BUỔI 2: NGÔN NGỮ LẬP TRÌNH PYTHON (2)

7. Hàm (Function)

Trong Python, một hàm là một khối mã có thể được gọi lại nhiều lần để thực hiện một tác vụ cụ thể. Hàm giúp chia nhỏ mã thành các phần nhỏ, dễ quản lý và tái sử dụng.

Cú pháp của một hàm:

```
def function_name(parameters):
    # Code của hàm
    return value
```

- **def**: Khai báo bắt đầu của một hàm.
- **function_name**: Tên của hàm, đặt theo quy tắc đặt tên biến.
- **parameters**: Các tham số (đầu vào) mà hàm có thể nhận.
- **return**: Trả về giá trị từ hàm (không bắt buộc).

Ví dụ về hàm:

- Hàm đơn giản:

```
def greet(name):
    return f"Hello, {name}!"

message = greet("Alice")
print(message)  # Output: "Hello, Alice!"
```

- Hàm nhiều tham số:

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

result = add_numbers(5, 3)
print(result)  # Output: 8
```

- Hàm không có giá trị trả về:

```
def say_hello():
    print("Hello!")

say_hello()  # Output: "Hello!"
```

- Hàm có giá trị mặc định:

```
def greet(name="User"):
return f"Hello, {name}!"
```

```
message = greet()
print(message)  # Output: "Hello, User!"
```

- Hàm với nhiều tham số tùy chọn:

```
def calculate_sum(*args):
    total = 0
    for num in args:
        total += num
    return total

result = calculate_sum(1, 2, 3, 4, 5)
print(result)  # Output: 15
```

Giải thích:

- Hàm giúp tạo ra các khối mã có thể tái sử dụng.
- Tham số là các giá trị mà hàm nhận để thực hiện công việc cụ thể.
- Giá trị trả về được sử dụng để trả về kết quả từ hàm. Nếu không có lệnh return, hàm sẽ trả về None.
- Hàm có thể có bất kỳ số lượng tham số nào, bao gồm cả không có tham số.
- Hàm có thể nhận giá trị đầu vào dưới nhiều dạng như các biến đơn, danh sách, hoặc thậm chí một số lượng không xác định các biến (dùng *args).
- Giá trị mặc định của tham số có thể được định nghĩa, điều này cho phép gọi hàm mà không cần truyền giá trị cho tất cả các tham số.
- Hàm có thể gọi trong bất kỳ phần nào của chương trình.

8. Lớp (Class)

Trong Python, một lớp là một mô hình để tạo ra các đối tượng. Đối tượng là một thể hiện cụ thể của một lớp. Lớp định nghĩa các thuộc tính (đặc điểm) và phương thức (hành động) của đối tượng.

Cú pháp của một lớp:

```
class ClassName:
    def __init__(self, parameters):
        self.attribute = parameters

    def method(self):
        # Code của phương thức
```

- class: Khai báo bắt đầu của một lớp.
- ClassName: Tên của lớp, đặt theo quy tắc đặt tên biến.

- __init__: Phương thức khởi tạo, được gọi khi một đối tượng mới của lớp được tạo ra.
- self: Đại diện cho chính đối tượng đó, cho phép truy cập các thuộc tính và phương thức của lớp.

Ví dụ về lớp:

- Lớp đơn giản:

```
class Person:
   def __init__(self, name, age):
        self.name = name
        self.age = age

    def introduce(self):
        return f"My name is {self.name} and I am {self.age} years old."

alice = Person("Alice", 30)
print(alice.introduce())  # Output: "My name is Alice and I am 30 years old."
```

- Kế thừa lớp:

```
class Employee(Person):
   def __init__(self, name, age, employee_id):
        super().__init__(name, age)
        self.employee_id = employee_id

    def introduce(self):
       return f"My name is {self.name}, I am {self.age} years old, and my employee
ID is {self.employee_id}."

john = Employee("John", 35, "EMP123")
print(john.introduce())
```

- Kế thừa đa hình:

```
class Animal:
    def make_sound(self):
        pass

class Dog(Animal):
    def make_sound(self):
        return "Bark"

class Cat(Animal):
    def make_sound(self):
        return "Meow"
```

```
class Duck(Animal):
    def make_sound(self):
        return "Quack"

def animal_sounds(animal):
    return animal.make_sound()

dog = Dog()
cat = Cat()
duck = Duck()

print(animal_sounds(dog))  # Output: "Bark"
print(animal_sounds(cat))  # Output: "Meow"
print(animal_sounds(duck))  # Output: "Quack"
```

Giải thích:

- Lớp giúp tạo ra các đối tượng có các đặc điểm và hành động tương ứng.
- __init__ là phương thức khởi tạo, được gọi khi một đối tượng mới của lớp được tạo ra. Nó cũng có thể khởi tạo các thuộc tính của đối tượng.
- self là một đối tượng đặc biệt trong Python, đại diện cho chính đối tượng đó. Khi gọi một phương thức của lớp, self cho phép truy cập các thuộc tính và phương thức khác của lớp.
- Kế thừa cho phép một lớp mới (lớp con) kế thừa các thuộc tính và phương thức của lớp đã có (lớp cha).
- Kế thừa đa hình (Polymorphism) cho phép các đối tượng có thể thực hiện các hành động khác nhau dựa trên cùng một phương thức.