



Hàm - function

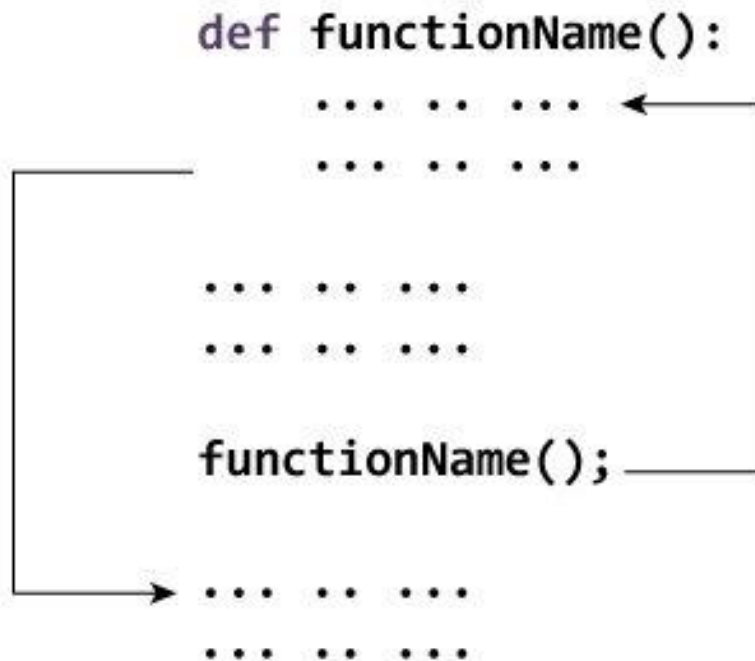
Hàm

- ❑ Trong Python, một hàm là một nhóm các câu lệnh có liên quan với nhau để thực hiện một tác vụ cụ thể
- ❑ Các hàm giúp chia nhỏ chương trình của chúng tôi thành các phần nhỏ hơn và theo mô-đun
- ❑ Khi chương trình của chúng tôi ngày càng phát triển lớn hơn, các chức năng làm cho nó có tổ chức và dễ quản lý hơn
- ❑ Tránh lặp lại và làm cho mã có thể được sử dụng lại
- ❑ Dễ phát hiện và sửa lỗi
- ❑ Dễ bảo trì phát triển



Định nghĩa hàm

```
def function_name(parameters):  
    """docstring"""  
    statement(s)
```



Cú pháp khai báo chung

- ❑ Từ khóa **def** đánh dấu sự bắt đầu của tiêu đề hàm
- ❑ **Tên hàm để xác định duy nhất hàm**
- ❑ Việc đặt tên hàm **function_name** tuân theo các quy tắc viết mã định danh
- ❑ Các tham số **parameters** mà qua đó chúng ta truyền các giá trị cho một hàm, chúng là tùy chọn.
- ❑ **Dấu hai chấm :** để đánh dấu phần cuối của tiêu đề hàm
- ❑ Chuỗi tài liệu tùy chọn **docstring** để mô tả chức năng hoạt động
- ❑ Một hoặc nhiều câu lệnh python hợp lệ tạo nên **thân hàm**. **Các câu lệnh phải có cùng mức thụt lề**
- ❑ Một câu lệnh trả về tùy chọn để trả về một giá trị từ hàm **return values**



Định nghĩa hàm

```
def greet(name):  
    """  
    This function greets to  
    the person passed in as  
    a parameter  
    """  
    print("Hello, " + name + ". Good  
morning!")  
  
greet('Paul')  
print(greet.__doc__)
```

```
→Hello, Paul. Good morning!  
This function greets to  
    the person passed in as  
    a parameter
```



Định nghĩa hàm

```
# function call
greet('Paul')

# function definition
def greet(name):
    """
    This function greets to
    the person passed in as
    a parameter
    """
    print("Hello, " + name + ". Good
morning!")

# Error: name 'greet' is not defined
```

- ❑ Trong 1 module ([file_name.py](#)) hàm phải định nghĩa trước khi gọi hàm
- ❑ Có thể định nghĩa nhiều hàm
- ❑ Hàm có thể định nghĩa bên trong 1 hàm (nested function)



