

Bài 31: KIỂU DỮ LIỆU FUNCTION TRONG PYTHON - BIẾN LOCALS VÀ GLOBALS

Xem bài học trên website để ủng hộ Kteam: [Kiểu dữ liệu Function trong Python - Biến locals và globals](#)

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage [How Kteam](#) nhé!

Dẫn nhập

Trong bài trước, Kteam đã giới thiệu đến bạn KIỂU DỮ LIỆU FUNCTION TRONG PYTHON.

Và ở bài này Kteam sẽ lại tìm hiểu với các bạn **Kiểu dữ liệu Function trong Python - Biến locals và globals**.

Nội dung

Để đọc hiểu bài này tốt nhất bạn cần:

- Cài đặt sẵn [MÔI TRƯỜNG PHÁT TRIỂN CỦA PYTHON](#).
- Xem qua bài [CÁCH CHẠY CHƯƠNG TRÌNH PYTHON](#).

- Năm [CÁCH GHI CHÚ](#) và [BIẾN TRONG PYTHON](#).
- CÁC KIỂU DỮ LIỆU ĐƯỢC GIỚI THIỆU TRONG PYTHON
- [CÂU ĐIỀU KIỆN IF TRONG PYTHON](#)
- [VÒNG LẶP WHILE](#) và [VÒNG LẶP FOR TRONG PYTHON](#)
- [NHẬP XUẤT TRONG PYTHON](#)

Bạn và Kteam sẽ cùng tìm hiểu những nội dung sau đây

- Sự khác biệt giữa khai báo biến ở trong hàm và ngoài hàm
- Thay đổi giá trị argument gián tiếp qua parameter
- Sử dụng lệnh global
- Giới thiệu hàm locals và globals

Khai báo biến ở trong hàm

Giả sử, bạn có một đoạn script như sau với việc khai báo một biến ngoài hàm và sử dụng ở trong hàm

```
kteam = 'How Kteam'

def say_slogan():
    print("We are", kteam)
say_slogan()
```

Ta dễ dàng biết được kết quả

```
1 kteam = 'How Kteam'
2
3 def say_slogan():
4     print("We are", kteam)
5
6 say_slogan()
7
We are How Kteam
[Finished in 0.3s]
```

Thử một trường hợp tiếp theo là khai báo biến ở trong hàm và sử dụng ở trong hàm

Kết quả cũng không có gì ngoài dự đoán

```
1 #kteam = 'How Kteam'
2
3 def say_slogan():
4     kteam = 'How Kteam'
5     print("We are", kteam)
6
7 say_slogan()
8
```

We are How Kteam
[Finished in 0.3s]

Giờ hãy thử khai báo một biến trong hàm mà sử dụng ngoài hàm xem nào

```
def make_slogan():
    kteam = 'How Kteam'

print(kteam)
```

Có ai đoán trật kết quả không nhỉ?

```
1 def make_slogan():
2     kteam = 'How Kteam'
3
4     print(kteam)
5
```

File "D:\Stuff\a.py", line 4, in <module> x

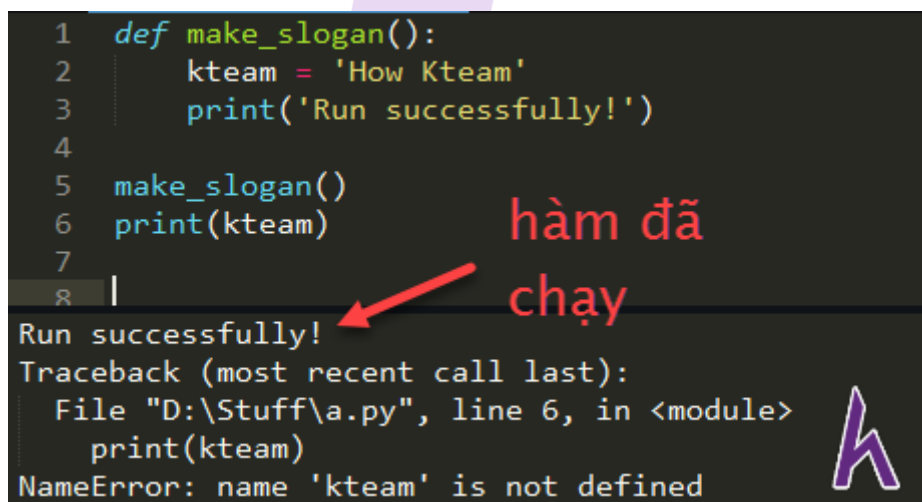
Traceback (most recent call last):
File "D:\Stuff\a.py", line 4, in <module>
print(kteam)
NameError: name 'kteam' is not defined

