# Python ドキュメントの作成

Fred L. Drake, Jr.
日本語訳: Python ドキュメント翻訳プロジェクト

September 20, 2004
PythonLabs
Email: fdrake@acm.org

**Abstract**

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 のドキュメント作成に使われるマークアップは LaTeX に基づいていて、Python のドキュメント作成用に特別に書かれたかなりの数のマクロセットを必要とします。このドキュメントでは、Python のドキュメント作成をサポートするために採用されているマクロ、および広範な出力形式をサポートするためにマクロをどのように使うかについて述べます。

このドキュメントでは、Python ドキュメント作成で使うドキュメントクラスと特殊なマークアップについて述べます。ドキュメントの作者は、このガイドと Python 配布物で提供されている雛形ファイルを組み合わせて使い、ドキュメント全体やドキュメントの各セクションを作成したり維持したりできます。

## Contents

# 1 初めに

Python のドキュメントは、このフリーなプログラミング言語の長所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、単にドキュメントがより完全で利用しやすいものになりうる場合に素朴な提案を発するといったことまで、数多くの形式があります。作者がドキュメントの維持に参加している中で、こうしたコミュニティからの入力の全てが有用なものであると証明されました。

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。

このガイドに書かれていることは、Python ドキュメント作成ツールを使う著者の支援を目的としています。ガイドには、標準ドキュメントのソース配布に関する情報、ドキュメントタイプに関する議論、ドキュメントクラス中でで定義されているマークアップについてのリファレンス、ドキュメントを処理する上で必要な外部ツールのリスト、ドキュメントリソースと共に提供されているツールに関するリファレンスが入っています。ガイドの末尾には、Python ドキュメントの将来の方向性と、詳細な情報をどこで取得できるかについて議論した章もあります。

# 2 ディレクトリ構造

標準 Python ドキュメントのソース配布物には、数多くのディレクトリが入っています。サードパーティ製のドキュメントは、このディレクトリ構造下や似たディレクトリ構造下におく必要はありませんが、Python ドキュメント作成ツールを利用して新たなドキュメントを開発する際に、どこで例題やツールを探せばよいか知っておけば便利でしょう。そこでこの節では、ディレクトリ構造について述べます。

ドキュメントソースは通常、'Doc/' をトップレベルディレクトリとして Python ソース配布物中に配置されていますが、Python ソース配布物には全く依存していません。

'Doc/' ディレクトリには、いくつかのファイルとサブディレクトリが入っています。ファイルは 'README' や 'Makefile' を含め、ほとんどが自明なものです。ディレクトリは以下の三つのカテゴリーに分類されます:

**ドキュメントソース**
各ドキュメントの LaTeX ソースは個別のディレクトリに配置されています。ディレクトリには、各々のドキュメントを漠然と説明する短い名前がついています:

| ディレクトリ | タイトル |
|---|---|
| api/ | *Python/C API* |
| dist/ | *Python* **モジュールの配布** |
| doc/ | *Python* **ドキュメントの作成** |
| ext/ | *Python* **インタプリタの拡張と埋め込み** |
| inst/ | *Python* **モジュールのインストール** |
| lib/ | *Python* **ライブラリリファレンス** |
| mac/ | *Macintosh* **モジュールリファレンス** |
| ref/ | *Python* **リファレンスマニュアル** |
| tut/ | *Python* **チュートリアル** |

**出力書式ごとの出力ディレクトリ**

ほとんどの出力書式に対して、個々の出力書式の生成を制御する 'Makefile' が入っている、書式化されたドキュメントを格納するためのディレクトリがあります。このカテゴリの例外となるのは Portable Document Format (PDF) および PostScript 版で、これらは 'paper-a4/' および 'paper-letter/' に配置されます (こうすることで、LaTeX が生成する全ての一時ファイルは用紙サイズごとに同じ場所に置かれ、目立たなくできます)

| ディレクトリ | 出力書式 |
|---|---|
| html/ | HTML 形式 |
| info/ | GNU info 形式 |
| isilo/ | iSilo ドキュメント (Palm OS 機器用) |
| paper-a4/ | PDF と PostScript、A4 サイズ |
| paper-letter/ | PDF と PostScript、US-Letter サイズ |

**補助ファイル群**

その他のディレクトリは、様々な処理で用いられる補助ファイル群を格納するためのものです。補助ファイル群は、共有される LaTeX ドキュメントクラス、LaTeX2HTML のサポートファイル、様々なドキュメント構成要素のための雛形ファイル、書式化処理の様々な処理段階を実現するスクリプトなどです。

| ディレクトリ | 内容 |
|---|---|
| commontex/ | ドキュメント間で共通の文章 |
| perl/ | LaTeX2HTML 処理のサポート |
| templates/ | ソースドキュメントの例 |
| texinputs/ | LaTeX スタイルファイル |
| tools/ | 独自の処理用スクリプト |

# 3 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。

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が使われています。こうした名前の多くはかなり古い時期に LaTeX マクロに割り当てられ、その便利さから長い間利用されつづけてきました。現在のマークアップでは、こうしたエンティティのほとんどには特殊なマークアップを割り当てません。その代わり、推奨される表記法を提供して、ドキュメント作者が Python ドキュメントにおける表現の一貫性を維持できるよう手助けしています。

その他の用語や単語についても特筆する必要があります; 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、かならずそうしてください!)。 Python ドキュメントでは、読者がこの略語を最初に読むのがどこかを予測できる合理的な方法がないので、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** ある特定の標準仕様グループにつけられた名前です。常に大文字だけからなります。この名前を表現するにはマクロ `\POSIX` を使用してください。

**Python** 私たちの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** 文字セットと、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。この名前を使うにはマクロの `\UNIX` を使用してください。

# 4 LaTeX 入門

この節では、LaTeX の概念と構文法について簡単に紹介し、ドキュメントの作者が “TeXnician” にならなくてもドキュメントを十分生産的に書けるような情報を提供します。

おそらく、Python ドキュメントのマークアップにおいて心に留めておくべき最も重要な概念は、TeX が非体系的なのに対して、LaTeX は TeX の上のレイヤとして設計されていて、本質的には構造化マークアップをサポートしているということです。 Python 特有のマークアップは、標準の LaTeX ドキュメントクラスで提供されている構造を拡張し、Python 特有の情報のサポートを目的としています。

LaTeX ドキュメントには、二つの構成要素: プリアンブルと本体が入っています。プリアンブルは、タイトル、著者のリスト、日付、ドキュメントが属する **クラス** といった、ドキュメント自体に関するメタデータを指定するために使用されます。その他の情報で、インデクスの生成や参考文献データベースの使用を制御するものもプリアンブル内に置けます。ほとんどの著者にとっては、既存のドキュメントからプリアンブルをコピーし、いくつかの情報を修正するだけで、簡単にプリアンブルを作成できます。

ドキュメントの**クラス** は、広範なドキュメントカテゴリにおける位置づけを行い、基本的な書式化プロパティを設定するために使われます。 Python ドキュメントの場合、二つのクラス: `manual` クラスと `howto` クラスが使われています。これらのクラスではまた、Python における概念やデータ構造を記述する際に使う追加のマークアップを定義しています。これらのクラスに関する詳しい説明は後述の 5 節、 “ドキュメントクラス” にあります。プリアンブルの最初に置かれているのが、ドキュメントクラスの宣言です。

クラス宣言の後ろには数多くの**マクロ** があり、ドキュメントに関する詳しい情報を与えたり、追加の必要なマークアップを設定しています。プリアンブルからは何も出力生成されません; 自由文 (free text) は出力生成につながるため、プリアンブルに自由文をおくとエラーになります。

ドキュメント本体はプリアンブルの後に続けます。本体には、出力すべきドキュメントの構成要素を構造的にマークアップして入れます。一般的な LaTeX の構造には、階層化された章節、番号付けリストや箇条書きリスト、そしてドキュメントの概要や索引といった特殊な構造があります。

## 4.1 構文法

Python ドキュメントの作者は、LaTeX の構文法についていくつか知っておかねばならないことがあります。

**コメント** は、“パーセント” 文字 (‘%’) から始まり、行末まで、または次の行の先頭にある空白文字まで続きます。これは著者の知っているどのプログラミング言語とも少し違っています。そこで、順序立てて例を挙げます:

```
This is text.% comment
    This is more text.  % another comment
Still more text.
```

最初の行のコメントの後ろに続くコメントでない文字は、二行目の ‘T’ になります; 二行目の先頭にある空白文字は、最初の行のコメントの一部として消費されてしまいます。これは、最初と二つ目の文との間には全くスペースがなく、最初の行のピリオドと二行目の文字 ‘T’ はタイプセット時に直接くっつけることを意味します。

二つめのコメントの直後にくる非コメント文字は ‘S’ ですが、二つ目のコメントの直前に空白を入れているため、意図通りに二つの単語が分割されることにも注意してください。

**グループ** は一連のテキストやコマンドに対する囲いで、書式化処理のコンテキストを囲い、グループ内のコマンドが処理コンテキストに及ぼす変更の範囲を制限します。グループは入れ子構造にして階層化で

きます。書式化処理のコンテキストは、フォントや、追加のマクロ定義 (またはグループ外部で定義されているマクロに対するオーバライド) を含みます。構文法的には、グループは波括弧で囲います:

```
{text in a group}
```