Lệnh return

```
return [expression_list]
```

- ❑ Dùng để trả về dữ liệu kết quả mong muốn sau khi thoát khỏi hàm
- ❑ Thoát khỏi hàm và kết thúc hàm
- ❑ Quay lại nơi gọi hàm
- ❑ `expression_list` các giá trị trả về
- ❑ Nếu không có biểu thức nào trong câu lệnh hoặc `return` không có trong thân hàm, thì hàm sẽ trả về đối tượng `None` theo mặc định
- ❑ Giải phóng tất cả các biến cục bộ bên trong hàm
- ❑ Dữ liệu `return` là các kiểu object trong python hay do lập trình viên định nghĩa (`class object`, `list`, `tuple...`)



Lệnh return

```
def thong_ke(lst):  
    return sum(lst) / len(lst), min(lst), max(lst)  
  
lst = [15, 9, 55, 41, 35, 20, 62, 49]  
avg, min_val, max_val = thong_ke(lst)  
  
print("avg, min_val, max_val =", avg, min_val, max_val)
```

→ avg, min_val, max_val = 35.75 9 62

Tâm vực (phạm vi) của biến

Python Variable Scope

module_abc.py

```
s = "Python" # Biến s cấp module
def f():
    print(s)
    ss = "Java" # Biến ss cấp hàm f
    def f1():
        print(s)
        print(ss)
        def f11():
            print(s)
            print(ss)
            return
        f11()
    f1()
    return
f()
```



Python
Python
Java
Python
Java



Tâm vực (phạm vi) của biến

Python Variable Scope

- ❑ Biến cấp module có phạm vi hoạt động trên cả module và bên trong các hàm khai báo trong module (cả hàm lồng bên trong) → phạm vi toàn cục global
- ❑ Nếu bên trong hàm có biến cùng tên biến bên ngoài được tạo ra → Python xem như biến cục bộ local và biến toàn cục bị che khuất không thể truy xuất
- ❑ Tương tự biến cấp hàm có tầm hoạt động trong hàm và các hàm cấp con khai báo bên trong nó....



Tâm vực (phạm vi) của biến

Python Variable Scope

```
s = "Python" # Biến s cấp module
def f():
    s = "Java" # glocal
    def f1():
        print(s)
        def f11():
            print(s)
            return
        f11()
        return
    f1()
    return
f()
print(s)
```



Java

Java

Python



Tâm vực (phạm vi) của biến

Python Variable Scope

- ❑ Từ khóa **global** là để khai báo biến trong hàm là tham chiếu ra biến cấp module

```
a=[1,2]
s="Python"
def f():
    global a
    global s
    a+= [200]
    s=s+" 3.10"
f()
print(a)
print(s)
```



[1, 2, 200]
Python 3.10



Tâm vực (phạm vi) của biến

Python Variable Scope

```
name = "Python" #global
def foo():
    name = "Java" # Our local variable
    print(name)
    def bar():
        nonlocal name          # Reference
        name in the upper scope
        name = 'C++' # Overwrite this
        variable
        print(name)
    # Calling inner function
    bar()
    # Printing local variable
    print(name)

foo()
print(name)
```

❑ Từ khóa **nonlocal** là khai báo biến tham chiếu ra hàm cấp cao hơn nhưng không phải cấp global (module)

