps1 と sys.ps2 を変更してプロンプトを変更することもできます。

PYTHONY2K
この変数に空でない文字列を設定すると、 time モジュールが文字列で指定される日付に 4 桁の年を含むことを要求するようになります。そうでなければ、2 桁の年は time モジュールのドキュメントに書かれているルールで変換されます。

PYTHONOPTIMIZE

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

PYTHONDEBUG
この変数に空でない文字列を設定すると、 *-d* オプションを指定したのと同じになります。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と、 *-i* オプションを指定したのと同じになります。

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と、 *-u* オプションを指定したのと同じになります。

PYTHONVERBOSE
この変数に空でない文字列を設定すると、 *-v* オプションを指定したのと同じになります。整数を指定した場合、 *-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と、 Python は `import` 文で大文字/小文字を区別しません。これは Windows でのみ動作します。

PYTHONDONTWRITEBYTECODE
この環境変数が設定されていると、Python はソースモジュールの import 時に `.pyc`, `.pyo` ファイルを生成しません。バージョン 2.6 で追加.

PYTHONIOENCODING
stdin/stdout/stderr のエンコーディングを強制します。シンタックスは `encodingname:errorhandler` です。`:errorhandler` の部分はオプションで、 `str.encode()` の引数と同じ意味です。バージョン 2.6 で追加.

PYTHONNOUSERSITE
If this is set, Python won't add the user site directory to sys.path この環境変数が設定されている場合、 Python はユーザー site ディレクトリを sys.path に追加しません。バージョン 2.6 で追加.

参考:

PEP 370 – Per user site-packages directory

PYTHONUSERBASE
ユーザー site ディレクトリのベースディレクトリを設定します。バージョン 2.6 で追加.

参考:

PEP 370 – Per user site-packages directory

PYTHONEXECUTABLE
この環境変数が設定されていると、 `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。

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。バージョン 2.6 で変更: 以前は、この変数は `THREADDEBUG` という名前でした。

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リファレンスカウントをダンプします。

PYTHONMALLOCSTATS
設定された場合、Python は、新しいオブジェクトアリーナを作成するときと、シャットダウン時に、メモリアロケーション統計情報を表示します。

第 2

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、それ以外の Linux ディストリビューションでも 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:

http://www.linux.com/articles/60383 for Debian users

http://linuxmafia.com/pub/linux/suse-linux-internals/chapter35.html for OpenSuse users

http://docs.fedoraproject.org/drafts/rpm-guide-en/ch-creating-rpms.html for Fedora users

http://www.slackbook.org/html/package-management-making-packages.html for Slackware users

2.1.2 FreeBSD と OpenBSD

- FreeBSD ユーザーは、 Python のパッケージを追加するために次のようにしてください。

```
pkg_add -r python
```

- OpenBSD ユーザーはこうです。

```
pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/4.2/packages/<アーキテクチャ>/pyth
```

たとえば、 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-2.5.1p
```

2.1.3 OpenSolaris

OpenSolaris に最新版の Python をインストールするには、 blastwave (http://www.blastwave.org/howto.html) をインストールして、プロンプトから "pkg_get -i python" とタイプしてください。

2.2 Python のビルド

If you want to compile CPython yourself, first thing you should do is get the CPython を自分でコンパイルしたい場合、最初にするべきことはソース 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することができます。

ビルド手順は通常次のステップで構成されます。

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

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

> **警告:** `make install` は `python` バイナリを上書きまたは覆い隠すかもしれません。そのため、 `make install` の代わりに `exec_prefix/bin/pythonversion` しかインストールしない `make altinstall` が推奨されます。

2.3 Python に関係するパスとファイル

ローカルインストールの慣習による違いを扱うために、 `prefix` (`${prefix}`) と `exec_prefix` (`${exec_prefix}`) はインストール依存で、 GNU software によって解釈されます。これらは同じかもしれません。

例えば、ほとんどの Linux システムでは、デフォルトでは両方が /usr です。

ファイル/ディレクトリ	意味
exec_prefix/bin/python	インタプリタの推奨される場所
prefix/lib/pythonversion, exec_prefix/lib/pythonversion	標準モジュールを格納するディレクトリの、推奨される場所
prefix/include/pythonversion, exec_prefix/include/pythonversion	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
~/.pythonrc.py	user モジュールによって読み込まれる、ごとの初期化ファイル。デフォルトでは、ほとんどのアプリケーションは利用しません。

2.4 その他

Python スクリプトを Unix で簡単に使うために、例えば次のようにして、そのスクリプトを実行ファイルにし

```
$ chmod +x script
```

そして適切な 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 python
```

で、 PATH 全体から Python インタプリタを探します。しかし、幾つかの Unix は **env** コマンドをもっていないので、インタプリタのパスを /usr/bin/python のようにハードコードしなければならないかもしれません。

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

2.5 エディタ

Vim と Emacs は Python をよくサポートした、素晴らしいエディタです。これらのエディタで Python のコードを書く方法についての詳しい情報は、次の場所を参照してください。

- http://www.vim.org/scripts/script.php?script_id=790
- http://sourceforge.net/projects/python-mode

Geany はたくさんの言語をサポートした素晴らしい IDE です。さらなる情報は、http://gean.uvena.de/ を読んでください。

Komodo edit も非常に良い IDE です。これもたくさんの言語をサポートしています。さらなる情報は、http://www.activestate.com/store/productdetail.aspx?prdGuid=20f4ed15-6684-4118-a78b-d37ff4058c5f を読んでください。

第 3

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 独特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。

3.1 Python のインストール

ほとんどの Unix システムやサービスと異なり、Windows は Python に依存しておらず、プリインストールの Python はありません。しかし、 CPython チームは長年にわたり、コンパイル済みの Windows インストーラ (MSI パッケージ) を リリース毎に用意しています。

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 (ユーザーと開発者の不足のために) サポートされなくなっています。全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。

- DOS と Windows 3.x は Python 2.0 から廃止予定になり、Python 2.1 でこれらのシステム専用のコードは削除されました。
- 2.5 まで、 Python は Windows 95, 98, ME で動きました (ですが、すでにインストール時に廃止予定の警告をだしていました)。 Python 2.6 (とその後の全てのリリース) は、これらの OS のサポートが止められ、新しいリリースは Windows NT ファミリーしか考慮されていません。
- 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。これは “Interpreters” の下に置かれています。(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 (and DOS) を参照してください。

参考:

Python on XP “7 Minutes to “Hello World!”” by Richard Dooling, 2006

Installing on Windows in “Dive into Python: Python from novice to pro” by Mark Pilgrim, 2004, ISBN 1-59059-356-1

