

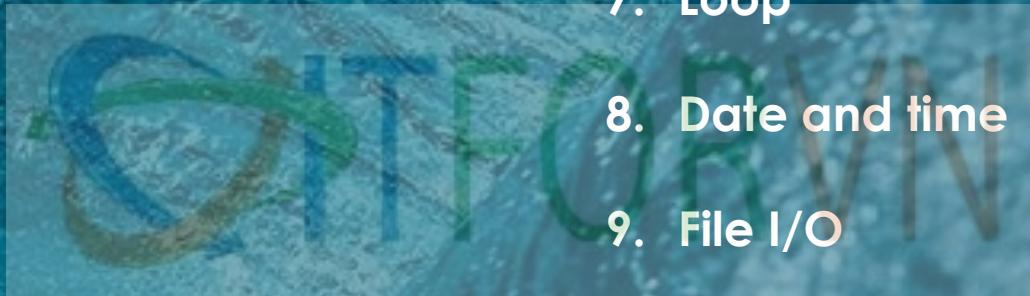


Python basic

ITFROVN - Python 1

Python basic

03

- 
- 1. Variable
 - 2. Math Operator
 - 3. Condition
 - 4. Function
 - 5. Error & Exception
 - 6. List, tuple & dictionary
 - 7. Loop
 - 8. Date and time
 - 9. File I/O
 - 10. Class

Variable

01



1. Variable – 4 basic types of data

- **Integers:** là các số nguyên, đếm được như 1, 2, 3 và bao gồm cả số 0 và số âm. Ví dụ:
 - Số người trong lớp học.
 - Ngày trong 1 tháng.
- **Floats:** còn gọi là số dấu phẩy động, là số có chứa dấu thập phân. Ví dụ:
 - Điểm trung bình: 8.5, 9.0
 - Số Pi: 3.14159265359
- **Strings:** hay còn gọi là văn bản, là một chuỗi các kí tự (character). Ví dụ:
 - Họ và tên người: Phạm Văn A.
 - Địa chỉ nhà: Số 86, đường Lê Thánh Tôn, phường Bến Nghé, quận 1, TP. Hồ Chí Minh (where is it?)
- **Booleans:** có 2 giá trị là **True** (đúng) và **False** (sai).

1. Variable

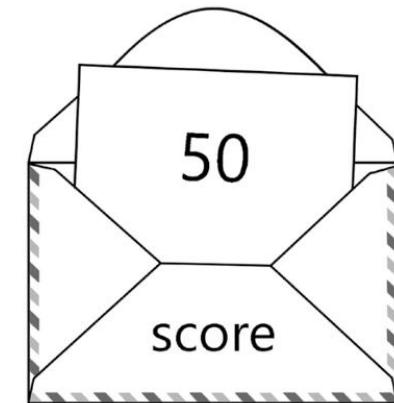
“Variable – biến, là một vị trí trong bộ nhớ (memory) được đặt tên để chứa một giá trị”

- Nội dung của một biến có thể thay đổi hoặc thay đổi theo thời gian.

- Variable assignment: `<variable_name> = <expression>`

- `variable_name`: là tên được sử dụng để đặt cho biến, theo quy tắc:

- bắt đầu bằng chữ, hoặc gạch dưới _ (underscore)
- không thể bắt đầu bằng số
- độ dài tối đa 256 ký tự
- không chứa khoảng trắng (space)
- không chứa các ký tự: + - * / % ()



Ví dụ về biến, với tên là “score” và nội dung là 50

- Ví dụ: `my_name = "Pham Van A"` hoặc `my_age = 20 + 5` hoặc `is_dead = False`

1. Variable – printing

- Cú pháp cơ bản:
 - Python 2: `print` “chỗ này cần được in ra màn hình”
 - Python 3: `print("chỗ này cần được in ra màn hình")`
- Python ≥ 3.6 , sử dụng f-strings: `print(f"")`

```
>>> age = 30
>>> name = "Phạm Văn A"
>>> print(f"Tên tôi là {name}, {age} tuổi")
Tên tôi là Phạm Văn A, 30 tuổi
>>>
```

- Xuống dòng: xác định bằng `os.linesep`

```
PS C:\Program Files\Terminus> py
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 1
Type "help", "copyright", "credits" or "license"
>>> import os
>>> os.linesep
'\r\n'
>>>
```

```
~/Projects > python3
Python 3.8.5 (default, Jan 27 2021, 15:41:15)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license"
>>> import os
>>> os.linesep
'\n'
>>>
```

1. Variable – dynamic typing

- Python là một ngôn ngữ có “dynamic typing” – các biến khai báo không cần ràng buộc type, và được xác định lúc “runtime” (code được chạy).
- Mặc dù là “dynamic typing” nhưng Python được xem là “Strong typing” – kiểu dữ liệu phải được xác định rõ ràng. Đối lập với một số ngôn ngữ “Weak typing” như Javascript.
- **mypy** – công cụ dùng để kiểm tra kiểu dữ liệu (type checker) được phát triển có sự tham gia của Guido Van Rossum. Kể từ phiên bản Python 3.5, type hinting được thêm vào.
- Cài đặt và sử dụng:
 - pip install mypy
 - mypy source_file.py
 - mypy --strict source_file.py
- Chú ý: type không phải là ràng buộc và không gây lỗi. Type chỉ có tác dụng với các chương trình hỗ trợ như mypy, hoặc các IDE hỗ trợ type checking.



