

Functions & Module

Khóa học: Python căn bản

Kiểm tra bài trước

Hỏi và trao đổi về các khó khăn gặp phải trong bài “Cấu trúc lặp”

Tóm tắt lại các phần đã học từ bài “Cấu trúc lặp”

Mục tiêu

- Trình bày được cú pháp khai báo hàm
- Trình bày được cú pháp gọi hàm
- Giải thích được tham số của hàm
- Giải thích cách sử dụng câu lệnh return trong hàm
- Trình bày được phạm vi của biến
- Khai báo và sử dụng được hàm không tham số
- Khai báo và sử dụng được hàm có tham số
- Khai báo và sử dụng được hàm có return

Thảo luận

Hàm

Hàm

- Hàm (function) là một nhóm các câu lệnh thực hiện một nhiệm vụ nhất định
- *Hàm* là thuật ngữ được sử dụng phổ biến trong Lập trình hướng đối tượng. Trong nhiều trường hợp khác, các tên gọi được sử dụng là *phương thức* (method) và *thủ tục* (procedure)
- `print()`, `input()`, `int()` là các hàm đã được định nghĩa sẵn cho chúng ta sử dụng

Lưu ý: Mặc dù tên gọi phương thức, hàm, procedure đôi khi có thể sử dụng thay thế cho nhau, nhưng giữa 3 khái niệm này có sự khác nhau.

Ví dụ về hàm số

f, g, h là tên hàm

x, y là tham số

biểu thức

• $f(x) = x^2$

• $g(x, y) = \sqrt{x} + \sqrt{y}$

• $h(x, y) = f(x) + a(x, y)$

Đây được gọi là hàm số

• $f(5) = 5^2 = 25$

• $g(9, 25) = \sqrt{9} + \sqrt{25} = 3 + 5 = 8$

• $h(9, 25) = 81 + 8 = 89$

Đánh giá hàm số

Các thành phần trong Hàm

- **Nhiệm vụ** của hàm
- **Tên** hàm
- **Đầu vào** của hàm
- **Đầu ra** của hàm

Ví dụ

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

Nhiệm vụ: Tính tổng hai số

Tên hàm: sum

Đầu vào: 2 tham số

Đầu ra: tổng hai tham số

Sử dụng hàm

Gọi hàm

Truyền giá trị tham số cho hàm

```
numb1 = 12
```

```
numb2 = 14
```

```
result = sum(numb1 , numb2)
```

```
print(result)
```



Hàm giúp tái sử dụng mã nguồn (1)

- Ví dụ, để tính tổng của các số từ 1 đến 10, tổng của các số từ 20 đến 37, tổng của các số từ 35 – 49:

```
sum = 0
for i in range(11):
    sum += i
print("Sum from 0 to 10 is " + str(sum))
```

```
sum = 0
for i in range(20, 38):
    sum += i
print("Sum from 20 to 37 is " + str(sum))
```

```
sum = 0
for i in range(35, 50):
    sum += i
print("Sum from 35 to 49 is " + str(sum))
```

Hàm giúp tái sử dụng mã nguồn (1)