Cụ thể, Python nói với chúng ta rằng biến `kteam` chưa được khai báo. Nhưng rõ ràng là chúng ta đã khai báo nó ở trong hàm rồi mà?

À, là do chưa chạy hàm. Giờ ta sẽ thử lại. Lần này, ta để một dòng lệnh trong hàm luôn để chắc chắn rằng hàm đã chạy

```
def make_slogan():  
    kteam = 'How Kteam'  
    print('Run successfully!')  
  
make_slogan()  
print(kteam)
```

Vẫn lỗi tương tự mặc dù hàm đã chạy



```
1 def make_slogan():  
2     kteam = 'How Kteam'  
3     print('Run successfully!')  
4  
5 make_slogan()  
6 print(kteam)  
7  
8 |  
Run successfully!  
Traceback (most recent call last):  
  File "D:\Stuff\a.py", line 6, in <module>  
    print(kteam)  
NameError: name 'kteam' is not defined
```

hàm đã chạy

Từ đó, chúng ta có thể kết luận rằng việc khai báo biến ở trong hàm là có vấn đề.

Và quả thực là như vậy. Việc bạn khai báo một biến ở một hàm nào đó thì biến đó chỉ có thể sử dụng trong hàm đó. Còn nếu đi ra ngoài hàm khai báo nó thì biến đó hoàn toàn vô danh > thông báo chưa được khởi tạo.

Hãy tưởng tượng như thế này. Bạn sinh ra ở Việt Nam thế nên ở nước Việt Nam người ta có thể biết bạn là ai (có thể là qua CMND hoặc Căn Cước Công Dân). Nhưng nếu bạn đi qua Úc, Nhật hoặc thậm chí là Lào, Campuchia thì không ai biết được bạn là ai, tên gì cả.

Và như ví dụ trên. Bạn đã khai báo biến `kteam` trong hàm thôi thì chỉ trong hàm đó mới biết bạn. Còn nếu quăng ra ngoài thì chương trình chẳng biết nó là thằng nào cả.

Một lần nữa lặp lại đó là biến khai báo ở hàm nào thì chỉ hàm đó mới biết biến đó còn thoát ra ngoài hàm đó thì coi như không có. **Biến khai báo ở hàm cha có thể sử dụng trong hàm con nhưng biến ở hàm con không thể sử dụng ở hàm cha.**

Giống như ví dụ mà bạn thấy, nếu bạn khai báo biến ở ngoài hàm thì bạn hoàn toàn có thể sử dụng biến đó ở trong hàm.

Lưu ý: Biến là đối tượng nên bị ràng buộc bởi điều này. Do đó các HÀM (FUNCTION), LỚP (CLASS) cũng chịu sự ràng buộc này tương tự. Khai báo ở hàm nào thì chỉ dùng ở hàm đó.

Thay đổi giá trị argument gián tiếp qua parameter

Nói về vấn đề này, bạn nên biết 2 thuật ngữ là **pass by reference** và **pass by value**.

