## 1.7　Python 演習

本書の各章は，アルゴリズムの説明部分だけを読んでも理解できるように構成されているが，学んだアルゴリズムを実際にコンピュータ上で実行させたい人のために，プログラムも紹介している．本書では，プログラミング言語として **Python（パイソン）**を用い，各章に「Python 演習」という節を設けている．本章の Python 演習では，すでに Python の開発環境を整えた読者を対象に，Python の基本的なデータと命令について紹介する．

▶[Python の開発環境]
Python の導入方法と，プログラミングから実行までの手順は付録で紹介しているので，まずは付録を確認してほしい．

### 1.7.1　オブジェクトと型

Python におけるデータや処理の実体を総称して**オブジェクト (object)** と呼ぶ．各オブジェクトは**型 (type)** と呼ばれる「オブジェクトの種類」のいずれかに属している．Python の主な型と，その型に属するオブジェクトの例などを表 1.1 に示す．

▶[型]
各オブジェクトの型の名前は，Python インタプリタで type 関数を各オブジェクトに適用すれば自分で確認できる．

**表 1.1**　Python の主な型

| 型 | オブジェクトの例 | オブジェクトの性質 | | |
|---|---|---|---|---|
| int | 0 や 1 や-2 | | | |
| float | 0.0 や-1.5 | | | |
| bool | True と False のみ | | | |
| NoneType | None のみ | | | |
| str | ’’ や’abc’ | シーケンス | イテラブル | |
| tuple | () や (2,) や (1,2,’a’) | シーケンス | イテラブル | |
| range | range(0,4) | シーケンス | イテラブル | |
| list | [] や [2] や [1,2,’a’] | シーケンス | イテラブル | ミュータブル |
| set | set() や{2}や{1,2,’a’} | | イテラブル | ミュータブル |
| dict | {}や{1:’a’,2:’b’} | | イテラブル | ミュータブル |

型が重要なのは，型によってオブジェクトに適用できる演算が異なるためである，たとえば，int 型には整数，float 型には小数，str 型には**文字列 (string)** が，それぞれ属している．+演算は整数や小数に対しては足し算を意味するが，文字列に対しては連結を意味する．

```
>>> 1 + 2
3
>>> ’1’ + ’2’
’12’
```

bool 型には真偽値の True（真）と False（偽）のみが属する．NoneType 型は「値が指定されていない」ことを意味する値 None のみが属する特殊な型である．

tuple 型のオブジェクトは**タプル (tuple)**，list 型のオブジェクトは**リスト (list)** とも呼ばれる．どちらも有限個のオブジェクトからなる列を表すが，後述するミュータブルという性質があるかどうかが主な違いである．set 型のオブジェクトは**集合 (set)** とも呼ばれ，タプルやリストと異なり要素の順序と重複を無視する．このことを比較演算子==を使って確かめよう．==は，二つのオブジェクトの見た目が異なっていても，意味が同じであれば True を返す演算子である．==は連続して使えることにも注意しよう．

```
>>> [3,2,3] == [2,3,3]
False
>>> {3,2,3} == {2,3,3} == {2,3}
True
```

型が異なっていても，より広い種類への型変換が自動的に行われて「意味が同じ」と判定されることがある．

```
>>> True == 1 == 1.0
True
```

range 型は 3 章で，dict 型は 13 章で解説する．

### 1.7.2　関数と変数

Python に実行させたい「処理」を記述して名前をつけたものを**関数 (function)** という．1.1 節で例に挙げた整数の和を求めるアルゴリズム 1,2 をそれぞれ関数 sum1 と sum2 として実装した Python プログラムをソースコード 1.1 に示す．

**ソースコード 1.1**　sum_of_numbers.py

```
1 def sum1(n):
2     s = 1
3     for i in range(2, n + 1):
4         s = s + i
5     return s
6
7 def sum2(n):
8     return n * (n + 1)//2
```

関数を呼び出して実行するには，関数名のあとに処理に必要な情報をカッコに囲んで与えればよい．この「処理に必要な情報」を**引数 (argument)** と呼ぶ．今回の関数 sum1 と sum2 の引数は正の整数である．引数を複数取る関数もある．

```
>>> from sum_of_numbers import sum1,sum2
>>> sum1(4)
10
>>> sum2(4)
10
```

