Distributing Python Modules

リリース *2.6ja2*

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

2011 年 11 月 06 日

Python Software Foundation
Email: docs@python.org

目 次

Release 2.6

Date 2011 年 11 月 06 日

このドキュメントでは、 Python 配布ユーティリティ (Python Distribution Utilities, “Distutils”) について、モジュール開発者の視点に立ち、多くの人々がビルド/リリース/インストールの手間をほとんどかけずに Python モジュールや拡張モジュールを入手できるようにする方法について述べます。

第 1 章

Distutilsの紹介

このドキュメントで扱っている内容は、 Distutils を使った Python モジュールの配布で、とりわけ開発者/配布者の役割に重点を置いています: Python モジュールのインストールに関する情報を探しているのなら、*install-index* を参照してください。

1.1 概念と用語

Distutils の使い方は、モジュール開発者とサードパーティ製のモジュールをインストールするユーザ/管理者のどちらにとってもきわめて単純です。開発者側のやるべきことは (もちろん、しっかりした実装で、詳しく文書化され、よくテストされたコードを書くことは別として！) 以下の項目になります:

- setup スクリプト (`setup.py` という名前にするのがならわし) を書く
- (必要があれば) setup 設定ファイルを書く
- ソースコード配布物を作成する
- (必要があれば) 一つまたはそれ以上のビルド済み (バイナリ) 形式の配布物を作成する

これらの作業については、いずれもこのドキュメントで扱っています。

全てのモジュール開発者が複数の実行プラットフォームを利用できるわけではないので、全てのプラットフォーム向けにビルド済みの配布物を提供してもらえると期待するわけにはいきません。ですから、仲介を行う人々、いわゆる **パッケージ作成者** *(packager)* がこの問題を解決すべく立ち上がってくれることが望ましいでしょう。パッケージ作成者はモジュール開発者がリリースしたソースコード配布物を、一つまたはそれ以上のプラットフォーム上でビルドして、得られたビルド済み配布物をリリースすることになります。したがって、ほとんどの一般的なプラットフォームにおけるユーザは、setup スクリプト一つ実行せず、コードを一行たりともコンパイルしなくても、使っているプラットフォーム向けのきわめて普通の方法でほとんどの一般的な Python モジュール配布物をインストールできるでしょう。

1.2 簡単な例

setup スクリプトは通常単純なものですが、Python で書かれているため、スクリプト中で何かを処理しようと考えたとき特に制限はありません。とはいえ、setup スクリプト中に何かコストの大きな処理を行うときは十分注意してください。 autoconf 形式の設定スクリプトとは違い、 setup スクリプトはモジュール配布物をビルドしてインストールする中で複数回実行されることがあります。

`foo.py` という名前のファイルに収められている `foo` という名前のモジュールを配布したいだけなら、setup スクリプトは以下のような単純なものになります:

```
from distutils.core import setup
setup(name='foo',
      version='1.0',
      py_modules=['foo'],
      )
```

以下のことに注意してください:

- Distutils に与えなければならない情報のほとんどは、 `setup()` 関数のキーワード引数として与えます。
- キーワード引数は二つのカテゴリ: パッケージのメタデータ (パッケージ名、バージョン番号) 、パッケージに何が収められているかの情報 (上の場合は pure Python モジュールのリスト)、に行き着きます。
- モジュールはファイル名ではなく、モジュール名で指定します (パッケージと拡張モジュールについても同じです)
- 作者名、電子メールアドレス、プロジェクトの URL といった追加のメタデータを入れておくよう奨めます (*setup* **スクリプトを書く** の例を参照してください)

このモジュールのソースコード配布物を作成するには、上記のコードが入った setup スクリプト `setup.py` を作成して、以下のコマンド:

```
python setup.py sdist
```

を実行します。

この操作を行うと、アーカイブファイル (例えば Unix では tarball、Windows では ZIP ファイル) を作成します。アーカイブファイルには、setup スクリプト `setup.py` と、配布したいモジュール `foo.py` が入っています。アーカイブファイルの名前は `foo-1.0.targ.gz` (または `.zip`) になり、展開するとディレクトリ `foo-1.0` を作成します。

エンドユーザが `foo` モジュールをインストールしたければ、 `foo-1.0.tar.gz` (または `.zip`) をダウンロードし、パッケージを展開して、以下のスクリプトを — `foo-1.0` ディレクトリ中で — 実行します:

```
python setup.py install
```

この操作を行うと、インストールされている Python での適切なサードパーティ製モジュール置き場に `foo.py` を完璧にコピーします.

ここで述べた簡単な例では、 Distutils の基本的な概念のいくつかを示しています。まず、開発者とインストール作業者は同じ基本インタフェース、すなわち setup スクリプトを使っています。二人の作業の違いは、使っている Distutils **コマンド** *(command)* にあります: **sdist** コマンドは、ほぼ完全に開発者だけが対象

となる一方、 **install** はどちらかというとインストール作業者向けです (とはいえ、ほとんどの開発者は自分のコードをインストールしたくなることがあるでしょう)。

ユーザにとって本当に簡単なものにしたいのなら、一つまたはそれ以上のビルド済み配布物を作ってあげられます。例えば、Windows マシン上で作業をしていて、他の Windows ユーザにとって簡単な配布物を提供したいのなら、実行可能な形式の (このプラットフォーム向けのビルド済み配布物としてはもっとも適切な) インストーラを作成できます。これには **bdist_wininst** を使います。例えば:

```
python setup.py bdist_wininst
```

とすると、実行可能なインストーラ形式、 `foo-1.0.win32.exe` が現在のディレクトリに作成されます。

その他の有用な配布形態としては、 **bdist_rpm** に実装されている RPM 形式、 Solaris **pkgtool** (**bdist_pkgtool**) 、 HP-UX **swinstall** (**bdist_sdux**) があります。例えば、以下のコマンドを実行すると、 `foo-1.0.noarch.rpm` という名前の RPM ファイルを作成します:

```
python setup.py bdist_rpm
```

(**bdist_rpm** コマンドは **rpm** コマンドを使うため、 Red Hat Linux や SuSE Linux、 Mandrake Linux といった RPM ベースのシステムで実行しなければなりません)

どの配布形式が利用できるかは、

```
python setup.py bdist --help-formats
```

を実行すれば分かります。

1.3 Python 一般の用語

このドキュメントを読んでいるのなら、モジュール (module)、拡張モジュール (extension) などが何を表すのかをよく知っているかもしれません。とはいえ、読者がみな共通のスタートポイントに立って Distutils の操作を始められるように、ここで一般的な Python 用語について以下のような用語集を示しておきます:

モジュール (module) Python においてコードを再利用する際の基本単位: すなわち、他のコードから import されるひとかたまりのコードです。ここでは、三種類のモジュール: pure Python モジュール、拡張モジュール、パッケージが関わってきます。

pure Python モジュール Python で書かれ、単一の `.py` ファイル内に収められたモジュールです (`.pyc` かつ/または `.pyo` ファイルと関連があります) 。 “pure モジュール (pure module)” と呼ばれることもあります。

拡張モジュール (extension module) Python を実装している低水準言語: Python の場合は C/C++ 、 Jython の場合は Java 、で書かれたモジュールです。通常は、動的にロードできるコンパイル済みの単一のファイルに入っています。例えば、Unix 向け Python 拡張のための共有オブジェクト (`.so`)、Windows 向け Python 拡張のための DLL (`.pyd` という拡張子が与えられています)、Jython 拡張のための Java クラスといった具合です。 (現状では、 Distutils は Python 向けの C/C++ 拡張モジュールしか扱わないので注意してください。)

パッケージ (package) 他のモジュールが入っているモジュールです; 通常、ファイルシステム内のあるディレクトリに収められ、 `__init__.py` が入っていることで通常のディレクトリと区別できます。

ルートパッケージ (root package) 階層的なパッケージの根 (root) の部分にあたるパッケージです。(この部分には `__init__.py` ファイルがないので、本当のパッケージではありませんが、便宜上そう呼びます。) 標準ライブラリの大部分はルートパッケージに入っています、また、多くの小規模な単体のサードパーティモジュールで、他の大規模なモジュールコレクションに属していないものもここに入ります。正規のパッケージと違い、ルートパッケージ上のモジュールの実体は様々なディレクトリにあります: 実際は、 `sys.path` に列挙されているディレクトリ全てが、ルートパッケージに配置されるモジュールの内容に影響します。

1.4 Distutils 固有の用語

以下は Distutils を使って Python モジュールを配布する際に使われる特有の用語です:

モジュール配布物 (module distribution) 一個のファイルとしてダウンロード可能なリソースの形をとり、**一括して** インストールされることになっている形態で配られる Python モジュールのコレクションです。よく知られたモジュール配布物には、Numeric Python、 PyXML、 PIL (the Python Imaging Library)、mxBase などがあります。 (**パッケージ** *(package)* と呼ばれることもありますが、 Python 用語としてのパッケージとは意味が違います: 一つのモジュール配布物の中には、場合によりゼロ個、一つ、それ以上の Python パッケージが入っています。)

pure モジュール配布物 (pure module distribution) pure Python モジュールやパッケージだけが入ったモジュール配布物です。”pure 配布物 (pure distribution)” とも呼ばれます。

非 pure モジュール配布物 (non-pure module distribution) 少なくとも一つの拡張モジュールが入ったモジュール配布物です。 “非 pure 配布物” とも呼びます。

配布物ルートディレクトリ (distribution root) ソースコードツリー (またはソース配布物) ディレクトリの最上階層で、 `setup.py` のある場所です。一般的には、`setup.py` はこのディレクトリ上で実行します。

第 2 章

setup スクリプトを書く

setup スクリプトは、Distutils を使ってモジュールをビルドし、配布し、インストールする際の全ての動作の中心になります。 setup スクリプトの主な目的は、モジュール配布物について Distutils に伝え、モジュール配布を操作するための様々なコマンドを正しく動作させることにあります。上の **簡単な例** の節で見てきたように、 setup スクリプトは主に `setup()` の呼び出しからなり、開発者が distuils に対して与えるほとんどの情報は `setup()` のキーワード引数として指定されます。

ここではもう少しだけ複雑な例: Distutils 自体の setup スクリプト、を示します。これについては、以降の二つの節でフォローします。(Distutils が入っているのは Python 1.6 以降であり、 Python 1.5.2 ユーザが他のモジュール配布物をインストールできるようにするための独立したパッケージがあることを思い出してください。ここで示した、Distutils 自身の setup スクリプトは、Python 1.5.2 に Distutils パッケージをインストールする際に使います。)

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

from distutils.core import setup

setup(name='Distutils',
      version='1.0',
      description='Python Distribution Utilities',
      author='Greg Ward',
      author_email='gward@python.net',
      url='http://www.python.org/sigs/distutils-sig/',
      packages=['distutils', 'distutils.command'],
     )
```

上の例と、 **簡単な例** で示したファイル一つからなる小さな配布物とは、違うところは二つしかありません: メタデータの追加と、モジュールではなくパッケージとして pure Python モジュール群を指定しているという点です。この点は重要です。というのも、 Distutils は 2 ダースものモジュールが (今のところ) 二つのパッケージに分かれて入っているからです; 各モジュールについていちいち明示的に記述したリストは、作成するのが面倒だし、維持するのも難しくなるでしょう。その他のメタデータについては、 **追加のメタデータ** を参照してください。

setup スクリプトに与えるパス名 (ファイルまたはディレクトリ) は、 Unix におけるファイル名規約、つまりスラッシュ ('/') 区切りで書かねばなりません。Distutils はこのプラットフォーム中立の表記を、実際にパス名として使う前に、現在のプラットフォームに適した表記に注意深く変換します。この機能のおかげで、setup スクリプトを異なるオペレーティングシステム間にわたって可搬性があるものにできます。言う

までもなく、これは Distutils の大きな目標の一つです。この精神に従って、このドキュメントでは全てのパス名をスラッシュ区切りにしています。

もちろん、この取り決めは Distutils に渡すパス名だけに適用されます。もし、例えば `glob.glob()` や `os.listdir()` のような、標準の Python 関数を使ってファイル群を指定するのなら、パス区切り文字 (path separator) をハードコーディングせず、以下のように可搬性のあるコードを書くよう注意すべきです:

```
glob.glob(os.path.join('mydir', 'subdir', '*.html'))
os.listdir(os.path.join('mydir', 'subdir'))
```

2.1 パッケージを全て列挙する

packages オプションは、 *packages* リスト中で指定されている各々のパッケージについて、パッケージ内に見つかった全ての pure Python モジュールを処理 (ビルド、配布、インストール、等) するよう Distutils に指示します。このオプションを指定するためには、当然のことながら各パッケージ名はファイルシステム上のディレクトリ名と何らかの対応付けができなければなりません。デフォルトで使われる対応関係はきわめてはっきりしたものです。すなわち、パッケージ `distutils` が配布物ルートディレクトリからの相対パス `distutils` で表されるディレクトリ中にあるというものです。つまり、setup スクリプト中で `packages = ['foo']` と指定したら、スクリプトの置かれたディレクトリからの相対パスで `foo/__init__.py` を探し出せると Distutils に確約したことになります。この約束を裏切ると Distutils は警告を出しますが、そのまま壊れたパッケージの処理を継続します。

ソースコードディレクトリの配置について違った規約を使っていても、まったく問題はありません: 単に *package_dir* オプションを指定して、 Distutils に自分の規約を教えればよいのです。例えば、全ての Python ソースコードを `lib` 下に置いて、"ルートパッケージ" 内のモジュール (つまり、どのパッケージにも入っていないモジュール) を `lib` 内に入れ、 `foo` パッケージを `lib/foo` に入れる、といった具合にしたいのなら、

```
package_dir = {'': 'lib'}
```

を setup スクリプト内に入れます。辞書内のキーはパッケージ名で、空のパッケージ名はルートパッケージを表します。キーに対応する値はルートパッケージからの相対ディレクトリ名です、この場合、 `packages = ['foo']` を指定すれば、`lib/foo/__init__.py` が存在すると Distutils に確約したことになります。

もう一つの規約のあり方は `foo` パッケージを `lib` に置き換え、 `foo.bar` パッケージが `lib/bar` にある、などとするものです。このような規約は、 setup スクリプトでは

```
package_dir = {'foo': 'lib'}
```

のように書きます。 *package_dir* 辞書に `package: dir` のようなエントリがあると、 *package* の下にある全てのパッケージに対してこの規則が暗黙のうちに適用され、その結果 `foo.bar` の場合が自動的に処理されます。この例では、`packages = ['foo', 'foo.bar']` は、 Distutils に `lib/__init__.py` と `lib/bar/__init__.py` を探すように指示します。 (*package_dir* は再帰的に適用されますが、この場合 *packages* の下にある全てのパッケージを明示的に指定しなければならないことを心に留めておいてください: Distutils は `__init__.py` を持つディレクトリをソースツリーから再帰的にさがしたりは **しません** 。)

2.2 個々のモジュールを列挙する

小さなモジュール配布物の場合、パッケージを列挙するよりも、全てのモジュールを列挙するほうがよいと思うかもしれません — 特に、単一のモジュールが “ルートパッケージ” にインストールされる (すなわち、パッケージは全くない) ような場合がそうです。この最も単純なケースは**簡単な例** で示しました; ここではもうちょっと入り組んだ例を示します:

```
py_modules = ['mod1', 'pkg.mod2']
```

ここでは二つのモジュールについて述べていて、一方は “ルート” パッケージに入り、他方は `pkg` パッケージに入ります。ここでも、デフォルトのパッケージ/ディレクトリのレイアウトは、二つのモジュールが `mod1.py` と `pkg/mod2.py` にあり、`pkg/__init__.py` が存在することを暗示しています。また、パッケージ/ディレクトリの対応関係は *`package_dir`* オプションでも上書きできます。

2.3 拡張モジュールについて記述する

pure Python モジュールを書くより Python 拡張モジュールを書く方がちょっとだけ複雑なように、 Distutils での拡張モジュールに関する記述もちょっと複雑です。pure モジュールと違い、単にモジュールやパッケージを列挙して、Distutils が正しいファイルを見つけてくれると期待するだけでは十分ではありません; 拡張モジュールの名前、ソースコードファイル (群) 、そして何らかのコンパイル/リンクに関する必要事項 (include ディレクトリ、リンクすべきライブラリ、等) を指定しなければなりません。

こうした指定は全て、 `setup()` の別のキーワード引数、 *`ext_modules`* オプションを介して行えます。 *`ext_modules`* は、 `Extension` インスタンスからなるただのリストで、各インスタンスに一個の拡張モジュールを記述するようになっています。仮に、 `foo.c` で実装された拡張モジュール `foo` が、配布物に一つだけ入ってるとします。コンパイラ/リンカに他の情報を与える必要がない場合、この拡張モジュールのための記述はきわめて単純です:

```
Extension('foo', ['foo.c'])
```

`Extension` クラスは、 `setup()` によって、 `distutils.core` から import されます。従って、拡張モジュールが一つだけ入っていて、他には何も入っていないモジュール配布物を作成するための setup スクリプトは、以下のようになるでしょう:

```
from distutils.core import setup, Extension
setup(name='foo',
      version='1.0',
      ext_modules=[Extension('foo', ['foo.c'])],
      )
```

`Explained` クラス (実質的には、 `Explained` クラスの根底にある **build_ext** コマンドで実装されている、拡張モジュールをビルドする機構) は、Python 拡張モジュールをきわめて柔軟に記述できるようなサポートを提供しています。これについては後の節で説明します。

2.3.1 拡張モジュールの名前とパッケージ

`Extension` クラスのコンストラクタに与える最初の引数は、常に拡張モジュールの名前にします。これにはパッケージ名も含めます。例えば、

```
Extension('foo', ['src/foo1.c', 'src/foo2.c']p)
```

とすると、拡張モジュールをルートパッケージに置くことになります。一方、

```
Extension('pkg.foo', ['src/foo1.c', 'src/foo2.c'])
```

は、同じ拡張モジュールを pkg パッケージの下に置くよう記述しています。ソースコードファイルと、作成されるオブジェクトコードはどちらの場合でも同じです; 作成された拡張モジュールがファイルシステム上のどこに置かれるか (すなわち Python の名前空間上のどこに置かれるか) が違うにすぎません。

同じパッケージ内に (または、同じ基底パッケージ下に) いくつもの拡張モジュールがある場合、*ext_package* キーワード引数を `setup()` に指定します。例えば、

