



Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh

TRUNG TÂM TIN HỌC



Lập trình Python cơ bản

Bài 8: Phương thức / Hàm



Phòng Lập Trình - Mạng

2023

Nội dung

1. Định nghĩa
2. Khai báo và xây dựng hàm
3. Gọi hàm
4. Tầm vực của biến
5. Tham số
6. Hàm ẩn danh
7. Built-in Functions

1. Định nghĩa

- ❑ Hàm (Function) là tập hợp các dòng lệnh được viết để thực hiện một chức năng nào đó.
- ❑ Python cung cấp rất nhiều hàm xây dựng sẵn (built-in function), và người dùng cũng có thể tự xây dựng các hàm cho riêng mình (user-defined function).
- ❑ Hàm giúp mức độ tái sử dụng mã lệnh tốt hơn.

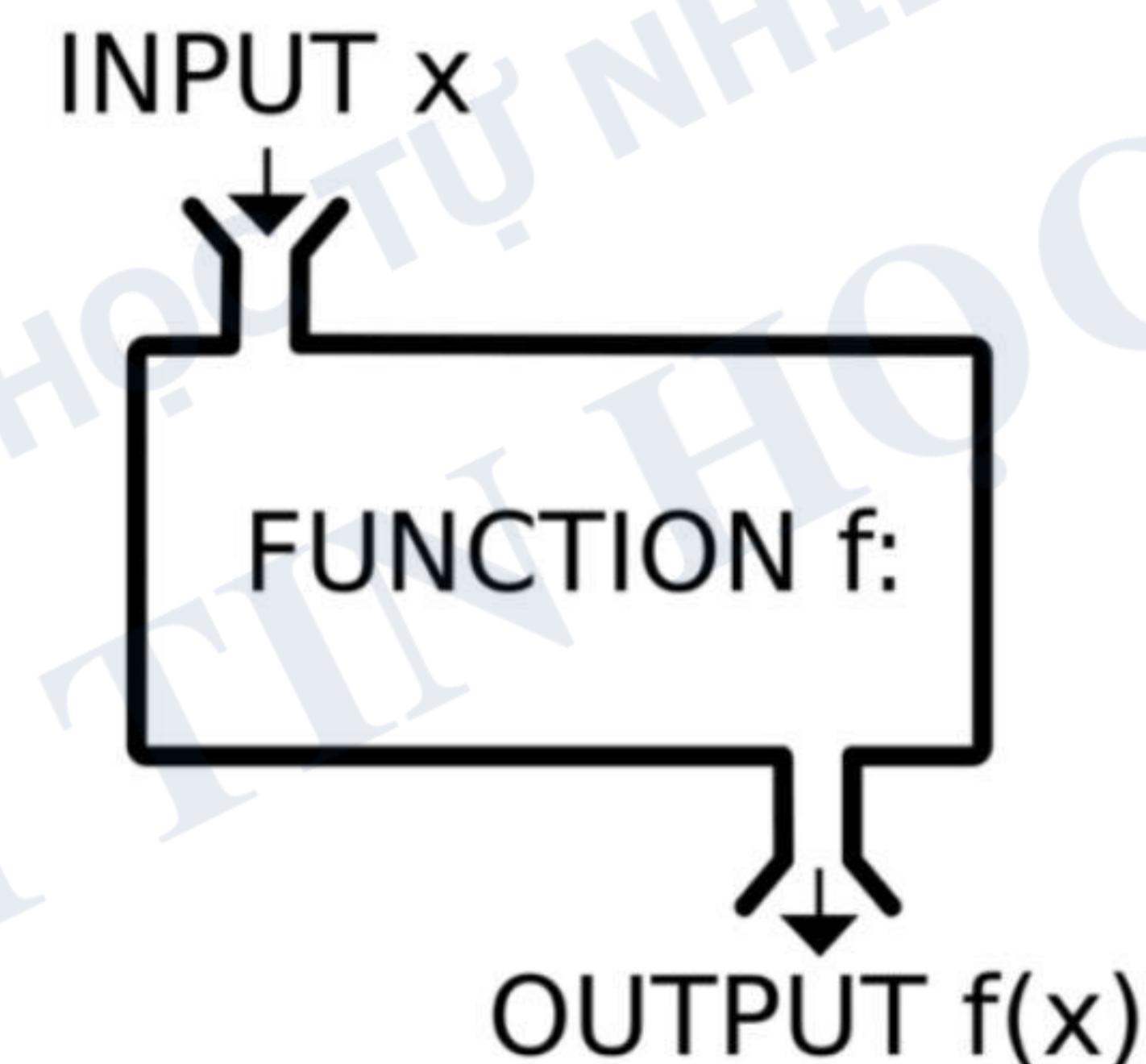
Nội dung

1. Định nghĩa
2. Khai báo và xây dựng hàm
3. Gọi hàm
4. Tầm vực của biến
5. Tham số
6. Hàm ẩn danh
7. Built-in Functions

2. Khai báo và xây dựng hàm

□ Cú pháp:

```
def function_name([parameters]):  
    # mô tả về function nếu cần thiết  
    # các dòng lệnh  
    [return [expression]]
```



□ Kết quả trả về dữ liệu (*return*) của hàm:

- Hàm có thể trả về nhiều giá trị.
- Nếu không trả về (không có lệnh *return*) thì mặc định sẽ trả về giá trị *None*.

2. Khai báo và xây dựng hàm

□ Ví dụ:

- Hàm không return:

```
def tinh_diem_trung_binh(hk1, hk2):  
    dtb = (hk1 + hk2 * 2) / 3  
    print('Điểm trung bình:', dtb)
```

- Hàm return 1 giá trị:

```
def tinh_diem_trung_binh(hk1, hk2):  
    dtb = (hk1 + hk2 * 2) / 3  
    return dtb
```

2. Khai báo và xây dựng hàm

□ Ví dụ:

- Hàm return nhiều giá trị:

```
def tinh_diem_trung_binh(hk1, hk2):  
    dtb = (hk1 + hk2 * 2) / 3  
    if dtb >= 5:  
        kq = 'Đậu'  
    else:  
        kq = 'Rớt'  
    return dtb, kq
```

Nội dung

1. Định nghĩa
2. Khai báo và xây dựng hàm
3. Gọi hàm
4. Tầm vực của biến
5. Tham số
6. Hàm ẩn danh
