第一章 Pythonガイド

作成：Weiwei Wan
修正：Weiwei Wan, Takuya Kiyokawa

2024 年度 春〜夏学期

Python は非常に重要なプログラミング言語です．シンプルで直感的な構文を持ち，初心者にも優しいです．コードを読みやすく，書きやすいことが特徴です．データ解析，機械学習，ウェブ開発，科学計算など，様々な分野に特化したライブラリが豊富に揃っています．これにより，開発効率が大幅に向上します．Python は Windows，macOS，Linux など，さまざまなプラットフォームで動作します．このため，開発環境に依存せずに利用できます．また，データサイエンスや機械学習，現在流行っている AI の分野で特に強力です．NumPy，Scikit-learn，PyTorch などのライブラリを使用して，複雑なデータ解析やモデルの構築が簡単に行えます．これらの理由から，WRS システムの開発は Python 言語を利用します．本章は Python の本的な概要から，すぐに始められるように簡単なコード例を交えて Python の快速入門を説明します．

1　コンパイルする必要はありません

Python はインタプリタ方式のプログラミング言語の一種です．皆さんは C 言語の方が馴染みがあるかもしれません．C 言語はインタプリタ方式ではなく，コンパイル方式のプログラミング言語です．C 言語で記述されたソースコードは，前処理（文字列調整など）と機械語への翻訳（狭義のコンパイル）を通してオブジェクトファイルに変換され，それらがリンクされることで，最終的な実行可能なプログラムに変換されます．コンパイル方式の場合は，実行前にコンパイルする手間と時間はかかりますが，実行時の速度が速い点が長所です．Python はインタプリタ方式で，Python で書かれたプログラムは実行前にコンパイルを必要としない代わりに，プログラムを一行ずつ機械語として解釈しながら実行されます．コンパイル方式と比較すると実行速度が遅いのが短所です．ただ，作成したソースコード（スクリプトとも呼ばれる）をただちに実行できるという利点があります．

また，インタプリタ方式の場合は，ソースコードを機械語に翻訳するためのソフトウェア（コンパイラーと呼ばれる）は必要はありません．その代わりに逐次解釈用のソフトウェア（インタプリターと呼ばれる）が必要です．C 言語でコンパイルされたプログラムはコンパイラーのない計算機上でも実行できる一方，Python を用いて作成したプログラムはインタプリターのない計算機上では実行できません．皆さんは，コンパイルしたプログラムをインストールする際に，依存ライブラリーを同時にインストールすると思います．インタプリタ方式の場合は，インタプリターも，実行に必要な同時にインストールすべき依存ソフトウェアであると捉えると，この点も Python を含むインタプリタ方式の短所と言えるかもしれません．

2　CPython は C 言語で作成された Python のインタプリター

Python で書かれたプログラムを翻訳するインタプリターは，C 言語のコンパイラーのように，異なる組織によって開発された様々な種類が存在しています．その中でも広く利用される，リファレンス実装である C 言語で書かれた CPython があります．CPython の特徴として，一般的な OS（Windows，Linux など）がインストールされた環境で動作し，様々な C/C++のライブラリとのインターフェースを提供しており，Python によるプログラムの中で C/C++のライブラリの関数を呼び出すことができます．CPython の他には，Java の仮想マシン上で実行される Java で書かれた Jython やマイクロソフト社の.Net 言語で書かれた IronPython も利用できます．それぞれ，Jython

は Java クラスや Java のライブラリをアクセスする機能，IronPython は.Net のライブラリをアクセスする機能を提供しています．本書で紹介する WRS の開発においては，C/C++で記述された多くのライブラリにアクセスする必要があるため，CPython を利用することを想定しています．

CPython は，Python の公式コミュニティ（https://www.python.org）によって管理されています．CPython は「cpip」という管理ソフトを使用して簡易的に様々なライブラリ（.py ファイルのモジュールを集めたパッケージをインストール可能にしたもの）をインストールおよび削除することが可能です．CPython の他にも，異なる組織によって様々な C ベーストインタープリターのディストリビューションが開発されていますが，そのほとんどが CPython に基づいて作成および修正されています．例えば，Anaconda は人気のあるディストリビューションの一つです.「cpip」を使用して Python のパッケージを管理する CPython に対して，Anaconda は独自のライブラリリポジトリを運営しており,「conda」を使用してパッケージを管理できます.「cpip」のライブライリポジトリには時々壊れたパッケージや使えないパッケージが存在しますが，Anaconda のライブライリポジトリは細かくチェックされており,「conda」によって管理されるパッケージの問題は比較的少ないです．その故，Anaconda が人気のあるディストリビューションと考えられます．