```
setup(...,
      ext_package='pkg',
      ext_modules=[Extension('foo', ['foo.c']),
                   Extension('subpkg.bar', ['bar.c'])],
     )
```

とすると、 `foo.c` をコンパイルして `pkg.foo` にし、 `bar.c` をコンパイルして `pkg.subpkg.bar` にします。

2.3.2 拡張モジュールのソースファイル

`Extension` コンストラクタの二番目の引数は、ソースファイルのリストです。 Distutils は現在のところ、C、C++、そして Objective-C の拡張しかサポートしていないので、引数は通常 C/C++/Objective-C ソースコードファイルになります。(C++ソースコードファイルを区別できるよう、正しいファイル拡張子を使ってください: `.cc` や `.cpp` にすれば、Unix と Windows 用の双方のコンパイラで認識されるようです。)

ただし、 SWIG インタフェース (`.i`) ファイルはリストに含められます; **build_ext** コマンドは、SWIG で書かれた拡張パッケージをどう扱えばよいか心得ています: **build_ext** は、インタフェースファイルを SWIG にかけ、得られた C/C++ ファイルをコンパイルして拡張モジュールを生成します。

この警告にかかわらず、現在次のようにして SWIG に対してオプションを渡すことができます。

```
setup(...,
      ext_modules=[Extension('_foo', ['foo.i'],
                             swig_opts=['-modern', '-I../include'])],
      py_modules=['foo'],
     )
```

もしくは、次のようにコマンドラインからオプションを渡すこともできます。

```
> python setup.py build_ext --swig-opts="-modern -I../include"
```

プラットフォームによっては、コンパイラで処理され、拡張モジュールに取り込まれるような非ソースコードファイルを含められます。非ソースコードファイルとは、現状では Visual C++向けの Windows メッセージテキスト (`.mc`) ファイルや、リソース定義 (`.rc`) ファイルを指します。これらのファイルはバイナリリソース (`.res`) ファイルにコンパイルされ、実行ファイルにリンクされます。

2.3.3 プリプロセッサオプション

Extension には三種類のオプション引数: include_dirs, define_macros, そして undef_macros があり、検索対象にするインクルードディレクトリを指定したり、プリプロセッサマクロを定義 (define)/定義解除 (undefine) したりする必要があるとき役立ちます。

例えば、拡張モジュールが配布物ルート下の include ディレクトリにあるヘッダファイルを必要とするときには、include_dirs オプションを使います:

```
Extension('foo', ['foo.c'], include_dirs=['include'])
```

ここには絶対パスも指定できます; 例えば、自分の拡張モジュールが、 /usr の下に X11R6 をインストールした Unix システムだけでビルドされると知っていれば、

```
Extension('foo', ['foo.c'], include_dirs=['/usr/include/X11'])
```

のように書けます。

自分のコードを配布する際には、このような可搬性のない使い方は避けるべきです: おそらく、 C のコードを

```
#include <X11/Xlib.h>
```

のように書いた方がましでしょう。

他の Python 拡張モジュール由来のヘッダを include する必要があるなら、 Distutils の **install_header** コマンドが一貫した方法でヘッダファイルをインストールするという事実を活用できます。例えば、Numerical Python のヘッダファイルは、 (標準的な Unix がインストールされた環境では) /usr/local/include/python1.5/Numerical にインストールされます。 (実際の場所は、プラットフォームやどの Python をインストールしたかで異なります。) Python の include ディレクトリ — 今の例では/usr/local/include/python1.5 — は、 Python 拡張モジュールをビルドする際に常にヘッダファイル検索パスに取り込まれるので、 C コードを書く上でもっともよいアプローチは、

```
#include <Numerical/arrayobject.h>
```

となります。

Numerical インクルードディレクトリ自体をヘッダ検索パスに置きたいのなら、このディレクトリを Distutils の distutils.sysconfig モジュールを使って見つけさせられます:

```
from distutils.sysconfig import get_python_inc
incdir = os.path.join(get_python_inc(plat_specific=1), 'Numerical')
setup(...,
      Extension(..., include_dirs=[incdir]),
      )
```

この書き方も可搬性はあります — プラットフォームに関わらず、どんな Python がインストールされていても動作します — が、単に実践的な書き方で C コードを書く方が簡単でしょう。

define_macros `` および `` undef_macros オプションを使って、プリプロセッサマクロを定義 (define) したり、定義解除 (undefine) したりもできます。 define_macros はタプル (name, value) からなるリストを引数にとります。 name は定義したいマクロの名前 (文字列) で、 value はその値です: value は文字列か None `` になります。(マクロ `` FOO `` を `` None `` にすると、C ソース

コード内で `` #define FOO と書いたのと同じになります: こう書くと、ほとんどのコンパイラは FOO を文字列 1 に設定します。) undef_macros には、定義解除したいマクロ名からなるリストを指定します。

例えば、以下の指定:

```
Extension(...,
          define_macros=[('NDEBUG', '1'),
                         ('HAVE_STRFTIME', None)],
          undef_macros=['HAVE_FOO', 'HAVE_BAR'])
```

は、全ての C ソースコードファイルの先頭に、以下のマクロ:

```
#define NDEBUG 1
#define HAVE_STRFTIME
#undef HAVE_FOO
#undef HAVE_BAR
```

があるのと同じになります。

2.3.4 ライブラリオプション

拡張モジュールをビルドする際にリンクするライブラリや、ライブラリを検索するディレクトリも指定できます。 libraries はリンクするライブラリのリストで、 library_dirs はリンク時にライブラリを検索するディレクトリのリストです。また、 runtime_library_dirs は、実行時に共有ライブラリ (動的にロードされるライブラリ) を検索するディレクトリのリストです。

例えば、ビルド対象システムの標準ライブラリ検索パスにあることが分かっているライブラリをリンクする時には、以下のようにします。

```
Extension(...,
          libraries=['gdbm', 'readline'])
```

非標準のパス上にあるライブラリをリンクしたいなら、その場所を library_dirs に入れておかなければなりません:

```
Extension(...,
          library_dirs=['/usr/X11R6/lib'],
          libraries=['X11', 'Xt'])
```

(繰り返しになりますが、この手の可搬性のない書き方は、コードを配布するのが目的なら避けるべきです。)

2.3.5 その他の操作

他にもいくつかオプションがあり、特殊な状況を扱うために使います。

extra_objects オプションには、リンカに渡すオブジェクトファイルのリストを指定します。ファイル名には拡張子をつけてはならず、コンパイラで使われているデフォルトの拡張子が使われます。

extra_compile_args および *extra_link_args* には、それぞれコンパイラとリンカに渡す追加のコマンドライン引数を指定します。

export_symbols は Windows でのみ意味があります。このオプションには、公開 (export) する (関数や変数の) シンボルのリストを入れられます。コンパイルして拡張モジュールをビルドする際には、このオプションは不要です: Distutils は公開するシンボルを自動的に `initmodule` に渡すからです。

2.4 パッケージと配布物の関係 (Relationships between Distributions and Packages)

配布物はパッケージと 3 種類の方法で関係します:

1. パッケージかモジュールを要求する。
2. パッケージかモジュールを提供する。
3. パッケージかモジュールを廃止する。

これらの関係は、 `distutils.core.setup()` 関数のキーワード引数を利用して指定することができます。

他の Python モジュールやパッケージに対する依存は、 `setup()` の *requires* キーワード引数で指定できます。引数の値は文字列のリストでなければなりません。各文字列は、必要とするパッケージと、オプションとしてパッケージのバージョンを指定します。

あるモジュールかパッケージの任意のバージョンが必要な場合、指定する文字列はモジュール名かパッケージ名になります。例えば、 `'mymodule'` や `'xml.parsers.expat'` を含みます。

特定のバージョンが必要な場合、修飾子 (qualifier) の列を加えることができます。各修飾子は、比較演算子とバージョン番号からなります。利用できる比較演算子は:

```
<     >     ==
<=    >=    !=
```

これらの修飾子はカンマ (空白文字を入れても良いです) で区切って複数並べることができます。その場合、全ての修飾子が適合する必要があります; 評価する時に論理 AND でつなげられます。

いくつかの例を見てみましょう:

require 式	説明
`==1.0`	version `1.0` のみが適合します
`>1.0, !=1.5.1, <2.0`	`1.5.1` を除いて、 `1.0` より後ろで `2.0` より前の全てのバージョンに適合します。

これで、依存を指定することができました。同じように、この配布物が他の配布物に必要とされる何を提供するのかを指定する必要があります。これは、 `setup()` の *provide* キーワード引数によって指定できます。この引数の値は文字列のリストで、各要素は Python モジュールかパッケージの名前です。バージョンを指定することもできます。もしバージョンが指定されなかった場合、配布物のバージョン番号が利用されます。

いくつかの例です:

provide 式	説明
`mypkg`	`mypkg` を提供します。バージョンは配布物のものを使います。
`mypkg (1.1)`	`mypkg` version 1.1 を提供します。配布物のバージョン番号は気にしません

パッケージは *obsoletes* キーワードを利用することで、他のパッケージを廃止することを宣言することもできます。この値は *requires* キーワードと似ています: モジュールやパッケージを指定する文字列のリストです。各文字列は、モジュールかパッケージの名前と、オプションとして一つ以上のバージョン指定から構成されています。バージョン指定は、モジュールやパッケージの名前のうしろに、丸括 (parentheses) でかこわれています。

指定されたバージョンは、その配布物によって廃止されるバージョンを示しています。バージョン指定が存在しない場合は、指定された名前のモジュールまたはパッケージの全てが廃止されたと解釈されます。

2.5 スクリプトをインストールする

ここまでは、スクリプトから import され、それ自体では実行されないような pure Python モジュールおよび非 pure Python モジュールについて扱ってきました。

スクリプトとは、Python ソースコードを含むファイルで、コマンドラインから実行できるよう作られているものです。スクリプトは Distutils に複雑なことを一切させません。唯一の気の利いた機能は、スクリプトの最初の行が `#!` で始まっていて、 “python” という単語が入っていた場合、Distutils は最初の行を現在使っているインタプリタを参照するよう置き換えます。デフォルトでは現在使っているインタプリタと置換しますが、オプション `--executable` (または `-e`) を指定することで、明示的にインタプリタのパスを指定して上書きすることができます。

scripts オプションには、単に上で述べた方法で取り扱うべきファイルのリストを指定するだけです。PyXML の setup スクリプトを例に示します:

```
setup(...,
      scripts=['scripts/xmlproc_parse', 'scripts/xmlproc_val']
      )
```

2.6 パッケージデータをインストールする

しばしばパッケージに追加のファイルをインストールする必要があります。このファイルは、パッケージの実装に強く関連したデータや、そのパッケージを使うプログラマーが必要とするドキュメントなどです。これらのファイルを **パッケージデータ** と呼びます。

パッケージデータは関数 `setup()` にキーワード引数 `package_data` を与えることで追加できます。この値はパッケージ名から、パッケージへコピーされる相対パス名リストへのマップである必要があります。それぞれのパスは対応するパッケージが含まれるディレクトリ (もし適切なら `package_dir` のマッピングが利用されます) からの相対パスとして扱われます。つまり、ファイルはソースディレクトリ中にパッケージの一部として存在すると仮定されています。この値にはグロブパターンを含むことができます。

パス名にはディレクトリ部分を含むことができます。必要なディレクトリはインストール時に作成されます。

たとえば、パッケージがいくつかのデータファイルを含むサブディレクトリを含んでいる場合、ソースツリーでは以下のように配置できます:

```
setup.py
src/
    mypkg/
        __init__.py
```

```
        module.py
        data/
            tables.dat
            spoons.dat
            forks.dat
```

対応する `setup()` 呼び出しは以下のようになります:

```
setup(...,
      packages=['mypkg'],
      package_dir={'mypkg': 'src/mypkg'},
      package_data={'mypkg': ['data/*.dat']},
      )
```

バージョン 2.4 で追加.

2.7 追加のファイルをインストールする

data_files オプションを使うと、モジュール配布物で必要な追加のファイル: 設定ファイル、メッセージカタログ、データファイル、その他これまで述べてきたカテゴリに収まらない全てのファイルを指定できます。

data_files には、(*directory*, *files*) のペアを以下のように指定します:

```
setup(...,
      data_files=[('bitmaps', ['bm/b1.gif', 'bm/b2.gif']),
                  ('config', ['cfg/data.cfg']),
                  ('/etc/init.d', ['init-script'])]
     )
```

データファイルのインストール先ディレクトリ名は指定できますが、データファイル自体の名前の変更はできないので注意してください。

各々の (*directory*, *files*) ペアには、インストール先のディレクトリ名と、そのディレクトリにインストールしたいファイルを指定します。 *directory* が相対パスの場合、インストールプレフィクス (installation prefix、pure Python パッケージなら `sys.prefix`、拡張モジュールの入ったパッケージなら `sys.exec_prefix`) からの相対パスと解釈されます。*files* 内の各ファイル名は、パッケージソースコード配布物の最上階層の、`setup.py` のあるディレクトリからの相対パスと解釈されます。*files* に書かれたディレクトリ情報は、ファイルを最終的にどこにインストールするかを決めるときには使われません; ファイルの名前だけが使われます。

data_files オプションは、ターゲットディレクトリを指定せずに、単にファイルの列を指定できます。しかし、このやり方は推奨されておらず、指定すると **install** コマンドが警告を出力します。ターゲットディレクトリにデータファイルを直接インストールしたいなら、ディレクトリ名として空文字列を指定してください。

2.8 追加のメタデータ

setup スクリプトには、名前やバージョンにとどまらず、その他のメタデータを含められます。以下のような情報を含められます:

メタデータ	説明	値	注記
name	パッケージの名前	短い文字列	(1)
version	リリースのバージョン	短い文字列	(1)(2)
author	パッケージ作者の名前	短い文字列	(3)
author_email	パッケージ作者の電子メールアドレス	電子メールアドレス	(3)
maintainer	パッケージメンテナンス担当者の名前	短い文字列	(3)
maintainer_email	パッケージメンテナンス担当者の電子メールアドレス	電子メールアドレス	(3)
url	パッケージのホームページ	URL	(1)
description	パッケージについての簡潔な概要説明	短い文字列	
long_description	パッケージについての詳細な説明	長い文字列	
download_url	パッケージをダウンロードできる場所	URL	(4)
classifiers	分類語のリスト	文字列からなるリスト	(4)
platforms	プラットフォームのリスト	文字列からなるリスト	
license	パッケージのライセンス	文字列からなるリスト	(6)

注記:

1. 必須のフィールドです。
2. バージョン番号は *major.minor[.patch[.sub]]* の形式をとるよう奨めます。
3. 作者かメンテナのどちらかは必ず区別してください。
4. これらのフィールドは、2.2.3 および 2.3 より以前のバージョンの Python でも互換性を持たせたい場合には指定してはなりません。リストは PyPI ウェブサイト にあります。
6. `license` フィールドは、パッケージのライセンスが Trove Classifier の “License” から選べない場合に、そのライセンスを示すテキストです。`Classifier` を参照してください。非推奨ですが、 `licence` というオプションも存在していて、 `license` のエイリアスになっています。

「短い文字列」 200 文字以内の一行のテキスト。

「長い文字列」 複数行からなり、ReStructuredText 形式で書かれたプレーンテキスト (http://docutils.sf.net/ を参照してください)。

「文字列のリスト」 下記を参照してください。

これらの文字列はいずれも Unicode であってはなりません。

バージョン情報のコード化は、それ自体が一つのアートです。 Python のパッケージは一般的に、 *major.minor[.patch][sub]* というバージョン表記に従います。メジャー (major) 番号は最初は 0 で、これはソフトウェアが実験的リリースにあることを示します。メジャー番号は、パッケージが主要な開発目標を達成したとき、それを示すために加算されてゆきます。マイナー (minor) 番号は、パッケージに重要な新機能が追加されたときに加算されてゆきます。パッチ (patch) 番号は、バグフィクス版のリリースが作成されたときに加算されます。末尾にバージョン情報が追加され、サブリリースを示すこともあります。これは “a1,a2,...,aN” (アルファリリースの場合で、機能や API が変更されているとき)、 “b1,b2,...,bN” (ベータリリースの場合で、バグフィクスのみのとき) 、そして “pr1,pr2,...,prN” (プレリリースの最終段階で、リリーステストのとき) になります。以下に例を示します:

0.1.0 パッケージの最初の実験的なリリース

1.0.1a2 1.0 の最初のパッチバージョンに対する、2 回目のアルファリリース

classifiers は、 Python のリスト型で指定します:

```
setup(...,
      classifiers=[
          'Development Status :: 4 - Beta',
          'Environment :: Console',
          'Environment :: Web Environment',
          'Intended Audience :: End Users/Desktop',
          'Intended Audience :: Developers',
          'Intended Audience :: System Administrators',
          'License :: OSI Approved :: Python Software Foundation License',
          'Operating System :: MacOS :: MacOS X',
          'Operating System :: Microsoft :: Windows',
          'Operating System :: POSIX',
          'Programming Language :: Python',
          'Topic :: Communications :: Email',
          'Topic :: Office/Business',
          'Topic :: Software Development :: Bug Tracking',
          ],
      )
```

`setup.py` に *classifiers* を入れておき、なおかつ 2.2.3 よりも以前のバージョンの Python と後方互換性を保ちたいなら、 `setup.py` 中で `setup()` を呼び出す前に、以下のコードを入れます。

```
# patch distutils if it can't cope with the "classifiers" or
# "download_url" keywords
from sys import version
if version < '2.2.3':
    from distutils.dist import DistributionMetadata
    DistributionMetadata.classifiers = None
    DistributionMetadata.download_url = None
```

2.9 setup スクリプトをデバッグする

setup スクリプトのどこかがまずいと、開発者の思い通りに動作してくれません。

Distutils は setup 実行時の全ての例外を捉えて、簡単なエラーメッセージを出力してからスクリプトを終了します。このような仕様にしているのは、Python にあまり詳しくない管理者がパッケージをインストールする際に混乱しなくてすむようにするためです。もし Distutils のはらわた深くからトレースバックした長大なメッセージを見たら、管理者はきっと Python のインストール自体がおかしくなっているのだと勘違いして、トレースバックを最後まで読み進んで実はファイルパーミッションの問題だったと気づいたりはしないでしょう。

しかし逆に、この仕様は開発者にとってはうまくいかない理由を見つける役には立ちません。そこで、DISTUTILS_DEBUG 環境変数を空文字以外の何らかの値に設定しておけば、 Distutils が何を実行しているか詳しい情報を出力し、例外が発生した場合には完全なトレースバックを出力するようにできます。

第 3 章

setup 設定ファイル (setup configuration file) を書く

時に、配布物をビルドする際に必要な全ての設定を **あらかじめ** 書ききれない状況が起きます: 例えば、ビルドを進めるために、ユーザに関する情報や、ユーザのシステムに関する情報を必要とするかもしれません。こうした情報が単純 — C ヘッダファイルやライブラリを検索するディレクトリのリストのように — であるかぎり、ユーザに設定ファイル (configuration file) `setup.cfg` を提供して編集してもらうのが、安上がりで簡単な特定方法になります。設定ファイルはまた、あらゆるコマンドにおけるオプションにデフォルト値を与えておき、インストール作業者がコマンドライン上や設定ファイルの編集でデフォルト設定を上書きできるようにします。

setup 設定ファイルは setup スクリプト —理想的にはインストール作業者から見えないもの [1] —と、作者の手を離れて、全てインストール作業者次第となる setup スクリプトのコマンドライン引数との間を橋渡しする中間層として有効です。実際、 `setup.cfg` (と、ターゲットシステム上にある、その他の Distutils 設定ファイル) は、 setup スクリプトの内容より後で、かつコマンドラインで上書きする前に処理されます。この仕様の結果、いくつかの利点が生まれます:

- インストール作業者は、作者が `setup.py` に設定した項目のいくつかを `setup.cfg` を変更して上書きできます。
- `setu.py` では簡単に設定できないような、標準でないオプションのデフォルト値を設定できます。
- インストール作業者は、 `setup.cfg` に書かれたどんな設定も `setup.py` のコマンドラインオプションで上書きできます。

設定ファイルの基本的な構文は簡単なものです:

```
[command]
option=value
...
```

ここで、 *command* は Distutils コマンドのうちの一つ (例えば **build_py**, **install**) で、 *option* はそのコマンドでサポートされているオプションのうちの一つです。各コマンドには任意の数のオプションを設定でき、一つの設定ファイル中には任意の数のコマンドセクションを収められます。空白行は無視されます、 `'#'` 文

[1] Distutils が自動設定機能 (auto-configuration) をサポートするまで、おそらくこの理想状態を達成することはないでしょう

字で開始して行末まで続くコメントも同様に無視されます。長いオプション設定値は、継続行をインデントするだけで複数行にわたって記述できます。

あるコマンドがサポートしているオプションのリストは、 *--help* オプションで調べられます。例えば以下のように。

```
> python setup.py --help build_ext
[...]
Options for 'build_ext' command:
  --build-lib (-b)     directory for compiled extension modules
  --build-temp (-t)    directory for temporary files (build by-products)
  --inplace (-i)       ignore build-lib and put compiled extensions into the
                       source directory alongside your pure Python modules
  --include-dirs (-I)  list of directories to search for header files
  --define (-D)        C preprocessor macros to define
  --undef (-U)         C preprocessor macros to undefine
  --swig-opts          list of SWIG command line options
[...]
```

コマンドライン上で *--foo-bar* と綴るオプションは、設定ファイル上では *foo_bar* と綴るので注意してください。

例えば、拡張モジュールを “インプレース (in-place)” でビルドしたいとします — すなわち、 `pkg.ext` という拡張モジュールを持っていて、コンパイル済みの拡張モジュールファイル (例えば Unix では `ext.so`) を pure Python モジュール `pkg.mod1` および `pkg.mod2` と同じソースディレクトリに置きたいとします。こんなときには、 *--inplace* を使えば、確実にビルドを行えます。

```
python setup.py build_ext --inplace
```

しかし、この操作では、常に **build_ext** を明示的に指定しなければならず、 *--inplace* オプションを忘れずに与えなければなりません。こうした設定を “設定しっ放しにする” 簡単な方法は、 `setup.cfg` に書いておくやり方で、設定ファイルは以下のようになります:

```
[build_ext]
inplace=1
```

この設定は、明示的に **build_ext** を指定するかどうかに関わらず、モジュール配布物の全てのビルドに影響します。ソース配布物に `setup.cfg` を含めると、エンドユーザの手で行われるビルドにも影響します — このオプションの例に関しては `setup.cfg` を含めるのはおそらくよくないアイデアでしょう。というのは、拡張モジュールをインプレースでビルドすると常にインストールしたモジュール配布物を壊してしまうからです。とはいえ、ある特定の状況では、モジュールをインストールディレクトリの下に正しく構築できるので、機能としては有用だと考えられます。 (ただ、インストールディレクトリ上でのビルドを想定するような拡張モジュールの配布は、ほとんどの場合よくない考え方です。)

もう一つ、例があります: コマンドによっては、実行時にほとんど変更されないたくさんのオプションがあります; 例えば、 **bdist_rpm** には、RPM 配布物を作成する際に、”spec” ファイルを作成するために必要な情報を全て与えなければなりません。この情報には setup スクリプトから与えるものもあり、 (インストールされるファイルのリストのように) Distutils が自動的に生成するものもあります。しかし、こうした情報の中には **bdist_rpm** のオプションとして与えるものがあり、毎回実行するごとにコマンドライン上で指定するのが面倒です。そこで、以下のような内容が Distutils 自体の `setup.cfg` には入っています:

```
[bdist_rpm]
release = 1
packager = Greg Ward <gward@python.net>
doc_files = CHANGES.txt
```

```
README.txt
USAGE.txt
doc/
examples/
```

doc_files オプションは、単に空白で区切られた文字列で、ここでは可読性のために複数行をまたぐようにしています。

参考:

“Python モジュールのインストール” の *inst-config-syntax* 設定ファイルに関する詳細情報は、システム管理者向けのこのマニュアルにあります。

ソースコード配布物を作成する

簡単な例 節で示したように、ソースコード配布物を作成するには **sdist** コマンドを使います。最も単純な例では、

```
python setup.py sdist
```

のようにします (ここでは、 **sdist** に関するオプションを setup スクリプトや設定ファイル中で行っていないものと仮定します)。**sdist** は、現在のプラットフォームでのデフォルトのアーカイブ形式でアーカイブを生成します。デフォルトの形式は Unix では gzip で圧縮された tar ファイル形式 (`.tar.gz`) で、Windows では ZIP 形式です。

`--formats` オプションを使えば、好きなだけ圧縮形式を指定できます。例えば:

```
python setup.py sdist --formats=gztar,zip
```

は、gzip された tarball と zip ファイルを作成します。利用可能な形式は以下の通りです:

形式	説明	注記
`zip`	zip ファイル (`.zip`)	(1),(3)
`gztar`	gzip 圧縮された tar ファイル (`.tar.gz`)	(2),(4)
`bztar`	bzip2 圧縮された tar ファイル (`.tar.bz2`)	(4)
`ztar`	compress 圧縮された tar ファイル (`.tar.Z`)	(4)
`tar`	tar ファイル (`.tar`)	(4)

注記:

1. Windows でのデフォルトです
2. Unix でのデフォルトです
3. 外部ユーティリティの **zip** か、 `zipfile` モジュールが必要です (Python 1.6 からは標準ライブラリになっています)
4. 外部ユーティリティ: **tar** 、場合によっては **gzip**, **bzip2** 、または **compress** も必要です

4.1 配布するファイルを指定する

明確なファイルのリスト (またはファイルリストを生成する方法) を明示的に与えなかった場合、 **sdist** コマンドはソース配布物に以下のような最小のデフォルトのセットを含めます:

- *py_modules* と *packages* オプションに指定された Python ソースファイル全て
- *ext_modules* や *libraries* オプションに記載された C ソースファイル
- *scripts* オプションで指定されたスクリプト
- テストスクリプトと思しきファイル全て: `test/test*.py` (現状では、Distutils はテストスクリプトをただソース配布物に含めるだけですが、将来は Python モジュール配布物に対するテスト標準ができるかもしれません)
- `README.txt` (または `README`)、 `setup.py` (または setup スクリプトにしているもの) 、および `setup.cfg`

上記のセットで十分なこともありますが、大抵他のファイルを配布物に含めたいと思うでしょう。普通は、`MANIFEST.in` と呼ばれる**マニフェストテンプレート** *(manifest template)* を使ってこれを行います。マニフェストテンプレートは、ソース配布物に含めるファイルの正確なリストであるマニフェストファイル `MANIFEST` をどうやって作成するか指示しているリストです。**sdist** コマンドはこのテンプレートを処理し、書かれた指示とファイルシステム上に見つかったファイルに基づいてマニフェストファイルを作成します。

自分用のマニフェストファイルを書きたいなら、その形式は簡単です: 一行あたり一つの通常ファイル (または通常ファイルに対するシンボリックリンク) だけを書きます。自分で `MANIFEST` を提供する場合、全てを自分で指定しなければなりません: ただし、上で説明したデフォルトのファイルセットは、この中には含まれません。

マニフェストテンプレートには一行あたり一つのコマンドがあります。各コマンドはソース配布物に入れたり配布物から除外したりするファイルのセットを指定します。例えば、Distutils 自体のマニフェストテンプレートの話に戻ると:

```
include *.txt
recursive-include examples *.txt *.py
prune examples/sample?/build
```

各行はかなり明確に意味を取れるはずです: 上の指定では、 `*.txt` にマッチする配布物ルート下の全てのファイル、 `examples` ディレクトリ下にある `*.txt` か `*.py` にマッチする全てのファイルを含め、`examples/sample?/build` にマッチする全てのファイルを除外します。これらの処理はすべて、標準的に含められるファイルセットの評価よりも **後に**行われるので、マニフェストテンプレートに明示的に指示をしておけば、標準セット中のファイルも除外できます。(*--no-defaults* オプションを設定して、標準セット自体を無効にもできます。) 他にも、このマニフェストテンプレート記述のためのミニ言語にはいくつかのコマンドがあります: **ソースコード配布物を作成する***: sdist command* 節を参照してください。

マニフェストテンプレート中のコマンドの順番には意味があります; 初期状態では、上で述べたようなデフォルトのファイルがあり、テンプレート中の各コマンドによって、逐次ファイルを追加したり除去したりしていいます。マニフェストテンプレートを完全に処理し終えたら、ソース配布物中に含めるべきでない以下のファイルをリストから除去します:

- Distutls の “build” (デフォルトの名前は `build`) ツリー下にある全てのファイル
- `RCS`, `CVS`, `.svn`, `.hg`, `.git`, `.bzr`, `_darcs` といった名前のディレクトリ下にある全てのファイル

こうして完全なファイルのリストができ、後で参照するためにマニフェストに書き込まれます。この内容は、ソース配布物のアーカイブを作成する際に使われます。

含めるファイルのデフォルトセットは *--no-defaults* で無効化でき、標準で除外するセットは *--no-prune* で無効化できます。

Distutils 自体のマニフェストテンプレートから、 **sdist** コマンドがどのようにして Distutils ソース配布物に含めるファイルのリストを作成するか見てみましょう:

1. distutils ディレクトリ、および distutils/command サブディレクトリの下にある全ての Python ソースファイルを含めます (これらの二つのディレクトリが、setup スクリプト下の *packages* オプションに記載されているからです — *setup* **スクリプトを書く** を参照してください)
2. README.txt, setup.py, および setup.cfg (標準のファイルセット) を含めます
3. test/test*.py (標準のファイルセット) を含めます
4. 配布物ルート下の *.txt を含めます (この処理で、 README.txt がもう一度見つかりますが、こうした冗長性は後で刈り取られます)
5. examples 下にあるサブツリー内で *.txt または *.py にマッチする全てのファイルを含めます
6. ディレクトリ名が examples/sample?/build にマッチするディレクトリ以下のサブツリー内にあるファイル全てを除外します— この操作によって、上の二つのステップでリストに含められたファイルが除外されることがあるので、マニフェストテンプレート内では recursive-include コマンドの後に prune コマンドを持ってくることが重要です
7. build ツリー全体、および RCS, CVS, .svn, .hg, .git, .bzr, _darcs ディレクトリ全てを除外します。

setup スクリプトと同様、マニフェストテンプレート中のディレクトリ名は常にスラッシュ区切りで表記します; Distutils は、こうしたディレクトリ名を注意深くプラットフォームでの標準的な表現に変換します。このため、マニフェストテンプレートは複数のオペレーティングシステムにわたって可搬性を持ちます。

4.2 マニフェスト (manifest) 関連のオプション

sdist コマンドが通常行う処理の流れは、以下のようになっています:

- マニフェストファイル MANIFEST が存在しなければ、 MANIFEST.in を読み込んでマニフェストファイルを作成します
- MANIFEST も MANIFEST.in もなければ、デフォルトのファイルセットだけでできたマニフェストファイルを作成します
- MANIFEST.in または (setup.py) が MANIFEST より新しければ、 MANIFEST.in を読み込んで MANIFEST を生成します
- (生成されたか、読み出された) MANIFEST 内にあるファイルのリストを使ってソース配布物アーカイブを作成します

上の動作は二種類のオプションを使って修正できます。まず、標準の “include” および “exclude” セットを無効化するには*--no-defaults* および *--no-prune* を使います

第 2 に、単にマニフェストを (再) 生成したいだけで、ソース配布物は作成したくない場合があるかもしれません:

```
python setup.py sdist --manifest-only
```

-o は *--manifest-only* のショートカットです。

第 5 章

ビルド済み配布物を作成する

“ビルド済み配布物” とは、おそらく皆さんが通常 “バイナリパッケージ” とか “インストーラ” (背景にしている知識によって違います) と考えているものです。とはいえ、配布物が必然的にバイナリ形式になるわけではありません。配布物には、 Python ソースコード、かつ/またはバイトコードが入るからです; また、我々はパッケージという呼び方もしません。すでに Python の用語として使っているからです (また、 “インストーラ” という言葉は主流のデスクトップシステム特有の用語です)

ビルド済み配布物は、モジュール配布物をインストール作業者にとってできるだけ簡単な状況にする方法です: ビルド済み配布物は、RPM ベースの Linux システムユーザにとってはバイナリ RPM 、Windows ユーザにとっては実行可能なインストーラ、 Debian ベースの Linux システムでは Debian パッケージ、などといった具合です。当然のことながら、一人の人間が世の中にある全てのプラットフォーム用にビルド済み配布物を作成できるわけではありません。そこで、 Distutils の設計は。開発者が自分の専門分野 — コードを書き、ソース配布物を作成する— に集中できる一方で、 **パッケージ作成者** *(packager)* と呼ばれる、開発者とエンドユーザとの中間に位置する人々がソースコード配布物を多くのプラットフォームにおけるビルド済み配布物に変換できるようになっています。

もちろん、モジュール開発者自身がパッケージ作成者かもしれません; また、パッケージを作成するのはオリジナルの作成者が利用できないプラットフォームにアクセスできるような “外部の” ボランティアかもしれませんし、ソース配布物を定期的に取り込んで、アクセスできるかぎりのプラットフォーム向けにビルド済み配布物を生成するソフトウェアかもしれません。作業を行うのが誰であれ、パッケージ作成者は setup スクリプトを利用し、**bdist** コマンドファミリを使ってビルド済み配布物を作成します。

単純な例として、Distutils ソースツリーから以下のコマンドを実行したとします:

```
python setup.py bdist
```

すると、Distutils はモジュール配布物 (ここでは Distutils 自体) をビルドし、” 偽の (fake)” インストールを (`build` ディレクトリで) 行います。そして現在のプラットフォームにおける標準の形式でビルド済み配布物を生成します。デフォルトのビルド済み形式とは、Unix では “ダム (dumb)” の tar ファイルで、 Windows ではシンプルな実行形式のインストーラになります。(tar ファイルは、特定の場所に手作業で解凍しないと動作しないので、 “ダム: 賢くない” 形式とみなします。)

従って、Unix システムで上記のコマンドを実行すると、`Distutils-1.0.`*`plat`*`.tar.gz` を作成します; この tarball を正しい場所で解凍すると、ちょうどソース配布物をダウンロードして `python setup.py install` を実行したのと同じように、正しい場所に Distutils がインストールされます。(“正しい場所 (right

place)" とは、ファイルシステムのルート下か、Python の *prefix* ディレクトリ下で、これは **bdist_dumb** に指定するコマンドで変わります; デフォルトの設定では、 *prefix* からの相対パスにインストールされるダム配布物が得られます。)

言うまでもなく、 pure Python 配布物の場合なら、 `python setup.py install` するのに比べて大して簡単になったとは言えません—しかし、非 pure 配布物で、コンパイルの必要な拡張モジュールを含む場合、拡張モジュールを利用できるか否かという大きな違いになりえます。また、 RPM パッケージや Windows 用の実行形式インストーラのような "スマートな" ビルド済み配布物を作成しておけば、たとえ拡張モジュールが一切入っていなくてもユーザにとっては便利になります。

bdist コマンドには、 *--formats* オプションがあります。これは **sdist** コマンドの場合に似ていて、生成したいビルド済み配布物の形式を選択できます: 例えば、

```
python setup.py bdist --format=zip
```

とすると、Unix システムでは、 `Distutils-1.0.`*plat*`.zip` を作成します— 先にも述べたように、Distutils をインストールするには、このアーカイブ形式をルートディレクトリ下で展開します。

ビルド済み配布物として利用できる形式を以下に示します:

形式	説明	注記
`gztar`	gzip 圧縮された tar ファイル (`.tar.gz`)	(1),(3)
`ztar`	compress 圧縮された tar ファイル (`.tar.Z`)	(3)
`tar`	tar ファイル (`.tar`)	(3)
`zip`	zip ファイル (`.zip`)	(4)
`rpm`	RPM 形式	(5)
`pkgtool`	Solaris **pkgtool** 形式	
`sdux`	HP-UX **swinstall** 形式	
`wininst`	Windows 用の自己展開形式 ZIP ファイル	(2),(4)

注記:

1. Unix でのデフォルト形式です
2. Windows でのデフォルト形式です

   ** to-do! **
3. 外部ユーティリティが必要です: **tar** と、 **gzip** または **bzip2** または **compress** のいずれか
4. 外部ユーティリティの **zip** か、 `zipfile` モジュール (Python 1.6 からは標準 Python ライブラリの一部になっています) が必要です
5. 外部ユーティリティの **rpm** 、バージョン 3.0.4 以上が必要です (バージョンを調べるには、 `rpm --version` とします)

bdist コマンドを使うとき、必ず *--formats* オプションを使わなければならないわけではありません; 自分の使いたい形式をダイレクトに実装しているコマンドも使えます。こうした **bdist** "サブコマンド (sub-command)" は、実際には類似のいくつかの形式を生成できます; 例えば、 **bdist_dumb** コマンドは、全ての "ダム" アーカイブ形式 (`tar`, `ztar`, `gztar`, および `zip`) を作成できますし、 **bdist_rpm** はバイナリ RPM とソース RPM の両方を生成できます。 **bdist** サブコマンドと、それぞれが生成する形式を以下に示します:

コマンド	形式
bdist_dumb	tar, ztar, gztar, zip
bdist_rpm	rpm, srpm
bdist_wininst	wininst

bdist_* コマンドについては、以下の節で詳しく述べます。

5.1 ダム形式のビルド済み配布物を作成する

5.2 RPM パッケージを作成する

RPM 形式は、Red Hat, SuSE, Mandrake といった、多くの一般的な Linux ディストリビューションで使われています。普段使っているのがこれらの環境のいずれか (またはその他の RPM ベースの Linux ディストリビューション) なら、同じディストリビューションを使っている他のユーザ用に RPM パッケージを作成するのはとるに足らないことでしょう。一方、モジュール配布物の複雑さや、Linux ディストリビューション間の違いにもよりますが、他の RPM ベースのディストリビューションでも動作するような RPM を作成できるかもしれません。

通常、モジュール配布物の RPM を作成するには、 **bdist_rpm** コマンドを使います:

```
python setup.py bdist_rpm
```

あるいは、 **bdist** コマンドを *--format* オプション付きで使います:

```
python setup.py bdist --formats=rpm
```

前者の場合、 RPM 特有のオプションを指定できます; 後者の場合、一度の実行で複数の形式を指定できます。両方同時にやりたければ、それぞれの形式について各コマンドごとにオプション付きで **bdist_*** コマンドを並べます:

```
python setup.py bdist_rpm --packager="John Doe <jdoe@example.org>" \
                bdist_wininst --target-version="2.0"
```

Distutils が setup スクリプトで制御されているのとほとんど同じく、 RPM パッケージの作成は、 `.spec` で制御されています。 RPM の作成を簡便に解決するため、 **bdist_rpm** コマンドでは通常、 setup スクリプトに与えた情報とコマンドライン、そして Distutils 設定ファイルに基づいて `.spec` ファイルを作成します。`.spec` ファイルの様々なオプションやセクション情報は、以下のようにして setup スクリプトから取り出されます:

RPM `.spec` ファイルのオプションまたはセクション	Distutils setup スクリプト内のオプション
Name	*name*
Summary (preamble 内)	*description*
Version	*version*
Vendor	*author* と *author_email*, または — & *maintainer* と *maintainer_email*
Copyright	*license*
Url	*url*
%description (セクション)	*long_description*

また、 .spec ファイル内の多くのオプションは、 setup スクリプト中に対応するオプションがありません。これらのほとんどは、以下に示す **bdist_rpm** コマンドのオプションで扱えます:

RPM .spec ファイルのオプションまたはセクション	bdist_rpm オプション	デフォルト値
Release	*release*	"1"
Group	*group*	"Development/Libraries"
Vendor	*vendor*	(上記参照)
Packager	*packager*	(none)
Provides	*provides*	(none)
Requires	*requires*	(none)
Conflicts	*conflicts*	(none)
Obsoletes	*obsoletes*	(none)
Distribution	*distribution_name*	(none)
BuildRequires	*build_requires*	(none)
Icon	*icon*	(none)

言うまでもなく、こうしたオプションをコマンドラインで指定するのは面倒だし、エラーの元になりますから、普通は setup.cfg に書いておくのがベストです — *setup* **設定ファイル** *(setup configuration file)* **を書く** 節を参照してください。沢山の Python モジュール配布物を配布したりパッケージ化したりしているのなら、配布物全部に当てはまるオプションを個人用の Distutils 設定ファイル (~/.pydistutils.cfg) に入れられます。

バイナリ形式の RPM パッケージを作成する際には三つの段階があり、 Distutils はこれら全ての段階を自動的に処理します:

1. RPM パッケージの内容を記述する .spec ファイルを作成します (.spec ファイルは setup スクリプトに似たファイルです; 実際、 setup スクリプトのほとんどの情報が .spec ファイルから引き揚げられます)
2. ソース RPM を作成します
3. "バイナリ (binary)" RPM を生成します (モジュール配布物に Python 拡張モジュールが入っているか否かで、バイナリコードが含まれることも含まれないこともあります)

通常、RPM は最後の二つのステップをまとめて行います; Distutils を使うと、普通は三つのステップ全てをまとめて行います。

望むなら、これらの三つのステップを分割できます。 **bdist_rpm** コマンドに *--spec-only* を指定すれば、単に .spec を作成して終了します; この場合、 .spec ファイルは "配布物ディレクトリ (distribution directory)"— 通常は dist/ に作成されますが、 *--dist-dir* オプションで変更することもできます。(通常、 .spec ファイルは "ビルドツリー (build tree)"、すなわち **build_rpm** が作成する一時ディレクトリの中から引き揚げられます。)

5.3 Windows インストーラを作成する

実行可能なインストーラは、Windows 環境ではごく自然なバイナリ配布形式です。インストーラは結構なグラフィカルユーザインタフェースを表示して、モジュール配布物に関するいくつかの情報を setup スクリプト内のメタデータから取り出して示し、ユーザがいくつかのオプションを選んだり、インストールを決行するか取りやめるか選んだりできるようにします。

メタデータは setup スクリプトから取り出されるので、Windows インストーラの作成は至って簡単で、以下を実行するだけです:

```
python setup.py bdist_wininst
```

あるいは、 **bdist** コマンドを *--formats* オプション付きで実行します:

```
python setup.py bdist --formats=wininst
```

(pure Python モジュールとパッケージだけの入った) pure モジュール配布物の場合、作成されるインストーラは実行バージョンに依存しない形式になり、 `foo-1.0.win32.exe` のような名前になります。 pure モジュールの Windows インストーラは Unix や Mac OS X でも作成できます。

非 pure 配布物の場合、拡張モジュールは Windows プラットフォーム上だけで作成でき、Python のバージョンに依存したインストーラになります。インストーラのファイル名もバージョン依存性を反映して、`foo-1.0.win32-py2.0.exe` のような形式になります。従って、サポートしたい全てのバージョンの Python に対して、別々のインストーラを作成しなければなりません。

インストーラは、ターゲットとなるシステムにインストールを実行した後、 pure モジュールを通常 (normal) モードと最適化 (optimizing) モードでバイトコード (*bytecode*) にコンパイルしようと試みます。何らかの理由があってコンパイルさせたくなければ、 **bdist_wininst** コマンドを*--no-target-compile* かつ/または *--no-target-optimize* オプション付きで実行します。

デフォルトでは、インストーラは実行時にクールな “Python Powered” ロゴを表示しますが、自作の 152x161 ビットマップ画像も指定できます。画像は Windows の `.bmp` ファイル形式でなくてはならず、 *--bitmap* オプションで指定します。

インストーラを起動すると、デスクトップの背景ウィンドウ上にでっかいタイトルも表示します。タイトルは配布物の名前とバージョン番号から作成します。 *--title* オプションを使えば、タイトルを別のテキストに変更できます。

インストーラファイルは “配布物ディレクトリ (distribution directory)” — 通常は `dist/` に作成されますが、*--dist-dir* オプションで指定することもできます。

5.4 Windows でのクロスコンパイル

Python 2.6 から、 distutils は Windows プラットフォーム間でのクロスコンパイルに対応しました。これによって、必要なツールがインストールされていれば、 32bit 版の Windows で 64bit 版の拡張を作成したり、その逆が可能になりました。

別のプラットフォーム用にビルドするには、 build コマンドの *--plat-name* オプションを指定します。有効な値は、現在のところ、 ‘win32’, ‘win-amd64’, ‘win-ia64’ です。例えば、次のようにして 32bit 版の Windows で 64bit 版の拡張をビルドできます。

```
python setup.py build --plat-name=win-amd64
```

Windows インストーラもこのオプションをサポートしています。なので次のコマンドを実行すると:

```
python setup.py build --plat-name=win-amd64 bdist_wininst
```

64bit のインストーラを 32bit の Windows で作成できます。

クロスコンパイルするためには、Python のソースコードをダウンロードして Python 自体をターゲットのプラットフォーム用にクロスコンパイルしなければなりません。Python のバイナリインストールからではクロスコンパイルできません。(他のプラットフォーム用の .lib などのファイルが含まれないからです。) 具体的に言えば、拡張のクロスコンパイルができるようになるには、 32bit OS のユーザーは Visual Studio 2008 を使って Python ソースツリー内の `PCBuild/PCbuild.sln` ソリューションファイルを開き、“x64” コンフィギュレーションで ‘pythoncore’ プロジェクトをビルドしなければなりません。

デフォルトでは、 Visual Studio 2008 は 64bit のコンパイラーなどのツールをインストールしないことに注意してください。Visual Studio セットアッププロセスを再実行して、それらのツールを選択する必要があるでしょう。(コントロールパネル -> [追加と削除] から簡単に既存のインストールをチェック、修正できます。)

5.4.1 インストール後実行スクリプト (postinstallation script)

Python 2.3 からは、インストール実行後スクリプトを *--install-script* オプションで指定できるようになりました。スクリプトはディレクトリを含まないベースネーム (basename) で指定しなければならず、スクリプトファイル名は setup 関数の scripts 引数中に挙げられていなければなりません。

指定したスクリプトは、インストール時、ターゲットとなるシステム上で全てのファイルがコピーされた後に実行されます。このとき `argv[1]` を *-install* に設定します。また、アンインストール時には、ファイルを削除する前に `argv[1]` を *-remove* に設定して実行します。

Windows インストーラでは、インストールスクリプトは埋め込みで実行され、全ての出力 (`sys.stdout`、 `sys.stderr`) はバッファにリダイレクトされ、スクリプトの終了後に GUI 上に表示されます。

インストールスクリプトでは、インストール/アンインストールのコンテキストで特に有用ないくつかの機能を、追加の組み込み関数として利用することができます。

directory_created(*path*)
file_created(*path*)
これらの関数は、インストール後実行スクリプトがディレクトリやファイルを作成した際に呼び出さなければなりません。この関数はアンインストーラに作成された *path* を登録し、配布物をアンインストールする際にファイルが消されるようにします。安全を期すために、ディレクトリは空の時にのみ削除されます。

get_special_folder_path(*csidl_string*)
この関数は、「スタートメニュー」や「デスクトップ」といった、 Windows における特殊なフォルダ位置を取得する際に使えます。この関数はフォルダのフルパスを返します。 *csidl_string* は以下の文字列のいずれかでなければなりません:

```
"CSIDL_APPDATA"

"CSIDL_COMMON_STARTMENU"
"CSIDL_STARTMENU"

"CSIDL_COMMON_DESKTOPDIRECTORY"
"CSIDL_DESKTOPDIRECTORY"

"CSIDL_COMMON_STARTUP"
"CSIDL_STARTUP"

"CSIDL_COMMON_PROGRAMS"
```

`"CSIDL_PROGRAMS"`

`"CSIDL_FONTS"`

該当するフォルダを取得できなかった場合、 `OSError` が送出されます。

どの種類のフォルダを取得できるかは、特定の Windows のバージョンごとに異なります。また、おそらく設定によっても異なるでしょう。詳細については、`SHGetSpecialFolderPath()` 関数に関する Microsoft のドキュメントを参照してください。

create_shortcut(*target*, *description*, *filename*[, *arguments*[, *workdir*[, *iconpath*[, *iconindex*]]]])

この関数はショートカットを作成します。 *target* はショートカットによって起動されるプログラムへのパスです。 *description* はショートカットに対する説明です。 *filename* はユーザから見えるショートカットの名前です。コマンドライン引数があれば、 *arguments* に指定します。 *workdir* はプログラムの作業ディレクトリです。 *iconpath* はショートカットのためのアイコンが入ったファイルで、*iconindex* はファイル *iconpath* 中のアイコンへのインデクスです。これについても、詳しくは `IShellLink` インタフェースに関する Microsoft のドキュメントを参照してください。

5.5 Vista User Access Control (UAC)

Python 2.6 から、 bdist_wininst は *--user-access-control* オプションをサポートしています。デフォルトは 'none' (UAC 制御をしないことを意味します) で、それ以外の有効な値は 'auto' (Python が全ユーザー用にインストールされていれば UAC 昇格を行う)、'force' (常に昇格を行う) です。

第 6 章

パッケージインデクスに登録する

Python パッケージインデクス (Python Package Index, PyPI) は、 distutils でパッケージ化された配布物に関するメタデータを保持しています。配布物のメタデータをインデクスに提出するには、 Distutils のコマンド **register** を使います。 **register** は以下のように起動します:

```
python setup.py register
```

Distutils は以下のようなプロンプトを出します:

```
running register
We need to know who you are, so please choose either:
 1. use your existing login,
 2. register as a new user,
 3. have the server generate a new password for you (and email it to you), or
 4. quit
Your selection [default 1]:
```

注意: ユーザ名とパスワードをローカルの計算機に保存しておくと、このメニューは表示されません。

まだ PyPI に登録したことがなければ、まず登録する必要があります。この場合選択肢 2 番を選び、リクエストされた詳細情報を入力してゆきます。詳細情報を提出し終えると、登録情報の承認を行うためのメールを受け取るはずです。

すでに登録を行ったことがあれば、選択肢 1 を選べます。この選択肢を選ぶと、PyPI ユーザ名とパスワードを入力するよう促され、**register** がメタデータをインデクスに自動的に提出します。

配布物の様々なバージョンについて、好きなだけインデクスへの提出を行ってかまいません。特定のバージョンに関するメタデータを入れ替えたければ、再度提出を行えば、インデクス上のデータが更新されます。

PyPI は提出された配布物の (名前、バージョン) の各組み合わせについて記録を保持しています。ある配布物名について最初に情報を提出したユーザが、その配布物名のオーナ (owner) になります。オーナは **register** コマンドか、web インタフェースを介して変更を提出できます。オーナは他のユーザをオーナやメンテナとして指名できます。メンテナはパッケージ情報を編集できますが、他の人をオーナやメンテナに指名することはできません。

デフォルトでは、 PyPI はあるパッケージについて全てのバージョンを表示します。特定のバージョンを非表示にしたければ、パッケージの Hidden プロパティを yes に設定します。この値は web インタフェースで編集しなければなりません。

6.1 .pypirc ファイル (The .pypirc file)

.pypirc ファイルのフォーマットを示します。

```
[distutils]
index-servers =
  pypi

[pypi]
repository: <repository-url>
username: <username>
password: <password>
```

repository は省略可能で、デフォルトでは http://www.python.org/pypi になります。

別のサーバーを定義した場合は、新しいセクションを作成します。

```
[distutils]
index-servers =
  pypi
  other

[pypi]
repository: <repository-url>
username: <username>
password: <password>

[other]
repository: http://example.com/pypi
username: <username>
password: <password>
```

そうすると、 register コマンドに -r オプションをつけて実行できます。

```
python setup.py register -r http://example.com/pypi
```

もしくは、セクション名を使うこともできます。

```
python setup.py register -r other
```

第 7 章

Uploading Packages to the Package Index

バージョン 2.5 で追加. Python Package Index (PyPI) は、パッケージ情報に加えて、作者が望むのであればパッケージデータを置くこともできます。distutils の **upload** コマンドは配布物を PyPI にアップロードします。

このコマンドは一つ以上の配布物ファイルをビルドした直後に呼び出されます。例えば、次のコマンド

```
python setup.py sdist bdist_wininst upload
```

は、ソース配布物と Windows のインストーラを PyPI にアップロードします。以前に `setup.py` を実行してビルドした配布物もアップロード対象になるけれども、アップロードされるのは **upload** コマンドと同時に指定された配布物だけだということに注意してください。

upload コマンドは、 `$HOME/.pypirc` ファイル (詳しくは *.pypirc* **ファイル** *(The .pypirc file)* セクションをご覧下さい) の、ユーザー名、パスワードとリポジトリ URL を利用します。

`--repository=*url*` オプションを使って別の PyPI サーバーを指定することができます。

```
python setup.py sdist bdist_wininst upload -r http://example.com/pypi
```

複数のサーバーを定義することについて、より詳しい情報は *.pypirc* **ファイル** *(The .pypirc file)* を参照してください。

`--sign` オプションで、アップロードする各ファイルに GPG (GNU Privacy Guard) を使うことができます。**gpg** プログラムが環境変数 `PATH` から実行可能である必要があります。署名にどの鍵を使うかを、*`--identity=*name*`* で指定することもできます。

他の **upload** のオプションには、 *`--repository=`* (*url* はサーバーの URL), *`--repository=`* (*section* は `$HOME/.pypirc` のセクション名), *`--show-response`* (アップロードの問題をデバッグするために、PyPI サーバーからの全てのレスポンスを表示する) があります。

第 8 章

例

この章は distutils を使い始めるのに役立つ幾つかの基本的な例を提供します。distutils を使うための追加の情報は Distutils Cookbook で見つけることができます。

参考:

Distutils Cookbook distutils をもっと制御するためのレシピ集

8.1 pure Python 配布物 (モジュール形式)

単に二つのモジュール、特定のパッケージに属しないモジュールを配布するだけなら、setup スクリプト中で *py_modules* オプションを使って個別に指定できます。

もっとも単純なケースでは、二つのファイル: setup スクリプト自体と、配布したい単一のモジュール、この例では `foo.py` について考えなければなりません:

```
<root>/
        setup.py
        foo.py
```

(この節の全ての図において、 *<root>* は配布物ルートディレクトリを参照します。) この状況を扱うための最小の setup スクリプトは以下のようになります:

```
from distutils.core import setup
setup(name='foo',
      version='1.0',
      py_modules=['foo'],
      )
```

配布物の名前は *name* オプションで個々に指定し、配布されるモジュールの一つと配布物を同じ名前にする必要はないことに注意してください (とはいえ、この命名方法はよいならわしでしょう)。ただし、配布物名はファイル名を作成するときに使われるので、文字、数字、アンダースコア、ハイフンだけで構成しなければなりません。

py_modules はリストなので、もちろん複数のモジュールを指定できます。例えば、モジュール `foo` と `bar` を配布しようとしているのなら、 setup スクリプトは以下のようになります:

```
<root>/
        setup.py
        foo.py
        bar.py
```

また、セットアップスクリプトは以下のようになります.