- Nếu sử dụng hàm, mã nguồn sẽ trở nên ngắn gọn hơn

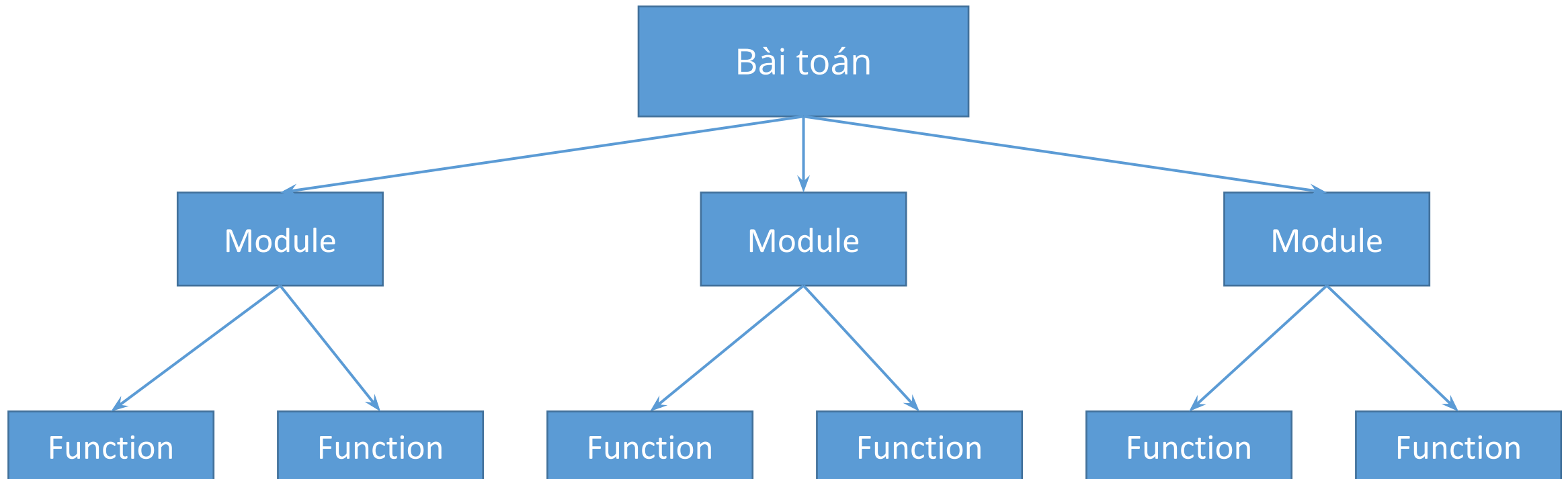
```
def sum(start, end):  
    total = 0;  
    for i in range(start, end):  
        total += i  
    return total
```

```
def main():  
    print("Sum from 1 to 10 is " + sum(1, 11))  
    print("Sum from 20 to 37 is " + sum(20, 38))  
    print("Sum from 35 to 49 is " + sum(35, 50))
```

```
main()
```

Ý nghĩa của việc sử dụng Hàm

- Thiết kế giải pháp theo hướng “chia để trị”



Khai báo hàm

- Cú pháp của hàm

```
def name(parameter1, parameter2, parameter3) :  
    # code to be executed
```

- Các thành phần quan trọng của hàm
 - name: Tên hàm
 - parameter1, parameter2, parameter 3: Tham số
 - code to be executed: phần thân hàm (các lệnh thực thi hàm)

Tên của hàm

- Tên của hàm tuân thủ quy tắc đặt tên của Python
- Tên của hàm nên bắt đầu bằng một động từ (vì hàm thực hiện một hành động)

Ví dụ tên không tốt	Tên tốt	Lý do
myInterest	calculateInterest	Bắt đầu bằng động từ
FindMaxValue	findMaxValue	Viết thường chữ đầu tiên
get_payment	getPayment	Sử dụng Camel Case

Lưu ý: Đặt tên cho hàm là một việc rất quan trọng và cần được thực hiện với sự cẩn trọng để đảm bảo mã nguồn được tốt nhất.

Demo

Hàm

Thảo luận

Kiểu dữ liệu trả về

Kiểu dữ liệu trả về

- Một hàm có thể trả về một giá trị
- Ví dụ, hàm kiểm tra số chẵn có kiểu dữ liệu trả về là boolean:

```
def isEven(number) :  
    return number % 2 == 0
```

Tham số (parameter) và đối số (argument)

- Tham số (còn được gọi đầy đủ là tham số hình thức – formal parameter) là các biến được khai báo trong phần header
- Khi gọi hàm thì giá trị của các biến này sẽ được truyền vào. Các giá trị này được gọi là tham số thực (actual parameter) hoặc đối số (argument)
- Ví dụ:

parameter
↓
def *isEven*(number):
 return number % 2 == 0

argument
↓
isEven(5)

