Object-Oriented Programming (OOP) is a programming paradigm that uses objects, which are instances of classes, for organizing code. Python is an object-oriented programming language, and it supports the principles of OOP. Here are some key concepts in Python's implementation of OOP:

1. **Classes and Objects:**
   - A class is a blueprint for creating objects. It defines attributes (data) and methods (functions) that operate on the data.
   - An object is an instance of a class. You can create multiple objects from the same class

```
class Car:
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def display_info(self):
        print(f"{self.make} {self.model}")

# Creating an object of the Car class
my_car = Car("Toyota", "Camry")
my_car.display_info()
```

Encapsulation:

- It involves bundling the data (attributes) and the methods that operate on the data into a single unit (class).
- Use of private and protected access modifiers (__ and _ prefixes) to control access to class members.

```
class BankAccount:
    def __init__(self, balance):
        self.__balance = balance  # private attribute

    def get_balance(self):
        return self.__balance

account = BankAccount(1000)
```

```
print(account.get_balance())  # Accessing balance through a method
```

Inheritance:

- It allows a new class (subclass or derived class) to inherit attributes and methods from an existing class (base class or parent class).
- Helps in code reusability.

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

class Dog(Animal):
    def speak(self):
        return "Woof!"

my_dog = Dog()
print(my_dog.speak())  # Outputs: Woof!
```

Polymorphism:

- It allows objects of different classes to be treated as objects of a common base class.
- Achieved through method overriding.

python

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

my_cat = Cat()
print(my_cat.speak())  # Outputs: Meow!
```