```
from distutils.core import setup
setup(name='foobar',
      version='1.0',
      py_modules=['foo', 'bar'],
      )
```

モジュールのソースファイルは他のディレクトリに置けますが、そうしなければならないようなモジュールを沢山持っているのなら、モジュールを個別に列挙するよりもパッケージを指定した方が簡単でしょう。

8.2 pure Python 配布物 (パッケージ形式)

二つ以上のモジュールを配布する場合、とりわけ二つのパッケージに分かれている場合、おそらく個々のモジュールよりもパッケージ全体を指定する方が簡単です。たとえモジュールがパッケージ内に入っていなくても状況は同じで、その場合はルートパッケージにモジュールが入っていると Distutils に教えることができ、他のパッケージと同様にうまく処理されます (ただし、 `__init__.py` があってはなりません)。

最後の例で挙げた setup スクリプトは、

```
from distutils.core import setup
setup(name='foobar',
      version='1.0',
      packages=[''],
      )
```

のようにも書けます (空文字はルートパッケージを意味します)

これら二つのファイルをサブディレクトリ下に移動しておいて、インストール先はルートパッケージのままにしておきたい、例えば:

```
<root>/
        setup.py
        src/      foo.py
                  bar.py
```

のような場合には、パッケージ名にはルートパッケージをそのまま指定しておきますが、ルートパッケージに置くソースファイルがどこにあるかを Distutils に教えなければなりません:

```
from distutils.core import setup
setup(name='foobar',
      version='1.0',
      package_dir={'': 'src'},
      packages=[''],
      )
```

もっと典型的なケースでは、複数のモジュールを同じパッケージ (またはサブパッケージ) に入れて配布しようと思うでしょう。例えば、 `foo` と `bar` モジュールがパッケージ `foobar` に属する場合、ソースツリーをレイアウトする一案として、以下が考えられます。

```
<root>/
        setup.py
        foobar/
                __init__.py
                foo.py
                bar.py
```

実際、 Distutils ではこれをデフォルトのレイアウトとして想定していて、setup スクリプトを書く際にも最小限の作業しか必要ありません:

```
from distutils.core import setup
setup(name='foobar',
      version='1.0',
      packages=['foobar'],
      )
```

モジュールを入れるディレクトリをパッケージの名前にしたくない場合、ここでも *package_dir* オプションを使う必要があります。例えば、パッケージ `foobar` のモジュールが `src` に入っているとします:

```
<root>/
        setup.py
        src/
                __init__.py
                foo.py
                bar.py
```

適切な setup スクリプトは、

```
from distutils.core import setup
setup(name='foobar',
      version='1.0',
      package_dir={'foobar': 'src'},
      packages=['foobar'],
      )
```

のようになるでしょう。

また、メインパッケージ内のモジュールを配布物ルート下に置くことがあるかもしれません:

```
<root>/
        setup.py
        __init__.py
        foo.py
        bar.py
```

この場合、 setup スクリプトは

```
from distutils.core import setup
setup(name='foobar',
      version='1.0',
      package_dir={'foobar': ''},
      packages=['foobar'],
      )
```

のようになるでしょう。 (空文字列も現在のディレクトリを表します。)

サブパッケージがある場合、 *packages* で明示的に列挙しなければなりませんが、 *package_dir* はサブパッケージへのパスを自動的に展開します。 (別の言い方をすれば、 Distutils はソースツリーを **走査せ**

ず、どのディレクトリが Python パッケージに相当するのかを `__init__.py` files. を探して調べようとします。) このようにして、デフォルトのレイアウトはサブパッケージ形式に展開されます:

```
<root>/
        setup.py
        foobar/
                __init__.py
                foo.py
                bar.py
                subfoo/
                        __init__.py
                        blah.py
```

対応する setup スクリプトは以下のようになります。

```
from distutils.core import setup
setup(name='foobar',
      version='1.0',
      packages=['foobar', 'foobar.subfoo'],
      )
```

(ここでも、 *package_dir* を空文字列にすると現在のディレクトリを表します。)

8.3 単体の拡張モジュール

拡張モジュールは、 *ext_modules* オプションを使って指定します。 *package_dir* は、拡張モジュールのソースファイルをどこで探すかには影響しません; pure Python モジュールのソースのみに影響します。もっとも単純なケースでは、単一の C ソースファイルで書かれた単一の拡張モジュールは:

```
<root>/
        setup.py
        foo.c
```

になります。

`foo` 拡張をルートパッケージ下に所属させたい場合、 setup スクリプトは

```
from distutils.core import setup
from distutils.extension import Extension
setup(name='foobar',
      version='1.0',
      ext_modules=[Extension('foo', ['foo.c'])],
      )
```

になります。

同じソースツリーレイアウトで、この拡張モジュールを `foopkg` の下に置き、拡張モジュールの名前を変えるには:

```
from distutils.core import setup
from distutils.extension import Extension
setup(name='foobar',
      version = '1.0',
      ext_modules=[Extension('foopkg.foo', ['foo.c'])],
      )
```

のようにします。

第 9 章

Distutilsの拡張

Distutils は様々な方法で拡張できます。ほとんどの拡張は存在するコマンドを新しいコマンドで置換する形でおこなわれます。新しいコマンドはたとえば存在するコマンドを置換して、そのコマンドでパッケージをどう処理するかの細部を変更することでプラットフォーム特有のパッケージ形式をサポートするために書かれているかもしれません

ほとんどの distutils の拡張は存在するコマンドを変更したい `setup.py` スクリプト中で行われます。ほとんどはパッケージにコピーされるファイル拡張子を `.py` の他に、いくつか追加するものです。

ほとんどの distutils のコマンド実装は `distutils.cmd` の `distutils.cmd.Command` クラスのサブクラスとして実装されています。新しいコマンドは `Command` を直接継承し、置換するコマンドでは置換対象のコマンドのサブクラスにすることで `Command` を間接的に継承します。コマンドは `Command` から派生したものである必要があります。

9.1 新しいコマンドの統合

新しいコマンド実装を統合するにはいくつかの方法があります。一番難しいものは新機能を distutils 本体に取り込み、それのサポートを提供する Python のバージョンが出ることを待つ (そして使う) ことです。これは様々な理由で本当に難しいことです。

もっとも一般的な、そしておそらくほとんどの場合にもっとも妥当な方法は、新しい実装をあなたの `setup.py` スクリプトに取り込み、`distutils.core.setup()` 関数でそれらを使うようにすることです。

```
from distutils.command.build_py import build_py as _build_py
from distutils.core import setup

class build_py(_build_py):
    """Specialized Python source builder."""

