Documenting Python

リリース *2.6ja2*

Georg Brandl

2011 年 11 月 06 日

Python Software Foundation
Email: docs@python.org

目 次

第 1 章　初めに **3**

第 2 章　スタイルガイド **5**

第 3 章　reStructuredText の基礎 **7**

3.1　段落 (Paragraphs) . . . 7
3.2　インラインマークアップ (Inline markup) . . . 7
3.3　リストとクォート (Lists and Quotes) . . . 8
3.4　ソースコード (Source Code) . . . 8
3.5　ハイパーリンク (Hyperlinks) . . . 9
3.6　明示的なマークアップ (Explicit Markup) . . . 10
3.7　ディレクティブ (Directives) . . . 10
3.8　脚注 (Footnotes) . . . 10
3.9　コメント (Comments) . . . 11
3.10　ソースのエンコード (Source encoding) . . . 11
3.11　判っていること . . . 11

第 4 章　拡張マークアップ構成部 (Additional Markup Constructs) **13**

4.1　メタ情報マークアップ (Meta-information markup) . . . 13
4.2　モジュール用のマークアップ (Module-specific markup) . . . 13
4.3　情報単位 (Information units) . . . 14
4.4　コードサンプルを表示する (Showing code examples) . . . 16
4.5　インラインマークアップ (Inline markup) . . . 17
4.6　クロスリンクのマークアップ (Cross-linking markup) . . . 21
4.7　段落レベルでのマークアップ (Paragraph-level markup) . . . 21
4.8　Table-of-contents マークアップ (Table-of-contents markup) . . . 23
4.9　インデックス生成マークアップ (Index-generating markup) . . . 23
4.10　文法導出表記 (Grammar production displays) . . . 24
4.11　置換 (Substitutions) . . . 24

第 5 章　LaTeX マークアップとの違い **25**

5.1 インラインマークアップ . . . 25
5.2 情報単位 (information units) . . . 27
5.3 構造 (Structure) . . . 28

第 6 章 ドキュメントをビルドする **31**
6.1 make を使う方法 . . . 31
6.2 make を使わない方法 . . . 31

第 7 章 このドキュメントについて **33**

第 8 章 翻訳者一覧 (敬称略) **35**

付 録 A 用語集 **37**

付 録 B このドキュメントについて **45**
B.1 Python ドキュメント 貢献者 . . . 45

付 録 C History and License **47**
C.1 Python の歴史 . . . 47
C.2 Terms and conditions for accessing or otherwise using Python . . . 48
C.3 Licenses and Acknowledgements for Incorporated Software . . . 51

付 録 D Copyright **59**

索引 **61**

Python には様々な著者により寄稿された非常に多くのドキュメント類があります。Python のドキュメントのマークアップには、 docutils プロジェクトによって開発された reStructuredText を、独自のディレクティブ (directive) で拡張して利用しています。そして、 Sphinx というツールセットを利用して、HTML 出力へと変換しています。

このドキュメントでは、ドキュメントを作成する上でのスタイルガイドと、Python のドキュメントのために作られた Sphinx による独自の reStructuredText マークアップとその利用方法を説明します。

ノート: Python のドキュメントを寄贈したいと思っているなら、そのために reStructuredText をわざわざ学ぶ必要はありません; 平文での寄贈も大歓迎です。E メールで docs@python.org に送るか、 *Tracker* に課題 (issue) を登録してください。

第 1 章

初めに

Python のドキュメントは長い間、フリーなプログラミング言語としては良いものであると考えられてきました。その理由は多々ありますが、最も重要なのは、Python の作者である Guido van Rossum が、言語やそのライブラリのドキュメントの提供と、ドキュメントの作成と維持の手助けする上でのユーザコミュニティの継続的な参加に早期から関わっていたことです。

コミュニティの参加には、バグ報告の作成から、単にドキュメントをより完全で利用しやすいものにできる場合に素朴な提案をするといったことまで、いくつものやりかたがあります。

このドキュメントは、Python のドキュメントの作者、あるいは潜在的な作者向けのものです。もっと具体的にいうと、標準ドキュメントに貢献したり、標準ドキュメントと同じツールを使って別のドキュメントを開発する人々向けです。このガイドは Python 以外のトピックに Python ドキュメント作成ツールを使う作者にとってはあまり有用ではなく、ツールを全く使用しない作者にもあまり有用ではないでしょう。

仮に、あなたが Python のドキュメントを寄贈したいと思っている一方で、reStructuredText を学んだり、このドキュメントに書かれているマークアップ構造を学んだりする時間や気力を持てないとしても、あなたを Python プロジェクトへの協力者として迎え入れる余地はあります。既存のドキュメントを改善したり、欠けているドキュメントを提供してもらえるなら、現在のドキュメント製作チームがいつでも喜んでマークアップを行い、テキストを組み込みます。手助けしたいという気持ちをお持ちなら、このドキュメントに書かれていることを障害のように思わないでくださいね！

第 2 章

スタイルガイド

Python ドキュメントは、可能な限り Apple Publications Style Guide に準拠することになっています。内容の合理性と、オンラインで容易に取得できることから、このスタイルガイドが選ばれました。

Apple のスタイルガイドがカバーしていないトピックについては、このドキュメントで必要に応じて議論していきます。

脚注は、何かの情報を提供するのにもっとも適した方法である場合は利用されますが、普通はお勧めできません。脚注への参照を文の最後に追加する場合は、句点の後に追加しなければなりません。reST 記法は次のようになります

この文には脚注への参照があります。 [#]_ ここは次の文になります。

脚注はファイルの終端か、ファイルが非常に大きい場合は節の終わりに集められます。docutils は自動的に、脚注の参照への逆リンクを作成します。

脚注は、文の途中でも適切な場所で使用することができます。

Python ドキュメントの中では、オペレーティングシステムやプログラミング言語、標準機関、その他の名前を含む沢山の特殊な名前を使っています。それらの名前のほとんどには特別なマークアップを割り当てていませんが、推奨される表記法をここで提供して、ドキュメント作者が Python ドキュメント内の表現の一貫性を維持しやすくします。

その他の用語や単語についても特に説明しておく必要があるでしょう; ドキュメントの作者はこれらの規約に従い、ドキュメント全体を通して一貫性を保証しなければなりません。

