



# Python Programming

# Python Programming

Chapter 0: Course Introduction

# PYTHON

Lecturer: Phan Thị Hà

Email: [hapt@ptit.edu.vn](mailto:hapt@ptit.edu.vn)

Zalo: 0948672246

# **1. What do we learn in this course?**

- Fundamental skills in Python programming.
- Introduce some well-known Python libraries.

# 1. What do we learn in this course?

## - Some Text-Books:

[1]. Eric Matthes. *Python crash course: a hands-on, project-based introduction to programming*, No Starch Press, 2016.

[2]. Allen B. Downey, *Think Python: How to Think Like a Computer Scientist*, O'Reilly, 2015

[3]. Zed A. Shaw, *Learn Python 3 the Hard Way*, Addison-Wesley, 2016

## **2. Why do we learn Python?**

- Mandatory course.
- Prepare for other courses.
- Prepare for the job market.

### **3. How do we learn this course**

- 15 lecture combine with practical exercises.
- 3 practical session.

## **4. Assessment**

- Practical exercises - 10%.
- Assignment 1 - 20%.
- Assignment 2 - 20%.
- Final Exam - 60%.

# Python Programming

Chapter 1: Introduction

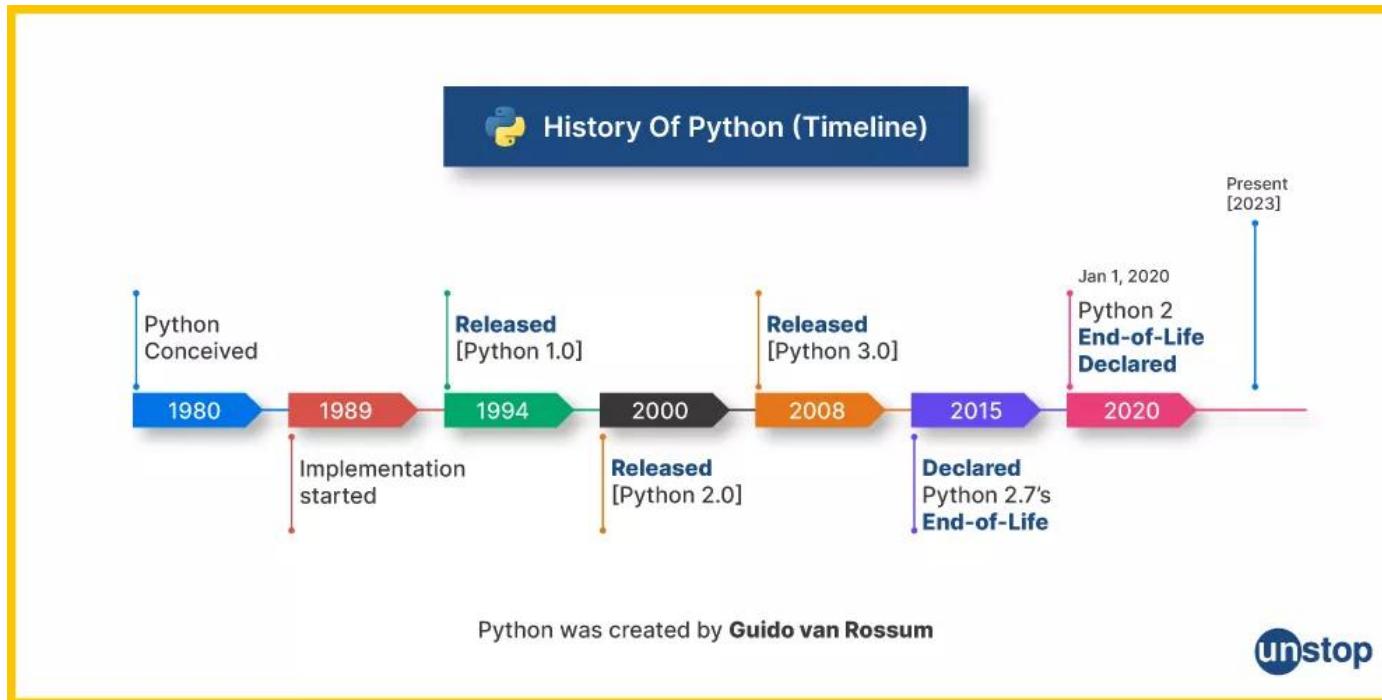
# Agenda

- Introduction
- Python History
- Getting Started
  - Installing Python.
  - Installing text editor.
- Your first Python program: Hello world!

## 1.1 Introduction

- Chapter 1: Introduction
- Chapter 2: Variables and Simple Data Types
- Chapter 3: Python Lists
- Chapter 4: Condition Statements (If else)
- Chapter 5: Dictionaries
- Chapter 6: User Input and While Loop
- Chapter 7: Functions
- Chapter 8: Classes
- Chapter 9: File and exceptions
- Chapter 10: Python Applications

# 1.1 Brief History of Python



[\*] <https://unstop.com/blog/what-is-python>

## 1.2 Getting Started

- Installing Python:

[\*] <https://realpython.com/installing-python/>

- A better way to do things: Using Anaconda to manage environment and package

[\*\*] <https://docs.anaconda.com/free/anaconda/install/index.html>

## 1.3 Getting Started

- Text Editor for Python: Any text editor can do the job.
- Some popular IDE for Python:
  - Pycharm
  - Visual Studio Code
    - Jupiter notebook
      - [\*] <https://www.dataquest.io/blog/jupyter-notebook-tutorial/>

## Khi nào thì nên chọn Jupyter Notebook?

### Trường hợp

- Làm bài tập, bài thực hành đơn giản
- Phân tích dữ liệu nhanh (Data Analysis)
- Học máy (Machine Learning) cơ bản
- Ghi chú, giải thích quá trình làm việc
- Làm báo cáo nhanh, trình bày kết quả

### Giải thích

Giao diện nhẹ, dễ viết code, chạy từng ô (cell) dễ dàng.

Dễ kết hợp **code + bảng biểu + đồ thị** trong cùng một file.

Thử nghiệm mô hình ML nhỏ (vì dễ chạy từng bước, dễ lưu kết quả).

Vừa viết văn bản mô tả (Markdown), vừa chèn code và hình ảnh.

Xuất file PDF hoặc HTML đẹp mắt để nộp bài, trình bày.

Files

Running

Select items to perform actions on them.



/

Name

anaconda\_projects

anaconda3

Contacts

Desktop

▼ New

▲ Upload

C

Python [conda env:base] \*

Terminal

Console

New File

New Folder

# Bạn nên chọn PyCharm nếu:

## Trường hợp

- Dự án Python lớn, nhiều file, nhiều thư mục
- Cần viết code chuyên nghiệp, nhiều tính năng hỗ trợ
- Cần debug, kiểm tra code chi tiết
- Cần quản lý môi trường ảo (virtual environment) dễ dàng
- Làm web app bằng Django, Flask, FastAPI...
- Tích hợp hệ thống quản lý phiên bản Git
- Viết unit test, TDD

## Giải thích

PyCharm quản lý project rất tốt: dễ nhìn cấu trúc file, thư mục.

