

第 9 講 データサイエンスにおけるプログラミング基礎

【学習到達目標】

- ・プログラミングの基本的な概念と構文を理解し、データ処理や分析に必要な基本操作を実行できる。
- ・代表的なプログラミング言語（例：Python）の基本的な文法とライブラリの使い方を習得し、簡単なデータ分析プログラムを作成できる。
- ・データの読み込み、加工、可視化といった基本的なプログラミングスキルを身につけ、データサイエンスの基礎的な作業を自律的にこなせる。

1. プログラミングの基本概念

(1)プログラミングとは

プログラミングとは、あらかじめ定めた計算やファイル操作などの手続きを計算機に処理させるための命令（プログラム）を作成することである。データサイエンスにおいては、データの収集、前処理、分析、可視化といった一連の作業を自動化し、効率的に行うためにプログラミングが必要となる。

プログラミングのために開発された言語をプログラミング言語という。プログラミング言語にはCやJavaなど様々な種類があるが、データサイエンス、特に機械学習分野においてはPythonという言語が良く用いられる。

(2)プログラミングの基本概念

プログラミングにおいて、具体的な値を変化させることができる存在を変数、値の種類を型と呼ぶ。型の具体例としては、整数型(int 型)、浮動小数点型(float 型)のような数値の他に、"apple"や"Hello World!"などの文字列型(str 型)や、[1, 3.14, "banana"]のように複数の要素を並べた配列型(list 型)などが存在する。これらの変数と、変数に対して四則演算などの演算を行う演算子や、条件分岐、ループ、関数の適用などを組み合わせることで様々な操作をコンピュータにさせることができるようになる。

統計学的な処理
に対しては、R という言語が使われることも多い。

2. Python と基本的な文法

以降は Python を例に、具体的なプログラミングの方法を解説する。この解説では、公式版の Python をコンソール(コマンドプロンプトやターミナル)上で動かすことを想定する。ダウンロードは公式サイト

(<https://www.python.org/downloads/>)から行い、インストール時はウィザードの指示に従うのが無難である。また、その際「Add python.exe to PATH」というチェックボックスがでたらチェックを入れておくこと。

プログラムを実行する際は、ソースコードをエディタで編集し、拡張子を「.py」に指定した上で、コンソール上でソースコードを格納したディレクトリ(フォルダ)に移動し、そのコードのファイル名を指定する。ただし、この際「python (ファイル名)」などのように「python で実行する」という意味の命令をする必要があるか、ファイル名のみを入力するだけで実行できるかは OS や環境構築などに依存して変わるので注意すること。また、Python の場合コンソールで単に「python」(環境によっては「python3」などとする必要がある場合もある。)と入力すると対話環境に入ることができる。こちらはソースコードを保存したファイルを経由せず、直接命令を書き込んで処理を行うことができるので、主に短いプログラムや関数の機能をチェックする際に便利である。

(1)変数への代入と基本操作

まずは変数に値を代入するプログラムを示す。

```
x = 1
fruit = "apple"
```

このプログラムは、「変数 x に 1 という値を代入し、変数 fruit に"apple"という値を代入せよ」という意味であり、実際に実行するとその通りに値が代入される。ここで、「=」という記号は数学のように「両辺が等しい」という意味ではなく、「左辺の変数に右辺の値を代入する」という意味になる点には注意すること。また、プログラム上では、文字列は引用符(' ')または二重引用符(" ")で囲う。そのため、例えば、単に x と書いた場合「変数 x」と解釈される一方、'x'または"x"と書くと「文字列 x」と解釈される。また、変数名は fruit のように複数の文字からなってもよい。

Python には Anaconda などの開発支援環境も存在するが、これを使う場合ライブラリのインストール時に使うコマンドなどが変わる点に注意。本稿ではこうした追加機能の詳細には立ち入らない。

このプログラムは代入以外の操作をしていないため、実行しても何も出力されない。

なお、Python は変数に値を代入した時点で、その変数の型を自動的に判定するが、C や Java など、Python 以外の言語の場合、変数の型をあらかじめ宣言しておく必要があるものも存在する。

