第 4 章

関数・モジュール・クラス

プログラムを書いていると，同じ処理を何度も繰り返し記述しなければならない場面に出くわします。そのような時に便利なのが**関数** (function) です。よく使う処理をひとまとまりにして名前を付けておけば，何度でも呼び出して再利用できます。さらに，複数の関数をファイル単位でまとめたものが**モジュール** (module) です。実用上重要な機能を集めたモジュールの代表的なものが，NumPy, SciPy, Matplotlib，Pandas です。本章では関数とモジュールの定義・利用方法を学び，データ構造と関数を一体化した**クラス** (class) についても実例を通じて理解を深めていきます。

4.1 関数 (function) とは？

Python 言語における関数とは，**必要な機能をまとめ，再利用しやすくしたもの**です。数学でいうところの関数とよく似ており，引数 (argument) を受け取り，処理を行い，返り値 (return value) を返すという構造を持っています。

例えば数学における関数 $f(x) = x^2 + 2x + 3$ を定義しておけば，$f(3) = 3^2 + 2 \cdot 3 + 3 = 18$ のように，引数として $x = 3$ を与えるだけで計算結果が得られます。Python でも全く同じ考え方で関数を定義できます。

下記の `func_ex.py` では，返り値あり (`func1`) と返り値なし (`func2`) の 2 種類の関数を定義しています。

ソースコード 4.1 `func_ex.py`

```
1  # func_ex.py: 関数の例
2
3  # f(x) = x^2 + 2x + 3
4  def func1(x): # x は引数
5      return x ** 2 + 2 * x + 3 # 返り値
6
7  # f(x) = x^2 + 2*x + 3を表示
8  def func2(x):
9      val = x**2 + 2 * x + 3
10     print(f'func2({x:g}) = {val:g}')
11     return # 返り値なし
12
13 # メイン関数
14
15 # 返り値あり
16 print('func1(3) = ', func1(3))
17 # 返り値なし
```

```
18 func2(3)
```

def キーワードで関数を定義し，return 文で返り値を指定します。返り値が不要な場合は return のみを記述するか，return 文を省略することもできます。

4.2 関数とモジュール

関数定義の部分を**別の Python スクリプトファイルに分離**し，必要に応じて読み込む仕組みがモジュール (module) です。関数定義をまとめたファイルを**モジュールファイル**，それを利用するファイルを**メインファイル**と呼びます。import **モジュール名**と記述することでモジュールを読み込めます。

以下の例では，func_def.py に関数定義をまとめ，func_main.py からそれを呼び出しています。

ソースコード 4.2 func_def.py：関数定義モジュール

```
1  # func_def.py: 関数の定義
2
3  # f(x) = x^2 + 2x + 3
4  def func1(x): # x は引数
5      return x ** 2 + 2 * x + 3 # 返り値
6
7  # f(x) = x^2 + 2*x + 3を表示
8  def func2(x):
9      val = x**2 + 2 * x + 3
10     print(f'func2({x:g}) = {val:g}')
11     return # 返り値なし
```

ソースコード 4.3 func_main.py：メイン関数部分

```
1  # func_main.py: 関数モジュールの呼び出し
2  import func_def as f # f という別名を付ける
3
4  # メイン関数
5
6  # 返り値あり
7  print('func1(3) = ', f.func1(3))
8  # 返り値なし
9  f.func2(3)
```

4.3 モジュールと（関数の）名前空間

モジュールを import する際には，**名前空間** (namespace) の扱い方によって 3 通りの書き方があります。

(1) **モジュール名＝名前空間**　import func_def と記述し，func_def.func1(3) のようにモジュール名を付けて呼び出す。

(2) **モジュール名の別名（エイリアス）**　import func_def as f と記述し，f.func1(3) のように短縮名で呼び出す。

(3) **同一名前空間**　from func_def import func1, func2 と記述し，モジュール名なしで func1(3) と直接呼び出す。

ソースコード 4.4　func_main.py：3 通りの import 方法

```
1  # func_main.py: 関数モジュールの呼び出し
2  import func_def # (1) 別名なし
3  import func_def as f # (2) f という別名を付ける
4  from func_def import func1, func2 # (3) 同一名前空間
5
6  # メイン関数
7
8  # (1) 別名なし
9  print('func_def.func1(3) = ', func_def.func1(3))
10 func_def.func2(3)
11
12 # (2) 別名あり
13 print('f.func1(3) = ', f.func1(3))
14 f.func2(3)
15
16 # (3) 同一名前空間
17 print('func1(3) = ', func1(3))
18 func2(3)
```

4.4　クラス (class) ＝データ構造＋関数定義

クラス (class) とは，よく使うデータの集まり（データ構造）と，そのデータに対する操作方法（関数）をひとまとめにして定義したものです。クラスもモジュールと同様に Python スクリプトとしてまとめておくのが一般的です。クラスから生成されたオブジェクトのことを**インスタンス** (instance) と呼びます。