3　統合開発環境を使えば開発は便利

本資料は Python の公式コミュニティに発行される CPython を使う予定です．最近，pip（公式パッケージ管理ツール）によって，CPython も Anaconda と同じように簡単に使えます．CPython を使うと，Python の基本メカニズムを集中的に理解できる利点もあるから，WRS の開発においては，Anaconda より CPython の使用をおすすめです．CPython は https://www.python.org から直接ダウンロードできます．

それでは，WRS を利用する第一歩として，CPython のインストール方法を説明します．https://www.python.org/ から希望のバージョン（3.11 の最新版，64 ビットをおすすめ）をダウンロードしてください．現時点で Open3D という点群処理用のパッケージが 3.11 および以下に依存しているため，3.12 とそれ以上のバージョンのインストールを控えてください．インストールする際，図 1 のように，インストール先のフォルダーを D の下まで修正しておきましょう．あとで Python のインタプリタフォルダーへの頻繁にアクセスする必要があるため，インストール先をアクセスしやすいように D しておきました．インストール先のフォルダは C の Program Files などの浅いかつアクセスしやすいところまで指定しても構いません．

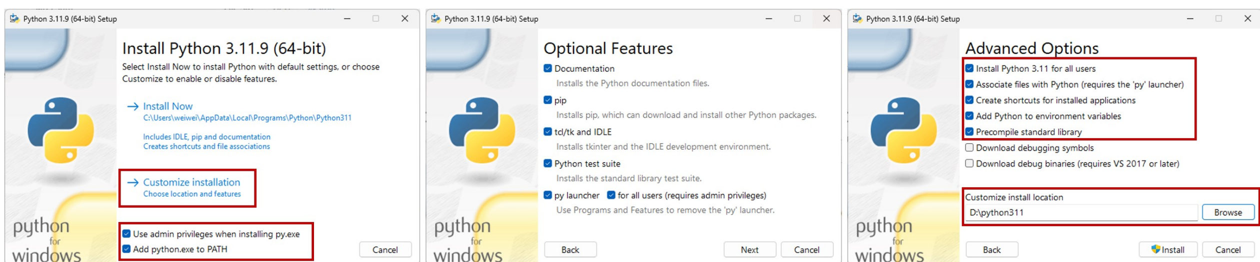

図 1: CPython インストール時に表示されるダイアログボックスと入力方法．インストール先をアクセスしやすいように修正しておきましょう．

正しくインストールされれば，コマンドラインから Python を実行すると次のように表示されます．

```
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr  2 2024, 10:12:12) [MSC v.1938 64 bit (AMD64)] on
  win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

試しに，2 と 3 の足し算を確かめましょう．

```
>>> 2+3
5
```

4　動的型付けと厳密な書式

初学者に一番理解してもらいたい Python の特徴は，動的型付けと厳密な書式です．まず，動的型付けとは，Python が実行中にデータ型を自動的に決定する機能です．Python の変数や関数を定義するとき，C 言語のようなデータ型を指定する必要はありません．例えば，C 言語の整数の定義は “int a =10;” であり，左辺値と右辺値のデータ型を合わせて記述されます．それに対し，Python の整数の定義は「a = 10」で，左側にデータ型の記述は要りません．具体的なデータ型はインタプリターに翻訳される際に動的に付けられます．

次に，厳密な書式について説明します．Python は C 言語のようなプログラムの構造（ブロック）を定義する波括弧を持っていません．代わりに，各行はどのブロックに属するかを判別可能なように，インデントを忘れずに行の前に挿入する必要があります．たとえば，C 言語の繰り返し構文である for 文はインデントとは関係がなく，“ {} ” で区切られれば，コンパイルは通るため実行もできます．“ { ” と “ } ” を照合しながら，プログラムの構造や論理などを表現されます．C 言語の “ {} ” で構造を定義する手法は柔軟性がある一方，プログラムの作成者はそれぞれ違う習慣を持つため，多様式の書式を作成し，可読性の悪いプログラムが出来やすい欠点があります．また，長いプログラムを作成する際には，“ {} ” のペアの照合エラーも頻繁に発生します．デバッグに大きい負荷を掛けます．一方，Python の繰り返し構文（for 文）は次のようの書式になります．改行やインデントが正しく挿入されていないといけません．

