Day 8: Object-Oriented Programming (OOP) Basics in Python

Topics Covered:

- *Introduction to OOP*
- *Classes and Objects*
- *Attributes and Methods*
- *Constructors*
- *Self Keyword*
- *Inheritance in OOP*

1. Introduction to OOP:

Object-Oriented Programming (OOP) is a programming paradigm that organizes code into objects. OOP focuses on encapsulating data and the methods that operate on that data into a single unit called a class. The main principles of OOP are encapsulation, inheritance, and polymorphism.

2. Classes and Objects:

A class is like a blueprint for creating objects. An object is an instance of a class. Each object can have attributes (data) and methods (functions).

Example:

```
class Dog:
    def __init__(self, name, breed):
        self.name = name
        self.breed = breed
    def bark(self):
        return f"{self.name} says woof!"
dog1 = Dog("Buddy", "Golden Retriever")
print(dog1.bark())  # Output: Buddy says woof!
```

3. Attributes and Methods:

Attributes are the variables that belong to an object, and methods are the functions that belong to a class.

Example:

```
dog1.name  # Output: Buddy
dog1.bark()  # Output: Buddy says woof!
```

4. Constructors:

A constructor is a special method used to initialize objects. In Python, the constructor method is `__init__`.

Example:

```
class Dog:
    def __init__(self, name, breed):
        self.name = name
        self.breed = breed
dog2 = Dog("Lucy", "Beagle")
print(dog2.name)  # Output: Lucy
```

5. Self Keyword:

The `self` keyword in Python refers to the instance of the class. It allows you to access instance variables and methods.

Example:

```
class Dog:
    def __init__(self, name):
        self.name = name
    def speak(self):
        return f"{self.name} barks!"
dog3 = Dog("Max")
dog3.speak()  # Output: Max barks!
```

6. Inheritance in OOP:

Inheritance allows a class to inherit attributes and methods from another class. This promotes code reuse and organization.

Example:

```
class Animal:
    def __init__(self, name):
        self.name = name
    def speak(self):
        return f"{self.name} makes a sound!"
class Dog(Animal):
    def speak(self):
        return f"{self.name} barks!"
dog4 = Dog("Charlie")
print(dog4.speak())  # Output: Charlie barks!
```

Summary of Day 8:

Today, we covered the basics of Object-Oriented Programming (OOP) in Python. We learned about classes, objects, attributes, and methods. We also explored constructors and the `self` keyword, and understood how inheritance allows code reuse by extending classes.