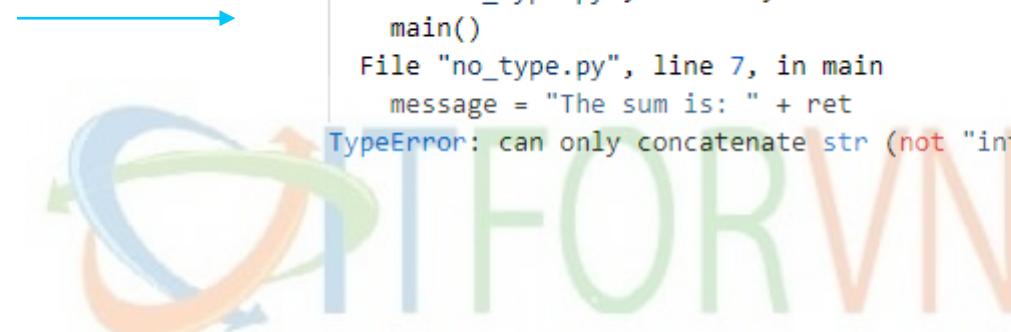
1. Variable – dynamic typing

- No type:

```
def sum(a, b):
    result = a + b
    return result

def main():
    ret = sum(2, 5)
    message = "The sum is: " + ret
    print(message)

main()
```



```
ubuntu@ubuntu:~/Projects/Python-1$ python3 no_type.py
Traceback (most recent call last):
  File "no_type.py", line 10, in <module>
    main()
  File "no_type.py", line 7, in main
    message = "The sum is: " + ret
TypeError: can only concatenate str (not "int") to str
```

- With type:

```
def sum(a: int, b: int) -> int:
    ret = a + b
    return ret

def main() -> None:
    ret = sum(2, 5)
    message = "The sum is: " + ret
    print(message)

main()
```

```
(mm) ubuntu@ubuntu:~/Projects/Python-1$ mypy --strict type.py
type.py:7: error: Unsupported operand types for + ("str" and "int")
Found 1 error in 1 file (checked 1 source file)
```

```
def sum(a: int, b: int) -> int:
    ret = a + b
    return ret

def main() -> None:
    ret = sum(2, 5)
    message = "The sum is: {}".format(ret)
    print(message)

main()
```

WRONG

RIGHT

1. Variable – printing

- Print nhiều argument:

```
>>> print("Hello", os.getlogin(), "good mornig")
Hello ubuntu good mornig
>>> █
```

- Print với separator (mặc định separator là 1 space):

```
>>> print('a', 'b')
a b
>>> print('a', 'b', sep='xxx')
axxxb
```

- Print newline:

```
>>> print("Line 1\nLine 2\nLine 3")
Line 1
Line 2
Line 3
>>> █
```



```
>>> print("Line 1", "Line 2", "Line 3", sep="\n")
Line 1
Line 2
Line 3
```

- Print variables:

```
>>> age = 20
>>> name = "Johny Deep"
>>> print(f"{name} is {age} years old")
Johny Deep is 20 years old
>>>
>>> print(name, "is", age, "years old")
Johny Deep is 20 years old
>>>
```

Math Operator

02



2. Simple math operator

- Python simple math opeartor:
 - **+ cộng**
 - **- trừ**
 - *** nhân**
 - **** mũ (raise to the power of)**
 - **/ chia**
 - **// chia lấy nguyên**
 - **% chia lấy dư (modulo)**

```
>>>
>>> a = 10
>>> b = 6
>>> a + b
16
>>> a - b
4
>>> a * b
60
>>> a ** b
1000000
>>> a / b
1.6666666666666667
>>> a // b
1
>>> a % b
4
>>>
```

- Lưu ý: Python 2 khi chia 2 số, kết quả là số nguyên nhưng Python 3 sẽ là số float

```
root@ubuntu:/usr/bin# python2.7
Python 2.7.18 (default, Mar  8 2021, 13:02:45)
[GCC 9.3.0] on linux2
Type "help", "copyright", "credits" or "license"
>>> a = 5
>>> b = 2
>>> a / b
2
>>> type(a / b)
<type 'int'>
>>>
```

```
root@ubuntu:/usr/bin# python3.8
Python 3.8.5 (default, Jan 27 2021, 15:41:15)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license"
>>> a = 5
>>> b = 2
>>> a / b
2.5
>>> type(a / b)
<class 'float'>
>>> █
```

2. Standard operator module

Opearator module: cung cấp các hàm chức năng tương tự như "simple math" và nhiều hơn. Là 1 thư viện chuẩn của Python.