グループを表すには、ブラケット `[...]` を使う代用構文もあります。この代用構文はマクロや環境のコンストラクタでオプションのパラメタをとるものが使います; ブラケットは通常、構文的な意味を持っていません。内容が唯一の原始的要素 (atomic bit) からなるような縮重グループ (degenerate group) の場合、あいまいさを防ぐために必須である場合をのぞき、明示的なグループにする必要はありません。とはいえ、Python は明示的であることをよしとするので、ドキュメントのマークアップにおいてもグループを明示します。

マクロや環境におけるパラメタをマークする場合を除き、Python ドキュメントではグループを控えめに使います。

**マクロ** は通常は単純な構成要素で、マクロ名で識別され、いくつかのパラメタを取ることがあります。通常の LaTeX の使用法では、パラメタのうちの一つがオプションになっていることがあります。マークアップはバックスラッシュ文字 (‘\’) から始め、マクロ名はアルファベット文字 (数字、ハイフン、アンダースコア以外) で与えます。必須パラメタはグループとしてマークし、オプションのパラメタはグループの代用構文でマークせねばなりません。

例えば、 単一のパラメタにとる “foo” という名のマクロは以下のようになります:

```
\name{parameter}
```

オプションのパラメタをとるマクロで、パラメタを指定するときには以下のように入力します:

```
\name[optional]
```

オプションと必須パラメタの両方が必要な場合には、以下のようになります:

```
\name[optional]{required}
```

マクロ名の後ろには空白や改行を入れてもかまいません; この場合、マクロ名とパラメタ間の空白は取り去られてしまいます。しかし、Python ドキュメントではこうした使い方を実践しません。空白はマクロにパラメタがない場合にも取り去られますが、この場合には空のグループ (`{}`) や、空白の明示的表現 (‘`\ `’) をマクロ名の直後に続けると、マクロ展開が後続の文字に及ぶのを避ける手助けになります。パラメタを取らないが、後ろに空白を続けたくないマクロの場合には、ドキュメントソース中で名前の後ろに名前に使わない文字 (区切り文字など) が入っていれば、特別扱いする必要はありません。

例題中の各行は、パラメタをとらないマクロが入った文章を書くための適切な方法を示しています。

```
This \UNIX{} is followed by a space.
This \UNIX\ is also followed by a space.
\UNIX, followed by a comma, needs no additional markup.
```

**環境** はマクロよりも大きな構文要素で、マクロの引数に入れるには内容がやや大きすぎて便利さを欠く場合に利用できます。本文の大きな断片の前後で書式化パラメタを変更する必要があり、かつ本文の記述に高い柔軟性を持たせたい場合に主に利用されます。コード例は環境を使って表現します。また、関数、メソッド、クラスの定義も環境を使ってマークします。

環境内に記述する内容は自由形式で複数の段落にできるため、実際には二つのマクロからなるペア: `\begin` および `\end` を使ってマークします。これらのマクロはいずれも環境の名前をパラメタにとります。以下の例は、ドキュメントの概要をマークするために使う環境です:

```
\begin{abstract}
  This is the text of the abstract.  It concisely explains what
  information is found in the document.

  It can consist of multiple paragraphs.
\end{abstract}
```

環境は、環境自体の必須パラメタやオプションパラメタを持つこともあります。これらのパラメタは `\begin` マクロのパラメタの後ろに続けます。以下に単一の必須パラメタをとる環境の例を示します:

```
\begin{datadesc}{controlnames}
  A 33-element string array that contains the \ASCII{} mnemonics for
  the thirty-two \ASCII{} control characters from 0 (NUL) to 0x1f
  (US), in order, plus the mnemonic \samp{SP} for the space character.
\end{datadesc}
```

ASCII に含まれない文字、特殊文字とされている文字、TEX や LATEX で **アクティブ** *(active)* な文字を入力するための、あまり使われないマークアップがあります。これらのマークアップはよく他の文字に隣接して使われるので、適切な文字を生成するためには、マークアップの後ろにスペースや空のグループを置く必要があるかもしれません。あるいは、マークアップをグループ内に囲うこともできます。Python ドキュメントで使われるようなマークアップを以下にいくつか示します:

| キャラクタ文字 | マークアップ |
|---|---|
| ^ | `\textasciicircum` |
| ~ | `\textasciitilde` |
| > | `\textgreater` |
| < | `\textless` |
| ç | `\c c` |
| ö | `\"o` |
| ø | `\o` |

## 4.2 階層構造

LATEX は、章、セクション、サブセクション、付録などといった、伝統的な階層化された方法でドキュメントが整理されると想定しています。これらの各階層は環境ではなくマクロでマークされますが、おそらくこれはある章構造とレベルが同じかより高い章構造が続くと、その章がそこで終わったと考えて差し支えないからでしょう。

Python ドキュメントが使っているクラスには 6 段階の章節分けの “レベル” があり、そのうち最も深い 2 レベル[1] は使いません。

| レベル | マクロ名 | 注釈 |
|---|---|---|
| 1 | `\chapter` | (1) |
| 2 | `\section` | |
| 3 | `\subsection` | |
| 4 | `\subsubsection` | |
| 5 | `\paragraph` | (2) |
| 6 | `\subparagraph` | |

注釈:

**(1)** 5 節、“ドキュメントクラス” で述べたように、`manual` ドキュメントだけで使います。

**(2)** テキストの段落とは違います; このレベルは誰も使っていないようです。

[1] 表中では、深いレベルほど番号が大きくなります。

# 5　ドキュメントクラス

2 つの LaTeX ドキュメントクラスが Python ドキュメントで使うために定義されています。`manual` クラスは、章単位で区分するような大きなドキュメントのためのクラスです。`howto` クラスはより小さなドキュメントのためのクラスです。

`manual` ドキュメントは大規模で、ほとんどの標準ドキュメントが使うクラスです。このドキュメントクラスは標準 LaTeX の `report` クラスに基づいていて、まるで長い技術的報告書のように書式化されます。ます。*Python* **リファレンスマニュアル** は `manual` ドキュメントのよい例です。また、*Python Library Reference* は大規模な文書の例です。

`howto` ドキュメントはより短く、`manual` ドキュメントのような大きな構造を持ちません。このクラスは標準 LaTeX の `article` クラスに基づいていて、Linux Documentation プロジェクトの “HOWTO” シリーズのように書式化されます。“HOWTO” シリーズはもともと LinuxDoc ソフトウェアを使って書式化を行っています。このドキュメントクラスの本来の目的は、 LDP の HOWTO シリーズと同様の役割として役立つことでしたが、このクラスにはより広範囲な適用性があることが判明したのです。このクラスは “how-to” ドキュメント (このドキュメントがその一例です) や、小規模で、かなり強い関連性で結ばれたモジュールライブラリ群のリファレンスマニュアルに用いられます。後者のドキュメントクラスの例には *Using Kerberos from Python* があります。これらのドキュメントは、大きなドキュメントにおける一個の章とだいたい同じ規模です。

# 6　特殊マークアップ構文

Python ドキュメントのクラスでは、環境やマクロを新たに数多く定義しています。この節にはこれらの機能についてのリファレンスマニュアルがあります。

## 6.1　プリアンブル用のマークアップ

**\release**{*ver*}
　このマークアップは、ドキュメント中で述べているソフトウェアのバージョン番号を設定します。

**\setshortversion**{*sver*}
　このマークアップは、ドキュメントが扱っているソフトウェアの “短い形式の” バージョン番号を *sver* にします。

## 6.2　メタ情報マークアップ

**\sectionauthor**{*author*}{*email*}
　現在の節の作者を指定します。*author* は作者の名前で、著者紹介に使える形式になっていなければなりません (もっとも、実際には使われません)。また、*email* は作者の電子メールアドレスです。アドレスのドメインネーム部分は小文字にしなければなりません。

　このマークアップを行っても著者紹介は生成されませんが、だれがドキュメントに貢献したかを追跡する上での補助として用いられます。

## 6.3　情報単位

モジュールが提供している特定の機能を説明するために使うマクロが数多くあります。これらの各環境はパラメタをとり、その環境で何を説明しようとしているかを表す基本的な情報を与える必要があります。そして、説明内容は環境の中で記述します。こうした環境のほとんどは、(ドキュメントが総合索引を生成する場合には) 総合索引上のエントリになります; 索引エントリが必要ない場合、各環境について索引を生成しない変化形を利用できます。環境は *feature*`desc` という形式の名前になり、索引を生成しない変化形は *feature*`descni` という形式の名前になります。利用可能な変化形がある場合は以下のリスト内に明示しています。

各々の環境について、最初のパラメタ *name* は、アクセスする機能を指定する名前です。

オブジェクトのメソッドやデータ属性といった、モジュール内のオブジェクトのもつ機能を記述するための環境では、オプションの *type name* パラメタを指定できます。該当する機能がクラスインスタンスの属性である場合、*type name* を与える必要があるのは、クラスがそのモジュールの説明で最後に説明した