❑ Dành cho hàm lồng nhau, cấp bên trong



Java
C++
C++
Python

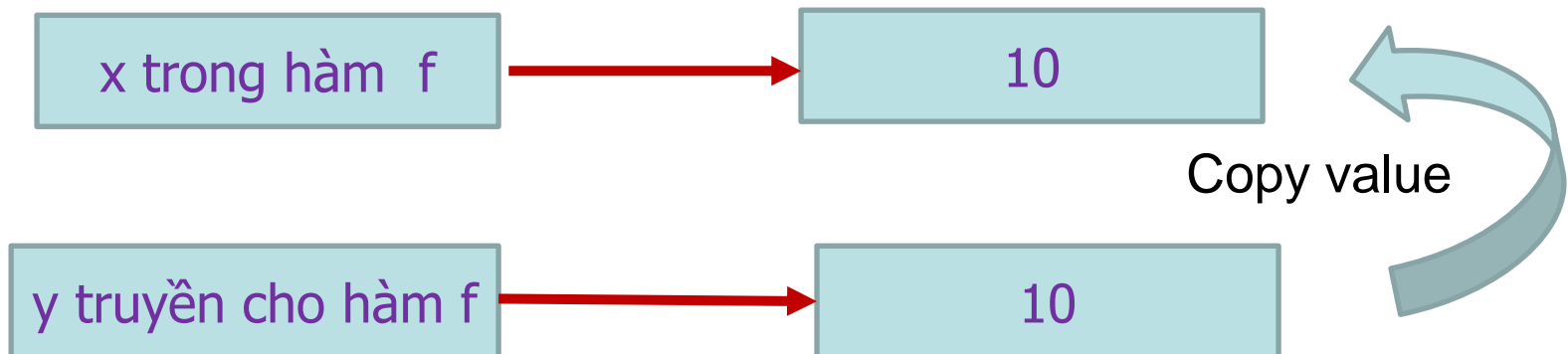


Truyền tham số

Passing Arguments

- ❑ Call by Value: mọi sự thay đổi tham số **x** bên trong hàm không ảnh hưởng đến biến truyền **y** bên ngoài

```
def f(x):  
    #Làm gì đó với x  
    pass  
  
y = [10]  
f(y) #Passing Arguments: truyền y cho f
```

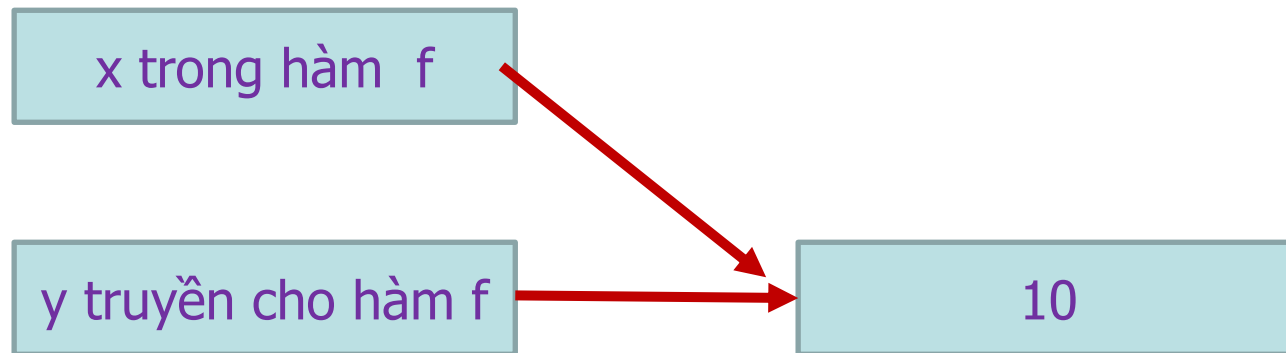


Truyền tham số

Passing Arguments

- ❑ Call by Reference: mọi sự thay đổi tham số **x** bên trong hàm ảnh hưởng trực tiếp đến biến truyền **y** bên ngoài

```
def f(x):  
    #Làm gì đó với x  
    pass  
  
y = [10]  
f(y) #Passing Arguments: truyền y cho f
```



Truyền tham số

Passing Arguments

- ❑ Python ban đầu hoạt động giống như call-by-reference
- ❑ Nếu trong thân hàm có lệnh làm thay đổi địa chỉ tham số thì Python chuyển sang call-by-value. Nghĩa là, một biến cục bộ x sẽ được tạo ra
- ❑ Lưu ý:
 - ✓ Các biến immutable objects sẽ đổi địa chỉ khi thay đổi giá trị
 - ➔ Không bao giờ bị thay đổi giá trị sau khi gọi hàm vì nếu bên trong hàm có đổi giá trị thì biến global sẽ tạo ra trong hàm
 - ✓ Biến mutable objects chỉ thay đổi địa chỉ khi gán một object khác. Khi cập nhật thêm, xóa phần tử thì địa chỉ không thay đổi địa chỉ
 - ➔ Có thể bị thay đổi giá trị nếu bên trong hàm có cập nhật thêm, xóa phần tử



