**Object Orientated Programming (OOP)**

**What is Object-Oriented Programming (OOP)?**

OOP is a programming paradigm that is based on the concept of **objects**. Objects are instances of **classes**, and classes are like blueprints or templates for creating objects.

In Python, everything is an object, and you can define your own custom objects using **classes**. Let's break it down:

* **Class**: A class is a blueprint for creating objects. It defines the attributes (properties) and methods (functions) that the objects created from the class will have.
* **Object**: An object is an instance of a class. When you create an object, you're instantiating a class.

**Key Concepts in OOP:**

1. **Attributes**: These are the properties or characteristics of an object. In Python, attributes are variables that are part of a class or an object.
2. **Methods**: These are functions that are defined inside a class and are used to perform operations on the attributes or interact with the object.
3. **Instantiation**: The process of creating an object from a class is called instantiating a class.
4. **Self**: The self keyword is used to represent the instance of the class. It allows you to access attributes and methods within the class.

**Let's Start with a Simple Example: Creating a Person Class**

**Step 1: Define the Class**

To define a class, we use the class keyword followed by the class name (in this case, Person).

class Person:

# This is the class definition

def \_\_init\_\_(self, name, age, city):

# This is the initializer method (constructor)

self.name = name

self.age = age

self.city = city

def greet(self):

# This is a method that prints a greeting message

print(f"Hello, my name is {self.name}, I'm {self.age} years old, and I live in {self.city}.")

**Step 2: Understanding the Code**

* **\_\_init\_\_ method**: This is a special method in Python (called a constructor). It's automatically called when you create an instance of a class. It initializes the attributes of the object (like name, age, and city).
* **self**: The self parameter is used to refer to the current instance of the class. It allows you to access attributes and methods that belong to the instance.
* **greet method**: This is a normal method that can be called on the object. It uses the attributes of the class to print a greeting.

**Step 3: Creating Objects (Instantiating the Class)**

Now, you can create objects (instances) of the Person class by passing values to the constructor.

# Creating instances (objects) of the Person class

person1 = Person("Alice", 30, "New York")

person2 = Person("Bob", 25, "Los Angeles")

# Calling the greet method

person1.greet()

person2.greet()

**Expected Output:**

Hello, my name is Alice, I'm 30 years old, and I live in New York.

Hello, my name is Bob, I'm 25 years old, and I live in Los Angeles.

**Breakdown of Concepts:**

1. **Class Definition**:
   * class Person: defines a new class called Person.
2. **Constructor (\_\_init\_\_)**:
   * The constructor is a special method that initializes the object's attributes. It is automatically called when you create an instance of the class.
   * The constructor has parameters (name, age, city) that are used to set the attributes of the object (self.name, self.age, self.city).
3. **Attributes**:
   * self.name, self.age, and self.city are attributes of the class. These are variables that are associated with an instance of the class.
4. **Methods**:
   * greet(self) is a method that can be called on an instance of the Person class. It accesses the instance's attributes and performs an action, in this case, printing a greeting.
5. **Instantiation**:
   * When you create an object using person1 = Person("Alice", 30, "New York"), you are instantiating the Person class, which means creating an object from the Person class blueprint.

**More Advanced Concepts in OOP:**

Once you're comfortable with the basics, there are a few more concepts in OOP that are useful to know:

1. **Encapsulation**: Keeping the details of how a class works hidden from the outside world, and only exposing necessary methods and properties.
   * Example: You can make attributes private (using self.\_\_attribute) to prevent direct modification.
2. **Inheritance**: A way to create new classes based on existing ones, inheriting attributes and methods from a parent class.
   * Example: If you have a Student class that inherits from Person, it will have all the Person attributes and methods but can also have its own unique attributes and methods.
3. **Polymorphism**: The ability of a method to behave differently based on the object it is acting upon.
4. **Abstraction**: Hiding the complexity of the implementation and showing only the necessary details to the user.

(SEE ROBOT DOG STUFF)