Chapter 01

インストールと環境設定

本章では，まず，Python を入手してお使いのコンピュータに正しくインストールする方法を説明します。つぎに，Python を快適に使うための環境設定と日本語を扱うための設定について紹介します。Python は日本語を問題なく扱うことができますが，はじめにいくつかの設定が必要になるので，次の章に進む前に本章でしっかり設定を行っておいてください。

Chapter 10

オブジェクト

本章では，Python で最も重要とも言えるオブジェクトについて説明します。オブジェクトについてはこれまでも何度か触れましたが，ここでもう一度整理しておきましょう。13 章では独自のデータ型を定義する方法を説明しますが, 本章の内容とも関連が深いので，オブジェクトについてここでしっかり押さえておいてください。

Chapter 14

テスト

本章で扱うテストとは，プログラムを細かな単位に分割し，その入出力を検証することにより，プログラムの欠落を発見するための手法です。テストを書いておくことで，後でプログラムの一部を変更した時に元の機能が正しく動作するかどうかを確かめることができます。

はじめに

本書は，2006年12月15日に株式会社ディー・アートより出版された『実践Python』をベースに，相当量の加筆修正を加えて技術評論社のポケットリファレンスにしたものです。『実践Python』は残念ながら版元の営業停止により本の供給が止まってしまいましたが，このたび装いと内容を新たにして技術評論社から出版できることは喜びに堪えません。前著と同様，本書もPython言語を学ぶ書籍として実践的な内容をわかりやすく解説することを目指しています。

Pythonは今やメジャーな言語としての地位を世界的な規模で確立しました。Pythonが使われる分野はあらゆるところに及び，Webやインターネットサービスの開発言語として，企業の社内システムの効率的な開発ツールとして，科学分野での装置制御およびデータ処理ツールとして，ゲームやグラフィックソフトウェアの機能拡張言語として，そして学習効率のよいコンピュータサイエンスの教育言語として活躍の場を広げています。

著者である私は，ゾープ・ジャパン株式会社でZope 3の開発を行っていた時も含めて，Pythonについて8年間のキャリアがあります。仕事においても，趣味においても，その軽快さとコーディングの気持ちよさ，そしてコーディングにとらわれることなく目の前の問題に集中できる感覚に未だワクワクします。そして，実際に仕事で使うとその切れ味に感心します。そのワクワクと切れ味が入門的なポケットリファレンスで伝えられ，なおかつ開発現場で使う実践的な内容にしたいと思い，今回のポケットリファレンスを執筆しました。

それでは，ワクワクと切れ味が伝えられかつ実践に即した内容を持つPython言語の入門的ポケットリファレンスはどういう特徴を持ってい

るべきでしょうか。それにつきましては、次の3つが満たされるべきであると考えました。

まず内容面について，開発現場でもめったに使わない細かい言語仕様などは大胆に割愛しました。その代わり開発現場で実践的に用いるものは繰り返してでも掲載するようにしました。そして，何よりもコンピュータ言語は実際に世の中で使われてナンボの世界ですので，入門的ポケットリファレンスにもかかわらず応用例を多く掲載しました。また現在，Pythonはバージョン2系列から3系列への移行期という重要な局面ですので，言語仕様の変更についてもできる限り載せるようにしました。

つぎに解説に関しては，シンプルでわかりやすく，文はできる限り短くするべきだと考えました。正確さに拘泥して長い説明を加えるよりは，概念を大掴みに把握した後で実際にコーディングすることで細かいことを取得するべきだという立場をとりました。

そして何よりも，ワクワクと切れ味を実感できかつ実践的であるためには，説明よりもサンプルコードを多く載せるべきであると考え，ほとんど全ての説明にサンプルコードを掲載しました。説明が抽象的で理解しにくくても，サンプルコードを見れば納得できるはずですし，実際に動かしてワクワク感と切れ味を体験していただければと思います。

対象とする読者につきましては，Pythonは経験したことがないが，何らかのプログラミング言語については触れてみたことがある方を想定しています。解説を簡潔にし，サンプルコードを多くしましたので，意欲的な方であればプログラミング経験がなくても問題ないかもしれません。本書との付き合い方は，実際にサンプルコードを入力しながら，本書を読み進めると効果的だと考えます。