クラスでない場合だけです; それ以外の場合、最後に説明した `classdesc` の *name* が適用されます。組み込み型や拡張型の機能については、*type name* を常に与えねばなりません。もう一つ特殊なケースとして、`formatter` モジュールにおける formatter や writer のような、汎用の “プロトコル” のメソッドやメンバがあります: これらのドキュメントは特定の実装クラスを伴わず、常に *type name* パラメタを指定する必要があります。

`\begin{`**`cfuncdesc`**`}{`*type*`}{`*name*`}{`*args*`}`
`\end{`**`cfuncdesc`**`}`
: C 関数を解説するときに使う環境です。*type* は `typedef` されている型名か、 `struct` *tag* か、基本型の名前でなければなりません。ポインタ型の場合、後続するアスタリスクの前にスペースを入れてはなりません。*name* は関数 (または関数に似た形式をとるプリプロセッサマクロ) の名前でなくてはならず、*args* にはパラメタの型と名前を与えねばなりません。名前は説明文内でも使える形式で書かねばなりません。

`\begin{`**`cmemberdesc`**`}{`*container*`}{`*type*`}{`*name*`}`
`\end{`**`cmemberdesc`**`}`
: 構造体メンバを説明するときに使う環境です。*container* には、`typedef` されている名前ならそれを使い、そうでなければ ‘`struct` *tag*’ と指定せねばなりません。メンバの型は *type* に指定し、メンバ名は *name* に指定します。説明文中には変数の許容範囲、値がどのように解釈されるか、そして値が変更可能かを記述せねばなりません。本文中で構造体メンバを参照する場合には、 `\member` マクロを使います。

`\begin{`**`csimplemacrodesc`**`}{`*name*`}`
`\end{`**`csimplemacrodesc`**`}`
: “単純” マクロを説明するときに使う環境です。単純マクロは、引数を取らないため関数で記述できないようなコードの展開に使われます。この環境は単純な定数定義には用いません。Python ドキュメント内でのこのマクロの用例としては、`PyObject_HEAD` や `Py_BEGIN_ALLOW_THREADS` があります。

`\begin{`**`ctypedesc`**`}[`*tag*`]{`*name*`}`
`\end{`**`ctypedesc`**`}`
: C の型を説明するときに使う環境です。*name* パラメタは `typedef` された名前でなければなりません。型が `typedef` を伴わずに `struct` で定義されている場合、 *name* は `struct` *tag* の形式を取らねばなりません。*tag* がない限り *name* が索引に追加されます。*tag* がある場合、 *tag* を索引に使います。*tag* は `typedef` された名前であってはなりません。

`\begin{`**`cvardesc`**`}{`*type*`}{`*name*`}`
`\end{`**`cvardesc`**`}`
: グローバルな C の変数を説明するときに使う環境です。*type* は `typedef` されている型名か、`struct` *tag* か、基本型の名前でなければなりません。ポインタ型の場合、後続するアスタリスクの前にスペースを入れては**なりません**。

`\begin{`**`datadesc`**`}{`*name*`}`
`\end{`**`datadesc`**`}`
: この環境は、モジュールにおけるグローバルなデータを説明する際に使います。グローバルなデータには、変数と “定義済み定数 (defined constant)” として使われる値の両方を含みます。クラスやオブジェクト属性の記述にはこの環境を使いません。

`\begin{`**`datadescni`**`}{`*name*`}`
`\end{`**`datadescni`**`}`
: `datadesc` に似ていますが、索引エントリを作成しません。

`\begin{`**`excclassdesc`**`}{`*name*`}{`*constructor parameters*`}`
`\end{`**`excclassdesc`**`}`
: クラスとして定義されている例外を説明するために使います。*constructor parameters* には、*self* やコンストラクタ呼び出し構文で使う丸括弧を含めてはなりません。コンストラクタに与えるパラメタの説明がないような例外クラスを記述するには、`excdesc` 環境を使用してください。

`\begin{`**`excdesc`**`}{`*name*`}`
`\end{`**`excdesc`**`}`
: 例外を説明するための環境です。クラス例外の場合には、コンストラクタのパラメタが記述されません: 例外クラスとコンストラクタを説明する場合には、`excclassdesc` を使ってください。

`\begin{`**`funcdesc`**`}{`*name*`}{`*parameters*`}`
`\end{`**`funcdesc`**`}`
: モジュールレベルの関数を説明するための環境です。*parameters* には、呼び出しの際に使う丸括弧を

含めてはなりません。オブジェクトメソッドはこの環境で記述しません。束縛オブジェクトメソッドをモジュールの公開インタフェースの一部としてモジュールの名前空間に配置している場合、メソッドはほとんどの用途でモジュールレベル関数と等価であるため、この環境を使って記述します。

説明には、必要なパラメタと、パラメタがどう処理されるか (とりわけ、パラメタとして渡した変更可能なオブジェクトが変更されるか否か) 、副作用、送出される可能性のある例外についての情報を含めねばなりません。小規模の例題を与えてもかまいません。

`\begin{`**`funcdescni`**`}{`*name*`}{`*parameters*`}`
`\end{`**`funcdescni`**`}`
: `funcdesc` に似ていますが、索引エントリを作成しません。

`\begin{`**`classdesc`**`}{`*name*`}{`*constructor parameters*`}`
`\end{`**`classdesc`**`}`
: クラスとそのコンストラクタを説明するための環境です。*constructor parameters* には、*self* やコンストラクタ呼び出し構文で使う丸括弧を含めてはなりません。

`\begin{`**`classdesc*`**`}{`*name*`}`
`\end{`**`classdesc*`**`}`
: コンストラクタの説明を伴わないクラスを説明するための環境です。属性をコンテナ化することがほとんどなかったり、ユーザコードからインスタンス化やサブクラス化を行わないようなクラスを記述する際に利用できます。

`\begin{`**`memberdesc`**`}[`*type name*`]{`*name*`}`
`\end{`**`memberdesc`**`}`
: オブジェクトのデータ属性を説明するための環境です。説明には、想定されているデータ型や、データが直接変更可能かについての情報がなければなりません。

`\begin{`**`memberdescni`**`}[`*type name*`]{`*name*`}`
`\end{`**`memberdescni`**`}`
: `memberdesc` に似ていますが、索引エントリを作成しません。

`\begin{`**`methoddesc`**`}[`*type name*`]{`*name*`}{`*parameters*`}`
`\end{`**`methoddesc`**`}`
: オブジェクトメソッドを説明するための環境です。*parameters* には、*self* パラメタや呼び出し構文で使う丸括弧を含めてはなりません。説明には、`funcdesc` 内で説明するのと同じような情報が入っていなければなりません。

`\begin{`**`methoddescni`**`}[`*type name*`]{`*name*`}{`*parameters*`}`
`\end{`**`methoddescni`**`}`
: `methoddesc` に似ていますが、索引エントリを作成しません。

## 6.4 コードの例示

Python ソースコードや対話セッションの例は `verbatim` 環境で表現します。この環境は LaTeX の標準です。TeX はタブをスペースに変換せずに捨てるので、コード例中でのインデントにはスペースだけを使用することが重要です。

対話セッションの表現には、プロンプトや Python コード由来の出力を含める必要があります。対話セッションには特殊なマークアップを行う必要はありません。最後の入力行または出力行を示した後に、“未使用の” 一次プロンプトを置いてはなりません; 以下に**やってはいけない** 例を示します:

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

`verbatim` 環境の中では、LaTeX の特殊文字を何がしかの特殊な方法でマークする必要はありません。例題は全体が等幅フォントで表示されます; この環境は Python コードの表示でなくても、コードの表示でさえなくても使えなければならないため、“整形印刷” は一切行おうとしません。`verbatim` で表示する際には、先頭や末尾に空行を入れてはなりません。

長い `verbatim` テキストを表示する場合には、平文テキストだけの入った例題のテキストを外部ファイルに入れて取り込めます。ファイルは標準マクロ `\verbatiminput` で取り込めます; このマクロは単一の引数名として、例題テキストの入ったファイル名をとります。例えば、 ‘example.py’ の入った Python ソースファイルを取り込むには、以下のようにします:

```
\verbatiminput{example.py}
```

`\verbatiminput` を使うと、取り込むファイルに対して特殊な編集モードを使いやすくできます。ファイルはドキュメントの LaTeX ファイルと同じディレクトリに置かねばなりません。

Python Documentation Special Interest Group では、コード表示や対話セッションに対して整形印刷を行うための数々のアプローチについて議論してきました; この話題に関する詳細は、Python Web サイトの Doc-SIG エリアを参照してください。

## 6.5 インラインマークアップ

この節で説明するマクロは、単にドキュメントテキスト中の興味深い内容をマークするために使われます。これらのマクロは、本文だけではなく、表題に利用してもかまいません (とはいえ、ハイパーリンクを含むものは除かねばなりません)。

