Python[注1]は、さまざまな分野で広く使われている汎用のプログラミング言語です。日本におけるPythonの知名度は、この数年間で飛躍的に向上しました。しかし、Pythonはその誕生からすでに30年が経とうとしている歴史ある言語で、古くからオープンな開発者コミュニティによって支えられています。

本章では、Pythonのプログラミング言語としての特徴、Pythonの歴史と現況、そしてPythonコミュニティについて説明します。

1.1 プログラミング言語としての特徴

読者のみなさんはPythonにどのような印象を持っているでしょうか。インデントが特徴的、動的型付き言語、可読性が高い、データ分析や機械学習でよく使われる、2系と3系で混乱しそう、実行速度が遅い……などがあると思います。これらの印象は読者のみなさんがPythonを知ったきっかけやタイミング、書いてきた言語や分野などのバックグラウンドによりさまざまです。中には正しいものもあれば、実際にPythonを使い始めると違う印象に変わるものもあるでしょう。ここではまず、Pythonのプログラミング言語としての特徴を紹介していきます。

シンプルで読みやすい動的型付き言語

Pythonは動的型付き言語と呼ばれる分類に属するプログラミング言語です。動的型付き言語は、変数や関数の戻り値の型を指定する必要がないため、C言語やJavaなどの型の指定が必要な静的型付き言語と比べるとプログラムの記述量が少なくなる特徴があります。

たとえば、整数どうしの足し算を行う簡単な関数をC言語で記述すると、次のようになります。

```
int add(int a, int b) {
    return a + b;
}
```

注1　PythonにはC言語で実装された標準実装のCPythonのほかにも、Javaで実装されたJython、Pythonで実装されたPyPyなど複数の実装が存在します。本書でPythonと書く際は、特に注意書きがない限りCPythonを指します。

これに対し、同じ足し算を行うPythonの関数は、次のようになります。

```
def add(a, b):
    return a + b
```

C言語の場合は、intを指定して引数と戻り値の型が整数であること明示しています。しかし、Pythonの場合はそのような型の指定がなく、スッキリした見た目になっています。

ただし、型の指定が不要であっても、Pythonに型がないわけではありません。たとえば、Pythonでは整数型はintクラス、文字列型はstrクラスとして表現されます。そのため、上記add()関数の引数aに整数、bに文字列を入れて実行した場合は、次のように型の不一致を表す例外TypeErrorが発生します[注2]。このように、Pythonは型の変換を自動的には行いません。したがって、コードを書くときにはこのようなエラーが起きないよう型を意識する必要があります。

```
>>> def add(a, b):
...   return a + b
...
>>> add(1, '2')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in add
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

●インデントによるブロックの表現

前項で示した例には、型の宣言以外にもう一つ大きな違いがあります。それはPythonのコードには処理のブロックを決めるための波括弧({})がない点です。Pythonでは、波括弧の代わりに、インデント(字下げ)の深さでブロックを表現します。

たとえば、if文を使ったコードは次のようになります。

```
def even_or_odd(n):
    if n % 2 == 0:
        print('偶数です')
    else:
        print('奇数です')
```

注2　ここでは2.2節で紹介する対話モードでPythonを実行しています。また、本書で対話モードを利用する際は、インデント幅をスペース2つとしています。

並行処理とは、複数の処理を同時に行うことを指します。並行処理は、複雑で学習コストの高い分野ですが、プログラム実行時のパフォーマンスを向上させるためには避けては通れません。

本章では、Pythonで並行処理を実現するための選択肢として、マルチスレッドを使う方法、マルチプロセスを使う方法、イベントループを使う方法の3つの方法を説明します。

10.1 並行処理と並列処理 —— 複数の処理を同時に行う

並行処理とは、複数の処理を同時に行うことを指す用語です。並行処理と似た用語に並列処理もあるため、ここでは並行処理、並列処理、そして逐次処理の違いを説明します。

本章を読み進めるためには、厳密な言葉の定義よりもイメージを持つことが重要です。そのため、3本の記事を書く作業を例に説明します。この例では、記事を書く人がCPUのコアに、記事を書く作業が各スレッド上でコアが処理する内容に対応します。

なお、本章で出てくるスレッドとプロセスは、どちらも処理の単位を表す用語です。プログラムが実行されるときには、プロセスが作成されてCPUやメモリなどのリソースが割り当てられます。各プロセスの中では、1つ以上のスレッドにより処理が行われています。

逐次処理で実行する

まずは、逐次処理から説明します。最初の記事を書き上げてから次の記事に取りかかり、その記事を書き上げてから最後の記事に取りかかる進め方は逐次処理と呼ばれます(**図10.1**)。シングルコア、シングルスレッドで処理を行う場合はこれに該当します。Pythonのプログラムでは、意識的に並行処理として実装しない限りは常に逐次処理となります。

並行処理で実行する

3本の記事を1人で少しずつ進めていくと、その進め方は並行処理(*Concurrent*

processing）と呼ばれます（**図10.2**）。ある瞬間を切り取ると1つの記事に集中していますが、長い目で見ると複数の記事が同時に進んでいると言えます。Pythonのプログラムでは、マルチスレッドを利用する場合はこれに該当します。

並列処理で実行する

友人2人に声をかけ、1人1記事ずつ3本同時に進めるとその進め方は並列処理（*Parallel processing*）と呼ばれます（**図10.3**）。並列処理では、ある瞬間を切り取

図10.1 逐次処理で記事を書く

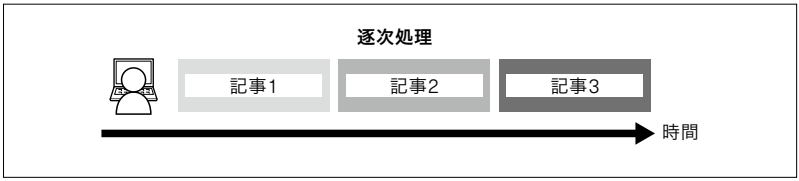

図10.2 並行処理で記事を書く

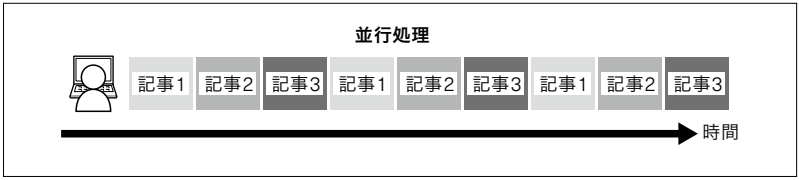

図10.3 並列処理で記事を書く