例として，住所録の 1 件分のデータを表現する MyAddress クラスを考えます。このクラスは次のフィールド (クラス内部の変数) を持ちます。

- no:　int — ID 番号（整数型）
- name:　str — 名前（文字列型）
- yomigana:　str — 名前の読み方（文字列型）
- postal_address:　str — 住所（文字列型）

4.4.1　MyAddress クラスの定義

クラス内に定義する関数のうち，名前の前後に__（アンダースコア 2 つ）が付くものを**定型関数**と呼びます。代表的なものとして，クラスのインスタンスを初期化する**イニシャライザ**__init__と，インスタンスを文字列化する__str__があります。その他に任意の名称で定義できるクラス定義関数（ここでは print_all）を追加できます。

ソースコード 4.5　address_class.py：住所録クラス

```
1  # address_class.py: 住所録クラス
2  class MyAddress:
3      # no, name, yomigana, postal_address, memo
4      # クラス初期化関数
5      def __init__(
6              self,
```

```
7             no: int, # ID番号
8             name: str, # 名前
9             yomigana: str, # 名前の読み方
10            postal_address: str # 住所
11    ):
12        self.no = no
13        self.name = name
14        self.yomigana = yomigana
15        self.postal_address = postal_address
16        return
17
18    # 表示
19    def print_all(self):
20        print(f'{self.no:5d}: {self.name}, {self.yomigana}, {self.postal_address}')
21        return
22
23    # 文字列化
24    def __str__(self):
25        return str([self.no, self.name, self.yomigana, self.postal_address])
```

4.4.2 MyAddress クラスの使用例

定義したクラスは，from モジュール名 import クラス名でインポートして使用します。クラス名に引数を渡して呼び出すことでインスタンスが生成されます。

ソースコード 4.6 example_myaddress.py：アドレス帳使用例

```
1  # example_myaddress.py: アドレス帳使用例
2  from address_class import MyAddress # MyAddress クラス定義
3
4  # メイン関数
5
6  # ai と ota は MyAddress インスタンス
7  ai = MyAddress(1, '安威太郎', 'あいたろう', '茨木市西安威２丁目')
8  ota = MyAddress(2, '太田次郎', 'おおたじろう', '茨木市太田東芝町１－１')
9
10 # 住所録表示
11 ai.print_all()
12 ota.print_all()
13
14 # 文字列化
15 print(str(ai))
16 print(ota)
```

このスクリプトを実行すると，次のような結果が得られます。

example_myaddress.py の実行結果

```
    1: 安威太郎，あいたろう，茨木市西安威２丁目
    2: 太田次郎，おおたじろう，茨木市太田東芝町１－１
[1, '安威太郎', 'あいたろう', '茨木市西安威２丁目']
[2, '太田次郎', 'おおたじろう', '茨木市太田東芝町１－１']
```

演習問題

1. モジュールファイル mytool.py に，2 次方程式 $ax^2 + bx + c = 0$ の解を求める関数 quad_eq(a, b, c) を実装せよ。判別式 $D = b^2 - 4ac$ を用いて，$D \geq 0$ の場合は実数解，$D < 0$ の場合は複素数解を表示すること。また，この関数を mytool モジュールとして読み込み，係数 a, b, c を入力として 2 次方程式の解を表示するメインスクリプト quadratic_eq2.py を作成せよ。
   実装例を以下に示す。

ソースコード 4.7　mytool.py：関数定義

```
# mytool.py: 関数定義
import math # sqrt 関数
def quad_eq(a, b, c):
    # 判別式
    d = b ** 2 - 4 * a * c

    if d >= 0: # 実数解
        print('Real␣solutions:\n')
        x1 = (-b + math.sqrt(d)) / (2 * a)
        x2 = (-b - math.sqrt(d)) / (2 * a)
        # 出力
        print(f'x1␣=␣{x1:25.17e}')
        print(f'x2␣=␣{x2:25.17e}')
    else: # 複素数解
        print('Complex␣solutions:\n')
```

ソースコード 4.8　quadratic_eq2.py：2 次方程式を解く（関数呼び出し版）

```
# quadratic_eq2.py: 2次方程式を解く(関数呼び出し版)
#import math # sqrt 関数
import mytool # quad_eq 関数

# 係数入力
a = input('a␣=?␣')
b = input('b␣=?␣')
c = input('c␣=?␣')
a, b, c = float(a), float(b), float(c)
print(f'␣␣{a:25.17g}␣*␣x^2')
print(f'+␣{b:25.17g}␣*␣x')
print(f'+␣{c:25.17g}␣=␣0')

# 2次方程式の解
mytool.quad_eq(a, b, c)
```

2. mytool.py に素因数分解を行う関数 prime_fact を追加し，メイン関数でこれを呼び出して，与えられた整数 n の素因数分解を表示する Python スクリプトを作成し，実行結果を確認せよ。
   [**ヒント**] 前回の課題で作成した e_sieve.py を改良すること。