Module documentation:

<https://docs.python.org/3.8/library/operator.html>

```
root@ubuntu:/usr/bin# python3.8
Python 3.8.5 (default, Jan 27 2021, 15:41:15)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import operator
>>> operator.__file__
'/usr/lib/python3.8/operator.py'
>>>
>>> a = 10
>>> b = 2
>>> operator.lt(a, b)
False
>>> operator.gt(a, b)
True
>>> operator.gt(b, a)
False
>>> operator.add(a, b)
12
>>> operator.sub(a, b)
8
>>> █
```

Condition

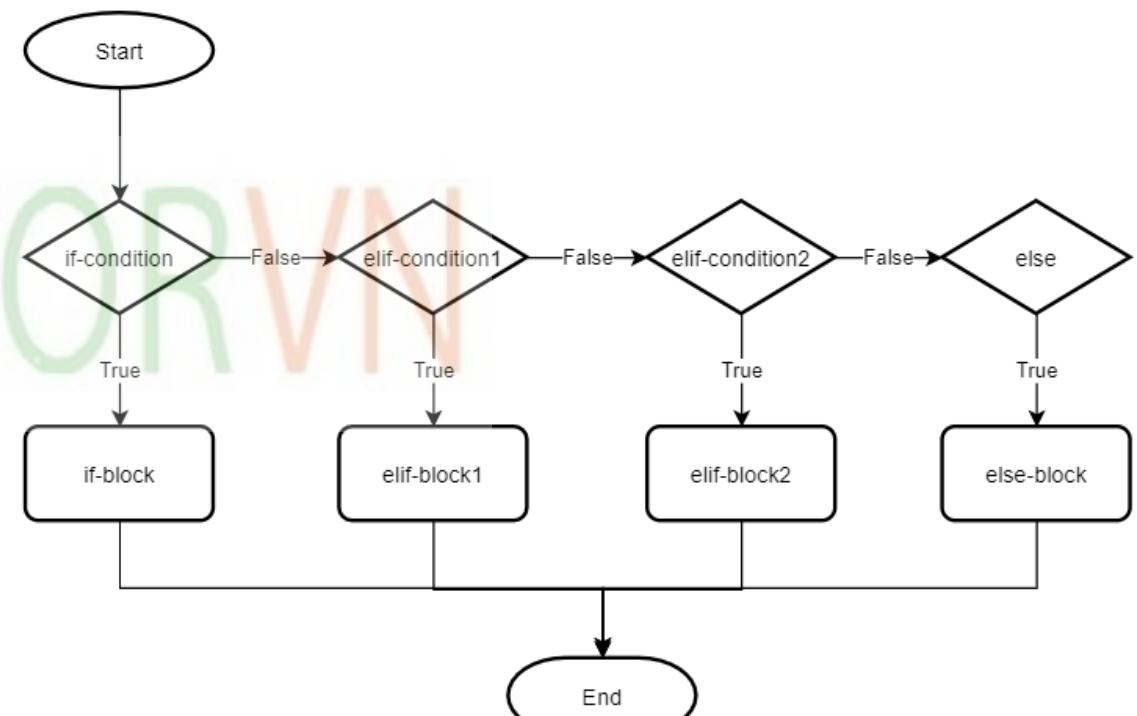
03



3. Condition – decision making

- Thực hiện dựa trên kết quả của một điều kiện nào đó:
 - If today is Monday, then I learn Python.
 - If it's rain, then I go to sleep.
 - If it's rain and I have some friends, then I go to drinking.
- Cú pháp tổng quát:

```
if <boolean expression>:  
    code block  
elif <boolean expression>:  
    code block  
elif <boolean expression>:  
    code block  
# as many elif's as you need  
else:  
    default code
```



Condition flowchart

3. Condition – decision making

- Comparison operators:

Operator	Meaning	Example
==	Equals	if a == b:
!=	Not equals	if a != b:
<	Less than	if a < b:
>	Greater than	if a > b:
<=	Less than or equal to	if a <= b:
>=	Greater than or equal to	if a >= b:

- Nested if:

```
if <boolean expression>:  
    code block  
    ....  
    if <other boolean expression>:  
        code block  
    elif <boolean expression>:  
        code block  
    else:  
        default code  
    elif <boolean expression>:  
        code block  
    else:  
        default code
```

☞ Python không có cấu trúc switch/case giống như các ngôn ngữ như Java, C/C++ hay Bash scripting.