次に、以下のように int 型や float 型の変数に対する四則演算を行うプログラムを作ってみる。

```
a = 5
b = 2

print(a + b, a - b, a * b, a / b, a // b, a % b)
```

こちらは

```
7 3 10 2.5 2 1
```

という結果が得られる。各演算子の意味は以下の表 1 の通りである。ここで、「a // b」と「a % b」は整除法（余りが出る割り算）の商と剰余に近いものだが、Python の場合 a や b が float 型でも定義できる点に注意。（他の言語では仕様が異なる例が多い。）四則演算以外で重要な演算子としては、「a ** b」（a の b 乗）などがある。

また、print() は「括弧の中身を表示せよ」という命令である。

演算子	意味
a + b	和
a - b	差
a * b	積
a / b	(実数としての)商
a // b	(a / b)を超えない最大の整数
a % b	a - b * (a // b)

表 1 Python における四則演算の演算子

(2)条件分岐と繰り返し

Python においては、条件分岐には if、繰り返し処理には for または while を使う。まずは if による条件分岐の例として、a に代入した数値（今回は 57）が 3 の倍数かどうかを判定するプログラムを考える。

Python においては、「1.0」などのように小数点を明記した値は float 型になることに注意。

```

a = 57

if a % 3 == 0:
    print("a is a multiple of 3")
else:
    print("a is not a multiple of 3")

```

実行結果は以下のようになる。

```
a is a multiple of 3
```

このプログラムにおいて、「a % 3 == 0」は「a を 3 で割った余り(a % 3) が 0 に等しい」という意味であり、その条件を満たす場合に行う命令

「print("a is a multiple of 3")」は一段階インデントを下げて記述する必要がある。条件を満たさないときに別の命令を行いたい場合は、if 文終了後に「else:」と記述してから、同様にインデントを下げて条件を満たさない場合の命令を記述する。また、条件 A を満たさないときに、更に別の条件 B を満たしているか判別してから命令をしたい場合、「else:」の代わりに「elif 条件 B:」とすればよい。ただし、どの場合においても、条件下での命令が複数行にわたる場合は、各行のインデントを揃える必要がある。インデントは半角スペース何個分でも問題ないが、4 つにするのが主流である。（タブを使うことも可能ではあるのだが、混乱の元になるため極力避けること。）

変数の大きさを比較したり、等しいかどうかを判別したりする演算子（比較演算子という）を表 2 にまとめた。int 型や float 型の場合、これらの演算子は単純に大きさを比較する。

比較演算子	意味
a == b	a と b が等しい
a != b	a と b が等しくない
a > b	a が b より大きい
a >= b	a が b 以上である
a < b	a が b より小さい
a <= b	a が b 以下である

表 2 Python における比較演算子

「elif」は「else if」の略である。

「a = b」は「a に b を代入する」という意味になるので、「a == b」とは全く異なる意味である点に注意。

また、str 型では文字列を辞書順に並べて後に来る方を「大きい」とみなすなど、他の変数型にも適用可能な場合があるので、それらに用いる場合は大小比較のルールなどを確認すること。

なお、Python の場合、これらの条件文は「True」または「False」のいずれかの値を返す bool 型という型として扱われている。bool 型の値には、表 3 のような論理演算を適用することもできる。そのため、例えば「a >= 0 and a < 10」は「a は 0 以上 10 未満」という意味になる。

論理演算子	意味
not A	A でない（否定）
A and B	A かつ B（論理積）
A or B	A または B（論理和）

表 3 Python における論理演算子

実際に論理演算の例を挙げてみる。

```
x = True
y = False

print(x and y, x or y)
```

実行結果は以下ようになる。

```
False True
```

このコードのように、変数に bool 型の値を代入することも可能である。また、bool 型変数を「if x:」のように条件文の判定に直接利用することもできる。

次に、繰り返しの例として 0 から 9 までの整数を足すプログラムを示す。

```
total = 0
for n in range(10):
    total += n

print(total)
```

これは以下のような結果になるはずである。

45

このコードにおいて、「range(10)」とは「0 以上 10 未満の整数値を順に取り出せるもの」を意味する特殊な型で、「for n in range(10):」は「n を 0 から 9 までの整数値を順番に取らせつつ、インデントを下げた指示を繰り返す。」という意味になる。