**\bfcode**{*text*}
: `\code` に似ていますが、ボールド書体にします。

**\cdata**{*name*}
: C 言語における変数の名前です。

**\cfunction**{*name*}
: C 言語における関数の名前です。*name* には関数名とその後ろの丸括弧を含めねばなりません。

**\character**{*char*}
: 1 バイトの文字列値ではなく、単一の文字について議論する場合の文字です。文字は `\samp` と同じようにタイプセットされます。

**\citetitle**[*url*]{*title*}
: 参照している出版物のタイトルです。*url* を指定すると、タイトル部分は HTML 形式に書式化した際にハイパーリンクになります。

**\class**{*name*}
: クラス名です; ドット名表記を使用できます。

**\code**{*text*}
: 短いコード断片や定数リテラル。引用符で囲わないので、スペースを入れてはなりません。

**\constant**{*name*}
: “定義済みの” 定数名です。C 言語における `#define` や、変更されないことになっている Python の変数の場合があります。

**\csimplemacro**{*name*}
: “単純” マクロの名前です。単純マクロは、引数を取らないため関数で記述できないようなコードの展開に使われます。この環境は単純な定数定義には用いません。Python ドキュメント内でのこのマクロの用例としては、`PyObject_HEAD` や `Py_BEGIN_ALLOW_THREADS` があります。

**\ctype**{*name*}
: C の `typedef` や構造体の名前です。型が `typedef` を伴わずに定義されている構造体の場合は、`\ctype{struct struct_tag}` を使って、`struct` が必須であることを明示してください。

**\deprecated**{*version*}{*what to do*}
: 記述対象がリリース *version* 以降で撤廃されていることを宣言します。*what to do* に指定したテキストでは、撤廃されたものの代わりに利用すべきものを推奨せねばなりません。テキストは完全な文にせねばなりません。文章全体に対して撤廃を注釈する場合、別個の段落にして表示せねばなりません; この場合、撤廃される機能の説明の前か後ろのどちらかに置きます。

**\dfn**{*term*}
: テキストにおける *term* のインスタンスの定義をマークします。(索引エントリを生成しません。)

**\e**
: バックスラッシュを生成します。`\code` や類似のマクロを記述する場合に便利で、そうした場合でだけ使えます。(`\file` マクロの中身のような) 通常の文章でバックスラッシュを生成する場合には標準の `\textbackslash` マクロを使ってください。

**\email**{*address*}
: 電子メールアドレスです。出力形式が何であれ、ハイパーリンクは行われ**ない** ので注意してください。アドレスのドメインネーム部分は小文字でなければなりません。

**\emph**{*text*}
強調テキストです。イタリックフォントで表示されます。

**\envvar**{*name*}
環境変数です。索引エントリを生成します。

**\exception**{*name*}
例外の名前です。ドット名表記を使えます。

**\file**{*file or dir*}
ファイルやディレクトリの名前です。PDF と PostScript 形式の出力ではファイル名を表現するのに単引用符で囲ってフォントを変更しますが、HTML 出力では引用符囲いを行いません。**警告:** 処理上の制約により、`\file` マクロはセクションタイトル内の内容には使えません。

**\filenq**{*file or dir*}
`\file` に似ていますが、単引用符を使いません。テーブルと組み合わせる際に、カラムにファイルやディレクトリ名だけが入るような場合に使えます。**警告:** 処理上の制約により、`\filenq` マクロはセクションタイトル内の内容には使えません。

**\function**{*name*}
Python 関数の名前です; ドット名表記を使えます。

**\infinity**
数学的な無限大を表すシンボル: $\infty$ です。このシンボルの HTML 表現を適切に描画できないブラウザもありますが、サポートは拡大しています。

**\kbd**{*key sequence*}
キーシーケンスをマークします。*key sequence* がどんな形式になるかはプラットフォームやアプリケーション固有の取り決めに依存します。適当な取り決めがない場合には、初心者や別のシステムを利用している人たちが分かりやすいように、モディファイアキー名を書き下さねばなりません。例えば、**xemacs** のキーシーケンスは`\kbd{C-x C-f}` のようにマークできますが、特定のアプリケーションやプラットフォームに対する参照を行わない場合には、`\kbd{Control-x Control-f}` とマークせねばなりません。

**\keyword**{*name*}
プログラミング言語における予約語 (keyword) の名前です。

**\mailheader**{*name*}
RFC 822 形式のメイルヘッダ名です。このマークアップは、そのヘッダが電子メイルメッセージ中で使われていることを示すのではなく、同じ “スタイル” のヘッダを表すのに使える名前であることを示します。様々な MIME 仕様が定義しているヘッダに対しても使います。ヘッダ名の表現方法は、実際に通常使われているのと同じ方法で入力しなくてはなりません。また、複数の用法があり、いずれも広く使われている場合には、キャメル形式 (camel-casting) を優先します。ヘッダ名の後ろに続くコロンは含めてはなりません。例: `\mailheader{Content-Type}`。

**\makevar**{*name*}
**make** における変数名です。

**\manpage**{*name*}{*section*}
UNIX マニュアルページへの参照です。

**\member**{*name*}
オブジェクトのデータ属性名です。

**\method**{*name*}
オブジェクトのメソッド名です。*name* にはメソッド名とその後ろに続く丸括弧を入れねばなりません。ドット名表記を使えます。

**\mimetype**{*name*}
MIME タイプの名前か、 MIME タイプの要素 (メジャータイプはマイナータイプのいずれか) の名前です。

**\module**{*name*}
モジュールの名前です: ドット名表記を使えます。パッケージ名にもこれを使います。

**\newsgroup**{*name*}
Usenet のニュースグループ名です。

**\note**{*text*}
ある API について、ユーザがその API を一部でも利用する場合には知っておかねばならないような特に重要な情報です。このマクロは注釈が終了する場所を視覚的にマークしないので、段落の末尾の

内容にせねばなりません。*text* の内容は完全な文で書き、適切な区切りを入れねばなりません。

**\pep**{*number*}
Python Enhancement Proposal への参照です。このマクロは索引エントリを生成します。'PEP *number*' という出力を生成します; HTML 出力の場合、テキストは指定された PEP のオンラインコピーに対するハイパーリンクになります。

**\plusminus**
ある値が、指定した量に対して正負の値をとることを示すシンボルで、通常はマイナス記号の上にプラス記号を置いて表します。例: `\plusminus 3%`

**\program**{*name*}
実行可能形式のプログラム名です。プラットフォームによっては、実行可能形式のファイル名とは異なる場合があります。特に、Windows プログラムでは '.exe' (やその他の) 拡張子を省略します。

**\programopt**{*option*}
実行可能プログラムのコマンドラインオプションです。"短い" オプションだけに使い、先頭にハイフンを入れてください。

**\longprogramopt**{*option*}
実行可能プログラムの長いコマンドラインオプションです。このマクロは二つのハイフンから始まる長いオプション名だけに使います; ハイフンを *option* に入れてはなりません。

**\refmodule**[*key*]{*name*}
`\module` に似ていますが、指定したモジュールに対するハイパーリンクを生成します。このマクロに対応する`\declaremodule` は同じドキュメント内に入っていなければなりません。`\declaremodule` がモジュール名とは異なるモジュールキーを定義している場合、`\refmodule` マクロ内でも *key* として指定せねばなりません。

**\regexp**{*string*}
正規表現をマークします。

**\rfc**{*number*}
Internet Request for Commments への参照です。適切な索引エントリを生成します。テキスト 'RFC *number*' を生成します; HTML 出力では、テキストは指定した RFC のオンラインコピーに対するハイパーリンクになります。

**\samp**{*text*}
短いコード例で、`\code` で指定するには長すぎる場合に使います。引用符記号を追加するので、テキストにスペースを入れてもかまいません。

**\shortversion**
プリアンブルの`\setshortversion` マクロに指定されているような、ドキュメントの対象となっているソフトウェアの "短い" バージョン番号です。Python の場合、あるリリースに対する短いバージョン番号とは。`sys.version` 値の最初の三文字です。例えば、バージョン 2.0b1 と 2.0.1 の短いバージョン番号はいずれも 2.0 になります。全てのパッケージにこれが当てはまるわけではありません; `\setshortversion` を使わなければ、このマクロは空文字に展開されます。`\version` マクロも参照してください。

**\strong**{*text*}
強く強調されたテキストです; ボールドフォントで表現されます。

**\ulink**{*text*}{*url*}
URL で指定したターゲットを指すハイパーテキストリンクになりますが、リンクテキストをドキュメントリソース中のタイトルにしてはなりません。名前を使ってリソースを参照する場合には、`\citetitle` マクロを使ってください。全ての形式の出力が任意のハイパーテキストリンクをサポートしているわけではありません。LaTeX 特有の多くの文字や、このマクロ特有の文字を使った場合、たいてい正しい出力になりません。特に、チルダ文字 ('~') は誤って処理されます; hex 形式の配列にエンコードすればうまくいくので、チルダの代わりには '`%7e`' を使ってください。

**\url**{*url*}
URL (または URN) です。 URL はテキストで与えます。HTML や PDF の出力形式では、URL 文字列自体もハイパーリンクになります。このマクロは、特定のタイトルを持たない外部リソースを参照する際に利用できます; タイトルのあるリソースへの参照は `\citetitle` マクロを使ってマークしてください。特殊文字に関する特別な扱いについては、`\ulink` マクロの記述中のコメントを参照してください。