Đầu tiên, thế nào là **pass by reference**. Giả sử bạn có một chiếc laptop. Thằng bạn nó muốn mượn dùng một ngày. Thế là bạn mang máy của mình cho nó mượn. Về nhà thằng bạn nó down phim, down game, cài virus. Sau một ngày, hắn đem trả lại bạn. Coi như chiếc laptop của bạn tan nát. Việc bạn đưa laptop của mình cho thằng bạn cũng giống như việc **pass by reference**. Có nghĩa là bạn đưa bản gốc.

Sang tuần sau. Thằng bạn lại đến mượn đồ tiếp và bây giờ là chiếc xe đạp. Bạn biết tổng hắn kiểu gì cũng phá, liền lấy bảo bối là "Gương nhân đôi", sau đó bạn nhân bản chiếc xe đạp của bạn ra. Bạn lấy bản nhân bản đưa hắn mượn. Và bạn thấy đó, dù cho hắn có đạp nát chiếc xe kia thì xe của bạn cũng không sao. Đó gọi là bạn **pass by value**, đưa giá trị hoặc là "đưa bản sao".

Trong Python, việc bạn **CHỈ CÓ THỂ pass by value**

Ta sẽ đến với ví dụ để hiểu hơn. Ta có đoạn script sau

```
num = 69
st = 'How Kteam'
lst = [1, 2, 3]
tup = tuple('Education')

def change(parameter):
    parameter = 'New value'
    print('Changed successfully!')

change(num)
change(st)
change(lst)
change(tup)
print('*' * 10)
print('{}\n{}\n{}\n{}\n'.format(num, st, lst, tup))
```

Và đây là kết quả

```

1  num = 69
2  st = 'How Kteam'
3  lst = [1, 2, 3]
4  tup = tuple('Education')
5
6  def change(parameter):
7      parameter = 'New value'
8      print('Changed successfully!')
9
10 change(num)
11 change(st)
12 change(lst)
13 change(tup)
14 print('*' * 10)
15 print('{}\n{}\n{}\n{}\n'.format(num, st, lst, tup))
16

```

```

Changed successfully!
Changed successfully!
Changed successfully!
Changed successfully!
*****
69
How Kteam
[1, 2, 3]
('E', 'd', 'u', 'c', 'a', 't', 'i', 'o', 'n')

[Finished in 0.5s]

```



Như bạn thấy, không một giá trị nào bị thay đổi vì đó là **pass by value**.

Tuy nhiên, chúng ta thử lật lại kiến thức một ít nào. Bạn nhớ phần **"Vấn đề cần lưu tâm khi sử dụng List"** Trong bài [KIỂU DỮ LIỆU LIST TRONG PYTHON – Phần 1](#) chứ?

Việc bạn truyền giá trị cho parameter giống hệt như lúc bạn khởi tạo biến parameter và đưa giá trị cho nó vậy.

Xem ví dụ để hiểu nhé!

```
lst = ['How Kteam', 1, 2]
```

```

def change(parameter):
    parameter[1] = 'New value'
    print('Changed successfully!')

```

```
print(lst)
change(lst)
print(lst)
```

Kết quả(*)

```
1 lst = ['How Kteam', 1, 2]
2
3 def change(parameter):
4     parameter[1] = 'New value'
5     print('Changed successfully!')
6
7 print(lst)
8 change(lst)
9 print(lst)
10
```

```
['How Kteam', 1, 2]
Changed successfully!
['How Kteam', 'New value', 2]
[Finished in 0.2s]
```

Bạn sẽ có một câu hỏi liên quan tới phần này ở cuối bài nhé!

Sử dụng lệnh global

Nếu như một biến nằm trong một hàm (như biến `kteam` trong ví dụ cuối ở phần đầu) thì người ta hay gọi đó là **local variable** (biến chỉ có hiệu lực trong một hàm nhỏ).

Đặt vấn đề là việc khai báo biến ở trong hàm trở nên cần thiết thì sao nhỉ?

Ta được Python hỗ trợ lệnh **global**.

Cú pháp:

global <variable>

Lệnh này như một phép màu mà bạn có thể tạo ra. Giống như bạn có thể biến một người thành tổng thống Mỹ vậy. Ai trên thế giới này cũng biết. Và như biến, ở nơi nào trong chương trình cũng dùng được.