より一般に、for は「for (変数名) in (変数を順に動かす範囲を指定するもの):」という書き方で使われる。ここで、「繰り返す範囲を指定できるもの」のことをイテラブルと呼ぶ。イテラブルとしては range() の他に list などもある。ただし、例えば「for n in [2,1,3]:」と書けば「n を 2, 1, 3 の順で変化させつつ、指定の処理を繰り返す。」という意味になるなど、list をイテラブルとして使用した際は変数が list に格納されている順で変化することに注意すること。なお、「total += n」は「total = total + n」、つまり「total に n を足したものを改めて total 自身に代入する」という意味の命令である。似た使い方をする演算子として、「-=」、「*=」、「/=」、「%=」なども存在する。

繰り返し処理には for の他に while を使う構文も存在する。こちらで 0 から 9 までの整数を足すプログラムを書くと、以下ようになる：

```
total = 0
n = 0
while n < 10:
    total += n
    n += 1

print(total)
```

ここで、「while (条件):」は「条件が満たされている間、指定された操作(= インデントを下げた記述した処理)を繰り返す」という意味である。while は for のように繰り返す範囲を明確に指定する必要が無いが、条件が満たされなくなる状況が発生しない限り繰り返しを続けてしまう点に注意が必要である。例えば上のプログラムにおいて「n += 1」を書き忘れてしまうと、n はいつまでも 0 のままとなってしまふ。こうなると、計算結果を間違えるだけでなく、n が 10 以上になることがないために、プログラム自体が永遠に止まらなくなる。

range() は開始値、終了値、刻み幅の最大 3 つの引数を指定できるが、本講では詳細は省略する。

また、以下のように、条件分岐や繰り返しの中に、別の条件分岐や繰り返しを書くこともできる：

```
for n in range(10):
    if n % 2 == 0:
        print(n, "is an even number.")
    else:
        print(n, "is an odd number.")
```

この際は、インデントを更に下げる必要があることに注意。

(3)関数の定義と呼び出し

print()のように、特定の処理をまとめて行えるものを関数という。関数にはこの print()同様に Python 自体が既に定義しているものや、後述するライブラリに含まれているものなど多くの種類がある。プログラムの作成者はそうした既に与えられている関数以外にも、独自の関数を定義して使うこともできる。そのため、自作プログラム内で頻繁に行いたい処理などを独自の関数としてまとめておくと、コードを簡潔に表すことができる。

以下に関数を定義し、使用するコードの例を示す：

```
def f(x):
    y = x + 1
    return 2 * y

print(f(1))
```

このコードにおいて、「def f(x):」という箇所は、「入力された値 x に対してある処理をする関数 f を定義する。」という意味になる。関数に inputsする値を引数と呼ぶ。今回の例では引数は x 一つのみだが、「def g(x, y):」などのように複数の引数を持つ関数を定義することも可能である。また、関数が行う処理も条件分岐や繰り返し処理同様にインデントを下げた記述する必要がある。return 以降に書かれている値は関数の出力となる。今回のコードの場合、関数 f(x)は引数 x に対して、「2 * y」、つまり「2 * (x + 1)」という値を計算して出力する関数として定義されている。その上で、最後の行で引数として x=1 を代入した場合の関数の値 f(1)を表示させているのである。よって、このコードの実行結果は以下になる。

関数名、引数名共に、「def func(xxx):」などのように複数の文字からなっても構わない。

3. Python のライブラリ

プログラム開発者向けに便利なプログラムをまとめたものをライブラリという。Python のメリットは、文法がシンプルだけでなく、以下の例のような多数のライブラリが開発されていることにある。

NumPy : 配列の操作、特に線形代数に関する演算を高速で行う。

pandas : データフレームの作成や欠損値処理など、データ管理において有用な機能が充実している。

scikit-learn : 各種の機械学習アルゴリズムを実行する。

Matplotlib, seaborn : データをグラフとして可視化する。

ライブラリをインストールするときは、コンソール上で

```
>>pip install numpy
```

などのように「pip install (ライブラリ名)」と入力する。また、実際のプログラムでライブラリの機能を使用する際は、ソースコード上で事前に「import」というコマンドによってライブラリを読み込む必要がある。