PyCharm hỗ trợ tự động hoàn thành code (autocomplete), kiểm tra lỗi chính tả (linting), gợi ý sửa lỗi thông minh.

PyCharm có công cụ **debug** rất mạnh: có thể đặt breakpoint, xem biến, theo dõi luồng chương trình.

Tạo, chọn môi trường Python cực kỳ nhanh gọn trong giao diện.

PyCharm có hỗ trợ đặc biệt cho lập trình web bằng Python.

Dùng Git ngay trong PyCharm không cần rời IDE.

PyCharm hỗ trợ chạy test và báo cáo kết quả trực tiếp.

## Bạn chọn VS Code nếu:

### Trường hợp

- Muốn IDE nhẹ, mở nhanh
- Viết nhiều loại ngôn ngữ (Python, JavaScript, HTML, C++,...)
- Làm Python dự án vừa và nhỏ
- Làm web app (HTML/CSS/JS + Python backend)
- Quản lý Git, GitHub ngay trong IDE
- Tuỳ chỉnh giao diện, thêm plugin theo nhu cầu
- Máy tính cấu hình trung bình hoặc yếu

### Giải thích

- VS Code khởi động rất nhanh, nhẹ hơn PyCharm.
- VS Code hỗ trợ đa ngôn ngữ qua extensions.
- VS Code có extension Python mạnh mẽ (formatting, linting, debug).
- Rất tiện khi bạn vừa viết web, vừa viết server code.
- VS Code tích hợp Git rất mượt, không cần mở Terminal riêng.
- Kho extension của VS Code rất phong phú (Jupyter, Docker, Remote SSH,...).
- VS Code chạy tốt với máy RAM thấp (4GB-8GB).

## 1.4 Introduction to Python programming

# First Python program

```
print("Hello World")
```

# 1.4 Introduction to Python programming

- Comments in Python are indicated by a pound sign (#), and anything on the line following the pound sign is ignored by the interpreter.
- End-of-Line terminates a Statement.
- If you'd like a statement to continue to the next line, it is possible to use the "\ " marker to indicate this.
- Semicolon can optionally terminate a statement
- In Python, code blocks are denoted by *indentation*.
- In Python, indented code blocks are always preceded by a colon (:) on the previous line.

# 1.4 Your First Python Program

## 1.4.1 Variables

- Variables are containers for storing data values.
- Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.

```
x = 5  
y = "John"  
print(x)  
print(y)
```

- If you want to specify the data type of a variable, this can be done with casting

```
x = str(3)      # x will be '3'  
y = int(3)      # y will be 3  
z = float(3)    # z will be 3.0
```

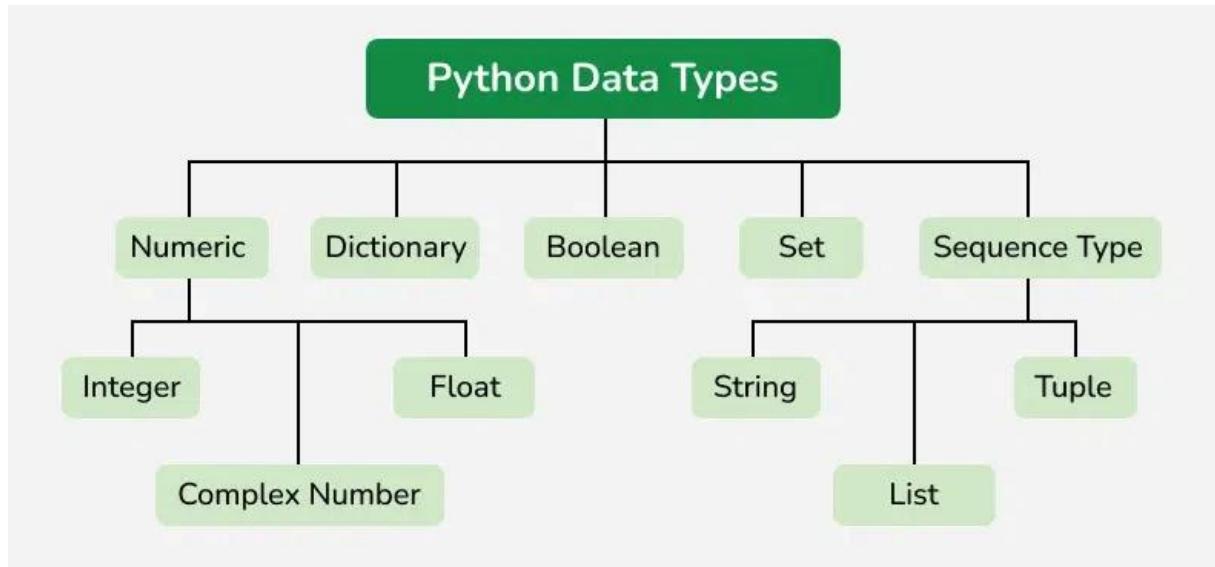
- You can get the data type of a variable with the `type()` function.

```
x = 5  
y = "John"  
print(type(x))  
print(type(y))
```

# 1.4 Your First Python Program

## 1.4.2 Data Types

- Everything is an object in Python programming, Python data types are classes and variables are instances (objects) of these classes.



# 1.4 Your First Python Program

## 1.4.3 Objects and Dynamic Typing

- In object-oriented programming languages like Python, an *object* is an entity that contains data along with associated metadata and/or functionality.

```
L = [1, 2, 3]
L.append(100)
print(L)
```

- In Python everything is an object, which means every entity has some metadata (called *attributes*) and associated functionality (called *methods*).

```
x = 4.5
x.is_integer()
```

- These attributes and methods are accessed via the dot syntax.

```
z = complex(x, 0);
print(z.real, "+", z.imag, "i")
```

```
import numpy as np
print(np.array([1,2,3]))
x=5;
y=6.0;
print(type(x))
print(type(y))
L = [1, 2, 3]
L.append(100)
print(L)
print("result",y.is_integer())
print("result",4.5.is_integer())
```

# 1.4 Your First Python Program

## 1.4.3 Objects and Dynamic Typing

- Python has types; however, the types are linked not to the variable names but *to the objects themselves*.

Trong Python:

- Khi bạn viết `x = 10`, thì:
  - `10` là một **đối tượng kiểu int**
  - `x` chỉ là **tên biến** trỏ tới **đối tượng đó**
- Nếu sau đó bạn viết `x = "hello"`, thì `x` lại trỏ đến một **đối tượng kiểu str**.  
→ Loại (kiểu dữ liệu) **gắn với đối tượng**, không gắn cố định với tên biến như trong một số ngôn ngữ như C hoặc Java.

# Python Programming

Chapter 2: Variables and Simple Data Types

# Agenda

- Variables
- Strings
- Numbers
- Comments

## 2.1 Variables

- What is Python variable?
  - Variables are containers for storing data values.
  - Python has no command for declaring a variable.
  - A variable is created the moment you first assign a value to it.
  - Variables do not need to be declared with any particular *type*, and can even change type after they have been set.
  - If you want to specify the data type of a variable, this can be done with casting.