7. Built-in Functions

3. Gọi hàm

- Quy ước: Phần khai báo hàm phải nằm trước (bên trên) lời gọi hàm trong chương trình.

```
# Xây dựng hàm
def tinh_diem_trung_binh(hk1, hk2):
    dtb = (hk1 + hk2 * 2) / 3
    print('Điểm trung bình:', dtb)

# Gọi hàm
diem_hk1 = eval(input('Điểm HK1: '))
diem_hk2 = eval(input('Điểm HK2: '))
tinh_diem_trung_binh(diem_hk1, diem_hk2)
```

Điểm HK1: 8
Điểm HK2: 9
Điểm trung bình: 8.666666666666666

3. Gọi hàm

- Quy ước: Phần khai báo hàm phải nằm trước (bên trên) lời gọi hàm trong chương trình.

```
# Gọi hàm
diem_hk1 = eval(input('Điểm HK1: '))
diem_hk2 = eval(input('Điểm HK2: '))
tinh_diem_trung_binh(diem_hk1, diem_hk2)

# Xây dựng hàm
def tinh_diem_trung_binh(hk1, hk2):
    dtb = (hk1 + hk2 * 2) / 3
    print('Điểm trung bình:', dtb)
```

```
Điểm HK1: 8
Điểm HK2: 9
Traceback (most recent call last):
  File "l:/GIANG_DAY/_PORTAL/Chuyen_de/LDS1_2022/Demo/
    tinh_diem_trung_binh(diem_hk1, diem_hk2)
NameError: name 'tinh_diem_trung_binh' is not defined
```

Nội dung

1. Định nghĩa
2. Khai báo và xây dựng hàm
3. Gọi hàm
4. Tầm vực của biến
5. Tham số
6. Hàm ẩn danh
7. Built-in Functions

4. Tầm vực của biến

- ❑ **Biến toàn cục (global variable)**: có thể được truy cập trên toàn chương trình của tất cả các function.
- ❑ **Biến cục bộ (local variable)**: có thể được truy cập chỉ trong hàm mà biến được khai báo. Khi biến toàn cục và biến cục bộ có trùng tên, biến cục bộ sẽ được ưu tiên sử dụng.
- ❑ **Biến nonlocal**: là một biến có tầm vực cao hơn *local* và thấp hơn *global*. Thường dùng khai báo biến dạng này trong các hàm có chứa hàm con (*sub Function*). Khi đó, biến được khai báo *nonlocal* trong sub function, biến này được function ‘nhìn thấy’, nhưng chương trình chính thì không.