Truyền tham số

Passing Arguments

```
def ref_demo(x):  
    print("x=",x, " id=",id(x))  
    x=42  
    print("x=",x, " id=",id(x))  
y=10  
print("y=",y, " id=",id(y))  
ref_demo(y)  
print("y=",y, " id=",id(y))
```



```
y= 10 id= 140711986036064  
x= 10 id= 140711986036064  
x= 42 id= 140711986037088  
y= 10 id= 140711986036064
```



Truyền tham số

Passing Arguments

```
def no_side_effects(cities):  
    print(cities)  
    cities = cities + ["Birmingham", "Bradford"]  
    print(cities)  
locations = ["London", "Leeds", "Glasgow",  
             "Sheffield"]  
no_side_effects(locations)  
print(locations)
```



```
['London', 'Leeds', 'Glasgow', 'Sheffield']  
['London', 'Leeds', 'Glasgow', 'Sheffield', 'Birmingham', 'Bradford']  
['London', 'Leeds', 'Glasgow', 'Sheffield']
```



Truyền tham số

Passing Arguments

```
def side_effects(cities):  
    print(cities)  
    cities += ["Birmingham", "Bradford"]  
    print(cities)  
  
locations = ["London", "Leeds", "Glasgow",  
             "Sheffield"]  
side_effects(locations)  
print(locations)
```



```
['London', 'Leeds', 'Glasgow', 'Sheffield']  
['London', 'Leeds', 'Glasgow', 'Sheffield', 'Birmingham', 'Bradford']  
['London', 'Leeds', 'Glasgow', 'Sheffield', 'Birmingham', 'Bradford']
```



Tham số với giá trị mặc định

Default Argument Values

- ❑ Hàm với giá trị mặc định cho các tham số nhất định= >tùy chọn cho người dùng.
- ❑ Có thể gọi hàm bằng cách chuyển các tham số tùy chọn đó, hoặc chỉ chuyển các tham số bắt buộc mà thôi.
- ❑ Có hai cách chính để truyền các tham số tùy chọn
 - Không sử dụng các đối số từ khóa
 - Bằng cách sử dụng các đối số từ khóa.

```
list_name.sort(reverse=..., key=... )
```

Tham số với giá trị mặc định

Default Argument Values

```
# a list of numbers
my_numbers = [10, 8, 3, 22, 33, 7, 11, 100, 54]
#sort list in-place in ascending order
my_numbers.sort()
#print modified list
print(my_numbers)
#output
#[3, 7, 8, 10, 11, 22, 33, 54, 100]
```

```
# a list of numbers
my_numbers = [10, 8, 3, 22, 33, 7, 11, 100, 54]
#sort list in-place in ascending order
my_numbers.sort(reverse=True)
#print modified list
print(my_numbers)
#output
# [100, 54, 33, 22, 11, 10, 8, 7, 3]
```



Tham số với giá trị mặc định

Default Argument Values

```
programming_languages = ["Python", "Swift", "Java",  
"C++", "Go", "Rust"]  
programming_languages.sort(key=len)  
print(programming_languages)
```

→ ['Go', 'C++', 'Java', 'Rust', 'Swift', 'Python']

Tham số với giá trị mặc định

Default Argument Values

```
def my_print(s, new_line = True):  
    if new_line:  
        print(s)  
    else:  
        print(s, end='')  
  
my_print('Các bạn thân mến,')  
my_print('Ngày mai là sinh nhật mình,', False)  
my_print('thân mời các bạn đến dự tại nhà mình.')  
my_print('Rất vui được gặp các bạn lúc 12h00,',  
new_line = False)  
my_print(' tại nhà mình.')  
my_print('Thân.')
```

Các bạn thân mến,
Ngày mai là sinh nhật mình, thân mời các
bạn đến dự tại nhà mình.
Rất vui được gặp các bạn lúc 12h00, tại
nhà mình.
Thân.



Command Line Arguments

- Các đối số được đặt sau tên của chương trình (file.py) khi gọi lệnh thực thi bằng dòng lệnh

`python file.py [danh sách tham số]`

- Python cung cấp nhiều cách khác nhau để xử lý các loại đối số này, phổ biến nhất là:

- ☐ [Using sys.argv](#)

- ☐ [Using getopt module](#)

- ☐ [Using argparse module](#)

Using sys.argv

- ❑ Mô-đun sys cung cấp các hàm và biến được sử dụng để thao tác các phần khác nhau của môi trường thời gian chạy Python.
- ❑ sys.argv là list chứa Mục đích chính của nó là: Nó là một danh sách bắt các đối số dòng lệnh
- ❑ len (sys.argv) cung cấp số lượng đối số dòng lệnh.
- ❑ sys.argv [0] là tên của file Python được gọi thi hành tại dòng lệnh



Using sys.argv

tong_day_so.py

```
import sys
# total arguments
n = len(sys.argv)
print("Total arguments passed:", n)
# Arguments passed
print("\nName of Python script:", sys.argv[0])
print("\nArguments passed:", end = " ")
for i in range(1, n):
    print(sys.argv[i], end = " ")
# Addition of numbers
Sum = 0
# Using argparse module
for i in range(1, n):
    Sum += int(sys.argv[i])
print("\n\nResult:", Sum)
```

```
PS C:\Users\dangq> python tong_day_so.py
1 2 3 4 5
Total arguments passed: 6
Name of Python script: tong_day_so.py
Arguments passed: 1 2 3 4 5
Result: 15
```



Using argparse module

test.py

```
import argparse
import sys
parser = argparse.ArgumentParser(description='Chương trình ABC')
parser.add_argument('--cuda', default=True, type=bool,
help='Use cuda for inference')
parser.add_argument('--show_time', default=False,
action='store_true', help='show processing time')
args = parser.parse_args()

if args.cuda:
    print('Chạy với CUDA')
else:
    print('Chạy với without CUDA')
```

```
python test.py --cuda=True
Chạy với CUDA
```



Using argparse module

```
test.py --help  
usage: test.py [-h] [--cuda CUDA] [--show_time]
```

Chương trình ABC

optional arguments:

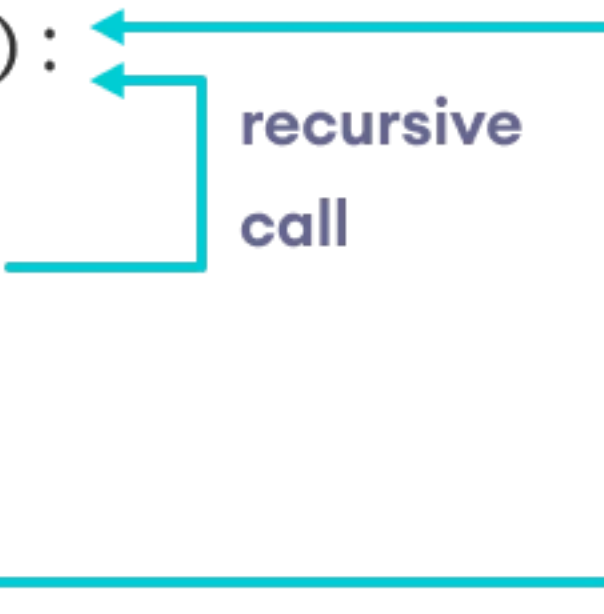
- h, --help show this help message and exit
- cuda CUDA Use cuda for inference
- show_time show processing time



Đệ qui, Recursion

- Hàm có lệnh gọi chính nó trong thân hàm.

```
def recurse():  
    ...  
    recurse()  
    ...  
  
recurse()
```



recursive call

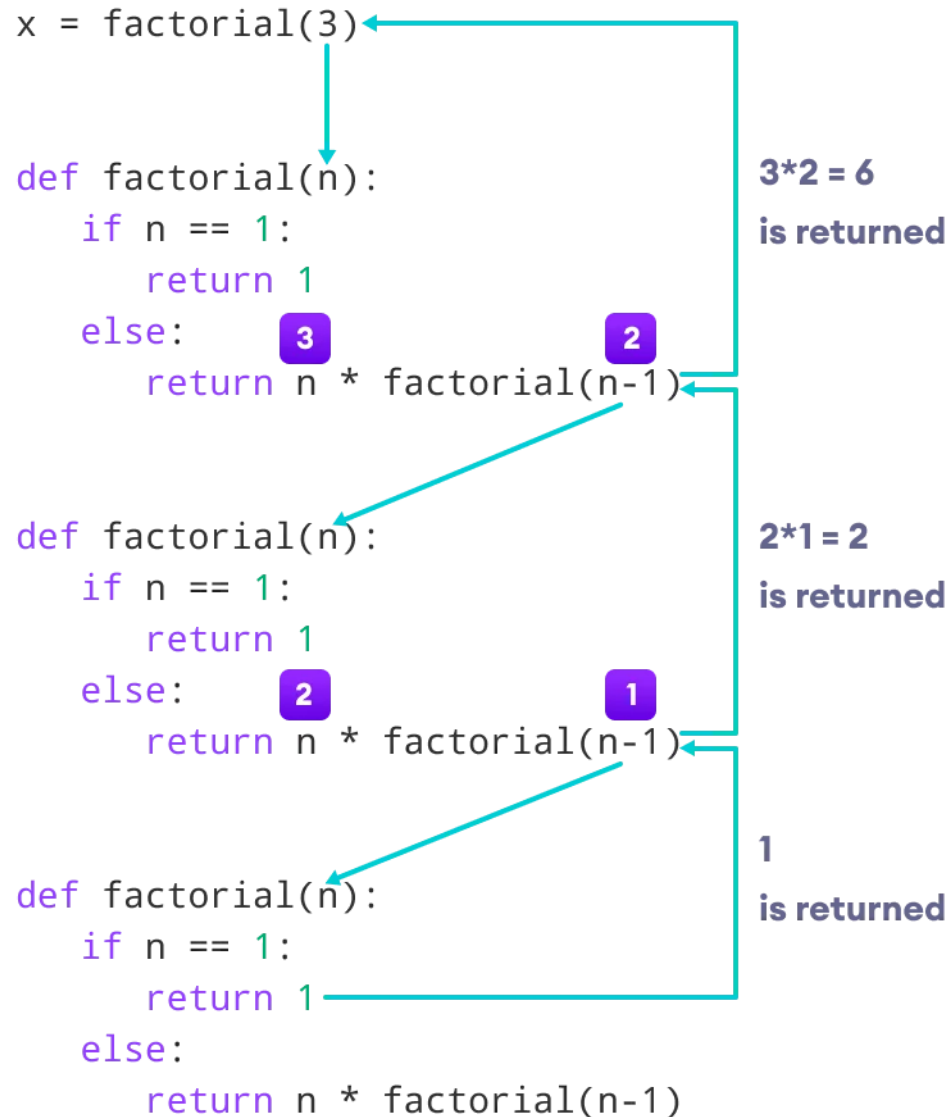
Đệ qui, Recursion

```
def factorial(x):  
    """This is a recursive function  
    to find the factorial of an integer"""  
  
    if x == 1:  
        return 1  
    else:  
        return (x * factorial(x-1))  
  
num = 3  
print("The factorial of", num, "is", factorial(num))
```

→ The factorial of 3 is 6



Đệ qui, Recursion



Đệ qui, Recursion

Ưu điểm của Đệ qui

1. Các hàm đệ qui làm cho mã trông ngắn gọn
2. Một nhiệm vụ phức tạp có thể được chia thành các bài toán con đơn giản hơn bằng cách sử dụng đệ qui
3. Tạo trình tự với đệ qui dễ dàng hơn so với sử dụng một số phép lặp lồng nhau.

Nhược điểm của Đệ qui

1. Đôi khi logic đằng sau đệ qui rất khó theo dõi
2. Cuộc gọi đệ qui rất tốn kém (không hiệu quả) vì chúng chiếm nhiều bộ nhớ và thời gian
3. Các hàm đệ qui rất khó gỡ lỗi

Import module ngoài

mod.py

```
s = "If Comrade Napoleon says it, it must be  
right."  
a = [100, 200, 300]  
  
def foo(arg):  
    print(f'arg = {arg}')  
class Foo:  
    pass
```

Several objects are defined in mod.py:

- s (a string)
- a (a list)
- foo() (a function)
- Foo (a class)



Import module ngoài

```
>>> import mod
>>> print(mod.s)
If Comrade Napoleon says it, it must be right.
>>> mod.a
[100, 200, 300]
>>> mod.foo(['quux', 'corge', 'grault'])
arg = ['quux', 'corge', 'grault']
>>> x = mod.Foo()
>>> x
<mod.Foo object at 0x03C181F0>
```



Import module ngoài

Khi trình thông dịch thực hiện câu `import mod`, trình thông dịch sẽ tìm kiếm `mod.py` trong danh sách các thư mục được tập hợp từ các nguồn sau

- ❑ Thư mục file chương trình python chạy
- ❑ Thư mục hiện tại nếu trình thông dịch đang được chạy chế độ tương tác
- ❑ Danh sách các thư mục có trong biến môi trường PYTHONPATH, nếu nó được đặt (PYTHONPATH phụ thuộc vào hệ điều hành, dạng giống biến PATH)
- ❑ Danh sách thư mục phụ thuộc vào cài đặt được định cấu hình tại thời điểm Python được cài đặt

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



Import module ngoài

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

```
['c:\\Users\\dangq', 'C:\\Program  
Files\\WindowsApps\\PythonSoftwareFoundation.Python.3.7_3.7.2544  
.0_x64__qbz5n2kfra8p0\\python37.zip', 'C:\\Program  
Files\\WindowsApps\\PythonSoftwareFoundation.Python.3.7_3.7.2544  
.0_x64__qbz5n2kfra8p0\\DLLs', 'C:\\Program  
Files\\WindowsApps\\PythonSoftwareFoundation.Python.3.7_3.7.2544  
.0_x64__qbz5n2kfra8p0\\lib', 'C:\\Program  
Files\\WindowsApps\\PythonSoftwareFoundation.Python.3.7_3.7.2544  
.0_x64__qbz5n2kfra8p0',  
'C:\\Users\\dangq\\AppData\\Local\\Packages\\PythonSoftwareFoundat  
ion.Python.3.7_qbz5n2kfra8p0\\LocalCache\\local-  
packages\\Python37\\site-packages', 'C:\\Program  
Files\\WindowsApps\\PythonSoftwareFoundation.Python.3.7_3.7.2544  
.0_x64__qbz5n2kfra8p0\\lib\\site-packages']
```



Import module ngoài

```
import mod  
print(mod.__file__)  
print(mod)
```



c:\Users\dangq\mod.py

<module 'mod' from 'c:\\Users\\dangq\\mod.py'>



Import module ngoài

```
import <module_name>
from <module_name> import <name(s)>
from <module_name> import *
from <module_name> import <name> as
<alt_name>
from <module_name> import <name> as
<alt_name>[, <name> as <alt_name> ...]
```



Python Packages

- ☐ Giả sử bạn đã phát triển một ứng dụng rất lớn bao gồm nhiều mô-đun.
- ☐ Khi số lượng mô-đun tăng lên, sẽ khó theo dõi tất cả chúng nếu chúng bị dồn vào một thư mục. Điều này đặc biệt đúng nếu chúng có tên hoặc chức năng tương tự.
- ☐ Bạn có thể mong muốn có một phương tiện để nhóm và sắp xếp chúng.
- ☐ Tương tự như cách mà các mô-đun giúp tránh xung đột giữa các tên biến toàn cục, các gói giúp tránh xung đột giữa các tên mô-đun
- ☐ Việc tạo một gói khá đơn giản, vì nó sử dụng cấu trúc thư mục phân cấp vốn có của hệ điều hành, tên thư mục là tên package, phân cấp bằng dấu chấm



Python Packages

app1

main.py

```
from simple_package import a, b
from simple_package.utils import my_math
a.bar()
b.foo()
print(my_math.cong(100, 200))
```

→ Hello from bar
Hello from foo
300

simple_package

a.py

```
def bar():
    print("Hello from bar")
```

b.py

```
def foo():
    print("Hello from foo")
```

utils

my_math.py

```
def cong(a, b):
    return a + b

def nhan(a, b):
    return a * b
```



Python Packages

app1

main.py

simple_package

utils

a.py

```
def bar():  
    print("Hello from bar")
```

b.py

```
def foo():  
    print("Hello from foo")
```

example.py

```
import sys  
import os  
parent = os.path.dirname("D:\\app1\\")  
# adding the parent directory to  
# the sys.path.  
sys.path.append(parent)  
from simple_package import a, b  
a.bar()  
b.foo()  
from simple_package.utils import my_math  
print(my_math.cong(100,200))
```



