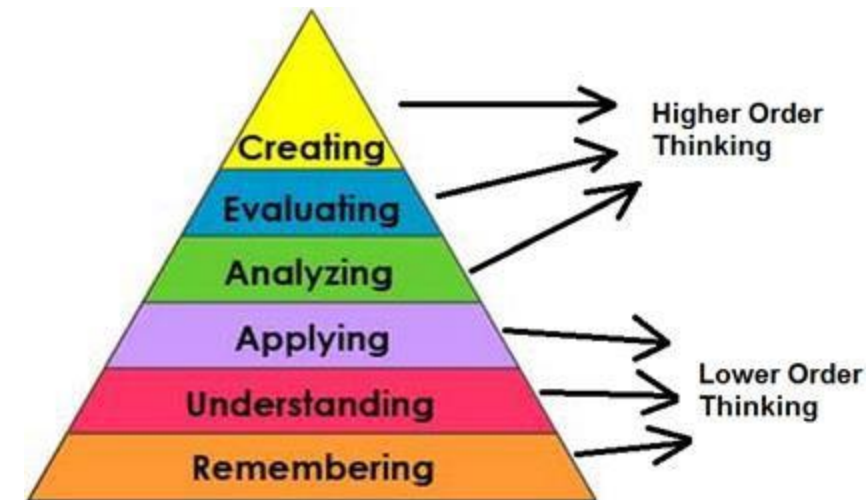


Kỹ Thuật Lập Trình Python

CHƯƠNG 5 HÀM



- Vận dụng các hàm có sẵn
- Phân tích xây dựng hàm
- Áp dụng biến toàn cục và cục bộ trong thiết kế hàm và chương trình



1. Hàm được tích hợp sẵn (Hàm có sẵn)
(Built-in Functions)
2. Xây dựng hàm
3. Các loại tham số
4. Phạm vi biến và hàm
5. Hàm đệ quy
6. Hàm không tên Lambda

Hàm được tích hợp sẵn

Giới thiệu

- Trong trình thông dịch của Python, một số hàm đã được xây dựng và tích hợp sẵn gọi là Built-in Function
- Ví dụ: hàm `print()`, `input()`, `list()`...
- Các xuất thông tin mô tả hàm:

```
print(func_name.__doc__)
```

- Ví dụ:

```
print(sorted.__doc__)
```

Return a new list containing all items from the iterable in ascending order.

A custom key function can be supplied to customize the sort order, and the reverse flag can be set to request the result in descending order.

Hàm được tích hợp sẵn

<u>abs()</u>	<u>delattr()</u>	<u>hash()</u>	<u>memoryview()</u>	<u>set()</u>
<u>all()</u>	<u>dict()</u>	<u>help()</u>	<u>min()</u>	<u>setattr()</u>
<u>any()</u>	<u>dir()</u>	<u>hex()</u>	<u>next()</u>	<u>slice()</u>
<u>ascii()</u>	<u>divmod()</u>	<u>id()</u>	<u>object()</u>	<u>sorted()</u>
<u>bin()</u>	<u>enumerate()</u>	<u>input()</u>	<u>oct()</u>	<u>staticmethod()</u>
<u>bool()</u>	<u>eval()</u>	<u>int()</u>	<u>open()</u>	<u>str()</u>
<u>breakpoint()</u>	<u>exec()</u>	<u>isinstance()</u>	<u>ord()</u>	<u>sum()</u>
<u>bytearray()</u>	<u>filter()</u>	<u>issubclass()</u>	<u>pow()</u>	<u>super()</u>
<u>bytes()</u>	<u>float()</u>	<u>iter()</u>	<u>print()</u>	<u>tuple()</u>
<u>callable()</u>	<u>format()</u>	<u>len()</u>	<u>property()</u>	<u>type()</u>
<u>chr()</u>	<u>frozenset()</u>	<u>list()</u>	<u>range()</u>	<u>vars()</u>
<u>classmethod()</u>	<u>getattr()</u>	<u>locals()</u>	<u>repr()</u>	<u>zip()</u>
<u>compile()</u>	<u>globals()</u>	<u>map()</u>	<u>reversed()</u>	<u>__import__()</u>
<u>complex()</u>	<u>hasattr()</u>	<u>max()</u>	<u>round()</u>	

Ref: <https://docs.python.org/3/library/functions.html>

Hàm được tích hợp sẵn

Đối tượng iterator là gì?