CPU “central processing unit” (中央処理装置) のことです。多くのスタイルガイドが、この語を最初に利用するときには略さずに書かねばならないとしています (ですから、どうしてもこの語を使う必要があるなら、必ずそうしてください!)。Python ドキュメントでは、読者が最初にどこを読むのか合理的に予測する方法がないので、略語の使用を避けねばなりません。代わりに “processor (プロセッサ)” を使う方がよいでしょう。

POSIX ある一連の標準仕様につけられた名前です。常に大文字だけからなります。

Python 私たちの大好きなプログラミング言語の名前は、常に大文字で始めます。

Unicode ある文字セットと、それに対応する符号化方式の名前です。常に大文字で始めます。

Unix 1970 年代初頭に AT&T ベル研究所で開発されたオペレーティングシステムの名前です。

第 3 章

reStructuredText の基礎

この節は reStructuredText (reST) のコンセプトと文法の要約です。ドキュメント作者の生産性のために十分な情報を提供することに注目しています。reST はシンプルで、出しゃばらないマークアップ言語として設計されたので、この文章はあまり長くありません。

参考:

The authoritative reStructuredText User Documentation.

3.1 段落 (Paragraphs)

段落は reST においてもっとも基本的なブロックです。一行以上の空行で区切られただけのテキストの固まりが段落になります。Python と同じく、 reST ではインデントは重要な意味を持つので、同じ段落に属す行は全て同じインデントレベルで左揃えする必要があります。

3.2 インラインマークアップ (Inline markup)

reST 標準のインラインマークアップは非常にシンプルです。

- アスタリスク一つ: `*text*` を強調 (斜体) に、
- アスタリスク二つ: `**text**` を強い強調 (太字) に、
- バッククォート: ``` ``text`` ``` をサンプルコードに、

使ってください。

もしアスタリスクやバッククォートが通常のテキストの中で出てきて、インラインマークアップ用の区切り文字と混合する場合は、バックスラッシュ(訳注: 日本語フォントでは、一般的に円記号になります) でエスケープする必要があります。

インラインマークアップの幾つかの制限に気をつけてください:

- ネストできません。

- マークアップされる内容の先頭や終端に空白文字があってはなりません: `* text*` は間違いです。
- マークアップの外側は、non-word 文字で囲まれていなければなりません。必要な場合はバックスラッシュでエスケープしたスペースを使ってください: `thisis\ *one*\ word`

これらの制限は将来のバージョンの docutils で解除されるかもしれません。

reST は独自の “interpreted text roles” に対応していて、囲まれたテキストを専用の方法で解釈することができます。別の節で解説しますが、Sphinx はこれを、意味に基づくマークアップと識別子のクロスリファレンスのために利用しています。“interpreted text roles” の一般的な文法は `:rolename:`ヽ内容ヽ になります。

3.3 リストとクォート (Lists and Quotes)

リストの表現は自然です: 段落の最初にアスタリスクをおいて、適切にインデントするだけです。番号付きリストも同じで、 # 記号を使えば自動的にナンバリングされます:

* これは番号なしリストです。
* リストの要素は二つあり、二つ目は
  二行使用しています。

#. これば番号付きリストです。
#. こちらも要素が二つあります。

リストはネストさせることもできます。親になるリスト要素との間に空行が必要なことに気をつけてください:

* これは
* リストで

  * ネストされたリストと
  * 子要素があります。

* そしてここは親のリストの続きです。

定義リストは次のようにして作ります:

単語 （行の終わりまで）
    単語の定義。インデントされている必要がある。

    複数の段落を持つことも可能。

次の単語
    定義。

定義の部分は周りの段落よりも深くインデントします。

3.4 ソースコード (Source Code)

リテラルコードブロックは、特別なマーカー `::` で終わる段落の次に始まります。リテラルブロックはインデントしなければなりません。

これは通常の段落です。次の段落はコードサンプルです::

```
ここには、インデントの除去以外の
処理が行われません。

複数行にまたがることもできます。
```

ここでまた通常の段落になります。

:: マーカーの処理はスマートです:

- 単体で段落になっていた場合は、その段落はドキュメントから完全に除去されます。
- マーカーの前に空白があれば、マーカーは削除されます。
- マーカーの前が空白でなければ、マーカーはコロン一つに置き換えられます。

なので、上の例での最初の段落の二つ目の文は、 “次の段落はコードサンプルです:” と出力されます。

3.5 ハイパーリンク (Hyperlinks)

3.5.1 外部リンク (External links)

インラインでの Web リンクには、 `リンク文字列 <http://target>`_ を使ってください。リンク文字列をアドレスにする場合は、マークアップしなくても、パーサーがリンクやメールアドレスを見つけて処理します。

3.5.2 内部リンク (Internal links)

内部リンクには、reST の特別な role を利用します。特別なマークアップのセクションを見てください。 **クロスリンクのマークアップ** *(Cross-linking markup)*

3.5.3 セクション

セクションヘッダは、記号を使って、セクションタイトルにそれ以上の長さのアンダーライン (とオプションでオーバーライン) を引いて作ります。:

```
==============
ここにタイトル
==============
```

通常、特定の文字に特定の見出しレベルが割り当てられておらず、ヘッダ構造から自動的にレベルが決まります。しかし、 Python ドキュメントにおいては、以下のルールを使います:

- # (オーバーライン付き) を編 (part) に
- * (オーバーライン付き) を章 (chapter) に
- = を節 (section) に

- - を項 (subsection) に
- ^ を小区分 (subsubsection) に
- " を段落 (paragraph) に

3.6 明示的なマークアップ (Explicit Markup)

reST において、 “明示的なマークアップ (explicit markup)” は、脚注、特別なハイライト付きの文、コメント、一般的な指定のために利用されます。

明示的なマークアップのブロックは、 .. に空白が続いたものが行頭にあるときに始まり、次の段落が同じインデントになるところで終わります。 (明示的なマークアップと通常の段落の間には空行が必要です。すこし複雑に思えるかもしれませんが、ドキュメントを書くときには十分に直感的です。)

3.7 ディレクティブ (Directives)

ディレクティブは一般的な、明示的なマークアップを行うブロックです。 role のような reST の拡張メカニズムの一つで、 Sphinx はディレクティブを多用しています。

基本的に、ディレクティブは、名前、引数、オプション、内容で構成されます。(この XXX を覚えておいてください。次の章でカスタムディレクティブを説明するときに出てきます。) 次の例を見てください

```
.. function:: foo(x)
              foo(y, z)
   :bar: no

   ユーザーから入力されたテキスト一行を返す.
```

function はディレクティブの名前です。ここでは引数が二つあり、一つは一行目の残りの部分で、もう一つは次の行です。オプションも一つ、 bar があります。(ごらんの通り、オプションは引数の行のすぐ次の行にあり、コロンで示されます.)