    # implement whatever needs to be different...

setup(cmdclass={'build_py': build_py},
      ...)
```

このアプローチは新実装をある特定のパッケージで利用したい時、そのパッケージに興味をもつ人全員がコマンドの新実装を必要とする時にもっとも価値があります。

Python 2.4 から、インストールされた Python を変更せずに、既存の `setup.py` スクリプトをサポートするための 3 つめの選択肢が利用できるようになりました。これは追加のパッケージングシステムのサポートを追加するサードパーティ拡張を提供することを想定していますが、これらのコマンドは distutils が利用されている何にでも利用可能です。新しい設定オプション *command_packages* (コマンドラインオプション *--command-packages*) は、コマンド実装モジュールを検索する追加のパッケージを指定するために利用できます。 distutils の全てのオプションと同様に、このオプションもコマンドラインまたは設定ファイルで指定できます。このオプションは設定ファイル中では `[global]` セクションか、コマンドラインのコマンドより前でだけ設定できます。設定ファイル中で指定する場合、コマンドラインで上書きすることができます。空文字列を指定するとデフォルト値が使われます。これはパッケージと一緒に提供する設定ファイルでは指定しないでください。

この新オプションによってコマンド実装を探すためのパッケージをいくつでも追加することができます。複数のパッケージ名はコンマで区切って指定します。指定がなければ、検索は `distutils.command` パッケージのみで行われます。ただし `setup.py` がオプション *--command-packages distcmds,buildcmds* で実行されている場合には、パッケージは `distutils.command` 、 `distcmds` 、そして `buildcmds` を、この順番で検索します。新コマンドはコマンドと同じ名前のモジュールに、コマンドと同じ名前のクラスで実装されていると想定しています。上のコマドラインオプションの例では、コマンド **bdist_openpkg** は、 `distcmds.bdist_openpkg.bdist_openpkg` か、`buildcmds.bdist_openpkg.bdist_openpkg` で実装されるかもしれません。

9.2 配布物の種類を追加する

配布物 (`dist/` ディレクトリの中のファイル) を作成するコマンドは、 **upload** がその配布物を PyPI にアップロードできるように、 `(command, filename)` のペアを `self.distributions.dist_files` に追加する必要があります。ペア中の *filename* はパスに関する情報を持たず、単にファイル名だけを持ちます。 dry-run モードでも、何が作成されたかを示すために、同じペアが必要になります。

第 10 章

コマンドリファレンス

10.1 モジュールをインストールする: install コマンド群

install コマンドは最初にビルドコマンドを実行しておいてから、サブコマンド **install_lib** を実行します。**install_data** and **install_scripts**.

10.1.1 install_data

このコマンドは配布物中に提供されている全てのデータファイルをインストールします。

10.1.2 install_scripts

このコマンドは配布物中の全ての (Python) スクリプトをインストールします。

10.2 ソースコード配布物を作成する: sdist command

マニフェストテンプレート関連のコマンドを以下に示します:

コマンド	説明
include pat1 pat2 ...	列挙されたパターンのいずれかにマッチする全てのファイルを対象に含めます
exclude pat1 pat2 ...	列挙されたパターンのいずれかにマッチする全てのファイルを対象から除外します
recursive-include dir pat1 pat2 ...	*dir* 下にある、列挙されたパターンのいずれかにマッチする全てのファイルを対象に含めます
recursive-exclude dir pat1 pat2 ...	*dir* 下にある、列挙されたパターンのいずれかにマッチする全てのファイルを対象から除外します
global-include pat1 pat2 ...	ソースツリー下にある、列挙されたパターンのいずれかにマッチする全てのファイルを対象に含めます
global-exclude pat1 pat2 ...	ソースツリー下にある、列挙されたパターンのいずれかにマッチする全てのファイルを対象から除外します
prune dir	*dir* 下の全てのファイルを除外します
graft dir	*dir* 下の全てのファイルを含めます

ここでいうパターンとは、Unix 式の “glob” パターンです: `*` は全ての正規なファイル名文字列に一致し、 `?` は正規なファイル名文字一字に一致します。また、 `[range]` は、 *range* の範囲 (例えば、 `a=z`, `a-zA-Z`, `a-f0-9_.`) 内にある、任意の文字にマッチします。 “正規なファイル名文字” の定義は、プラットフォームごとに特有のものです: Unix ではスラッシュ以外の全ての文字です; Windows では、バックラッシュとコロン以外です。

** Windows はまだサポートされていません **

第 11 章

API リファレンス

11.1 distutils.core — Distutils のコア機能

Distutils を使うためにインストールする必要がある唯一のモジュールが distutils.core モジュールです。 setup() 関数 (セットアップスクリプトから呼び出されます) を提供します。間接的に distutils.dist.Distribution クラスと distutils.cmd.Command クラスを提供します。

distutils.core.**setup**(*arguments*)

全てを実行する基本的な関数で、Distutils でできるほとんどのことを実行します。

setup 関数はたくさんの引数をとります。以下のテーブルにまとめます。

引数名	値	型
name	パッケージの名前	文字列
version	パッケージのバージョン番号	distutils.version を参照してください
description	1 行で書いたパッケージ解説	文字列
long_description	パッケージの長い解説	文字列
author	パッケージ作者の名前	文字列
author_email	パッケージ作者の email アドレス	文字列
maintainer	現在のメンテナの名前 (パッケージ作者と異なる場合)	文字列
maintainer_email	現在のメンテナの email アドレス (パッケージ作者と異なる場合)	
url	パッケージの URL(ホームページ)	URL
download_url	パッケージダウンロード用 URL	URL
packages	distutils が操作する Python パッケージのリスト	文字列のリスト
py_modules	distutils が操作する Python モジュールのリスト	文字列のリスト
scripts	ビルドおよびインストールする単体スクリプトファイルのリスト	文字列のリスト
ext_modules	ビルドする拡張モジュール	distutils.core.Extension インスタンスのリスト
classifiers	パッケージのカテゴリのリスト	利用可能なカテゴリ一覧は http://cheeseshop.python.org/pypi?:action=list_classifiers にあります。
distclass	使用する Distribution クラス	distutils.core.Distribution のサブクラス
script_name	setup.py スクリプトの名前 - デフォルトでは sys.argv[0]	文字列
script_args	セットアップスクリプトの引数	文字列のリスト
options	セットアップスクリプトのデフォルト引数	文字列
license	パッケージのライセンス	文字列
keywords	説明用メタデータ。 **PEP 314** を参照してください	
platforms		
cmdclass	コマンド名から Command サブクラスへのマッピング	辞書
data_files	インストールするデータファイルのリスト	リスト
package_dir	パッケージからディレクトリ名へのマッピング	辞書

distutils.core.**run_setup**(*script_name*[, *script_args=None*, *stop_after='run'*])

制御された環境でセットアップスクリプトを実行し、いろいろなものを操作する distutils.dist.Distribution クラスのインスタンスを返します。これはディストリビューションのメタデータ (キーワード引数 *script* として関数 setup() に渡される) を参照したり、設定ファイルやコマンドラインの内容を調べる時に便利です。

script_name は execfile() で実行されるファイルです。 sys.argv[0] は、呼び出しのために *script_name* と置換されます。 *script_args* は文字列のリストです。もし提供されていた場合、sys.argv[1:] は、呼び出しのために *script_args* で置換されます。

stop_after はいつ動作を停止するか関数 setup() に伝えます。とりうる値は:

値	説明
init	Distribution インスタンスを作成し、キーワード引数を setup() に渡したあとに停止する。
config	設定ファイルをパーズしたあと停止する (そしてそのデータは Distribution インスタンスに保存される)。
command-line	コマンドライン (sys.argv[1:] または *script_args*) がパーズされたあとに停止する (そしてそのデータは Distribution インスタンスに保存される)。
run	全てのコマンドを実行したあとに停止する (関数 setup() を通常の方法で呼び出した場合と同じ)。デフォルト値。

これに加えて、 distutils.core モジュールは他のモジュールにあるいくつかのクラスを公開しています。

- Extension は distutils.extension から。
- Command は distutils.cmd から。
- Distribution は distutils.dist から。

それぞれの簡単な説明を以下に記します。完全な説明についてはそれぞれのモジュールをごらんください。

class distutils.core.**Extension**

Extension クラスは、セットアップスクリプト中で C または C++拡張モジュールを表します。コンストラクタで以下のキーワード引数をとります。

引数名	値	型
name	拡張のフルネーム (パッケージを含む) — ファイル名やパス名では **なく** 、Python のピリオド区切りの名前	文字列
sources	ソースファイル名のリスト。配布物ルートディレクトリ (setup スクリプトのある場所) からの相対パス、プラットフォーム独立のため Unix 形式 (スラッシュで区切る) で記述します。ソースファイルは C, C++, SWIG (.i)、特定プラットフォーム用のリソースファイル、その他 **build_ext** コマンドがソースファイルだと認識するどの形式でもありえます。	文字列
include_dirs	C/C++ヘッダファイルを検索するディレクトリのリスト (プラットフォーム独立のため Unix 形式で記述する)	文字列
define_macros	定義するマクロのリスト; それぞれのマクロは 2 要素のタプル `(name, value)` で定義されます。 *value* には定義しようとしている文字列、または内容なしで定義する場合は `None` (ソースコード中で `#define FOO` と書く、または Unix C コンパイラのコマンドラインで *-DFOO* を指定するのと等価です) を指定します。	(文字列, 文字列) または (文字列, `None`) のタプル
undef_macros	定義を消すマクロのリスト	文字列
library_dirs	リンク時に C/C++ライブラリを検索するディレクトリのリスト	文字列
libraries	リンクするライブラリ名のリスト (ファイル名やパスではない)	文字列
runtime_library_dirs	実行時 (shared extension では、拡張が読み込まれる時) に C/C++ライブラリを探索するディレクトリのリスト	文字列
extra_objects	追加でリンクするファイル ('sources' に対応するコードが含まれていないファイル、バイナリ形式のリソースファイルなど) のリスト	文字列
extra_compile_args	'sources' のソースをコンパイルする時に追加するプラットフォーム特有またはコンパイラ特有の情報コマンドラインを利用できるプラットホームとコンパイラでは、これは通常コマンドライン引数のリストですが、他のプラットホームでも、それは何かに使えます。	文字列
extra_link_args	オブジェクトファイルをリンクして拡張 (または新しい Python インタプリタ) を作る時に追加するプラットフォーム特有またはコンパイラ特有の情報 'extra_compile_args' に似た実装です。	文字列
export_symbols	shared extension からエクスポートされるシンボルのリスト。全てのプラットフォームでは使われず、Python 拡張 (典型的には `init` + extension_name という 1 つのシンボルだけエクスポートする) に一般的に必要なものでもない。	文字列
depends	拡張が依存するファイルのリスト	文字列
language	拡張の言語 (例: `'c'`, `'c++'`, `'objc'`)。指定しなければソースの拡張子で検出される。	文字列

class `distutils.core.`**Distribution**

`Distribution` は Python ソフトウェアパッケージをどのようにビルド、インストール、パッケージするかを定義する。

Distribution のコンストラクタが取りうるキーワード引数のリストに関しては、 setup() 関数を見てください。setup() は Distribution のインスタンスを作ります。

class distutils.core.**Command**
Command クラス (そのサブクラスのインスタンス) は distutils のあるコマンドを実装します。

11.2 distutils.ccompiler — CCompiler ベースクラス

このモジュールは CCompiler クラスの抽象ベースクラスを提供します。CCompiler のインスタンスはプロジェクトにおける全てのコンパイルおよびリンクに使われます。コンパイラのオプションを設定するためのメソッドが提供されます — マクロ定義、include ディレクトリ、リンクパス、ライブラリなど。

このモジュールは以下の関数を提供します。

distutils.ccompiler.**gen_lib_options**(*compiler*, *library_dirs*, *runtime_library_dirs*, *libraries*)
ライブラリを探索するディレクトリ、特定のライブラリとのリンクをするためのリンカオプションを生成します。 *libraries* と *library_dirs* はそれぞれライブラリ名 (ファイル名ではありません!) のリストと、探索ディレクトリのリストです。compiler で利用できるコマンドラインオプションのリスト (指定されたフォーマット文字列に依存します) を返します。

distutils.ccompiler.**gen_preprocess_options**(*macros*, *include_dirs*)
C プリプロセッサオプション (*-D*, *-U*, *-I*) を生成します。これらは少なくとも 2 つのコンパイラで利用可能です。典型的な Unix のコンパイラと、VisualC++です。*macros* は 1 または 2 要素のタプルで (name,) は *name* マクロの削除 (-U) を意味し、*(name,value)* は *name* マクロを *value* として定義 (*-D*) します。 *include_dirs* はディレクトリ名のリストで、ヘッダファイルのサーチパスに追加されます (*-I*)。 Unix のコンパイラと、Visual C++で利用できるコマンドラインオプションのリストを返します。

distutils.ccompiler.**get_default_compiler**(*osname*, *platform*)
指定されたプラットフォームのデフォルトコンパイラを返します。

問い合わせの *osname* は Python 標準の OS 名 (os.name で返されるもの) のひとつであるべきで、*platform* は sys.platform で返される共通の値です。

パラメータが指定されていない場合のデフォルト値は os.name と sys.platform です。

distutils.ccompiler.**new_compiler**(*plat=None*, *compiler=None*, *verbose=0*, *dry_run=0*, *force=0*)
指定されたプラットフォーム/コンパイラの組み合わせ向けに、 CCompiler サブクラスのインスタンスを生成するファクトリ関数です。 *plat* のデフォルト値は os.name (例: 'posix', 'nt'), *compiler*)、*compiler* のデフォルト値はプラトフォームのデフォルトコンパイラです。現在は'posix' と 'nt' だけがサポートされています、デフォルトのコンパイラは “traditional Unix interface” (UnixCCompiler クラス) と、 Visual C++ (MSVCCompiler クラス) です。 Windows で Unix コンパイラオブジェクトを要求することも、Unix で Microsoft コンパイラオブジェクトを要求することも可能です。 *compiler* 引数を与えると *plat* は無視されます。

distutils.ccompiler.**show_compilers**()
利用可能なコンパイラのリストを表示します (**build**, **build_ext**, **build_clib** の、 *--help-compiler* オプションで使われます。)

class distutils.ccompiler.**CCompiler**([*verbose=0, dry_run=0, force=0*])

抽象ベースクラス CCompiler は実際のコンパイラクラスで実装される必要のあるインタフェースを定義しています。このクラスはコンパイラクラスで利用されるユーティリティメソッドも定義しています。

コンパイラ抽象クラスの基本的な前提は、各インスタンスはあるプロジェクトをビルドするときの全コンパイル/リンクで利用できるということです。そこで、コンパイルとリンクステップで共通する属性 — インクルードディレクトリ、マクロ定義、リンクするライブラリなど — はコンパイラインスタンスの属性になります。どのように各ファイルが扱われるかを変更できるように、ほとんどの属性はコンパイルごと、またはリンクごとに与えることができます。

各サブクラスのコンストラクタは Compiler クラスのインスタンスを作ります。フラグは *verbose* (冗長な出力を表示します)、*dry_run* (実際にはそのステップを実行しません)、そして *force* (依存関係を無視して全て再ビルドします) です。これらのフラグは全てデフォルト値が 0 (無効) になっています。CCompiler またはサブクラスを直接インスタンス化したくない場合には、かわりに distutils.CCompiler.new_compiler() ファクトリ関数を利用してください。

以下のメソッドで、Compiler クラスのインスタンスが使うコンパイラオプションを手動で変更できます。

add_include_dir(*dir*)

dir をヘッダファイル探索ディレクトリのリストに追加します。コンパイラは add_include_dir() を呼び出した順にディレクトリを探索するよう指定されます。

set_include_dirs(*dirs*)

探索されるディレクトリのリストを *dirs* (文字列のリスト) に設定します。先に実行された add_include_dir() は上書きされます。後で実行する add_include_dir() は set_include_dirs() のリストにディレクトリを追加します。これはコンパイラがデフォルトで探索する標準インクルードディレクトリには影響しません。

add_library(*libname*)

libname をコンパイラオブジェクトによるリンク時に使われるライブラリのリストに追加します。 *libname* はライブラリを含むファイル名ではなく、ライブラリそのものの名前です: 実際のファイル名はリンカ、コンパイラ、またはコンパイラクラス (プラットフォームに依存します) から推測されます。

リンカは add_library() と set_library() で渡された順にライブラリをリンクしようとします。ライブラリ名が重なることは問題ありません。リンカは指定された回数だけライブラリとリンクしようとします。

set_libraries(*libnames*)

コンパイラオブジェクトによるリンク時に使われるライブラリのリストを *libnames* (文字列のリスト) に設定します。これはリンカがデフォルトでリンクする標準のシステムライブラリには影響しません。

add_library_dir(*dir*)

add_library() と set_libraries() で指定されたライブラリを探索するディレクトリのリストに *dir* を追加します。リンカは add_library_dir() と set_library_dirs() で指定された順にディレクトリを探索されます。

set_library_dirs(*dirs*)

ライブラリを探索するディレクトリを *dirs* (文字列のリスト) に設定します。これはリンカがデ

フォルトで探索する標準ライブラリ探索パスには影響しません。

add_runtime_library_dir(*dir*)
実行時に共有ライブラリを探索するディレクトリのリストに *dir* を追加します。

set_runtime_library_dirs(*dirs*)
実行時に共有ライブラリを探索するディレクトリのリストを *dir* に設定します。これはランタイムリンカがデフォルトで利用する標準探索パスには影響しません。

define_macro(*name*[, *value=None*])
このコンパイラオブジェクトで実行される全てのコンパイルで利用されるプリプロセッサのマクロを定義します。省略可能なパラメータ *value* は文字列であるべきです。省略された場合は、マクロは特定の値をとらずに定義され、具体的な結果は利用されるコンパイラに依存します。(XXX 本当に? これについて ANSI で言及されている?)

undefine_macro(*name*)
このコンパイラオブジェクトで実行される全てのコンパイルで利用されるプリプロセッサのマクロ定義を消します。同じマクロを `define_macro()` で定義し、`undefine_macro()` で定義を削除した場合、後で呼び出されたものが優先される (複数の再定義と削除を含みます)。もしコンパイルごと (すなわち `compile()` の呼び出しごと) にマクロが再定義/削除される場合も後で呼び出されたものが優先されます。

add_link_object(*object*)
このコンパイラオブジェクトによる全てのリンクで利用されるオブジェクトファイル (または類似のライブラリファイルや “リソースコンパイラ” の出力) のリストに *object* を追加します。

set_link_objects(*objects*)
このコンパイラオブジェクトによる全てのリンクで利用されるオブジェクトファイル (または類似のもの) のリストを *objects* に設定します。これはリンカがデフォルト利用する標準オブジェクトファイル (システムライブラリなど) には影響しません。

以下のメソッドはコンパイラオプションの自動検出を実装しており、 GNU **autoconf** に似たいくつかの機能を提供します。

detect_language(*sources*)
与えられたファイルまたはファイルのリストの言語を検出します。インスタンス属性 `language_map` (辞書) と、`language_order` (リスト) を仕事に使います。

find_library_file(*dirs*, *lib*[, *debug=0*])
指定されたディレクトリのリストから、スタティックまたは共有ライブラリファイル *lib* を探し、そのファイルのフルパスを返します。もし *debug* が真なら、(現在のプラットフォームで意味があれば) デバッグ版を探します。指定されたどのディレクトリでも *lib* が見つからなければ `None` を返します。

has_function(*funcname*[, *includes=None*, *include_dirs=None*, *libraries=None*, *library_dirs=None*])
funcname が現在のプラットフォームでサポートされているかどうかをブール値で返します。省略可能引数は追加のインクルードファイルやパス、ライブラリやパスを与えることでコンパイル環境を指定します。

library_dir_option(*dir*)
dir をライブラリ探索ディレクトリに追加するコンパイラオプションを返します。

library_option(*lib*)
共有ライブラリまたは実行ファイルにリンクされるライブラリ一覧に *lib* を追加するコンパイラオプションを返します。

runtime_library_dir_option(*dir*)
ランタイムライブラリを検索するディレクトリのリストに *dir* を追加するコンパイラオプションを返します。

set_executables(***args*)
コンパイルのいろいろなステージで実行される実行ファイル (とその引数) を定義します。コンパイラクラス (の 'executables' 属性) によって実行ファイルのセットは変わる可能性がありますが、ほとんどは以下のものを持っています:

attribute	description
compiler	C/C++ コンパイラ
linker_so	シェアードオブジェクト、ライブラリを作るために使うリンカ
linker_exe	バイナリ実行可能ファイルを作るために使うリンカ
archiver	静的ライブラリを作るアーカイバ

コマンドラインをもつプラットフォーム (Unix, DOS/Windows) では、それぞれの文字列は実行ファイル名と (省略可能な) 引数リストに分割されます。(文字列の分割は Unix のシェルが行うものに似ています: 単語はスペースで区切られますが、クォートとバックスラッシュでオーバーライドできます。 `distutils.util.split_quoted()` をごらんください。)

以下のメソッドはビルドプロセスのステージを呼び出します。

compile(*sources*[, *output_dir=None*, *macros=None*, *include_dirs=None*, *debug=0*, *extra_preargs=None*, *extra_postargs=None*, *depends=None*])
1 つ以上のソースファイルをコンパイルします。オブジェクトファイルを生成 (たとえば `.c` ファイルを `.o` ファイルに変換) します。

sources はファイル名のリストである必要があります。おそらく C/C++ ファイルですが、実際にはコンパイラとコンパイラクラスで扱えるもの (例: `MSVCCompiler` はリソースファイルを *sources* にとることができます) なら何でも指定できます。*sources* のソースファイルひとつずつに対応するオブジェクトファイル名のリストを返します。実装に依存しますが、全てのソースファイルがコンパイルされる必要はありません。しかし全ての対応するオブジェクトファイル名が返ります。

もし *output_dir* が指定されていれば、オブジェクトファイルはその下に、オリジナルのパスを維持した状態で置かれます。つまり、`foo/bar.c` は通常コンパイルされて `foo/bar.o` になります (Unix 実装の場合) が、もし *output_dir* が *build* であれば、 `build/foo/bar.o` になります。

macros は (もし指定されていれば) マクロ定義のリストである必要があります。マクロ定義は `(name, value)` という形式の 2 要素のタプル、または `(name,)` という形式の 1 要素のタプルのどちらかです。前者はマクロを定義します。もし value が `None` であれば、マクロは特定の値をもたないで定義されます。1 要素のタプルはマクロ定義を削除します。後で実行された定義/再定義/削除が優先されます。

include_dirs は (もし指定されていれば) 文字列のリストである必要があります。このコンパイルだけで有効な、デフォルトのインクルードファイルの検索ディレクトリに追加するディレクトリ群を指定します。

debug はブーリアン値です。もし真なら、コンパイラはデバッグシンボルをオブジェクトファイルに (または別ファイルに) 出力します。

extra_postargs と *extra_postargs* は実装依存です。コマンドラインをもっているプラットフォーム (例 Unix, DOS/Windows) では、おそらく文字列のリスト: コンパイラのコマンドライン引数の前/後に追加するコマンドライン引数です。他のプラットフォームでは、実装クラスのドキュメントを参照してください。どの場合でも、これらの引数は抽象コンパイラフレームワークが期待に沿わない時の脱出口として意図されています。

depends は (もし指定されていれば) ターゲットが依存しているファイル名のリストです。ソースファイルが依存しているファイルのどれかより古ければ、ソースファイルは再コンパイルされます。これは依存関係のトラッキングをサポートしていますが、荒い粒度でしか行われません。

失敗すると `CompileError` を起こします。

create_static_lib(*objects*, *output_libname*[, *output_dir=None*, *debug=0*, *target_lang=None*])

静的ライブラリファイルを作るために元ファイル群をリンクします。「元ファイル群」は *objects* で指定されたオブジェクトファイルのリストを基礎にしています。追加のオブジェクトファイルを `add_link_object()` および/または `set_link_objects()` で指定し、追加のライブラリを `add_library()` および/または `set_libraries()` で指定します。そして *libraries* で指定されたライブラリです。

output_libname はライブラリ名で、ファイル名ではありません; ファイル名はライブラリ名から作られます。 *output_dir* はライブラリファイルが起かれるディレクトリです。*debug* はブール値です。真なら、デバッグ情報がライブラリに含まれます (ほとんどのプラットフォームではコンパイルステップで意味をもちます: *debug* フラグは一貫性のためにここにもあります。)。

target_lang はオブジェクトがコンパイルされる対象になる言語です。これはその言語特有のリンク時の処理を可能にします。

失敗すると `LibError` を起こします。