- Là đối tượng cho phép duyệt qua tuần tự từng phần tử => **lặp được.**
 - Phương thức `__iter__()`: Trả về chính đối tượng iterator.
 - Phương thức `__next__()`: Trả về phần tử tiếp theo trong đối tượng iterator.
 - Đối tượng iterator: danh sách (list), bộ (tuple), chuỗi (string), tập hợp (set) và từ điển (dictionary)
 - Phương thức `next()` để lấy từng phần tử từ đối tượng iterator cho đến khi gặp ngoại lệ `StopIteration`.
- => sẽ áp dụng sau khi học xong oop.

Hàm được tích hợp sẵn

enumerate(*iterable*, *start=0*)

- Tạo bộ đếm cho *iterable*
- *iterable*: string, list, tuple..., iterator hoặc bất cứ đối tượng hỗ trợ iteration.
- *start*: giá trị bắt đầu đếm. Giá trị mặc định của *start* là 0.

```
week = ['Thứ 2', 'Thứ 3', 'Thứ 4', 'Thứ 5', 'Thứ 6', 'Thứ 7', 'Chủ nhật']  
e = enumerate(week)
```

```
print(e)  
print(list(enumerate(week)))  
print(dict(enumerate(week)))  
print(tuple(enumerate(week)))  
print(set(enumerate(week)))
```

Hàm được tích hợp sẵn

filter(*function*, *iterable*)

- Xây dựng một vòng lặp từ các item của *iterable*.
- *iterable*: danh sách (list), bộ (tuple), chuỗi (string), tập hợp (set) và từ điển (dictionary)
- Lọc ra các item xét theo điều kiện của hàm *function*
- filter chỉ trả về những giá trị mà điều kiện trong hàm *function* chấp nhận (chân trị True).

```
ds = [None, -2, -3, [], 4, 0, 6, (), -7, ", 8, {}]  
d = filter(bool, ds)  
print(list(d))
```

[-2, -3, 4, 6, -7, 8]

Hàm được tích hợp sẵn

zip(*iterables)

- Tạo một trình lặp tổng hợp các phần tử từ mỗi vòng lặp.
- Trả về Iterator của Tuple

```
x = [1, 2, 3]
y = [4, 5, 6]
zipped = zip(x, y)
zipped
```

```
<zip at 0x18e20936508>
```

```
list(zipped)
```

```
[(1, 4), (2, 5), (3, 6)]
```

Hàm được tích hợp sẵn

`slice(start, stop, step)`

- Trả về một slice object.
- Slice object xác định cách cắt một đối tượng được chỉ định bằng `range()`.
- Đối tượng cần cắt có thể là string, bytes, tuple, list, range

Hàm được tích hợp sẵn

Ví dụ

```
seq = [3, 2, 'A', 8, 6, 8]  
tup = (12, 'C', 7, 9, 'Z')  
str = 'Hello Python'
```

```
slice(4)
```

```
slice(None, 4, None)
```

```
slice(2, 4)
```

```
slice(2, 4, None)
```

```
slice(1, 4, 2)
```

```
slice(1, 4, 2)
```

Tìm khác biệt giữa hàm slice và toán tử slice “:”

```
seq[slice(4)]
```

```
[3, 2, 'A', 8]
```

```
tup[slice(1, 4, 2)]
```

```
('C', 9)
```

```
str[slice(2, 4)]
```

```
'll'
```

Hàm được tích hợp sẵn

sorted(iterable, /, *, key=None, reverse=False)

- **iterable** có thể là seq.
- **reverse** False => tăng, True => giảm

```
x = [4,2,9,6,8,5,2]
```

```
sorted(x, reverse=False)
```

```
[2, 2, 4, 5, 6, 8, 9]
```

Hàm được tích hợp sẵn

map(function, iterable, ...)

- Duyệt tất cả các phần tử của **iterable** thông qua hàm **function** và trả về một **map object** sau khi thực thi.
- Kết quả **map object** có thể dùng **list()**, hoặc **set()** để tạo danh sách hoặc tập hợp.