Hàm lamda

`lambda arguments : expression`

- ❑ Một hàm lambda là một hàm ẩn danh
- ❑ Một hàm lambda có thể nhận bất kỳ số lượng đối số
- ❑ Chỉ có thể có một biểu thức trả về (return)
- ❑ Sử dụng trong các tình huống cần hàm xử lý nhỏ gọn mà không cần định nghĩa trước.
- ❑ Nhược điểm có thể tồn tại nhiều nơi cho một chức năng giống nhau nào đó → chuyển thành hàm định nghĩa một nơi duy nhất.



decorator

- ❑ Một hàm là một **đối tượng (object)** của lớp **function**.
- ❑ Một biến có thể tham chiếu đến một hàm.
- ❑ Đối số của một hàm có thể là một hàm khác.
- ❑ Kết quả trả về của một hàm có thể là một hàm khác.
- ❑ Có thể sử dụng các cấu trúc dữ liệu như hash table, list,... để lưu trữ hàm.

```
def cong(a,b):  
    return a+b  
  
x=cong  
print(x(20,30))
```



Hàm lambda

```
x = lambda a, b : a * b
print(x(5, 6))
x = lambda a, b: a + b
print(x(5, 6))
x = lambda a, b, c : max(a, b, c)
print(x(5, 6, 2))
```



30

11

6

Hàm lambda

```
x = lambda a, b : a * b
print(x(5, 6))
x = lambda a, b: a + b
print(x(5, 6))
x = lambda a, b, c : max(a, b, c)
print(x(5, 6, 2))
```



30

11

6



Hàm lambda

```
unfilteredList = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
filteredList = filter(lambda x: x < 7 and x > 2,
unfilteredList)
print(list(filteredList))
```

→ [3, 4, 5, 6]

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

```
unfilteredList = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
filteredList = [x for x in unfilteredList if (x > 2 and x
< 7)]
print(filteredList)
```

→ [3, 4, 5, 6]



Hàm lambda

```
danh_sach = [  
    {"name": "Ty", "mark": 8, "yob": 2000},  
    {"name": "Suu", "mark": 6, "yob": 1990},  
    {"name": "Dan", "mark": 10, "yob": 2002},  
    {"name": "Meo", "mark": 7, "yob": 1976}  
]  
  
danh_sach.sort(key=lambda student: student["mark"])  
print(danh_sach)
```

```
[{'name': 'Suu', 'mark': 6, 'yob': 1990}, {'name': 'Meo', 'mark': 7, 'yob':  
1976}, {'name': 'Ty', 'mark': 8, 'yob': 2000}, {'name': 'Dan', 'mark': 10,  
'yob': 2002}]
```

Hàm lambda

```
danh_sach = [  
    {"name": "Ty", "mark": 8, "yob": 2000},  
    {"name": "Suu", "mark": 6, "yob": 1990},  
    {"name": "Dan", "mark": 10, "yob": 2002},  
    {"name": "Meo", "mark": 7, "yob": 1976}  
]  
  
danh_sach.sort(key=lambda student: student["yob"])  
print(danh_sach)
```

```
[{'name': 'Meo', 'mark': 7, 'yob': 1976}, {'name': 'Suu', 'mark': 6, 'yob':  
1990}, {'name': 'Ty', 'mark': 8, 'yob': 2000}, {'name': 'Dan', 'mark': 10,  
'yob': 2002}]
```



Dùng hàm định nghĩa thay lambda (dễ hiểu, dễ bảo trì nâng cấp)

```
def key_for_sort(student):  
    return student["yob"]  
danhsach = [  
    {"name": "Ty", "mark": 8, "yob": 2000},  
    {"name": "Suu", "mark": 6, "yob": 1990},  
    {"name": "Dan", "mark": 10, "yob": 2002},  
    {"name": "Meo", "mark": 7, "yob": 1976}  
]  
danhsach.sort(key=key_for_sort)  
print(danhsach)
```

```
[{'name': 'Meo', 'mark': 7, 'yob': 1976}, {'name': 'Suu', 'mark': 6, 'yob':  
1990}, {'name': 'Ty', 'mark': 8, 'yob': 2000}, {'name': 'Dan', 'mark': 10,  
'yob': 2002}]
```



Q & A

Thank you!