**\var**{*name*}
テキスト中で変数や仮引数の名前です。

**version**
: プリアンブルの\release マクロに指定されているような、ドキュメントの対象となっているソフトウェアのバージョン番号です。\shortversion マクロも参照してください。

**versionadded**[*explanation*]{*version*}
: 記述対象の機能が C API のライブラリに追加された時点の Python のバージョン番号です。*explanation* は変更内容の**短い** 説明で、大文字から始まる文章断片からなります; 末尾のピリオドは書式化の処理時に追加されます。このマクロは普通、機能を説明する文の第一パラグラフの末尾で、かつ利用可能条件に関する注釈よりも前に追加します。マクロを追加する場所は説明文の意味が通るように選ばねばならず、必要に応じて場所を変えてもかまいません。

**versionchanged**[*explanation*]{*version*}
: 記述対象の機能が変化 (新たなパラメタ、副作用の変化、など) した時点の Python のバージョンです。*explanation* は変更内容の**短い** 説明で、大文字から始まる文章断片からなります; 末尾のピリオドは書式化の処理時に追加されます。このマクロは普通、機能を説明する文の第一パラグラフの末尾で、かつ利用可能条件に関する注釈と\versionadded マクロよりも前に追加します。マクロを追加する場所は説明文の意味が通るように選ばねばならず、必要に応じて場所を変えてもかまいません。

**warning**{*text*}
: ある API について、ユーザがその API を一部でも利用する場合には警告しておかねばならないような重要な情報です。このマクロは注釈が終了する場所を視覚的にマークしないので、段落の末尾の内容にせねばなりません。*text* の内容は完全な文で書き、適切な区切りを入れねばなりません。\note との違いは、セキュリティに関する情報の場合いは、\warning の方が推奨されているという点です。

## 6.6 雑多なテキストマークアップ

インラインマークアップに加えて、“ブロック” マークアップがいくつか定義されていて、様々なテキスト断片に対して読者の注意を引かせやすくしています。この節で述べているマークアップは、こうした目的の他に、(verbatim 環境のように) 一つ以上の段落や、他のブロック構造をマークする際に使うためのものです。

\begin{**notice**}[*type*]
\end{**notice**}
: 読者がさらに注意を払うべき段落をラベルします。どの種の注意を求めているかは、*type* に指定します。*type* 用に定義されている値は note と warning です; これらは、同名のインラインマークアップと同じ目的に使います。*types* を省略すると、note を使います。*type* が撮りうる値は将来新たに追加されるかもしれません。

## 6.7 モジュール特有のマークアップ

この節で説明しているマークアップは、記述対象のモジュールに関する情報を提供するために使います。このマークアップは、モジュールを説明している節の先頭でよく使います。典型的な例は、以下のようになります:

```
\section{\module{spam} ---
         SPAM 機能へのアクセス}

\declaremodule{extension}{spam}
  \platform{Unix}
\modulesynopsis{\UNIX の SPAM 機能へのアクセス手段。}
\moduleauthor{Jane Doe}{jane.doe@frobnitz.org}
```

Python のパッケージ — ひとまとまりの単位として記述できるようなモジュールの集合 — は、モジュールと同じマークアップを使って解説します。パッケージ内のモジュール名は “完全限定 (fully qualified)” 形式で入力せねばなりません (つまり、パッケージ名を含めねばなりません)。例えば、 “bar” というパッケージ内の “foo” というモジュールは\module{bar.foo} と書き、リファレンスの先頭は以下のようになります:

```
\section{\module{bar.foo} ---
         \module{bar} パッケージ内のモジュール}

\declaremodule{extension}{bar.foo}
\modulesynopsis{\module{bar} パッケージ内の小さなモジュール}
\moduleauthor{Jane Doe}{jane.doe@frobnitz.org}
```

パッケージの名前もまた、`\module` でマークします。

**\declaremodule**[*key*]{*type*}{*name*}
二つのパラメタ: モジュールの種類 (‘standard’、‘builtin’、‘extension’、または ‘’) と、モジュールの名前を必要とします。この節にリンクを張ったり参照を行ったりするための “キー” によっては、オプションのパラメタを指定せねばなりません。“キー” は、モジュールの名前にアンダースコアが入っている場合にのみ用い、モジュール名からアンダースコアを取り除いた名前でなければなりません。*type* パラメタは上に述べた値のいずれかでなければならず、そうでない値を指定するとエラーが出力されるので注意してください。パッケージに入っているようなモジュールの場合、*name* パラメタには完全限定形式の名前を指定せねばなりません。このマクロはモジュールを紹介する`\section` の先頭に置かねばなりません。

**\platform**{*specifier*}
モジュールの可搬性について述べるために使います。*specifier* はモジュールを利用できるようなプラットフォームを指定するキーをコンマで区切ったリストです。キーは短い識別子にします; 実際に使われている例には ‘IRIX’、‘Mac’、‘Windows’、‘Unix’ があります。あるプラットフォームを表す場合、すでに使われているキーがあるなら、それを使うようにするのが重要です。この情報は、モジュール索引や、HTML 形式、GNU info 形式の出力における注釈情報に使われます。

**\modulesynopsis**{*text*}
*text* はモジュールの短い “一行の” 説明で、各章の紹介の一部として使われることがあります。このマクロは `\declaremodule` の後ろに配置せねばなりません。モジュールの概要は、`\localmoduletable` マクロによって挿入される一覧表の内容を構築する際に使われます。マークアップを行った場所では何のテキストも生成されません。

**\moduleauthor**{*name*}{*email*}
このマクロはモジュールの作者についての情報を符号化するために使われます。現在は出力生成に使われていませんが、モジュールの起源を判断する際に役立つはずです。

## 6.8 ライブラリレベルのマークアップ

このマークアップは、一そろいのモジュールについて説明する際に使います。例えば、 *Macintosh* **ライブラリモジュール**ドキュメントでは、モジュールの集まりを概観できるようにするためにこのマクロを役立てています。また、*Python* **ライブラリリファレンス** も同様の目的にこのマクロを利用しています。

**\localmoduletable**
現在の章 (howto ドキュメントの場合はドキュメント全体) に対して ‘.syn’ ファイルが存在する場合には、‘.syn’ ファイルから読み出した内容で `synopsistable` を作成します。

## 6.9 表のマークアップ

表環境には汎用の三つの形式があり、できる限りこの環境を使うようにしてください。こうした環境は標準の LaTeX 表環境を置き換えるためのものではなく、Python ドキュメントを作成するための処理ツールで処理する際に便利なように作られています。とりわけ、この表環境で生成した HTML は見栄えがよくなります! また、いつかドキュメントを XML 形式に変換する時にも (8 節、 “将来の方向性” を参照してください)、利点があるでしょう。

各環境は `table`*cols* という形式の名前がついています。*cols* には、表のカラム数を小文字のローマ数字で指定します。各々の環境について、追加のマクロ`\line`*cols* が定義されています。この場合 *cols* は表環境の *cols* に対応します。サポートされている *cols* には `ii`、`iii`、`iv`、`v` があります。どの環境も全て `tabular` 環境の上に構築されています。`longtable` 環境に基づく変化形も提供されています。

標準 Python ドキュメント内の全ての表がカラム間に垂直線を使っており、各表のマークアップでこの垂直線を指定しなければならないので注意してください。表の外側の境界線はマークアップに使いませんが、

これは処理ツール側の役割です; ドキュメントのマークアップ時には外側の境界線を含めてはなりません。

`longtable` に基づいた表環境の変化形では、前後に余分にスペースを入れて書式化を行うため、表が長すぎて、複数のページに分割したほうが合理的な場合にのみ使ってください; 20 行よりも行数の少ない表の場合、決して長形式の表環境を使ってマークをしないようにしてください。ヘッダ行は、表の各ページにおける部分の先頭に繰り返し表示されます。

`\begin{`**`tableii`**`}{`*colspec*`}{`*col1font*`}{`*heading1*`}{`*heading2*`}`
`\end{`**`tableii`**`}`
: LaTeX のカラム情報指定子 (column specifier) *colspec* を使って、2 カラムの表を作成します。カラム情報指定子には表によって適切なカラム分割垂直線が入るように指定を行わねばなりませんが、表の外側の垂直線は指定してはなりません (この部分の指定はスタイルシートの問題と考えられているからです)。*con1font* パラメタは、テーブルの最初のカラムの表示スタイルをどうするかを指定します: その結果として、最初のカラムは`\`*col1font*`{column1}` のようになります。最初のカラムを特別扱いしないために、*col1font* を '`textrm`' にしてもかまいません。カラムのヘッダは *heading1* および *heading2* からとります。

`\begin{`**`longtableii`**`}...`
`\end{`**`longtableii`**`}`
: `tableii` に似ていますが、ページ境界を割って配置できるような表を生成します。パラメタは `tableii` と同じです。

`\`**`lineii`**`{`*column1*`}{`*column2*`}`
: `tableii` や `longtableii` 環境内の単一の行を生成します。第一カラムのテキストは `tableii` 環境を始めたときに指定した *col1font* を適用して生成します。

`\begin{`**`tableiii`**`}{`*colspec*`}{`*col1font*`}{`*heading1*`}{`*heading2*`}{`*heading3*`}`
`\end{`**`tableiii`**`}`
: `tableii` 環境に似ていますが、第三カラムがあります。第三カラムのヘッダは *heading3* に指定します。