## 2.1 Variables

```
x = 5           x = 4           # x is of type int
y = "John"       x = "Sally" # x is now of type str
print(x)         print(x)

x = str(3)      # x will be '3'
y = int(3)       # y will be 3
z = float(3)    # z will be 3.0
```

## 2.2 Strings

- Strings in python are surrounded by either single quotation marks, or double quotation marks.

```
a = "Hello"  
print(a)
```

- Strings are list of character

```
x = 'Hello '  
y = 'World'  
print(x, len(x))  
print(y, len(y))  
print(x+y, len(x+y))
```

```
x = '0123456789'  
print(x[0:4])  
  
for x in "banana":  
    print(x)
```

```
a = "Hello"  
print(a)  
x = 'Hello '  
y = 'World'  
print(x, len(x)) # In chuỗi x và độ dài  
print(y, len(y)) # In chuỗi y và độ dài  
print(x + y, len(x + y))  
x = '0123456789'  
print(x[0:4])  
for x in "banana":  
    print(x)
```

Hello  
Hello 6  
World 5  
Hello World 11  
0123  
b  
a  
n  
a

```
text = "hello world"
print(text.upper())
text = "HeLLo WoRLD"
print(text.lower())
text = "banana"
print(text.find("a"))    # Trả về vị trí đầu tiên của 'a'
print(text.find("na"))
```

- Some useful string methods:

- `upper()`: convert string to upper-case
- `lower()`: convert string to lower-case
- `find()`: find a value and return position where it was found
- `join()`: join all the elements of an iterable into a string

```
>>> x = ''.join(['H', 'e', 'l', 'l', 'o'])
>>> print(x)
Hello
>>>
```

**HELLO WORLD**  
**hello world**

**1**  
**2**

```
>>> x = '_'.join(['H', 'e', 'l', 'l', 'o'])
>>> print(x)
H_e_l_l_o
>>>
```

`-strip()`,`lstrip()`,`rstrip()`

- More at [\*] [https://www.w3schools.com/python/python\\_ref\\_string.asp](https://www.w3schools.com/python/python_ref_string.asp)

Dấu f trước một chuỗi trong Python là **f-string** (hay còn gọi là **formatted string literal**) – dùng để **chèn giá trị của biến hoặc biểu thức Python trực tiếp vào chuỗi** một cách tiện lợi và dễ đọc.

```
f"nội_dung {biến_hoặc_biểu_thức} nội_dung_tếp"
```

## 2.3 Numbers

- There are three numeric types in Python:
  - Integer - Float - Complex

```
x = 1      # int
y = 2.8    # float
z = 1j     # complex
```

## 2.4 Comments

- Single Line comment using: #comments
- Multiple Line comment using triple quotation: “” comments“”

```
# Đây là một dòng chú thích  
x = 5 # Gán giá trị 5 cho biến x  
print(x)  
""
```

Đây là một chú thích nhiều dòng.

Có thể ghi chú giải dài.

Không được thực thi khi chạy chương trình.

```
""  
  
x = 10  
print(x)  
Hoặc  
"""
```

Chú thích này cũng hợp lệ.

Có thể dùng để mô tả hàm, đoạn mã, v.v.

```
"""
```

# Python Programming

Chapter 3: Python Data Types - List

## 3.1 What are Lists?

- Lists are used to store multiple items in a single variable.
- Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

```
thislist = ["apple", "banana", "cherry"]  
print(thislist)
```

---

## 3.2 List Items

- List items are ordered, changeable, and allow duplicate values.
- List items are indexed, the first item has index [0], the second item has index [1] etc.
- When we say that lists are ordered, it means that the items have a defined order, and that order will not change.
- The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.
- Since lists are indexed, lists can have items with the same value:

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

## 3.2 List Items

- To determine how many items a list has, use the `len()` function:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

- List items can be of any data type:

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

- A list can contain different data types:

```
list1 = ["abc", 34, True, 40, "male"]
```

- The `list()` Constructor

```
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
print(thislist)
```

```
ist1 = ["apple", "banana", "cherry"] # Kiểu chuỗi (str)
list2 = [1, 5, 7, 9, 3]           # Kiểu số nguyên (int)
list3 = [True, False, False]       # Kiểu Boolean
list1 = ["abc", 34, True, 40, "male"]
print(list1)
thislist = list(("apple", "banana", "cherry")) # Chú ý
dấu ngoặc kép kép ()
print(thislist)
```

## 3.3 Access Lists Items

- List items are indexed and you can access them by referring number:

```
thislist = ["apple", "banana", "cherry"]  
print(thislist[1])
```

- Negative Indexing
  - Negative indexing means start from the end
  - **-1** refers to the last item, **-2** refers to the second last item etc.

```
thislist = ["apple", "banana", "cherry"]  
print(thislist[-1])
```

```
thislist = ["apple", "banana", "cherry"]  
print(len(thislist)) # Kết quả: 3  
list1 = ["apple", "banana", "cherry"] # Kiểu chuỗi  
(str)  
list2 = [1, 5, 7, 9, 3] # Kiểu số nguyên (int)  
list3 = [True, False, False] # Kiểu Boolean  
list1 = ["abc", 34, True, 40, "male"]  
print(list1)  
thislist = list(("apple", "banana", "cherry")) # Chú ý  
dấu ngoặc kép kép ()  
print(thislist, "\n")  
print(thislist[1])  
print(thislist[-1])  
print(thislist[-3])
```

phần tử  
cuối  
cùng

-1

phần tử  
ké cuối

-2

phần tử  
trước  
nữa

-3

## 3.3 Access Lists Items

- Range of Indexes
  - You can specify a range of indexes by specifying where to start and where to end the range.
  - When specifying a range, the return value will be a new list with the specified items.

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])           ['cherry', 'orange', 'kiwi']
print(thislist[:4])           ['apple', 'banana', 'cherry', 'orange']
print(thislist[2:])           ['cherry', 'orange', 'kiwi', 'melon', 'mango']
print(thislist[-4:-1])        ['orange', 'kiwi', 'melon']
```

## 3.3 Access Lists Items

- Check if Item Exists
  - To determine if a specified item is present in a list use the in keyword:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
    print("Yes, 'apple' is in the fruits list")
```

- list1 = ['ha noi', "mua nay", “vang”]
- if "ha noi" in list1:
  - print("oki")
- else:
  - print("not OKI")

### 3.3 Change list items

- To change the value of a specific item, refer to the index number:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

- To change the value of items within a specific range, change the values, and refer to the range of index numbers to insert new values:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)

thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist)
```

```
# Thay đổi một phần tử cụ thể bằng chỉ số
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print("Ví dụ 1:", thislist) # ['apple', 'blackcurrant', 'cherry']

# Thay đổi nhiều phần tử trong một đoạn (slice)
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print("Ví dụ 2:", thislist) # ['apple', 'blackcurrant', 'watermelon',
'orange', 'kiwi', 'mango']

# Thay đổi 1 phần tử (nhưng dùng slice thay vì chỉ số trực tiếp)
thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
print("Ví dụ 3:", thislist) # ['apple', 'blackcurrant', 'watermelon',
'cherry']

# Gán 1 phần tử mới cho 2 phần tử cũ (thu hẹp list)
thislist = ["apple", "banana", "cherry"]
thislist[1:3] = ["watermelon"]
print("Ví dụ 4:", thislist) # ['apple', 'watermelon']
```

## 3.4 Insert List Items

- To insert a new list item, without replacing any of the existing values, we can use the `insert()` method.
- The `insert()` method inserts an item at the specified index:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

## 3.5 Add List Items

- To add an item to the end of the list, use the `append()` method:

```
thislist = ["apple", "banana", "cherry"]  
thislist.append("orange")  
print(thislist)
```

```
thislist = ["apple", "banana", "cherry"]  
thislist += ["orange"]  
print(thislist)
```

- To append elements from *another list* to the current list, use the `extend()` method.

