

Ngôn Ngữ Lập Trình Python

Function, Module, Package, Exception

Trịnh Tấn Đạt

Đại Học Sài Gòn

trinhtandat@sgu.edu.vn

<http://sites.google.com/site/ttdat88>

Nội Dung

- Function
- Module
- Package
- Exception
- FILE (I/O)

Function

- Hàm là một khối các câu lệnh chỉ thực hiện khi được gọi
- Trong Python, hàm được định nghĩa bằng từ khóa **def** theo sau là **tên hàm** và dấu ngoặc đơn ():

```
def my_function():  
    print("Hello from a function")
```

Function

- Ý nghĩa canh lề trong hàm

```
1 #define a function
2 def func1():
3     print ("I am learning Python Function")
4
5 func1()
```

Khi hàm "print" được khai báo ngay dưới func1(), chương trình sẽ báo lỗi thụt lề

File "C:/Users/DK/Desktop/Python code/Python Test/Python 10/Python10 Code/Python10.1.py"
print ("I am learning Python Function")
^
IndentationError: expected an indented block

```
Python10.1.py x
1 #define a function
2 def func1():
3     print ("I am learning Python Function")
4
5 func1()
```

Khi bạn thụt lề (khoảng trắng) trước hàm "print", chương trình sẽ trả về kết quả như mong muốn.

Run Python10.1
"C:/Users/DK/Desktop/Python code/Python Test/Python 10/Python10 Code/Python10.1.py"
I am learning Python Function

Function

- Ý nghĩa canh lề trong hàm

```
1 #define a function
2 def func1():
3     print ("I am learning Python Function")
4     print("still in func1")
5
6 func1()
```

Khi bạn gọi một câu lệnh khác trong cùng một hàm, ví dụ: lệnh `print "still in function1"`, bạn cần chắc chắn rằng câu lệnh này nằm thẳng bên dưới câu lệnh `print` đầu tiên. Nếu không chương trình sẽ báo lỗi thụt lề.

File "C:/Users/DK/Desktop/Python code/Python Test/Python 10/Python10 Code/Python10.1.py"
print("still in func1")
^
IndentationError: unindent does not match any outer indentation level

```
1 #define a function
2 def func1():
3     print ("I am learning Python Function")
4     print("still in func1")
5
6 func1()
7
```

Chúng ta sẽ thu được kết quả như mong muốn nếu duy trì cùng một cách thụt lề cho cả hai câu lệnh

Run Python10.1
"C:/Users/DK/Desktop/Python code/Python Test/Python 10/Python10 Code/Python10.1.py"
I am learning Python Function
still in func1

Function

- Gọi một hàm: Để gọi một hàm, chúng ta sử dụng tên hàm, theo sau là cặp dấu ngoặc đơn ():

```
def my_function():  
    print("Hello, World!")  
my_function()  
# In ra màn hình: Hello, World!
```

Function

- Tham số:
 - Các tham số có thể thêm sau tên hàm, bên trong cặp dấu ngoặc đơn ().
 - Chúng ta có thể thêm bao nhiêu tham số tùy thích, chỉ cần tách chúng bằng dấu phẩy.

```
def my_function (name):  
    print ("Tên: " + name)  
my_function ("Emil")  
my_function ("Tobias")  
my_function ("Linus")  
""" In ra màn hình:  
Tên: Emil  
Tên: Tobias  
Tên: Linus"""
```

Function

- Tham số mặc định
 - Tham số mặc định là tham số đã có sẵn một giá trị trước khi hàm được gọi.
 - Nếu chúng ta gọi một hàm mà không truyền vào tham số, hàm sẽ sử dụng giá trị mặc định.
 - Ở trường hợp còn lại, tham số mặc định sẽ hoạt động như tham số thông thường.

```
def my_function(country = "Việt Nam"):
    print ("Tôi đến từ " + country)
my_function("Lào")
my_function("Campuchia")
my_function()
""" In ra màn hình:
Tôi đến từ Lào
Tôi đến từ Campuchia
Tôi đến từ Việt Nam"""
```


Function

- Hàm có kiểu trả về
 - Để cho một hàm có thể trả về, chúng ta cần phải sử dụng từ khóa **return**:

```
def my_function (x):  
    return 5 * x  
  
print (my_function(3))  
print (my_function(5))  
print (my_function(9))  
""" In ra màn hình:  
15  
25  
45"""
```

Function

- Trong phiên bản Python 3.6 có 68 hàm Python được tích hợp sẵn.

Hàm	Mô tả
<code>abs()</code>	Trả về giá trị tuyệt đối của một số
<code>all()</code>	Trả về True khi tất cả các phần tử trong iterable là đúng
<code>any()</code>	Kiểm tra bất kỳ phần tử nào của iterable là True
<code>ascii()</code>	Tả về string chứa đại diện (representation) có thể in
<code>bin()</code>	Chuyển đổi số nguyên sang chuỗi nhị phân
<code>bool()</code>	Chuyển một giá trị sang Boolean
<code>bytearray()</code>	Trả về mảng kích thước byte được cấp
<code>bytes()</code>	Trả về đối tượng byte không đổi
<code>callable()</code>	Kiểm tra xem đối tượng có thể gọi hay không
<code>chr()</code>	Trả về một ký tự (một chuỗi) từ Integer
<code>classmethod()</code>	Trả về một class method cho hàm
<code>compile()</code>	Trả về đối tượng code Python
<code>complex()</code>	Tạo một số phức
<code>delattr()</code>	Xóa thuộc tính khỏi đối tượng
<code>dict()</code>	Tạo Dictionary

Function

- Hàm main() trong python: `if __name__ == "__main__":`

```
1 def main():
2     print("Hello World!")
3     print("Guru99")
```

main()

Run Code4_1

"C:\Users\DK\Desktop\Python code\Python T...
4/Code4/Code4_1.py"

Guru99

Tại sao chỉ có "Guru99" được in ra?

```
1 def main():
2     print("Hello World!")
3
4
5 if __name__ == "__main__":
6     main()
7
8 print("Guru99")
9
10
11
12
13
14
15
```

Run Code4_2

"C:\Users\DK\Desktop\Python code\...
4/Code4/Code4_2"

Hello World!
Guru99

Khi bạn khai báo hàm main, chương trình sẽ gọi hàm main và in ra "Hello World!"