`\begin{`**`longtableiii`**`}...`
`\end{`**`longtableiii`**`}`
: `tableiii` に似ていますが、ページ境界を割って配置できるような表を生成します。パラメタは `tableiii` と同じです。

`\`**`lineiii`**`{`*column1*`}{`*column2*`}{`*column3*`}`
: `\lineii` マクロに似ていますが、第三カラムがあります。第三カラムのテキストは *column3* に指定します。

`\begin{`**`tableiv`**`}{`*colspec*`}{`*col1font*`}{`*heading1*`}{`*heading2*`}{`*heading3*`}{`*heading4*`}`
`\end{`**`tableiv`**`}`
: `tableiii` 環境に似ていますが、第四カラムがあります。第四カラムのヘッダは *heading4* に指定します。

`\begin{`**`longtableiv`**`}...`
`\end{`**`longtableiv`**`}`
: `tableiv` に似ていますが、ページ境界を割って配置できるような表を生成します。パラメタは `tableiv` と同じです。

`\`**`lineiv`**`{`*column1*`}{`*column2*`}{`*column3*`}{`*column4*`}`
: `\lineiv` マクロに似ていますが、第四カラムがあります。第四カラムのテキストは *column4* に指定します。

`\begin{`**`tablev`**`}{`*colspec*`}{`*col1font*`}{`*heading1*`}{`*heading2*`}{`*heading3*`}{`*heading4*`}{`*heading5*`}`
`\end{`**`tablev`**`}`
: `tableiii` 環境に似ていますが、第五カラムがあります。第五カラムのヘッダは *heading5* に指定します。

`\begin{`**`longtablev`**`}...`
`\end{`**`longtablev`**`}`
: `tablev` に似ていますが、ページ境界を割って配置できるような表を生成します。パラメタは `tablev` と同じです。

`\`**`linev`**`{`*column1*`}{`*column2*`}{`*column3*`}{`*column4*`}{`*column5*`}`
: `\lineiv` マクロに似ていますが、第五カラムがあります。第五カラムのテキストは *column5* に指定します。

表に似たもう一つ環境として `synopsistable` があります。この環境が生成する表は二つのカラムから

なり、各行は別のマクロ定義\modulesynopsis 中に定義されています。通常ドキュメントの作者がこの環境を使うことはありませんが、\localmoduletable マクロを使うとこの表を作成します。

ここで warning モジュールのドキュメントにある小さな表の例を示します; 表のマークアップ自体がかなり識別しやすく作られているので、表セル内のマークアップは最小限にしています。以下が表のマークアップです。

```
\begin{tableii}{l|l}{exception}{クラス}{説明}
\lineii{Warning}{全ての警告カテゴリの基底クラスです。
\exception{Exception} のサブクラスです。}
\lineii{UserWarning}{\function{warn()} の標準のカテゴリです。}
\lineii{DeprecationWarning}{機能の撤廃を警告するカテゴリの基底クラスです。}
\lineii{SyntaxWarning}{文法機能のあいまいさを警告するカテゴリの基底クラスです。}
\lineii{RuntimeWarning}{実行時システム機能のあいまいさを警告するカテゴリの基底クラスです。}
\lineii{FutureWarning}{変更予定のセマンティクスを警告するカテゴリの基底クラスです。}
\end{tableii}
```

これは以下のような表になります。

| クラス | 説明 |
|---|---|
| Warning | 全ての警告カテゴリの基底クラスです。Exception のサブクラスです。 |
| UserWarning | warn() の標準のカテゴリです。 |
| DeprecationWarning | 機能の撤廃を警告するカテゴリの基底クラスです。 |
| SyntaxWarning | 文法機能のあいまいさを警告するカテゴリの基底クラスです。 |
| RuntimeWarning | 実行時システム機能のあいまいさを警告するカテゴリの基底クラスです。 |
| FutureWarning | 変更予定のセマンティクスを警告するカテゴリの基底クラスです。 |

クラス名は\exception マクロを使って非明示的にマークされますが、これは\exception が tableii 環境の *col1font* の値として指定されているからです。最初のカラムを別のマークアップにした表を作成するには、textrm を *col1font* の値に指定して、各エントリを個別にマークしてください。

表を垂直方向に分割するために水平線を追加するには、標準のマクロ\hline を分割したい行の間に入れます:

```
\begin{tableii}{l|l}{constant}{言語}{対象読者}
  \lineii{APL}{マゾヒスト}
  \lineii{BASIC}{PC ハードウェア使いのプログラム初心者}
  \lineii{C}{\UNIX{} \&\ Linux カーネル開発者}
    \hline
  \lineii{Python}{誰でも!}
\end{tableii}
```

全ての出力形式でこの場所に水平線を引けるとは限らないので注意してください。今読者が読んでいるドキュメントの出力形式では、上の表は以下のようになります:

| Language | Audience |
|---|---|
| APL | Macintosh 野郎 |
| BASIC | PC ハードウェア使いのプログラム初心者 |
| C | UNIX & Linux カーネル開発者 |
| Python | 誰でも! |

## 6.10 参考文献リストのマークアップ

ドキュメントの各節の多くには、モジュールドキュメントや外部ドキュメントを指す参考文献のリストが入っています。こうしたリストは、seealso や seealso* 環境を使って作成します。これらの環境では、参考文献エントリを合理的なやり方で作成するための追加のマクロを定義しています。

seealso 環境は通常、節中で何らかの小節に入る直前の場所に配置します。これは、ある節に関係した参考文献リンクが、ドキュメントのハイパーテキスト解析を行う際に小節内に入り込んでしまわないようにするためです。HTML 出力の場合、このマクロの内容は本文の外側の “サイドバー” に表示されます。

seealso* 環境は、参考文献リストを主コンテンツの一部として表示したい場合に使う点が異なります; seealso* を使うと、リストを特別扱いしてテキスト内から押し出したりしません。

\begin{**seealso**}
\end{**seealso**}
この環境はヘッダ “See also:” を作成し、個々の参考文献を記述するためのマークアップを定義します。

\begin{**seealso***}
\end{**seealso***}
この環境は主コンテンツの一部として参考文献リストを生成する際に使います。リストに特別にヘッダをつけることはなく、ドキュメントの本文から押し出されることもありません。個々の参考文献を記述するために、上と同じマークアップを提供しています。

以下のマクロの各々において、*why* は一つかそれ以上の完全な文章で、大文字から始めねばなりません (文が識別子から始まる場合には別で、大文字にする必要はありません)。また、適切な区切り文字で終えねばなりません。

これらのマクロは seealso や seealso* 環境の中でしか定義できません。

**\seelink**{*url*}{*linktext*}{*why*}
特定のオンラインリソースへの参照を示す場合、ちゃんとした題名をそのリソースにつけられないが、短い説明をリンクの後につけたい場合には \seelink マクロを使います。タイトルで特定できるようなオンライン上のドキュメントには\seetitle マクロを使い，オプションに URL を指定するようにしてください.

**\seemodule**[*key*]{*name*}{*why*}
別のモジュールへの参照です。*why* は、なぜその参考文献が興味の対象となるのかを示す短い説明にせねばなりません。モジュール名は *name* で指定し、必要があれば *key* でリンクキーを指定します。HTML および PDF 版では、モジュール名部分は参照先のモジュールを指すハイパーリンクになります。**注意:** 該当のモジュールは同じドキュメント内で記述されていなければなりません (対応する\declaremodule が必要です)。

**\seepep**{*number*}{*title*}{*why*}
Python Enhancement Proposal (PEP) への参照です。*number* は PEP 編集者が割り当てた公式 PEP 番号でなければならず、*tiutle* は PEP ドキュメントの公式版に書かれている人間可読なタイトルでなければなりません。このマクロは、ドキュメント中の注釈がつけられている節の内容が関係しているインタフェースや言語機能 PEP への参照を読者に示すために使います。

**\seerfc**{*number*}{*title*}{*why*}
IETF による Request for Comment (RFC) を参照します。それ以外の点では、 \seepep とほとんど同じです。このマクロは、ドキュメント中の注釈がつけられている節の内容が関係しているプロトコルやデータ形式を仕様記述している RFC への参照を読者に示すために使います。

**\seetext**{*text*}
任意のテキスト *text* を “See also:” リストに追加します。このマクロはオフラインやオンラインの資料を \url マクロで参照するために使えます。テキストは一つかそれ以上の完全な文で構成しなければなりません。

**\seetitle**[*url*]{*title*}{*why*}
*title* という名前の外部ドキュメントに対する参照を追加します。*url* を指定した場合、HTML 版では外部ドキュメントのタイトル部分がハイパーリンクになり、組版版のドキュメントでは表題の下に URL が表示されます。

**\seeurl**{*url*}{*why*}
特定のオンラインリソースで、意味のあるタイトルがない場合には\seeurl マクロを使って参照を行います。はっきりしたタイトルのあるオンラインドキュメントの場合には、\seetitle マクロを使い、そのオプションパラメタに URL を指定して参照を行ってください。