For Windows users in “Installing Python” in “A Byte of Python” by Swaroop C H, 2003

3.2 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

Enthought Python Distribution (PyWin32 などの) 人気のあるモジュールとそのドキュメント、 Python の拡張をビルドするためのツールスイート

これらのパッケージは 古い バージョンの Python をインストールするかもしれないことに気をつけてください。

3.3 Python を構成する

Python を完全に動かすために、幾つかの環境設定を変更しなければならないかもしれません。

3.3.1 補足: 環境変数の設定

Windows は環境変数を変更するためのビルトインのダイアログを持っています。(以降のガイドは XP のクラシカルビューに適用されます。) マシンのアイコン (たいていデスクトップにあって “マイ コンピュータ” と呼ばれます) を右クリックして、そこにある プロパティ を選択します。詳細設定 タブを開いて、 環境変数 ボタンをクリックします。

ここまでのパスをまとめると:

マイ コンピュータ → プロパティ→ 詳細設定 → 環境変数

このダイアログで、ユーザーとシステムの環境変数を追加したり修正できます。システム変数を変更するには、マシンへの無制限アクセス (管理者権限) が必要です。

環境に変数を追加するもう一つの方法は、 **set** コマンドを使うことです。

```
set PYTHONPATH=%PYTHONPATH%;C:\My_python_lib
```

この設定を永続化するために、このコマンドラインを autoexec.bat に追加することができます。 **msconfig** はこのファイルを編集する GUI です。

もっと直接的な方法で環境変数を見ることができます。コマンドプロンプトはパーセント記号で囲まれた文字列を自動的に展開します。

```
echo %PATH%
```

この動作についての詳細は **set /?** を見てください。

参考:

http://support.microsoft.com/kb/100843 Windows NT の環境変数

http://support.microsoft.com/kb/310519 Windows XP での環境変数の管理方法

http://www.chem.gla.ac.uk/~louis/software/faq/q1.html Setting Environment variables, Louis J. Farrugia

3.3.2 Python 実行ファイルを見つける

スタートメニューに自動的に作られた Python interpreter のメニューエントリを使うのと別に、DOS プロンプトから Python を実行したいかもしれません。そのためには、 %PATH% 環境変数に Python ディストリビューションのディレクトリを、セミコロンで他のエントリと区切って含めるように設定する必要があります。変数の設定例は次のようになります (最初の 2 つのエントリが Windows のデフォルトだと仮定します):

```
C:\WINDOWS\system32;C:\WINDOWS;C:\Python26
```

コマンドプロンプトから **python** をタイプすると、 Python インタプリタを起動します。これで、スクリプトをコマンドラインオプション付きで実行することも可能です。 コマンドライン ドキュメントを参照してください。

3.3.3 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する。なので、 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\ に格納されます。

追加のフォルダを Python の import 機構の検索対象に含めることもできます。 PYTHONPATH を 環境変数 で解説されているように利用し、sys.path を変更してく

ださい。Windows では、 ドライブ識別子 (C:\ など) と区別するために、パスはセミコロンで区切られています。

モジュール検索パスの変更は、レジストリの HKLM\SOFTWARE\Python\PythonCore\version\Pyt キーからも可能です。このキーのデフォルト値と同じように、セミコロンで区切られたパス文字列を持ったサブキーがあれば、その各パスを探します。複数のサブキーを作成することができ、path に辞書順で追加されます。便利なレジストリエディタは **regedit** です。(スタート → ファイル名を指定して実行 から "regedit" とタイプすることで起動することができます。)

3.3.4 スクリプトを実行する

Python スクリプト (.py 拡張子を持ったファイル) はデフォルトで **python.exe** に起動されます。この実行ファイルは、プログラムが GUI を使う場合でもターミナルを開きます。ターミナル無しでスクリプトを実行したい場合は、拡張子 .pyw を使うとそのスクリプトがデフォルトでは **pythonw.exe** で実行されるようになります。(2 つの実行ファイルは両方とも Python をインストールしたディレクトリの直下にあります。) **pythonw.exe** は起動時にターミナルを開きません。

You can also make all .py scripts execute with **pythonw.exe**, setting this through the usual facilities, for example (might require administrative rights): 全ての .py スクリプトを **pythonw.exe** で実行するように設定することもできます。例えば (管理者権限が必要):

1. コマンドプロンプトを起動する
2. .py スクリプトに正しいファイルグループを関連付ける:

```
assoc .py=Python.File
```

3. 全ての Python ファイルを新しい実行ファイルにリダイレクトする:

```
ftype Python.File=C:\Path\to\pythonw.exe "%1" %*
```

Additional modules 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 専用の標準モジュールは、*mswin-specific-services* に書かれています。

3.3.5 PyWin32

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

3.3.6 Py2exe

Py2exe は `distutils` 拡張 (*extending-distutils* を参照) で、Python スクリプトを Windows 実行可能プログラム (`*.exe` ファイル) にラップします。これを使えば、ユーザーに Python のインストールをさせなくても、アプリケーションを配布することができます。

3.3.7 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フェースを作成するために利用することができます。

3.4 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、最初にすることはソース を取得することです。最新版リリースのソースをダウンロードするか、最新の チェックアウトを取得することができます。

公式の Python リリースをビルドするのに使われている Microsoft Visual C++ コンパイラのために、ソースツリーは ソリューション・プロジェクトファイルを含んでいます。適切なディレクトリにある `readme.txt` を参照してください。

ディレクトリ	**MSVC** バージョン	**Visual Studio** バージョン
`PC/VC6/`	6.0	97
`PC/VS7.1/`	7.1	2003
`PC/VS8.0/`	8.0	2005
`PCbuild/`	9.0	2008

これらのビルドディレクトリの全てが完全にサポートされているわけではありません。使用しているバージョンの公式リリースが利用しているコンパイラのバージョンについては、リリースノートを参照してください。

ビルドプロセスに関する一般的な情報は `PC/readme.txt` をチェックしてください。

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 by Trent Apted et al, 2007

3.5 その他のリソース

参考:

Python Programming On Win32 "Help for Windows Programmers" by Mark Hammond and Andy Robinson, O'Reilly Media, 2000, ISBN 1-56592-621-8

A Python for Windows Tutorial by Amanda Birmingham, 2004

第 4

Macintosh で Python を使う

Author 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。

Mac 特有のモジュールについては *mac-specific-services* に書かれています。

Mac OS 9 もしくはそれ以前の Mac 上の Python は Unix や Windows 上の Python とは大きく掛け離れていますが、そのプラットフォームは既にサポートされておらずこのマニュアルで扱う範囲を越えているので、Python 2.4 以降を扱うことにします。Mac OS 9 用の最新のバージョン 2.3 リリースのインストーラやそのドキュメントについては http://www.cwi.nl/~jack/macpython を参照してください。

4.1 MacPython の入手とインストール

Mac OS X 10.5 には Apple によって Python 2.5.1 がプリインストールされています。Python の Web サイト (http://www.python.org) から最新バージョンの Python 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:

- `Applications` フォルダにある `MacPython 2.5` フォルダ. 公式の Python ディストリビューションに含まれる開発環境 IDLE; Finder から Python スクリプトをダブルクリックしたときに起動する PythonLauncher; Python スクリプトを単独のアプリケーションに変換する “Build Applet” ツールがここにあります.
- 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 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ことにした場合には、異なるが動作する 2 つの Python 環境があなたのコンピュータにあることに注意し、パスの設定や Python の使い方と実際にしたいこととが食い違っていないことが重要です。

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 プラットフォーム上の Python に慣れている場合は 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.shtml を参照) の **BBEdit** や **TextWrangler** もしくは **TextMate** (see http://macromates.com/) は良い選択候補です。他には **Gvim** (http://macvim.org) 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のには 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 について、気を付けておくべき 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 **pythonw** を使ってください。

Python 2.5 では、:program:*python* も **pythonw** も使えます。

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://hkn.eecs.berkeley.edu/~dyoo/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:

- http://pythonmac.org/packages/ には Python 2.5、2.4、2.3 用のコンパイルされたパッケージがあります。
- 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** 拡張を使ってもインストールできます。

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 http://pyobjc.sourceforge.net にあります。

標準の Python の GUI ツールキットは `Tkinter` で、クロスプラットフォームの Tk ツールキット (http://www.tcl.tk) をベースにしています。Aqua ネイティブ版の Tk は OS X に入っており、最新バージョンが http://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 http://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。さらなる情報は http://www.riverbankcomputing.co.uk/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

フォルダ MacPython 2.5 にある “Built Applet” ツールはあなたのマシンの小さな Python スクリプトを標準の Mac アプリケーションとして実行できるようなパッケージを作るのに優れています。しかし、このツールは Python アプリケーションを他のユーザに配布するのには向いていません。

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法に関する情報は http://undefined.org/python/#py2app にあります。

4.6 アプリケーションスクリプティング

Python は Apple の Open Scripting Architecture (OSA) を通して他の Mac アプリケーションのスクリプトとしても使えます; http://appscript.sourceforge.net を参照してください。Appscript は高水準でユーザに優しい Apple event ブリッジで普通の Python スクリプトを使って Mac OS X アプリケーションを操作することができます。Appscript は Mac の自動操作を行うための Apple 自身による言語 *AppleScript* の代わりとして本格的に使えます。それに関連するパッケージ *PyOSA* は Python のための OSA 言語コンポーネントで、それを使うことで Python のコードをどんな OSA 対応のアプリケーション (Script Editor、Mail、 iTunes など) からも実行できます。PyOSA によって Python は完全な AppleScript との通信端末になります。

4.7 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しいサポートリソースです:

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

他の役に立つリソースは MacPython wiki です:

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

第 5

このドキュメントについて

この文書は、Python ドキュメント翻訳プロジェクトによる “Using Python” の日本語訳版です。日本語訳に対する質問や提案などがありましたら、Python ドキュメント翻訳プロジェクトのメーリングリスト

http://www.python.jp/mailman/listinfo/python-doc-jp

または、プロジェクトのバグ管理ページ

http://sourceforge.jp/tracker/?atid=116&group_id=11&func=browse

までご報告ください。

第 6

翻訳者一覧 (敬称略)

- masaki
- INADA Naoki <inada-n at klab.jp>
- cocoatomo

付 録 **A**

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。インタラクティブに実行されるコードサンプルとしてよく出てきます。

... インタラクティブシェルにおける、インデントされたコードブロックや対応する括弧 (丸括弧 ()、角括弧 []、curly brace{ }) の内側で表示されるデフォルトのプロンプト。

2to3 Python 2.x のコードを Python 3.x のコードに変換するツール。ソースコードを解析して、その解析木を巡回 (traverse) して、非互換なコードの大部分を処理する。

2to3 は、`lib2to3` モジュールとして標準ライブラリに含まれています。スタンドアローンのツールとして使うときのコマンドは `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) Abstract Base Classes (ABCs と略されます) 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になる場合にインタフェースを定義する方法を提供します。Python は沢山のビルトイン ABCs を、(`collections` モジュールで) データ構造、(`numbers` モジュールで) 数値型、(`io` モジュールで) ストリーム型で提供いています。`abc` モジュールを利用して独自の ABC を作成することもできます。

argument (引数) 関数やメソッドに渡された値。関数の中では、名前の付いたローカル変数に代入されます。

関数やメソッドは、その定義中に位置指定引数 (positional arguments, 訳注: `f(1, 2)` のように呼び出し側で名前を指定せず、引数の位置に引数の値を対応付けるもの) とキーワード引数 (keyword arguments, 訳注: `f(a=1, b=2)` のように、引数名に引数の値を対応付けるもの) の両方を持つことができます。位置指定引数とキーワード引数は可変長です。関数定義や呼び出しは、`*` を使って、不定数個の位置指定引数をシーケンス型に入れて受け取ったり渡したりすることができます。同じく、キーワード引数は `**` を使って、辞書に入れて受け取ったり渡したりできます。

引数リスト内では任意の式を使うことができ、その式を評価した値が渡されます。

attribute (属性) オブジェクトに関連付けられ、ドット演算子を利用して名前で参照される値。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。

BDFL 慈悲ぶかき独裁者 (Benevolent Dictator For Life) の略です。Python の作者、 Guido van Rossum のことです。

bytecode (バイトコード) Python のソースコードはバイトコードへとコンパイルされます。バイトコードは Python プログラムのインタプリタ内部での形です。バイトコードはまた、 `.pyc` や `.pyo` ファイルにキャッシュされ、同じファイルを二度目に実行した際により高速に実行できるようにします (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)。このバイトコードは、各々のバイトコードに対応するサブルーチンを呼び出すような "仮想計算機 (*virtual machine*)" で動作する "中間言語 (intermediate language)" といえます。

class (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。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。

classic class (旧スタイルクラス) `object` を継承していないクラス全てを指します。新スタイルクラス (*new-style class*) も参照してください。旧スタイルクラスは Python 3.0 で削除されます。

coercion (型強制) 同じ型の 2 つの引数を要する演算の最中に、ある型のインスタンスを別の型に暗黙のうちに変換することです。例えば、`int(3.15)` は浮動小数点数を整数の `3` にします。しかし、`3+4.5` の場合、各引数は型が異なっていて (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しなければいけません。そうでないと、`TypeError` 例外が投げられます。2 つの被演算子間の型強制は組み込み関数の `coerce` を使って行えます。従って、`3+4.5` は `operator.add(*coerce(3, 4.5))` を呼び出すことに等しく、`operator.add(3.0, 4.5)` という結果になります。型強制を行わない場合、たとえ互換性のある型であっても、すべての引数はプログラマーが、単に `3+4.5` とするのではなく、`float(3)+4.5` というように、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。虚数は虚数単位元 (`-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、工業では `j` と書かれます。

Python は複素数に組込みで対応し、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` となります。`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ー) `with` 文で扱われる、環境を制御するオブジェクト。`__enter__()` と `__exit__()` メソッドを定義することで作られる。

PEP 343 を参照。

CPython Python プログラミング言語の基準となる実装。CPython という単語は、この実装を Jython や IronPython といった他の実装と区別する必要が有る文脈で利用されます。

decorator (デコレータ) 関数を返す関数。通常、`@wrapper` という文法によって関数を変換するのに利用されます。デコレータの一般的な利用レとして、`classmethod()` と `staticmethod()` があります。

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(...):
    ...
```

デコレータについてのより詳しい情報は、*the documentation for function definition* 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__()` が定義されている 新スタイル *(new-style)* のオブジェクトです。あるクラス属性がデスクリプタである場合、その属性を参照するときに、そのデスクリプタに束縛されている特別な動作を呼び出します。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* がデスクリプタの場合にはデスクリプタで定義されたメソッドを呼び出します。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、そしてスーパクラスの参照といった多くの機能の基盤だからです。

dictionary (辞書) 任意のキーを値に対応付ける連想配列です。`dict` の使い方は `list` に似ていますが、ゼロから始まる整数に限らず、`__hash__()` 関数を実装している全て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。

docstring クラス、関数、モジュールの最初の式となっている文字列リテラルです。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正しい場所です。

duck-typing Python 的なプログラムスタイルではオブジェクトの型を（型オブジェクトとの関係ではなく）メソッドや属性といったシグネチャを見ることで判断します。(「もしそれがガチョウのようにみえて、ガチョウのように鳴けば、それはガチョウである」) 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 (polymorphic な置換を許可することによって) 柔軟性を増すことができます。

duck-typing は `type()` や `isinstance()` を避けます。(ただし、duck-typing を抽象ベースクラス (abstract base classes) で補完することもできます。) その代わりに `hasattr()` テストや *EAFP* プログラミングを利用します。

EAFP 「認可をとるより許しを請う方が容易 (easier to ask for forgiveness than permission、マーフィーの法則)」の略です。 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) 何かの値に評価される、一つづきの構文 (a piece of syntax). 言い換えると、リテラル、名前、属性アクセス、演算子や関数呼び出しといった、値を返す式の要素の組み合わせ。他の多くの言語と違い、Python 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。`print` や `if` のように、式にはならない、文 (*statement*) 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。ユーザーコードや Python のコアとやりとりするために、Python の C API を利用します。

finder モジュールの *loader* を探すオブジェクト。`find_module()` という名前のメソッドを実装していなければなりません。詳細については **PEP 302** 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。ゼロ個以上の引数を受け取り、それを関数の本体を実行するときに諒できます。*argument* や *method* も参照してください。

__future__ 互換性のない新たな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。例えば、式 `11/4` は現状では `2` になります。この式を実行しているモジュールで

```
from __future__ import division
```

を行って 真の除算操作 *(true division)* を有効にすると、式 `11/4` は `2.75` になります。実際に `__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、いつデフォルトの機能になる予定かわかります。

```
>>> 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) もう使われなくなったメモリを開放する処理。Python は、Python は参照カウントと循環参照を見つけて破壊する循環参照コレクタを使ってガベージコレクションを行います。

generator (ジェネレータ) イテレータを返す関数です。 `return` 文の代わりに `yield` 文を使って呼び出し側に要素を返す他は、通常の関数と同じに見えます。

よくあるジェネレータ関数は一つまたはそれ以上の `for` ループや `while` ループを含んでおり、ループの呼び出し側に要素を返す (`yield`) ようになっています。ジェネレータが返すイテレータを使って関数を実行すると、関数は `yield` キーワードで (値を返して) 一旦停止し、 `next()` を呼んで次の要素を要求するたびに実行を再開します。

generator expression (ジェネレータ式) ジェネレータを返す式です。普通の式に、ループ変を定義している `for` 式、範囲、そしてオプショナルな `if` 式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、外側の関数に対して値を生成します。:

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

GIL グローバルインタプリタロック (*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* の VM(*virtual machine*) の中で一度に 1 つのスレッドだけが動作することを保証するために使われているロックです。このロックによって、同時に同じメモリにアクセスする 2 つのプロセスは存在しないと保証されているので、CPython を単純な構造にできるのです。インタプリタ全体にロックをかけると、多重プロセサ計算機における並列性の恩恵と引き換えにインタプリタの多重スレッド化を簡単に行えます。かつて “スレッド自由な (free-threaded)” インタプリタを作ろうと努力したことがありましたが、広く使われている単一プロセッサの場合にはパフォーマンスが低下するという事態に悩まされました。

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__()` か `__cmp__()` 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。

辞書のキーや集合型のメンバーは、内部でハッシュ値を使っているので、ハッシュ可能オブジェクトである必要があります。

Python の全ての不変 (*immutable*) なビルドインオブジェクトはハッシュ可能です。リストや辞書といった変更可能なコンテナ型はハッシュ可能ではありません。

ユーザー定義クラスのインスタンスはデフォルトでハッシュ可能です。それらは、比較すると常に不等で、ハッシュ値は `id()` になります。

IDLE Python の組み込み開発環境 (Integrated DeveLopment Environment) です。IDLE は Python の標準的な配布物についてくる基本的な機能のエディタとインタプリタ環境です。初心者に向いている点として、 IDLE はよく洗練され、複数プラットフォームで動作する GUI アプリケーションを実装したい人むけの明解なコード例にもなっています。

immutable (不変オブジェクト) 固定の値を持ったオブジェクトです。変更不能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。不変オブジェクトは、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。辞書におけるキーがその例です。

integer division (整数除算) 剰余を考慮しない数学的除算です。例えば、式 `11/4` は現状では `2.75` ではなく `2` になります。これは 切り捨て除算 *(floor division)* とも呼ばれます。二つの整数間で除算を行うと、結果は (端数切捨て関数が適用されて) 常に整数になります。しかし、被演算子の一方が (`float` のような) 別の数値型の場合、演算の結果は共通の型に型強制されます (型強制 (*coercion*) 参照)。例えば、浮動小数点数で整数を除算すると結果は浮動小数点になり、場合によっては端数部分を伴います。`//` 演算子を `/` の代わりに使うと、整数除算を強制できます。*__future__* も参照してください。

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、新しいあ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 (*interactive*) も参照してください。

iterable (反復可能オブジェクト) 要素を一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 `file` といった幾つかの非シーケンス型、あるいは `__iter__()` か `__getitem__()` メソッドを実装した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。反復可能オブジェクトを使う際には、通常 `iter()` を呼んだり、イテレータオブジェクトを自分で扱う必要はありません。`for` 文ではこの操作を自動的に行い、無名の変数を作成してループの間イテレータを記憶します。イテレータ (*iterator*)

シーケンス (*sequence*), およびジェネレータ (*generator*) も参照してください。

iterator 一連のデータ列 (stream) 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と、データ列中の要素を一つずつ返します。後続のデータがなくなると、データの代わりに `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全てのオブジェクトを出し尽くしており、それ以降は `next()` を何度呼んでも `StopIteration` を送出します。イテレータは、そのイテレータオブジェクト自体を返す `__iter__()` メソッドを実装しなければならなくなっており、そのため全てのイテレータは他の反復可能オブジェクトを受理できるほとんどの場所で利用できます。著しい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では、`iter()` 関数にオブジェクトを渡したり、`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。このイテレータをさらに別の場所でイテレータとして使おうとすると、前回のイテレーションパスで使用された同じイテレータオブジェクトを返すため、空のコンテナのように見えます。

より詳細な情報は *typeiter* にあります。

keyword argument (キーワード引数) 呼び出し時に、`variable_name=` が手前にある引数。変数名は、その値が関数内のどのローカル変数に渡されるかを指定します。キーワード引数として辞書を受け取ったり渡したりするために `**` を使うことができます。 *argument* も参照してください。

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 (*expression*) を持ちます。ラムダ関数を作る構文は、`lambda [arguments]: 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、:keyword:*if* 文がたくさん使われるのが特徴的です。

list (リスト) Python のビルトインのシーケンス型 (*sequence*) 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) シーケンス内の全てあるいは一部の要素を処理して、その結果からなるリストを返す、コンパクトな書き方です。`result = ["0x%02x" % 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。詳細は **PEP 302** を参照してください。

mapping (マップ) 特殊メソッド `__getitem__()` を使って、任意のキーに対する検索をサポートする (`dict` のような) コンテナオブジェクトです。

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 はカスタムのメタクラスを作成できる点が特別です。ほとんどのユーザーに取って、メタクラスは全く必要のないものです。しかし、一部の場面では、メタクラスは強力でエレガントな方法を提供します。たとえば属性アクセスのログを取ったり、スレッドセーフ性を追加したり、オブジェクトの生成を追跡したり、シングルトンを実装するなど、多くの場面で利用されます。

method クラス内で定義された関数。クラス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 (*argument*) として受け取ります (この第一引数は普段 `self` と呼ばれます)。*function* と *nested scope* も参照してください。

mutable (変更可能オブジェクト) 変更可能なオブジェクトは、 `id()` を変えることなく値を変更できます。変更不能 (*immutable*) も参照してください。

named tuple (名前付きタプル) タプルに似ていて、インデックスによりアクセスする要素に名前付き属性としてもアクセス出来るクラス。(例えば、`time.localtime()` はタプルに似たオブジェクトを返し、その *year* には `t[0]` のようなインデックスによるアクセスと、`t.tm_year` のような名前付き要素としてのアクセスが可能です。)

名前付きタプルには、 `time.struct_time` のようなビルトイン型もありますし、通常のクラス定義によって作成することもできます。名前付きタプルを `collections.namedtuple()` ファクトリ関数で作成することもできます。最後の方法で作った名前付きタプルには自動的に、`Employee(name='jones', title='programmer')` のような自己ドキュメント表現 (self-documenting representation) 機能が付いてきます。

namespace (名前空間) 変数を記憶している場所です。名前空間は辞書を用いて実装されています。名前空間には、ローカル、グローバル、組み込み名前空間、そして (メソッド内の) オブジェクトのネストされた名前空間があります。例えば、関数 `__builtin__.open()` と `os.open()` は名前空間で区別されます。名前空間はまた、ある関数をどのモジュールが実装しているかをはっきりさせることで、可読性やメンテナンス性に寄与します。例えば、`random.seed()`,`itertools.izip()` と書くことで、これらの関数がそれぞれ `random` モジュールや `itertools` モジュールで実装されていることがはっきりします。

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。具体的に言えば、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。変数の代入は、常に最も内側のスコープにある変数に対する書き込みになります。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。

new-style class (新スタイルクラス) `object` から継承したクラス全てを指します。これ

には list や dict のような全ての組み込み型が含まれます。 __slots__(), デスクリプタ、プロパティ、__getattribute__() といった、Python の新しい機能を使えるのは新スタイルクラスだけです。

より詳しい情報は *newstyle* を参照してください。

object 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 (*new-style class*) の基底クラスのこと。

positional argument (位置指定引数) 引数のうち、呼び出すときの順序で、関数やメソッドの中のどの名前に代入されるかが決定されるもの。複数の位置指定引数を、関数定義側が受け取ったり、渡したりするために、* を使うことができます。*argument* も参照してください。

Python 3000 Python の次のメジャーバージョンである Python 3.0 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繋がる、考え方やコード。例えば、Python の一般的なイディオムに iterable の要素を for 文を使って巡回することです。この仕組を持たない言語も多くあるので、Python に慣れ親しんでいない人は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]
```

これと対照的な、よりきれいな Pythonic な方法はこうなります。

```
for piece in food:
    print piece
```

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。

__slots__ 新スタイルクラス (*new-style class*) 内で、インスタンス属性の記憶に必要な領域をあらかじめ定義しておき、それとひきかえにインスタンス辞書を排除してメモリの節約を行うための宣言です。これはよく使われるテクニックですが、正しく動作させるのには少々手際を要するので、例えばメモリが死活問題となるようなアプリケーション内にインスタンスが大量に存在するといった稀なケースを除き、使わないのがベストです。

sequence (シーケンス) 特殊メソッド __getitem__() で整数インデックスによる効率的な要素へのアクセスをサポートし、len() で長さを返すような反復可能オブジェクト (*iterable*) です。組み込みシーケンス型には、 list, str, tuple, unicode

などがあります。dict は __getitem__() と __len__() もサポートしますが、検索の際に任意の変更不能 (*immutable*) なキーを使うため、シーケンスではなくマップ (mapping) とみなされているので注意してください。

slice (スライス) 多くの場合、シーケンス (*sequence*) の一部を含むオブジェクト。スライスは、添字記号 [] で数字の間にコロンを書いたときに作られます。例えば、 variable_name[1:3:5] です。添字記号は slice オブジェクトを内部で利用しています。(もしくは、古いバージョンの、 __getslice__() と __setslice__() を利用します。)

special method (特殊メソッド) ある型に対する特定の動作をするために、Python から暗黙的に呼ばれるメソッド。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 つを持ちます。特殊メソッドについては *specialnames* で解説されています。

statement (文) 文は一種のコードブロックです。文は *expression* か、それ以外のキーワードにより構成されます。例えば if, while, print は文です。

triple-quoted string (三重クォート文字列) 3 つの連続したクォート記号 (") かアポストロフィー (') で囲まれた文字列。通常の (一重) クォート文字列に比べて表現できる文字列に違いはありませんが、幾つかの理由で有用です。1 つか 2 つの連続したクォート記号をエスケープ無しに書くことができますし、行継続文字 (\) を使わなくても複数行にまたがることができるので、ドキュメンテーション文字列を書く時に特に便利です。

type (型) Python のオブジェクトの型は、そのオブジェクトの種類を決定します。全ての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__ 属性からアクセスしたり、type(obj) で取得することができます。

virtual machine (仮想マシン) 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 (*bytecode*) 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、 *Sphinx* を利用して、 reStructuredText から生成されました。

このドキュメントのオンライン版では、コメントや変更の提案を、ドキュメントのページから直接投稿することができます。

ドキュメントとそのツール群の開発は、 docs@python.org メーリングリスト上で行われています。私たちは常に、一緒にドキュメントの開発をしてくれるボランティアを探しています。気軽にこのメーリングリストにメールしてください。

多大な感謝を:

- Fred L. Drake, Jr., the creator of the original Python documentation toolset and writer of much of the content;
- the Docutils project for creating reStructuredText and the Docutils suite;
- Fredrik Lundh for his Alternative Python Reference project from which Sphinx got many good ideas.

Python 自体のバグ報告については、 *reporting-bugs* を参照してください。

B.1 Python ドキュメント 貢献者

この節では、Python ドキュメントに何らかの形で貢献した人をリストアップしています。このリストは完全ではありません – もし、このリストに載っているべき人を知っていたら、docs@python.org にメールで教えてください。私たちは喜んでその問題を修正します。

Aahz, Michael Abbott, Steve Alexander, Jim Ahlstrom, Fred Allen, A. Amoroso, Pehr Anderson, Oliver Andrich, Jesús Cea Avión, Daniel Barclay, Chris Barker, Don Bashford, Anthony Baxter, Alexander Belopolsky, Bennett Benson, Jonathan Black, Robin Boerdijk, Michal Bozon, Aaron Brancotti, Georg Brandl, Keith Briggs, Ian Bruntlett, Lee Busby, Lorenzo M.

Catucci, Carl Cerecke, Mauro Cicognini, Gilles Civario, Mike Clarkson, Steve Clift, Dave Cole, Matthew Cowles, Jeremy Craven, Andrew Dalke, Ben Darnell, L. Peter Deutsch, Robert Donohue, Fred L. Drake, Jr., Josip Dzolonga, Jeff Epler, Michael Ernst, Blame Andy Eskilsson, Carey Evans, Martijn Faassen, Carl Feynman, Dan Finnie, Hernán Martínez Foffani, Stefan Franke, Jim Fulton, Peter Funk, Lele Gaifax, Matthew Gallagher, Ben Gertzfield, Nadim Ghaznavi, Jonathan Giddy, Shelley Gooch, Nathaniel Gray, Grant Griffin, Thomas Guettler, Anders Hammarquist, Mark Hammond, Harald Hanche-Olsen, Manus Hand, Gerhard Häring, Travis B. Hartwell, Tim Hatch, Janko Hauser, Thomas Heller, Bernhard Herzog, Magnus L. Hetland, Konrad Hinsen, Stefan Hoffmeister, Albert Hofkamp, Gregor Hoffleit, Steve Holden, Thomas Holenstein, Gerrit Holl, Rob Hooft, Brian Hooper, Randall Hopper, Michael Hudson, Eric Huss, Jeremy Hylton, Roger Irwin, Jack Jansen, Philip H. Jensen, Pedro Diaz Jimenez, Kent Johnson, Lucas de Jonge, Andreas Jung, Robert Kern, Jim Kerr, Jan Kim, Greg Kochanski, Guido Kollerie, Peter A. Koren, Daniel Kozan, Andrew M. Kuchling, Dave Kuhlman, Erno Kuusela, Thomas Lamb, Detlef Lannert, Piers Lauder, Glyph Lefkowitz, Robert Lehmann, Marc-André Lemburg, Ross Light, Ulf A. Lindgren, Everett Lipman, Mirko Liss, Martin von Löwis, Fredrik Lundh, Jeff MacDonald, John Machin, Andrew MacIntyre, Vladimir Marangozov, Vincent Marchetti, Laura Matson, Daniel May, Rebecca McCreary, Doug Mennella, Paolo Milani, Skip Montanaro, Paul Moore, Ross Moore, Sjoerd Mullender, Dale Nagata, Ng Pheng Siong, Koray Oner, Tomas Oppelstrup, Denis S. Otkidach, Zooko O'Whielacronx, Shriphani Palakodety, William Park, Joonas Paalasmaa, Harri Pasanen, Bo Peng, Tim Peters, Benjamin Peterson, Christopher Petrilli, Justin D. Pettit, Chris Phoenix, François Pinard, Paul Prescod, Eric S. Raymond, Edward K. Ream, Sean Reifschneider, Bernhard Reiter, Armin Rigo, Wes Rishel, Armin Ronacher, Jim Roskind, Guido van Rossum, Donald Wallace Rouse II, Mark Russell, Nick Russo, Chris Ryland, Constantina S., Hugh Sasse, Bob Savage, Scott Schram, Neil Schemenauer, Barry Scott, Joakim Sernbrant, Justin Sheehy, Charlie Shepherd, Michael Simcich, Ionel Simionescu, Michael Sloan, Gregory P. Smith, Roy Smith, Clay Spence, Nicholas Spies, Tage Stabell-Kulo, Frank Stajano, Anthony Starks, Greg Stein, Peter Stoehr, Mark Summerfield, Reuben Sumner, Kalle Svensson, Jim Tittsler, Ville Vainio, Martijn Vries, Charles G. Waldman, Greg Ward, Barry Warsaw, Corran Webster, Glyn Webster, Bob Weiner, Eddy Welbourne, Jeff Wheeler, Mats Wichmann, Gerry Wiener, Timothy Wild, Collin Winter, Blake Winton, Dan Wolfe, Steven Work, Thomas Wouters, Ka-Ping Yee, Rory Yorke, Moshe Zadka, Milan Zamazal, Cheng Zhang.

Python がこの素晴らしいドキュメントを持っているのは、Python コミュニティによる情報提供と貢献のおかげです。 – ありがとう！

付 録 C

History and License

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, http://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, http://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, http://www.zope.com/ 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、http://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 http://www.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	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

総索引

表 C.1 – 前のページからの続き

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
2.2	2.1.1	2001	PSF	yes
2.1.2	2.1.1	2002	PSF	yes
2.1.3	2.1.2	2002	PSF	yes
2.2.1	2.2	2002	PSF	yes
2.2.2	2.2.1	2002	PSF	yes
2.2.3	2.2.2	2002-2003	PSF	yes
2.3	2.2.2	2002-2003	PSF	yes
2.3.1	2.3	2002-2003	PSF	yes
2.3.2	2.3.1	2003	PSF	yes
2.3.3	2.3.2	2003	PSF	yes
2.3.4	2.3.3	2004	PSF	yes
2.3.5	2.3.4	2005	PSF	yes
2.4	2.3	2004	PSF	yes
2.4.1	2.4	2005	PSF	yes
2.4.2	2.4.1	2005	PSF	yes
2.4.3	2.4.2	2006	PSF	yes
2.4.4	2.4.3	2006	PSF	yes
2.5	2.4	2006	PSF	yes
2.5.1	2.5	2007	PSF	yes
2.5.2	2.5.1	2008	PSF	yes
2.5.3	2.5.2	2008	PSF	yes
2.6	2.5	2008	PSF	yes
2.6.1	2.6	2008	PSF	yes

ノート: 「GPL 互換」という表現は、Python が GPL で配布されているという意味ではありません。Python のライセンスは全て、 GPL と違い、変更したバージョンを配布する際に変更をオープンソースにしなくてもかまいません。 GPL 互換のライセンスの下では、GPL でリリースされている他のソフトウェアと Python を組み合わせられますが、それ以外のライセンスではそうではありません。

Guido の指示の下、これらのリリースを可能にしてくださった多くのボランティアのみなさんに感謝します。

C.2 Terms and conditions for accessing or otherwise using Python

PSF LICENSE AGREEMENT FOR PYTHON 2.6.2

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 2.6.2 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 2.6.2 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001-2009 Python Software Foundation; All Rights Reserved" are retained in Python 2.6.2 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 2.6.2 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 2.6.2.

4. PSF is making Python 2.6.2 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 2.6.2 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 2.6.2 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 2.6.2, 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 2.6.2, Licensee agrees to be bound by the terms and conditions of this License Agreement.

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 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.

CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

1. This LICENSE AGREEMENT is between the Corporation for National Research Initia-

tives, 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 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.

ACCEPT

CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

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.keio.ac.jp/matumoto/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.keio.ac.jp/matumoto/emt.html
email: matumoto@math.keio.ac.jp
```

C.3.2 Sockets

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

C.3.3 Floating point exception control

The source for the `fpectl` module includes the following notice:

```
  ---------------------------------------------------------------------
 /                      Copyright (c) 1996.                            \
|          The Regents of the University of California.                 |
|                        All rights reserved.                           |
|                                                                       |
|   Permission to use, copy, modify, and distribute this software for   |
|   any purpose without fee is hereby granted, provided that this en-   |
|   tire 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  work was produced at the University of California, Lawrence   |
```

```
|   Livermore National Laboratory under  contract  no.  W-7405-ENG-48   |
|   between  the  U.S.  Department  of  Energy and The Regents of the   |
|   University of California for the operation of UC LLNL.              |
|                                                                       |
|                              DISCLAIMER                               |
|                                                                       |
|   This  software was prepared as an account of work sponsored by an   |
|   agency of the United States Government. Neither the United States   |
|   Government  nor the University of California nor any of their em-   |
|   ployees, makes any warranty, express or implied, or  assumes  any   |
|   liability  or  responsibility  for the accuracy, completeness, or   |
|   usefulness of any information,  apparatus,  product,  or  process   |
|   disclosed,  or  represents  that  its  use  would  not  infringe    |
|   privately-owned rights. Reference herein to any specific  commer-   |
|   cial  products,  process,  or  service  by trade name, trademark,   |
|   manufacturer, or otherwise, does not  necessarily  constitute  or   |
|   imply  its endorsement, recommendation, or favoring by the United   |
|   States Government or the University of California. The views  and   |
|   opinions  of authors expressed herein do not necessarily state or   |
|   reflect those of the United States Government or  the  University   |
|   of  California,  and shall not be used for advertising or product   |
 \  endorsement purposes.                                              /
  ------------------------------------------------------------------------
```

C.3.4 MD5 message digest algorithm

The source code for the `md5` module contains the following notice:

```
Copyright (C) 1999, 2002 Aladdin Enterprises.  All rights reserved.

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

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

1. The origin of this software must not be misrepresented; you must not
   claim that you wrote the original software. If you use this software
   in a product, an acknowledgment in the product documentation would be
   appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be
   misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

L. Peter Deutsch
ghost@aladdin.com
```

```
Independent implementation of MD5 (RFC 1321).

This code implements the MD5 Algorithm defined in RFC 1321, whose
text is available at
      http://www.ietf.org/rfc/rfc1321.txt
The code is derived from the text of the RFC, including the test suite
(section A.5) but excluding the rest of Appendix A.  It does not include
any code or documentation that is identified in the RFC as being
copyrighted.

The original and principal author of md5.h is L. Peter Deutsch
<ghost@aladdin.com>.  Other authors are noted in the change history
that follows (in reverse chronological order):

2002-04-13 lpd Removed support for non-ANSI compilers; removed
      references to Ghostscript; clarified derivation from RFC 1321;
      now handles byte order either statically or dynamically.
1999-11-04 lpd Edited comments slightly for automatic TOC extraction.
1999-10-18 lpd Fixed typo in header comment (ansi2knr rather than md5);
      added conditionalization for C++ compilation from Martin
      Purschke <purschke@bnl.gov>.
1999-05-03 lpd Original version.
```

C.3.5 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
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.3.6 Cookie management

The Cookie 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.3.7 Profiling

The profile and pstats modules contain the following notice:

```
Copyright 1994, by InfoSeek Corporation, all rights reserved.
Written by James Roskind

Permission to use, copy, modify, and distribute this Python software
and its associated documentation for any purpose (subject to the
restriction in the following sentence) 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 InfoSeek not be used in
advertising or publicity pertaining to distribution of the software
without specific, written prior permission.  This permission is
explicitly restricted to the copying and modification of the software
to remain in Python, compiled Python, or other languages (such as C)
wherein the modified or derived code is exclusively imported into a
Python module.

INFOSEEK CORPORATION DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
```

```
FITNESS. IN NO EVENT SHALL INFOSEEK CORPORATION 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 Execution tracing

The `trace` module contains the following notice:

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

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

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

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

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


Permission to use, copy, modify, and distribute this Python software and
its associated documentation for any purpose without fee is hereby
granted, provided that the above copyright notice appears in all copies,
and that both that copyright notice and this permission notice appear in
supporting documentation, and that the name of neither Automatrix,
Bioreason or Mojam Media be used in advertising or publicity pertaining to
distribution of the software without specific, written prior permission.
```

C.3.9 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.3.10 XML Remote Procedure Calls

The `xmlrpclib` 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.3.11 test_epoll

The `test_epoll` contains the following notice:

```
Copyright (c) 2001-2006 Twisted Matrix Laboratories.

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

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

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

C.3.12 Select kqueue

The `select` and contains the following notice for the kqueue interface:

```
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.

付 録 D

Copyright

Python and this documentation is:

ライセンスおよび許諾に関する完全な情報は、 *History and License* を参照してください。

索引

–help
　コマンドラインオプション, 5
–version
　コマンドラインオプション, 6
-3
　コマンドラインオプション, 9
-B
　コマンドラインオプション, 6
-c <command>
　コマンドラインオプション, 4
-d
　コマンドラインオプション, 6
-E
　コマンドラインオプション, 6
-h
　コマンドラインオプション, 5
-i
　コマンドラインオプション, 6
-J
　コマンドラインオプション, 9
-m <module-name>
　コマンドラインオプション, 4
-O
　コマンドラインオプション, 6
-OO
　コマンドラインオプション, 6
-Q <arg>
　コマンドラインオプション, 6
-S
　コマンドラインオプション, 7
　コマンドラインオプション, 7
-t
　コマンドラインオプション, 7
-U
　コマンドラインオプション, 9
　コマンドラインオプション, 7
-V
　コマンドラインオプション, 6
　コマンドラインオプション, 7
-W arg
　コマンドラインオプション, 7
-X
　コマンドラインオプション, 9
　コマンドラインオプション, 9
..., 33
%PATH%, 19
__future__, 36
__slots__, 41
>>>, 33
2to3, 33

abstract base class, 33
argument, 33
attribute, 33

BDFL, 34
bytecode, 34

class, 34
classic class, 34
coercion, 34
complex number, 34

context manager, 34
CPython, 35

decorator, 35
descriptor, 35
dictionary, 35
docstring, 35
duck-typing, 35

EAFP, 36
exec_prefix, 14
expression, 36
extension module, 36

finder, 36
function, 36

garbage collection, 36
generator, 36
generator expression, 37
GIL, 37
global interpreter lock, 37

hashable, 37

IDLE, 37
immutable, 37
importer, 38
integer division, 38
interactive, 38
interpreted, 38
iterable, 38
iterator, 39

keyword argument, 39

lambda, 39
LBYL, 39
list, 39
list comprehension, 39
loader, 39

mapping, 39
metaclass, 39
method, 40
mutable, 40

named tuple, 40
namespace, 40
nested scope, 40
new-style class, 40

object, 41

PATH, 10, 15
positional argument, 41
prefix, 14
Python 3000, 41
Python Enhancement Proposals
    PEP 11, 17
    PEP 230, 8
    PEP 238, 7
    PEP 302, 36, 39
    PEP 338, 5
    PEP 343, 34
    PEP 370, 7, 11
PYTHON*, 6
PYTHONDEBUG, 6
PYTHONDONTWRITEBYTECODE, 6
PYTHONHOME, 6, 10
Pythonic, 41
PYTHONINSPECT, 6
PYTHONOPTIMIZE, 6
PYTHONPATH, 6, 10, 19, 25
PYTHONSTARTUP, 6
PYTHONUNBUFFERED, 7
PYTHONVERBOSE, 7

reference count, 41

sequence, 41
slice, 42
special method, 42
statement, 42

triple-quoted string, 42
type, 42

virtual machine, 42

Zen of Python, 42

コマンドラインオプション

–help, 5
–version, 6
-3, 9
-B, 6
-c <command>, 4
-d, 6
-E, 6
-h, 5
-i, 6
-J, 9
-m <module-name>, 4
-O, 6
-OO, 6
-Q <arg>, 6
-S, 7
-s, 7
-t, 7
-U, 9
-u, 7
-V, 6
-v, 7
-W arg, 7
-X, 9
-x, 9

環境変数
%PATH%, 19
exec_prefix, 14
PATH, 10, 15
prefix, 14
PYTHON*, 6
PYTHONCASEOK, 11
PYTHONDEBUG, 6, 11
PYTHONDONTWRITEBYTECODE, 6, 11
PYTHONDUMPREFS, 12
PYTHONEXECUTABLE, 12
PYTHONHOME, 6, 10
PYTHONINSPECT, 6, 11
PYTHONIOENCODING, 11
PYTHONMALLOCSTATS, 12
PYTHONNOUSERSITE, 11
PYTHONOPTIMIZE, 6, 10
PYTHONPATH, 6, 10, 19, 25
PYTHONSTARTUP, 6, 10
PYTHONTHREADDEBUG, 12
PYTHONUNBUFFERED, 7, 11
PYTHONUSERBASE, 11
PYTHONVERBOSE, 7, 11
PYTHONY2K, 10