Function

- Hàm Lambda trong Python:
 - Trong Python, **hàm vô danh** là hàm được định nghĩa mà không có tên.
 - Nếu các hàm bình thường được định nghĩa bằng cách sử dụng từ khóa `def`, thì **hàm vô danh** được định nghĩa bằng cách sử dụng từ khóa **lambda**
- Một hàm Lambda trong Python có cú pháp sau:

`lambda` tham_so: bieu_thuc

```
myfunc = lambda i: i*2
print(myfunc(2))
# In ra màn hình: 4
```

Function

- Hàm vô danh có thể được định nghĩa với nhiều hơn một tham số đầu vào:

```
myfunc = lambda x,y: x*y  
print(myfunc(3,6))  
# In ra màn hình: 18
```

Function

- Sức mạnh của hàm vô danh được thể hiện khi chúng ta tạo các hàm ẩn trong thời gian chạy của chương trình:

```
def myfunc(n):  
    return lambda i: i*n  
  
doubler = myfunc(2)  
tripler = myfunc(3)  
val = 11  
print("Doubled: " + str(doubler(val)) + ". Tripled: " + str(tripler(val)))  
# In ra màn hình: Doubled: 22. Tripled: 33
```

Function

- Ví dụ dùng hàm lambda với filter():
 - Hàm filter() sẽ lấy các tham số là **một hàm** và **một list**.
 - Hàm được gọi với tất cả các mục trong list và list mới sẽ được trả về, chứa các mục mà hàm đánh giá là True.

```
list_goc = [10, 9, 8, 7, 6, 1, 2, 3, 4, 5]
list_moi = list(filter(lambda a: (a%2 == 0) , list_goc))
# Kết quả: [10, 8, 6, 2, 4]
print(list_moi)
```

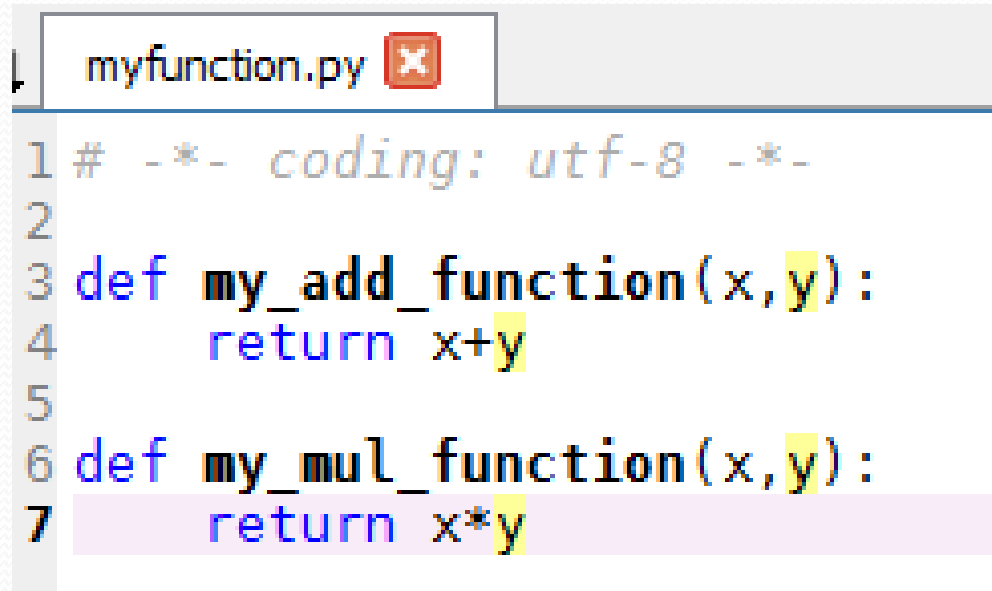
Function

- Ví dụ dùng hàm Lambda với map():
 - Hàm map() cũng lấy các tham số là một hàm và một list.
 - Hàm được gọi với tất cả các mục trong list và list mới được trả về chứa các mục được hàm trả về tương ứng cho mỗi mục.

```
list_goc = [10, 9, 8, 7, 6, 1, 2, 3, 4, 5]
list_moi = list(map(lambda a: a*2, list_goc))
# Kết quả: [20, 18, 16, 14, 12, 2, 4, 6, 8, 10]
print(list_moi)
```


Module

- Module đề cập đến một file (.py) chứa những câu lệnh Python, các hàm và các định nghĩa.
- Một file chứa code Python, ví dụ **myfunction.py** được gọi là module và tên của module sẽ là **myfunction**.



```
myfunction.py ✕  
1 # -*- coding: utf-8 -*-  
2  
3 def my_add_function(x,y):  
4     return x+y  
5  
6 def my_mul_function(x,y):  
7     return x*y
```

