14. Truyền đối số (parameter passing)

Truyền giá trị hoặc truyền một tham chiếu ("call by value" and "call by name") tới hàm là hai cách phổ biến.

Truyền giá trị:

Đây là cách phổ biến nhất và nó có thể được đề cập với hai tên sau “call-by-value”, hay “called pass-by-value”. Cách này được sử dụng trong C và C++ ví dụ. Trong call-by-value, giá trị truyền vào sẽ được sao chép tới một vùng nhớ bị giới hạn trong phạm vị của hàm. Nghĩa là biến trong hàm gọi sẽ không bị thay đổi sau khi hàm được trả về.

Truyền tham chiếu:

Cách này còn được gọi là pass-by-reference, hàm sẽ nhận một tham chiếu ngầm tới một đối số, thay vì phải sao chép toàn bộ giá trị của đối số. Do đó, hàm có thể thay đổi giá trị của đối số được truyền vào, giá trị mà chỉ có thể thay đổi ở phạm vi của hàm gọi. Cách này có lợi thế về cả về thời gian cũng như bộ nhớ. Tuy nhiên, nó cũng đem lại một bất lợi là giá trị có thể bị thay đổi “vô tình” làm thay đổi chức năng của hàm. Vì vậy, khi dùng phương pháp này cần kết hợp một số cách để “bảo vệ” giá trị đó như sử dụng “const” hay “final”.

Có rất nhiều ngôn ngữ hỗ trợ call-by-reference như c, c++, nhưng Perl sử dụng nó như mặc định.

Vậy Python dùng như thế nào?

Nói chính xác, Python sử dụng một cơ chế, được gọi là "Call-by-Object", đôi khi còn được gọi là "Call by Object Reference" hoặc "Call by Sharing".

Nếu bạn truyền các đối số bất biến như số nguyên, chuỗi hoặc tuple tới một hàm, cách này hoạt động giống với call-by-value. Đối tượng tham chiếu được truyền đến các tham số của hàm. Chúng không thể thay đổi trong thân hàm, bởi vì chúng không thể thay đổi, nghĩa là chúng không thay đổi.

Nó khác biệt, nếu chúng ta truyền một đối số có thể thay đổi được (mutable arguments). Chúng cũng được truyền một tham chiếu tới đối tượng, nhưng chúng có thể được thay đổi được trong thân hàm. Vậy nó hoạt động giống “call-by-reference”.

Vậy kết luận, khi truyền một đối số tới hàm ta cần quan tâm đến liệu đối số đó là immutable object hay mutable object. Phân tích ba ví dụ dưới đây để hiểu chi tiết hơn.

Ví dụ 1. Đối số là một số mô tả hoạt động kiểu call-by-value.

```
def ref_demo(x):  # khai bao ham
    print "x=", x, "id=", id(x)
    x = 100
    print "x=", x, "id=", id(x)

var = 20  # var la mot immutable object
print "id(var) = ", id(var)
ref_demo(var)
print "var = ", var  # var khong bi thay doi sau khi ra khoi ham ref_demo

Output:

id(var) =  34432768

x= 20 id= 34432768
```

```
x= 100 id= 34434832

var =  20
```

Ví dụ 2. Mô tả hoạt động call-by-reference với đối số là immutable object. Lỗi sẽ xảy ra khi thân hàm cố thay đổi immutable object.

```
def ref_demo_mutable_object(x):
    print "x=", x, "id=", id(x)
    x[0] = 100000
    print "x=", x, "id=", id(x)


var = (1, 2, 3)  # var la mot immutable object
ref_demo_mutable_object(var)  # loi vi khong the thay doi gia tri cua immutable object

Output:

x= (1, 2, 3) id= 38451744

Traceback (most recent call last):

  File "C:/Users/cuongtran/PycharmProjects/BigData/course.py", line 7, in <module>

    ref_demo_mutable_object(var)  #loi vi khong the thay doi gia tri cua immutable object

  File "C:/Users/cuongtran/PycharmProjects/BigData/course.py", line 3, in ref_demo_mutable_object

    x[0] = 100000

TypeError: 'tuple' object does not support item assignment
```

Ví dụ 3: Mô tả hoạt động call-by-reference với đối số là mutable object.

```
def ref_demo_mutable_object(x):
    print "x=", x, "id=", id(x)
    x[0] = 100000
    print "x=", x, "id=", id(x)


var = [1, 2, 3]  # var la mot mutable object

print "var= ", var, "id=", id(var)
ref_demo_mutable_object(var)  # co gang thay doi gia tri cua var

print "var= ", var, "id=", id(var)

Output:



var=  [1, 2, 3] id= 50947528

x= [1, 2, 3] id= 50947528

x= [100000, 2, 3] id= 50947528

var=  [100000, 2, 3] id= 50947528
```

Nếu bạn quan sát đến sự thay đổi giá trị của id trong ví dụ 1 phía trên. Điều này có nghĩa là Python ban đầu hoạt động giống như call-by-reference, nhưng ngay khi chúng ta thay đổi giá trị của một biến như vậy, Python "switch" thành call-by-value.

Tương tự như trong bài về Hàm, hai kí tự * và ** khi gọi hàm cũng giúp ta linh hoạt hơn rất nhiều trong khi truyền đối số.

Kí tự '*' khi gọi hàm: Ta có hai ví dụ sau để so sánh:

Ví dụ 1	Ví dụ 2

def arbitrary(x, y, *more):
  print x, y, more
 	
more = (10, 20)
arbitrary(1, 2, more[0], more[1])
 	
Output:
 	
1 2 (10, 20)
	def arbitrary(x, y, *more):
	  print x, y, more
	
	more = (10, 20)
	arbitrary(1, 2, *more)
	
	Output:
	
	1 2 (10, 20)

So sánh hai ví dụ, ta thấy cách thứ 2 dễ dàng và thuận tiện hơn rất nhiều. Ngoài ra ví dụ 1 còn không hoạt động trong trường hợp chung, nghĩa là các danh sách độ dài không xác định. "Không xác định" nghĩa là chiều dài chỉ được biết khi chạy và không phải ở thời điểm khi chúng ta viết code.

Kí tự '**' khi gọi hàm, để hỗ trợ truyền tham số khi sử dụng dictionary chỉ tường minh tên tham số.

Ví dụ:

```
def f(a, b, x, y):
    print(a, b, x, y)


d = {'a': 'append', 'b': 'block', 'x': 'extract', 'y': 'yes'}
f(**d)

Output:

('append', 'block', 'extract', 'yes')
```

Kết luận

Có hai cách phổ biến được sử dụng để truyền đối số trong C/C++/Perl là truyền giá trị và truyền tham chiếu. Python cũng dùng một cách tương tự như vậy dựa trên kiểu của dữ liệu truyền vào là immutable object hay mutable object mà sử dụng truyền giá trị hay truyền tham chiếu tương ứng. Hơn nữa trong Python việc truyền đối số cũng trở lên linh hoạt hơn nhiều nhờ vào việc hỗ trợ 2 kí tự '*' và '**' khi gọi hàm. Nhờ đó, chúng ta không phải viết tường minh hay cần biết trước số lượng các tham số đầu vào.