```
1    >>> for i in range(10):
2    ...     print(i)
```

下のような形式を認められません．厳しく書式を指定しています．

```
1    >>> for i in range(10): print(i)
```

```
1    >>> for i in range(10):
2    ... print(i)
```

C 言語と比べると，統一的かつ読みやすい記述と考えられます．

5　基本三構造

Python はプログラミング言語として,「順次進行」「条件分岐」「繰り返し」という 3 つの基本構造を持っています.「順次実行」は Python のデフォルトのモードです.「条件分岐」は if; if ... else ...; if ... elif ... else ... などの構文によって構成されます．3.10 以上の Python には match 文も追加され，C 言語の switch 文のように使われると想定されます.「繰り返し」は while; for ... in ... などで作成されます．特に，for は C 言語のような（初期化; 継続条件; 増分処理）ではなく，代わりに，for i in range(10); for element in list_type; for i, element in enumerate(list_type) などの構文になること，気を付けてもらいたいと思います．

```
1    >>> for i in range(2):
2    . . .     print(i)
3    . . .
4    0
5    1
6    >>> for i, element in enumerate(['a', 'b']):
7    . . .     print(i, element)
8    . . .
9    0 a
10   1 b
```

Python では，for ループと zip 関数を組み合わせることで，2 つ以上のリストを同時にループ処理することができます．zip 関数は複数のリストを引数に取り，それらを組み合わせたタプルのイテラブルを返します．これにより，複数のリストを同時に処理することが可能になります．zip 関数は Python に提供くれた便利な組み込み関数の一つであり，その詳細は次の小節に説明します．

```
1    >>> list1 = [1, 2, 3]
2    >>> list2 = ['a', 'b', 'c']
3    >>> for item1, item2 in zip(list1, list2):
4    . . .     print(item1, item2)
5    . . .
```

```
6     1 a
7     2 b
8     3 c
```

6 便利な組み込み関数，標準的なデータ構造，豊富なパッケージが同梱

6.1 組み込み関数

Python には多くの便利な組み込み関数があり，特別なインポートなしに直接利用できます．これには，上記の zip() の他，print()，abs()，map() などの関数が含まれます．

◇ 基本的な操作関数

print()：引数として与えられたオブジェクトをコンソールに出力します．

```
1 >>> print("Hello, world!")
2 Hello, world!
```

input()：ユーザーからの入力を受け取ります．

```
1 >>> name = input("Enter your name: ")
2 >>> print("Hello, " + name + "!")
3 Hello #入力された内容 |
```

len()：オブジェクト（例えばリスト，文字列など）の長さを返します．

```
1 >>> length = len("Python")
2 >>> print(length)
3 6
```

◇ 数値操作関数

abs()：数値の絶対値を返します．

```
1 >>> print(abs(-5))
2 5
```

max(),min()：引数の中で最大あるいは最小の値を返します．

```
1 >>> print(max(1, 3, 2))
2 3
```

sum()：イテラブル内の全ての要素の合計を返します．

```
1 >>> numbers = [1, 2, 3]
2 >>> print(sum(numbers))
3 6
```

◇ 型変換関数

これには，int(), float(), str(), list(), tuple() などの関数が含まれます．C 言語のキャスト関数と似ています．

```
1 >>> print(int(1.1))
2 1
```

◇ その他の便利な関数

zip()：前小節の例を参考ください．

map()：関数をイテラブルの各要素に適用し，その結果を返します．

```
1 >>> def square(x):
2 ...     return x * x
3 ...
4 >>> numbers = [1, 2, 3]
5 >>> squared = map(square, numbers)
6 >>> print(list(squared))
7 [1, 4, 9]
```

filter()：関数を適用して真となる要素のみを含むイテラブルを返します.

```
>>> def is_even(x):
...     return x % 2 == 0
...
>>> numbers = [1, 2, 3, 4, 5]
>>> evens = filter(is_even, numbers)
>>> print(list(evens))
[2, 4]
```

sorted()：イテラブルの要素を並べ替えて新しいリストを返します.