Module

- Module thường được sử dụng khi muốn chia chương trình lớn thành những file nhỏ hơn để dễ quản lý và tổ chức.
- Module cho phép tái sử dụng code
- **Làm sao để nhập module trong Python?**
 - Chúng ta có thể nhập các định nghĩa từ module này vào module khác hoặc vào trình thông dịch trong Python.
 - Chúng ta sử dụng từ khóa **import** để thực hiện việc này

Module

- Ví dụ: tạo file **testfunction.py** (lưu cùng folder với module **myfuction**) để sử dụng module myfuction

```
myfunction.py ✕
1 # -*- coding: utf-8 -*-
2
3 def my_add_function(x,y):
4     return x+y
5
6 def my_mul_function(x,y):
7     return x*y
```

```
myfunction.py ✕ testfunction.py* ✕
1 # -*- coding: utf-8 -*-
2 import myfunction
3 m = myfunction.my_add_function(1,2)
4 print(m) # 3
5 n = myfunction.my_mul_function(3,2)
6 print(n) # 6
```

Module

❖ Sử dụng lệnh import:

- Có thể gọi nhiều module

```
import module1, module2,...
```

hoặc

```
import module1
```

```
import module2
```

```
Import module3
```

Module

- Ví dụ: module **math** tích hợp sẵn trong Python

```
import math
```

```
a = 3.2
```

```
# làm tròn lên 1 số
```

```
print(math.ceil(a)) # 4
```

```
# làm tròn xuống 1 số
```

```
print(math.floor(a)) # 3
```

Module

- **Đặt lại tên cho module:** có thể tạo bí danh khi bạn nhập module bằng cách sử dụng từ khóa **as** :

```
import math as m
```

```
a = 3.2
```

```
# làm tròn lên 1 số
```

```
print(m.ceil(a)) # 4
```

```
# làm tròn xuống 1 số
```

```
print(m.floor(a)) # 3
```

Module

- **Lệnh `from...import` trong Python:** Bạn có thể chọn chỉ nhập các phần từ module (một vài hàm hoặc biến, ...) thay vì dùng tất cả thành phần trong module.

`from` modules `import` something, something2,...

```
from math import pi,e  
print(pi) # 3.141592653589793  
print(e) # 2.718281828459045
```

Module

import mỗi phương thức ceil ở trong module math.

```
from math import ceil
```

```
a = 3.2
```

```
print(ceil(a)) # kết quả: 4
```

```
print(floor(a)) # Kết quả: name 'floor' is not defined
```


Module

import tất cả mọi thứ được cho phép từ module math

```
from math import *
```

```
a = 3.2
```

```
print(ceil(a)) # kết quả: 4
```

```
print(floor(a)) # Kết quả: 3
```

- Đối với trường hợp các bạn sử dụng **from ... import *** thì mặc định python nó sẽ **không import** được các đối tượng có tên được **bắt đầu bằng ký tự _**.
- Trong trường hợp này, nếu như bạn muốn import được các đối tượng đó thì bạn sẽ phải chỉ đích danh các đối tượng đó.

Module

```
# file mathplus.py
def _get_sum (a, b):
    return a + b
```

```
from mathplus import _get_sum

print(_get_sum(5,7))
# kết quả: 12
```

```
from mathplus import *

print(_get_sum(5,7))
# kết quả: name '_get_sum' is not defined
```

Module

- **Sử dụng hàm `dir()`:**

- Có chức năng liệt kê tất cả các tên hàm (hoặc tên biến) trong một module.

```
import math
x = dir(math)
print(x)
```

```
['__doc__', '__loader__', '__name__', '__package__',  
'__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan',  
'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',  
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',  
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma',  
'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf',  
'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',  
'log2', 'modf', 'nan', 'pi', 'pow', 'radians',  
'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh',  
'tau', 'trunc']
```

Module

❖ Đường dẫn tìm kiếm module Python

- Khi nhập module, Python sẽ tìm một vài nơi. Trình thông dịch tìm các module có sẵn, nếu không thấy nó sẽ vào danh sách các thư mục được định nghĩa trong `sys.path`. Thứ tự tìm kiếm sẽ là:
 - Thư mục hiện tại.
 - `PYTHONPATH` (một biến môi trường với danh sách thư mục).
 - Thư mục mặc định có vị trí phụ thuộc vào chọn lựa trong quá trình cài đặt.

```
import sys  
print(sys.path)
```

Module

- Ví dụ: nếu module myfunction và file testfunction được lưu cùng một folder (C:\Users\sony\Desktop\python_co_ban\test\mymodule) thì import ko lỗi

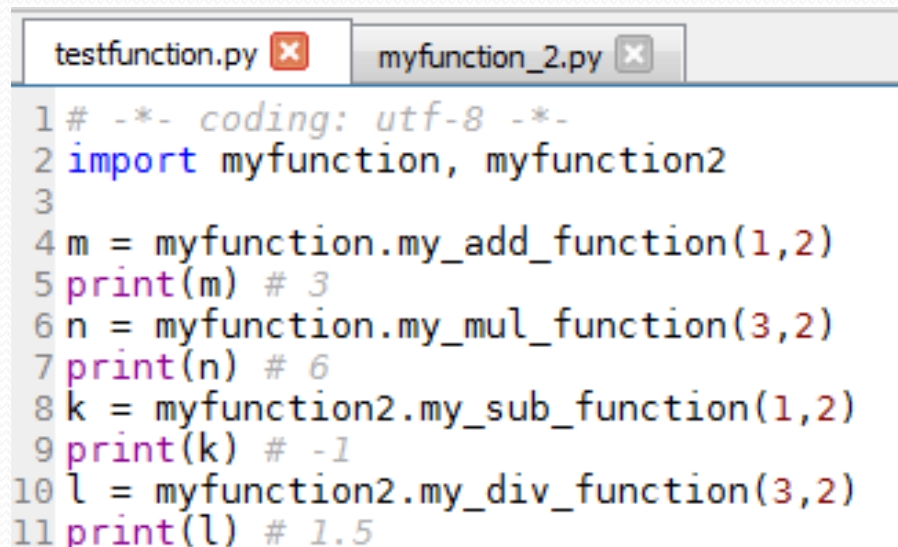
```
myfunction.py
1 # -*- coding: utf-8 -*-
2
3 def my_add_function(x,y):
4     return x+y
5
6 def my_mul_function(x,y):
7     return x*y
```

```
myfunction.py  testfunction.py*
1 # -*- coding: utf-8 -*-
2 import myfunction
3 m = myfunction.my_add_function(1,2)
4 print(m) # 3
5 n = myfunction.my_mul_function(3,2)
6 print(n) # 6
```