一行の空行を挟んでディレクティブの内容が続きます。内容はディレクティブの開始位置と同じ場所までインデントされます。

3.8 脚注 (Footnotes)

脚注を使うときは、 [#]_ を使って脚注を入れる場所を示し、脚注の内容はドキュメントの最後に、次の例のように、” 脚注” という rubic ヘッダの後に書きます。

```
Lorem ipsum [#]_ dolor sit amet ... [#]_

.. rubric:: 脚注

.. [#] 最初の脚注の内容
.. [#] 二つ目の脚注の内容
```

3.9 コメント (Comments)

（上の脚注のような）有効なマークアップになっていない、全ての明示的なマークアップブロックは、コメントとして扱われます。

3.10 ソースのエンコード (Source encoding)

em dash や copyright sign のような特別な文字を reST に含める最も簡単な方法は Unicode 文字を使って直接記述することなので、そのエンコードを決める必要があります。

全ての Python ドキュメントのソースファイルは UTF-8 エンコードでなければなりません。そして HTML ドキュメントも UTF-8 で出力するのが良いでしょう。

3.11 判っていること

reST ドキュメントをオーサリングするときに良く問題になることがあります:

- **インラインマークアップの区切り:** 上で述べたように、インラインマークアップは囲っているテキストと non-word 文字で区切られています。スペースを囲むときにはエスケープが必要になります。

第 4 章

拡張マークアップ構成部 (Additional Markup Constructs)

Sphinx は標準の reST マークアップに対して、たくさんのディレクティブと “interpreted text roles” を拡張しています。このセクションにはそれらの拡張された構成部のリファレンスがあります。“標準” の reST 構成部は、 Python ドキュメントで使用されてはいますが、そのドキュメントはここにはありません。

ノート: これは Sphinx の拡張マークアップの機能の概要です。網羅された情報は *Sphinx* **のドキュメント** <*http://sphinx.pocoo.org/contents.html*> にあります。

4.1 メタ情報マークアップ (Meta-information markup)

sectionauthor
現在のセクションの著者を示します。引数には、(公表されないにしても) 公表されても良いような名前と、email アドレスを含むべきです。アドレスのドメイン名部分は小文字で記述されるべきです。例:

```
.. sectionauthor:: Guido van Rossum <guido@python.org>
```

現在のところ、このマークアップは出力には全く利用されていませんが、だれが貢献したのかを把握するのに役に立っています。

4.2 モジュール用のマークアップ (Module-specific markup)

この節では、ドキュメント中のモジュールに関する情報を提供するために使われるマークアップについて説明します。各モジュールは各々のファイルでドキュメントされるべきです。通常このマークアップは、そのファイルのタイトルヘッダの後に使います。典型的なファイルは次のように始まります:

```
:mod:`parrot` -- Dead parrot access
===================================
```

```
.. module:: parrot
   :platform: Unix, Windows
   :synopsis: Analyze and reanimate dead parrots.
.. moduleauthor:: Eric Cleese <eric@python.invalid>
.. moduleauthor:: John Idle <john@python.invalid>
```

ごらんの通り、モジュール専用マークアップには、 module と moduleauthor という二つのディレクティブを持ちます。

module

このディレクティブはモジュールの説明の始まりを示します。（パッケージのサブモジュールの場合は、モジュール名はパッケージ名を含めて全体を記述すること）

platform オプションは、そのモジュールが利用可能なプラットフォームをカンマで区切ったリストです。（全てのプラットフォームで利用可能であるなら、このオプションは外すべきです）要素は短い識別子で、 “IRIX”, “Mac”, “Windows”, “Unix” などが使われています。できるだけ、すでに使われている識別子を使うようにしてください。

synopsis オプションは、モジュールの目的を説明する一文で構成されます。これは、現在のところ、 Global Module Index でのみ利用されています。

moduleauthor

moduleauthor ディレクティブは、 sectionauthor と同じで、作者の名前になります。このディレクティブは、作者の人数だけ繰り返して利用できます。現在、このディレクティブは出力に利用されていません。

ノート: モジュールを解説するファイルのセクションタイトルは、概要ファイルの中の table-of-contents ツリーに利用されるので、意味が解るようにしてください。

4.3 情報単位 (Information units)

モジュールが提供する機能を解説するために使うディレクティブが幾つかあります。各ディレクティブは、何を説明しようとしているのかを判別する情報として一つかそれ以上のシグネチャを必要とします。そして、ディレクティブの内容はその解説であるべきです。デフォルトではディレクティブはインデックスのエントリに登録されます。インデックスのエントリが必要ない場合は、ディレクティブオプションとして :noindex: というフラグを追加します。次の例は、ここまでで説明した要素を全て含んだディレクティブになります:

```
.. function:: spam(eggs)
              ham(eggs)
   :noindex:

   Spam or ham the foo.
```

オブジェクトのメソッドやデータ属性 (attribute) のシグネチャは、文脈からどの型に属しているかが明らかな場合であっても、 (.. method::FileInput.input(...)) のように型名を含める必要があります。これは、一貫したクロスリファレンスを実現するためです。“context managers” といった抽象プロトコルに属するメソッドを解説する場合にも、インデックスを判りやすくするために、（仮想）型名を付けてください。

ディレクティブは以下の通りです。

c:function
C の関数を説明します。シグネチャは C 言語のまま付けてください。例:

```
.. cfunction:: PyObject* PyType_GenericAlloc(PyTypeObject *type, Py_ssize_t nitems)
```