4. Tâm vực của biến

❑ Biến toàn cục:

```
s = 'Hello'      # Biến toàn cục
# Xây dựng hàm
def greeting():
    print('Lời chào:', s)

# Gọi hàm
greeting()
```

Lời chào: Hello

❑ Biến cục bộ:

```
s = 'Hello'      # Biến toàn cục
# Xây dựng hàm
def greeting():
    s = 'Hi'       # Biến cục bộ
    print('Lời chào:', s)

# Gọi hàm
greeting()
```

Lời chào: Hi

4. Tầm vực của biến

□ Từ khóa **global** trong Python:

- Được sử dụng khi cần thay đổi giá trị của biến toàn cục trong thân các hàm.
- Việc sử dụng từ khóa **global** bên ngoài các hàm sẽ không có ý nghĩa.

□ Ví dụ:

```
# Xây dựng hàm
def tong():
    print(c)

# Gọi hàm
c = 10
tong()
```

10

```
# Xây dựng hàm
def tong():
    '''Khai báo từ khóa global khi thay đổi giá trị
    của biến toàn cục bên trong function'''
    global c
    c = c + 5
    print(c)

# Gọi hàm
c = 10
tong()
```

15

4. Tâm vực của biến

- ☐ Từ khóa **global** trong các hàm lồng nhau (nested function):

```
def ham_1():
    x = 20
    def ham_2():
        global x
        x = 25
    print('x trước khi gọi ham_2:', x)

    # Gọi hàm ham_2
    ham_2()
    print('x sau khi gọi ham_2:', x)

# Gọi hàm ham_1
ham_1()
print('x từ chương trình chính:', x)

x trước khi gọi ham_2: 20
x sau khi gọi ham_2: 20
x từ chương trình chính: 25
```

4. Tâm vực của biến

- ☐ Từ khóa **nonlocal** trong các hàm lồng nhau (nested function):

```
def ham_ngoai():
    x = 'biến local'
    print('Trước khi gọi ham_trong():', x)
    def ham_trong():
        nonlocal x
        x = 'biến nonlocal'
        print('Bên trong ham_trong():', x)
    ham_trong()
    print('Sau khi gọi ham_trong():', x)
ham_ngoai()
```

Trước khi gọi ham_trong(): biến local

Bên trong ham_trong(): biến nonlocal

Sau khi gọi ham_trong(): biến nonlocal

Nội dung

1. Định nghĩa
2. Khai báo và xây dựng hàm
3. Gọi hàm
4. Tầm vực của biến
5. Tham số
6. Hàm ẩn danh
7. Built-in Functions

5. Tham số

□ Tham chiếu (pass by reference) và Tham trị (pass by value)

- Tham số có kiểu tham trị (pass by value):
 - Khi giá trị của tham số có bị thay đổi giá trị trong function thì sau khi function kết thúc, tham số vẫn nhận lại giá trị ban đầu.
 - Trong Python, mọi tham số luôn truyền theo kiểu tham trị.
- Tham số có kiểu tham chiếu (pass by reference):
 - Khi giá trị của tham số có bị thay đổi giá trị trong function thì sau khi function kết thúc, tham số sẽ nhận giá trị vừa được thay đổi đó.
 - Trong Python, các tham số có kiểu dữ liệu dạng danh sách như *list*, *tuple*, *dictionary* luôn là tham chiếu khi trong thân hàm có thực hiện các thao tác thêm, xóa phần tử của danh sách.