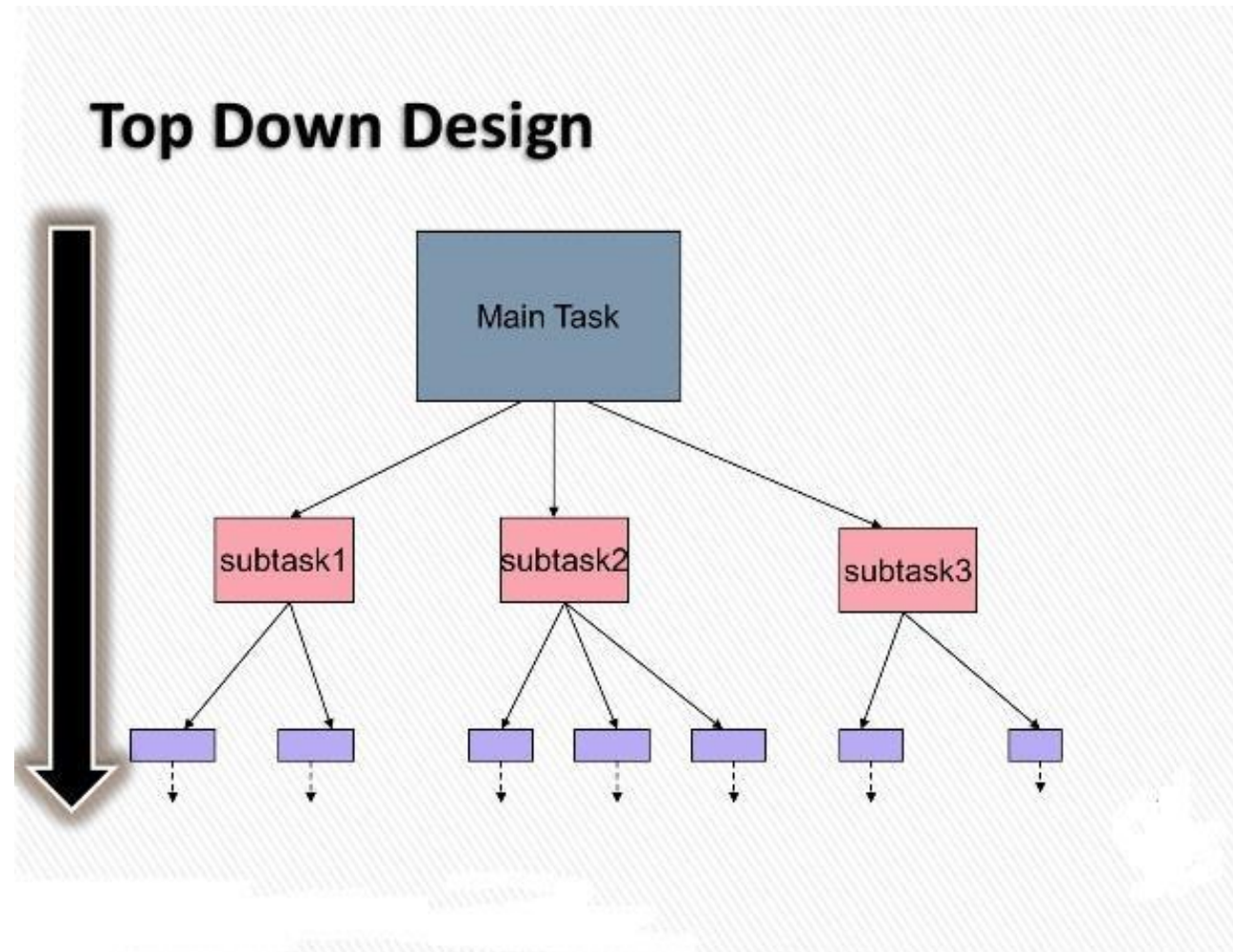
```
def tinh(x):  
    return x + 3  
  
num1 = [1, 2, 3, 4, 5, 6]  
num2 = map(tinh, num1)  
  
print(list(num2))
```

Hàm - function (viết tắt func)

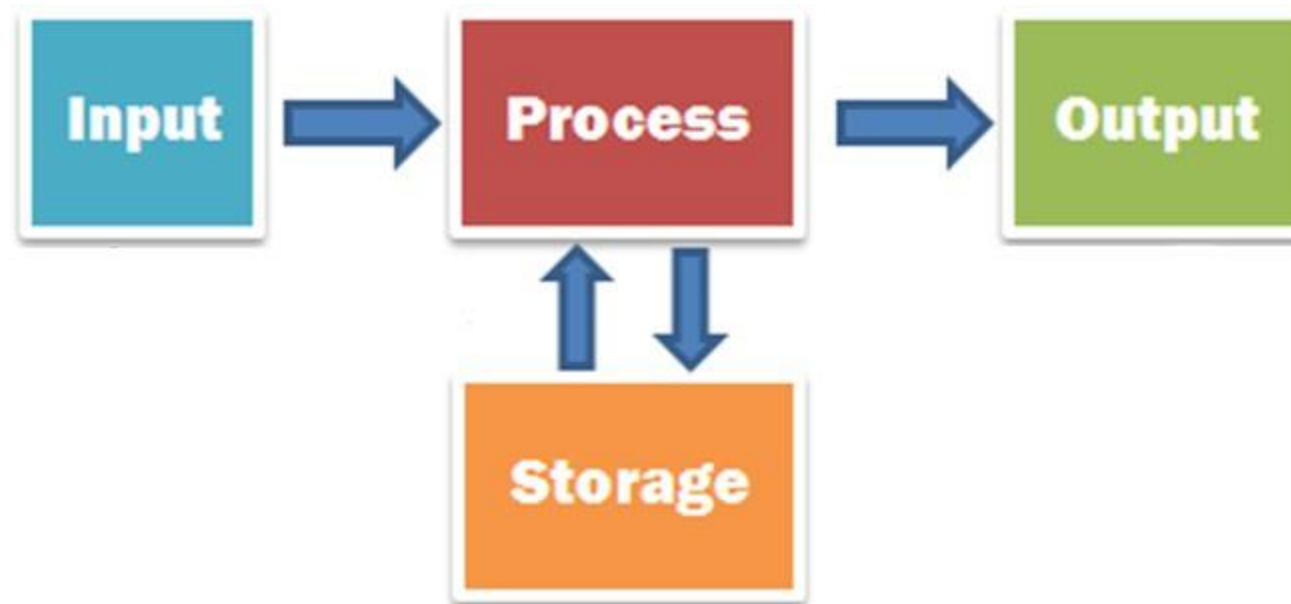
func là gì?