このディレクティブは関数ライクなプリプロセッサマクロを説明するのにも使います。引数の名前を省略しないでください。引数の名前を説明の中で利用できます。

シグネチャの中のアスタリスクをバックスラッシュでエスケープしなくても良いことを覚えておいてください。reST のインラインに対するパース処理は行われません。

c:member
C の構造体メンバを説明します。シグネチャの例:

```
.. cmember:: PyObject* PyTypeObject.tp_bases
```

説明文は、値の取り得る範囲、値がどのように扱われるか、値を変更しても良いのかどうかについて記述するべきです。テキストの中で構造体のメンバを参照するときには `member` role を利用するべきです。

c:macro
“シンプル” な C 言語のマクロについて説明します。シンプルなマクロとは、引数を取らず、関数として解説されないものです。このディレクティブは単純な定数の定義には利用しません。Python ドキュメントの中でこのディレクティブが使われている例には、`PyObject_HEAD` と `Py_BEGIN_ALLOW_THREADS` があります。

c:type
C の型を説明します。シグネチャは単に型の名前であるべきです。

c:var
C のグローバル変数を説明します。シグネチャは、次の例のように、型を含めるべきです:

```
.. cvar:: PyObject* PyClass_Type
```

data
モジュール内のグローバルなデータを説明します。変数にも、 “定数として宣言された” 値にも利用します。クラスとオブジェクトの属性には使いません。

exception
例外クラスについて説明します。シグネチャは、必要ではありませんが、コンストラクタ引数と丸括弧を含むことができます。

function
モジュールレベル関数を説明します。シグネチャには引数を記述するべきです。オプションの引数は角括弧で囲みます。明快さのために必要であれば、デフォルト値を含めることもできます。例:

```
.. function:: Timer.repeat([repeat=3[, number=1000000]])
```

このディレクティブはオブジェクトメソッドには利用されません。モジュールの名前空間にあり、モジュールの公開インタフェースになっている、束縛済みのオブジェクトメソッド (Bound object method) については、通常の関数とほとんど変わらないので、このディレクティブを使います。

説明文は、必要とされる引数と、それがどのように使われるか（特に、可変 (mutable) オブジェクトが変更されるかどうか）、副作用、発生しうる例外についての情報を含むべきです。小さな例を提供するのも良いでしょう。

class
クラスを説明します。シグネチャには丸括弧とコンストラクタ引数を含めることができます。

attribute
オブジェクトの属性を説明します。説明文は、期待されるデータ型と、直接変更しても良いかどうかを含むべきです。

method
オブジェクトメソッドを説明します。パラメータからは、 `self` パラメータを除外するべきです。説明文は `function` と同じような情報を提供するべきです。

opcode
Python バイトコード (*bytecode*) の命令を説明します。

cmdoption
Python のコマンドラインオプションもしくはスイッチを説明します。オプションの引数名は <> の括弧で囲います。 例:

```
.. cmdoption:: -m <module>

   module をスクリプトとして実行します。
```

envvar
Python が利用したり定義している環境変数を説明します。

もっと汎用的なバージョンの以下のディレクティブもあります:

describe
このディレクティブは、上で説明したディレクティブと同じフォーマットを生成しますが、インデックスエントリやクロスリファレンスターゲットは生成しません。このディレクティブは、たとえば、このドキュメントでディレクティブの説明をするために利用しています。例:

```
.. describe:: opcode

   Python バイトコードの命令を説明します。
```

4.4 コードサンプルを表示する (Showing code examples)

Python ソースコードやインタラクティブセッションの例は、 reST 標準のリテラルブロックを利用して書きます。手前の段落の最後を `::` にして、インデントで範囲を指定します。

インタラクティブセッションを表現するときは、プロンプトと出力を Python コードと一緒に書いてください。インタラクティブセッションに対して特別なマークアップは用意されていません。入力か出力の最後の行の後に、 “使用されない” プロンプトを入れてはいけません。次の例のように **してはいけません**

```
>>> 1 + 1
2
>>>
```

シンタックスハイライトはスマートに処理されます:

- 各ソースファイルには、“ハイライト言語” があります。多数のファイルで Python のコードをハイライトするために、デフォルトでは 'python' に設定されています。
- Python ハイライティングモードでは、インタラクティブセッションは自動的に認識されて適切にハイライトされます。
- ハイライト言語は highlightlang ディレクティブを利用して変更することができます。以下のようにして利用します:

  ```
  .. highlightlang:: c
  ```

  このディレクティブで設定されたハイライト言語は、次の highlightlang ディレクティブまで有効になります。
- ハイライト言語のよく使われる値は以下の通りです:
  - python (デフォルト)
  - c
  - rest
  - none (ハイライトなし)
- 現在のハイライト言語でのハイライティングに失敗した場合、そのブロックは全くハイライトされません。

長い、そのまま表示されるテキストは、外部のプレインテキストのみで書かれたファイルに格納して、取り込む (include) こともできます。その場合、標準の include ディレクティブに literal オプションフラグを付けて利用します。たとえば、 example.py という Python ソースファイルを取り込む場合は:

```
.. include:: example.py
   :literal:
```

4.5 インラインマークアップ (Inline markup)

前に述べたように、 Sphinx はドキュメント内に意味に基づくマークアップを挿入するために、 “interpreted text roles” を使います。

関数/メソッドの引数のようなローカル変数名は例外で、シンプルに *var* とマークされます。

その他の全ての role について、 :rolename:`content` のように書く必要があります。

そのほかにもクロスリファレンス role をより他用途にする便利な機能があります。

- 明示的なタイトルと参照ターゲットを、 reST の直接ハイパーリンクのように書くことができます: :role:`title <target>` は *target* を参照しますが、リンクテキストは *title* になります。
- コンテントに prefix ! を付けると、参照もハイパーリンクも作られません。
- Python オブジェクトのロールにおいて、コンテントに ~ という prefix をつけると、リンクターゲットはターゲットの最後の部分になります。例えば、 :meth:`~Queue.Queue.get` は Queue.Queue.get を参照しますが、リンクテキストとしては get だけを表示します。

