Functions

Ba Nguyễn



Functions

Hàm (phương thức - function) được sử dụng để đóng gói các khối mã (code block), cho phép tái sử dụng mã ở nhiều nơi, giảm thiểu code, giảm lỗi, ...

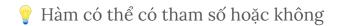
Cú pháp khai báo hàm:

```
def func_name(parameter_list):
    pass
```

Cú pháp gọi hàm:

```
func_name(argument_list)
```

Pham số (parameter) khai báo các giá trị (biến) được sử dụng bên trong hàm, các cuộc gọi hàm cung cấp các giá trị tương ứng (argument) với các tham số được khai báo





Functions

```
def greet():
    print("Hi there")
    print("Welcome")
greet()
def greet(first_name, last_name): # two parameters: first_name, last_name
    print(f"Hi, {first_name} {last_name}")
    print("Welcome")
greet("Ba", "Nguyễn") # first name = "Ba", last name = "Nguyễn"
```



Return

- Có 2 loại hàm
 - Thực thi một nhiệm vụ nào đó
 - Tính toán và trả về một giá trị
- Mặc định hàm luôn trả về một giá trị (None), để chỉ định một giá trị trả về từ hàm, sử dụng câu lệnh return

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

print(sum(1, 5)) # 6
```

Prophet câu lệnh return dừng (ngắt) hàm, các câu lệnh phía dưới bị bỏ qua



Type Annotation, Docstring

Fig. Khai báo hàm nên sử dụng type annotation và docstring để ghi chú rõ ràng chức năng của hàm, các tham số truyền vào có kiểu dữ liệu gì, loại dữ liệu trả về

```
def sum(a: int, b: int) → int:
    """

    Return sum of two number
    Parameters:
    - a: integer
    - b: integer
    Return: a + b → integer
    """
    return a + b
```

```
(function) sum: (a: int, b: int) → int
Return sum of two number
Parameters:
• a: integer
• b: integer
Return: a + b -> integer
```



Scope

Trong Python, có 2 kiểu của biến, tương ứng với phạm vi (scope) tồn tại của biến

- Biến cục bộ function scope (được khai báo trong một function)
- Biến toàn cục file scope (được khai báo bên ngoài tất cả các function)

Biến cục bộ chỉ tồn tại bên trong hàm mà nó được khai báo, ngược lại, biến toàn cục có thể được gọi ở bất kỳ đâu, bên trong bất kỳ hàm nào

```
first_name = "Ba"

def greet():
    last_name = "Nguyễn"
    print(f"Hello, {first_name} {last_name}")

greet()
```



Scope

- 💡 Python không có block scope giống như một số ngôn ngữ khác 😍
- Mặc định việc thay đổi giá trị một biến toàn cục bên trong hàm không ảnh hưởng đến biến toàn cục (trừ khi khai báo biến bên trong hàm với từ khóa global don't do this)

```
first_name = "Ba"
last_name = "Nguyễn"
def greet():
    first_name = "Béo"
    global last_name
    last_name = "Ú"

print(first_name, last_name) # Ba Ú
```



Default parameter, keyword arguments

Có thể gán một giá trị mặc định cho các tham số hàm, giá trị mặc định sẽ được sử dụng nếu cuộc gọi hàm không cung cấp đối số tương ứng

Ngoài ra, trong cuộc gọi hàm, có thể sử dụng *keyword arguments* để chỉ định rõ ràng giá trị sẽ được truyền cho tham số nào của hàm

```
def greet(first_name: str = "Ba", last_name: str = "Nguyễn") → None:
    print(f"Hello, {first_name} {last_name}")

greet() # Hello, Ba Nguyễn
greet(first_name="Béo Ú") # Hello, Béo Ú Nguyễn
greet(last name="Còi") # Hello, Ba Còi
```



*args

Cú pháp *args cho phép truyền vào hàm số lượng đối số bất kỳ, các đối số truyền vào được đặt trong một *tuple*

```
def multiply(*numbers):
    total = 1
    for number in numbers:
        total *= number
    return total
```

```
multiply(1, 2, 3, 4, 5, 6) # numbers = (1, 2, 3, 4, 5, 6) \rightarrow 720
```

Có thể dùng kết hợp cả *args và tham số hàm thông thường, tuy nhiên, *args phải đặt sau tham số, hoặc sử dụng keyword argument trong cuộc gọi hàm



**kwargs

Cú pháp **kwargs là biến thể của *args, khác biệt ở chỗ, các đối số trong cuộc gọi hàm được truyền vào với keyword argument và args là dictionaries, không phải tuple

```
def create_user(**user):
    print(user)
    print(user["name"])

create_user(id=1, name="Ba")
# user = {'id': 1, 'name': 'Ba'}
# Ba
```

Có thể dùng kết hợp cả *args và **kwargs và tham số thông thường, tuy nhiên, *args phải đặt trước **kwargs và đối số truyền vào cũng tương tự



Debugging