```
>>> numbers = [3, 1, 2]
>>> print(sorted(numbers))
[1, 2, 3]
```

他にも多くの関数がありますが，ここでは省略します.

6.2 データ構造

Python はまた「Ready-to-use」の状態で提供されています．Python に多くの標準的なデータ構造とパッケージが同梱されています．同梱されたデータ構造である List，String，および Dictionary は非常に多くの機能を簡単に実現できる Python の特徴的なデータ構造です．それぞれのデータ構造の機能を簡単に説明していきます.

List：Python のリストは C 言語のデータ構造のキューのようなものです．リストに挿入したりそこからポップしたりすることができます．足し算の符号を使って 2 つのリストを連結することも簡単に実装できます．リストは “ [] ” を使って宣言・定義されます.

```
>>> a = []
>>> a.append(1)
>>> a.append(2)
>>> print(a)
[1, 2]
```

リストの各要素は繰り返し構文でアクセスできます.

```
>>> for i in a:
...     print(i)
...
1
2
3
```

String：C 言語の文字列に似たデータ構造です．C 言語の文字の配列と同じように，文字のリストとして取り扱われます．リストと同じように 2 つの文字列はを足し算の符号で簡単に連結できます．また，文字列内の各文字はリストの各要素と同じよう，繰り返し構文でアクセスできます．文字列の定義ではダブルクォート「”...”」或いはシングルクォート「’...’」のどちらも使用できます.

```
>>> a = "hello"
>>> b = 'world'
>>> print(a+b)
helloworld
```

Dictionary：ハッシュ関数に基づいた抽象データ構造です．キーと値のペアによって構成されます．キーを指定すると値を素早くアクセスできることが特徴です．リストと比べて，辞書のキーのデータ型は int に限定されておらず，文字列やオブジェクトなども可能です．辞書は “ {} ” を使って宣言されます.

```
>>> a = {}
>>> a['book'] = 100
>>> a['pen'] = 200
>>> a['cake'] = .003
>>> for key, value in a.items():
...     print(key, value)
...
book 100
pen 200
cake .003
```

6.3 パッケージ

次に，標準パッケージを紹介します．Python をインストールすると，os，math，time などいくつかの標準パッケージがすぐに使用可能です．C 言語の#include の命令と同じように，Python のプログラム内でパッケージに実装された関数を使うためには，事前にそのパッケージを import で始まる命令文によって，インポートします．例えば，os パッケージをインポートすれば，現在の作業ディレクトリを os.getcwd() によって取得できます．math パッケージをインポートすれば，math.radians() によって度数値をラジアン値に変換できます．time パッケージをインポートすれば，システムが起動してからの総経過時間を time.time() によって取得できます．あるブロックの実行前に time.time() で時刻を取得しておき，実行後に time.time() で時刻を取得しておけば，前後の時間差を 2 つの time.time() の差分によって計算することで，任意のブロックの実行時の経過時間を取得できます．Python 固有のグローバルな関数 dir() を使って各パッケージにどのような関数があるかをリストアップすることもできます．

```
1  >>> import os
2  >>> print(os.getcwd())
3  C:\Users\weiwei
4  >>> import math
5  >>> print(math.radians(60))
6  1.0471975511965876
7  >>> import time
8  >>> tic = time.time()
9  >>> ... #プログラムのブロック
10 >>> toc = time.time()
11 >>> print(toc-tic)
12 3.4580702781677246
13 >>> dir(math)
14 ['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin',
15  'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e',
16  'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum',
17  'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp',
18  'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians',
19  'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

同梱された標準パッケージの他に，よく使用されるサードパーティパッケージも数多く存在します．サードパーティパッケージをインストールおよび削除するには，pip コマンドを用いて「pip install packagename」または「pip uninstall packagename」によって実現できます．例えば，numpy は行列計算用のサードパーティパッケージとして，WRS にて頻繁に利用されます．WRS を使う前にそれをインストールする必要があります．コマンドラインで「pip install numpy」することで，インストールは自動的に完了します．そして,「import numpy」で Python のプログラムにインポートすることで，numpy が提供する様々な行列演算処理の関数が使用できます．例えば，numpy を使って x 軸と z 軸の内積を計算する方法は次の通りです

```
1  >>> import numpy
2  >>> a = numpy.array([0,0,1])
3  >>> b = numpy.array([1,0,0])
4  >>> print(numpy.dot(a,b))
5  0
```

