**Session 3**

Introduction to Object-Oriented Programming

## The OOP's based approach.

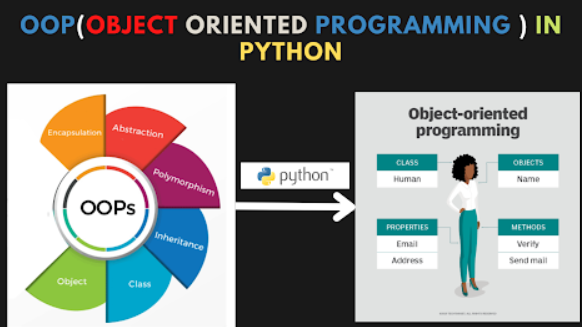
After solving the above practice problem, you might have noticed how difficult it is to maintain all the variables associated with different balls and keep track of them.

This is a good example where the OOP's based approach can come in handy. We can create circle objects using the circle class.

We can start by identifying the attributes of a circle so that we can define these inside the class itself.

Let’s take an example of one of the OOPs concepts with real-time examples: If you had a class called “Expensive Cars,” it could contain objects like Mercedes, BMW, Toyota, and so on. The price or speed of these autos could be one of its attributes (data). Driving, reversing, braking, and other techniques can be used with these vehicles.

## OBJECT-ORIENTED PROGRAMMING

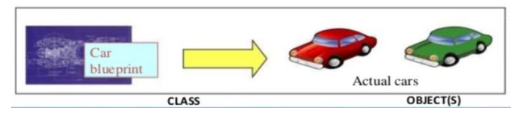


## CLASS

Class is a blueprint of objects.

Objects created by class can be similar but not the same.

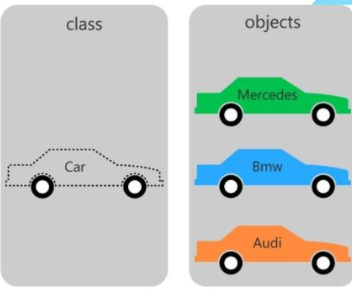
One of the fundamental notions in OOP concepts with real-time examples is the class, which is a collection of comparable entities. It is a mental component rather than a physical thing.



## OBJECT

The object is an instance of the class.

In other words, we can create multiple objects from the class.

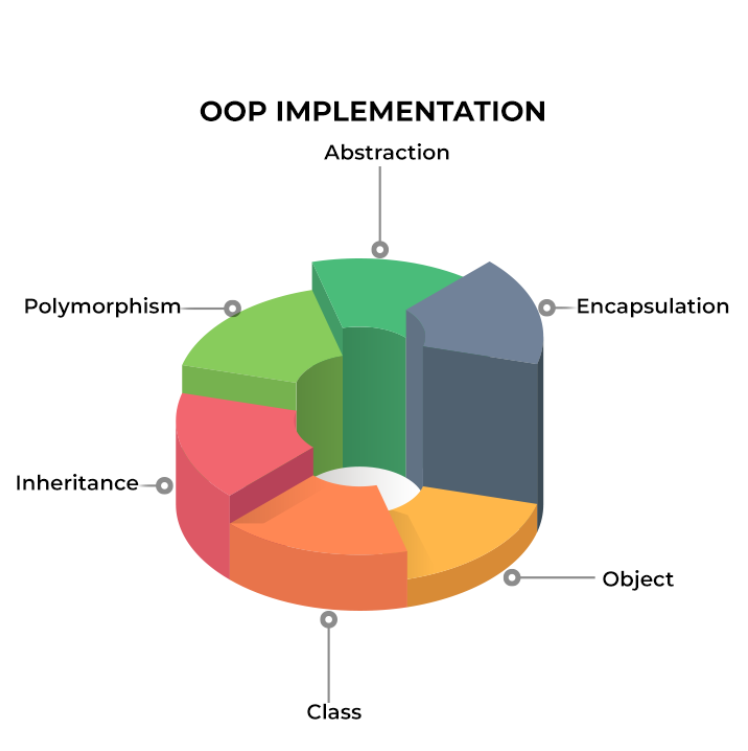


An object is a real-life entity that is defined as an instance of a class. The objects of a class called Animals, for example, will be a cat, dog, elephant, and so on. Each object has its own identity, attribute, and behavior. The code below shows how to utilize classes, objects, and methods in the Java programming language.

*#creation of class*

To create a class we need to use the ‘class’ keyword then the name of the class and ‘(‘ and ‘)’ parentheses. and at the end ‘:’.  
  
**class** employee():  
 **pass**

*#checking type on python*   
  
print(type(employee()))



We will try to cover all topics with the help of one example Employee.

**1. Encapsulation:** Encapsulation is utilized to bundle the attributes and behaviors of an employee within a class. The `Employee` class encapsulates the employee's ID, name, and salary, ensuring that these details are self-contained and accessible only through well-defined methods.