Function

04



4. Function

- Function – “hàm”, là một đoạn code được tổ chức để thực hiện một tác vụ nào đó, chỉ được chạy khi được gọi và có tính tái sử dụng. Function là kiến trúc cơ bản của 1 chương trình, hiện thực tính năng “modular” trong lập trình, giúp chương trình tổ chức tốt hơn và rõ ràng hơn (khi làm việc nhiều người).
- Function có thể được cho các giá trị đầu vào (pass data, parameters) và trả về các kết quả hoặc không.

```
def my_function():
    print("Hello world!")

my_function()

def other_function(name):
    print(f"Hello {name}")

other_function("Trump")

def another_function(a, b):
    return a + b

c = another_function(2, 5)
print(c)
```



Hàm cơ bản

Hàm có truyền vào tham số (data)

Hàm có truyền vào tham số (data) và trả
về kết quả

User-defined function

4. Function

- Built-in function – là các hàm được thiết kế sẵn và sẵn sàng để sử dụng mà không cần phải viết mới, đi kèm với bản cài đặt Python. Có rất nhiều built-in function có sẵn trong Python.
- Các built-in function phổ biến nhất:
 - `input`: lấy input từ user. Ví dụ: `your_name = input("What is your name: ")`
 - `type`: xác định type của bất kỳ một biến hoặc một giá trị:
 - `type_of_10 = type(10)`
 - `var = "I'm Robot" -> type_of_var = type(var)`
 - `conversion function`: thay đổi giá trị từ một kiểu dữ liệu sang một kiểu dữ liệu khác:
 - `int`: chuyển từ kiểu dữ liệu thích hợp sang integer. Ví dụ: `dollar = "10" -> dollar_num = int(dollar)`
 - `float`: chuyển từ kiểu dữ liệu thích hợp sang float. Ví dụ: `grade = "9.5" -> grade_num = float(grade)`
 - `str`: chuyển từ kiểu dữ liệu thích hợp sang string. Ví dụ: `street_num = 86 -> street = str(street_num)`

```
>>> import builtins
>>> dir(builtins)
['ArithmetError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundException', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'ZeroDivisionError', '_', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
>>> 
```

4. Function

Built-in function

Error & Exception

05



5. Error

- Là các **Lỗi** xảy ra trong quá trình lập trình, dẫn đến kết quả không mong muốn xảy ra khi chương trình được thực thi.
- Có thể phân loại thành 3 loại lỗi:
 - **Syntax error:** các lỗi liên quan đến khai báo biến, cú pháp, indentation. Python interpreter sẽ kiểm tra loại này trước tiên khi python code được chạy.
 - **Exception error:** sau khi vượt qua “syntax check”, code được thực thi có thể xảy ra lỗi (runtime error). Ví dụ lỗi xảy ra khi cộng một số với một chuỗi, hoặc chia một số cho 0.
 - **Logic error:** không xảy ra syntax error và exception error nhưng kết quả không như mong muốn. Ví dụ phát triển hàm cộng số nguyên nhưng gõ nhầm dấu + bằng dấu *.

5. Exception

- Python built-in exception:
<https://docs.python.org/3/library/exceptions.html>
- Cú pháp tổng quát:

```
try:  
    # code } 1  
    except <ExceptionType>:  
        # code } 2  
    else:  
        # code } 3  
    finally:  
        # code } 4
```



- 1 đoạn code được thực thi và cần được kiểm tra Exception có xảy ra hay không.
- 2 đoạn code được thực thi tương ứng với loại Exception xảy ra.
- 3 đoạn code được thực thi nếu không có Exception nào xảy ra.
- 4 đoạn code luôn được thực thi, dù có hay không Exception xảy ra.

5. Exception

```
try:  
    a = int(input("Nhập vào số a: "))  
    b = int(input("Nhập vào số b: "))  
except ValueError:  
    print("Số nhập vào phải là SỐ NGUYÊN!")  
else:  
    print(f"Tổng của a và b là: {a + b}")  
finally:  
    print("---Chương trình kết thúc---")
```



```
~/Projects/Python-1/src > python3 int.py  
Nhập vào số a: 1  
Nhập vào số b: 2  
Tổng của 2 số là: 3  
---Chương trình kết thúc---
```

```
from pprint import pprint
```

```
try:  
    a = int(input("Nhập vào số a: "))  
    b = int(input("Nhập vào số b: "))  
except Exception as e:  
    pprint(e)  
else:  
    print(f"Tổng của a và b là: {a + b}")  
finally:  
    print("---Chương trình kết thúc---")
```



```
~/Projects/Python-1/src > python3 int.py  
Nhập vào số a: 1  
Nhập vào số b: a  
Số nhập vào phải là SỐ NGUYÊN!  
---Chương trình kết thúc---
```

```
~/Projects/Python-1/src > python3 int.py
```

```
Nhập vào số a: 1  
Nhập vào số b: a  
ValueError("invalid literal for int() with base 10: 'a'")  
---Chương trình kết thúc---
```

