**Objects in Python Programming**

In Python, **objects** are the core building blocks of the language. Everything in Python—whether it's a number, string, list, dictionary, function, or even a class—is an object.

**1. What is an Object?**

An **object** is an instance of a class that contains **data (attributes)** and **behavior (methods)**. Each object is associated with:

* **A unique identity** (like an address in memory)
* **A type (class it belongs to)**
* **Attributes (characteristics of the object)**
* **Methods (functions associated with the object)**

**Example: A Simple Object**

class Car:

def \_\_init\_\_(self, brand, color):

self.brand = brand # Attribute

self.color = color # Attribute

def start(self): # Method

return f"{self.brand} car is starting!"

# Creating an object (instance of Car)

my\_car = Car("Toyota", "Red")

# Accessing attributes

print(my\_car.brand) # Output: Toyota

print(my\_car.color) # Output: Red

# Calling a method

print(my\_car.start()) # Output: Toyota car is starting!

**2. Object Characteristics**

Every object in Python has three key characteristics:

1. **Identity**: A unique identifier (memory address)
2. print(id(my\_car)) # Unique memory address
3. **Type**: Defines what kind of object it is
4. print(type(my\_car)) # Output: <class '\_\_main\_\_.Car'>
5. **State (Attributes)**: The properties of the object
6. print(my\_car.color) # Output: Red

**3. Object-Oriented Features in Python**

Objects in Python are part of **Object-Oriented Programming (OOP)**, which includes:

**a) Class and Object Relationship**

* **Class**: A blueprint for creating objects.
* **Object**: A specific instance of a class.

class Dog:

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

self.name = name

dog1 = Dog("Buddy") # Object of Dog class

**b) Encapsulation**

Encapsulation means bundling data (attributes) and methods together within a class.

class Person:

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

self.\_\_name = name # Private attribute

self.\_\_age = age # Private attribute

def get\_info(self):

return f"Name: {self.\_\_name}, Age: {self.\_\_age}"

person = Person("Alice", 25)

print(person.get\_info()) # Output: Name: Alice, Age: 25

**c) Inheritance**

Inheritance allows a class to **inherit** methods and properties from another class.

class Animal:

def speak(self):

return "Animal makes a sound"

class Dog(Animal): # Dog inherits from Animal

def speak(self):

return "Bark!"

dog = Dog()

print(dog.speak()) # Output: Bark!

**d) Polymorphism**

Polymorphism allows methods in different classes to have the same name but different behavior.

class Cat:

def speak(self):

return "Meow!"

animals = [Dog(), Cat()]

for animal in animals:

print(animal.speak()) # Output: Bark! Meow!

**4. Built-in Objects in Python**

Python provides many built-in objects, such as:

* **Numeric Objects**: int, float, complex
* **Sequence Objects**: list, tuple, str, range
* **Mapping Objects**: dict
* **Set Objects**: set, frozenset
* **Boolean Objects**: bool
* **File Objects**: open()
* **User-defined Objects**: Created using class

Example:

num = 10 # num is an instance of int

print(type(num)) # Output: <class 'int'>

**5. Object Memory Management in Python**

Python uses **Garbage Collection (GC)** to manage memory automatically. When an object is no longer referenced, it is deleted by the garbage collector.

import gc

print(gc.isenabled()) # Checks if garbage collection is enabled

**Conclusion**

* Objects are instances of classes.
* They have attributes (data) and methods (behavior).
* Python supports OOP principles like **Encapsulation, Inheritance, and Polymorphism**.
* Everything in Python is an object!