**2. Abstraction:** Abstraction is employed to hide the complex implementation details and provide a simplified interface. The `display\_employee\_details` method serves as an abstraction, allowing users to view essential information about an employee without needing to understand the internal workings of the class.

**3. Inheritance:** Inheritance is leveraged to create specialized classes that inherit attributes and behaviors from a more generalized class. For instance, if there are different types of employees (e.g., full-time and part-time), one can create subclasses inheriting from the `Employee` class to capture additional details specific to each type.

**4. Polymorphism:** Polymorphism is demonstrated through the flexibility of the `add\_employee` method, which can handle different types of employees based on the parameters provided. This method can create and return an instance of the `Employee` class, showcasing polymorphic behavior.

**5. Encapsulation:** Encapsulation is essential for protecting the integrity of the employee data. By encapsulating data within the `Employee` class and providing controlled access through methods, the system ensures that the internal representation of an employee remains consistent and secure, preventing unintended external modifications.

**ATTRIBUTES**

* Here we created a blueprint for the employee by creating a class.
* Now we will create employees as objects with some variables related to them called **attributes**

*# creating objects*   
  
  
emp1 = employee()  
emp2 = employee()

* We can add attributes about the employee in two ways

*#WAY 1: Attributes addition for employees individually*  
emp1.first = 'test1'  
emp1.last = 'user'  
emp1.email = 'test1.user@companymail.com'  
emp1.pay = 30000  
  
emp2.first = 'test2'  
emp2.last = 'user'  
emp2.email = 'test2.user@companymail.com'  
emp2.pay = 30000  
  
*#checking of information*  
print(emp1.email)  
print(emp2.email)

* if we add attributes like this it is very tedious and more or less similar to the solution to the bank problem using functions
* so instead we declare the attributes while creating the object instance itself
* we can accept these attributes being passed in using the special **init** method
* Note: we will discuss the self keyword in some time

*#WAY 2: attributes added in class*  
**class** employee():  
   
 **def** \_\_init\_\_(self,first,last,pay):  
 self.first = first  
 self.last = last  
 self.pay = pay  
   
emp1 =employee("user1","-test1",10000)  
emp2 =employee("user2","-test2",20000)  
emp3 =employee("user3","-test3",15000)

**Initialization Method (init):**

**The \_\_init\_\_ method is called every time a new instance of a class is created It is also called a special method**

* every time we create an instance it means we are creating an object of that class
* method is similar to a function but inside a class
* special methods are inbuilt functions that have a special meaning like the \_\_init\_\_ methods which get executed at the time of the creation of a new object

**ABOUT ACCESSING ATTRIBUTES:**

* Accessing an attribute just like other built-in objects in Python
* we can write the object name followed by a . (dot) then press the tab key
* when accessing attributes we don't need () brackets after the attribute name
* when accessing the method we need to add () brackets along with the parameters if any
* it is not compulsory to keep the name of the attribute the same as the one being passed in

**self - keyword**

* The self keyword tells the class that the attributes and the methods belong to that particular object
* in other words, we are passing the object itself to the methods
* whenever we call a method or a special method the self keyword is passed in by default and is hidden
* This can be verified by making a definition of the init method with no self

**Functions(Methods)**

A method in a class is a function that is associated with that class. Methods are defined within a class and operate on the class's attributes or perform actions related to the class. Methods are essentially functions that are bound to an instance of the class or the class itself. Methods in Python classes are called on instances of the class and can access and modify the instance's attributes. The self parameter in the method definition refers to the instance on which the method is called, allowing it to interact with the instance's data.

we can add more functions to perform multiple tasks.

*# Add different methods to make a full name, post, and email ID*   
*# Here you can pass the company name as an argument to make an email ID*  
  
**class** employee():  
 **def** \_\_init\_\_(self,first,last,pay):  
 self.first\_name = first  
 self.last\_name = last  
 self.salary = pay  
   
   
   
 **def** full\_name(self):  
 **return** self.first\_name+' '+self.last\_name  
   
   
 **def** post(self):  
 self.post = "manager"  
   
   
 **def** email(self,company):  
 self.company = company  
   
 self.email = self.first\_name+ "."+self.last\_name+ "@"+self.company +".com"  
   
 print(self.email)

emp1 = employee('test1', 'cat',3000)  
emp2 =employee('test2', 'dog',1000)

emp2.email("xyz") *# method 1 of calling functions : object\_name. function\_name()*

## Oops Job Opportunities:

Talented software developers with experience using C# and Java are in high demand right now. Employers prefer OOP programmers who have additional credentials, such as Certified Information Security Manager (CISM) or AWS-Certified Cloud Practitioner



The following industries have the most need for OOP developers:

* Services in the financial sector
* Healthcare
* technologically sophisticated
* Services provided by professionals
* Investing in real estate
* E-commerce and retail

## Activity Sheet

Q1. Create a class employee and add the "city" attribute to the class.

Q2. Create a class employee and create a function(method) to add new employees to the company.