link(*target_desc*, *objects*, *output_filename*[, *output_dir=None*, *libraries=None*, *library_dirs=None*, *runtime_library_dirs=None*, *export_symbols=None*, *debug=0*, *extra_preargs=None*, *extra_postargs=None*, *build_temp=None*, *target_lang=None*])

実行ファイルまたは共有ライブラリファイルを作るために元ファイル群をリンクします。

「元ファイル群」は *objects* で指定されたオブジェクトファイルのリストを基礎にしています。*output_filename* はファイル名です。もし *output_dir* が指定されていれば、それに対する相対パスとして *output_filename* は扱われます (必要ならば *output_filename* はディレクトリ名を含むことができます。)。

libraries はリンクするライブラリのリストです。これはファイル名ではなくライブラリ名で指定します。プラットフォーム依存の方式でファイル名に変換されます (例: *foo* は Unix では `libfoo.a` に、DOS/Windows では `foo.lib` になります。)。ただしこれらはディレクトリ名を含むことができ、その場合はリンカは通常の場所全体を探すのではなく特定のディレクトリを参照します。

library_dirs はもし指定されるならば、修飾されていない (ディレクトリ名を含んでいない) ライブラリ名で指定されたライブラリを探索するディレクトリのリストです。これはシステムのデフォルトより優先され、 `add_library_dir()` と/または `set_library_dirs()` に渡されます。 *runtime_library_dirs* は共有ライブラリに埋め込まれるディレクトリのリストで、実行時

にそれが依存する共有ライブラリのパスを指定します (これは Unix でだけ意味があるかもしれません。)。

export_symbols は共有ライブラリがエクスポートするシンボルのリストです。(これは Windows だけで意味があるようです。)

debug は `compile()` や `create_static_lib()` と同じですが、少しだけ違いがあり、(`create_static_lib()` では *debug* フラグは形式をあわせるために存在していたのに対して) ほとんどのプラットフォームで意識されます。

extra_preargs と *extra_postargs* は `compile()` と同じですが、コンパイラではなくリンカへの引数として扱われます。

target_lang は指定されたオブジェクトがコンパイルされた対象言語です。リンク時に言語特有の処理を行えるようにします。

失敗すると `LinkError` が起きます。

link_executable(*objects*, *output_progname*[, *output_dir=None*, *libraries=None*, *library_dirs=None*, *runtime_library_dirs=None*, *debug=0*, *extra_preargs=None*, *extra_postargs=None*, *target_lang=None*])
実行ファイルをリンクします。*output_progname* は実行ファイルの名前です。*objects* はリンクされるオブジェクトのファイル名のリストです。他の引数は `link()` メソッドと同じです。

link_shared_lib(*objects*, *output_libname*[, *output_dir=None*, *libraries=None*, *library_dirs=None*, *runtime_library_dirs=None*, *export_symbols=None*, *debug=0*, *extra_preargs=None*, *extra_postargs=None*, *build_temp=None*, *target_lang=None*])
共有ライブラリをリンクします。*output_libname* は出力先のライブラリ名です。*objects* はリンクされるオブジェクトのファイル名のリストです。他の引数は `link()` メソッドと同じです。

link_shared_object(*objects*, *output_filename*[, *output_dir=None*, *libraries=None*, *library_dirs=None*, *runtime_library_dirs=None*, *export_symbols=None*, *debug=0*, *extra_preargs=None*, *extra_postargs=None*, *build_temp=None*, *target_lang=None*])
共有オブジェクトをリンクします。*output_filename* は出力先の共有オブジェクト名です。*objects* はリンクされるオブジェクトのファイル名のリストです。他の引数は `link()` メソッドと同じです。

preprocess(*source*[, *output_file=None*, *macros=None*, *include_dirs=None*, *extra_preargs=None*, *extra_postargs=None*])
source で指定されたひとつの C/C++ソースファイルをプリプロセスします。出力先のファイルは *output_file* か、もし *output_file* が指定されていなければ *stdout* になります。 *macro* は `compile()` と同様にマクロ定義のリストで、`define_macro()` や `undefine_macro()` によって引数になります。*include_dirs* はデフォルトのリストに追加されるディレクトリ名のリストで、 `add_include_dir()` と同じ方法で扱われます。

失敗すると `PreprocessError` が起きます。

以下のユーティリティメソッドは具体的なサブクラスで使うために、`CCompiler` クラスで定義されています。

executable_filename(*basename*[, *strip_dir=0*, *output_dir=''*])
basename で指定された実行ファイルのファイル名を返します。Windows 以外の典型的なプラッ

トフォームでは basename そのままが、Windows では .exe が追加されたものが返ります。

library_filename(*libname*[, *lib_type='static'*, *strip_dir=0*, *output_dir=''*])
現在のプラットフォームでのライブラリファイル名を返します。Unix で *lib_type* が 'static' の場合、 liblibname.a の形式を返し、 *lib_type* が'dynamic' の場合は liblibname.so の形式を返します。

object_filenames(*source_filenames*[, *strip_dir=0*, *output_dir=''*])
指定されたソースファイルに対応するオブジェクトファイル名を返します。 *source_filenames* はファイル名のリストです。

shared_object_filename(*basename*[, *strip_dir=0*, *output_dir=''*])
basename に対応する共有オブジェクトファイルのファイル名を返します。

execute(*func*, *args*[, *msg=None*, *level=1*])
distutils.util.execute() を呼びだします。このメソッドはログを取り、 *dry_run* フラグを考慮にいれて、Python 関数 *func* に引数 *args* を与えて呼びだします。

spawn(*cmd*)
distutils.util.spawn() を呼び出します。これは指定したコマンドを実行する外部プロセスを呼び出します。

mkpath(*name*[, *mode=511*])
distutils.dir_util.mkpath() を呼び出します。これは親ディレクトリ込みでディレクトリを作成します。

move_file(*src*, *dst*)
distutils.file_util.move_file() を呼び出します。 *src* を *dst* にリネームします。

announce(*msg*[, *level=1*])
distutils.log.debug() 関数を使ってメッセージを書き出します。

warn(*msg*)
警告メッセージ *msg* を標準エラー出力に書き出します。

debug_print(*msg*)
もしこの CCompiler インスタンスで *debug* フラグが指定されていれば *msg* を標準出力に出力し、そうでなければ何も出力しません。

11.3 distutils.unixccompiler — Unix C コンパイラ

このモジュールは UnixCCompiler クラスを提供します。CCompiler クラスのサブクラスで、典型的な Unix スタイルのコマンドライン C コンパイラを扱います:

- マクロは *-Dname[=value]* で定義されます。
- マクロは *-Uname* で削除されます。
- インクルードファイルの探索ディレクトリは *-Idir* で指定されます。
- ライブラリは *-llib* で指定されます。
- ライブラリの探索ディレクトリは *-Ldir* で指定されます。

- コンパイルは **cc** (またはそれに似た) 実行ファイルに、 *-c* オプションをつけて実行します: `.c` を `.o` にコンパイルします。
- 静的ライブラリは **ar** コマンドで処理されます (**ranlib** を使うかもしれません)
- 共有ライブラリのリンクは **cc** *-shared* で処理されます。

11.4 distutils.msvccompiler — Microsoft コンパイラ

このモジュールは `MSVCCompiler` クラスを提供します。抽象クラス `CCompiler` の具象クラスで Microsoft Visual Studio 向けのものです。一般的に、拡張モジュールは Python をコンパイルしたのと同じコンパイラでコンパイルする必要があります。Python 2.3 やそれ以前では、コンパイラは Visual Studio 6 でした。Python 2.4 と Python 2.5 では、コンパイラは Visual Studio .NET 2003 です。 AMD64 と Itanium バイナリは Platform SDK を利用して作成されました。

`MSVCCompiler` は大体正しいコンパイラ、リンカその他を選びます。この選択を上書きするためには、環境変数 *DISTUTILS_USE_SDK* と *MSSdk* の両方を設定する必要があります。 *MSSdk* は現在の環境をセットアップした `SetEnv.Cmd` スクリプト、もしくは環境変数が SDK をインストールした時に登録されたものであることを示します。*DISTUTILS_USE_SDK* は distutils のユーザーが明示的に `MSVCCompiler` が選んだコンパイラを上書きすることを示します。

11.5 distutils.bcppcompiler — Borland コンパイラ

このモジュールは `BorlandCCompiler` クラスを提供します。抽象クラス `CCompiler` の具象クラスで Borland C++ コンパイラ向けです。

11.6 distutils.cygwincompiler — Cygwin コンパイラ

このモジュールは `CygwinCCompiler` クラスを提供します。 `UnixCCompiler` のサブクラスで Cygwin に移植された Windows 用の GNU C コンパイラ向けです。さらに `Mingw32CCompiler` クラスを含んでおり、これは mingw32 向けに移植された GCC (cygwin の no-cygwin モードと同じ) 向けです。

11.7 distutils.emxccompiler — OS/2 EMX コンパイラ

このモジュールは `EMXCCompiler` クラスを提供します。 `UnixCCompiler` のサブクラスで GNU C コンパイラの OS/2 向け EMX ポートを扱います。

11.8 distutils.mwerkscompiler — Metrowerks CodeWarrior サポート

MWerksCompiler クラスを提供します。抽象クラス CCompiler の具象クラスで Mac OS X 以前の Macintosh の MetroWerks CodeWarrior 向けです。 Windows や Mac OS X の CW をサポートするには作業が必要です。

11.9 distutils.archive_util — アーカイブユーティリティ

このモジュールはアーカイブファイル (tar や zip) を作成する関数を提供します。

distutils.archive_util.**make_archive**(*base_name*, *format*[, *root_dir=None*, *base_dir=None*, *verbose=0*, *dry_run=0*])

アーカイブファイル (例: zip や tar) を作成します。 *base_name* は作成するファイル名からフォーマットの拡張子を除いたものです。*format* はアーカイブのフォーマットで zip, tar, ztar, gztar のいずれかです。*root_dir* はアーカイブのルートディレクトリになるディレクトリです: つまりアーカイブを作成する前に *root_dir* に *chdir* します。*base_dir* はアーカイブの起点となるディレクトリです: つまり *base_dir* はアーカイブ中の全ファイルおよびディレクトリの前につくディレクトリ名です。*root_dir* と *base_dir* はともにカレントディレクトリがデフォルト値です。アーカイブファイル名を返します。

distutils.archive_util.**make_tarball**(*base_name*, *base_dir*[, *compress='gzip'*, *verbose=0*, *dry_run=0*])

base_dir 以下の全ファイルから、tar ファイルを作成 (オプションで圧縮) します。*compress* は 'gzip', 'compress', 'bzip2', または None である必要があります。 **tar** と *compress* で指定された圧縮ユーティリティにはパスが通っている必要があるので、これはおそらく Unix だけで有効です。出力 tar ファイルは base_dir.tar という名前になり、圧縮によって拡張子がつきます (.gz, .bz2 または .Z)。出力ファイル名が返ります。

distutils.archive_util.**make_zipfile**(*base_name*, *base_dir*[, *verbose=0*, *dry_run=0*])

base_dir 以下の全ファイルから、zip ファイルを作成します。出力される zip ファイルは *base_dir* + .zip という名前になります。 zipfile Python モジュール (利用可能なら) または InfoZIP zip ユーティリティ(インストールされていてパスが通っているなら) を使います。もしどちらも利用できなければ、 DistutilsExecError が起きます。出力 zip ファイル名が返ります。

11.10 distutils.dep_util — 依存関係のチェック

このモジュールはシンプルなタイムスタンプを元にしたファイルやファイル群の依存関係を処理する関数を提供します。さらに、それらの依存関係解析を元にした関数を提供します。

distutils.dep_util.**newer**(*source*, *target*)

source が存在して、 *target* より最近変更されている、または *source* が存在して、 *target* が存在していない場合は真を返します。両方が存在していて、 *target* のほうが *source* より新しいか同じ場合には偽を返します。 *source* が存在しない場合には DistutilsFileError を起こします。

distutils.dep_util.**newer_pairwise**(*sources*, *targets*)
ふたつのファイル名リストを並列に探索して、それぞれのソースが対応するターゲットより新しいかをテストします。 newer() の意味でターゲットよりソースが新しいペアのリスト (*sources*,*targets*) を返します。

distutils.dep_util.**newer_group**(*sources*, *target*[, *missing='error'*])
target が *source* にリストアップされたどれかのファイルより古ければ真を返します。言い換えれば、*target* が存在して *sources* の全てより新しいなら偽を返し、そうでなければ真を返します。 *missing* はソースファイルが存在しなかった時の振る舞いを決定します。デフォルト ('error') は os.stat() で OSError 例外を起こします。もし 'ignore' なら、単に存在しないソースファイルを無視します。もし 'newer' なら、存在しないソースファイルについては *target* が古いとみなします (これは"dry-tun" モードで便利です: 入力がないのでコマンドは実行できませんが実際に実行しようとしていないので問題になりません)。

11.11 distutils.dir_util — ディレクトリツリーの操作

このモジュールはディレクトリとディレクトリツリーを操作する関数を提供します。

distutils.dir_util.**mkpath**(*name*[, *mode=0777*, *verbose=0*, *dry_run=0*])
ディレクトリと、必要な親ディレクトリを作成します。もしディレクトリが既に存在している (*name* が空文字列の場合、カレントディレクトリを示すのでもちろん存在しています) 場合、何もしません。ディレクトリを作成できなかった場合 (例: ディレクトリと同じ名前のファイルが既に存在していた)、DistutilsFileError を起こします。もし *verbose* が真なら、それぞれの mkdir について 1 行、標準出力に出力します。実際に作成されたディレクトリのリストを返します。

distutils.dir_util.**create_tree**(*base_dir*, *files*[, *mode=0777*, *verbose=0*, *dry_run=0*])
files を置くために必要な空ディレクトリを *base_dir* 以下に作成します。 *base_dir* ディレクトリは存在している必要はありません。*files* はファイル名のリストで *base_dir* からの相対パスとして扱われます。 *base_dir* + *files* のディレクトリ部分が (既に存在していなければ) 作成されます。 *mode*, *verbose* と *dry_run* フラグは mkpath() と同じです。

distutils.dir_util.**copy_tree**(*src*, *dst*[, *preserve_mode=1*, *preserve_times=1*, *preserve_symlinks=0*, *update=0*, *verbose=0*, *dry_run=0*])
src ディレクトリツリー全体を *dst* にコピーします。 *src* と *dst* はどちらもディレクトリ名である必要があります。もし *src* がディレクトリでなければ、 DistutilsFileError を起こします。もし *dst* が存在しなければ、 mkpath() で作成されます。実行結果は、 *src* 以下の全てのファイルが *dst* にコピーされ、*src* 以下の全てのディレクトリが *dst* に再帰的にコピーされます。コピーされた (またはされるはず) のファイルのリストを返します。返り値は *update* または *dry_run* に影響されません: *src* 以下の全ファイルを単に *dst* 以下に改名したリストが返されます。

preserve_mode と *preserve_times* は distutils.file_util の copy_file() と同じです: 通常のファイルには適用されますが、ディレクトリには適用されません。 もし *preserve_symlinks* が真なら、シンボリックリンクは (サポートされているシステムでは) シンボリックリンクとしてコピーされます。そうでなければ (デフォルト) シンボリックリンクは参照されている実体ファイルがコピーされます。*update* と *verbose* は copy_file() と同じです。

distutils.dir_util.**remove_tree**(*directory*[, *verbose=0*, *dry_run=0*])
再帰的に *directory* とその下の全ファイルを削除します。エラーは無視されます (*verbose* が真の時は sys.stdout に出力されます)

11.12 distutils.file_util — 1 ファイルの操作

このモジュールはそれぞれのファイルを操作するユーティリティ関数を提供します。

distutils.file_util.**copy_file**(*src*, *dst*[, *preserve_mode=1*, *preserve_times=1*, *update=0*, *link=None*, *verbose=0*, *dry_run=0*])

ファイル *src* を *dst* にコピーします。もし *dst* がディレクトリなら、 *src* はそこへ同じ名前でコピーされます; そうでなければ、ファイル名として扱われます。 (もしファイルが存在するなら、上書きされます。) mosil *preserve_mode* が真 (デフォルト) なら、ファイルのモード (タイプやパーミッション、その他プラットフォームがサポートするもの) もコピーされます。もし *preserve_times* が真 (デフォルト) なら、最終更新、最終アクセス時刻もコピーされます。もし *update* が真なら、 *src* は *dst* が存在しない場合か、 *dst* が *src* より古い時にだけコピーします。

link は値を `'hard'` または `'sym'` に設定することでコピーのかわりにハードリンク (`os.link()` を使います) またはシンボリックリンク (`os.symlink()` を使います) を許可します。 `None` (デフォルト) の時には、ファイルはコピーされます。 *link* をサポートしていないシステムで有効にしないでください。`copy_file()` はハードリンク、シンボリックリンクが可能かチェックしていません。ファイルの内容をコピーするために`_copy_file_contents()` を利用しています。

`(dest_name, copied)` のタプルを返します: *dest_name* は出力ファイルの実際の名前、 *copied* はファイルがコピーされた (*dry_run* が真の時にはコピーされることになった) 場合には真です。

distutils.file_util.**move_file**(*src*, *dst*[, *verbose*, *dry_run*])

ファイル *src* を *dst* に移動します。もし *dst* がディレクトリなら、ファイルはそのディレクトリに同じ名前で移動されます。そうでなければ、 *src* は *dst* に単にリネームされます。新しいファイルの名前を返します。

> **警告:** Unix ではデバイスをまたがる移動は `copy_file()` を利用して扱っています。
> (todo:他のシステムではどうなっている?)

distutils.file_util.**write_file**(*filename*, *contents*)

filename を作成し、 *contents* (行末文字がない文字列のシーケンス) を書き込みます。

11.13 distutils.util — その他のユーティリティ関数

このモジュールは他のユーティリティモジュールにあわないものを提供しています。

distutils.util.**get_platform**()

現在のプラットフォームを示す文字列を返します。これはプラットフォーム依存のビルドディレクトリやプラットフォーム依存の配布物を区別するために使われます。典型的には、('os.uname()' のように) OS の名前とバージョン、アーキテクチャを含みますが、厳密には OS に依存します。たとえば IRIX ではアーキテクチャはそれほど重要ではありません (IRIX は SGI のハードウェアだけで動作する) が、Linux ではカーネルのバージョンはそれほど重要ではありません。

返り値の例:

- `linux-i586`
- `linux-alpha`

- solaris-2.6-sun4u
- irix-5.3
- irix64-6.2

POSIX でないプラットフォームでは、今のところ単に sys.platform が返されます。

Mac OS X システムでは、OS バージョンは、現在の OS バージョンではなく、実行するバイナリの最小バージョンを表しています。(これは、Python をビルドするときの MACOSX_DEPLOYMENT_TARGET の値です。)

Mac OS X のユニバーサルバイナリビルドでは、アーキテクチャの値は現在のプロセッサではなく、ユニバーサルバイナリの状態を表しています。32bit ユニバーサルバイナリではアーキテクチャは fat で、64bit ユニバーサルバイナリではアーキテクチャは fat64 で、4-way ユニバーサルバイナリではアーキテクチャは universal になります。Python 2.7 と Python 3.2 から 3-way ユニバーサルバイナリ (ppc, i386, x86_64) には fat3 が i386 と x86_64 ユニバーサルバイナリには intel が使われるようになりました。

Mac OS X で返される値の例:

- macosx-10.3-ppc
- macosx-10.3-fat
- macosx-10.5-universal
- macosx-10.6-intel

distutils.util.**convert_path**(*pathname*)

'pathname' をファイルシステムで利用できる名前にして返します。すなわち、'/' で分割し、現在のディレクトリセパレータで接続しなおします。セットアップスクリプト中のファイル名は Unix スタイルで提供され、実際に利用する前に変換する必要があるため、この関数が必要になります。もし *pathname* の最初または最後がスラッシュの場合、Unix 的でないシステムでは ValueError が起きます。

distutils.util.**change_root**(*new_root*, *pathname*)

pathname の前に *new_root* を追加したものを返します。もし *pathname* が相対パスなら、os.path.join(new_root,pathname) と等価です。そうでなければ、*pathname* を相対パスに変換したあと接続します。これは DOS/Windows ではトリッキーな作業になります。

distutils.util.**check_environ**()

'os.environ' に、ユーザが config ファイル、コマンドラインオプションなどで利用できることを保証している環境変数があることを確認します。現在は以下のものが含まれています:

- HOME - ユーザのホームディレクトリ (Unix のみ)
- PLAT - ハードウェアと OS を含む現在のプラットフォームの説明。(get_platform() を参照)

distutils.util.**subst_vars**(*s*, *local_vars*)

shell/Perl スタイルの変数置換を *s* について行います。全ての $ に名前が続いたものは変数とみなされ、辞書 *local_vars* でみつかった値に置換されます。*local_vars* で見つからなかった場合には os.environ で置換されます。*os.environ* は最初にある値を含んでいることをチェックされます: check_environ() を参照。 *local_vars* or os.environ のどちらにも値が見つからなかった場合、 ValueError を起こします。

これは完全な文字列挿入関数ではないことに注意してください。 `$variable` の名前には大小英字、数字、アンダーバーだけを含むことができます。 { } や ( ) を使った引用形式は利用できません。

distutils.util.**grok_environment_error**(*exc*[, *prefix='error: '*])
例外オブジェクト EnvironmentError (IOError または OSError) から、エラーメッセージを生成します。 Python 1.5.1 またはそれ以降の形式を扱い、ファイル名を含んでいない例外オブジェクトも扱います。このような状況はエラーが 2 つのファイルに関係する操作、たとえば `rename()` や `link()` で発生します。*prefix* をプレフィクスに持つエラーメッセージを返します。

distutils.util.**split_quoted**(*s*)
文字列を Unix のシェルのようなルール (引用符やバックスラッシュの扱い) で分割します。つまり、バックスラッシュでエスケープされるか、引用符で囲まれていなければ各語はスペースで区切られます。一重引用符と二重引用符は同じ意味です。引用符もバックスラッシュでエスケープできます。2 文字でのエスケープシーケンスに使われているバックスラッシュは削除され、エスケープされていた文字だけが残ります。引用符は文字列から削除されます。語のリストが返ります。

distutils.util.**execute**(*func*, *args*[, *msg=None*, *verbose=0*, *dry_run=0*])
外部に影響するいくつかのアクション (たとえば、ファイルシステムへの書き込み) を実行します。そのようなアクションは *dry_run* フラグで無効にする必要があるので特別です。この関数はその繁雑な処理を行います。関数と引数のタプル、(実行する「アクション」をはっきりさせるための) 表示に使われる任意のメッセージを渡してください。

distutils.util.**strtobool**(*val*)
真偽値をあらわす文字列を真 (1) または偽 (0) に変換します。

真の値は `y`, `yes`, `t`, `true`, `on` そして `1` です。偽の値は `n`, `no`, `f`, `false`, `off` そして `0` です。 *val* が上のどれでもない時は `ValueError` を起こします。

distutils.util.**byte_compile**(*py_files*[, *optimize=0*, *force=0*, *prefix=None*, *base_dir=None*, *verbose=1*, *dry_run=0*, *direct=None*])
Python ソースファイル群をバイトコンパイルして `.pyc` または `.pyo` ファイルを同じディレクトリに作成します。 *py_files* はコンパイルされるファイルのリストです。`.py` で終わっていないファイルはスキップされます。 *optimize* は以下のどれかです:

- `0` - 最適化しない (`.pyc` ファイルを作成します)
- `1` - 通常の最適化 (`python -O` のように)
- `2` - さらに最適化 (`python -OO` のように)

もし *force* が真なら、全てのファイルがタイムスタンプに関係なく再コンパイルされます。