- Hàm sẽ chứa một khối các dòng **code/script** có hệ thống và tổ chức.
- Hàm có thể tái sử dụng để thực hiện một công việc có liên quan nhau.
- Hàm dùng để thực hiện, xử lý các công việc đã được **chia nhỏ ra từ công việc lớn**, việc phân chia này giúp người lập trình có khả năng kiểm soát lỗi và dễ phát hiện để sửa lỗi.
- Khuyến nghị khi thiết kế hàm, LTV nên tuân theo nguyên tắc sau
I-P-O
tức là Input-Process-Output.

Nguyên tắc thiết kế hàm (cơ bản)



Nguyên tắc thiết kế hàm (cơ bản)



Cú pháp chung

```
def func_name(<para#1, para#2, ..., para#n>):  
    """ docstring: Mô tả hàm """  
    <Lệnh hoặc Khối lệnh trong hàm>  
    return <Tùy chọn. Dùng khi cần hàm trả về giá trị>
```

Output nhiều giá trị

```
def func_name(<para#1, para#2, ..., para#n>):  
    """ docstring: Mô tả hàm """  
    <Lệnh hoặc Khối lệnh trong hàm>  
    return x, y, x, ...
```

Chú ý:

Có thể khai báo kèm thêm kiểu dữ liệu cho tham số và kiểu dữ liệu của hàm

Ví dụ minh họa

Hàm tính cộng, trừ, nhân, chia của hai số nhập vào.

```
def tinh(a, b):  
    cong = a + b  
    tru = a - b  
    nhan = a * b  
    chia = a / b  
  
    return cong, tru, nhan, chia  
  
print(tinh(9, 10))
```

```
def tinh(a:int, b:float) -> tuple:  
    cong = a + b  
    tru = a - b  
    nhan = a * b  
    chia = a / b  
  
    return cong, tru, nhan, chia  
  
print(tinh(9, 10))
```

Ví dụ minh họa

Hàm tính giai thừa của một số n

```
def giai_thua (n):  
    x = 1  
    for i in range(1, n + 1):  
        x = x * i  
    return x
```

```
giai_thua(3)
```

6

```
giai_thua(5)
```

120

Ví dụ minh họa

Hàm vừa tính **giai thừa**
và vừa tính **tổng** của một số **n**

```
def giai_thua_va_tong (n):  
    x = 1  
    y = 0  
    for i in range(1, n + 1):  
        x = x * i  
        y = y + i  
    return x, y
```

```
giai_thua_va_tong(3)
```

```
(6, 6)
```

```
giai_thua_va_tong(5)
```

```
(120, 15)
```

Ví dụ minh họa

Hàm in ra dãy số Fibonacci

```
def fib(n): # write Fibonacci series up to n
    """Print a Fibonacci series up to n."""
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()
```

```
fib(2018)
```

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

Cách gọi docstring

tên_hàm.__doc__

- Kiểu dữ liệu trả về là chuỗi mô tả.

Ví dụ minh họa

```
def fib(n):  
    """Print a Fibonacci series up to n."""  
    a, b = 0, 1  
    while a < n:  
        print(a, end=' ')  
        a, b = b, a+b  
    print()
```

```
fib(1000)
```

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

```
fib.__doc__
```

```
'Print a Fibonacci series up to n.'
```


Tạo hàm bên trong một hàm

- Cú pháp cơ bản:

```
def func_outside(<para#1, para#2, ..., para#n>):  
    """ Outside (Hàm ngoài) """  
    <Lệnh hoặc Khối lệnh trong hàm>
```

```
    def func_inside(<para#1, para#2, ..., para#n>):  
        """ Inside (Hàm trong) """  
        <Lệnh hoặc Khối lệnh trong hàm>  
        return <Tùy chọn. Dùng khi cần hàm trả về giá trị>
```

```
return <Tùy chọn. Dùng khi cần hàm trả về giá trị>
```

Tạo hàm bên trong một hàm

- Ví dụ minh họa:

```
def outside(x, y):  
    z = x - y  
    print('Kết quả tính toán: ', x)  
  
    def inside(a):  
        if a%2==0:  
            print('Kiểm tra ', a, 'là số chẵn')  
        else:  
            print('Kiểm tra ', a, 'là số lẻ')  
    inside(x)
```

```
outside(2, 6)
```

```
Kết quả tính toán: 2  
Kiểm tra 2 là số chẵn
```

Tham số (Parameter/Argument)

Giới thiệu

- Tham số theo vị trí (Positional Arguments)
- Tham số mặc định (Default Arguments)
- Tham số theo keyword (Keyword Arguments - kwargs)
- Tham số tùy biến dạng *args (Arbitrary Arguments - *args)
- Tham số tùy biến dạng **kwargs (Keyword Arguments - **kwargs)

Tham số theo vị trí (Positional Arguments)

Tham số theo vị trí được truyền theo thứ tự từ trái qua phải

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

```
add(7, 9)
```

16

```
add(9, 8)
```

17

Tham số mặc định (Default Arguments)

Tham số mặc định là tham số sẽ có giá trị trong lúc tạo hàm

- Khi gọi hàm, nếu không cung cấp giá trị cho tham số thì ***giá trị mặc định*** sẽ được sử dụng.

```
1 def add(a = 1, b = 2):  
2     return a + b
```

```
1 add(3, 6)
```

9

```
1 add(3)
```

5

```
1 add()
```

3

Tham số theo keyword (Keyword Arguments - kwargs)

Tham số theo keyword được truyền theo tên tham số

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

```
add(7, 9)
```

16

```
add(9, 8)
```

17

```
add(b=8, a=9)    <= Truyền bằng keyword thì không cần theo thứ tự
```

17

Tham số tùy biến dạng `*args` (Arbitrary Arguments - `*args`)

Tham số cho phép nhận một số lượng tham số không xác định trước

Dạng `*args`: truyền nhiều tham số vị trí.

```
def add(*n):  
    r = 0  
    for i in n:  
        r = r + i  
    return r
```

```
add(9, 8, 3)
```

20

```
add(9)
```

9

```
add(9, 8)
```

17

Tham số tùy biến dạng ****kwargs** (Keyword Arguments - ****kwargs**)

Tham số cho phép nhận một số lượng tham số không xác định trước
Dạng **kwargs**: Truyền nhiều tham số theo từ khóa.**

```
def demo_kwargs(**kwargs):  
    print(kwargs)
```

```
demo_kwargs(k=2, e=6)
```

```
{'k': 2, 'e': 6}
```

```
demo_kwargs(k=2, e=6, t=1)
```

```
{'k': 2, 'e': 6, 't': 1}
```


Kết hợp dạng **args* và ***kwargs*

```
def demo1(name, *args, **kwargs):  
    print('Giá trị name    = ', name)  
    print('Giá trị args    = ', args)  
    print('Giá trị kwargs = ', kwargs)
```

```
demo1(2, 3, 5)
```

```
Giá trị name    = 2  
Giá trị args    = (3, 5)  
Giá trị kwargs  = {}
```

```
demo1(6, k=2, j=3, t=5)
```

```
Giá trị name    = 6  
Giá trị args    = ()  
Giá trị kwargs  = {'k': 2, 'j': 3, 't': 5}
```

Kết hợp dạng **args* và ***kwargs*

```
def demo2(*args, name, **kwargs):  
    print('Giá trị name    = ', name)  
    print('Giá trị args    = ', args)  
    print('Giá trị kwargs  = ', kwargs)
```

```
demo2(2, 3, 5, name=1)
```

```
Giá trị name    = 1  
Giá trị args    = (2, 3, 5)  
Giá trị kwargs  = {}
```

```
demo2(k=2, j=3, t=5, name=2)
```

```
Giá trị name    = 2  
Giá trị args    = ()  
Giá trị kwargs  = {'k': 2, 'j': 3, 't': 5}
```

Tham số trong hàm

Bài tập áp dụng

- Dùng các kỹ thuật tham số dạng `*args`, `**kwargs`... thiết kế hàm tính tổng, tích của nhiều số.
- Viết hàm dùng các kỹ thuật tham số dạng `*args`, `**kwargs`... tính **chu vi** hình vuông, chữ nhật, tròn.
- Viết hàm dùng các kỹ thuật tham số dạng `*args`, `**kwargs`... tính **chu vi hoặc diện tích** hình vuông, chữ nhật, tròn.

Phạm vi biến đối với hàm

- Biến toàn cục - global variable
- Biến cục bộ - local variable
- Biến nonlocal

Biến toàn cục - global variable

- Biến được khai báo bên ngoài hàm hoặc lớp.
- Được truy cập và sử dụng trong toàn bộ chương trình.

```
x = 36 #Biến toàn cục

def funcA():
    print('Biến toàn cục x = ', x)

if __name__ == '__main__':
    funcA()
```

Biến toàn cục - global variable

- Biến toàn cục có thể được truy cập trong một hàm.
- Nếu muốn thay đổi giá trị của biến toàn cục bên trong một hàm thì sử dụng từ khóa global.

```
x = 36 # Biến toàn cục

def funcB():
    print('Biến toàn cục x =', x) #Lỗi
    x = 369

if __name__ == '__main__':
    funcB()
    print('Giá trị biến toàn cục x =', x)
```

```
x = 36 # Biến toàn cục

def funcB():
    global x
    print('Biến toàn cục x =', x)
    x = 369

if __name__ == '__main__':
    funcB()
    print('Giá trị biến toàn cục x =', x)
```

Biến cục bộ - local variable

- Khai báo bên trong hàm hoặc một khối lệnh.
- Được truy cập và sử dụng trong phạm vi hàm hoặc khối lệnh chứa biến.

```
def funcC():  
    y = 369 # Biến cục bộ  
    print('Giá trị biến cục bộ y =', y)  
  
if __name__ == '__main__':  
    funcC()  
    print(y) # Gọi bên ngoài bị lỗi
```

Biến nonlocal

Xảy ra khi định nghĩa hàm bên trong hàm

```
def func():  
    x = 36    # Biến cục bộ của hàm func  
  
    def inner_func():  
        nonlocal x    # Khai báo nonlocal để tham chiếu đến x bên ngoài  
        x = 369        # Thay đổi giá trị của biến x bên ngoài  
        print('Giá trị bên trong inner_func, x =', x)  
  
    inner_func()  
    print('Giá trị tại hàm func, x =', x)  
  
if __name__ == '__main__':  
    func()
```


Reference type

- list
- dict
- set
- object

Phạm vi biến và hàm

```
# Function definition is here
def changeme( mylist ):
    "This changes a passed list into this function"
    mylist.append([1,2,3,4])
    print("Values inside the function: ", mylist)
    return

# Now you can call changeme function
mylist = [10,20,30]
changeme( mylist )
print("Values outside the function: ", mylist)
```

Kết quả:

```
Values inside the function:  [10, 20, 30, [1, 2, 3, 4]]
Values outside the function:  [10, 20, 30, [1, 2, 3, 4]]
```

Phạm vi biến và hàm

```
# Function definition is here
def changeme( mylist ):
    "This changes a passed list into this function"
    mylist = [1,2,3,4]
    print("Values inside the function: ", mylist)
    return

# Now you can call changeme function
mylist = [10,20,30]
changeme( mylist )
print("Values outside the function: ", mylist)
```

Kết quả, đối số mylist là một biến nội bộ của hàm changeme. Thay đổi mylist trong hàm không ảnh hưởng đến mylist. Kết quả như sau:

```
Values inside the function:  [1, 2, 3, 4]
Values outside the function: [10, 20, 30]
```

Phạm vi biến và hàm

```
1  x_public = 10
2  def funcA():
3      x_public = 18
4      print("Trong hàm A: ", x_public)
5
6  def funcB():
7      global x_public
8      x_public = 2
9      print("Trong hàm B: ", x_public)
10
11 def funcC():
12     global x_public
13     print("Trong hàm C: ", x_public)
14
15 print("Ngoài hàm: ", x_public)
16
17 def funcD():
18     x = "local var"
19     def funcE():
20         nonlocal x
21         x = "nonlocal var"
22         print("Inside:", x)
23
24     funcE()
25     print("Outside:", x)
```

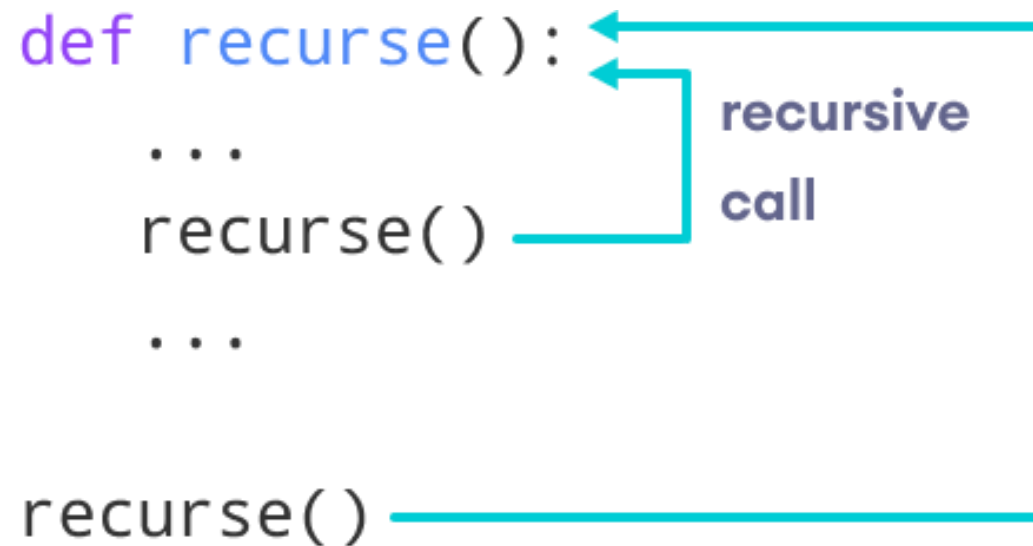
```
▶ if name == 'main':
    funcA()
    funcB()
    funcC()
    print("Cuối cùng 1: ", x_public)
    x_public = 88
    print("Cuối cùng 2: ", x_public)
    funcC()
```

Đệ quy - Recursion

Đệ quy?

- Hàm tự gọi lại chính hàm đó một hoặc nhiều lần thì gọi là đệ quy.
- Phải có điều kiện chấm dứt trong quá trình đệ quy để tránh việc hàm đệ quy trở thành một vòng lặp vô hạn.

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



recursive call

Ví dụ minh họa

- Dùng đệ quy để in ra dãy số từ 10 giảm về 1.

```
def recursion(x):  
    x += 1  
    if x < 10:  
        recursion(x)  
    print(x, end=' ')
```

```
recursion(0)
```

10 9 8 7 6 5 4 3 2 1

Ví dụ minh họa

- Dùng đệ quy để in ra dãy số từ 1 đến n.

```
def print_numbers(n):  
    if n >= 1:  
        print_numbers(n - 1) # Gọi lại chính hàm với đối số nhỏ hơn  
        print(n)  
  
print_numbers(5)
```

Lambda function

- Hàm không cần đặt tên và được định nghĩa ngắn gọn.

Cú pháp

lambda arguments : expression

Giải thích:

- Có thể có nhiều arguments. Ví dụ: x, hoặc x, y
- expression là biểu thức, hoặc **kết hợp hàm**. Ví dụ: $x^{**2} + y$

Lambda function

Ví dụ minh họa

- Hàm tính giá trị $y = 12x + 1$

```
f = lambda x: 12*x + 1
```

```
f(3)
```

37

```
f(5)
```

61

```
f(2)
```

25

Lambda function

Ví dụ minh họa

- Hàm lambda gọi một hàm khác

```
def cong(x = 1, y = 2):  
    return x + y  
  
f = lambda a, b, c: cong(a, b) + c**2  
  
ket_qua = f(3, 6, 9)  
  
print(ket_qua)
```

Lambda function

Ví dụ minh họa

- Hàm lambda gọi một hàm khác.
- Vừa định nghĩa vừa gọi

```
def cong(x = 1, y = 2):  
    return x + y  
  
f = (lambda a, b, c: cong(a, b) + c**2)(3, 6, 9)  
  
print(f)
```

Nhắc lại: Ứng dụng for để tạo list mới

- Tạo một danh sách bằng cách kết hợp lệnh for và biểu thức

Cú pháp – còn gọi là cú pháp comprehension

`new_list = [expression for item in iterable if condition]`

- expression: biểu thức để tính toán giá trị của từng phần tử trong list mới.
- item: biến được sử dụng để lặp qua từng phần tử trong list ban đầu.
- condition: điều kiện kiểm tra mà các item phải thỏa để được thêm vào new_list

Ví dụ

```
ds = [2, 4, 6, 8, 10]
new_ds = [item - 1 for item in ds]
print(new_ds)
```

Tạo list

```
def tinh(x = 1):  
    return 2 * x + 3  
  
ds = [1, 3, 5, 7, 9, 2, 4, 6, 8, 10]  
new_ds = [(lambda x: x + tinh(x))(item) for item in ds if item % 2 == 0]  
  
print(new_ds) #[9, 15, 21, 27, 33]
```

