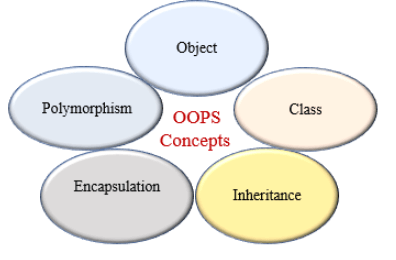


**What is OOPS**

**Object-Oriented Programming System (OOPS)** is a most used fundamental programming paradigm used by developers. It aims to mock real world entities in programming. The main concept of OOPs is to bind the data and the functions that work on that together as a single unit so that no other part of the code can access this data.

**Main Concepts of Object-Oriented Programming (OOPs)**



**1. Class**

A class is a collection of objetcs. We can tell it contain the prototype of objects. It is a logical entity that contains some attributes and methods.

Some points on Python class:

* Classes are created by keyword class.
* Attributes are the variables that belong to a class.
* Attributes are always public and can be accessed using the dot (.) operator. Eg.: Myclass.Myattribute

|  |  |
| --- | --- |
| Class Definition Syntax:  class ClassName:  # Statement-1  …………………..  # Statement-N | Ex: class Person:  pass |

**2. Objects**

The object is an entity that has a state and behavior associated with it. It can be any real-world object like a mouse, keyboard, chair, table, pen, etc. Integers, strings, floating-point numbers, even arrays, and dictionaries, are all objects. More specifically, any single integer or any single string is an object. The number 8 is an object, the string “Hello, world” is an object. Thers will be object which can hold other objects, and so on.

An object consists of:

* **State**: It is represented by the attributes of an object. It also reflects the properties of an object.
* **Behavior**: It is represented by the methods of an object. It also reflects the response of an object to other objects.
* **Identity**: It gives a unique name to an object and enables one object to interact with other objects.

For example we will cosider person object. All person will have a first name and a last name right ? Also age. We can define the class as below:

class Person:

  def \_\_init\_\_(self, fname, lname):  
    self.firstname = fname  
    self.lastname = lname

def printName(self):  
    print(self.firstname, self.lastname)

We can create the object of above class as :

P1 = Person(‘Vinay, ‘Kumar’)

P2 = Person(‘John’, ‘Albert’)

P1. printName() // Will print Vinay Kumar

**Self**

* Class methods must have an extra first parameter in the method definition. This is the object itself and we do not give a value for this parameter when we call the method as Python provides it
* If we have a method that takes no arguments, then we still have to have one argument, which is self.

For more info visit [self in Python class](https://www.geeksforgeeks.org/self-in-python-class/)

**\_\_init\_\_ method**

The \_\_init\_\_ is like a constructor in C++ and Java, that means which will get called while creating an object. The method is useful to do any initialization you want to do with your object. In the aove example

**3. Encapsulation**

[Encapsulation](https://en.wikipedia.org/wiki/Encapsulation_(computer_programming)) in Python describes the concept of **bundling data and**[methods](https://pynative.com/python-instance-methods/)**within a single unit**. So, for example, when you create a [class](https://pynative.com/python-classes-and-objects/), it means you are implementing encapsulation as it binds all the data members ([instance variables](https://pynative.com/python-instance-variables/)) and methods into a single unit.

**4. Inheritance**

Inheritance is the capability of one class to derive or inherit the properties from another class. The class that derives properties is called the derived class or child class and the class from which the properties are being derived is called the base class or parent class. The benefits of inheritance are:

* It represents real-world relationships well.
* It provides the reusability of a code. We don’t have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
* It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A

class Employee(Person):

def \_\_init\_\_(self, fname, lname, idnumber, salary, post):

self.idnumber = idnumber

self.salary = salary

self.post = post

# invoking the \_\_init\_\_ of the parent class

Person.\_\_init\_\_(self, fname, lname)

def printDetails(self):

print("My name is {} {}".format(self.firstname, self.lastname))

print("IdNumber: {}".format(self.idnumber))

print("Post: {}".format(self.post))

e1 = Employee('Aaryan', 'Nambiar', 3, 10000, 'Intern')

e1.printDetails()

In the above article, we have created two classes i.e. Person (parent class) and Employee (Child Class). The Employee class inherits from the Person class. We can use the methods of the person class through employee class as seen in the printName function in the above code. A child class can have its on method like printDetails().

**5. Polymorphism**

Polymorphism simply means having many forms. For example there can be many type of employees.

Here Employee is derived from Person and GovtEmployee derived from Employee. So GovtEmployee will have all features of person class and Employee.

Employee

Freelancer

Private

Government

We can create the classes as below:

class GovtEmployee(Employee):

def printDetails(self):

print("I am a government employee!!")

class PrivateEmployee(Employee):

def printDetails(self):

print("I am a private employee!!")

class freelancer(Employee):

def printDetails(self):

print("I am a freelancer!!")

g = GovtEmployee('Arun', 'Mathew', 3134, 50000, 'Bank')

g.printName()

g.printDetails()

p = PrivateEmployee('Ajay', 'Sing', 1253, 100000, 'IT')

p.printName()

p.printDetails()