- Ví dụ 2: Tạo một module **myfunction2.py** và lưu tại ổ F:\myfunction_2

```
# -*- coding: utf-8 -*-  
  
def my_sub_function(x,y):  
    return x-y  
  
def my_div_function(x,y):  
    return x/y
```

- Hàm testfunction.py được lưu cùng một folder
(C:\Users\sony\Desktop\python_co_ban\test\mymodule)



```
1 # -*- coding: utf-8 -*-  
2 import myfunction, myfunction2  
3  
4 m = myfunction.my_add_function(1,2)  
5 print(m) # 3  
6 n = myfunction.my_mul_function(3,2)  
7 print(n) # 6  
8 k = myfunction2.my_sub_function(1,2)  
9 print(k) # -1  
10 l = myfunction2.my_div_function(3,2)  
11 print(l) # 1.5
```

```
File "C:/Users/sony/Desktop/python_co_ban/test/  
mymodule/testfunction.py", line 2, in <module>  
    import myfunction, myfunction2  
ModuleNotFoundError: No module named 'myfunction2'
```

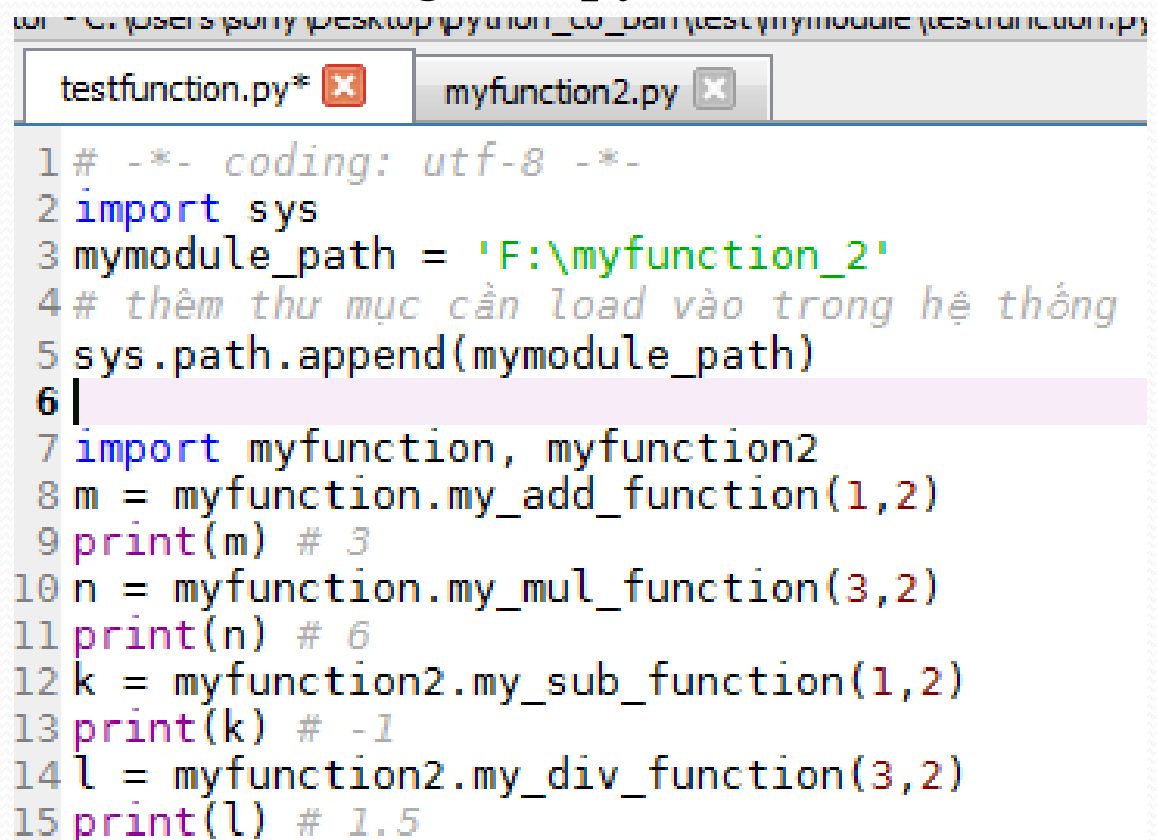
- Để khắc phục thì chúng ta phải thêm đường dẫn của thư mục **F:\myfunction_2** chứa **module myfunction2** vào biến môi trường cho python (PYTHONPATH) dùng **sys.path**

```
import sys
```

```
mymodule_path = 'F:\myfunction_2'
```

```
# thêm thư mục cần load vào trong hệ thống
```

```
sys.path.append(mymodule_path)
```



```
1 # -*- coding: utf-8 -*-
2 import sys
3 mymodule_path = 'F:\myfunction_2'
4 # thêm thư mục cần load vào trong hệ thống
5 sys.path.append(mymodule_path)
6
7 import myfunction, myfunction2
8 m = myfunction.my_add_function(1,2)
9 print(m) # 3
10 n = myfunction.my_mul_function(3,2)
11 print(n) # 6
12 k = myfunction2.my_sub_function(1,2)
13 print(k) # -1
14 l = myfunction2.my_div_function(3,2)
15 print(l) # 1.5
```