Hàm là chiếc hộp đen

- Có thể hình dung hàm như là những chiếc hộp đen có công dụng thực hiện các nhiệm vụ nhất định
- Tham số là đầu vào của chiếc hộp
- Giá trị trả về là đầu ra của chiếc hộp
- Người sử dụng hàm không quan tâm đến bên trong chiếc hộp, chỉ quan tâm đến đầu vào và đầu ra



Gọi hàm

- Gọi (*call* hoặc *invoke*) phương thức là cách để thực thi một hàm đã được định nghĩa trước đó
- Khi gọi hàm thì cần truyền đối số vào
- Ví dụ, gọi hàm không có giá trị trả về:

```
print("Welcome to Python!")
```

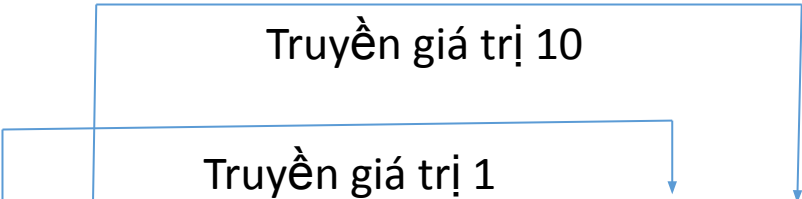
- Ví dụ, gọi hàm có giá trị trả về:

```
larger = max(3, 4)
```

Luồng điều khiển (control flow)


- Khi gọi hàm mức, luồng điều khiển được chuyển cho hàm đó

```
def main() :  
    print("Sum from 1 to 10 is " + sum(1, 10))  
    print("Sum from 20 to 37 is " + sum(20, 37))  
    print("Sum from 35 to 49 is " + sum(35, 49))  
  
def sum(start, end) :  
    total = 0  
    for i in range(start, end) :  
        total += i  
    return total
```



Giá trị trả về

- Giá trị trả về là kết quả sẽ được trả về tại vị trí hàm được gọi
- Mỗi một hàm chỉ có một giá trị trả về. Các giá trị này có thể là một biến, một mảng hay một đối tượng, danh sách đối tượng.
- Cú pháp sử dụng câu lệnh **return**

```
def toCelsius(fahrenheit):  Giá trị trả về  
    return (5/9) * (fahrenheit-32)
```

```
celsius = toCelsius(77)  
print(celsius)
```

Giá trị trả về: Ví dụ

```
def concatenate(first, last):  
    full = first + last  
    return full
```

```
def secondFunction():  
    result = concatenate('Zara', 'Ali')  
    print (result)
```

```
secondFunction()
```



Output
ZaraAli

Thảo luận

Truyền giá trị vào hàm

Truyền giá trị vào hàm

- Khi gọi hàm thì cần truyền giá trị vào nếu hàm đó có khai báo tham số
- Trật tự các giá trị truyền vào phải tương ứng với trật tự khai báo các tham số
- Ví dụ:

```
def printMultiple(message, n):  
    for i in n:  
        print(message)
```

Gọi đúng: `printMultiple("Hello", 4)`

Gọi sai: `printMultiple(4, "Hello")`

Pass-by-value: Ví dụ 1

```
def increment(n):  
    n++  
    print("n inside the method is " + n)
```

```
def main():  
    x = 1  
    print("Before the call, x is " + x)  
    increment(x)  
    print("After the call, x is " + x)
```

```
main()
```

```
Before the call, x is 1  
n inside the method is 2  
After the call, x is 1
```