HTML 出力において、そのリンクの `title` 属性 (例えばマウスオーバー時のツールチップに表示される) は完全なターゲット名になります。

以下の roles はモジュール内のオブジェクトを参照し、該当する識別子があればハイパーリンクを作成します。

mod
モジュールの名前。ドット付きの名前も使われる。これはパッケージの名前にも使う。

func
Python 関数の名前。ドット付きの名前も使われる。可読性のために、 role のテキストには後ろの丸括弧も含めるべきである。丸括弧は該当する識別子を検索するときには無視される。

data
モジュールレベル変数や定数の名前。

const
定数として “宣言された” 名前。これは C 言語の `#define` か、Python の変更されないことを意図された変数である。

class
クラス名。ドット付きの名前も使われる。

meth
オブジェクトメソッドの名前。 role テキストには型の名前と、メソッド名、後続の丸括弧を含めるべきである。ドット付きの名前も使われる。

attr
オブジェクトのデータ属性の名前。

exc
例外の名前。ドット付きの名前も使われる。

このマークアップで囲まれた名前は、モジュール名とクラス名の両方あるいは片方を含めることができます。たとえば、 `:func:`filter`` は、現在のモジュール内にある `filter` という名前の関数か、その名前のビルトイン関数を参照できます。それに対して、 `:func:`foo.filter`` とすると、はっきりと `foo` モジュールの中の `filter` 関数だけを参照します。

同じようなことが、ある名前が現在ドキュメントしているクラスの属性かどうかを決定する際にも行われます。

以下の roles は、その C 言語の要素が API ドキュメントにあれば、それに対するクロスリファレンスを作成します。

cdata
C 言語の変数の名前。

cfunc
C 言語の関数の名前。後続の丸括弧も含めるべきである。

cmacro
前述した、 “シンプルな” C のマクロの名前。

ctype
C 言語の型の名前。

以下の role はクロスリファレンスは作るかもしれませんが、オブジェクトを参照する事はありません。

token
文法上のトークンの名前。(リファレンスマニュアルにおいて、出力間のリンクを作成するために使われます)

以下の roles はテキストのフォーマットスタイルを変更する以外何もしません。

command
`rm` のような、OS レベルのコマンドの名前。

dfn
テキストの中で定義される語をマークする。 (インデックスエントリは作成されない)

envvar
環境変数。インデックスエントリが作成される。

file
ファイルやディレクトリの名前。この中では、 “可変” な部分を示すために波括弧 “{}” を利用できる。例:

```
... は :file:`/usr/lib/python2.{x}/site-packages` にインストールされます ...
```

ビルドされたドキュメントの中では、この `x` は、 Python マイナーバージョンで置き換えられることを示すために、違った形式で表示されます。

guilabel
インタラクティブなユーザーインタフェースの一部として表示されているラベルは、`guilabel` を使ってマークされるべきです。これには、 `curses` やその他のテキストベースのライブラリを利用して作られた、テキストベースのインタフェースの中のラベルも含みます。ボタンラベル、ウィンドウタイトル、フィールド名、メニューとその項目、選択リスト内の要素など、インタフェース内のどんなラベルにも、この role を利用するべきです。

kbd
キーストロークシーケンスをマークアップします。キーシーケンスをどんな形式で表現するかは、プラットフォームやアプリケーションごとに慣習があります。適切な慣習が無い場合は、初心者や非ネイティブスピーカーにも判るように、修飾キー (modifier key) を省略形にしないでください。例えば、*xemacs* キーシーケンスは、 `:kbd:`C-x C-f`` のように記述できますが、特定のアプリケーションやプラットフォームに関連づけられていない場合は、このキーシーケンスは `:kbd:`Control-x Control-f`` とマークアップされるべきです。

keyword
プログラミング言語の予約後 (keyword).

mailheader
RFC 822 形式のメールヘッダの名前。このマークアップは、そのヘッダが e-mail で利用されることを意味するわけではなく、同じ “スタイル” のどんなヘッダを参照するのにも使えます。多種の MIME 仕様で定義されているヘッダにも利用されます。ヘッダの名前は、実際に利用される場合と同じように書くべきで、一般的な使い方が複数ある場合は camel-case が好まれます。例: `:mailheader:`Content-Type``.

makevar
make の変数名。

manpage
セクションを含む、Unix manual page への参照。例: :manpage:`ls(1)`.

menuselection
メニュー項目は menuselection role を使ってマークアップされるべきです。これは、サブメニューや特定の操作のの選択を含め、完全なメニュー項目の並びや、その一部をマークアップするのに使われます。各項目の名前は --> を使って区切るべきです。

例えば、" スタート > プログラム" をマークアップする場合は、次の様にします:

```
:menuselection:`スタート --> プログラム`
```

幾つかの OS で、メニュー項目の後ろに何か記号を付けてダイアログボックスを開く事を示すといったことがあります。そういったメニュー項目の後ろに続く表記は、メニュー項目名に含めないべきです。

mimetype
MIME type もしくは MIME type の構成要素 (メジャーもしくはマイナー部分だけ) の名前。

newsgroup
Usenet ニュースグループの名前。

option
実行可能プログラムのコマンドラインオプション。先頭のハイフンも含めなければならない。

program
実行可能プログラムの名前。幾つかのプラットフォームでは、実行可能ファイル名と異なるかもしれない。特に、Windows のプログラムでは、 .exe (もしくは他の) 拡張子は除くべきである。

regexp
正規表現。クォートを含めるべきではない。

samp
コードのようなリテラルテキスト。:file: と同じく、この中では "可変" な部分を示すために波括弧を利用できます。

"可変" 部分が要らないのであれば、通常の ``code`` を使ってください。

var
Python か C の、変数か引数の名前。

以下の roles は外部リンクを生成する:

pep
Python Enhancement Proposal への参照。これは適切なインデックスのエントリを生成する。HTML 出力では、 "PEP *number*" というテキストが生成され、このテキストは指定された PEP のオンラインコピーへのハイパーリンクになる。

rfc
Internet Request for Comments (RFC) への参照。これは適切なインデックスのエントリを生成する。HTML 出力では "RFC *number*" というテキストが生成され、このテキストは指定された RFC のオンラインコピーへのハイパーリンクになる。

ハイパーリンクのために特別な role が用意されていないことに注意してください。reST 標準の方法がその目的に利用できるからです。

4.6 クロスリンクのマークアップ (Cross-linking markup)