```
thislist = ["apple", "banana", "cherry"]  
tropical = ["mango", "pineapple", "papaya"]  
thislist.extend(tropical)  
print(thislist)
```

```
thislist1 = ["apple"]  
thislist2 = thislist1*3  
print(thislist2)
```

['apple', 'apple', 'apple']

## 3.6 Remove List Items

- The **remove()** method removes the specified item.
- If there are more than one item with the specified value, the **remove()** method removes the first occurrence.
- The **pop()** method removes the specified index.
- If you do not specify the index, the **pop()** method removes the last item.

```
fruits = ["apple", "banana",
"cherry", "banana"]
fruits.remove("banana")
print(fruits)
fruits = ["apple", "banana",
"cherry"]
fruits.pop(1)
print(fruits)
['apple', 'cherry', 'banana']
['apple', 'cherry']
```

## 3.7 Loop through a list

- You can loop through the list items by using a for loop:

```
thislist = ["apple", "banana", "cherry"]  
for x in thislist:  
    print(x)
```

- You can also loop through the list items by referring to their index number. Use

```
thislist = ["apple", "banana", "cherry"]  
for i in range(len(thislist)):  
    print(thislist[i])
```

## 3.7 Loop through a list

- You can loop through the list items by using a while loop. Use the `len()` function to determine the length of the list, then start at `0` and loop your way through the list items by referring to their indexes. Remember to increase the index by `1` after each iteration.

```
thislist = ["apple", "banana", "cherry"]  
i = 0  
while i < len(thislist):  
    print(thislist[i])  
    i = i + 1
```

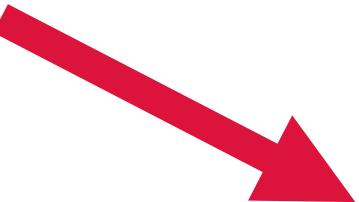
## 3.8 List comprehension

- List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
    if "a" in x:
        newlist.append(x)

print(newlist)
```



```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []
for x in fruits:
    if "a" in x:
        newlist.append(x)

print(newlist)
# Expected output: ['apple', 'banana', 'mango']

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x for x in fruits if "a" in x]
print(newlist)
```

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

## 3.8 List comprehension

### The Syntax

```
newlist = [expression for item in iterable if condition == True]
```

[f(x) if condition else g(x) for x in iterables]

Or

[expression\_1 if condition else expression\_2 for x in iterables]

# 3.8 List comprehension

```
loopTheList.py > ...
1  from time import time
2
3  N = 1_000_000
4  even_numbers = []
5  odds_numbers = []
6
7  startLoopTime = time()
8
9  for x in range(N):
10     if(x%2 == 0): even_numbers += [x]
11     else: odds_numbers += [x]
12
13 endLoopTime = time()
14
15 print('Run Time:{:.3f}'.format(endLoopTime - startLoopTime))
```

# 3.8 List comprehension

```
comprehension.py > ...
1  from time import time
2
3  N = 1_000_000
4  even_numbers = []
5  odds_numbers = []
6
7  startLoopTime = time()
8
9  even_numbers = [x for x in range(N) if x%2 == 0]
10 odds_numbers = [x for x in range(N) if x%2 == 1]
11
12 endLoopTime = time()
13
14 print('Run Time:{:.3f}'.format(endLoopTime - startLoopTime))
```

# 3.8 List comprehension

```
comprehension2.py > ...
1  from time import time
2
3  N = 1_000_000
4  even_numbers = []
5  odds_numbers = []
6
7  startLoopTime = time()
8
9  [even_numbers.append(x) if x%2 == 0 else odds_numbers.append(x) for x in range(N)]
10
11 endLoopTime = time()
12
13 print('Run Time:{:.3f}'.format(endLoopTime - startLoopTime))
```

## 3.9 Sort list

- List objects have a `sort()` method that will sort the list alphanumerically, ascending, by default:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

- To sort descending, use the keyword argument `reverse = True`:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```

## 3.10 Copy list

- You cannot copy a list simply by typing `list2 = list1`, because: `list2` will only be a *reference* to `list1`, and changes made in `list1` will automatically also be made in `list2`

```
thislist1 = ["apple", "banana", "cherry"]
thislist2 = thislist1
thislist1[:] = [1,2,3]
print(thislist2)
```

- You can use the built-in List method `copy()` to copy a list.

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

```
thislist1 = ["apple", "banana", "cherry"]
thislist2 = thislist1
thislist1[0] = "orange" # Thay đổi phần tử đầu tiên
                      # của thislist1
print(thislist2)
# Expected output: ["orange", "banana", "cherry"]
# (thislist2 cũng thay đổi theo vì nó tham chiếu đến
# cùng một đối tượng)
```

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy() # Creates a true copy of
                        # thelist

# Bây giờ, mylist là một danh sách độc lập.
# Các thay đổi trên thislist sẽ không ảnh hưởng
# đến mylist.
thislist[0] = "orange" # Ví dụ thay đổi thislist

print(mylist)
```

## 3.10 Copy list

- You can use the built-in List method `copy()` to copy a list.

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

- Another way to make a copy is to use the built-in method `list()`.

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

- Use the slice Operator

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist[:]
print(mylist)
```

<b>Method</b>	<b>Description</b>
<u>append()</u>	Adds an element at the end of the list
<u>clear()</u>	Removes all the elements from the list
<u>copy()</u>	Returns a copy of the list
<u>count()</u>	Returns the number of elements with the specified value
<u>extend()</u>	Add the elements of a list (or any iterable), to the end of the current list
<u>index()</u>	Returns the index of the first element with the specified value
<u>insert()</u>	Adds an element at the specified position
<u>pop()</u>	Removes the element at the specified position
<u>remove()</u>	Removes the item with the specified value
<u>reverse()</u>	Reverses the order of the list
<u>sort()</u>	Sorts the list

```
firstName = ["Steven", "Frank", "Luis", "Andy"]
```

```
lastName = ["G.", "L.", "S.", "C."]
```

Print to console:

1. Steven G.
2. Frank L.
3. Luis S.
4. Andy C.

# Python Dictionary

# 1. What is dictionary

- Dictionaries are used to store data values in key:value pairs.
- A dictionary is a collection which is ordered\*, changeable and do not allow duplicates.