Giá trị của biến x không
thay đổi sau khi gọi
phương thức increment

Pass-by-value: Ví dụ 2

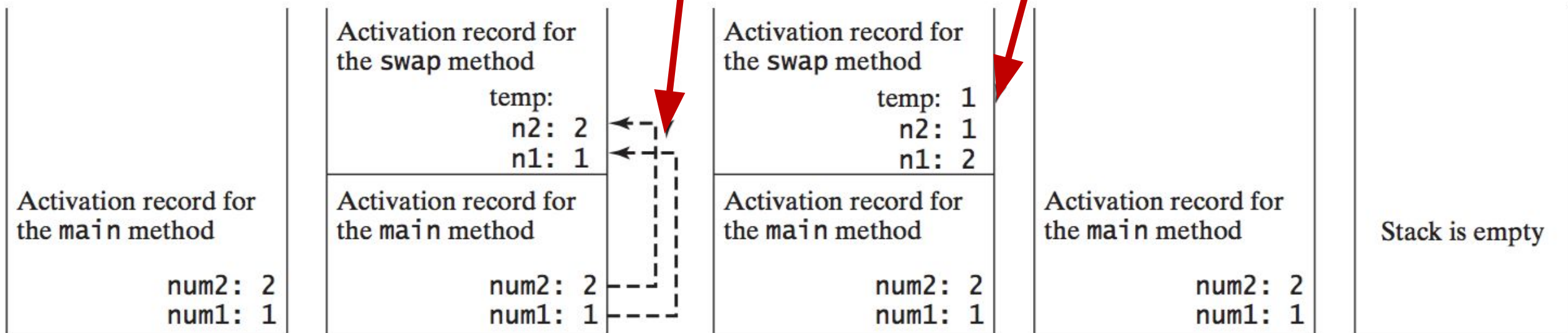
```
def swap(n1, n2):  
    print("Inside the swap method")  
    print("Before swapping, n1 is " + n1 + " and n2 is " + n2)  
    temp = n1  
    n1 = n2  
    n2 = temp  
    print("After swapping, n1 is " + n1 + " and n2 is " + n2)  
  
def main():  
    num1 = 1  
    num2 = 2  
    print("Before invoking the swap method, num1 is " + num1 + " and num2 is " + num2)  
    swap(num1, num2)  
    print("After invoking the swap method, num1 is " + num1 + " and num2 is " + num2)
```

```
Before invoking the swap method, num1 is 1 and num2 is 2  
    Inside the swap method  
        Before swapping, n1 is 1 and n2 is 2  
        After swapping, n1 is 2 and n2 is 1  
After invoking the swap method, num1 is 1 and num2 is 2
```

Mô phỏng pass-by-value

Giá trị của num1 và num2
được truyền cho n1 và n2

Giá trị của n1 và n2 được hoán đổi cho nhau
nhưng không ảnh hưởng đến num1 và
num2