Package

- Các file tương tự hoặc cùng liên quan đến một chủ đề nào đó sẽ được để trong cùng một thư mục.
- Python có các **package cho thư mục** và **module cho file**.
- Khi chương trình đang code ngày càng lớn với rất nhiều module, chúng ta sẽ đặt những module giống nhau vào một package, và những nhóm module khác vào package khác.
- Trong một package có thể có package con và các module khác.
- Một thư mục phải chứa file có tên **__init__.py** để Python hiểu thư mục này là một package. File này có thể để trống.

/home/dattt/Downloads/alpr-unconstrained-master/src/



```
import numpy as np
import cv2
import sys

from glob import glob

def im2single(I):
    assert(I.dtype == 'uint8')
    return I.astype('float32')/255.

def getWH(shape):
    return np.array(shape[1::-1]).astype(float)

def IOU(tl1,br1,tl2,br2):
    wh1,wh2 = br1-tl1,br2-tl2
    assert((wh1>=.0).all() and (wh2>=.0).all())

    intersection_wh = np.maximum(np.minimum(br1,br2) - np.maximum(tl1,tl2),0.)
    intersection_area = np.prod(intersection_wh)
    area1,area2 = (np.prod(wh1),np.prod(wh2))
    union_area = area1 + area2 - intersection_area;
    return intersection_area/union_area
```

Ta có thể nhập các module từ package sử dụng toán tử dấu chấm (.)

Ví dụ:

`import src.label`

`import src.utils`

`from src import loss`

`from src.utils import IOU,getWH`

Exception

- Khi chương trình đang thực thi và phát sinh ra lỗi (runtime error), các lỗi này thường được gọi là exception (ngoại lệ).
- Ngoại lệ được tạo ra để xử lý vấn đề đó nhằm tránh cho chương trình bị hỏng (crash).
- Ngoại lệ có thể là bất kỳ tình huống bất thường nào trong chương trình làm hỏng luồng thực thi của chương trình.
- Ví dụ về exception trong python:
 - FileNotFoundError
 - ZeroDivisionError
 - ImportError
 - ValueError
- Bất kỳ khi nào lỗi runtime xảy ra, Python sẽ tạo một đối tượng ngoại lệ tương ứng. Chương trình sẽ ngừng thực thi, chuyển sang quá trình gọi và in ra lỗi cho đến khi nó được xử lý.

Exception

- Có nhiều loại exception được tạo ra khi gặp các lỗi tương ứng.
- Để xem các exception được hỗ trợ dùng hàm locals()
`print(dir(locals()['__builtins__']))`
- Người dùng có thể tự định nghĩa ngoại lệ nếu cần thiết.

```
In [9]: print(dir(locals()['__builtins__']))
```

```
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError',  
'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetErr  
or', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundEr  
ror', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'I  
ndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotF  
oundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'Pe  
ndingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', 'Runt  
imeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExi  
t', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeE  
rror', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError',  
'__IPYTHON__', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__', 'a  
bs', 'all', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'co  
mplex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'display', 'divmod', 'enumerate', 'eval', 'exec', 'execfile', 'filte  
r', 'float', 'format', 'frozenset', 'get_ipython', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'in  
t', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'objec  
t', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'range', 'repr', 'reversed', 'round', 'runfile', 'set', 'setattr', 'slic  
e', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

Exception

- Một vài ví dụ về exception

Ngoại lệ	Lý do gây ra
AssertionError	Xảy ra khi câu lệnh assert thất bại.
AttributeError	Xảy ra khi gán thuộc tính hoặc tham chiếu thất bại.
EOFError	Xảy ra khi hàm input () chạm vào điều kiện end-of-file.
FloatingPointError	Xảy ra khi một số thực dấy phẩy động thực thi không thành công
GeneratorExit	Xảy ra khi phương thức close() của hàm generator được gọi.
ImportError	Xảy ra khi không tìm thấy module được import.
IndexError	Xảy ra khi một chỉ số trong chuỗi (sequence) nằm ngoài phạm vi.
KeyError	Xảy ra khi không tìm thấy khóa ánh xạ (từ điển) trong tập hợp các khóa hiện có.
KeyboardInterrupt	Xảy ra khi người dùng nhấn phím ngắt (thông thường là Ctrl-C hoặc Delete).
MemoryError	Xảy ra khi một operation hết bộ nhớ nhưng tình huống vẫn có thể được sửa chữa (bằng cách xóa một số đối tượng).

Exception

- Ví dụ các ngoại lệ phổ biến

```
In [6]: 100/0
```

```
-----  
ZeroDivisionError                                Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_5676\621970991.py in <module>  
----> 1 100/0  
  
ZeroDivisionError: division by zero
```

```
In [8]: open("text.txt")
```

```
-----  
FileNotFoundError                                Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_5676\683032829.py in <module>  
----> 1 open("text.txt")  
  