上記のプログラムにおいて，numpy.array() は配列を宣言しています．numpy.dot は 2 つの numpy 配列の内積を計算します．プログラムに示されているように，パッケージで定義されているデータ型と関数にアクセスするには，毎回 numpy と入力する必要があります．このモジュール名（numpy）. 関数名（またはクラス名や変数名）という記述におけるモジュール名を省略できるように，Python では import ... as ... または from ... import *のようなコマンドを用意しています．パッケージの名前を簡略してパッケージ内の関数などにアクセスすることが可能です．それらのコマンドを使うと，上記プログラムは以下のように少し簡略された記述に変更できます．

```
1  >>> import numpy as np
2  >>> a = np.array([0,0,1])
```

或いは

```
1  >>> from numpy import *
2  >>> a = array([0,0,1])
```

ここで，import ... as ... はユーザーがインポートしたパッケージを明記するため，from ... import *よりも好ましいです．from ... import *はすべてのパッケージをインポートするため，メモリの無駄使用や後の段階で命名の

問題を引き起こす可能性が生じます．従って，特段の理由がない限りは，前者のように import ... as ... と省略名を定義するか，from ... import 関数名にて関数名を指定してください．

7　オブジェクト指向

また，Python はオブジェクト指向プログラミングも可能です．オブジェクト指向プログラミングにおいて，プログラム作成者はクラスを使って，抽象データ型を宣言します．クラスで宣言した抽象データ型は，メンバー変数とメンバー関数（クラス内の関数をメソッドと呼びます）を持ちます．オブジェクトとはこの抽象データ型の変数です．つまり，クラスは設計図であり，オブジェクトはその設計図から作られる実体です．たとえば，下記のコードでは，車を表すクラスを作成し，それを元に具体的な車のオブジェクトを作成します．

```
>>> class Car: # クラスの定義
...     def __init__(self, brand, speed):
...         self.brand = brand  # メンバ変数
...         self.speed = speed  # メンバ変数
...
...     def accelerate(self):
...         self.speed += 10  # メンバ関数
...         print(f"The car is now going at {self.speed} km/h")
...
>>> my_car = Car("Toyota", 50) # オブジェクトの作成
>>> print(my_car.brand)
Toyota
>>> my_car.accelerate()
The car is now going at 60 km/h
```

オブジェクト指向プログラミングの利点として，カプセル化，継承，およびポリモーフィズムの 3 つの機能を活用できる点があります．カプセル化は，変数とメソッドの両方が 1 つの単位にまとまることを指します．上記の Car の場合，メンバ変数とメンバ関数はどちらも class Car(object) 構造にまとめられます．

継承は，既存のクラスを基に新しいクラスを作ることです．新しいクラスは親クラスの属性とメソッドを引き継ぎます．また，いくつかの新しく追加したメンバ変数とメンバ関数も追加できます．例えば，電気車は class ElectricCar(Car) で Car から継承して定義できます．ElectricCar，Car 型のすべてのパブリックおよびプライベートメンバー変数とメンバー関数を継承するうえ，charge 関数を新しく追加しています．

```
>>> class Car:  # 親クラス
...     def __init__(self, brand, speed):
...         self.brand = brand
...         self.speed = speed
...
...     def accelerate(self):
...         self.speed += 10
...         print(f"The car is now going at {self.speed} km/h")
...
... class ElectricCar(Car): # 子クラス
...     def __init__(self, brand, speed, battery):
...         super().__init__(brand, speed)  # 親クラスの初期化を呼び出す
...         self.battery = battery  # 新しい属性を追加
...
...     def charge(self):
...         print(f"The battery is now charged to {self.battery} kWh")
...
>>> my_electric_car = ElectricCar("Tesla", 70, 100)
>>> my_electric_car.accelerate()
The car is now going at 80 km/h
>>> my_electric_car.charge()
The battery is now charged to 100 kWh
```

ポリモーフィズムは，親クラスのメソッドを子クラスで再定義（オーバーライド）することです．メンバー関数がどのように機能するかは，属するクラス型によって異なります．たとえは，下記のコードでは，

```
>>> class Car:
...     def accelerate(self):
...         print("The car is accelerating")
...
... class ElectricCar(Car):
```

```
...     def accelerate(self):  # メソッドのオーバーライド
...         print("The electric car is accelerating silently")
...
>>> my_car = Car()
>>> my_electric_car = ElectricCar()
>>> my_car.accelerate()
The car is accelerating
>>> my_electric_car.accelerate()
The electric car is accelerating silently
```

オブジェクト指向プログラミングも WRS にて活用されています．オブジェクト指向は興味深いプログラミングの技術で，なれるまでは時間がかかります．WRS を使って実際にコードを書き，実装や実行を通じてオブジェクト指向の利点を理解していきましょう．