## 6.11 索引生成のためのマークアップ

技術文書における索引の効果的な作成は、とりわけ作者がある事柄には詳しくても索引の生成には不慣れな場合には非常に困難です。ドキュメントの困難さのほとんどは技術用語の範囲からきています: ある概念を説明する際に、熟練者が使うような用語を入れるだけでは不十分です。ドキュメントの作者は普通、自分の書いているドキュメントの領域においては熟練者であり、初心者が立ち止まってしまうような用語を見つけ出すのはかなり難しいのです。

索引生成の本当に困難な側面は、ドキュメント作成ツールで補助できるような領域のことがらではあり

ません。とはいえ、ひとたび用語の内容を決めたら、索引の作成を簡単にするのはツールの役割です。様々な種類の索引を最小限の努力で生成できるようにするためにドキュメント生成ソフトウェアが利用できるようなマークアップが提供されています。加えて、6.3 節、“情報単位” で説明した多くの環境は、総合索引やモジュール索引上に適切なエントリを作成します。

以下のマクロは、索引情報を生成するために利用できます。これらはドキュメントのプリアンブル内で使わねばなりません。

**\makemodindex**
ドキュメントで多くのモジュールに対する参照が入った “モジュール索引” が必要な場合、プリアンブル内で使います。このマクロは一連の`\declaremodule` マクロからデータファイル `lib`*jobname*`.idx` を生成します。このファイルは **makeindex** プログラムで使われ、モジュール索引を入れたい場所に `\input` を使って取り込めるようなファイルを生成します。

様々な概念について索引エントリを追加する上で便利なマクロがあります。これらの多くはプログラミング言語や、Python に特有の概念です。

**\bifuncindex**{*name*}
*name* という名前の組み込み関数に対する索引エントリを追加します; *name* の後ろに丸括弧を入れてはなりません。

**\exindex**{*exception*}
*exception* という名前の例外に対する索引エントリを追加します。例外はクラスベースの例外でなければなりません。

**\kwindex**{*keyword*}
言語における予約語 (keyword、関数やメソッド呼び出しにおけるキーワード引数とは違います) に対する索引エントリを追加します。

**\obindex**{*object type*}
組み込みオブジェクト型に対する索引エントリを追加します。

**\opindex**{*operator*}
‘+’ のような演算子に対する索引エントリを追加します。

**\refmodindex**[*key*]{*module*}
モジュール *module* に対する索引エントリを追加します; *module* にアンダースコアが入っている場合、オプションのパラメタ *key* には、*module* からアンダースコアを除去した名前を指定せねばなりません。索引エントリ “*module* (module)” を生成します。Python で書かれた非標準モジュールで使うためのものです。

**\refexmodindex**[*key*]{*module*}
`\refmodindex` のようなものですが、索引エントリは “*module* (extension module)” になります。このマクロは Python 以外で書かれた非標準モジュールで使うためのものです。

**\refbimodindex**[*key*]{*module*}
`\refmodindex` のようなものですが、索引エントリは “*module* (built-in module)” になります。このマクロは Python 以外で書かれた標準モジュールで使うためのものです。

**\refstmodindex**[*key*]{*module*}
`\refmodindex` のようなものですが、索引エントリは “*module* (standard module)” になります。このマクロは Python で書かれた標準モジュールで使うためのものです。

**\stindex**{*statement*}
`print` や `try`/`finally` のような、構文型に対する索引エントリを追加します。

総合索引を簡単に作成する上で便利なマクロが提供されています。索引エントリは、索引の様々な場所で語順を入れ替えて表示しなければなりません。これらのマクロは単に `\index` を使って複数の索引エントリを作成するような単純なマクロです。こうしたマクロを使って構築された索引エントリには、一次および二次の索引テキストが入ります。

**\indexii**{*word1*}{*word2*}
二つの索引語からなる索引エントリを構築します。`\index{`*word1*`!`*word2*`}` と `\index{`*word2*`!`*word1*`}` を組み合わせて使うのと同じです。

**\indexiii**{*word1*}{*word2*}{*word3*}
三つの索引語からなる索引エントリを構築します。`\index{`*word1*`!`*word2 word3*`}`、`\index{`*word2*`!`*word3, word1*`}`、および`\index{`*word3*`!`*word1 word2*`}` を組み合わせて使うのと同じです。

**\indexiv**{*word1*}{*word2*}{*word3*}{*word4*}

四つの索引語からなる索引エントリを構築します。\index{*word1*!*word2 word3 word4*}、\index{*word2*!*word3 word4, word1*}、\index{*word3*!*word4, word1 word2*}、および \index{*word4*!*word1 word2 word3*} を組み合わせて使うのと同じです。

## 6.12 文法における導出の表示

形式文法における導出 (producution) を表示するために、特殊なマークアップを使えます。マークアップは単純なもので、 BNF (やその派生形) の全ての側面をモデル化しようとはしていませんが、シンボルを記述するとそのシンボルの定義へのハイパーリンクになるような表示を文脈自由文法に対して実現しています。環境が一つと、二つのマクロがあります:

\begin{**productionlist**}[*language*]
\end{**productionlist**}
この環境は、ひとまとまりの導出規則を囲むために使います。この環境ではマクロを二つだけ定義しています。ドキュメントで二つ以上の言語について説明する場合、オプションのパラメタ *language* を使って、言語間で導出規則を区別せねばなりません。パラメタの値はファイル名の一部に使えるような短い名前にせねばなりません; コロンや、複数のプラットフォーム間で同時にファイル名に使えないような文字は含めてはなりません。

**production**{*name*}{*definition*}
文法における導出規則です。導出規則は、*name* を *definition* であると定義します。*name* にはマークアップを入れてはならず、ハイフンを使った一つ以上の文法のサポートは定義されていません。一つの \production には単一のシンボルだけを定義できます — 複数シンボルの定義を行ってはなりません。

**token**{*name*}
\production マクロが定義しているシンボルの名前で、シンボルの *definition* の中で使います。シンボル *name* は可能な場合にはシンボル定義へのハイパーリンクになります。

必ずしも文法全体を単一の productionlist 環境に定義する必要はないので注意してください; 任意の数のグループを作って文法を記述してかまいません。\token を使う場合には常に\production と対応付けせねばなりません。

以下は *Python* **リファレンスマニュアル**からとった例です:

```
\begin{productionlist}
  \production{identifier}
             {(\token{letter}|"_") (\token{letter} | \token{digit} | "_")*}
  \production{letter}
             {\token{lowercase} | \token{uppercase}}
  \production{lowercase}
             {"a"..."z"}
  \production{uppercase}
             {"A"..."Z"}
  \production{digit}
             {"0"..."9"}
\end{productionlist}
```

## 6.13 グラフィカルインタフェースの構成要素

グラフィカルインタフェースの構成要素についてマークアップを割り当てることになっていますが、その詳細はまだほとんど決まっていません。

**menuselection**{*menupath*}
メニューの選択肢は \menuselection と \sub を組み合わせてマークします。このマクロは、サブメニューを選択して特定の操作を選択するといった完全な形のメニュー選択手順や、任意の部分的な手順をマークするために使います。個々の選択肢の名前は \sub を入れて分割せねばなりません。

例えば、“スタート ＞ プログラム” をマークするには、以下のコードを使います:

```
\menuselection{スタート \sub プログラム}
```

選択肢がダイアログを開くようなコマンドであることを示すためにオペレーティングシステムが使う省略記号のように、選択肢の末尾に末尾に識別記号が入っている場合、選択肢名からは識別記号を省いてください。

**\sub**
複数の階層からなるメニュー選択肢を分割するための記号です。このマクロは `\menuselection` マクロのコンテキスト下でしか定義されていません。

# 7 処理ツール

## 7.1 外部ツール

Python ドキュメントがサポートしている全ての出力形式を処理できるようにするには、数多くのツールが必要です。この節では、処理に使われている各ツールについて列挙し、各々がいつ必要になるのか説明しています。これらのツールのいずれかが特定のバージョンを要求しているかどうかは、'Doc/README' ファイルを調べてみてください。

**dvips** このプログラムはよく TEX インストールの一部になっています。**dvips** は "デバイス非依存 (dvice independent)" の '.dvi' ファイルから PostScript を生成するために使われます。PostScript への変換を行う際に必要です。

**emacs** Emacs はごちゃまぜ (kitchen sink) の、さらに言うとクソ素晴らしいごちゃまぜのプログラマ向けエディタです。info 形式に変換したい場合に、Texinfo ドキュメントの適切なメニュー階層の構築をサポートする上で必要な処理に関係しています。FSF の **emacs** の代わりに **xemacs** を使うと、変換がうまくいかないことがありますが、これは誰も Emacs の Texinfo コードをメンテナンスして可搬性のあるものにしないからだと思われます。

**latex** LATEX は、Laslie Lamport による拡張可能な大規模マクロパッケージで、Donald Knuth が作成した世界規模で使われているタイプセッタ TEX に基づいています。PostScript 形式への変換に必要で、 HTML への変換にも必要です (LATEX2HTML は LATEX が生成する中間ファイルの一つを必要とするためです)。

**latex2html** これまでにメンテナンスされたことのある Perl スクリプトの中でおそらく最長のスクリプトでしょう。このツールは LATEX ドキュメントを HTML ドキュメントに変換し、そこそこ立派な仕事をします。HTML や GNU info 形式への変換に必要です。