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
print(thisdict)
```

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964,  
    "year": 2020  
}  
print(thisdict)
```

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
print(thisdict)
```

## 2. Access Dictionary Items

- We can access the items of a dictionary by referring to its key name, inside square brackets.
- There is also a method called `get()` that will give the same result.

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
print(thisdict["brand"])  
  
x = thisdict.get("model")
```

- The `keys()` method will return a list of all the keys in the dictionary.

```
x = thisdict.keys()
```

### 3. Change/Add Dictionary Items

- We can change the value of a specific item by referring to its key name.
- The `update()` method will update the dictionary with the items from the given argument. The argument must be a dictionary, or an iterable object with **key:value** pairs.

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
  
thisdict["year"] = 2018
```

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
  
thisdict.update({"year": 2020})
```

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict["year"] = 2018  
print(thisdict)  
thisdict.update({"mod  
el": "new"})  
print(thisdict)
```

### 3. Remove Dictionary Items

- The `pop()` method removes the item with the specified key name:
- The `popitem()` method removes the last inserted item (in versions before 3.7, a random item is removed instead).
- The `clear()` method empties the dictionary.

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict.pop("model")  
print(thisdict)
```

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict.popitem()  
print(thisdict)
```

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict.clear()  
print(thisdict)
```

# 4. Loop through Dictionary

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
for x in thisdict:  
    print(x)
```

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
for x in thisdict.values():  
    print(x)
```

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
for x in thisdict:  
    print(x)  
  
for x in thisdict.values():  
    print(x,end=' ')  
print()  
for x in thisdict.keys():  
    print(x,end=' ')  
print()  
for x in thisdict.items():  
    print(x,end=' ')  
print()  
for x,y in thisdict.items():  
    print(x,y)
```

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
for x in thisdict.keys():  
    print(thisdict[x])  
  
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
for x, y in thisdict.items():  
    print(x, y)
```

# 5. Copy Dictionary

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
mydict = thisdict.copy()  
print(mydict)
```

- Make a copy of a dictionary with the `copy()` method:
- Another way to make a copy is to use the built-in function `dict()`.

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
mydict = thisdict.copy()  
print(mydict)
```

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
mydict = dict(thisdict)  
print(mydict)
```

# 6. Nested Dictionary

```
myfamily = {  
    "child1" : {  
        "name" : "Emil",  
        "year" : 2004  
    },  
    "child2" : {  
        "name" : "Tobias",  
        "year" : 2007  
    },  
    "child3" : {  
        "name" : "Linus",  
        "year" : 2011  
    }  
}
```

```
child1 = {  
    "name" : "Emil",  
    "year" : 2004  
}  
child2 = {  
    "name" : "Tobias",  
    "year" : 2007  
}  
child3 = {  
    "name" : "Linus",  
    "year" : 2011  
}  
  
myfamily = {  
    "child1" : child1,  
    "child2" : child2,  
    "child3" : child3  
}
```

```
print(myfamily["child2"]["name"])  
  
for x, obj in myfamily.items():  
    print(x)  
  
    for y in obj:  
        print(y + ':', obj[y])
```

```
myfamily = {
    "child1" : {
        "name" : "Emil",
        "year" : 2004
    },
    "child2" : {
        "name" : "Tobias",
        "year" : 2007
    },
    "child3" : {
        "name" : "Linus",
        "year" : 2011
    }
}

print(myfamily["child1"]["name"])

for x in myfamily:
    print(myfamily[x])
for x in myfamily:
    print(myfamily[x]["name"])
print("Kết quả tung tự")
for x in myfamily.values():
    print(x["name"])
print("1")
for x,obj in myfamily.items():
    print(x)
print("2")
for x,obj in myfamily.items():
    print(x,obj)
print("3")
for y in obj:
    print(y)
```

Emil  
{'name': 'Emil', 'year': 2004}  
{'name': 'Tobias', 'year': 2007}  
{'name': 'Linus', 'year': 2011}  
Emil  
Tobias  
Linus  
Kết quả tung tự  
Emil  
Tobias  
Linus  
1  
child1  
child2  
child3  
2  
child1 {'name': 'Emil', 'year': 2004}  
child2 {'name': 'Tobias', 'year': 2007}  
child3 {'name': 'Linus', 'year': 2011}  
3  
name  
year

Method	Description
<a href="#"><u>clear()</u></a>	Removes all the elements from the dictionary
<a href="#"><u>copy()</u></a>	Returns a copy of the dictionary
<a href="#"><u>fromkeys()</u></a>	Returns a dictionary with the specified keys and value
<a href="#"><u>get()</u></a>	Returns the value of the specified key
<a href="#"><u>items()</u></a>	Returns a list containing a tuple for each key value pair
<a href="#"><u>keys()</u></a>	Returns a list containing the dictionary's keys
<a href="#"><u>pop()</u></a>	Removes the element with the specified key
<a href="#"><u>popitem()</u></a>	Removes the last inserted key-value pair
<a href="#"><u>setdefault()</u></a>	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
<a href="#"><u>update()</u></a>	Updates the dictionary with the specified key-value pairs
<a href="#"><u>values()</u></a>	Returns a list of all the values in the dictionary

# Python Programming

Chapter 4: Python Function

## 4.1 Define a Function

- Using “def” keyword
- Function Name
- Input Argument

## 4.2 Input Arguments

- Immutable Arguments:
  - Numeric
  - String
  - Tuples

```
Ví dụ 1:  
--- Thông tin Mặt hàng ---  
Tên: Bàn phím cơ  
Giá: $99.50  
Nhãn: gaming, phụ kiện máy tính, có dây  
-----
```

```
Ví dụ 2:  
--- Thông tin Mặt hàng ---  
Tên: Chuột không dây  
Giá: $25.00  
Nhãn: phụ kiện máy tính  
-----|
```

```
Ví dụ 3:  
--- Thông tin Mặt hàng ---  
Tên: Màn hình cong 27 inch  
Giá: $299.99
```

```
def display_item_info(item_name: str, item_price: float, tags: tuple):  
    print(f"--- Thông tin Mặt hàng ---")  
    print(f"Tên: {item_name}")  
    print(f"Giá: ${item_price:.2f}")  
    print(f"Nhãn: {' '.join(tags) if tags else 'Không có'}")  
    print(f"-----")  
# Ví dụ 1: Một mặt hàng thông thường  
print("Ví dụ 1:")  
display_item_info("Bàn phím cơ", 99.50, ("gaming", "phụ kiện máy tính", "có dây"))  
# Ví dụ 2: Mặt hàng với ít nhãn hơn  
print("\nVí dụ 2:")  
display_item_info("Chuột không dây", 25.00, ("phụ kiện máy tính",))  
# Tuple có một phần tử  
# Ví dụ 3: Mặt hàng không có nhãn  
print("\nVí dụ 3:")  
display_item_info("Màn hình cong 27 inch", 299.99, ()) # Tuple rỗng
```

- Mutable Arguments:
  - List
  - Dictionary

```
[1, 2, 3, 4]
{'name': 'Huy', 'city': 'Hanoi'}
1
2
3
4
name
city
('name', 'Huy')
('city', 'Hanoi')
```

```
def modify_collections(data_list: list, data_dict: dict):
    data_list.append(4)
    data_dict["city"] = "Hanoi"

my_nums = [1, 2, 3]
my_info = {"name": "Huy"}

modify_collections(my_nums, my_info)

print(my_nums) # Output: [1, 2, 3, 4]
print(my_info) # Output: {'name': 'Huy', 'city': 'Hanoi'}
for x in range (len(my_nums)):
    print(my_nums[x])
for k in my_info:
    print(k)
for item in my_info.items():
    print(item)
```

## 4.2 Input Arguments

- A Function can take multiple arguments.