List, tuple and dictionary

06



6. Data structure

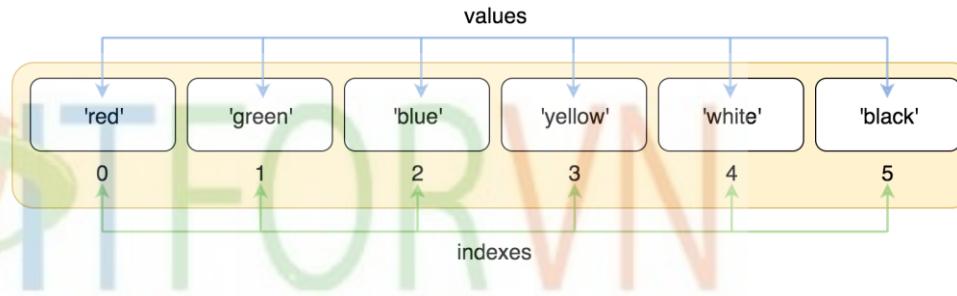
- Ngoài 4 kiểu dữ liệu cơ bản (int, float, string, boolean), Python cung cấp các “Data structure”:
 - List: tập hợp các item theo thứ tự, có thể thay đổi tăng/giảm số lượng item (mutable).
 - Tuple: tập hợp các item như list nhưng không thể tăng/giảm số lượng item (immutable).
(thao tác trên tuples nhanh hơn lists vì tính immutable của tuples)
 - Dictionary: tập hợp các item được tổ chức bởi một cặp “key” – “value”, không theo thứ tự, có thể thay đổi tăng/giảm số lượng item (mutable).
 - Set: tập hợp các item duy nhất, không trùng lặp, có thể thay đổi tăng/giảm số lượng item (mutable).
 - Frozenset: giống như set nhưng không thể thay đổi tăng/giảm số lượng item (immutable).
- Các data structure này đều là object, nên các hàm chức năng được gọi là methods.



Liệt kê các methods có thể sử dụng được trên một object: `dir(object)`. Ví dụ: `dir(my_list)` để xem tất cả methods trên object kiểu list là `my_list`.

6. List

- Cấu trúc:
 - `my_empty_list = []`
 - `my_list = ["a", 1, True, 9.5]`
- Truy cập các item: `my_list[index_number]`
- `len(my_list)`: đếm số lượng item trong list
- Các methods:
 - `append("item")`: thêm 1 item vào list
 - `extend(["item", "item"])`: thêm nhiều hơn một item vào list
 - `count("item")`: đếm số lần item xuất hiện trong list
 - `index("item")`: search item và trả về index của item nếu có tồn tại, trả về `ValueError` nếu không tồn tại.
 - `clear()`: xóa toàn bộ item trong list
 - `insert(2, "abc")`: chèn 1 item vào 1 vị trí thứ 2 trong list
 - `remove("item")`: remove “first occurrence” trong list
 - `pop()`: remove last item trong list
 - `sort()`: sắp xếp thứ tự các item (cùng type) trong list
 - `reverse()`: đảo ngược thứ tự item trong list



6. List

- Operator:
 - + cộng 2 list thành một
 - * double item trong 1 list
- Slicing: sử dụng dấu : để truy cập một phần của list
 - `my_list[:2]` các item có index từ đầu đến 2
 - `my_list[2:]` các item từ index 2 đến cuối cùng
 - `my_list[2:5]` các item từ index 2 đến item thứ 5
 - `my_list[2:-1]` các item từ index 2 đến – item cuối cùng trừ đi 1 item
- Iterate (duyệt) tuần tự các item trong list:

```
for item in my_list:  
    # code
```

6. Tuple

- Tuple gần giống như list, khác biệt ở tính immutable của tuple so với mutable của list.
- Các thao tác thực hiện trên tuple có tốc độ nhanh hơn trên list.
- Cấu trúc:

- `my_tuple = (1, 2, 3)`
- `my_tuple = 1, 2, 3`
- `my_tuple = (1,)` hoặc `my_tuple = 1,` → khai báo tuple với 1 item



- Các methods:
 - `count()`: đếm số lần item xuất hiện trong list
 - `index()`: search item và trả về index của item nếu có tồn tại, trả về ValueError nếu không tồn tại.
- `len(my_tuple)`: đếm số lượng item trong tuple