Hãy đến với ví dụ giống như ví dụ cuối phần đầu bạn này như có khác biệt một chút

```
def make_slogan():  
    # khởi tạo với global không có giá trị nhé  
    global kteam  
    # sau khi khởi tạo xong, ta mới gán giá trị  
    kteam = 'How Kteam'  
  
# nhớ là phải chạy hàm nữa  
make_slogan()  
  
print(kteam)
```

Và đây là kết quả khi ta có global

```
1  def make_slogan():  
2      # khởi tạo với global không có giá trị nhé  
3      global kteam  
4      # sau khi khởi tạo xong, ta mới gán giá trị  
5      kteam = 'How Kteam'  
6  
7      # nhớ là phải chạy hàm nữa  
8      make_slogan()  
9  
10     print(kteam)  
11  
How Kteam  
[Finished in 0.2s]
```



Ở đây Kteam muốn bạn lưu ý một trường hợp là tên **biến local trùng** với tên **biến global**.

```
def make_global():  
    global x  
    x = 1  
  
def local():  
    x = 5  
    print('x in local', x)  
  
make_global()  
print(x)  
local()  
print(x)
```

Kết quả

```
1  def make_global():  
2      global x  
3      x = 1  
4  
5  def local():  
6      x = 5  
7      print('x in local', x)  
8  
9  make_global()  
10 print(x)  
11 local()  
12 print(x)  
13  
  
1  
x in local 5  
1  
[Finished in 0.3s]
```

Như bạn thấy ở ví dụ trên, biến **x** trong hàm **local** đã trùng với biến **global x**. Tuy nhiên hai biến **x** này là hoàn toàn khác nhau. Biến **x** dùng trong hàm **local** thì có một địa chỉ riêng và một giá trị riêng, còn biến **x global** thì cũng có một

giá trị riêng và một địa chỉ riêng. Thêm nữa, nếu như ta sử dụng biến `x` ngoài hàm thì Python sẽ tìm tới biến `x global` chứ không phải là biến `x local`.

Lưu ý:

BẠN KHÔNG NÊN SỬ DỤNG GLOBAL trừ khi hết cách. Nó giống như **hàm eval** vậy. Việc sử dụng biến global làm cho chương trình rối, khó kiểm soát cho nên hạn hã chế tối đa việc sử dụng.

Giới thiệu hàm locals và globals

Cái tên hàm nói lên tất cả. **Hàm locals** cho ta biết được những biến `local` (những biến được khai báo trong hàm) nằm trong chương trình của chúng ta. Còn **globals** là hàm giúp chúng ta biết được những biến `global` trong chương trình.

Kết quả trả ra của hai hàm này là một Dict. Với **key** là tên biến và **value** là giá trị của biến.

Lưu ý:

Với hàm `globals()` thì với biến `globals` có giá trị mới được trả về.

```
>>> locals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class
'_frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annotations__': {},
 '__builtins__': <module 'builtins' (built-in)>}
>>> globals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class
'_frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annotations__': {},
 '__builtins__': <module 'builtins' (built-in)>}
>>>
```

```
>>> global x
>>>
>>> globals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class
'__frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annotations__': {},
'__builtins__': <module 'builtins' (built-in)>}
>>> x = 'How Kteam'

>>> globals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class
'__frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annotations__': {},
'__builtins__': <module 'builtins' (built-in)>, 'x': 'How Kteam'}
>>> globals()['x']
'How Kteam'
```

Kết luận

Qua bài viết này, Bạn đã biết thêm về hàm trong Python qua các khái niệm local và global trong hàm.

Ở bài viết sau, Kteam sẽ tiếp tục giới thiệu thêm với các bạn về [KIỂU DỮ LIỆU FUNCTION TRONG PYTHON - RETURN](#)

Cảm ơn bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên **“Luyện tập – Thử thách – Không ngại khó”**.