バイトコード (*bytecode*) ファイルにエンコードされるソースファイル名は、デフォルトでは *py_files* が使われます。これを *prefix* と *base_dir* で変更することができます。 *prefix* はそれぞれのソースファイル名から削除される文字列で、*base_dir* は (*prefix* を削除したあと) 先頭に追加されるディレクトリ名です。任意に *prefix* と *base_dir* のどちらか、両方を与える (与えない) ことができます。

もし *dry_run* が真なら、ファイルシステムに影響することは何もされません。

バイトコンパイルは現在のインタプリタプロセスによって標準の `py_compile` モジュールを使って直接行われるか、テンポラリスクリプトを書いて間接的に行われます。通常は `byte_compile()` に直接かそうでないかをまかせます (詳細についてはソースをごらんください)。*direct* フラグは関節

モードで作成されたスクリプトで使用されます。何をやっているか理解していない時は None のままにしておいてください。

distutils.util.**rfc822_escape**(*header*)
RFC 822 ヘッダに含められるよう加工した *header* を返します。改行のあとには 8 つのスペースが追加されます。この関数は文字列に他の変更はしません。

11.14 distutils.dist — Distribution クラス

このモジュールは Distribution クラスを提供します。これは構築/インストール/配布される配布物をあらわします。

11.15 distutils.extension — Extension クラス

このモジュールは Extension クラスを提供します。 C/C++拡張モジュールをセットアップスクリプトで表すために使われます。

11.16 distutils.debug — Distutils デバッグモード

このモジュールは DEBUG フラグを提供します。

11.17 distutils.errors — Distutils 例外

distutils のモジュールで使用される例外を提供します。 distutils のモジュールは標準的な例外を起こします。特に、 SystemExit はエンドユーザによる失敗 (コマンドライン引数の間違いなど) で起きます。

このモジュールは `from ... import *` で安全に使用することができます。このモジュールは Distutils ではじまり、 Error で終わるシンボルしか export しません。

11.18 distutils.fancy_getopt — 標準 getopt モジュールのラッパ

このモジュールは以下の機能を標準の getopt モジュールに追加するラッパを提供します:

- 短いオプションと長いオプションを関連づけます
- オプションはヘルプ文字列を持ちます。可能性としては `fancy_getopt()` に完全な利用方法サマリを作らせることができます。
- オプションは渡されたオブジェクトの属性を設定します。
- 真偽値をとるオプションは “負のエイリアス” を持ちます。— たとえば *--quiet* の “負のエイリアス” が*--verbose* の場合、コマンドラインで *--quiet* を指定すると *verbose* は偽になります。

distutils.fancy_getopt.**fancy_getopt**(*options*, *negative_opt*, *object*, *args*)
ラッパ関数。 *options* は FancyGetopt のコンストラクタで説明されている (long_option, short_option, help_string) の 3 要素タプルのリストです。 *negative_opt* はオプション名からオプション名のマッピングになっている辞書で、キー、値のどちらも *options* リストに含まれている必要があります。*object* は値を保存するオブジェクト (FancyGetopt クラスの getopt() メソッドを参照してください) です。*args* は引数のリストです。*args* として None を渡すと、sys.argv[1:] が使われます。

distutils.fancy_getopt.**wrap_text**(*text*, *width*)
text を *width* 以下の幅で折り返します。

class distutils.fancy_getopt.**FancyGetopt**([*option_table=None*])
option_table は 3 つ組タプルのリストです。(long_option, short_option, help_string)

もしオプションが引数を持つなら、 *long_option* に '=' を追加する必要があります。 *short_option* は一文字のみで、 ':' はどの場合にも不要です。 *long_option* に対応する *short_option* がない場合、*short_option* は None にしてください。全てのオプションタプルは長い形式のオプションを持つ必要があります。

FancyGetopt クラスは以下のメソッドを提供します:

FancyGetopt.**getopt**([*args=None*, *object=None*])
args のコマンドラインオプションを解析します。 *object* に属性として保存します。

もし *args* が None もしくは与えられない場合には、 sys.argv[1:] を使います。もし *object* が None もしくは与えられない場合には、新しく OptionDummy インスタンスを作成し、オプションの値を保存したのち (args, object) のタプルを返します。もし *object* が提供されていれば、その場で変更され、 getopt() は *args* のみを返します。どちらのケースでも、返された *args* は渡された *args* リスト (これは変更されません) の変更されたコピーです。

FancyGetopt.**get_option_order**()
直前に実行された getopt() が処理した (option, value) タプルのリストを返します。getopt() がまだ呼ばれていない場合には RuntimeError を起こします。

FancyGetopt.**generate_help**([*header=None*])
この FancyGetopt オブジェクトのオプションテーブルからヘルプテキスト (出力の一行に対応する文字列のリスト) を生成します。

もし与えられていれば、 *header* をヘルプの先頭に出力します。

11.19 distutils.filelist — FileList クラス

このモジュールはファイルシステムを見て、ファイルのリストを構築するために使われる FileList クラスを提供します。

11.20 distutils.log — シンプルな PEP 282 スタイルのロギング

11.21 distutils.spawn — サブプロセスの生成

このモジュールは spawn() 関数を提供します。これは様々なプラットフォーム依存の他プログラムをサブプロセスとして実行する関数に対するフロントエンドになっています。与えられた実行ファイルの名前からパスを探索する find_executable() 関数も提供しています。

11.22 distutils.sysconfig — システム設定情報

distutils.sysconfig モジュールでは、 Python の低水準の設定情報へのアクセス手段を提供しています。アクセスできる設定情報変数は、プラットフォームと設定自体に大きく左右されます。また、特定の変数は、使っているバージョンの Python のビルドプロセスに左右されます; こうした変数は、 Unix システムでは、 Makefile や Python と一緒にインストールされる設定ヘッダから探し出されます。設定ファイルのヘッダは、2.2 以降のバージョンでは pyconfig.h 、それ以前のバージョンでは config.h です。

他にも、 distutils パッケージの別の部分を操作する上で便利な関数がいくつか提供されています。

distutils.sysconfig.**PREFIX**
os.path.normpath(sys.prefix) の結果です。

distutils.sysconfig.**EXEC_PREFIX**
os.path.normpath(sys.exec_prefix) の結果です。

distutils.sysconfig.**get_config_var**(*name*)
ある一つの設定変数に対する値を返します。get_config_vars().get(name) と同じです。

distutils.sysconfig.**get_config_vars**(...)
定義されている変数のセットを返します。引数を指定しなければ、設定変数名を変数の値に対応付けるマップ型を返します。引数を指定する場合、引数の各値は文字列でなければならず、戻り値は引数に関連付けられた各設定変数の値からなるシーケンスになります。引数に指定した名前の設定変数に値がない場合、その変数に対する戻り値には None が入ります。

distutils.sysconfig.**get_config_h_filename**()
設定ヘッダのフルパス名を返します。 Unix の場合、このヘッダファイルは **configure** スクリプトによって生成されるヘッダファイル名です; 他のプラットフォームでは、ヘッダは Python ソース配布物中で直接与えられています。ファイルはプラットフォーム固有のテキストファイルです。

distutils.sysconfig.**get_makefile_filename**()
Python をビルドする際に用いる Makefile のフルパスを返します。 Unix の場合、このファイルは **configure** スクリプトによって生成されます; 他のプラットフォームでは、この関数の返す値の意味は様々です。有意なファイル名を返す場合、ファイルはプラットフォーム固有のテキストファイル形式です。この関数は POSIX プラットフォームでのみ有用です。

distutils.sysconfig.**get_python_inc**([*plat_specific*[, *prefix*]])
C インクルードファイルディレクトリについて、一般的なディレクトリ名か、プラットフォーム依存のディレクトリ名のいずれかを返します。*plat_specific* が真であれば、プラットフォーム依存のインクルードディレクトリ名を返します; *plat_specific* が偽か、省略された場合には、プラットフォームに

依存しないディレクトリを返します。 *prefix* が指定されていれば、 PREFIX の代わりに用いられます。また、 *plat_specific* が真の場合、 EXEC_PREFIX の代わりに用いられます。

distutils.sysconfig.**get_python_lib**([*plat_specific*[, *standard_lib*[, *prefix*]]])
ライブラリディレクトリについて、一般的なディレクトリ名か、プラットフォーム依存のディレクトリ名のいずれかを返します。*plat_specific* が真であれば、プラットフォーム依存のライブラリディレクトリ名を返します; *plat_specific* が偽か、省略された場合には、プラットフォームに依存しないディレクトリを返します。*prefix* が指定されていれば、 PREFIX の代わりに用いられます。また、 *plat_specific* が真の場合、 EXEC_PREFIX の代わりに用いられます。*standard_lib* が真であれば、サードパーティ製の拡張モジュールをインストールするディレクトリの代わりに、標準ライブラリのディレクトリを返します。

以下の関数は、 distutils パッケージ内の使用だけを前提にしています。

distutils.sysconfig.**customize_compiler**(*compiler*)
distutils.ccompiler.CCompiler インスタンスに対して、プラットフォーム固有のカスタマイズを行います。

この関数は現在のところ、Unix だけで必要ですが、将来の互換性を考慮して一貫して常に呼び出されます。この関数は様々な Unix の変種ごとに異なる情報や、Python の Makefile に書かれた情報をインスタンスに挿入します。この情報には、選択されたコンパイラやコンパイラ/リンカのオプション、そして共有オブジェクトを扱うためにリンカに指定する拡張子が含まれます。

この関数はもっと特殊用途向けで、Python 自体のビルドプロセスでのみ使われるべきです。

distutils.sysconfig.**set_python_build**()
distutils.sysconfig モジュールに、モジュールが Python のビルドプロセスの一部として使われることを知らせます。これによって、ファイルコピー先を示す相対位置が大幅に変更され、インストール済みの Python ではなく、ビルド作業領域にファイルが置かれるようになります。

11.23 distutils.text_file — TextFile クラス

このモジュールは TextFile クラスを提供します。これはテキストファイルへのインタフェースを提供し、コメントの削除、空行の無視、バックスラッシュでの行の連結を任意に行えます。

class distutils.text_file.**TextFile**([*filename=None*, *file=None*, ***options*])
このクラスはファイルのようなオブジェクトを提供します。これは行指向のテキストファイルを処理する時に共通して必要となる処理を行います: (# がコメント文字なら) コメントの削除、空行のスキップ、 (行末のバックスラッシュでの) 改行のエスケープによる行の連結、先頭/末尾の空白文字の削除。これらは全て独立して任意に設定できます。

クラスは warn() メソッドを提供しており、物理行つきの警告メッセージを生成することができます。この物理行は論理行が複数の物理行にまたがっていても大丈夫です。また unreadline() メソッドが一行先読みを実装するために提供されています。

TextFile のインスタンスは *filename*, *file*, またはその両方をとって作成されます。両方が None の場合 RuntimeError が起きます。*filename* は文字列、*file* はファイルオブジェクト (または readline() と close() のメソッドを提供する何か) である必要があります。TextFile が生成する警告メッセージに含めることができるので、*filename* を与えることが推奨されます、もし *file* が提供されなければ、TextFile は組み込みの open() を利用して自分で作成します。

オプションは全て真偽値で、 `readline()` で返される値に影響します。

option name	説明	デフォルト値
strip_comments	バックスラッシュでエスケープされていない限り、 `'#'` から行末までと、 `'#'` の先にある空白文字の並びを削除します。	true
lstrip_ws	行を返す前に先頭の空白文字の並びを削除します。	false
rstrip_ws	行を返す前に行末の空白文字 (改行文字を含みます!) の並びを削除します。	true
skip_blanks	コメントと空白を除いた*あとで*内容がない行をスキップします。(もし lstrip_ws と rstrip_ws がともに偽なら、空白文字だけの行があるかもしれません。これは *skip_blanks* が真でない限りスキップされません。)	true
join_lines	もしコメントと空白文字を削除したあとで、バックスラッシュが最後の改行文字でない文字なら、次の行を接続して一つの論理行とします: N 行の連続した行がバックスラッシュで終わる場合、N+1 行の物理行が 1 行の論理行として扱われます。	false
collapse_join	前の行と接続するとき、行頭の空白文字を削除します。 (`join_lines and not lstrip_ws`) の時だけ意味をもちます。	false

rstrip_ws は行末の改行を削除するので、 `readline()` のセマンティクスが組み込みファイルオブジェクトの `readline()` メソッドとは変わってしまいます! 特に、 *rstrip_ws* が真で *skip_blanks* が偽のとき、 `readline()` はファイルの終端で `None` を返し、空文字列を返したときは空行 (または全て空白文字の行) です。

open(*filename*)
新しいファイル *filename* を開きます。これはコンストラクタ引数の *file* と *filename* を上書きします。

close()
現在のファイルを閉じ、(ファイル名や現在の行番号を含め) 現在のファイルについての情報を全て消します。

warn(*msg*[, *line*=*None*])
標準エラー出力に現在のファイルの論理行に結びついた警告メッセージを出力します。もし現在の論理行が複数の物理行に対応するなら、警告メッセージは以下のように全体を参照します: `"lines 3-5"`。もし *line* が与えられていれば、現在の行番号を上書きします; 物理行のレンジをあらわすリストまたはタプル、もしくはある物理行をあらわす整数のどれでも与えられます。

readline()
現在のファイル (または `unreadline()` で"unread" を直前に行っていればバッファ) から論理行を 1 行読み込んで返します。もし *join_lines* オプションが真なら、このメソッドは複数の物理行を読み込んで接続した文字列を返します。現在の行番号を更新します。そのため `readline()` のあとに `warn()` を呼ぶと丁度読んだ行についての警告を出します。*rstrip_ws* が真で、 *strip_blanks* が偽のとき空文字列が返るので、ファイルの終端では `None` を返します。

readlines()
現在のファイルで残っている全ての論理行のリストを読み込んで返します。行番号を、ファイルの最後の行に更新します。

unreadline(*line*)
line (文字列) を次の `readline()` 用に、内部バッファに push します。行の先読みを必要とするパーサを実装する時に便利です。`unreadline()` で"unread" された行は `readline()` で読

み込む際に再度処理 (空白の除去など) されません。もし unreadlinee() を、 readline() を呼ぶ前に複数回実行すると、最後に unread した行から返されます。

11.24 distutils.version — バージョン番号クラス

11.25 distutils.cmd — Distutils コマンドの抽象クラス

このモジュールは抽象ベースクラス Command を提供します。

class distutils.cmd.**Command**(*dist*)

コマンドクラスを定義するための抽象ベースクラス — distutils の「働きバチ」 — です。コマンドクラスは *options* とよばれるローカル変数を持ったサブルーチンと考えることができます。オプションは initialize_options() で宣言され、 finalize_options() で定義さ (最終的な値を与えら) れます。どちらも全てのコマンドクラスで実装する必要があります。この 2 つの区別は必要です。なぜならオプションの値は外部 (コマンドライン、設定ファイルなど) から来るかもしれず、他のオプションに依存しているオプションは外部の影響を処理した後で計算される必要があるからです。そのため finalize_options() が存在します。サブルーチンの本体は全ての処理をオプションの値にもとづいて行う run() メソッドで、これも全てのコマンドクラスで実装される必要があります。

クラスのコンストラクタは Distribution のインスタンスである単一の引数 *dist* をとります。

11.26 distutils.command — Distutils 各コマンド

11.27 distutils.command.bdist — バイナリインストーラの構築

11.28 distutils.command.bdist_packager — パッケージの抽象ベースクラス

11.29 distutils.command.bdist_dumb — “ダム” インストーラを構築

11.30 distutils.command.bdist_msi — Microsoft Installer バイナリパッケージをビルドする

class distutils.command.bdist_msi.**bdist_msi**(*Command*)

Windows Installer (.msi) バイナリパッケージをビルドします。

多くの場合、bdist_msi インストーラは Win64 のサポートが優れていて、管理者が非インタラクティブインストールできたり、グループポリシーを利用したインストールができるので、bdist_wininst インストーラよりも良い選択です。

11.47 新しい Distutils コマンドの作成

このセクションでは Distutils の新しいコマンドを作成する手順の概要をしめします。

新しいコマンドは distutils.command パッケージ中のモジュールに作られます。command_template というディレクトリにサンプルのテンプレートがあります。このファイルを実装しようとしているコマンドと同名の新しいモジュールにコピーしてください。このモジュールはモジュール (とコマンド) と同じ名前のクラスを実装する必要があります。そのため、peel_banana コマンド (ユーザは setup.py peel_banana と実行できます) を実装する際には、 command_template を distutils/command/peel_banana.py にコピーし、distutils.cmd.Command のサブクラス peel_banana クラスを実装するように編集してください。

Command のサブクラスは以下のメソッドを実装する必要があります。

Command.**initialize_options**()
: このコマンドがサポートする全てのオプションのデフォルト値を設定します。これらのデフォルトは他のコマンドやセットアップスクリプト、設定ファイル、コマンドラインによって上書きされるかもしれません。そのためオプション間の依存関係を記述するには適切な場所ではありません。一般的に initialize_options() は単に self.foo = None のような定義だけを行います。

Command.**finalize_options**()
: このコマンドがサポートする全てのオプションの最終的な値を設定します。これは可能な限り遅く呼び出されます。つまりコマンドラインや他のコマンドによるオプションの代入のあとに呼び出されます。そのため、オプション間の依存関係を記述するのに適した場所です。もし *foo* が *bar* に依存しており、かつまだ *foo* が initialize_options() で定義された値のままなら、 *foo* を *bar* から代入しても安全です。

Command.**run**()
: コマンドの本体です。実行するべきアクションを実装しています。 initialize_options() で初期化され、他のコマンドされ、セットアップスクリプト、コマンドライン、設定ファイルでカスタマイズされ、 finalize_options() で設定されたオプションがアクションを制御します。端末への出力とファイルシステムとのやりとりは全て run() が行います。

sub_commands はコマンドの” ファミリー” を定式化したものです。たとえば install はサブコマンド install_lib install_headers などの親です。コマンドファミリーの親は *sub_commands* をクラス属性として持ちます。 2 要素のタプル (command_name, predicate) のリストで、*command_name* には文字列、 *predicate* には親コマンドのメソッドで、現在の状況がコマンド実行にふさわしいかどうか判断するものを指定します。 (例えば install_headers はインストールするべき C ヘッダファイルがある時だけ有効です。) もし *predicate* が None なら、そのコマンドは常に有効になります。

sub_commands は通常クラスの最後で定義されます。 これは predicate は bound されていないメソッドになるので、全て先に定義されている必要があるためです。

標準的な例は **install** コマンドです。

第 12 章

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 Distributing 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、Google Project Hosting 上の Issue Tracker

http://code.google.com/p/python-doc-ja/issues/list

までご報告ください。

12.1 翻訳者一覧 (敬称略)

- Yasushi Masuda <y.masuda at acm.org> (March 12, 2004)
- Kazuo Moriwaka (May 8, 2006)
- 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 class*) 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) イテ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、そして省略可能な `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、`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、Python のドキュメントを主要な目的として作られたドキュメントプロセッサの Sphinx を利用して、 reStructuredText 形式のソースから生成されました。

ドキュメントとそのツール群の開発は、 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.

このドキュメントや、 Python 自体のバグ報告については、 *reporting-bugs* を参照してください。

ノート: 訳注: 日本語訳の問題については、 Google Project Hosting 上の Issue Tracker <http://code.google.com/p/python-doc-ja/issues/list> に登録するか、python-doc-jp@python.jp にメールで報告をお願いします。

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, Jes s Cea Avi n, 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, Hern n Mart nez 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 H ring, 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-Andr Lemburg, Ross Light, Ulf A. Lindgren, Everett Lipman, Mirko Liss, Martin von L wis, 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, Fran ois 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！

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
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

次のページに続く

表 C.1 – 前のページからの続き

リリース	ベース	年	権利	GPL 互換
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
2.6.2	2.6.1	2009	PSF	yes
2.6.3	2.6.2	2009	PSF	yes
2.6.4	2.6.3	2009	PSF	yes
2.6.5	2.6.4	2010	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 2.6ja2

1. This LICENSE AGREEMENT is between the Python Software Foundation (“PSF”), and the Individual or Organization (“Licensee”) accessing and otherwise using Python 2.6ja2 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 deriva-

tive works, distribute, and otherwise use Python 2.6ja2 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright 2001-2010 Python Software Foundation; All Rights Reserved" are retained in Python 2.6ja2 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 2.6ja2 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 2.6ja2.

4. PSF is making Python 2.6ja2 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 2.6ja2 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 2.6ja2 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 2.6ja2, 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 2.6ja2, 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 Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 ("CNRI"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI's License Agreement and CNRI's notice of copyright, i.e., "Copyright 1995-2001 Corporation for National Research Initiatives; All Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI's License Agreement, Licensee may substitute the following text (omitting the quotes): "Python 1.6.1 is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: http://hdl.handle.net/1895.22/1013."

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. This License Agreement shall be governed by the federal intellectual property 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。

Python モジュール索引

d

索引