Python Object Oriented Programming

Main Concepts of Object-Oriented Programming (OOPs)
Class
Objects
Polymorphism
Encapsulation
Inheritance
Data Abstraction
Python is a versatile programming language that supports various programming styles, including object-oriented programming (OOP) through the use of objects and classes.

An object is any entity that has attributes and behaviors. For example, a parrot is an object. It has

attributes - name, age, color, etc.
behavior - dancing, singing, etc.
Similarly, a class is a blueprint for that object.

PYTHON CLASS AND OBJECT:

```
class Parrot:

    # class attribute
    name = ""
    age = 0

# create parrot1 object
parrot1 = Parrot()
parrot1.name = "Blu"
parrot1.age = 10

# create another object parrot2
parrot2 = Parrot()
parrot2.name = "Woo"
parrot2.age = 15

# access attributes
print(f"{parrot1.name} is {parrot1.age} years old")
print(f"{parrot2.name} is {parrot2.age} years old")
```

Run Code
Output

Blu is 10 years old
Woo is 15 years old

In the above example, we created a class with the name Parrot with two attributes: name and age.

Then, we create instances of the Parrot class. Here, parrot1 and parrot2 are references (value) to our new objects.

We then accessed and assigned different values to the instance attributes using the objects name and the . notation.

To learn more about classes and objects, visit Python Classes and Objects

PYTHON INHERITANCE:

Inheritance is a way of creating a new class for using details of an existing class without modifying it.

The newly formed class is a derived class (or child class). Similarly, the existing class is a base class (or parent class).

Example 2: Use of Inheritance in Python

```
# base class
class Animal:

    def eat(self):
        print( "I can eat!")

    def sleep(self):
        print("I can sleep!")

# derived class
class Dog(Animal):

    def bark(self):
        print("I can bark! Woof woof!!")

# Create object of the Dog class
dog1 = Dog()

# Calling members of the base class
dog1.eat()
dog1.sleep()
```

```
# Calling member of the derived class
dog1.bark();
```
Run Code
Output

```
I can eat!
I can sleep!
I can bark! Woof woof!!
```
Here, dog1 (the object of derived class Dog) can access members of the base class Animal. It's because Dog is inherited from Animal.

```
# Calling members of the Animal class
dog1.eat()
dog1.sleep()
```
To learn more about inheritance, visit Python Inheritance.

PYTHON ENCAPSULATION:

Encapsulation is one of the key features of object-oriented programming. Encapsulation refers to the bundling of attributes and methods inside a single class.

It prevents outer classes from accessing and changing attributes and methods of a class. This also helps to achieve data hiding.

In Python, we denote private attributes using underscore as the prefix i.e single _ or double __. For example,

```
class Computer:

    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Selling Price: {}".format(self.__maxprice))

    def setMaxPrice(self, price):
        self.__maxprice = price

c = Computer()
c.sell()

# change the price
```

```
c.__maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()
```

Run Code

Output

```
Selling Price: 900
Selling Price: 900
Selling Price: 1000
```

In the above program, we defined a Computer class.

We used __init__() method to store the maximum selling price of Computer. Here, notice the code

```
c.__maxprice = 1000
```

Here, we have tried to modify the value of __maxprice outside of the class. However, since __maxprice is a private variable, this modification is not seen on the output.

As shown, to change the value, we have to use a setter function i.e setMaxPrice() which takes price as a parameter.

POLYMORPHISM:

Polymorphism is another important concept of object-oriented programming. It simply means more than one form.

That is, the same entity (method or operator or object) can perform different operations in different scenarios.

Let's see an example,

```
class Polygon:
    # method to render a shape
    def render(self):
        print("Rendering Polygon...")

class Square(Polygon):
    # renders Square
    def render(self):
        print("Rendering Square...")
```

```
class Circle(Polygon):
    # renders circle
    def render(self):
        print("Rendering Circle...")

# create an object of Square
s1 = Square()
s1.render()

# create an object of Circle
c1 = Circle()
c1.render()
```

Run Code

Output

```
Rendering Square...
Rendering Circle...
```