FileNotFoundError: [Errno 2] No such file or directory: 'text.txt'
```

Exception handling

- Các ngoại lệ trong Python được xử lý bằng khối lệnh **try ... except**
- Phần khối lệnh của try : gồm những đoạn code có thể tạo ra exception. Nếu một exception xảy ra, các lệnh còn lại trong khối bị bỏ qua.
- Phần khối lệnh của except: được gọi bởi exception handler, dùng để bắt lỗi. Khối except được thực thi khi lỗi được sinh ra (nếu không có lỗi thì bỏ qua).

Exception handling

- Ví dụ:

```
# import module sys to get the type of exception
import sys
randomList = ['a', 0, 2, 5]
for entry in randomList:
    try:
        print("The entry is", entry)
        r = 1/int(entry)
        break
    except:
        print("Oops!", sys.exc_info()[0], "occurred.")
        print("Next entry.")
        print()
print("The reciprocal of", entry, "is", r)
```

print the name of the exception using the `exc_info()` function inside `sys` module

```
In [17]: # import module sys to get the type of exception
import sys
randomList = ['a', 0, 2, 5]
for entry in randomList:
    try:
        print("The entry is", entry)
        r = 1/int(entry)
        break
    except:
        print("Oops!", sys.exc_info()[0], "occurred.")
        print("Next entry.")
        print()
print("The reciprocal of", entry, "is", r)
```

```
The entry is a
Oops! <class 'ValueError'> occurred.
Next entry.
```

```
The entry is 0
Oops! <class 'ZeroDivisionError'> occurred.
Next entry.
```

```
The entry is 2
The reciprocal of 2 is 0.5
```


Exception handling

```
# import module sys to get the type of exception
import sys
```

```
randomList = ['a', 0, 2, 5]
```

```
for entry in randomList:
```

```
    try:
```

```
        print("The entry is", entry)
```

```
        r = 1/int(entry)
```

```
        break
```

```
    except Exception as e:
```

```
        print("Oops!", e.__class__, "occurred.")
```

```
        print("Next entry.")
```

```
        print()
```

```
print("The reciprocal of", entry, "is", r)
```

```
The entry is a
Oops! <class 'ValueError'> occurred.
Next entry.
```

```
The entry is 0
Oops! <class 'ZeroDivisionError'> occurred.
Next entry.
```

```
The entry is 2
The reciprocal of 2 is 0.5
```

every exception in Python inherits from the base Exception class

Exception handling

- Xử lý cụ thể một ngoại lệ: ta có thể chỉ định các ngoại lệ cụ thể cho khối lệnh except.
- Cú pháp:

```
try:  
    # khối code lệnh try  
except exceptionName:  
    # khối code lệnh except
```

Trong đó: tham số exceptionName là tên các exception tương ứng.

Exception handling

- Mệnh đề try có thể có nhiều mệnh đề except khác nhau để xử lý
- Nếu khối lệnh trong try có lỗi, thì chương trình sẽ tìm đến các except phía dưới, except nào thỏa mãn thì sẽ thực thi code trong khối lệnh đó.
- Ví dụ:

```
try :  
    # khối code lệnh try  
  
except ValueError:  
    # code xử lý ValueError  
  
except RuntimeError:  
    # code xử lý RuntimeError
```

Exception handling

- Người dùng có thể xử lý nhiều exception trên một lần khai báo except, cách nhau bằng dấu phẩy.
- Ví dụ

```
try:  
    # do something  
    pass  
except ValueError:  
    # handle ValueError exception  
    pass  
except (TypeError, ZeroDivisionError):  
    # handle multiple exceptions  
    # TypeError and ZeroDivisionError  
    pass  
except:  
    # handle all other exceptions  
    pass
```

Ví dụ

```
try:
    a = int(input("Tell me one number: "))
    b = int(input("Tell me another number: "))
    print("a/b = ", a/b)
    print("a+b= ", a+b)
except ValueError:
    print("Could not convert to a number.")
except ZeroDivisionError:
    print("Can't divide by zero")
except:
    print("Something went very wrong.")
```

Exception handling

- Xây dựng một exception sử dụng câu lệnh **raise** là một cách khác để xử lý ngoại lệ.
- Khi đó, người dùng có thể tạo ra riêng một exception đó là exception được nêu ra khi vấn đề nằm bên ngoài phạm vi dự kiến của lỗi xảy ra
- Ví dụ:

```
try:
    x = input('Nhập một số trong khoảng 1-10: ')
    if x<1 or x>10:
        raise Exception
    print('Bạn vừa nhập một số hợp lệ :D')
except:
    print('Số bạn vừa nhập nằm ngoài khoảng cho phép mất rồi!')
```

Exception handling

- Module traceback là cách khác để xử lý ngoại lệ. Được sử dụng để in ra dấu vết của chương trình khi exception xảy ra.
- Ví dụ:

```
-----  
NameError                                Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_4276\1707110226.py in <module>  
      3 # splits the training and test data set in 80% : 20%  
      4 # assign random_state to any value.This ensures consistency.  
----> 5 X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=5)  
      6 print(X_train.shape)  
      7 print(X_test.shape)  
  
NameError: name 'X' is not defined
```

Exception handling

- **try ... finally**: là một cách khác để viết lệnh try.
- **finally**: còn gọi là mệnh đề cleanup/termination, mệnh đề này luôn thực thi bất kể có hay không có lỗi trong lệnh try. Thường được dùng để giải phóng tài nguyên.
- Ví dụ:

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


Exception handling

- Ví dụ:

```
In [8]: mauso = input("Bạn hãy nhập giá trị mẫu số: ")
try:
    ketqua = 15/int(mauso)
    print("Kết quả là:",ketqua)
finally:
    print("Bạn đã nhập số không thể thực hiện phép tính.")
```

Bạn hãy nhập giá trị mẫu số: 5

Kết quả là: 3.0

Bạn đã nhập số không thể thực hiện phép tính.