ここでは、例として NumPy を使うコードを紹介する。具体的には配列の要素の合計、平均値、中央値を計算する。

```
import numpy as np

a = np.array([1.0, 5.0, 3.0])

print(np.sum(a)) # 配列の要素の和
print(np.mean(a)) # 配列の要素の平均
print(np.median(a)) # 配列の要素の中央値
```

このコードの1行目の「import numpy as np」という文は、「以降 numpy を np という名前で使う」という宣言である。その上で、numpy に含まれる関数「numpy.sum()」、「numpy.mean()」、「numpy.median()」をそれぞれ「np.sum()」、「np.mean()」、「np.median()」という名前で呼び出している。このコードは以下のような結果になるはずである。

```
9.0
3.0
3.0
```

なお、「pip install」でインストールする際と「import」で実際に使用する際の名前が異なるライブラリ（scikit-learn など）も存在することには注意。そのため、可能であれば各ライブラリの公式サイトの手順に従ってインストール、及び使用することが望ましい。

4. 大規模言語モデル(LLM)の利用

Python を使いこなすには、Python 本体だけでなく、各種ライブラリの機能を把握する必要がある。従来はそれらを習得する際、教科書や Web 上の資料を調べる必要があったが、近年では ChatGPT や Copilot といった大規模言語モデル (LLM) の性能が向上し、自然言語による指示でもかなり正確なコードを書けるようになった。そのため、特に使い慣れていないライブラリがある場合や、ライブラリがサポートしているかどうか分からない機能を使いたい場合などは、LLM に相談することも検討する価値がある。以下に ChatGPT 5 にコードを書いてもらった場合の例を挙げる。

ChatGPT 5 への指示 : Matplotlib で関数 $y = x^2$ のグラフを描くソースコードを作ってください。

ChatGPT 5 からの回答 : はい、Matplotlib を使って関数 $y = x^2$ のグラフを描く基本的なソースコードを示します。

```
import numpy as np
import matplotlib.pyplot as plt

# x の範囲を設定
x = np.linspace(-10, 10, 200) # -10 から 10 まで 200 点
y = x ** 2 # 関数 y = x^2

# グラフの描画
plt.figure(figsize=(6, 4))
plt.plot(x, y, label="y = x^2")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Graph of y = x^2")
plt.grid(True)
plt.legend()
plt.show()
```

NumPy のように、正式名称には大文字が混じりますが、使用する際は小文字でライブラリ名を指定する必要がある例にも注意が必要。

(以下、コードの要点に関する説明があったが、省略)

このように、LLM はライブラリを使うようなコードでも容易に書いてくれる。ただし、現状では必ず正しいコードを出力してくれるとは限らないため、必ず動作確認をすること。特に、長く複雑なコードの場合はどこかで齟齬を発生させる危険性が増すため、そうした場合は想定している操作全体を幾つかの小さい部分に分け、別々にコードを作ってもらうのも良い。(例えば操作 A で計算した値を操作 B に処理させるならば、操作 A のコードと B のコードを別々に作ってもらうなど) また、漠然とした指示を出した場合 LLM 側が指示内容を誤解する恐れがあるため、「どのライブラリを使ってどんな計算をさせたいのか」などといった指示内容は、可能な限り具体的に書くのが望ましい。

参考文献

[1] 三谷純、「Python ゼロからはじめるプログラミング」(翔泳社、2021)

課題

1. Python を用いて、リストに格納された数値データの平均値と中央値を計算するプログラムを作成してください。
2. pandas ライブラリを使って、CSV ファイルからデータを読み込み、特定の列の欠損値を平均値で埋める処理を行うコードを書いてください。
3. matplotlib や seaborn を用いて、データの散布図とヒストグラムを作成し、データの分布や関係性を視覚的に表現してください。

Python 本体、および NumPy や pandas などの主要ライブラリは利用者人口が多く、LLM が参考にできるような学習データが多いので、比較的高確率で正確なコードを書ける。逆に、利用者人口が少ない言語やライブラリについては精度が落ちる可能性があるので注意。

講義やテキストで説明した内容だけで今回の課題のコードを書くのは難しいので、LLM に質問しながら書くことを推奨する。