ドキュメント中の任意のセクションに対してのクロスリファレンスをサポートするには、reST 標準のラベルはあまり良くありません。全てのラベルはセクションタイトルの前におかなければならず、全てのラベルの名前はドキュメントのソース全体に渡ってユニークでなければなりません。

そこで、セクションを参照するのには `:ref:`label-name`` という role を、利用できます。

例:

```
.. _my-reference-label:

クロスリファレンスされるセクション
----------------------------------

セクションの文字列。

このセクション自体を参照します。 :ref:`my-reference-label` を見てください。
```

`:ref:` の部分はセクションタイトルで置き換えられます。

4.7 段落レベルでのマークアップ (Paragraph-level markup)

以下のディレクティブは、通常のテキストと同じように情報単位の中で利用でき、短いパラグラフを作成します。

note
: この note に関係あるどの API を利用するときにも、ユーザーが気をつけるべき特に重要な情報。このディレクティブの内容は完全な文で、適切な句読点を全て含めなければなりません。

warning
: この warning に関係あるどの API を使うときにでも、ユーザーがとても慎重になるべき重要な情報。このディレクティブの内容は完全な文で、適切な句読点を全て含めなければなりません。警告だらけのページでユーザーを怖がらせないように、 `node` ではなく warning を使うのは、クラッシュ、データ損失、セキュリティに関する情報だけに留めるべきです。

versionadded
: このディレクティブは、どのバージョンの Python で対象の要素がライブラリや C API に追加されたのかを示します。このディレクティブがモジュール全体に適用する場合、ディレクティブをモジュールセクションのどの文章よりも先におかれるべきです。

  最初の引数は必須で、バージョンです。二つ目の引数は任意で、変更点の **簡潔な**説明です。

  例:

```
.. versionadded:: 2.5
   *spam* 引数.
```

ディレクティブの先頭行と説明との間に空行を入れてはならないことに注意してください。これはマークアップされたときにブロックが視覚的に連続するためです。

versionchanged
versionadded とほとんど同じですが、対象の要素がいつどのように変更 (新しい引数が追加された、副作用が変わった、等) されたかを説明します。

---

impl-detail
このディレクティブは、 CPython に限定された情報を区別するために使います。ブロック要素としても、一文の引数としても利用できます。例えば

```
.. impl-detail::

   This describes some implementation detail.

   More explanation.
```

または、

```
.. impl-detail:: This shortly mentions an implementation detail.
```

内容の先頭に、自動的に "**CPython implementation detail:**" という一文が入ります。

seealso
たくさんのセクションで、モジュールドキュメントや外部ドキュメントが参照されています。これらのリストは、 seealso ディレクティブで作成されます。

seealso ディレクティブは一般的に、セクションの中で、どのサブセクションより前に置かれます。HTML 出力では、本文の流れから切り離された区画の中に表示されます。

seealso ディレクティブの中身は、 reST の定義リストであるべきです。例:

```
.. seealso::

   Module :mod:`zipfile`
      :mod:`zipfile` 標準モジュールのドキュメント。

   `GNU tar manual, Basic Tar Format <http://link>`_
      GNU tar 拡張を含む、 tar アーカイブファイルのドキュメント。
```

rubric
このディレクティブは、目次 (table of contents) の項目にならない段落見出しを作ります。現在のところ、 "脚注" キャプションに利用されています。

centered
このディレクティブは、センタリングされた太字の段落を作ります。次のようにして使います:

```
.. centered::

   段落の内容
```

4.8 Table-of-contents マークアップ (Table-of-contents markup)

reST が複数のドキュメントを繋いだり、ドキュメントを複数のファイルに分割して出力する機能を持たないので、 Sphinx は table-of-contents を作成したり、ドキュメントの元ファイル間に関連を持たせたりするためにカスタムのディレクティブを利用しています。 `toctree` ディレクティブはその中心になる要素です。

`toctree`

このディレクティブは、ディレクティブの要素として与えられたファイルの中の TOCs ("sub-TOC trees" を含む) から作成した "TOC tree" をその場所に挿入します。`maxdepth` オプションに数値を指定することで、 "TOC tree" の深さを指定できます。デフォルトでは全レベルを利用します。

4.9 インデックス生成マークアップ (Index-generating markup)

Sphinx は自動的にインデックスのエントリを、先に述べた全ての情報の単位 (function, class, attribute のような) から作成します。

しかし、インデックスをより有用なものにしたり、言語リファレンスのような情報が情報の単位の中に含まれないようなドキュメントでもインデックスのエントリを作成できるようにするために、明示的なディレクティブも利用可能です。

そのディレクティブは `index` で、一つかそれ以上のインデックスエントリを含みます。各エントリは、種類と値をコロンで区切ったもので構成されます。

例:

```
.. index::
   single: execution!context
   module: __main__
   module: sys
   triple: module; search; path
```

このディレクティブは５つのエントリを持ち、 index 文の場所へのリンクになっているインデックスエントリに変換されます。(もしくは、オフラインメディアの場合、該当するページ番号になります)

利用可能なエントリの種類は:

single 単独のインデックスエントリを生成します。サブエントリのテキストをセミコロンで区切る（これは以降の種類でも、どんなエントリを作るのかを指定するときに使います）ことによってサブエントリを作成できます。

pair `pair: loop; statement` は、 `loop; statement` と `statement; loop` という名前の二つのインデックスエントリを一度に作成するショートカットです。

triple 同じように、 `triple: module; search; path;` は、 `module; search path, search; path, module, path; module search` というエントリを作成するショートカットです。

module, keyword, operator, object, exception, statement, builtin これらは全て二つのインデックスエントリを作成します。例えば、 `module: hashlib` は、`module; hashlib` と `hashlib; module` を作ります。

4.10 文法導出表記 (Grammar production displays)

形式的な文法の導出を表示するための特別なマークアップが利用可能です。このマークアップはシンプルで BNF (やその派生系) の全ての側面を表そうとはしていませんが、文脈自由文法 (context-free grammer) を、記号が使われている部分からその記号の定義部分へハイパーリンクが張られている形で表記するために十分な能力を提供しています。

productionlist

このディレクティブは導出のグループを囲むために使われます。各導出は一つの行として渡され、名前と、コロンで区切られた残りの定義で構成されます。定義が複数行に渡る場合は、継続する各行は最初の行のコロンと同じ位置にあるコロンで始まらなければなりません。