```
def greet_person(name, age, city):  
    print(f"Xin chào, tôi là {name}, {age} tuổi và  
đến từ {city}.")  
# Gọi hàm với nhiều đối số  
greet_person("An", 25, "Hà Nội")  
# Kết quả: Xin chào, tôi là An, 25 tuổi và đến từ  
Hà Nội.
```

- Positional Arguments:
- Keyword Arguments:

```
def display_product_details(product_id, price, quantity):  
    print(f"Mã sản phẩm: {product_id}")  
    print(f"Giá: ${price:.2f}")  
    print(f"Số lượng: {quantity}")  
# Sử dụng đối số từ khóa, thứ tự không quan trọng  
display_product_details("ABC001",99.99,10)  
display_product_details(product_id="ABC001",  
quantity=10, price=99.99)  
# Kết quả:  
# Mã sản phẩm: ABC001  
# Giá: $99.99  
# Số lượng: 10  
# Mã sản phẩm: ABC001  
# Giá: $99.99  
# Số lượng: 10
```

# 4.3 Function as an Argument

- Function is first class object.

```
# 1. Gán hàm cho một biến ✓ 6
def greet(name): 1 usage
    return f"Xin chào, {name}!"

my_greeting_function = greet # Gán hàm 'greet' cho biến 'my_greeting_function'
print(my_greeting_function("Minh"))
# Output: Xin chào, Minh!

# 2. Lưu trữ hàm trong một danh sách
def add(a, b): return a + b 1 usage
def subtract(a, b): return a - b 1 usage
operations = [add, subtract] # Danh sách chứa các hàm
print(operations[0](a: 5, b: 3)) # Gọi hàm 'add' từ danh sách
# Output: 8
```

- Function can be pass as an argument to another function

```
# Hàm "xử lý" (higher-order function)
def process_numbers(operation, num1, num2): 2 usages
    """
    Nhận một hàm 'operation' và áp dụng nó lên num1 và num2.
    """
    print(f"Đang thực hiện phép toán: {operation.__name__}") # Lấy tên hàm
    result = operation(num1, num2)
    return result

# Các hàm đơn giản để truyền vào
def add(a, b): 1 usage
    return a + b
def subtract(a, b): 1 usage
    return a - b

# Truyền hàm 'add' làm đối số cho 'process_numbers'
sum_result = process_numbers(add, num1: 20, num2: 5)
print(f"Kết quả cộng: {sum_result}")

# Truyền hàm 'subtract' làm đối số cho 'process_numbers'
diff_result = process_numbers(subtract, num1: 20, num2: 5)
print(f"Kết quả trừ: {diff_result}")

# Output:
# Đang thực hiện phép toán: subtract
# Kết quả trừ: 15
```

## ● Lambda Function: Anonymous Function

```
# 1. Lambda đơn giản (thay thế một hàm def nhỏ)
# Thay vì:
# def multiply_by_two(x):
#     return x * 2
# Sử dụng lambda:
multiply_by_two_lambda = lambda x: x *
2
print(f"Nhân 5 với 2:
{multiply_by_two_lambda(5)}")
# Output: Nhân 5 với 2: 10
```

2. Sử dụng lambda trong các hàm bậc cao hơn (rất phổ biến)

```
# Ví dụ với hàm sorted() để sắp xếp danh sách các dictionary theo một khóa cụ thể
students = [
    {'name': 'Nam', 'age': 22, 'score': 85},
    {'name': 'Lan', 'age': 20, 'score': 92},
    {'name': 'Tuan', 'age': 23, 'score': 78}
]
```

```
# Sắp xếp học sinh theo 'score' (điểm)
# Hàm key sẽ nhận một dictionary và trả về giá trị của khóa 'score'
sorted_by_score = sorted(students, key=lambda student:
student['score'])
print("Sắp xếp theo điểm:")
for s in sorted_by_score:
    print(s)
# Output:
# Sắp xếp theo điểm:
# {'name': 'Tuan', 'age': 23, 'score': 78}
# {'name': 'Nam', 'age': 22, 'score': 85}
# {'name': 'Lan', 'age': 20, 'score': 92}

# Sắp xếp học sinh theo 'age' (tuổi)
sorted_by_age = sorted(students, key=lambda student: student['age'])
print("\nSắp xếp theo tuổi:")
for s in sorted_by_age:
    print(s)
```

## 4.4 Python Closures

- A function nested in a function
- Useful for:
  - Function Factories
  - Callbacks: event-driven programming
  - Wrapper Function
  - Caching

```
def make_multiplier(factor):
    # Đây là hàm bên trong (nested
    function)
    def multiplier(number):
        return number * factor

    return multiplier # Trả về hàm bên
    trong, không gọi nó!
double_it = make_multiplier(2)
triple_it = make_multiplier(3)
print(f"Nhân đôi 5: {double_it(5)}")  #
Output: Nhân đôi 5: 10
print(f"Nhân đôi 10: {double_it(10)}") #
Output: Nhân đôi 10: 20

print(f"Nhân ba 5: {triple_it(5)}")   #
Output: Nhân ba 5: 15
print(f"Nhân ba 10: {triple_it(10)}") #
Output: Nhân ba 10: 30
```

## 4.5 Generator

- Generators are special Python function that return iterator.

- Generators are lazy.

- Only compute on demand

- Generator helps to save memory and cpu in many cases.

```
def count_up_to(n):
    i = 1
    while i <= n:
        yield i # dùng yield thay vì return
        i += 1
gen = count_up_to(3)
print(gen)
print(next(gen))
print(next(gen))
print(next(gen))
```

# Python Programming

Chapter 5: Python OOPs Concepts

## OOPs Concepts in Python

- Class in Python
- Objects in Python
- Polymorphism in Python
- Encapsulation in Python
- Inheritance in Python
- Data Abstraction in Python

# 1. Python Class

“A class is a collection of objects. **Classes** are blueprints for creating objects. A class defines a set of attributes and methods that the created **objects (instances)** can have.”

## Some points on Python class:

- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator.

## 2. Python Objects

An Object is an instance of a Class. It represents a specific implementation of the class and holds its own data.

An object consists of:

- **State:** It is represented by the attributes and reflects the properties of an object.
- **Behavior:** It is represented by the methods of an object and reflects the response of an object to other objects.
- **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

## 2. Python Objects

### Self Parameter

`self` parameter is a reference to the current instance of the class. It allows us to access the attributes and methods of the object.

## 2. Python Objects

### **\_\_init\_\_ Method**

**\_\_init\_\_** method is the constructor in Python, automatically called when a new object is created. It initializes the attributes of the class.

### **\_\_str\_\_ / \_\_repr\_\_ Method**

Return a string representation for an object

# 3. Class and Instance Variables

In Python, variables defined in a class can be either class variables or instance variables, and understanding the distinction between them is crucial for object-oriented programming.

## Class Variables

These are the variables that are shared across all instances of a class. It is defined at the class level, outside any methods. All objects of the class share the same value for a class variable unless explicitly overridden in an object.

## Instance Variables

Variables that are unique to each instance (object) of a class. These are defined within the `__init__` method or other instance methods. Each object maintains its own copy of instance variables, independent of other objects.

# 4. Python Inheritance

Inheritance allows a class (child class) to acquire properties and methods of another class (parent class). It supports hierarchical classification and promotes code reuse.

## Types of Inheritance:

1. **Single Inheritance:** A child class inherits from a single parent class.
2. **Multiple Inheritance:** A child class inherits from more than one parent class.
3. **Multilevel Inheritance:** A child class inherits from a parent class, which in turn inherits from another class.
4. **Hierarchical Inheritance:** Multiple child classes inherit from a single parent class.
5. **Hybrid Inheritance:** A combination of two or more types of inheritance.

## 4. Python Inheritance

**super()** is used to call methods of a superclass (parent class) from a subclass (child class). It returns a proxy object that delegates method calls to the superclass. This is useful for accessing inherited methods that have been overridden in a subclass.

```
class ChildClass(ParentClass):
    def __init__(self, arg1, arg2):
        super().__init__(arg1) # Calls the __init__() method of the ParentClass
        self.arg2 = arg2
```

## 5. Polymorphism

Polymorphism allows methods to have the same name but behave differently based on the object's context. It can be achieved through method overriding or overloading.

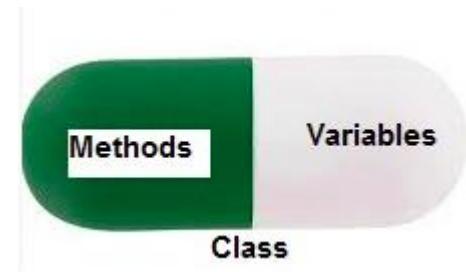
# 6. Encapsulation

Encapsulation is the bundling of data (attributes) and methods (functions) within a class, restricting access to some components to control interactions.

A class is an example of encapsulation as it encapsulates all the data that is member functions, variables, etc.

## Types of Encapsulation:

1. **Public Members:** Accessible from anywhere.
2. **Protected Members:** Accessible within the class and its subclasses.
3. **Private Members:** Accessible only within the class.



# 7. Data Abstraction

**Abstraction** hides the internal implementation details while exposing only the necessary functionality. It helps focus on “what to do” rather than “how to do it.”

## Types of Abstraction:

- **Partial Abstraction:** Abstract class contains both abstract and concrete methods.
- **Full Abstraction:** Abstract class contains only abstract methods (like interfaces).

# Cú pháp tạo ra lớp: vd\_class.py

```
class TenLop:  
    # Thuộc tính của lớp (biến lớp)  
    bien_lop = "Đây là biến lớp"  
  
    # Hàm khởi tạo (constructor) - được gọi khi tạo đối tượng mới  
    def __init__(self, tham_so1, tham_so2):  
        self.tham_so1 = tham_so1 # Thuộc tính của đối tượng (biến instance)  
        self.tham_so2 = tham_so2  
  
    # Phương thức của lớp (hàm thành viên)  
    def phuong_thuc_1(self):  
        print(f"Đây là phương thức 1 của đối tượng với tham số 1: {self.tham_so1}")  
  
    # Phương thức lớp (classmethod) - hoạt động trên lớp chứ không phải đối tượng  
    @classmethod  
    def phuong_thuc_lop(cls):  
        print(f"Đây là phương thức lớp. Biến lớp: {cls.bien_lop}")  
  
    # Phương thức tĩnh (staticmethod) - không truy cập thuộc tính của lớp hay đối tượng  
    @staticmethod  
    def phuong_thuc_tinh(tin_nhan):  
        print(f"Đây là phương thức tĩnh: {tin_nhan}")
```

--- Ví dụ sử dụng lớp ---

```
# Tạo đối tượng từ lớp
doi_tuong1 = TenLop("giá trị 1", "giá trị 2")

# Truy cập thuộc tính của đối tượng
print(f"Tham số 1 của đối tượng 1: {doi_tuong1.tham_so1}")

# Gọi phương thức của đối tượng
doi_tuong1.phuong_thuc_1()
doi_tuong1.phuong_thuc_2("Hello từ phương thức 2")

# Truy cập thuộc tính lớp (qua lớp hoặc đối tượng)
print(f"Biến lớp (qua lớp): {TenLop.bien_lop}")
print(f"Biến lớp (qua đối tượng): {doi_tuong1.bien_lop}")

# Gọi phương thức lớp
TenLop.phuong_thuc_lop()

# Gọi phương thức tĩnh
TenLop.phuong_thuc_tinh("Một thông điệp từ phương thức tĩnh")
```

# **Python Programming**

**Python Applications**

# I. Working with file

- The key function for working with files in Python is the `open()` function.
- The `open()` function takes two parameters; *filename*, and *mode*.
- There are four different methods (modes) for opening a file:

`"r"` - Read - Default value. Opens a file for reading, error if the file does not exist

`"a"` - Append - Opens a file for appending, creates the file if it does not exist

`"w"` - Write - Opens a file for writing, creates the file if it does not exist

`"x"` - Create - Creates the specified file, returns an error if the file exists

- In addition you can specify if the file should be handled as binary or text mode

`"t"` - Text - Default value. Text mode

`"b"` - Binary - Binary mode (e.g. images)

# I. Working with file

- The `open()` function returns a file object, which has a `read()` method for reading the content of the file:

```
f = open("demofile.txt", "r")  
  
print(f.read())
```

- By default the `read()` method returns the whole text, but you can also specify how many characters you want to return:

```
f = open("demofile.txt", "r")  
  
print(f.read(5))
```

- You can return one line by using the `readline()` method:

```
f = open("demofile.txt", "r")  
  
print(f.readline())
```

# I. Working with file

- By looping through the lines of the file, you can read the whole file, line by line:

```
f = open("demofile.txt", "r")
```

```
for x in f:
```

```
    print(x)
```

- It is a good practice to always close the file when you are done with it.