本書は最新版のPython2.6およびPython2.5をベースに執筆しています。Python2.6に限定される機能については，その旨の断り書きをしています。一部のモジュールでは現在のところPython2.4でしか動作しないものもありますが，モジュールの重要性を考えて掲載しております。さらにPython3.0についての最新の情報を積極的に盛り込みました。ほとんどの内容はPython2.4, 2.3, 2.2においても問題ないはずです。動作環境はWindows Vista/2003 Server/XP/2000 Server/2000/NT4.0/98とMac OS X10.5/10.4/10.3/10.2，そして一般的なUnix系OSと数々のLinuxディストリビューション，FreeBSD, NetBSD, Soralis, HP UX, などを想定しています。

本書の内容に曖昧な点や誤りがあるとしたら，全て著者の浅学に帰するものです。その場合は本書のサポートページにて訂正をお伝えいたします。

本書は多くの人の支えがなければ実現しませんでした。本書を書く機会を与えてくださった技術評論社書籍編集部編集長の池本公平さん，池本さんの忍耐強さとクレバーな切り込みがなかったらこの本は誕生しませんでした。ありがとうございました。この本のベースとなった著作の共著者であった小松亮介さんと穂苅実紀夫さん，特に小松さんは現在の会社におけるパートナーであり，テックグルとして現場で活躍する姿から常に学んでいます。ありがとう。また，Zope CorporationのJim Fultonさんをはじめとするエンジニアのみなさん，そしてPython Software Foundationの開発者のみなさんに，感謝をささげます。最後に，執筆の時間を割くことを許してくれた著者の家族にありがとうを伝えます。

2009年3月
柏野 雄太

Chapter 07

7-4　with 文

with 文は Python2.6 から正式採用されたフロー制御の文で，try-except や try-finally 文よりも一層高い抽象化を提供します。本節では，with 文を利用する方法，with 文で利用できるオブジェクトの記述の仕方を説明します。

7-4-1　with 文を利用する

with 文は低レベルの処理を隠蔽化する1つの方法で，例外処理の **try-except** や **try-finally** よりも一層高い抽象化を実現するために Python2.6 から導入された構文です。抽象化が高いということは，実装が大変になる反面，その利用は簡単になります。

with 文を利用するには **with** の後にコンテキスト依存の式を書き，その後に省略可能の **as** 式による式の評価の戻りを表す変数名を続け，コロン「**:**」で行末を区切ったあと，**with** ブロックを続けます（**構文 7-4-1-1**）。

構文 7-4-1-1

```
with 式 [as 変数名 ]:
    with ブロック
```

with 文を利用するためには，コンテキスト依存の式から戻ってくるオブジェクトが「コンテキストマネジメントプロトコル」を実装している必要があります。そのために，クラスや関数というオブジェクトを実装する側の負担が大きくなるのですが，その分利用が簡単になります。

例えば **file** オブジェクトはすでにコンテキストマネジメントプロトコルが実装されていますから，すぐに **with** 文を試してみることができます（**サンプルコード 7-4-1-1**）。

サンプルコード 7-4-1-1　with 文の利用

```
>>> with open('some.txt', 'w') as f:
...     f.write('write and close it¥n')
...
>>> f
<closed file 'some.txt', mode 'w' at 0x00A47A70>
>>> f.write('try to write')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file
```

サンプルコードではファイルオブジェクトに文字列を書き込むということを行っています。通常のファイルの扱いは，ファイルを **open()** し，それに書き出しを行ったら，その後に **close()** する必要があります。しかし，ファイルオブジェクトにはすでにコンテキストマネジメントプロトコルとして，コンテキストを抜けるとき，つまりファイルオブジェクトの利用が終わるときに **close()** するように実装されていますので，**close()** をする必要がありません。このために，**with** 文を抜けると **with** 文で利用していたファイルオブジェクトは **closed file** オブジェクトになりますし，このオブジェクトに書き込みを行おうとしても **ValueError: I/O operation on closed file** というエラーとなります。