6. Tuple

- Iterate (duyệt) tuần tự các item trong tuple:

```
for item in my_tuple:  
    # code
```

- Truy cập các item: `my_tuple[index_number]`
- Tuple có thể chứa mutable item, thì mutable item có thể thay đổi được. Ví dụ: trong tuple có 1 item là list.

```
>>> my_tuple = (1, 2, 3, ['a', 'b'])  
>>> my_tuple[3][0] = 'c'  
>>> my_tuple  
(1, 2, 3, ['c', 'b'])  
>>>
```

6. Dictionary

- Còn gọi tắt là “Dict” (kiểu từ điển) – bao gồm các item ở dạng “pair” gồm 1 “key” và 1 “value” tương ứng.
- Khai báo:
 - `my_dict = {"key1": "value1", "key2": "value2"}`
 - `my_dict = { "key1": [1, 2, 3], "key2": ("a", "b", "c")}`
- Truy xuất item trong dict thông qua key: `my_dict[key]`
- Các keys trong dict: `my_dict.keys()`
- Các values trong dict: `my_dict.values()`

⚠️ Lưu ý

- các key trong một dict là duy nhất.
- không có khái niệm thứ tự trong một dict vì các item được truy xuất độc lập thông qua “key”.

6. Dictionary

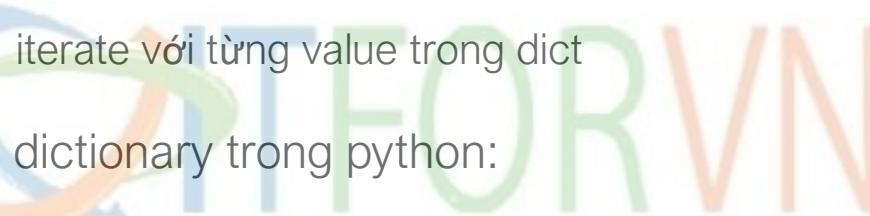
- Các methods thường dùng:
 - `items()`: trả về dạng (key, value) với từng item trong dict
 - `get("key_name", "Item not found")`: lấy giá trị của key tương ứng, trả về “Item not found” string nếu không tìm thấy, hoặc kiểm tra xem key có tồn tại trong dict hay không.
 - `pop("key_name")`: lấy giá trị của key tương ứng đồng thời remove item ra khỏi dict
 - `popitem()`: lấy item cuối trong dict, đồng thời remove item ra khỏi dict (thường là item được display cuối cùng khi print dict)
 - `clear()`: xóa toàn bộ item trong dict
- Thêm 1 item vào dict: `my_dict["new_key"] = "new_value"`
- Xóa 1 item vào dict: `del my_dict["exist_key"]`

6. Dictionary

- Iterate dict:
 - `for item in my_dict.items()`: iterate từng item
 - `for key in my_dict.keys()`: iterate với từng key trong dict
 - `for val in my_dict.values()`: iterate với từng value trong dict
- Implement cấu trúc `switch/case` với dictionary trong python:

```
def week_day(i):
    switcher={
        0:'Sunday',
        1:'Monday',
        2:'Tuesday',
        3:'Wednesday',
        4:'Thursday',
        5:'Friday',
        6:'Saturday'
    }
    return switcher.get(i, "Ngày không hợp lệ!")
```

```
week_day(1)
```



```
~/Projects/Python-1/src > python3 case.py
Monday
~/Projects/Python-1/src >
```

Loop

07



7. Loop

- Loop – vòng lặp, là một cấu trúc xử lý lặp lại việc thực thi một đoạn code đến khi có một điều kiện dừng được thỏa mãn.
- Có 2 cấu trúc xử lý loop trong Python là **for** và **while**.

```
for <var> in <iterable>:  
    # code  
  
while <boolean expression>:  
    # code
```

- **iterable** – là những object có thể “duyệt”, ví dụ như list, tuple, dict hoặc ...string
- Điều kiện dừng:
 - Với iterable object, thì điều kiện dừng tự nhiên là khi đã duyệt toàn bộ object, hoặc bị điều khiển bởi **break** và **continue**.
 - Tự tạo điều kiện dừng dựa vào logic xử lý.