**lynx** HTML から平文テキストへの変換機能を持つテキストモードの Web ブラウザです。`howto` ドキュメントをテキスト形式に変換する際に使います。

**make** どのバージョンの **make** も標準ドキュメントの処理に使えるはずですが、 実験段階にある 'Doc/tools/sgmlconv/' の処理には、少なくとも実験段階にある間は GNU **make** が必要です。**mkhowto** スクリプトを走らせる場合には必要ありません。

**makeindex** LATEX の索引データを書式化済みの索引に変換するための標準プログラムです; 全ての LATEX インストール物に入っているはずです。PDF 形式と PostScript 形式への変換に必要です。

**makeinfo** Texinfo から GNU info ファイルへの変換には GNU **makeinfo** を使います。Texinfo は info への変換における中間形式の処理に使われるので、info への変換にもこのプログラムが必要です。

**pdflatex** pdfTEX は比較的新しく登場した TEX の変種で、マニュアルの PDF 版を生成するために使います。通常、ほとんどの大規模な TEX 配布物の一部としてインストールされています。**pdflatex** は LATEX 形式を扱う pdfTEX です。

**perl** LATEX2HTML や LATEX2HTML 出力に対する仕上げ作業、HTML から Texinfo への変換には Perl が必要です。 HTML 形式や GNU info 形式への変換に必要です。

**python** Python は 'Doc/tools/' ディレクトリ内の多くのスクリプトで使われています; 全ての形式への変換で必要です。Python のドキュメントを書こうと思っているのなら、何も問題はないはずですよね!

## 7.2 内部用ツール

この節では、ドキュメント処理の様々な段階を実装したり、ビルド手順全体を演出する様々なスクリプトについて説明します。これらのツールのほとんどは標準ドキュメントを構築するというコンテキストでのみ有用ですが、汎用的なものもあります。

**mkhowto** サードパーティ製のドキュメントを書式化する際に最初に使うスクリプトです。このスクリプトには、“正しく作業を行う” 上で必要なロジックが全て入っています。このスクリプトを正しく使うには、スクリプトに対するシンボリックリンクを張って、その場所で実行します; 実際のスクリプトファイルはドキュメントソースツリーの一部として収められていなければなりませんが、ツリーの外部にあるドキュメントの書式化にも利用できます。コマンドラインオプションのリストを見るには、**mkhowto --help** を使ってください。

**mkhowto** は `howto` と `manual` クラスのどちらのドキュメントにも使えます。このツールは、古い Python のソースリリースに入っているバージョンではなく、常に最新のバージョンを使うようにするのがよいでしょう。

## 7.3 Cygwin での作業

多くのパッケージは Cygwin 環境下でのインストールが難しいために、Cygwin 用に必要なツールの Cygwin 環境下でのインストールは少し厄介です。

Cygwin インストーラを使って、 Cygwin のインストールによって Perl、Python、そして TEX パッケージが入っていることを確認してください。Perl と Python はインストーラの Interpreters セクションの下に配置されています。TEX パッケージは Text セクションの下です; `tetex-beta`、`texmkf`、`texmf-base`、および `texmf-extra` をインストールすると、必要な全てのパッケージを利用できるようになります。 (より小規模な構成もありますが、筆者はインストール構成の縮小を試みるために時間を費やしたことがありません。)

LATEX2HTML は **netpbm** パッケージを使うので、たとえ Python ドキュメントのほとんどが **netpbm** を使わないとしても、LATEX2HTML をインストールする前にインストール**しておかねばなりません**。ダウンロードできる場所に関する参考文献は、netpbm README にあります。説明に従ってインストールしてください。

LATEX2HTML はソースアーカイブからのインストールもできますが、配布物中のあるファイルをいじってからでないとできません。展開した配布物のトップレベルにある ‘L2hos.pm’ を編集してください; ファイルの末尾近くにある、`$^O` というテキストを`'unix'` に変更してください。以下のコマンドを使ってソフトウェアをビルドしてインストールします:

```
% ./configure && make install
```

これで少なくとも HTML、PDF、および PostScript 形式版の書式化済みドキュメントをビルドできます。

# 8 将来の方向性

Python ドキュメントは変更に次ぐ変更の歴史を重ねてきており、そのほとんどはやや小規模の漸進的なものでした。マークアップ言語やドキュメントの処理に使われるツールに関して非常に多くの話し合いが行われてきました。この節では、変更のさまと、将来の開発において最もありえそうな道筋について議論します。

## 8.1 構造化ドキュメント

LATEX マークアップに対する小規模な変更のほとんどは、マークアップから表現方法を切り離すためとメンテナンス性の向上の両方を狙ってなされてきたものです。 1998 年からこのかた、まさにこの目的のもとに数多くの変更が行われました; それ以前は、変更はありましたがあまり体系的なやりかたではなく、既存の内容を更新する必要はないと考えられていました。その結果、高度に構造化され、意味論的には LATEX の上に読み出されることになるような、LATEX で実装されたマークアップ言語になってしまいました。とはいえ、基本的な TEX や LATEX によるマークアップが使われていなければ、実際のドキュメントソースで LATEX が使われている証拠となるのはマークアップの構文法くらいです。

この副作用のせいで、例えば LaTeX や LaTeX2HTML のような、ドキュメントを操作するための標準の “エンジン” を使えるにも関わらず、実際の変換のほとんどは Python に特化して作られてしまいました。LaTeX ドキュメントクラスと LaTeX2HTML サポートは、いずれもこのドキュメントのために設計された特殊なマークアップを完全に実装しています。

高度にカスタマイズされたマークアップが、ドキュメントを処理するための深遠なる機構と組み合さってしまうと、こんな疑問が出てきます: もっと簡単にできないの? あるいは、もっとましにならないの? コミュニティとのたくさんの議論を重ねた結果、我々は近代的な構造化ドキュメント生成システムの追求にも時間を費やす価値があるという結論に達しました。

この議論の場には、二つの現実的な競合案: 標準化汎用マークアップ言語 (Standard General Markup Language, SGML) と拡張可能なマークアップ言語 (Extensible Markup Language, XML) が挙がっています。これらの標準には、いずれも長所と短所があり、多くの長所を共有しています。

SGML はほとんどのドキュメント作者、とりわけ普通のテキストエディタを利用している人たちをそそらせるような長所を持っています。また、コンテンツのモデルを定義できるという別の機能もあります。完成度が実証されている多くの高品質なツールを利用できますが、そのほとんどはフリーではありません; またこれらのツールでは、可搬性がいまだに問題になっています。

XML には、多数の発展中のツールを使えるという利点があります。残念ながら、XML の標準自体の多くがまだ発展中なので、ツールも当分はそれに追従しなければならないでしょう。つまり、基本 XML 1.0 勧告を超えた仕様を利用するような安定したツールセットを短い期間で開発するのは不可能です。最も重要な関連標準仕様をのいくつかをサポートするような広範な種類の高品質ツールが利用できると自信を持っていえるのははまだ先の話です。多くのツールはフリーのようであり、ツール間の可搬性はまだ明らかではないようです。

XML から SGML システムへの変換が、トランスレータにとって有望であることも判明しました。トランスレータの負荷を軽減するためにどれだけのことを行わねばならないかはそのうち判明し、スキーマや使われている特定の技術に対してインパクトを与えることになるかもしれません。

将来はドキュメントを XML に移行することになっていて、ドキュメントを現在の形式から変換して、最終的なバージョンに近いものにするツールを開発しているところです。開発は必要な情報が変換後のドキュメントに入るという程度まで進んでいます。基礎的な XML 形式を HTML 形式に表現するような XSLT スタイルシートの開発も始まりましたが、成果はまだ荒削りの状態です。

作業にかける十分な時間がないため、変換にどれくらいかかるかははっきりしていませんが、変換によってもたらされうる利益はかなり早いペースで現実的なものになりつつあります。

## 8.2 議論の場

Python ドキュメント作成の将来に関する議論とドキュメントに関する議論は、Documentation Special Interest Group 、“Doc-SIG” で行われています。Doc-SIG にはメーリングリストのアーカイブや購読情報があり、http://www.python.org/sigs/doc-sig/ で利用できます。SIG はドキュメンテーションに関心を持つ全ての人々に解放されています。

標準ドキュメントに対するコメントやバグ報告は docs@python.org に送ってください。これには書式化、内容、文法的間違いやつづり間違い、そしてこのドキュメント自体に対するコメントも含みます。このドキュメントに対するコメントは作者である fdrake@acm.org に直接送っていただいてもかまいません。

**注意:** 和訳に対するコメントやバグ報告は、SourceForge.jp のプロジェクトページにお寄せください。

# Index

## N



## O



## P



## R



## S



## T



## U



## V



## W

# A　日本語訳について

## A.1　このドキュメントについて

この文書は、Python ドキュメント翻訳プロジェクトによる 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、プロジェクトのバグ管理ページ

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

までご報告ください。

## A.2　翻訳者一覧

2.3.3 和訳 sakito, Yasushi Masuda (May 21, 2004)
2.3.4 差分 Yasushi Masuda (September 20, 2004)