5. Tham số

□ Tham chiếu (pass by reference) và Tham trị (pass by value)

- Tham số có kiểu tham trị (pass by value):

```
# Xây dựng hàm tính bình phương các phần tử trong lst
def tinh_binh_phuong(danh_sach):
    for gia_tri in danh_sach:
        gia_tri *= gia_tri
    print('Danh sách trong hàm:', danh_sach)

# Gọi hàm, kiểm tra kết quả
list_so = [1, 2, 3]
print('Danh sách trước khi gọi hàm:', list_so)
tinh_binh_phuong(list_so)
print('Danh sách sau khi gọi hàm:', list_so)
```

Danh sách trước khi gọi hàm: [1, 2, 3]

Danh sách trong hàm: [1, 2, 3]

Danh sách sau khi gọi hàm: [1, 2, 3]

5. Tham số

□ Tham chiếu (pass by reference) và Tham trị (pass by value)

- Tham số có kiểu tham chiếu (pass by reference):

```
# Xây dựng hàm thực hiện thêm phần tử vào lst
def them_phan_tu(danh_sach):
    for gia_tri in range(4, 6):
        danh_sach.append(gia_tri)
    print('Danh sách trong hàm:', danh_sach)

# Gọi hàm, kiểm tra kết quả
list_so = [1, 2, 3]
print('Danh sách trước khi gọi hàm:', list_so)
them_phan_tu(list_so)
print('Danh sách sau khi gọi hàm:', list_so)
```

Danh sách trước khi gọi hàm: [1, 2, 3]

Danh sách trong hàm: [1, 2, 3, 4, 5]

Danh sách sau khi gọi hàm: [1, 2, 3, 4, 5]

5. Tham số

□ Phân loại tham số

- Có 4 loại tham số:
 - Tham số bắt buộc (Required argument)
 - Tham số từ khóa (Keyword argument)
 - Tham số mặc định (Default argument)
 - Tham số thay đổi không xác định (Variable-length argument)

5. Tham số

□ Tham số bắt buộc (Required argument)

- Là tham số khi truyền vào hàm phải đúng thứ tự và đúng số lượng tham số như khi hàm được xây dựng.

```
def tinh_luy_thua(a, b):  
    print(a, '^', b, '=', a ** b)
```

```
x = 5  
y = 3  
tinh_luy_thua(x, y)  
tinh_luy_thua(y, x)
```

```
5 ^ 3 = 125  
3 ^ 5 = 243
```

```
def tinh_luy_thua(a, b):  
    print(a, '^', b, '=', a ** b)
```

```
x = 5  
y = 3  
tinh_luy_thua(x)
```

```
Traceback (most recent call last):  
  File "1:/GIANG_DAY/_PORTAL/Chuyen_de/LDS1_2022/Demo/function.py", line  
    tinh_luy_thua(x)  
TypeError: tinh_luy_thua() missing 1 required positional argument: 'b'
```

5. Tham số

□ Tham số từ khóa (*Keyword argument*)

- Là tham số khi truyền vào function không cần theo đúng thứ tự của các tham số khi xây dựng function. Tuy nhiên lại cần chỉ rõ tham số nào sẽ nhận giá trị gì hay nói cách khác là đặt tên tham số khi gọi hàm.
- Số lượng tham số khi gọi function cũng phải đúng với số lượng tham số khi xây dựng function.

```
def tinh_luy_thua(a, b):  
    print(a, '^', b, '=', a ** b)  
  
x = 5  
y = 3  
tinh_luy_thua(a=x, b=y)  
tinh_luy_thua(a=y, b=x)
```

5 ^ 3 = 125
3 ^ 5 = 243

5. Tham số

□ Tham số mặc định (*Default argument*)

- Là tham số sẽ được hàm *lấy giá trị mặc định* để thực hiện nếu khi gọi hàm, người dùng *không cung cấp giá trị cho tham số*.
- *Giá trị mặc định* của tham số được cung cấp ngay khi *khai báo tham số* của hàm.

```
def tinh_tong(a, b=None):  
    if b == None:  
        return 'Truyền thiếu đối số b khi gọi hàm'  
    else:  
        return a + b  
  
x = 10  
y = 5  
print(tinh_tong(x, y)) # 15  
print(tinh_tong(x)) # Truyền thiếu đối số b khi gọi hàm
```

5. Tham số

❑ Tham số thay đổi không xác định (Variable-length argument)

- Là tham số được sử dụng khi chưa xác định được số lượng các giá trị truyền vào function, tức là một tham số loại này có thể bao gồm rất nhiều giá trị đi kèm.
- Để cho biết số lượng tham số truyền cho hàm là không biết trước, Python cho phép dùng dấu * trước tên tham số.