7. Loop

- **break** – câu lệnh dừng hoàn toàn cấu trúc lặp và thực thi tiếp các code ngay sau cấu trúc lặp.
- **continue** – câu lệnh dừng vòng lặp hiện tại và thực hiện vòng lặp kế tiếp trong cấu trúc lặp đang thực hiện.

```
>>> for i in ['foo', 'bar', 'baz', 'qux']:  
...     if 'b' in i:  
...         break  
...     print(i)  
...  
foo
```

```
>>> for i in ['foo', 'bar', 'baz', 'qux']:  
...     if 'b' in i:  
...         continue  
...     print(i)  
...  
foo  
qux
```

- Với **for** có thể dùng statement **else**:

```
>>> for i in ['foo', 'bar', 'baz', 'qux']:  
...     print(i)  
... else:  
...     print('Done.') # Will execute  
...  
foo  
bar  
baz  
qux  
Done.
```

```
>>> for i in ['foo', 'bar', 'baz', 'qux']:  
...     if i == 'bar':  
...         break  
...     print(i)  
... else:  
...     print('Done.') # Will not execute  
...  
foo
```

⚠ else không hoạt động nếu có break
trong cấu trúc lặp

7. Loop

- List comprehension – một cấu trúc for kết hợp với một cấu trúc list để xử lý một số tác vụ nào đó, và trả về kết quả ở dạng list.

```
[f(item) for item in iterable_object if condition]
```

- Ví dụ: bài toán đếm số chẵn:

```
evens = []
for x in range(10):
    mod = x % 2 # mod là số dư khi chia x cho 2
    if mod == 0: # số chẵn chia cho 2 sẽ dư 0
        evens.append(x)
```



```
[evens.append(x) for x in range(10) if x % 2 == 0]
```

Date & time

08



8. Date & time

- Date và time không phải là kiểu dữ liệu cơ bản trong Python.
- Sử dụng standard library datetime: import datetime

```
>>> import datetime  
>>> current = datetime.datetime.now()  
>>> type(current)  
<class 'datetime.datetime'>  
>>> print(current)  
2021-04-12 00:12:44.226682  
>>>
```



- Có thể trích xuất năm, tháng, ngày, giờ, phút, giây từ object datetime:

```
>>> print(current.year)  
2021  
>>> print(current.month)  
4  
>>> print(current.day)  
12  
>>> print(current.hour)  
0  
>>> print(current.minute)  
12  
>>> print(current.second)  
44
```

8. Date & time

- Tạo object datetime với thời gian tùy chỉnh:

```
>>> from datetime import datetime  
>>> my_time = datetime(2021, 4, 12, 01, 30, 29)  
SyntaxError: leading zeros in decimal integer literals are not permitted; use an  
0o prefix for octal integers  
>>> my_time = datetime(2021, 4, 12, 1, 30, 29)  
>>> print(my_time)  
2021-04-12 01:30:29
```

- Format datetime: strftime()

```
>>> print(my_time.strftime("%B"))  
April  
>>> print(my_time.strftime("%A, %d-%m-%Y"))  
Monday, 12-04-2021
```

- Các method và format khác của datetime: <https://docs.python.org/3/library/datetime.html>

File I/O

09



9. File I/O

- File I/O – bao gồm việc tương tác với file, directory trên hệ thống. Việc thao tác với 1 file bao gồm các bước theo thứ tự sau:
 1. Open file
 2. Read/write file
 3. Close file
- Open file:

```
f = open("test.txt")          # open file (đọc) tại thư mục hiện tại  
f = open("/tmp/test.txt")    # open full (đọc) file path
```

```
f = open("test.txt", 'w')    # write in text mode  
f = open("image.png", 'r+b') # read and write in binary mode
```

```
f = open("test.txt", 'a')    # write in append mode  
f = open("test.txt", 'a+')   # read + write in append mode
```



9. File I/O

- Write to file:

```
# Write with write()
f.write("Content wil be written/appened into the file")
```

```
# Write with print()
print("Content wil be written/appened into the file", file=f)
```

- Close file: `f.close()`



Best practice:

```
try:
    f = open("test.txt", encoding='utf-8')
    # file operations
finally:
    f.close()
```

(luôn close file sau khi thao tác)

```
# Python context manager
with open("test.txt", encoding='utf-8') as f:
    # file operations
```

(Context manager tự động handle close file
sau khi thao tác – khi các code thoát khỏi `with`)

9. File I/O

```
>>> f = open("file.txt", "r")
>>> for line in f:
...     print(line)
...
```

Đánh vần theo tao "D"

Đánh vần theo tao "A"

Đánh vần theo tao "T"

Đánh vần và theo tao đi, tao cho mà yết đến lilce, b2c, SB và Dicky

Tao đã thích nghi được cái flow mới

Dùng vần câu với từng đòn roi cho mà yết

Tao đã trôi qua bao năm, không biết mình từ đâu tới

Tao có quyền được nói

Âm nhạc đang chết liệt xác

```
>>> for line in f:
...     print(line)
... }
```



```
>>> f = open("file.txt", "r")
>>> while True:
...     line = f.readline()
...     if not line:
...         break
...     print(line)
...
```

Đánh vần theo tao "D"

Đánh vần theo tao "A"

Đánh vần theo tao "T"