コンテキストマネジメントプロトコルが実装済みで，**with** 文で利用できるオブジェクトの一覧をファイルオブジェクトを含めて列挙しました（**図 7-4-1-1**）。

図 7-4-1-1　with で利用できるオブジェクト

file
decimal.Context
thread.LockType
threading.Lock
threading.RLock
threading.Condition
threading.Semaphore
threading.BoundedSemaphore

7-4-2 with 文を利用するオブジェクトを書く

with 文の利用は簡単ですが，その **with** 文で利用できるようなオブジェクトを書くとなると少し骨です。それはコンテキストマネジメントプロトコルを実装しなければならないからです。コンテキストマネジメントプロトコルは Python インタプリタにあるコンテキストマネージャーが解釈するために必要な一連の手続きで，そのオブジェクトが利用開始されてから終了されるまでの手続きを定義しています。つまり，コンテキストマネジメントプロトコルは，コンテキストマネージャーに対してそのオブジェクトの取り扱いを教えてあげる一連の手続きなのです。

このことからわかるように，コンテキストマネジメントプロトコルには，コンテキストを開始するときと終了するときに，コンテキストマネージャーがそのオブジェクトをどうやって取り扱ったらよいかを決める定義が最低限必要です。これを定義するのが特殊メソッドである **__enter__()** と **__exit__()** です。

例えば，ファイルオブジェクトを **dir** すればわかりますが，コンテキストマネジメントプロトコルを持ったオブジェクトの定義には全て **__enter__()** と **__exit__()** があります（**サンプルコード 7-4-2-1**）。

サンプルコード 7-4-2-1　dir（file）

```
>>> dir(file)
['__class__', '__delattr__', '__doc__', '__enter__', '__exit__',
'__getattribute__', '__hash__', '__init__', '__iter__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__str__',
'close', 'closed', 'encoding', 'fileno', 'flush', 'isatty', 'mode', 'name',
'newlines', 'next', 'read', 'readinto', 'readline', 'readlines', 'seek',
'softspace', 'tell', 'truncate', 'write', 'writelines', 'xreadlines']
```

コンテキストマネジメントプロトコルをきちんと実装するには，コンテキストマネージャーの詳細が必要で，この本の範囲を超えてしまいます。必要に応じて Python ランゲージリファレンスを参照してください。ここでは，非常に簡単なコンテキストマネジメントプロトコルの実装を行います（**サンプルコード 7-4-2-2**）。

サンプルコード 7-4-2-2　コンテキストマネジメントプロトコルの実装

```
>>> class closing(object):
...     def __init__(self, obj):
...         self.obj = obj
...     def __enter__(self):
...         return self.obj
...     def __exit__(self, *exc_info):
...         try:
...             close_it = self.obj.close
...         except AttributeError:
...             pass
...         else:
...             close_it()
...
>>> with closing(open("argument.txt")) as f:
...     for line in f:
...         print line
...

>>> f
<closed file 'c:¥Python26¥LICENSE.txt', mode 'r' at 0x00A47D90>
```

サンプルコードでは **closing** というオブジェクト強制的に **close** するクラスを定義しています。**__init__** メソッドでオブジェクトをメンバとし，**__enter__** ではただそのオブジェクトをコンテキストマネージャーに渡すだけ，**__exit__** においては，オブジェクトに **close** メソッドがればそれを呼び，もしもなければ強制的に **close** させる **close_it()** メソッドを呼ぶだけです。この **closing()** を呼べば，オブジェクトのコンテキストが終了すれば，全てのオブジェクトが **close** されます。

with 文で利用できるオブジェクトを書くためにもう少し楽な方法があります。それには，**contextlib** パッケージの **contextmanager** を利用します。ここでは，ファイルの入出力について，**file** オブジェクトのコンテキストマネジメントプロトコルを利用しない実装を示します。（**サンプルコード 7-4-2-3**）。

サンプルコード 7-4-2-3

```
>>> from contextlib import contextmanager
>>> @contextmanager
... def open2(file, mode="r"):
... f = open(file,mode)
...     try:
...         yield f
...     finally:
...         f.close()
...
>>> with open2("c:¥¥Python26¥¥LICENSE.txt") as f:
... for line in f:
        print line.rstrip()
```