```
f = open("demofile.txt", "r")
```

```
print(f.readline())
```

```
f.close()
```

# I. Working with file

- Best practice is using **Context Manager**:

```
with open("demofile.txt", "r") as f:  
    for x in f:  
        print(x)
```

# I. Working with file

- To write to an existing file, you must add a parameter to the `open()` function:

`"a"` - Append - will append to the end of the file

`"w"` - Write - will overwrite any existing content

```
f = open("demofile2.txt", "a")
f.write("Now the file has more
content!")
f.close()
```

```
f = open("demofile2.txt", "r")
print(f.read())
```

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the
content!")
f.close()
```

```
f = open("demofile3.txt", "r")
print(f.read())
```

# I. Working with file

- To create a new file in Python, use the `open()` method, with one of the following parameters:

`"x"` - Create - will create a file, returns an error if the file exists

`"a"` - Append - will create a file if the specified file does not exists

`"w"` - Write - will create a file if the specified file does not exists

```
f = open("myfile.txt", "w")
```

```
f = open("myfile.txt", "x")
```

# I. Working with file

- To delete a file, you must import the OS module, and run its `os.remove()` function:

```
import os  
  
os.remove("demofile.txt")
```

- To avoid getting an error, you might want to check if the file exists before you try to delete it:

```
import os  
  
if os.path.exists("demofile.txt"):  
    os.remove("demofile.txt")  
  
else:  
  
    print("The file does not exist")
```

- To delete an entire folder, use the `os.rmdir()` method:

```
import os  
  
os.rmdir("myfolder")
```

## II. Requests

- Make a request to a web page, and print the response text:

```
import requests

x = requests.get('https://w3schools.com/python/demopage.htm')
#print(x.text)
print(x.content)
```

- The `requests` module allows you to send HTTP requests using Python.
- The HTTP request returns a Response Object with all the response data (content, encoding, status, etc).

# II. Requests

Syntax: `requests.methodname(params)`

Methods:

<code>delete(url, args)</code>	Sends a DELETE request to the specified url
<code>get(url, params, args)</code>	Sends a GET request to the specified url
<code>head(url, args)</code>	Sends a HEAD request to the specified url
<code>patch(url, data, args)</code>	Sends a PATCH request to the specified url
<code>post(url, data, json, args)</code>	Sends a POST request to the specified url
<code>put(url, data, args)</code>	Sends a PUT request to the specified url
<code>request(method, url, args)</code>	Sends a request of the specified method to the specified url

# III. BeautifulSoup

- BeautifulSoup is used to parse HTML and XML in python.

```
import requests  
  
from bs4 import BeautifulSoup  
  
URL = "https://realpython.github.io/fake-jobs/"  
  
page = requests.get(URL)  
  
soup = BeautifulSoup(page.content, "html.parser")
```

# Pandas

- **Pandas** là một **thư viện mã nguồn mở** trong Python, chuyên dùng để **phân tích và xử lý dữ liệu**. Tên "pandas" xuất phát từ cụm từ "**panel data**" – thuật ngữ trong kinh tế học dùng để chỉ tập dữ liệu có nhiều chiều. Pandas hỗ trợ rất nhiều công việc liên quan đến dữ liệu, đặc biệt là **dữ liệu dạng bảng** (giống như Excel hay bảng trong cơ sở dữ liệu).

Tính năng	Mô tả ngắn gọn
 Đọc dữ liệu	Đọc dữ liệu từ các định dạng như CSV, Excel, SQL, JSON,...
 Biểu diễn dữ liệu	Dùng <b>DataFrame</b> (bảng 2 chiều) và <b>Series</b> (một cột) để lưu trữ dữ liệu
 Làm sạch dữ liệu	Xử lý giá trị thiếu, định dạng sai, loại bỏ dữ liệu trùng lặp,...
 Truy cập và chỉnh sửa dữ liệu	Lọc, cắt, sắp xếp, đổi tên cột/dòng, tính toán theo nhóm,...
 Tính toán thống kê	Tính trung bình, tổng, max, min, đếm, phân tách (groupby),...
 Biến đổi dữ liệu	Gộp cột, tách cột, chuyển đổi định dạng ngày giờ, pivot,...
 Xuất dữ liệu	Ghi dữ liệu ra file CSV, Excel, JSON,...

# IV. Pandas

- Load a CSV file into a Pandas DataFrame:

```
import pandas as pd  
  
df = pd.read_csv('data.csv')  
  
print(df.to_string())
```

# IV. Pandas

- A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

```
import pandas as pd

data = {
    "calories": [420, 380, 390],
    "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df)
```

# IV. Pandas

- Pandas use the `loc` attribute to return one or more specified row(s)

```
#refer to the row index:
```

```
print(df.loc[0])
```

```
#use a list of indexes:
```

```
print(df.loc[[0, 1]])
```

# IV. Pandas

- With the `index` argument, you can name your own indexes.

```
import pandas as pd

data = {
    "calories": [420, 380, 390],
    "duration": [50, 40, 45]
}

df = pd.DataFrame(data, index = ["day1", "day2", "day3"])

print(df)
```

- Use the named index in the `loc` attribute to return the specified row(s).

```
#refer to the named index:

print(df.loc["day2"])
```

## Matplotlib

- **Matplotlib** là thư viện dùng để **vẽ đồ thị 2D** (và một số đồ thị 3D cơ bản).
- Được sử dụng rộng rãi trong khoa học dữ liệu, học máy, phân tích thống kê, và kỹ thuật.
- Giao diện rất giống với cách vẽ đồ thị trong **Matlab**.
  - Vẽ biểu đồ đường (line plot)
  - Vẽ biểu đồ cột (bar chart)
  - Biểu đồ phân tán (scatter plot)
  - Biểu đồ hình tròn (pie chart)
  - Biểu đồ ảnh (imshow)...
- Kết hợp với **NumPy** và **Pandas** để vẽ dữ liệu từ mảng hoặc bảng.  
vd2\_numpy\_36.py

## - Seaborn

Là thư viện giúp **vẽ biểu đồ đẹp, trực quan và thống kê dễ dàng**.

- Hỗ trợ vẽ các loại biểu đồ như:
  - Biểu đồ phân tán (scatter plot)
  - Biểu đồ đường (line plot)
  - Biểu đồ phân phối (histogram, KDE)
  - Biểu đồ hộp (boxplot)
  - Heatmap (bản đồ nhiệt)...
- Làm việc **rất tốt với dữ liệu kiểu pandas.DataFrame**.
- Có sẵn các **mẫu dữ liệu tích hợp** như tips, iris, titanic để thực hành.  
(Seaborn.py)

## - cv2 (OpenCV – Open Computer Vision)

- Là thư viện xử lý ảnh và video mã nguồn mở, rất mạnh mẽ.
- Dùng cho:
  - Đọc, ghi, hiển thị ảnh/video
  - Biến đổi ảnh: đổi màu, làm mờ, xoay, resize,...
  - Nhận diện khuôn mặt, vật thể, camera,...
  - Dùng với NumPy để thao tác ảnh như mảng
- Hỗ trợ cả ảnh tĩnh lẫn video và camera thời gian thực.

## - Scikit-learn: (Machine learning)

- Là **thư viện học máy (machine learning)** mạnh mẽ, dễ dùng trong Python.
- Cung cấp các thuật toán như:
- **Phân loại** (classification): SVM, KNN, Decision Tree,...
- **Hồi quy** (regression): Linear Regression,...
- **Phân cụm** (clustering): K-Means,...
- **Giảm chiều**: PCA, t-SNE,...
- Còn hỗ trợ: Tiền xử lý dữ liệu, Đánh giá mô hình (cross-validation, confusion matrix,...)  
**(Không dùng cho deep learning, mà tập trung vào mô hình cổ điển)**

# Cài đặt

- OpenCV
- Scikit-learn

cv2 (OpenCV)

`pip install opencv-python`

sklearn (scikit-learn)

`pip install scikit-learn`

matplotlib

`pip install matplotlib`

seaborn

`pip install seaborn`