

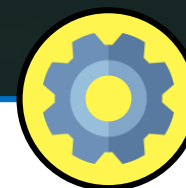


HÀM



Hàm:

- Hàm giúp các lập trình viên có thể sử dụng lại code của mình (code reuse), chúng cho phép bạn định nghĩa một khối lệnh và có thể thực hiện đi thực hiện lại nhiều lần trong 1 chương trình.
- Python cũng cung cấp các hàm có sẵn (built-in function) như `print()`, `max()`, `len()`, `type()`... Giờ đây bạn có thể tự xây dựng những hàm của riêng mình.



1. Xây dựng hàm:

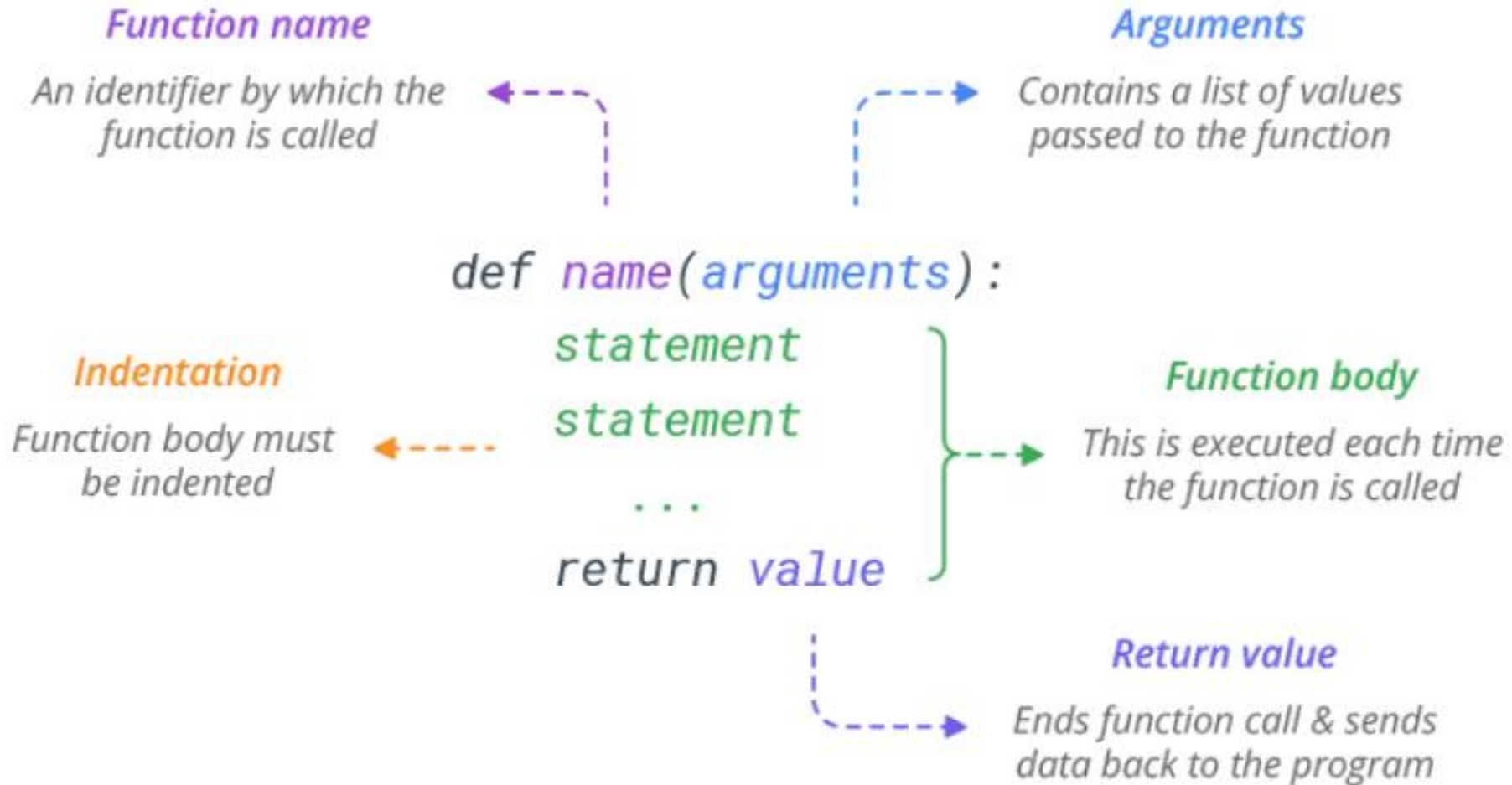


- Để **xây dựng hàm** ta sử dụng **từ khóa def**, trước khi xây dựng hàm **cần xác định các tham số** (giá trị mà bạn truyền cho hàm), **câu lệnh return** nếu mong muốn hàm trả về một giá trị nào đó.
- Hàm chỉ được chạy khi nó được gọi (**function call**).

CÚ PHÁP

```
def function_name(parameter):  
    #code
```

1. Xây dựng hàm:



1. Xây dựng hàm:

EXAMPLE

Lời gọi hàm

```
def hello():  
    print('hello 28tech')  
  
if __name__ == '__main__':  
    hello() #function call
```

OUTPUT

hello 28tech


EXAMPLE

Tham số và đối số

```
def hello(name):  
    print('hello', name)  
  
if __name__ == '__main__':  
    hello('Teo')  
    hello('Ti')
```

OUTPUT

hello Teo
hello Ti

Trong ví dụ này thì name là tham số (parameter) của hàm còn 'Teo' và 'Ti' là 2 giá trị bạn truyền cho hàm này khi gọi hàm được gọi là đối số (Argument) hoặc tham số thực sự. 