空行は `productionlist` ディレクティブの引数として許可されていません。

定義には interpreted text としてマークアップされたトークン名を使うことができます。(例: `unaryneg ::= "-" `integer`) – これは、各トークンの導出に対するクロスリファレンスを作成します。代替を示すために利用される縦棒はバックスラッシュでエスケープしなければならないことに気をつけてください。そうしないと、 reST パーサーは縦棒を置換参照 (substitution reference) として認識するからです。

production においては、これ以上の reST パース処理が行われない事に注意してください。なので、 `*` や `|` といった文字をエスケープする必要がありません。

以下は Python リファレンスマニュアルの中の例です:

```
.. productionlist::
   try_stmt: try1_stmt \| try2_stmt
   try1_stmt: "try" ":" :token:`suite`
            : ("except" [:token:`expression` ["," :token:`target`]] ":" :token:`suite`)+
            : ["else" ":" :token:`suite`]
            : ["finally" ":" :token:`suite`]
   try2_stmt: "try" ":" :token:`suite`
            : "finally" ":" :token:`suite`
```

4.11 置換 (Substitutions)

ドキュメントシステムはデフォルトで定義されている 3 種類の置換を用意しています。それらはビルド設定ファイル `conf.py` で設定されます。

|release|

ドキュメントが言及している Python のリリースへ置換されます。これは、例えば `2.5.2b3` のような、 alpha/beta/release candiate を含む完全バージョン文字列です。

|version|

ドキュメントが言及している Python バージョンへ置換されます。これは、たとえばバージョン 2.5.1 において `2.5` の様に、バージョン文字列のうちメジャー・マイナー部のみで構成されます。

|today|

今日の日付か、ビルド設定ファイルで指定された日付のどちらかに置換されます。通常は `April 14, 2007` のようなフォーマットになります。

第 5 章

LaTeX マークアップとの違い

マークアップ言語は変わりましたが、もとの LaTeX ドキュメントにあったコンセプトやドキュメント型 (markup types) はほとんど残っています – 環境 (environments) は reST ディレクティブ、インラインコマンドは reST roles などなど。

しかし、それらがどう動作するのか (the way these work) については、マークアップ言語の違いのため、もしくは Sphinx の改善のために、いくらか異なっています。このセクションでは、古いフォーマットに慣れた人に新しいフォーマットでどう変わったのか概要を示すために、違いをリストアップしていきます。

5.1 インラインマークアップ

インラインマークアップには以下の変更点があります:

- **クロスリファレンス roles**

  以下の semantic roles は以前はインラインコマンドで、 code としてフォーマットする以外には何もしませんでした。今は、既知のターゲットに対してクロスリファレンスになります。(あと、いくつかの名前は短くなっています):

  mod (previously *refmodule* or *module*)
  func (previously *function*)
  data (new)
  const
  class
  meth (previously *method*)
  attr (previously *member*)
  exc (previously *exception*)
  cdata
  cfunc (previously *cfunction*)
  cmacro (previously *csimplemacro*)

ctype

func と *meth* の扱いにも違いがあります: 以前は丸括弧を呼び出し可能オブジェクト名の後ろに (`\func{str()}` のように) 書きましたが、ビルドシステムが丸括弧をつけるようになりました。 – ソースファイルに丸括弧を書くと、出力では二重に括弧がついてしまいます。`:func:`str(object)`` も期待通りになりません。代わりに ``str(object)`` を使ってください!

- **インラインコマンドはディレクティブとして実装されました**

LaTeX にはインラインコマンドがありましたが、 reST ではディレクティブになりました:

deprecated

versionadded

versionchanged

次のようにして使います:

```
.. deprecated:: 2.5
   Reason of deprecation.
```

同じく、 *versionadded* と *versionchanged* のテキストにはピリオドがつきません。*versionchanged*.

note

warning

これらは次のように使います:

```
.. note::

   Content of note.
```

- **その他の変更されたコマンド**

以前の *samp* コマンドは、 code フォーマットでクォーテーションマークで囲まれていました。*samp* role では、 *file* と同じくコードハイライトの機能が新しく追加されました:

`:samp:`open({filename}, {mode})`` results in `open(`*filename*`, `*mode*`)`

- **無くなったコマンド**

次の LaTeX にあったコマンドは、現在 role では対応していません:

bfcode

character (use ``` ``'c'`` ```)

citetitle (use `'Title <URL>'_`)

code (use ``` ``code`` ```)

email (just write the address in body text)

filenq

filevar (use the `{...}` highlighting feature of *file*)

programopt, *longprogramopt* (use *option*)

ulink (use `'Title <URL>'_`)

url (just write the URL in body text)

var (use `*var*`)

infinity, *plusminus* (use the Unicode character)

shortversion, *version* (use the `|version|` and `|release|` substitutions)

emph, *strong* (use the reST markup)

- **バックスラッシュによるエスケープ**

  reST では、バックスラッシュは通常のテキストや role の中ではエスケープしないといけません。しかし、 code リテラルやリテラルブロックではエスケープしてはいけません。例えば: `:file:'C:\\Temp\\my.tmp'` vs. ``` ``open("C:\Temp\my.tmp")`` ```.

5.2 情報単位 (information units)

情報単位 (information units) (latex では *...desc* という環境) は reST ディレクティブで作ります。説明しておかないといけない情報単位に関する変更点は:

- **新しい名前**すべての名前から “desc” が無くなりました。新しい名前は:

  cfunction (以前は *cfuncdesc*)
  cmacro (以前は *csimplemacrodesc*)
  exception (以前は *excdesc*)
  function (以前は *funcdesc*)
  attribute (以前は *memberdesc*)

  classdesc と *excclassdesc* 環境は無くなりました。代わりに、 *class* と *exception* ディレクティブがコンストラクタの引数あり・なしでクラスのドキュメントをサポートします。

- **複数のオブジェクト***...line* というコマンドと等価なのは:

```
.. function:: do_foo(bar)
              do_bar(baz)

   Description of the functions.
```