1) Hàm main
được gọi

2) Hàm swap
được gọi

3) Hàm swap đang
thực thi



4) Hàm swap
được thực thi
xong



5) Hàm main
được thực thi
xong

Thảo luận

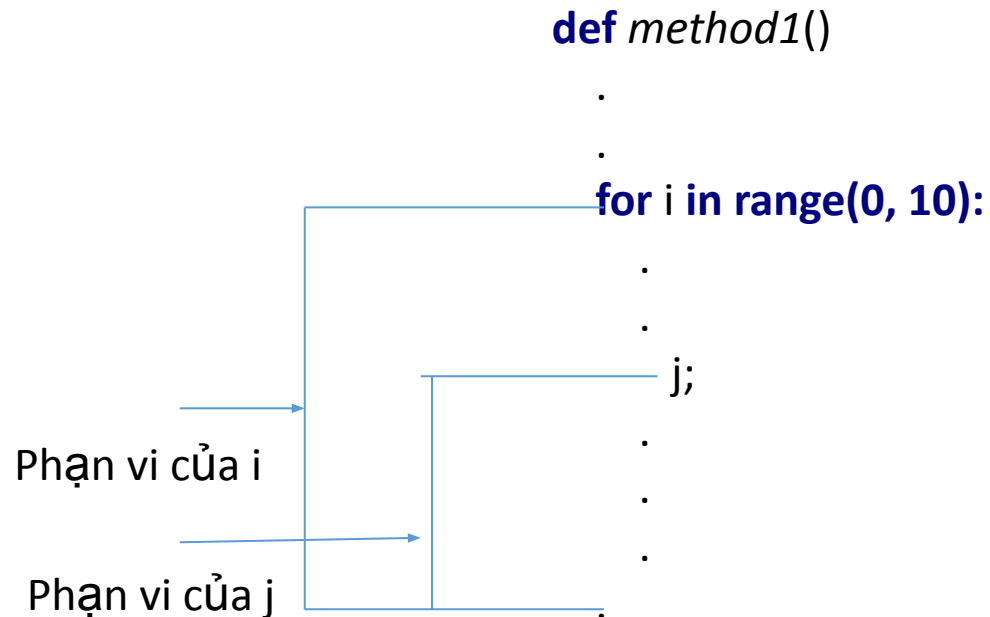
Phạm vi của biển

Phạm vi của biến

- Phạm vi (scope) của biến là các vị trí trong chương trình mà một biến có thể được sử dụng
- Một biến được khai báo trong một phương thức thì được gọi là *biến địa phương* (local variable)
- Phạm vi của biến địa phương bắt đầu từ vị trí nó được khai báo cho đến điểm kết thúc của khối lệnh chứa nó
- Một biến địa phương cần được khai báo và gán giá trị trước khi sử dụng
- Tham số của hàm cũng là các biến địa phương
- Phạm vi của các tham số là trong toàn bộ hàm đó

Phạm vi biến trong vòng lặp for

- Biến được khai báo trong phần header của vòng lặp thì có phạm vi trong toàn bộ vòng lặp
- Biến được khai báo trong phần thân của vòng lặp thì chỉ có phạm vi bên trong thân vòng lặp (tính từ vị trí được khai báo cho đến hết khối lệnh chứa nó)



Mutable Object và Immutable Object

- Đối tượng trong Python có 2 loại:
 - Đối tượng có thể thay đổi (mutable object)
 - Đối tượng không thể thay đổi (immutable object)
- Giá trị của mutable object có thể được sửa đổi tại nơi sau khi nó tạo ra
 - mutable object gồm: *int, float, long, complex, string, tuple, bool*
- Giá trị của immutable object thì không thể thay đổi
 - immutable object gồm: *list, dict, set, byte array, user-defined classes*

Mutable Object và Immutable Object

Để kiểm tra một đối tượng có khả năng thay đổi không, ta thử sửa đổi nó xem nó có còn là đối tượng ban đầu hay không.

Có 2 cách để kiểm tra:

- Sử dụng hàm tích hợp sẵn **id()**: hàm này trả về nhận dạng duy nhất của một đối tượng.

Lưu ý: Không có hai đối tượng nào có cùng định danh

- Sử dụng toán tử **is** và **is not**: các toán tử này đánh giá xem các đối tượng có cùng định danh hay không

Truyền tham chiếu và tham trị

- “Bất kỳ thứ gì có thể được đặt tên trong Python đều là đối tượng”. Do vậy tất cả các tham số (*đối số*) trong ngôn ngữ Python được truyền bằng tham chiếu.
- Nếu thay đổi những gì một tham số truyền đến bên trong một hàm, thay đổi đó có khả năng phản ánh lại trong hàm đang gọi hoặc không.
- Dựa vào loại đối tượng của tham chiếu mà bạn truyền vào mà phân biệt truyền tham chiếu hay tham trị.

Truyền tham chiếu

- Mặc định, Python truyền biến cho một hàm dưới dạng tham chiếu. Trong kỹ thuật này, vị trí bộ nhớ của biến được gửi đến hàm.
- Tham số là một đối tượng kiểu Mutable, hàm có thể thay đổi giá trị ban đầu của biến bằng cách gán một giá trị mới, nhưng định danh của tham số không thay đổi.
- Lưu ý: Một kiểu dữ liệu phải là Mutable mới có thể thay đổi dữ liệu gốc.

Truyền tham chiếu: Ví dụ

Ví dụ :

```
# Một danh sách được tạo sẵn
myList = ["Orange", "Apple", "Grapes"]
# Hàm sẽ thêm 1 phần tử vào danh sách
def addItem(myTempList):
    print('id of myTempList', id(myTempList))
    myTempList += ["Mango"]
    print(myTempList)
#In ra kết quả
print('id of myList', id(myList))
addItem(myList)
print(myList)
print(type(myList))
```

Truyền tham chiếu: Ví dụ

Kết quả:

```
id of myList 4431516928
```

```
id of myTempList 4431516928
```

```
['Orange', 'Apple', 'Grapes', 'Mango']
```

```
['Orange', 'Apple', 'Grapes', 'Mango']
```

```
<class 'list'>
```

Kết quả cho thấy rằng myList gốc và myTempList giống nhau. Vì cả hai cùng lưu đến cùng một vị trí bộ nhớ. Đây là kết quả của việc truyền tham chiếu vì nó thay đổi dữ liệu ban đầu.