5. Tham số

□ Tham số thay đổi không xác định (Variable-length argument)

- *args và **kwargs:

- args (viết tắt của arguments - đối số): trong Python, khi truyền tham số cho hàm ký hiệu *args ám chỉ danh sách các đối số truyền cho hàm. Các đối số này sẽ được Python đưa vào 1 list để xử lý. Do đó nếu đối số là 1 list thì cũng chỉ được xem là 1 thành phần trong list vừa nêu (sub List).

5. Tham số

□ Tham số thay đổi không xác định (Variable-length argument)

- *args và **kwargs:

- args (viết tắt của arguments - đối số):

```
def in_loi_chao(loi_chao, *danh_sach_ten):
    for ten in danh_sach_ten:
        print(loi_chao, ten)

in_loi_chao('Xin chào') # Không có kết quả vì danh_sach_ten rỗng
in_loi_chao('Xin chào', 'Minh', 'Linh', 'Nam', 'Mai')
```

Xin chào Minh
Xin chào Linh
Xin chào Nam
Xin chào Mai

5. Tham số

□ Tham số thay đổi không xác định (Variable-length argument)

- *args và **kwargs
 - kwargs (viết tắt của keyword arguments - đối số từ khóa): **kwargs tạo ra một dictionary chứa toàn bộ các đối số khi gọi hàm (mỗi cặp trong các đối số sẽ trở thành 1 thành phần của dictionary).

5. Tham số

□ Tham số thay đổi không xác định (Variable-length argument)

- *args và **kwargs

- kwargs (viết tắt của keyword arguments - đối số từ khóa):

```
def print_pet_names(owner, **pets):
    print(f'Owner name: {owner}')
    print(pets)
    print(type(pets))

print_pet_names('John',
                mouse='Mickey',
                fish=['Larry', 'Curly', 'Moe'],
                cat='Tom')

Owner name: John
{'mouse': 'Mickey', 'fish': ['Larry', 'Curly', 'Moe'], 'cat': 'Tom'}
<class 'dict'>
```

Nội dung

1. Định nghĩa
2. Khai báo và xây dựng hàm
3. Gọi hàm
4. Tầm vực của biến
5. Tham số
6. Hàm ẩn danh
7. Built-in Functions



6. Hàm ẩn danh

- **Hàm ẩn danh (Anonymous Function):** Gọi là ẩn danh vì function không được khai báo theo cách tiêu chuẩn bằng cách dùng từ khóa **def** mà được dùng 1 cách ngắn gọn bằng từ khóa **lambda**. Thường các function dạng này được viết chỉ trên 1 dòng lệnh.
- Cú pháp: **lambda [argument1, argument2, ...]: expression**

6. Hàm ẩn danh

- Cú pháp: **lambda [argument1, argument2, ...]: expression**
- Giải thích:
 - **lambda function:**
 - Không thể gọi trực tiếp *lambda function* như các *function* bình thường.
 - *Lambda function* có *local namespace* riêng và không thể truy cập các biến khác và các biến trong phạm vi *global namespace*.
 - Kết quả chỉ trả về 1 giá trị duy nhất.
 - **argument:** Tương tự như các hàm thông thường, số lượng đối số của *lambda* là 0 hay 1 hoặc nhiều đối số truyền vào.

6. Hàm ẩn danh

- Cú pháp: **lambda [argument1, argument2, ...]: expression**
- Giải thích:
 - **expression:**
 - Các hàm *lambda* chỉ chấp nhận một và chỉ một biểu thức (một lệnh đơn).
 - Có thể sử dụng tối đa 1 lệnh *if* trong *expression*.
 - Có thể gọi hàm khác trong *expression*.