Sắp xếp list

```
num = [2, 8, 9, 3, 5, 1]  
new_num = sorted(num, key=lambda x: x)  
  
print(new_num)  #[1, 2, 3, 5, 8, 9]
```

```
num = [2, 8, 9, 3, 5, 1]  
new_num = sorted(num, key=lambda x: -x)  
  
print(new_num)  #[9, 8, 5, 3, 2, 1]
```

Lọc các giá trị

```
num = [2, 8, 9, 3, 5, 1]
new_num = list(filter(lambda x: x % 2 == 1, num))

print(new_num) #[9, 3, 5, 1]
```

Lambda kết hợp với map

```
num = [1, 3, 5, 7, 9]

new_num = map(lambda x: x + 3, num)

print(list(new_num)) # [4, 6, 8, 10, 12]
```


BÀI #1

- Viết hàm tính căn bậc n từ một số thực.
- Viết hàm tính giá trị bình phương của một số dương.
- Viết hàm kiểm tra một số nhập vào là số chẵn có giá trị âm. Đúng trả về True. Sai trả về False.
- Viết hàm kiểm tra một số nhập vào: nếu là số âm có giá trị lẻ thì trả về -1, nếu là số dương có giá trị chẵn thì trả về 1, trường hợp khác trả về 0
- Viết hàm kiểm tra giá trị nhập vào phải thuộc đoạn $[-89, 90]$, nếu sai bắt nhập lại.

BÀI #2

Cho danh sách sau:

[('Tiền Giang', 63), ('Long An', 62), ('Vĩnh Long', 64), ('Bình Dương', 60)]

Sắp xếp các phần tử trong danh sách tăng dần theo số.

Bài #3

Cho dãy tên **seqA**:

- Viết hàm khởi tạo giá trị tự động cho **seqA** gồm **N** phần tử các số nguyên (âm, dương) và số thực (âm, dương).
 - N** chọn ngẫu nhiên từ 30 đến 80.
 - Các giá trị số nguyên, số thực chọn ngẫu nhiên từ -79 đến 39. Giá trị số thực được làm tròn 2 số thập phân.
- Viết hàm kiểm tra kiểu dữ liệu từng phần tử.
- Viết hàm thống kê số lượng phần tử có trong **seqA**.
- Viết hàm sắp xếp dãy **seqA** thành dãy **seqB** tăng dần.
- Viết hàm tính trung bình các phần tử trong **seqA**.

Yêu cầu SV thiết kế các datatype lưu trữ kết quả trả về

BÀI #3 (tiếp theo)

6. Viết hàm tính giá trị trung bình giữa hai phần tử nằm giữa trong dãy **seqB** khi N chẵn. Khi N lẻ, thì hàm tính trả về giá trị nằm giữa.
7. Viết hàm tính khoảng cách giữa hai giá trị max, min trong dãy **seqA** hoặc **seqB**.
8. Viết hàm so sánh các kết quả của của câu 5 và câu 6.

Lưu ý: Anh/chị được sử dụng các lib/func hỗ trợ.

Yêu cầu SV thiết kế các datatype lưu trữ kết quả trả về

Bài tập QLVN cơ bản

Viết chương trình quản lý nhân viên cho một công ty có tối đa 300 nhân viên. Công ty gồm 2 loại nhân viên: nhân viên văn phòng và nhân viên bán hàng. Công ty cần quản lý các thông tin sau của nhân viên: mã nhân viên, họ và tên, lương cơ bản, lương hằng tháng.

Lương hằng tháng tính bằng công thức sau:

- NV văn phòng = lương cơ bản + số ngày là việc * 150_000đ
- NV bán hàng = lương cơ bản + số sản phẩm * 18_000đ

Chương trình đáp ứng các yêu cầu sau:

1. Khởi tạo tự động các nhân viên. Chỉ cung cấp: maNV, hoTen, luongCB
2. Xuất các nhân viên (mã nhân viên, họ tên, lương cơ bản, **hằng tháng**)
3. Tính lương các nhân viên
4. Tìm nhân viên theo mã nhân viên
5. Tìm nhân viên có lương hằng tháng cao nhất
6. Tìm nhân viên bán hàng có lương hằng tháng thấp nhất

Hướng dẫn

- Phạm vi kiến thức: Không dùng OOP
- Chọn data type cơ sở => chọn kiểu cho từng nhân viên
 - Để biểu diễn 1 nhân viên thì dùng kiểu gì?
 - Lưu trữ nhiều nhân viên thì dùng kiểu gì?
- Thiết kế hàm

Hỏi - Đáp