Truyền tham trị

- Trong trường hợp tham số là một đối tượng kiểu Immutable, hàm không thể thay đổi giá trị của tham số.
- Do đó, Python tạo ra một biến số mới với một định danh mới để nhận giá trị được gán. Kỹ thuật này sẽ không thay đổi giá trị của tham số truyền vào hàm.

Truyền tham trị: Ví dụ

```
def addInt(myTempInt):  
    print("[addInt] myTempInt=", myTempInt)  
    print("[addInt] id of myTempInt=", id(myTempInt))  
  
    myTempInt += 5  
    print("[addInt] myTempInt after add=", myTempInt)  
    print("[addInt] id of myTempInt after add=", id(myTempInt))  
  
myInt = 45  
print("myInt", myInt)  
print("id of myInt", id(myInt))  
  
addInt(myInt)  
print("myInt", myInt)  
print("id of myInt", id(myInt))  
  
print(type(myInt))
```

Truyền tham trị: Ví dụ

Kết quả:

```
myInt 45
id of myInt 4387262624
[addInt] myTempInt= 45
[addInt] id of myTempInt= 4387262624
[addInt] myTempInt after add= 50
[addInt] id of myTempInt after add= 4387262784
myInt 45
id of myInt 4387262624
<class 'int'>
```


Truyền tham trị: Giải thích ví dụ

- Ở ví dụ khai báo một số nguyên, ta sẽ thấy có sự thay đổi id của giá trị trước và sau khi gán giá trị mới ở bên trong hàm. Tuy nhiên id giá trị ban đầu (ở bên ngoài hàm) vẫn giữ nguyên không thay đổi.
- Ta có thể thấy khi thay đổi giá trị thì hàm đã tạo ra một id mới, lưu giá trị mới ở một địa chỉ mới khác với địa chỉ lưu giá trị ban đầu
- Ta cũng biết đây là truyền tham trị khi kiểm tra kiểu của đối tượng, **int (Immutable Object)**

Khác nhau của tham chiếu ,tham trị

Tham chiếu	Tham trị
Thay đổi trong biến cũng ảnh hưởng đến biến toàn cục ngoài hàm	Thay đổi chỉ ảnh hưởng trên bản sao của biến, không ảnh hưởng đến biến ngoài hàm
Cho phép thực hiện thay đổi các giá trị của biến bằng cách sử dụng các lệnh gọi hàm	Không cho phép thực hiện bất kỳ sự thay đổi trên giá trị các biến
Giá trị ban đầu thay đổi	Giá trị ban đầu không thay đổi

Python sẽ sử dụng hàm phụ thuộc vào đối số được truyền vào:

- *Nếu đối tượng không thay đổi khi được truyền vào thì dùng tham trị*
- *Nếu đối tượng thay đổi khi được truyền vào thì dùng tham chiếu*

Tóm tắt bài học

- Phương thức giúp tái sử dụng mã nguồn
- Các thành phần của một phương thức: modifier, kiểu dữ liệu trả về, tên phương thức, danh sách tham số
- Đặt tên cho phương thức là một thao tác rất quan trọng để đảm bảo clean code
- Tham số của phương thức (parameter) là các biến hình thức
- Đối số là các giá trị được truyền vào khi gọi phương thức
- Câu lệnh return được sử dụng để trả về giá trị của một phương thức
- Phạm vi của biến là các vị trí trong chương trình mà một biến có thể được sử dụng

Hướng dẫn

Hướng dẫn làm bài thực hành và bài tập

Chuẩn bị bài tiếp theo