# HÀM (FUNCTION) TRONG PYTHON

- **Hàm là một khối mã chỉ chạy khi nó được gọi.**
- **Bạn có thể truyền dữ liệu, được gọi là tham số vào một hàm.**
- **Hàm có thể trả về dữ liệu hoặc không.**

## 1. Tạo hàm

**Trong Python, một hàm được định nghĩa bằng từ khóa *def* :**

In [ ]:
```
def my_function():
  print("Hello from a function")
```

## 2. Gọi hàm

**Để gọi một hàm, hãy sử dụng tên hàm theo sau dấu ngoặc đơn:**

In [ ]:
```
def my_function():
      print("Hello from a function")

my_function()
```

```
Hello from a function
```

## 3. Đối số

- **Thông tin có thể được truyền vào các hàm dưới dạng đối số.**
- **Các đối số được chỉ định sau tên hàm, bên trong dấu ngoặc đơn. Bạn có thể thêm bao nhiêu đối số tùy thích, và ngăn cách chúng bằng dấu phẩy.**

**Ví dụ: Có một hàm với một đối số (fname). Khi hàm được gọi, chúng ta đặt vào giá trị của fname, và giá trị này được sử dụng bên trong hàm để in ra tên đầy đủ.**

In [ ]:
```
def my_function(fname):
      print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

```
Emil Refsnes
Tobias Refsnes
Linus Refsnes
```

***Đối số trong Python thường được viết tắt là args***

## 4. Tham số và Đối số

- **Các thuật ngữ tham số và đối số có thể được sử dụng cho cùng một thứ, đó là thông tin được truyền vào một hàm.**

- **Từ quan điểm của một chức năng:**
  **+ Tham số là biến được liệt kê bên trong dấu ngoặc đơn trong định nghĩa hàm.**
  **+ Đối số là giá trị được gửi đến hàm khi nó được gọi.**

**5. Số lượng đối số**

**Theo mặc định, một hàm phải được gọi với số đối số chính xác. Có nghĩa là nếu hàm của bạn yêu cầu 2 đối số, bạn phải gọi hàm có 2 đối số, không nhiều hơn và không ít hơn.**

**Ví dụ: Hàm sau có 2 đối số và lấy ra giá trị từ 2 đối số:**

In [ ]:
```
def my_function(fname, lname):
      print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

```
Emil Refsnes
```

***Nếu bạn cố gắng gọi một hàm có nhiều hơn hoặc ít hơn so với số lượng tham số của hàm thì bạn sẽ nhận được thông báo lỗi từ Python***

**Ví dụ: Hàm sau có 2 đối số nhưng chỉ truyền vào 1 giá trị:**

In [ ]:
```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[4], line 4
      1 def my_function(fname, lname):
      2   print(fname + " " + lname)
----> 4 my_function("Emil")

TypeError: my_function() missing 1 required positional argument: 'lname'
```

**6. Đối số tuỳ ý, *args</br>**

- **Nếu bạn không biết có bao nhiêu đối số sẽ được truyền vào hàm của mình, hãy thêm một ký tự * trước tên tham số trong định nghĩa hàm.**
- **Bằng cách này, hàm sẽ nhận được một bộ đối số và có thể truy cập các giá trị một cách tương ứng:**

**Ví dụ: Nếu số lượng đối số không xác định thì bạn hãy thêm ký tự * trước tên tham số:**

In [ ]:
```
def my_function(*kids):
      print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

```
The youngest child is Linus
```

***Đối số tùy ý thường được rút ngắn thành *args trong tài liệu Python.***

**7. Đối số từ khoá**

- **Bạn cũng có thể gửi đối số bằng cú pháp key = value.**
- **Với cách này thì thứ tự của các đối số không quan trọng.**

In [ ]:
```
def my_function(child3, child2, child1):
      print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

```
The youngest child is Linus
```

***Cụm từ Đối số từ khóa thường được rút ngắn thành kwargs trong tài liệu Python.***

**8. Đối số từ khóa tùy ý, **kwargs**

- **Nếu bạn không biết có bao nhiêu đối số từ khóa sẽ được truyền vào hàm thì hãy thêm ký tự ** trước tên tham số trong định nghĩa hàm.**
- **Bằng cách này, hàm sẽ nhận được một từ điển các đối số và có thể truy cập các mục tương ứng:**

In [ ]:
```
def my_function(**kid):
      print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

```
His last name is Refsnes
```

***Đối số Kword tùy ý thường được rút ngắn thành **kwargs trong tài liệu Python.***

**9. Giá trị tham số mặc định**

- **Nếu trong lời gọi hàm có truyền giá trị cho tham số tương ứng thì hàm sẽ dùng giá trị đó để tính toán,**
- **Ngược lại, nếu không truyền giá trị cho tham số thì hàm sẽ dùng giá trị mặc định để tính toán:**

In [ ]:
```
def my_function(country = "Norway"):
      print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

```
I am from Sweden
I am from India
I am from Norway
I am from Brazil
```

**10. Dùng đối tượng List như đối số**

**Bạn có thể dùng bất kỳ kiểu dữ liệu nào của đối số định nghĩa hàm (string,**

**int/float, List, dictionary,...) và nó sẽ được xem như là cùng một loại dữ liệu bên trong hàm.**

In [ ]:
```
def my_function(food):
    for x in food:
        print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

```
apple
banana
cherry
```

**11. Giá trị trả về Để cho phép một hàm trả về một giá trị, hãy sử dụng câu lệnh *return***

In [ ]:
```
def my_function(x):
      return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

```
15
25
45
```

**12. Câu lệnh *pass***

**Phần định nghĩa hàm không thể để trống, nhưng nếu vì lý do nào đó bạn có một định nghĩa hàm không có nội dung, hãy đưa vào câu lệnh *pass* để tránh gặp lỗi.**

In [ ]:
```
def myfunction():
  pass
```

**13. Hàm đệ quy**

- **Python cũng chấp nhận hàm đệ quy, nghĩa là một hàm đã xác định có thể gọi lại chính nó.**
- **Đệ quy là một khái niệm toán học và lập trình phổ biến. Nó có nghĩa là một hàm gọi chính nó. Điều này có lợi là bạn có thể lặp qua dữ liệu để đạt được kết quả.**
- **Cần cẩn thận với đệ quy vì có thể khá dễ dàng viết một hàm không bao giờ kết thúc hoặc một hàm sử dụng quá nhiều bộ nhớ hoặc sức mạnh của bộ xử lý. Tuy nhiên, khi được viết chính xác, đệ quy có thể là một cách tiếp cận lập trình rất hiệu quả và thanh lịch về mặt toán học.**

**Ví dụ: Hàm tri_recursion() là một hàm mà chúng ta đã xác định để gọi chính nó ("recurse"). Chúng ta sử dụng biến k làm dữ liệu, giá trị này giảm dần ( -1 ) mỗi khi chúng ta lặp lại. Đệ quy kết thúc khi điều kiện k không lớn hơn 0 (tức là khi nó bằng 0).**

In [ ]:
```
def tri_recursion(k):
  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

print("\n\nRecursion Example Results")
tri_recursion(6)
```

```
Recursion Example Results
1
3
6
10
15
21
```

Out[ ]: 21

---

**Biên dịch: Nguyễn Ngọc Dương**

**Facebook: https://www.facebook.com/danh.kean.353803/**

**Email: ngocduong5642@gmail.com**