Object Oriented Programming in Python

Created By :-
The EasyLearn Academy
9662512857

What Is Object-Oriented Programming?

- Object-oriented Programming, or *OOP* for short, is a programming paradigm(technique) which provides a means of structuring programs(organizeing code) so that properties (variables) and behaviors (methods/functions) are bundled (grouped together) into individual *objects*.
- For instance, an object could represent a person with a name property, age, address, etc., with behaviors like walking, talking, breathing, and running. Or an email with properties like recipient list, subject, body, etc., and behaviors like adding attachments and sending.
- Put another way, object-oriented programming is an approach for modeling concrete, real-world things like cars as well as relations between things like companies and employees, students and teachers, etc.
- OOP models real-world entities as software objects, which have some data associated with them and can perform certain functions.
- Objects are at the center of the object-oriented programming paradigm, it represent both the data (variables) and methods which works upon variable.

What is class

- The class is at the heart of any OOP language.
- class is collection of **variable**, and **methods** and **constants**.
- It's important to note that a class just provides structure, it's a blueprint for how something should be defined, but it doesn't actually provide any real content itself.
- For Example the Animal() class may specify that the name and age are necessary for defining an animal, but it will not actually state what a specific animal's name or age is.
- Class is used to define new data type which can be later used to create variable (object) of class type.
- Once a class is create, one can create any number of object of class type.
- So class is an template for object.

How to create a class?

- Syntax

```
class <ClassName>:
    #method  1
    def <MethodName1>(self):
          #method code
    # method 2
    def <MethodName2>(self):

          #method code
```

- Example

```
# A simple example class
class Test:

    # A sample method
    def fun(self):
          print("Hello")
```

The self

- Class methods must have an **extra first parameter** in method definition.
- **We do not give a value for this parameter when we call the method**, Python provides it
- If we have a method which takes no arguments, then we still have to have one argument – the self.
- See fun() in our first example.
- This is similar to this pointer in C++ and this reference in Java.
- When we call a method of object let say a person.detail('Ankit',35) where person is object of MyPerson. type
- It is automatically converted by Python into MyPerson.detail(person,'ankit',35).

What is Objects?

- **Class type variables are known as object.**
- One can only create object only after class is created.
- Once a object is created one can call methods of class.
- One can create any number of object of class type.
- All the variable of class occupy separate memory location for each of the object created of the same class.

Example of class & Object

```
#class definition (creating class) / class body
class Person:
    # Sample Method
    def say_hi(self):
        print('Hello, I am Person”)
#creating variable of class type (object)
#syntax
#object_name = class_name()
p1 = Person() #p1 is variable of person class (person class object)
#object.method() this is how you call method of the class
p1.say_hi()
```

What is constructor?

- Constructor is special member function of class.
- Constructor is used to allocate memory for the instance variables (object) of class or to do specific operation when object is created.
- It is called automatically, when object of class is created.
- Constructor can not return.
- **Name of the constructor method must be __init__**

Class and Instance Variables (Or attributes)

- Instance variables are owned by instances of the class. This means that for each object or instance of a class, the instance variables are different.
- It is created inside a constructor or method using self keyword.

Class Variables

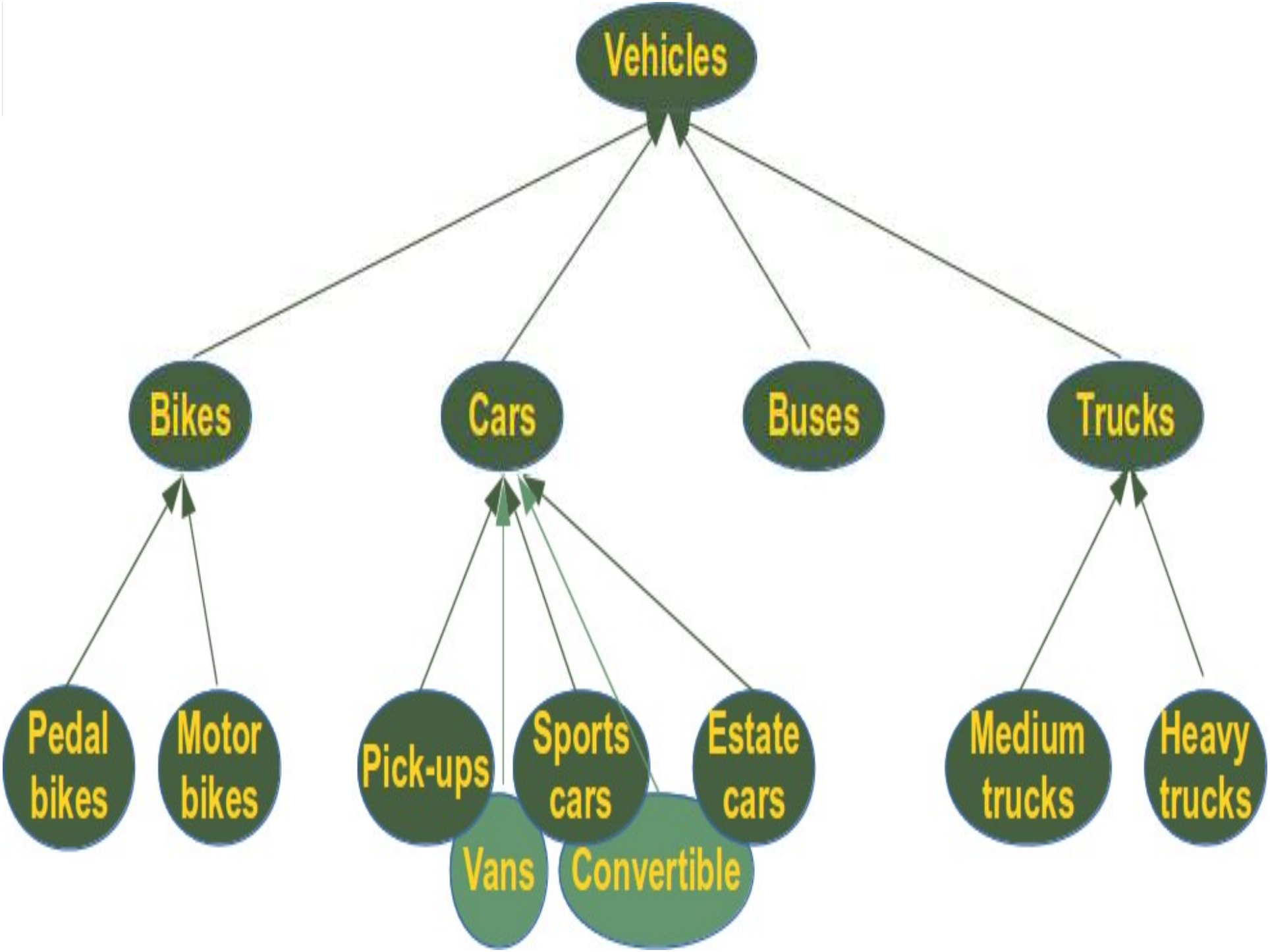
- Class variables are defined within the class construction.
- Class variables are owned by the class itself.
- Class variables are shared by all instances of the class.
- They **generally** have the same value for every instance **unless** you are using the object to initialize or change class variable.
- **It is typically placed inside the class but outside class methods.**

example

```
# Class for Computer Science Student
class Student:
    # Class Variable /shared variables
    stream = 'Computer Science'
    # The init method or constructor
    def __init__(self, roll):
          # Instance Variable
          self.roll = roll

# Objects of Student class
a = Student(101)
b = Student(102)

print(a.stream) # prints "Computer Science "
print(b.stream) # prints "Computer Science"
print(a.roll) # prints 101
print(b.roll) # prints 102

# Class variables can be accessed using class
# name also
print(Student.stream) # prints " 'Computer Science "
Student.stream = "IT"
print(Student.stream) # prints “IT"
```

What is Inheritance?

- Inheritance is a powerful feature in object oriented programming.
- It refers to defining a new class with little or no modification to an existing class.
- The new class is called **derived (or child) class** and the one from which it inherits is called the **base (or parent) class**.
- Derived class inherits features from the base class, adding new features to it.
- This results into re-usability of code.


Vehicles
Bikes
Cars
Buses
Trucks
Pedal bikes
Motor bikes
Pick-ups
Sports cars
Estate cars
Vans
Convertible
Medium trucks
Heavy trucks

syntax

```
class BaseClass
  Body of base class
class DerivedClass(BaseClass):
  Body of derived class
```

example

```
class Person:
    # Constructor
    def __init__(self, name):
            self.name = name
    def getName(self):
            return self.name
    def isEmployee(self):
            return False
class Employee(Person):
    def isEmployee(self):
            return True
emp = Person("Ankit Patel") # An Object of Person
print(emp.getName(), emp.isEmployee())
emp = Employee("Jiya Patel") # An Object of Employee
print(emp.getName(), emp.isEmployee())
```

More about inheritance

- **What is object class?**
  Like Java Object class, in Python (from version 3.x), object is root of all classes.
- In Python 3.x, "class Test(object)" and "class Test" are same.
- **Subclassing (Calling constructor of parent class)**
  A child class needs to identify which class is its parent class.
- This can be done by mentioning the parent class name in the definition of the child class.
  Eg: class **subclass_name (superclass_name)**:

Example of Inheritance with constructor

```
class Person( object ):
        # __init__ is known as the constructor
        def __init__(self, name, idnumber):
                self.name = name
                self.idnumber = idnumber
                def display(self):
                print(self.name)
                print(self.idnumber)
class Employee( Person ):
        def __init__(self, name, idnumber, salary, post):
                self.salary = salary
                self.post = post
                # invoking the __init__ of the parent class
                Person.__init__(self, name, idnumber)
a = Person('jiya', 1234)
# calling a function of the class Person using its instance
a.display()
```

- **If you forget to invoke the __init__() of the parent class then its instance variables would not be available to the child class.**

Different forms of Inheritance:

- **1. Single inheritance**: When a child class inherits from only one parent class, it is called as single inheritance.
- **2. Multiple inheritance**: When a child class inherits from multiple parent classes, it is called as multiple inheritance. Like C++, Python supports multiple inheritance. We specify all parent classes as comma separated list in bracket.
- **3 Multilevel inheritance**: When we create a new class from already derived class, it is called multilevel inheritance
- **4. