Installing Python Modules

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

Author Greg Ward — 日本語訳: Python ドキュメント翻訳プロジェクト

Release 2.6

Date 2011 年 01 月 23 日

Abstract

このドキュメントでは、 Python モジュール配布ユーティリティ (Python Distribution Utilities, “Distutils”) について、 エンドユーザの視点に立ち、サードパーティ製のモジュールや拡張モジュールの構築やインストールによって標準の Python に機能を追加する方法について述べます。

第 1

はじめに

Python の広範な標準ライブラリは、プログラミングにおける多くの要求をカバーしていますが、時には何らかの新たな機能をサードパーティ製モジュールの形で追加する必要に迫られます。自分がプログラムを書くときのサポートとして必要な場合もあるし、自分が使いたいアプリケーションがたまたま Python で書かれていて、そのサポートとして必要な場合もあるでしょう。

以前は、すでにインストール済みの Python に対して、サードパーティ製モジュールを追加するためのサポートはほとんどありませんでした。しかし Python 配布ユーティリティ (Python Distribution Utilities, 略して Distutils) が Python 2.0 から取り入れられ、状況は変わりました。

このドキュメントが主要な対象としているのは、サードパーティモジュールをインストールする必要がある人たち: 単に何らかの Python アプリケーションを稼動させたいだけのエンドユーザやシステム管理者、そしてすでに Python プログラマであって、新たな道具を自分のツールキットに加えたいと思っている人たちです。このドキュメントを読むために、 Python について知っておく必要はありません; インストールしたモジュールを調べるために Python の対話モードにちょっとだけ手を出す必要がありますが、それだけです。自作の Python モジュールを他人が使えるようにするために配布する方法を探しているのなら、 *distutils-index* マニュアルを参照してください。

1.1 もっとも簡単な場合: ありふれたインストール作業

最も楽なのは、インストールしたいモジュール配布物の特殊なバージョンをインストールしたいプラットフォーム向けに誰かがすでに用意してくれていて、他のアプリケーションと同じようにインストールするだけであるような場合です。例えば Windows ユーザ向けには実行可能形式のインストーラ、 RPM ベースの Linux システム (Red Hat, SuSE, Mandrake その他多数) 向けには RPM パッケージ、 Debian ベースの Linux システム向け

には Debian パッケージといった具合に、モジュール開発者はビルド済み配布物を作成しているかもしれません。

このような場合、自分のプラットフォームに合ったインストーラをダウンロードして、実行可能形式なら実行し、RPM なら `rpm --install` するといった、分かりきった作業をするだけです。Python を起動したり、setup スクリプトを実行する必要はなく、何もコンパイルする必要はありません— 説明書きを読む必要すら全くないかもしれません (とはいえ、説明書きを読むのはよいことですが)。

もちろん、いつもこう簡単とは限りません。自分のプラットフォーム向けのお手軽なインストーラがないモジュール配布物に興味を持つこともあるでしょう。そんな場合には、モジュールの作者やメンテナがリリースしているソース配布物から作業をはじめねばなりません。ソース配布物からのインストールは、モジュールが標準的な方法でパッケージ化されている限りさほど大変ではありません。このドキュメントの大部分は、標準的なソース配布物からのビルドとインストールに関するものです。

1.2 新しい標準: Distutils

モジュールのソースコード配布物をダウンロードしたら、配布物が標準のやり方、すなわち Distutils のやり方に従ってパッケージされて配布されているかどうかすぐに分かります。Distutils の場合、まず配布物の名前とバージョン番号が、例えば `foo-1.0.tar.gz` や `widget-0.9.7.zip` のように、ダウンロードされたアーカイブファイルの名前にはっきりと反映されます。次に、アーカイブは同様の名前のディレクトリ、例えば `foo-1.0` や `widget-0.9.7` に展開されます。さらに、配布物には setup スクリプト `setup.py` が入っています。また、 `README.txt` 場合によっては `README` という名前のファイルも入っていて、そこには、モジュール配布物の構築とインストールは簡単で、

```
python setup.py install
```

とするだけだ、という説明が書かれているはずです。

上記の全てが当てはまるなら、ダウンロードしたものをビルドしてインストールする方法はすでに知っていることになります: 上記のコマンドを実行するだけです。非標準の方法でインストールを行ったり、ビルドプロセスをカスタマイズ行いたいのでない限り、このマニュアルは必要ありません。別の言葉で言えば、上のコマンドこそが、このマニュアルから習得すべき全てということになります。

第 2

標準的なビルド・インストール作業

新しい標準: *Distutils* 節で述べたよいうに、 Distutils を使ったモジュール配布物のビルドとインストールは、通常は単純なコマンド:

```
python setup.py install
```

で行います。

Unix では、このコマンドをシェルプロンプトで行います; Windows では、コマンドプロンプトウィンドウ (“DOS ボックス”) を開いて、そこで行います; Mac OS X の場合、**Terminal** ウィンドウを開いてシェルプロンプトを出してください。

2.1 プラットフォームによる違い

setup コマンドは常に配布物ルートディレクトリ、すなわちモジュールのソース配布物を展開した際のトップレベルのサブディレクトリ内で実行しなければなりません。例えば、あるモジュールのソース配布物 `foo-1.0.tar.gz` を Unix システム上にダウンロードしたなら、通常は以下の操作を行います:

```
gunzip -c foo-1.0.tar.gz | tar xf -     # unpacks into directory foo-1.0
cd foo-1.0
python setup.py install
```

Windows では、おそらく `foo-1.0.zip` をダウンロードしているでしょう。アーカイブファイルを `C:\Temp` にダウンロードしたのなら、(WinZip のような) グラフィカルユーザインタフェースつきのアーカイブ操作ソフトや、 (**unzip** や **pkunzip** のような) コマンドラインツールを使ってアーカイブを展開します。次に、コマンドプロンプトウィンドウ (“DOS ボックス”) を開いて、以下を実行します:

```
cd c:\Temp\foo-1.0
python setup.py install
```

2.2 ビルド作業とインストール作業を分割する

`setup.py install` を実行すると、一度の実行で全てのモジュールをビルドしてインストールします。段階的に作業をしたい場合 — ビルドプロセスをカスタマイズしたり、作業がうまくいかない場合に特に便利です — には、setup スクリプトに一度に一つづつ作業を行わせるようにできます。この機能は、ビルドとインストールを異なるユーザで行う場合にも便利です — 例えば、モジュール配布物をビルドしておいてシステム管理者に渡して (または、自分でスーパユーザになって) 、インストールしたくなるかもしれません.

最初のステップでは全てをビルドしておき、次のステップで全てをインストールするには、 setup スクリプトを二度起動します:

```
python setup.py build
python setup.py install
```

この作業を行ってみれば、 **install** コマンドを実行するとまず **build** コマンドを実行し、さらに — この場合には — `build` ディレクトリの中が全て最新の状態になっているので、**build** は何も行わなくてよいと判断していることがわかるでしょう。

インターネットからダウンロードしたモジュールをインストールしたいだけなら、上のように作業を分割する機能は必要ないかもしれませんが、この機能はより進んだ使い方をする際にはとても便利です。自作の Python モジュールや拡張モジュールを配布することになれば、個々の Distutils コマンドを自分で何度も実行することになるでしょう。

2.3 ビルドの仕組み

上で示唆したように、 **build** コマンドは、インストールすべきファイルを ビルドディレクトリ *(build directory)* に置く働きがあります。デフォルトでは、ビルドディレクトリは配布物ルート下の `build` になります; システムの処理速度に強いこだわりがあったり、ソースツリーに指一本触れたくないのなら、 *`--build-base`* オプションを使ってビルドディレクトリを変更できます。例えば:

```
python setup.py build --build-base=/tmp/pybuild/foo-1.0
```

(あるいは、システム全体向け、あるいは個人用の Distutils 設定ファイルにディレクティブを書いて、永続的に設定を変えられます; *Distutils* 設定ファイル 節を参照してください。) 通常は必要ない作業です。

ビルドツリーのデフォルトのレイアウトは以下のようになっています:

```
--- build/ --- lib/
または
--- build/ --- lib.<plat>/
               temp.<plat>/
```

<plat> は、現在の OS/ハードウェアプラットフォームと Python のバージョンを記述する短い文字列に展開されます。第一の lib ディレクトリだけの形式は、 “pure モジュール配布物” — すなわち、pure Python モジュールだけの入ったモジュール配布物 — の場合に使われます。モジュール配布物に何らかの拡張モジュール (C/C++ で書かれたモジュール) が入っている場合、第二の <plat> 付きディレクトリが二つある形式が使われます。この場合、 temp.plat ディレクトリには、コンパイル/リンク過程で生成され、実際にはインストールされない一時ファイルが収められます。どちらの場合にも、 lib (または lib.plat) ディレクトリには、最終的にインストールされることになる全ての Python モジュール (pure Python モジュールおよび拡張モジュール) が入ります。

今後、Python スクリプト、ドキュメント、バイナリ実行可能形式、その他 Python モジュールやアプリケーションのインストール作業に必要なディレクトリが追加されるかもしれません。

2.4 インストールの仕組み

build コマンドを実行した後 (明示的に実行した場合も、 **install** コマンドが代わりに実行してくれた場合も) は、 **install** コマンドの仕事は比較的単純なもの: build/lib (または build/lib.plat) の下にあるもの全ての指定したインストールディレクトリへのコピー、になります。

インストールディレクトリを選ばなかった場合 — すなわち、 setup.py install を実行しただけの場合 — には、 **install** コマンドはサードパーティ製 Python モジュールを置くための標準の場所にインストールを行います。この場所は、プラットフォームや、Python 自体をどのようにビルド/インストールしたかで変わります。 Unix (と、Unix をベースとした Mac OS X) では、インストールしようとするモジュール配布物が pure Python なのか、拡張モジュールを含む (“非 pure”) のかによっても異なります:

プラットフォーム	標準のインストール場所	デフォルト値	注記
Unix (pure)	prefix/lib/pythonX.Y/site-packages	/usr/local/lib/pythonX.Y/site-packages	(1)
Unix (non-pure)	exec-prefix/lib/pythonX.Y/site-packages	/usr/local/lib/pythonX.Y/site-packages	(1)
Windows	prefix	C:\Python	(2)

注記:

1. ほとんどの Linux ディストリビューションには、システムの標準インストール物として Python が入っているので、 Linux では普通、`prefix` や `exec-prefix` はどちらも `/usr` になります。Linux (または Unix ライクなシステム) 上で自分で Python をビルドした場合、デフォルトの `prefix` および `exec-prefix` は `/usr/local` になります。

2. Windows での Python のデフォルトインストールディレクトリは、 Python 1.6a1、1.5.2、およびそれ以前のバージョンでは `C:\Program Files\Python` です。

`prefix` および `exec-prefix` は、Python がインストールされているディレクトリと、実行時にライブラリを探しにいく場所を表します。これらのディレクトリは、Windows では常に同じで、 Unix と Mac OS X でもほぼ常に同じです。自分の Python がどんな `prefix` や `exec-prefix` を使っているかは、Python を対話モードで起動して、単純なコマンドをいくつか入力すればわかります。 Windows では、:menuselection:スタート –> *(*すべての*)* プログラム –> *Python X.Y* –> *Python (command line)* を選びます。 Mac OS 9 では、`PythonInterpreter` を起動します。インタプリタを起動すると、プロンプトに Python コードを入力できます。例えば、作者の使っている Linux システムで、三つの Python 文を以下のように入力すると、出力から作者のシステムの `prefix` と `exec-prefix` を得られます:

```
Python 2.4 (#26, Aug  7 2004, 17:19:02)
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.prefix
'/usr'
>>> sys.exec_prefix
'/usr'
```

モジュールを標準の場所にインストールしたくない場合や、標準の場所にインストールするためのファイル権限を持っていない場合、別の場所へのインストール 節にある、別の場所へのインストール方法の説明を読んでください。インストール先のディレクトリを大幅にカスタマイズしければ、カスタムのインストール 節のカスタムインストールに関する説明を読んでください。

第 3

別の場所へのインストール

しばしば、サードパーティ製 Python モジュールをインストールするための標準の場所以外にモジュールをインストールしなければならなかったり、単にそうしたくなるときがあります。例えば Unix システムでは、標準のサードパーティ製モジュールディレクトリに対する書き込み権限を持っていないかもしれません。または、あるモジュールを、ローカルで使っている Python に標準のモジュールの一部として組み込む前にテストしてみたいと思うかもしれません。既存の配布物をアップグレードする際には特にそうでしょう: 実際にアップグレードする前に、既存のスクリプトの基本となる部分が新たなバージョンでも動作するか確認したいはずです。

Distutils の **install** コマンドは、別の場所へ配布物をインストールする作業を単純で苦労のない作業にするように設計されています。基本的なアイデアは、インストール先のベースディレクトリを指定しておき、**install** コマンドがそのベースディレクトリ下にファイル群をインストールするための一連のディレクトリ (インストールスキーム *: installation scheme*) を作成するというものです。詳細はプラットフォームによって異なるので、以下の節から自分のプラットフォームに当てはまるものを読んでください。

3.1 別の場所へのインストール: home スキーム

“home スキーム” の背後にある考え方は、Python モジュールを個人用のモジュール置き場でビルドし、維持するというものです。このスキームの名前は Unix の「ホーム」ディレクトリの概念からとりました。というのも、 Unix のユーザにとって、自分のホームディレクトリを `/usr/` や`/usr/local/` のようにレイアウトするのはよくあることだからです。とはいえ、このスキームはどのオペレーティングシステムのユーザでも使えます。新たなモジュールのインストールは単純で、

```
python setup.py install --home=<dir>
```

のようにします。このとき、 *--home* オプションを使ってディレクトリを指定します。面倒臭がりの人は、単にチルダ (~) をタイプするだけでかまいません; **install** コマンドがチルダをホームディレクトリに展開してくれます。

```
python setup.py install --home=~
```

--home オプションは、インストールのベースディレクトリを指定します。ファイルはインストールベース下の以下のディレクトリに保存されます:

Type of file	Installation Directory	Override option
pure module distribution	home/lib/python	*--install-purelib*
non-pure module distribution	home/lib/python	*--install-platlib*
scripts	home/bin	*--install-scripts*
data	home/share	*--install-data*

バージョン 2.4 で変更: *--home* は Unix でしかサポートされていませんでした.

3.2 別の場所へのインストール: Unix (prefix スキーム)

あるインストール済みの Python を使ってモジュールのビルド/インストールを (例えば setup スクリプトを実行して) 行いたいけれども、別のインストール済みの Python のサードパーティ製モジュール置き場 (あるいは、そう見えるようなディレクトリ構造) に、ビルドされたモジュールをインストールしたい場合には、”prefix スキーム” が便利です。そんな作業はまったくありえそうにない、と思うなら、確かにその通りです — “home スキーム” を先に説明したのもそのためです。とはいえ、prefix スキームが有用なケースは少なくとも二つあります。

まず、多くの Linux ディストリビューションは、 Python を /usr/local ではなく /usr に置いていることを考えてください。この場合は、 Python はローカルの計算機ごとのアドオン (add-on) ではなく、” システム” の一部となっているので、 /usr に置くのは全く正当なことです。しかしながら、Python モジュールをソースコードからインストールしていると、モジュールを /usr/lib/python2.X ではなく /usr/local/lib/python2.X に置きたいと思うかもしれません。これを行うには

```
/usr/bin/python setup.py install --prefix=/usr/local
```

と指定します。

もう一つありえるのは、ネットワークファイルシステムにおいて、遠隔のディレクトリに対する読み出しと書き込みの際に違う名前を使う場合です。例えば、 /usr/local/bin/python でアクセスするような Python インタプリタは、 /usr/local/lib/python2.X からモジュールを探すでしょうが、モジュールは別の場所、例えば/mnt/@server/export/lib/python2.X にインストールしなければならないかもしれません。この場合には、

```
/usr/local/bin/python setup.py install --prefix=/mnt/@server/export
```

のようにします。

どちらの場合も、 *--prefix* オプションでインストールベースディレクトリを決め、*--exec-prefix* でプラットフォーム固有のファイル置き場名として使う、プラットフォーム固有インストールベースディレクトリを決めます。(プラットフォーム固有のファイルとは、現状では単に非 pure モジュール配布物のことを意味しますが、 C ライブラリやバイナリ実行可能形式などに拡張されるかもしれません。) *--exec-prefix* が指定されていなければ、デフォルトの *--prefix* になります。ファイルは以下のようにインストールされます:

Type of file	Installation Directory	Override option
pure module distribution	prefix/lib/pythonX.Y/site-packages	*--install-purelib*
non-pure module distribution	exec-prefix/lib/pythonX.Y/site-packages	*--install-platlib*
scripts	prefix/bin	*--install-scripts*
data	prefix/share	*--install-data*

--prefix や *--exec-prefix* が実際に他のインストール済み Python の場所を指している必要はありません; 上に挙げたディレクトリがまだ存在しなければ、インストール時に作成されます。

ちなみに、prefix スキームが重要な本当の理由は、単に標準の Unix インストールが prefix スキームを使っているからです。ただし、そのときには、*--prefix* や *--exec-prefix* は Python 自体が `sys.prefix` `` や `` `sys.exec_prefix` を使って決めます。というわけで、読者は prefix スキームを決して使うことはあるまいと思っているかもしれませんが、 `python setup.py install` をオプションを何もつけずに実行していれば、常に prefix スキームを使っていることになるのです。

拡張モジュールを別のインストール済み Python にインストールしても、拡張モジュールのビルド方法による影響を受けることはありません: 特に、拡張モジュールをコンパイルする際には、 setup スクリプトを実行する際に使う Python インタプリタと一緒にインストールされている Python ヘッダファイル (`Python.h` とその仲間たち) を使います。上で述べてきたやり方でインストールされた拡張モジュールを実行するインタプリタと、インタプリタをビルドする際に用いた別のインタプリタとの互換性を保証するのはユーザの責任です。

これを行うには、二つのインタプリタが同じバージョンの Python (ビルドが違っていたり、同じビルドのコピーということもあり得ます) であるかどうかを確かめます。(もちろん、 *--prefix* や *--exec-prefix* が別のインストール済み Python の場所すら指していなければどうにもなりません。)

3.3 別の場所へのインストール (prefix を使う方法): Windows

Windows はユーザのホームディレクトリという概念がなく、 Windows 環境下で標準的にインストールされた Python は Unix よりも単純な構成をしているので、 Windows で追加のパッケージを別の場所に入れる場合には、伝統的に *--prefix* が使われてきました。

```
python setup.py install --prefix="\Temp\Python"
```

とすると、モジュールを現在のドライブの `\Temp\Python` ディレクトリにインストールします

インストールベースディレクトリは、 *--prefix* オプションだけで決まります; *--exec-prefix* オプションは、Windows ではサポートされていません。ファイルは以下のような構成でインストールされます:

Type of file	Installation Directory	Override option
pure module distribution	`prefix`	*--install-purelib*
non-pure module distribution	`prefix`	*--install-platlib*
scripts	`prefix\Scripts`	*--install-scripts*
data	`prefix\Data`	*--install-data*

第 4

カスタムのインストール

たまに、 別の場所へのインストール 節で述べたような別の場所へのインストールスキームが、自分のやりたいインストール方法と違うことがあります。もしかすると、同じベースディレクトリ下にあるディレクトリのうち、一つか二つだけをいじりたかったり、インストールスキームを完全に再定義したいと思うかもしれません。どちらの場合にせよ、こうした操作では カスタムのインストールスキーム を作成することになります。

別の場所へのインストールスキームに関するこれまでの説明で、“オーバライドするためのオプション” というコラムにお気づきかもしれません。このオプションは、カスタムのインストールスキームを定義するための手段です。各オーバライドオプションには、相対パスを指定しても、絶対パスを指定しても、インストールベースディレクトリのいずれかを明示的に指定してもかまいません。(インストールベースディレクトリは二種類あり、それら二つは通常は同じディレクトリです — Unix の “prefix スキーム” を使っていて、*--prefix* と *--exec-prefix* オプションを使っているときだけ異なります。)

例えば、 Unix 環境でモジュール配布物をホームディレクトリにインストールしたい — とはいえ、スクリプトは `~/bin` ではなく `~/scripts` に置きたい — とします。ご想像の通り、スクリプトを置くディレクトリは、*--install-scripts* オプションで上書きできます; この場合は相対パスで指定もでき、インストールベースディレクトリ (この場合にはホームディレクトリ) からの相対パスとして解釈されます:

```
python setup.py install --home=~ --install-scripts=scripts
```

Unix 環境での例をもう一つ紹介します: インストール済みの Python が、`/usr/local/python` を prefix にしてビルドされ、インストールされていて、標準のインストールスクリプトは `/usr/local/python/bin` に入るようになっているとします。 `/usr/local/bin` に入るようにしたければ、絶対パスを *--install-scripts* オプションに与えて上書きすることになるでしょう:

```
python setup.py install --install-scripts=/usr/local/bin
```

(この操作を行うと、“prefix スキーム” を使ったインストールになり、 prefix は Python イ

ンタプリタがインストールされている場所— この場合には `/usr/local/python` になります。)

Windows 用の Python を管理しているのなら、サードパーティ製モジュールを `prefix` そのものの下ではなく、`prefix` の下にあるサブディレクトリに置きたいと考えるかもしれません。この作業は、インストールディレクトリのカスタマイズとほぼ同じくらい簡単です — 覚えておかねばならないのは、モジュールには二つのタイプ、pure モジュールと非 pure モジュール (非 pure モジュール配布物内のモジュール) があるということです。例えば以下のようにします:

```
python setup.py install --install-purelib=Site --install-platlib=Site
```

指定したインストール先ディレクトリは、`prefix` からの相対です。もちろん、`prefix` を `.pth` ファイルに入れるなどして、これらのディレクトリが Python のモジュール検索パス内に入るようにしなければなりません。Python のモジュール検索パスを修正する方法は、 *Python サーチパスの変更* 節を参照してください。

インストールスキーム全体を定義したいのなら、全てのインストールディレクトリオプションを指定しなければなりません。この作業には、相対パスを使った指定を勧めます; 例えば、全ての Python モジュール関連ファイルをホームディレクトリ下の `python` ディレクトリの下に置き、そのホームディレクトリをマウントしている各プラットフォームごとに別のディレクトリを置きたければ、以下のようにインストールスキームを定義します:

```
python setup.py install --home=~ \
                        --install-purelib=python/lib \
                        --install-platlib=python/lib.$PLAT \
                        --install-scripts=python/scripts
                        --install-data=python/data
```

また、以下のようにも指定できます:

```
python setup.py install --home=~/python \
                        --install-purelib=lib \
                        --install-platlib='lib.$PLAT' \
                        --install-scripts=scripts
                        --install-data=data
```

`$PLAT` は、(必ずしも) 環境変数ではありません — この表記は、Distutils がコマンドラインオプションの解釈と同じやり方で展開します。設定ファイルを解釈する際と同じです。

言うまでもないことですが、毎回新たなモジュール配布物をインストールする度にインストールスキーム全体の指定を行っていては面倒です。そこで、オプションは Distutils 設定ファイル (*Distutils 設定ファイル* 参照) にも指定できます:

```
[install]
install-base=$HOME
install-purelib=python/lib
install-platlib=python/lib.$PLAT
```

```
install-scripts=python/scripts
install-data=python/data
```

あるいは、以下のようにも指定できます:

```
[install]
install-base=$HOME/python
install-purelib=lib
install-platlib=lib.$PLAT
install-scripts=scripts
install-data=data
```

これら二つは、 setup スクリプトを異なるインストールベースディレクトリから実行した場合には同じには ならない ので注意してください。例えば、

```
python setup.py install --install-base=/tmp
```

とすると、最初の書き方では pure モジュールが `/tmp/python/lib` に入り、二番目の書き方では`/tmp/lib` に入ります。(二番目のケースでは、インストールベースを `/tmp/python` に指定しようと考えるでしょう。)

読者は、設定ファイル例で、入力値に `$HOME` や `$PLAT` を使っていることに気づいているかもしれませんね。これらは Distutils の設定変数で、環境変数を彷彿とさせます。実際、この表記が使えるプラットフォーム上では、設定ファイル中に環境変数を入れられますが、 Distutils は他にも、例えば `$PLAT` のようにおそらくユーザの環境中にないような変数をいくつか持っています。(そしてもちろん、 Mac OS 9 のような環境変数のないシステムでは、設定ファイル中で使える変数は Distutils が提供しているものだけです。) 詳細は *Distutils* 設定ファイル を参照してください。

4.1 Python サーチパスの変更

Python インタプリタが `import` 文を実行するとき、インタプリタは Python コードや拡張モジュールをモジュール検索パス中から探します。検索パスのデフォルト値は、インタプリタをビルドする際に Python のバイナリ内に設定されます。検索パスは、 `sys` を import して、 `sys.path` を出力すればわかります。

```
$ python
Python 2.2 (#11, Oct  3 2002, 13:31:27)
[GCC 2.96 20000731 (Red Hat Linux 7.3 2.96-112)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.path
['', '/usr/local/lib/python2.3', '/usr/local/lib/python2.3/plat-linux2',
 '/usr/local/lib/python2.3/lib-tk', '/usr/local/lib/python2.3/lib-dynload',
 '/usr/local/lib/python2.3/site-packages']
>>>
```

sys.path 内の空文字列は、現在の作業ディレクトリを表します。

ローカルでインストールされるパッケージは、 .../site-packages/ ディレクトリに入るのが決まりですが、ユーザはどこか任意のディレクトリに Python モジュールをインストールしたいと思うかもしれません。例えば、自分のサイトでは、 web サーバに関連する全てのソフトウェアを /www に置くという決まりがあるかもしれません。そこで、アドオンの Python モジュールが /www/python 置かれることになると、モジュールを import するためにはディレクトリを sys.path に追加せねばなりません。ディレクトリを検索パスに追加するには、いくつかの異なる方法が存在します。

最も手軽な方法は、パス設定ファイルをすでに Python の検索パスに含まれるディレクトリ、通常は .../site-packages/ ディレクトリに置くというものです。パス設定ファイルは拡張子が .pth で、ファイルには sys.path に追加するパスを一行に一つづつ記述しなければなりません。 (新たなパスは今の sys.path の後ろに追加されるので、追加されたディレクトリ内にあるモジュールが標準のモジュールセットを上書きすることはありません。つまり、このメカニズムを使って、標準モジュールに対する修正版のインストールはできないということです。)

パスは絶対パスでも相対パスでもよく、相対パスの場合には .pth ファイルのあるパスからの相対になります。詳しくは site モジュールを参照してください。

やや便利さには欠けますが、Python の標準ライブラリ中にある site.py ファイルを編集することでも、 sys.path を変更できます。 site.py は、 *-S* スイッチを与えて抑制しないかぎり、Python インタプリタが実行される際に自動的に import されます。ただし、設定するには、単に site.py を編集して、例えば以下のような二行を加えます:

```
import sys
sys.path.append('/www/python/')
```

しかしながら、(例えば 2.2 から 2.2.2 にアップグレードするときのように) 同じメジャーバージョンの Python を再インストールすると、site.py は手持ちのバージョンで上書きされてしまいます。ファイルが変更されていることを覚えておき、インストールを行う前にコピーを忘れずとっておかねばなりません。

また、 sys.path を修正できる二つの環境変数があります。 PYTHONHOME を使うと、インストールされている Python のプレフィクスを別の値に設定できます。例えば、 PYTHONHOME を /www/python に設定すると、検索パスは ['', '/www/python/lib/pythonX.Y/', '/www/python/lib/pythonX.Y/plat-linux2', ...] といった具合になります。

PYTHONPATH を使うと、 sys.path の先頭に一連のパスを追加できます。例えば、 PYTHONPATH を /www/python:/opt/py に設定すると、検索パスは ['/www/python', '/opt/py'] から始まります。 (‘‘ sys.path‘‘ にディレクトリを追加するには、そのディレクトリが実在しなければなりません; site は実在しないディレクトリを除去します。)

最後に、 sys.path はただの普通の Python のリストなので、どんな Python アプリケーションもエントリを追加したり除去したりといった修正を行えます。

第 5

Distutils 設定ファイル

上で述べたように、 Distutils 設定ファイルを使えば、任意の Distutils オプションに対して個人的な設定やサイト全体の設定を記録できます。すなわち、任意のコマンドの任意のオプションを二つか三つ (プラットフォームによって異なります) の設定ファイルに保存でき、コマンドラインを解釈する前にオプションを問い合わせさせるようにできます。つまり、設定ファイルはデフォルトの値を上書きし、さらにコマンドライン上で与えた値が設定ファイルの内容を上書きするわけです。さらに、複数の設定ファイルが適用されると、”先に” 適用されたファイルに指定されていた値は “後に” 適用されたファイル内の値で上書きされます。

5.1 設定ファイルの場所と名前

設定ファイルの名前と場所は、非常にわずかですがプラットフォーム間で異なります。Unix と Mac OS X では、三種類の設定ファイルは以下のようになります (処理される順に並んでいます):

設定ファイルのタイプ	場所とファイル名	注記
system	`prefix/lib/pythonver/distutils/distutils.cfg`	(1)
personal	`$HOME/.pydistutils.cfg`	(2)
local	`setup.cfg`	(3)

Windows では設定ファイルは以下のようになります:

設定ファイルのタイプ	場所とファイル名	注記
system	`prefix\Lib\distutils\distutils.cfg`	(4)
personal	`%HOME%\pydistutils.cfg`	(5)
local	`setup.cfg`	(3)

注記:

1. 厳密に言えば、システム全体向けの設定ファイルは、 Distutils がインストールされているディレクトリになります; Unix の Python 1.6 以降では、表の通りの場所になります。 Python 1.5.2 では、 Distutils は通常 `prefix/lib/python1.5/site-packages/distutils` にインストールされるため、 Python 1.5.2 では設定ファイルをそこに置かなければなりません。

2. Unix では、環境変数 `HOME` が定義されていない場合、標準モジュール `pwd` の `getpwuid()` 関数を使ってユーザのホームディレクトリを決定します。

3. 現在のディレクトリ (通常は setup スクリプトがある場所) です。

4. (注記 (1) も参照してください) Python 1.6 およびそれ以降のバージョンでは、Python のデフォルトの “インストールプレフィクス” は `C:\Python` なので、システム設定ファイルは通常 `C:\Python\Lib\distutils\distutils.cfg` になります。Python 1.5.2 ではデフォルトのプレフィクスは `C:\Program Files\Python` であり、Distutils は標準ライブラリの一部ではありません — 従って、システム設定ファイルは、 Windows 用の標準の Python 1.5.2 では `C:\Program Files\Python\distutils\distutils.cfg` になります。

5. Windows では、環境変数 `HOME` が設定されていない場合、個人用の設定ファイルはどこにもなく、使われることもありません。(言い換えれば、Windows での Distutils はホームディレクトリがどこにあるか一切推測しようとしないということです。)

5.2 設定ファイルの構文

Distutils 設定ファイルは、全て同じ構文をしています。設定ファイルはセクションでグループ分けされています。各 Distutils コマンドごとにセクションがあり、それに加えて全てのコマンドに影響するグローバルオプションを設定するための `global` セクションがあります。各セクションには `option=value` の形で、一行あたり一つのオプションを指定します。

例えば、以下は全てのコマンドに対してデフォルトでメッセージを出さないよう強制するための完全な設定ファイルです:

```
[global]
verbose=0
```

この内容のファイルがシステム全体用の設定ファイルとしてインストールされていれば、そのシステムの全てのユーザによる全ての Python モジュール配布物に対する処理に影響します。ファイルが (個人用の設定をサポートしているシステムで) 個人用の設定ファイルとしてインストールされていれば、そのユーザが処理するモジュール配布物にのみ影響します。この内容を特定のモジュール配布物の `setup.cfg` として使えば、その配布物だけに影響します。

以下のようにして、デフォルトの “ビルドベース” ディレクトリをオーバライドしたり、**build*** コマンドが常に強制的にリビルドを行うようにもできます:

```
[build]
build-base=blib
force=1
```

この設定は、コマンドライン引数の

```
python setup.py build --build-base=blib --force
```

に対応します。ただし、後者ではコマンドライン上で **build** コマンドを含めて、そのコマンドを実行するよう意味しているところが違います。特定のコマンドに対するオプションを設定ファイルに含めると、このような関連付けの必要はなくなります; あるコマンドが実行されると、そのコマンドに対するオプションが適用されます。(また、設定ファイル内からオプションを取得するような他のコマンドを実行した場合、それらのコマンドもまた設定ファイル内の対応するオプションの値を使います。)

あるコマンドに対するオプションの完全なリストは、例えば以下のように、 *--help* を使って調べます:

```
python setup.py build --help
```

グローバルオプションの完全なリストを得るには、コマンドを指定せずに *--help* オプションを使います:

```
python setup.py --help
```

“Python モジュールの配布” マニュアルの、“リファレンスマニュアル” の節も参照してください。

第 6

拡張モジュールのビルド: 小技と豆知識

Distutils は、可能なときにはいつでも、 `setup.py` スクリプトを実行する Python インタプリタが提供する設定情報を使おうとします。例えば、拡張モジュールをコンパイルする際には、コンパイラやリンカのフラグには Python をコンパイルした際と同じものが使われます。通常、この設定はうまくいきますが、状況が複雑になると不適切な設定になることもあります。この節では、通常の Distutils の動作をオーバライドする方法について議論します。

6.1 コンパイラ/リンカのフラグをいじるには

C や C++ で書かれた Python 拡張をコンパイルする際、しばしば特定のライブラリを使ったり、特定の種類のオブジェクトコードを生成したりする上で、コンパイラやリンカに与えるフラグをカスタマイズする必要があります。ある拡張モジュールが自分のプラットフォームではテストされていなかったり、クロスコンパイルを行わねばならない場合にはこれが当てはまります。

最も一般的なケースでは、拡張モジュールの作者はすでに拡張モジュールのコンパイルが複雑になることを見越していて、 `Setup` ファイルを提供して編集できるようにしています。 `Setup` ファイルの編集は、モジュール配布物に多くの個別の拡張モジュールがあったり、コンパイラに拡張モジュールをコンパイルさせるために細かくフラグをセットする必要があるような場合にのみ行うことになるでしょう。

`Setup` ファイルが存在する場合、ビルドするべき拡張モジュールのリストを得るために解釈されます。 `Setup` ファイルの各行には単一のモジュールを書きます。各行は以下のような構造をとります:

```
module ... [sourcefile ...] [cpparg ...] [library ...]
```

次に、各フィールドについて見てみましょう。

- *module* はビルドする拡張モジュールの名前で、Python の識別子名として有効でなければなりません。モジュールの名前変更は、このフィールドを変えるだけではできない (ソースコードの編集も必要です) ので、このフィールドに手を加えるべきではありません。
- *sourcefile* は、少なくともファイル名から何の言語で書かれているかがわかるようになっているソースコードファイル名です。 `.c` で終わるファイルは C で書かれているとみなされ、 `.C` 、 `.cc` 、および `.c++` で終わるファイルは C++ で書かれているとみなされます。 `.m` や `.mm` で終わるファイルは Objective C で書かれているとみなされます。
- *cpparg* は C プリプロセッサへの引数で、 *-I* 、 *-D* 、 *-U* または *-C* のいずれかから始まる文字列です。
- *library* は `.a` で終わるか、 *-l* または *-L* のいずれかから始まる文字列です。

特定のプラットフォームにおいて、プラットフォーム上の特殊なライブラリが必要な場合、 `Setup` ファイルを編集して `python setup.py build` を実行すればライブラリを追加できます。例えば、以下の行

```
foo foomodule.c
```

で定義されたモジュールを、自分のプラットフォーム上の数学ライブラリ `libm.a` とリンクしなければならない場合、 `Setup` 内の行に *-lm* を追加するだけです:

```
foo foomodule.c -lm
```

コンパイラやリンカ向けの任意のスイッチオプションは、*-Xcompiler arg* や *-Xlinker arg* オプションで与えます:

```
foo foomodule.c -Xcompiler -o32 -Xlinker -shared -lm
```

-Xcompiler および *-Xlinker* の後にくるオプションは、それぞれ適切なコマンドラインに追加されます。従って、上の例では、コンパイラには *-o32* オプションが渡され、リンカには *-shared* が渡されます。コンパイラオプションに引数が必要な場合、複数の *-Xcompiler* オプションを与えます; 例えば、 `-x c++` を渡すには、`Setup` ファイルには `-Xcompiler -x -Xcompiler c++` を渡さねばなりません。

コンパイラフラグは、環境変数 `CFLAGS` の設定でも与えられます。 `CFLAGS` が設定されていれば、 `Setup` ファイル内で指定されているコンパイラフラグに `CFLAGS` の内容が追加されます。

6.2 Windows で非 Microsoft コンパイラを使ってビルドするには

6.2.1 Borland/CodeGear C++

この小節では、 Borland C++ コンパイラのバージョン 5.5 で Distutils を使うために必要な手順について述べています。まず、Borland のオブジェクトファイル形式 (OMF) は、Python 公式サイトや ActiveState の Web サイトからダウンロードできるバージョンの Python が使っている形式とは違うことを知っておかねばなりません (Python は通常、 Microsoft Visual C++ でビルドされています。Microsoft Visual C++ は COFF をオブジェクトファイル形式に使います。) このため、以下のようにして、Python のライブラリ `python25.lib` を Borland の形式に変換する必要があります:

```
coff2omf python25.lib python25_bcpp.lib
```

`coff2omf` プログラムは、 Borland コンパイラに付属しています。 `python25.lib` は Python インストールディレクトリの `Libs` ディレクトリ内にあります。拡張モジュールで他のライブラリ (zlib, ...) を使っている場合、それらのライブラリも変換しなければなりません。

変換されたファイルは、通常のライブラリと同じディレクトリに置かねばなりません。

さて、 Distutils は異なる名前を持つこれらのライブラリをどのように扱うのでしょうか? 拡張モジュールで (例えば `foo` という名の) ライブラリが必要な場合、 Distutils はまず `_bcpp` が後ろに付いたライブラリ (例えば `foo_bcpp.lib`) が見つかるかどうか調べ、あればそのライブラリを使います。該当するライブラリがなければ、デフォルトの名前 (`foo.lib`) を使います [#]_。

Borland C++ を使って Distutils に拡張モジュールをコンパイルさせるには、以下のように入力します:

```
python setup.py build --compiler=bcpp
```

Borland C++ コンパイラをデフォルトにしたいなら、自分用、またはシステム全体向けに、 Distutils の設定ファイルを書くことを検討した方がよいでしょう (*Distutils 設定ファイル* 節を参照してください)。

参考:

C++Builder Compiler Borland によるフリーの C++ コンパイラに関する情報で、コンパイラのダウンロードページへのリンクもあります。

Creating Python Extensions Using Borland's Free Compiler Borland 製のフリーのコマンドライン C++ を使って Python をビルドする方法について述べたドキュメントです。

6.2.2 GNU C / Cygwin / MinGW

この手引きは 2.4.1 以降の Python と 3.0.0 (binutils-2.13.90-20030111-1) 以上の MinGW でのみ有効です。

この節では、 Cygwin や MinGW [1] 配布物中の GNU C/C++ コンパイラで Distutils を使うために必要な手順について述べます。 Cygwin 向けにビルドされている Python インタプリタを使っているなら、以下の手順をとらなくても Distutils はまったく問題なく動作します。

上記のコンパイラは、いくつかの特殊なライブラリを必要とします。この作業は Borland の C++ よりもやや複雑です。というのは、ライブラリを変換するためのプログラムが存在しないからです。 まず、 Python DLL が公開している全てのシンボルからなるリストを作成しなければなりません。 (この作業むけの良いプログラムは、http://www.emmestech.com/software/cygwin/pexports-0.43/download_pexports.html から入手できます。)

```
pexports python25.dll >python25.def
```

これで、上で得られた情報をもとに、 gcc 用の import ライブラリを作成できます。

インストールされた `python25.dll` の位置はインストールオプションと、Windows のバージョンと言語に依存します。”自分だけのため” のインストールの場合には、インストールディレクトリのルートに配置されます。共有インストールの場合にはシステムディレクトリに配置されます。

```
dlltool --dllname python25.dll --def python25.def --output-lib libpython25.a
```

出来上がったライブラリは、`python25.lib` と同じディレクトリ (Python インストールディレクトリの `libs` ディレクトリになるはずです) に置かなければなりません。

拡張モジュールが他のライブラリ (zlib, ...) を必要とする場合、それらのライブラリも変換しなければなりません。変換されたファイルは、それぞれ通常のライブラリが置かれているのと同じディレクトリに置かねばなりません。

Cygwin を使って Distutils に拡張モジュールをコンパイルさせるには、

```
python setup.py build --compiler=cygwin
```

のように入力します。また、非 cygwin モードの Cygwin [2] や MinGW では、

```
python setup.py build --compiler=mingw32
```

のように入力します。

[1] つまり、全ての既存の COFF ライブラリを同名の OMF ライブラリに置き換えてもかまわないということです

[2] 詳しくは http://sources.redhat.com/cygwin/ や http://www.mingw.org/ を参照してください

上記のオプションやコンパイラをデフォルトにしたいなら、自分用、またはシステム全体向けに、Distutils の設定ファイルを書くことを検討した方がよいでしょう (*Distutils* 設定ファイル 節を参照してください)。

参考:

Building Python modules on MS Windows platform with MinGW MinGW 環境で必要なライブラリのビルドに関する情報があります。

脚注

第 7

日本語訳について

7.1 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 Installing Python Modules 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。

7.1.1 翻訳者一覧 (敬称略)

- Yasushi Masuda <y.masuda@acm.org> (April 2, 2004)
- Kazuo Moriwaka
- INADA Naoki <inada-n at klab.jp>

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

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.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.
```

Copyright

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