  言い換えると、同じインデントレベルに複数のシグネチャを一行ずつ書くだけです。

- **引数**

  optional コマンドはありません。単純に関数のシグネチャを出力で表示されるのと同じ形で書いてください。

  ```
  .. function:: open(filename[, mode[, buffering]])

     Description.
  ```

  注意: シグネチャの中ではマークアップはサポートされません。

- **Indexing**

  ...descni 環境は無くなりました。情報単位をインデックスエントリに含めないようにするには、*noindex* オプションを次のように利用してください:

  ```
  .. function:: foo_*
     :noindex:

     Description.
  ```

- **新しい情報単位**

  新しい汎用情報単位があります。一つは “describe” と呼ばれ、他の情報単位の対象にならない単位に使うことができます:

  ```
  .. describe:: a == b

     The equals operator.
  ```

  他には次のような単位があります:

  ```
  .. cmdoption:: -O

     Describes a command-line option.

  .. envvar:: PYTHONINSPECT

     Describes an environment variable.
  ```

5.3 構造 (Structure)

LaTeX ドキュメントはいくつかのトップレベルマニュアルに分割されていました。今は、すべてのファイルは *toctree* ディレクティブで指定される一つのドキュメントツリーの一部です。(各出力フォーマットでまたファイルを分割することもできます) すべての *toctree* ディレクティブは他のファイルを現在のファイルのサブドキュメントとして埋め込みます。(この構造をファイルシステムレイアウトに反映させる必要はありません) トップレベルのファイルは `contents.rst` です。

しかし、今までのディレクトリ構造の大部分は、次のように名前を変更されながら残っています:

- `api` -> `c-api`
- `dist` -> `distutils`, with the single TeX file split up
- `doc` -> `documenting`

- `ext` -> `extending`
- `inst` -> `installing`
- `lib` -> `library`
- `mac` -> merged into `library`, with `mac/using.tex` moved to `using/mac.rst`
- `ref` -> `reference`
- `tut` -> `tutorial`, with the single TeX file split up

第 6 章

ドキュメントをビルドする

ドキュメントをビルドするために使われるツールセットは Python で書かれているので、 Python 2.4 以上をインストールしておく必要があります。このツールセットは *Sphinx* と呼ばれていて、 Python ソースコードツリーには含まれておらず、別に管理されています。他にも、 Sphinx が利用しているマークアップの基礎を提供している docutils と、テンプレートエンジンの Jinja, オプションでコードハイライトをしたい場合は Pygments が必要です。

6.1 make を使う方法

Makefile が用意されているので、 Python と Subversion がインストールされている Unix 環境では、シンプルに次のコマンドを実行するだけです。

```
make html
```

このコマンドは必要なツールセットを *tools/* サブディレクトリ配下にチェックアウトし、 HTML 形式の出力ファイルをビルドします。出力された HTML ファイルを見るには、 “make” が完了してから、お好きなブラウザでトップページになる *build/html/index.html* ファイルを開いてください。

利用できる make の target は次のとおりです。

“make update” を実行すると、 *tools/* 以下の Subversion チェックアウトを更新します。

6.2 make を使わない方法

Sphinx パッケージを、 PyPI からインストールするか、次のコマンドでチェックアウトしてください。

```
svn co http://svn.python.org/projects/external/Sphinx-0.6.5/sphinx tools/sphinx
```

次に、 docutils を、 http://docutils.sf.net/ からインストールするか、次のコマンドでチェックアウトしてください。

```
svn co http://svn.python.org/projects/external/docutils-0.6/docutils tools/docutils
```

Jinja2 も必要です。 PyPI からインストールするか、次のコマンドでチェックアウトしてください。

```
svn co http://svn.python.org/projects/external/Jinja-2.3.1/jinja2 tools/jinja2
```

任意で、 Pygments をインストールすることもできます。 PyPI の http://pypi.python.org/pypi/Pygments からインストールするか、次のコマンドでチェックアウトしてください。

```
svn co http://svn.python.org/projects/external/Pygments-1.3.1/pygments tools/pygments
```

出力ディレクトリ (例では *build/* とします) を作成し、次のコマンドを実行します。

```
python tools/sphinx-build.py -b<builder> . build/<出力ディレクトリ>
```

<*builder*> は、 html, text, latex, htmlhelp のどれかです。(説明は上にある make の target を参照してください)

第 7 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 “Documenting Python” の日本語訳版です。日本語訳に対する質問や提案などがありましたら、Python ドキュメント翻訳プロジェクトのメーリングリスト

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

または、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。

第 8 章

翻訳者一覧 (敬称略)

- 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

Python and this documentation is:

Copyright 2001-2010 Python Software Foundation. All rights reserved.

Copyright 2000 BeOpen.com. All rights reserved.

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

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

Japanese translation is: Copyright 2003-2009 Python Document Japanese Translation Project. All rights reserved.

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

索引

..., **37**
__future__, **39**
__slots__, **44**
>>>, **37**
2to3, **37**

abstract base class, **37**
argument, **37**
attribute, **37**

BDFL, **37**
bytecode, **38**

class, **38**
classic class, **38**
coercion, **38**
complex number, **38**
context manager, **38**
CPython, **38**

decorator, **38**
descriptor, **39**
dictionary, **39**
docstring, **39**
duck-typing, **39**

EAFP, **39**
expression, **39**
extension module, **39**

finder, **39**
function, **39**

garbage collection, **40**
generator, 40, **40**
generator expression, 40, **40**

GIL, **40**
global interpreter lock, **40**

hashable, **40**

IDLE, **41**
immutable, **41**
importer, **41**
integer division, **41**
interactive, **41**
interpreted, **41**
iterable, **41**
iterator, **41**

keyword argument, **42**

lambda, **42**
LBYL, **42**
list, **42**
list comprehension, **42**
loader, **42**

mapping, **42**
metaclass, **42**
method, **42**
mutable, **42**

named tuple, **42**
namespace, **43**
nested scope, **43**
new-style class, **43**

object, **43**

positional argument, **43**
Python 3000, **43**

Python Enhancement Proposals
    PEP 302, 39, 42
    PEP 343, 38
Pythonic, **43**

reference count, **43**

sequence, **44**
slice, **44**
special method, **44**
statement, **44**

triple-quoted string, **44**
type, **44**

virtual machine, **44**

Zen of Python, **44**