6. Hàm ẩn danh

- Cú pháp: **lambda [argument1, argument2, ...]: expression**
- Ví dụ: Tính chỉ số BMI

```
# Sử dụng hàm thông thường (def)
def tinh_bmi(can_nang, chieu_cao):
    return can_nang / chieu_cao ** 2

print('Chỉ số BMI:')
print(tinh_bmi(60, 1.7))

# Sử dụng hàm ẩn danh (lambda)
tinh_bmi = lambda can_nang, chieu_cao: can_nang / chieu_cao ** 2

print('Chỉ số BMI:')
print(tinh_bmi(60, 1.7))
```

Chỉ số BMI:
20.761245674740486

6. Hàm ẩn danh

- Cú pháp:

lambda [argument1, argument2, ...]: expression

- Ví dụ 2: Kiểm tra giá trị nhập vào là số chẵn hay lẻ

```
# Sử dụng hàm thông thường (def)
def xet_chan_le(x):
    if x % 2 == 0:
        return 'là số chẵn'
    else:
        return 'là số lẻ'
n = 10
print(n, xet_chan_le(n))

# Sử dụng hàm ẩn danh (lambda)
xet_chan_le = lambda x: str(x) + ' là số chẵn' if x % 2 == 0 else str(x) + ' là số lẻ'
print(xet_chan_le(10))
```

10 là số chẵn

Nội dung

1. Định nghĩa
2. Khai báo và xây dựng hàm
3. Gọi hàm
4. Tầm vực của biến
5. Tham số
6. Hàm ẩn danh
7. Built-in Functions

7. Built-in Functions

- ❑ **map()**, **reduce()**, **filter()** là những built-in function hỗ trợ việc xử lý các sequence rất hiệu quả.



TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
TRUNG TÂM TIN HỌC

7. Built-in Functions

□ map():

- Tạo ra một **sequence mới** dựa trên một **phương thức** và **sequence cũ**. Mỗi phần tử trong sequence cũ sẽ áp dụng phương thức để thành phần tử mới.
- Nếu có nhiều hơn một sequence được cung cấp thì phương thức sẽ được gọi kết hợp cho từng phần tử của các sequence.
- Nếu một sequence ngắn hơn so với sequence khác => kết quả sẽ có số phần tử bằng với sequence ngắn.

7. Built-in Functions

□ map():

- Cú pháp: **map(function, sequence[, sequence, ...]) → list**
- Ví dụ 1: Tính bình phương các phần tử trong **list_1**

```
list_1 = [1, 2, 3, 4, 5]
list_2 = [6, 7, 8, 9, 10]
```

```
# Cách thông thường
list_3 = []
for so in list_1:
    list_3.append(so ** 2)
print('list_1:', list_1)
print('list_3:', list_3)
```

```
# Sử dụng phương thức map()
list_3 = list(map(lambda so: so ** 2, list_1))
print('list_1:', list_1)
print('list_3:', list_3)
```

```
list_1: [1, 2, 3, 4, 5]
list_3: [1, 4, 9, 16, 25]
```

7. Built-in Functions

□ map():

- Cú pháp: **map(function, sequence[, sequence, ...]) → list**
- Ví dụ 2: Tính tổng các phần tử trong **list_1** và **list_2**

```
list_1 = [1, 2, 3, 4, 5]
list_2 = [6, 7, 8, 9, 10]
```

```
# Cách thông thường
list_4 = []
for i in range(len(list_1)):
    tong = list_1[i] + list_2[i]
    list_4.append(tong)
print('list_4:', list_4)
```

```
# Sử dụng phương thức map()
list_4 = list(map(lambda item1, item2: item1 + item2, list_1, list_2))
print('list_4:', list_4)
```

list_4: [7, 9, 11, 13, 15]

7. Built-in Functions

❑ map():

- Cú pháp: **map(function, sequence[, sequence, ...]) → list**
- Ví dụ 3: Tính hiệu các phần tử trong **list_1** và **tuple_1** (sử dụng thư viện **operator**)

```
from operator import sub
list_1 = [1, 2, 3, 4, 5]
tuple_1 = [9, 8, 7, 6, 5]
list_5 = list(map(sub, list_1, tuple_1))
print(list_5)

[-8, -6, -4, -2, 0]
```

7. Built-in Functions

□ map():

- Cú pháp: **map(function, sequence[, sequence, ...]) → list**
- Ví dụ 4: Tính tích các phần tử trong **tuple_1** và **dict_1** (sử dụng thư viện **operator**)

```
from operator import mul
tuple_1 = [9, 8, 7, 6, 5]
dict_1 = {1: 'one', 3: 'three', 5: 'five'}
list_6 = list(map(mul, tuple_1, dict_1))
print('list_6:', list_6)
```

```
list_6: [9, 24, 35]
```

7. Built-in Functions

❑ filter():

- Dùng để lọc các item trong sequence, tạo ra một sequence mới với các item thỏa điều kiện của function.
- Cú pháp: **filter(function, sequence) → list**

7. Built-in Functions

□ filter():

- Cú pháp: **filter(function, sequence) → list**
- Ví dụ 1: Lọc danh sách số chẵn từ danh sách số cho trước

```
# Cách thông thường
list_numbers = [num for num in range(1, 11)]
list_even_numbers = []
for number in list_numbers:
    if number % 2 == 0:
        list_even_numbers.append(number)
print('list_numbers:', list_numbers)
print('list_even_numbers:', list_even_numbers)
```

```
list_numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
list_even_numbers: [2, 4, 6, 8, 10]
```

```
# Sử dụng phương thức filter()
list_numbers = [num for num in range(1, 11)]
list_even_numbers = list(filter(lambda number: number % 2 == 0, list_numbers))
print('list_numbers:', list_numbers)
print('list_even_numbers:', list_even_numbers)
```

7. Built-in Functions

□ filter():

- Cú pháp: **filter(function, sequence) → list**
- Ví dụ 2: Lọc danh sách tên các loại trái cây có chứa chữ 'n'

```
# Cách thông thường
list_fruits = ['apple', 'banana', 'lemon', 'orange', 'peach']
list_fruits_n = []
for fruit in list_fruits:
    if 'n' in fruit:
        list_fruits_n.append(fruit)
print(list_fruits_n)                                ['banana', 'lemon', 'orange']

# Sử dụng phương thức filter()
list_fruits = ['apple', 'banana', 'lemon', 'orange', 'peach']
list_fruits_n = list(filter(lambda fruit: 'n' in fruit, list_fruits))
print(list_fruits_n)
```

7. Built-in Functions

❑ reduce():

- Trả về kết quả là một *giá trị đơn* bằng cách áp dụng phương thức cho các item trong sequence.
- Import thư viện *functools* để sử dụng hàm **reduce()**
- Cú pháp: **reduce(function, sequence) → value**

7. Built-in Functions

❑ reduce():

- Cú pháp: **reduce(function, sequence) → value**
- Ví dụ 1: Tính tổng các phần tử trong list

```
from functools import reduce
from operator import add

danh_sach_so = [so for so in range(1, 11)]

tong_1 = reduce(lambda a, b: a + b, danh_sach_so)
print('Tổng =', tong_1) # Tổng = 55

tong_2 = reduce(add, danh_sach_so)
print('Tổng =', tong_2) # Tổng = 55
```

7. Built-in Functions

❑ reduce():

- Cú pháp: **reduce(function, sequence) → value**
- Ví dụ 2: Tìm giá trị lớn nhất, nhỏ nhất của các phần tử trong list

```
from functools import reduce

def tim_gia_tri_nho_nhat(a, b):
    return a if a < b else b

def tim_gia_tri_lon_nhat(a, b):
    return a if a > b else b

danh_sach_so = [3, 5, 2, 4, 7, 1]
print('Giá trị nhỏ nhất:', reduce(tim_gia_tri_nho_nhat, danh_sach_so))
print('Giá trị lớn nhất:', reduce(tim_gia_tri_lon_nhat, danh_sach_so))
```

Giá trị nhỏ nhất: 1
Giá trị lớn nhất: 7

7. Built-in Functions

❑ reduce():

- Cú pháp: **reduce(function, sequence) → value**
- Ví dụ 3: Nối các phần tử trong list để tạo thành 1 câu

```
from functools import reduce
from operator import add
list_words = ['Lập ', 'trình ', 'Python ', 'cơ ', 'bản ']
sentence = reduce(add, list_words)
print(sentence)
```

Lập trình Python cơ bản



TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
TIT TRUNG TÂM TIN HỌC
TTH