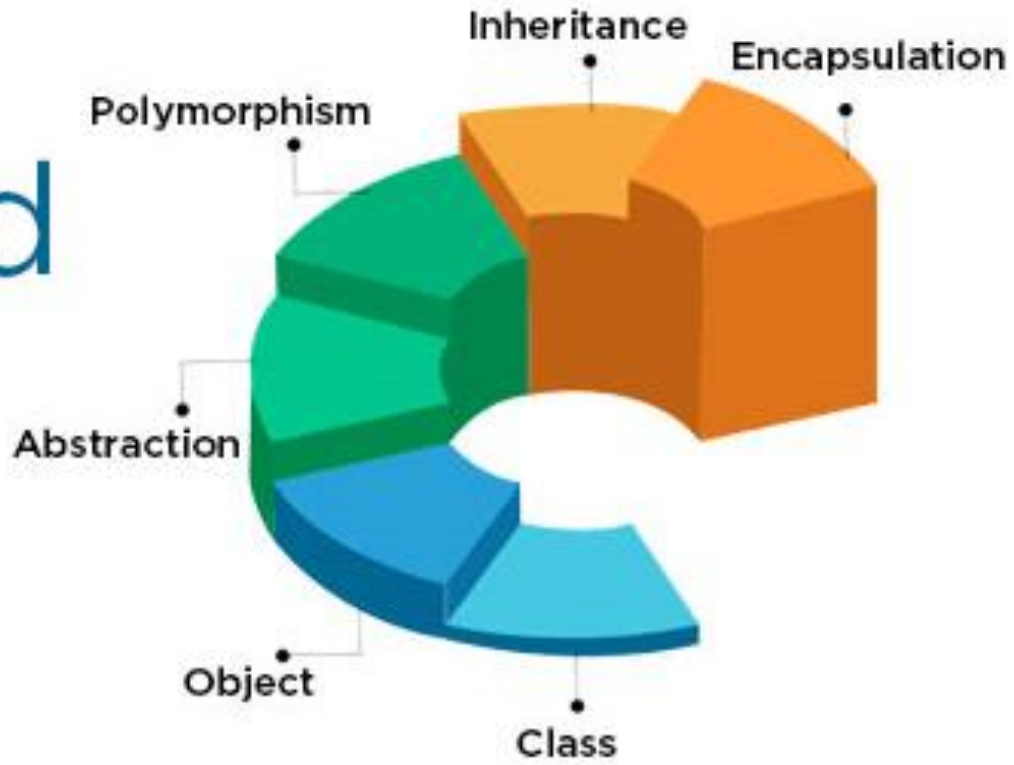
Object Oriented Programming with Python



WHAT IS OOPS IN PYTHON?

Python is an object-oriented programming language. In Python, almost everything is an object and has its properties and methods. Here a class is like an object constructor, or a “blueprint” for creating objects.

Object-oriented programming (OOP) is a methodology that focuses on classes and objects. The concept of OOP in Python focuses on building efficient and reusable code.

This is also known as DRY (don't repeat yourself). Every individual object represents a different part of the application having its logic.

For instance, an object could represent a car with a name, model, and color as attributes.

The four major OOP concepts are:

Inheritance: A process of using details from a new class without modifying an existing class.

Polymorphism: A concept of using common operations in different ways for different data inputs.

Abstraction: Handling complexity by hiding unnecessary information from the user

Encapsulation: Hiding the private details of a class from other objects.

WHAT IS A CLASS?

The class can be defined as a collection of objects that define the common attributes and behaviors of all the objects.

Class is defined under a “Class” keyword.

Example:

```
In [3]: #defining a class

        class car:
            pass
```

WHAT IS AN OBJECT?

The object is an entity that has a state and behavior. It is an instance of a class that can access the data.

Example:

```
In [3]: #defining a class
        class car:
            pass

        #defining an object
        obj = car()
```

THE _INIT_METHOD

The _init_ method is run as soon as an object of a class is created. This method is useful for passing initial value to your objects.

Example:

```
In [9]: class employee():
            def __init__(self,name,age,id,salary):   #creating a function
                self.name = name                     # self is an instance of a class
                self.age = age
                self.salary = salary
                self.id = id
```

1. INHERITANCE

This refers to defining a new class with little or no modification to an existing class

The new class is known as a derived (or child) class, and the one from which it inherits is known as the base (or parent) class.

The derived class inherits features from the base class, which adds new features to it.

Example:

We have created a class called childemployee, which inherits the properties of a parent class, and emp1 is passed as a parameter.

```
class employee():       #This is a parent class
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary

class childemployee(employee):   #This is a child class
    def __init__(self, name, age, salary,id):
        self.name = name
        self.age = age
        self.salary = salary
        self.id = id
emp1 = employee('jack',19,1400)

print(emp1.age)
```

```
19
```

2. POLYMORPHISM

In OOP, polymorphism refers to a programming language's ability to process objects in more than one form. It is the ability to redefine methods for derived classes.

Example:

In the preceding example, we have created two classes—childemployee1 and childemployee2—which are derived from the base class employee.

```
In [15]: class employee():
             def __init__(self,name,age,id,salary):
                 self.name = name
                 self.age = age
                 self.salary = salary
                 self.id = id
         def earn(self):
                 pass

         class childemployee1(employee):

             def earn(self):#Run-time polymorphism
                 print("Hello")

         class childemployee2(employee):

             def earn(self):
                 print("Hello there")

         a = childemployee1
         a.earn(employee)
         b = childemployee2
         b.earn(employee)

         Hello
         Hello there
```

3. ABSTRACTION

Abstraction is a programming methodology in which details of the programming code are hidden from the user—only the essential information is displayed. Abstraction focuses on ideas, rather than events.

Examples:

In the preceding example, we have imported an abstract method. An object is created for childemployee base class, and the functionality of the abstract class is used.

```
In [18]: from abc import ABC,abstractmethod
         class employee(ABC):
             def emp_id(self,id,name,age,salary):    #Abstraction
                 pass

         class childemployee(employee):
             def emp_id(self,id):
                 print("emp_id is 1")

         emp1 = childemployee()
         emp1.emp_id(id)

         emp_id is 1
```

4. ENCAPSULATION

The binding of data and function into one single unit is known as encapsulation. Encapsulation keeps the data safe from misuse and changes in the code can be done independently.

Example:

In the class “Person,” there are two variables, and we have accessed those variables directly and through a method.

```
In [23]: class Person:
             def __init__(self, name, age=0):
                 self.name = name
                 self.age = age

             def display(self):
                 print(self.name)
                 print(self.age)

         person = Person('Sam', 20)
         #accessing using class method
         person.display()
         #accessing directly from outside
         print(person.name)
         print(person.age)
```