Đánh vần và theo tao đi, tao cho mà yết đến lilce, b2c, SB và Dicky

Tao đã thích nghi được cái flow mới

Dùng vần câu với từng đòn roi cho mà yết

Tao đã trôi qua bao năm, không biết mình từ đâu tới

Tao có quyền được nói

Âm nhạc đang chết liệt xác

>>>

9. File I/O

- Read toàn bộ nội dung file:

```
>>> f = open("file.txt", "r")
>>> all = f.read()
>>> print(all)
Đánh vần theo tao "D"
Đánh vần theo tao "A"
Đánh vần theo tao "T"
Đánh vần và theo tao đi, tao cho mày biết đến lilce, b2c, SB và Dicky
Tao đã thích nghi được cái flow mới
Dùng vần câu với từng đòn roi cho mày toi
Tao đã trôi qua bao năm, không biết mình từ đâu tới
Tao có quyền được nói
Âm nhạc đang chết liệt xác
```

```
>>> print(f.read())
>>>
```



Class

10



10. Class

- Python là ngôn ngữ lập trình hướng đối tượng – Object-oriented programming – OOP.
- Object – là một sự “đóng gói” (encapsulated) của các data (variable) và các function thao tác trên các data. Class là template/blueprint để tạo nên Object.
- Ví dụ:
 - Class con người, bao gồm các “data” như cân nặng, chiều cao, màu da, giới tính, ...và các function như đi lại, ăn uống, làm việc, làm đẹp, ...
 - “Phạm Văn A” chính là một Object được tạo nên từ template “Con người”, một hiện thực hóa của lớp trừu tượng “Con người”.
- Các hàm định nghĩa thao tác trên dữ liệu của class:
 - Trong Class: function
 - Trên Object: method

10. Class

- Class:

```
class Person:  
    """  
        This is Human prototype  
        Created by GOD  
    """  
  
    def __init__(self, name, color):  
        self.name, self.color = name, color  
  
    # This is Instance method  
    def greet(self):  
        print("Hello, I'm human")  
        print(f"\tMy name: {self.name}")  
        print(f"\tMy color: {self.color}")  
  
    # This is Class method  
    @classmethod  
    def class_method(cls):  
        print(f"This is class method: {cls}")  
  
    # This is Static method  
    @staticmethod  
    def static_method():  
        print("This is static method")
```



- Object:

```
person = Person("Johny Deep", "White")  
person.greet()
```



```
~/Projects/Python-1/src > ./human.py  
Hello, I'm human  
My name: Johny Deep  
My color: White
```

- Các loại method:

- Instance method: truy cập `self`
- Class method: truy cập `cls`
- Static method: KHÔNG truy cập `self` và `cls`

10. Class

- Kế thừa:

```
class Person:  
    """  
    This is Human prototype  
    Created by GOD  
    """  
  
    def __init__(self, name, color):  
        self.name, self.color = name, color  
  
    # This is Instance method  
    def greet(self):  
        print("Hello, I'm human")  
        print(f"\tMy name: {self.name}")  
        print(f"\tMy color: {self.color}")  
  
    # Get name  
    def get_name(self):  
        print(f"Name: {self.name}")  
  
    # Get sex  
    def get_sex(self):  
        print(f"Sex: {self.sex}")
```

```
class Female(Person):  
    """  
    This is Female  
    --human also  
    """  
    sex = "Female"  
  
female = Female("Charlotte Linlin", "White")  
female.greet()  
female.get_sex()
```

```
~/Projects/Python-1/src > python3 test.py  
Hello, I'm human  
    My name: Charlotte Linlin  
    My color: White  
    Sex: Female
```

10. Class

- Lớp Female kế thừa toàn bộ thuộc tính (attribute) và phương thức (method) của lớp Person.
- Lớp cha và lớp con có attribute và method cùng tên nhau: sẽ ưu tiên thực thi và gọi các attribute, method được khai báo trong lớp “được khởi tạo”.
- Lớp con truy cập attribute, method trong lớp cha: `super().parent_xxx()`

```
class Person:  
    """  
    This is Human prototype  
    Created by GOD  
    """  
  
    id = "Person"  
  
    def __init__(self, name, color):  
        self.name, self.color = name, color  
  
    # This is Instance method  
    def greet(self):  
        print("Hello, I'm human")  
        print(f"\tMy name: {self.name}")  
        print(f"\tMy color: {self.color}")  
  
    # Get name  
    def get_name(self):  
        print(f"Name: {self.name}")  
  
    # Get sex  
    def get_sex(self):  
        print(f"Sex: {self.sex}")
```

```
class Female(Person):  
    """  
    This is Female  
    --human also  
    """  
    sex = "Female"  
  
    def get_id(self):  
        print(f"Parent ID: {super().id}")  
  
female = Female("Charlotte Linlin", "White")  
female.greet()  
female.get_sex()  
female.get_id()
```