関数は def 文で定義し，def のあとに関数名と，引数の名前（今回の関数 sum1 と sum2 の場合は n）を記述する．実行させる処理の内容は:（コロン）の次の行からインデント（字下げ）をして記述しなければならない．

return 文は関数の実行終了と返り値の指定をする文である．**返り値 (return value)**（あるいは**戻り値**）とは，関数の実行が終了した際にその関数を呼び出した側に伝えることのできる情報である．ソースコード 1.1 の関数 sum1 と sum2 の返り値はいずれも 1 から n までの和である．したがって，どちらも同じ結果を返す．

▶[インデント]
本書ではインデントは半角 4 文字で統一しておく．インデントは tab キーではなくスペースキーで行うこと．

「名前=オブジェクト」という文でオブジェクトに名前をつけることができ，その名前のことを**変数 (variable)** という．

正式には変数がオブジェクトを**参照する (refer)** という．

変数名は定義された関数の中でしか有効でない．たとえば，ソースコード 1.1 の中の n は変数であるが，sum1 が受け取る n と sum2 が受け取る n は区別される．

▶[変数]
変数名に使えない文字や単語もあるので，詳しくは「The Python Language Reference」を参照すること．

▶[参照]
変数にオブジェクトを「代入」するという表現を使うこともある．しかし，変数はオブジェクトを格納する箱のようなものではなく，オブジェクトにつける名前でしかないので注意すること．詳しいことは後述する．

### 1.7.3　オブジェクトの性質

表 1.1 で**シーケンス (sequence)** と書かれているオブジェクトはいずれも，要素を $0, 1, 2, \ldots$ と番号付けて格納しているオブジェクトであり，後ろに [番号] をつけることで個々の要素を評価したり，+演算で連結したりすることができる．後半の章で紹介するスライスという演算も使える．

```
>>> x = [10,20,30]
>>> x[1]
20
```

**イテラブル (iterable)** なオブジェクトは，格納している要素一つ一つについて同じ処理を繰り返して行わせるための，for 文が使える．詳しいことは 3 章以降で紹介するが，要素の数を len 関数で確認したり，特定のオブジェ

クトの有無を in 演算で判定したりもできる．ソースコード 1.1 の 3,4 行目は，変数 i に 2 から n の値を順に代入し，そのたびに文 s=s+i を実行させるという意味である．これにより関数 sum1 はアルゴリズム $1+2+\cdots+n$ を実現している．

ミュータブル (mutable) なオブジェクトは，オブジェクトの中身を変更できる．

▶[ミュータブル]
「変数がどのオブジェクトを参照しているか」という対応付けを変更するのではなく，オブジェクトそのものの中身を変更できることが特徴である．

ミュータブルなオブジェクトを変数で参照する場合は注意が必要である．なぜなら，同じオブジェクトを複数の変数が参照することもでき，ある変数の参照するオブジェクトの変更が，他の変数にも反映されることがあるからである．たとえば，以下の変数 x,y は同じリストを参照している．ここで x の中身を変更すると，y の中身も変更されることがわかる．

```
>>> x = [10,20,30]
>>> y = x
>>> x[1] = 40
>>> y
[10, 40, 30]
```

なお，プログラム中で二つの変数が参照しているオブジェクトが同じかどうか確認したい場合は，比較演算子 is を使えばよい．実はオブジェクトには製造番号のような番号がつけられており，is は両者の製造番号が同じ場合に True を返す演算子である．

```
>>> x = [1,2,3]
>>> y = x
>>> x is y
True
```

ミュータブルなオブジェクトにはデメリットもある．集合は「重複」を無視するが，重複とは下のように「意味が同じこと」を指す．

```
>>> {1, 1.0, True} == {1}
True
```

仮に，ミュータブルなオブジェクトを集合に入れてしまうと，その値を変更するたびに重複の判定が必要になるため，ミュータブルなオブジェクトは集合には入れられない決まりになっている．タプルはミュータブルではないので，集合に列を入れたい場合は，タプルを入れればよい．