```
In [9]: mauso = input("Bạn hãy nhập giá trị mẫu số: ")
try:
    ketqua = 15/int(mauso)
    print("Kết quả là:",ketqua)
finally:
    print("Bạn đã nhập số không thể thực hiện phép tính.")
```

Bạn hãy nhập giá trị mẫu số: 0

Bạn đã nhập số không thể thực hiện phép tính.

```
-----
ZeroDivisionError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_3732\1827368920.py in <module>
      1 mauso = input("Bạn hãy nhập giá trị mẫu số: ")
      2 try:
----> 3     ketqua = 15/int(mauso)
      4     print("Kết quả là:",ketqua)
      5 finally:

ZeroDivisionError: division by zero
```


Nội Dung

- Input
- Giới thiệu về FILE
- Đọc FILE
- Ghi FILE

Input

- Trong Python có cung cấp cho chúng ta hàm input để nhận dữ liệu từ người dùng nhập vào

```
input(something)
```

```
print("Hello guy!")  
age = input("How old are you? ")  
print("age: " + age)
```

Giới thiệu về FILE

- File là một thứ rất cần thiết trong các dự án, ví dụ như chúng ta cần phải ghi log ra một file để sau này có thể kiểm soát được.
- Và ngôn ngữ lập trình nào cũng hỗ trợ chúng ta làm việc với file
- Có 3 loại file thông dụng là file văn bản, hình ảnh và âm thanh. Trong nội dung này, chúng ta chỉ hướng đến đối tượng File văn bản (*.txt)
- Mỗi loại file có module xử lý khác nhau:
 - Module pandas dùng xử lý file CSV, Excel
 - Module PyPDF2 dùng xử lý file PDF
 - Module json dùng xử lý file JSON
 - ...

MỞ FILE

- Python cung cấp hàm `open()` trả về một đối tượng File mà được sử dụng với các hàm khác.
- Với File đã mở, bạn có thể thực hiện các hoạt động đọc, ghi, ... trên File đó. Cú pháp của hàm `open()` là:

`file_object = open(file_name [, access_mode][, buffering])`

- **file_name**: Đối số `file_name` là một giá trị chuỗi chứa tên của các file mà bạn muốn truy cập.
- **access_mode**: Các `access_mode` xác định các chế độ của file được mở ra như `read`, `write`, `append`,... Đây là thông số tùy chọn và chế độ truy cập file mặc định là `read (r)`.
- **buffering**: là thông số đệm cho file mặc định thì nó sẽ là 0.

MỞ FILE

Mode	Mô tả
r	Mở file chỉ để đọc
r+	Mở file để đọc và ghi
rb	Mở file trong chế độ đọc cho định dạng nhị phân, đây là chế độ mặc định. Con trỏ tại phần bắt đầu của file
rb+	Mở file để đọc và ghi trong định dạng nhị phân. Con trỏ tại phần bắt đầu của file
w	Tạo một file mới để ghi, nếu file đã tồn tại thì sẽ bị ghi mới
w+	Tạo một file mới để đọc và ghi, nếu file tồn tại thì sẽ bị ghi mới
wb	Mở file trong chế độ ghi trong định dạng nhị phân. Nếu file đã tồn tại, thì ghi đè nội dung của file đó, nếu không thì tạo một file mới
wb+	Mở file để đọc và ghi trong định dạng nhị phân. Nếu file tồn tại thì ghi đè nội dung của nó, nếu file không tồn tại thì tạo một file mới để đọc và ghi
a	Mở file để ghi thêm vào cuối file, nếu không tìm thấy file sẽ tạo mới một file để ghi mới
a+	Mở file để đọc và ghi thêm vào cuối file, nếu không tìm thấy file sẽ tạo mới một file để đọc và ghi mới
ab	Mở file trong chế độ append trong chế độ nhị phân. Con trỏ là ở cuối file nếu file này đã tồn tại. Nếu file không tồn tại, thì tạo một file mới để ghi
ab+	Mở file trong chế độ đọc và append trong định dạng nhị phân. Con trỏ file tại cuối nếu file đã tồn tại. Nếu không tồn tại thì tạo một file mới để đọc và ghi

Thuộc tính của FILE

Thuộc tính	Mô tả
<code>file.closed</code>	Trả về True nếu file đã đóng, ngược lại là False
<code>file.mode</code>	Trả về chế độ truy cập của file đang được mở
<code>file.name</code>	Trả về tên của file

Thuộc tính của FILE

- Ví dụ: Mở file myfile.txt

Mở file

```
file = open("myfile.txt", "w+")
```

```
print("Tên của file là: ", file.name)
```

```
print("File có đóng hoặc không? : ", file.closed)
```

```
print("Chế độ mở file : ", file.mode)
```

```
Tên của file là:  myfile.txt
File có đóng hoặc không? :  False
Chế độ mở file :  w+
```

Đóng File

- Khi bạn đã thực hiện xong các hoạt động trên file thì cuối cùng bạn cần đóng file đó.
- Sử dụng phương thức **close()** để đóng một file.

fileObject.close()

Mở file

```
file = open("myfile.txt", "r")
```

Đóng file

```
file.close()
```

Đọc FILE

- Phương thức **read** : `fileObject.read([size])`
- Phương thức này trả về một chuỗi có kích thước bằng size. Nếu không truyền size thì toàn bộ nội dung của file sẽ được đọc.
- Ví dụ: tạo file myfile.txt có nội dung

```
hello world
1 2 3 4 5 6
My name is Tom
```

```
f = open('myfile.txt', 'r')
str= f.read()
print (str)
f.close()
```

```
In [7]: runfile('C:/Users/sony/Desktop/python_co_ban/
test/mymodule/test_file.py', wdir='C:/Users/sony/
Desktop/python_co_ban/test/mymodule')
hello world
1 2 3 4 5 6
My name is Tom
```

Đọc FILE

- **Phương thức readline:** `fileObject.readline()`
- Phương thức này cho phép đọc một dòng trong file và trả về chuỗi.

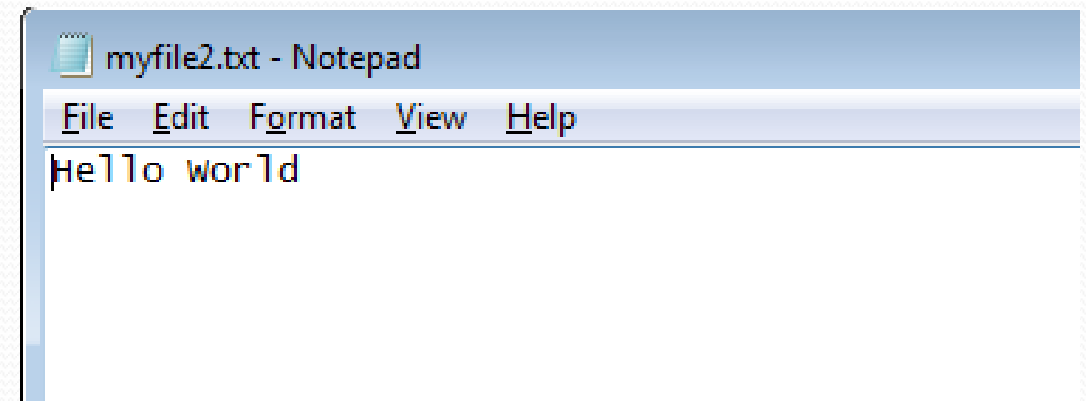
```
f = open('myfile.txt', 'r')  
line1 = f.readline()  
line2 = f.readline()  
print ('Dòng 1: ', line1)  
print ('Dòng 2: ', line2)  
f.close()
```

```
Dòng 1:  hello world  
Dòng 2:  1 2 3 4 5 6
```

Ghi FILE

- Để ghi một file ta cần mở file bằng cú pháp để ghi và sử dụng phương thức write để ghi vào: **fileObject.write(string)**
- Phương thức này cho phép ghi một chuỗi có nội dung là string vào vị trí của con trỏ trong file.

```
# Mở file
file = open("myfile2.txt", "w+")
file.write("Hello World");
# Đóng file
file.close()
```



Ghi FILE

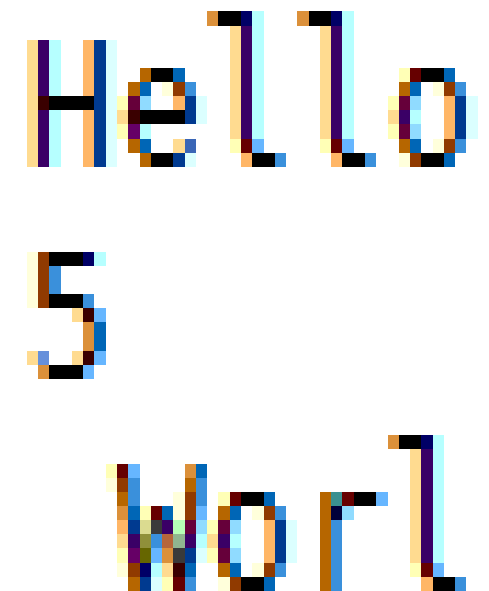
❖ Vị trí File trong Python

- Phương thức **tell()** nói cho bạn biết vị trí hiện tại bên trong file. Nói cách khác, việc đọc và ghi tiếp theo sẽ diễn ra trên các byte đó.
- Phương thức **seek(offset[, from])** thay đổi vị trí hiện tại bên trong file.
 - Tham số *offset* chỉ số byte để được di chuyển.
 - Tham số **from** xác định vị trí tham chiếu mà từ đó byte được di chuyển.
 - Nếu *from* được thiết lập là 0 nghĩa là sử dụng phần đầu file như là vị trí tham chiếu và 1 nghĩa là sử dụng vị trí hiện tại như là vị trí tham chiếu và nếu là 2 thì sử dụng phần cuối file như là vị trí tham chiếu.

Ghi FILE

- Vị trí File trong Python

```
# Mở file
file = open("myfile2.txt", "r+") # Hello World
str = file.read(5)
print(str)
# Kiểm tra con trỏ hiện tại
vitri = file.tell()
print(vitri)
# Đặt lại vị trí con trỏ tại vị trí hiện tại
vitri = file.seek(0, 1)
str = file.read(5)
print(str)
# Đóng file
file.close()
```



Hello
5
Worl

Ghi FILE

- Vị trí File trong Python

```
# Mở file
file = open("myfile2.txt", "rb+") # Hello World
str = file.read(5)
print(str)
# Kiểm tra con trỏ hiện tại
vitri = file.tell()
print(vitri)

vitri = file.seek(-5,2)
str = file.read(5)
print(str)
# Đóng file
file.close()
```

```
b'Hello'
5
b'World'
```