1. Xây dựng hàm:

EXAMPLE

Truyền nhiều tham số cho hàm

```
def hello(name1, name2, name3):  
    print('hello :', name1, name3, name2)
```

```
if __name__ == '__main__':  
    hello('Teo', 'Ti', 'Nam')
```

OUTPUT

```
hello Teo  
hello Ti
```

Bạn có thể truyền nhiều tham số cho hàm, chỉ cần phân cách giữa chúng bằng dấu phẩy. ●

EXAMPLE

Câu lệnh return

```
def findMax(a, b, c):  
    return max(a, b, c)
```

OUTPUT

```
200
```

```
print(findMax(100, 200, 50))
```

Chú ý: Trong python kể cả khi bạn không có câu lệnh return thì hàm trong Python vẫn có giá trị trả về là None.

Một khi câu lệnh return được thực thi thì hàm của bạn sẽ kết thúc ngay lập tức ●

2. Keyword argument:



Khi thực hiện gọi hàm thì thứ tự của đối số bạn truyền cho hàm là rất quan trọng.

EXAMPLE

```
def printNum(a, b, c):  
    print(a, b, c)
```

```
printNum(1, 2, 3)  
printNum(3, 1, 2)
```

OUTPUT

```
1 2 3  
3 1 2
```

2. Keyword argument:



Bạn có thể sử dụng cú pháp `parameter_name = value` khi gọi hàm để chỉ định giá trị được gán cho các tham số, khi đó thứ tự bạn gọi hàm không còn quan trọng nữa.

EXAMPLE

```
def hello(name1, name2, name3):  
    print(name1, name2, name3)  
  
hello(name2 = 'Lan', name3 = 'Huong', name1 = 'Ngoc')
```

OUTPUT

Ngoc Lan Huong

3. Default argument:



Bạn có thể chỉ định các giá trị mặc định cho các đối số khi xây dựng một hàm. Giá trị mặc định được sử dụng nếu hàm được gọi mà không có đối số tương ứng.



```
def infor(name, job = 'developer'):  
    print(name, job)
```

```
#Goi ham ma co doi so cho job  
infor('Teo', 'Xe om')  
#Goi ham ma khong co doi so  
infor('Teo')
```

OUTPUT

```
Teo Xe om  
Teo developer
```

4. Variable length argument (*args và **kwargs):



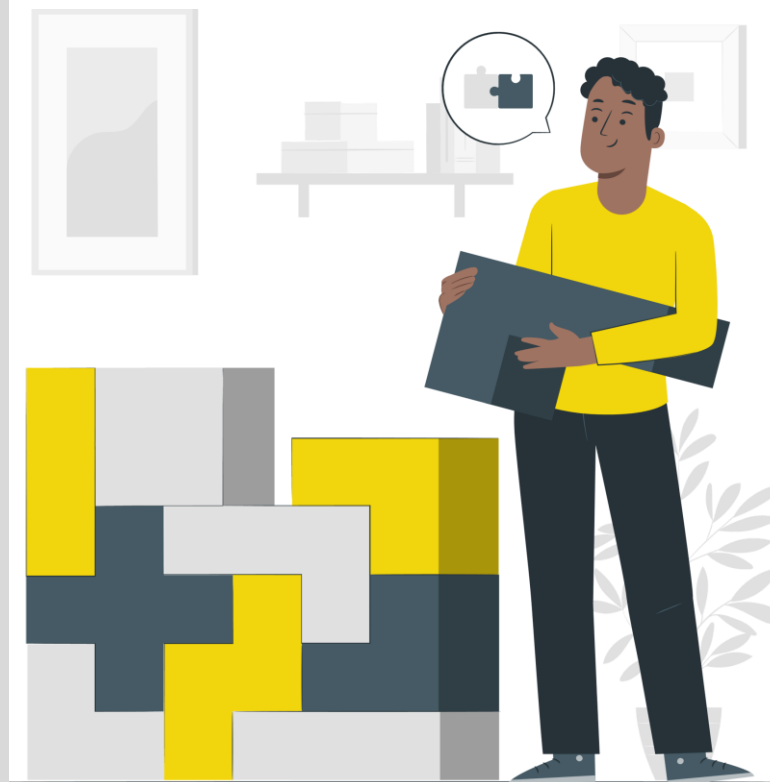
Variable length argument được sử dụng khi bạn không biết trước số lượng phần tử của tham số.



***args:** Khi tham số có thêm dấu * ở trước, hàm sẽ tiến hành thu thập tất cả các đối số thành một tuple, sau đó bạn có thể sử dụng args như một tuple, các bạn có thể đặt tên khác args, nhưng nên để tên này để mọi người dễ nhận biết.



****kwargs:** Được sử dụng với keyword argument, các đối số được thu thập thành một dictionary, trong đó tên đối số là key và giá trị tương ứng được gán cho value.



4. Variable length argument (*args và **kwargs):

EXAMPLE

```
def printNum(*args):  
    print(args)
```

```
printNum(1, 2, 3, 4, 5, 6,)
```

OUTPUT

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

EXAMPLE

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

```
printInfor(name = 'Teo', job = 'Developer', salary = '$5000')
```

OUTPUT

```
{'name': 'Teo', 'job': 'Developer', 'salary': '$5000'}
```

5. Nested function:



Một hàm có thể được định nghĩa trong 1 hàm khác



```
def out(a, b):  
    def inner(c, d):  
        return (c + d)  
    return inner(a, b)
```

```
print(out(1, 2))
```

OUTPUT

3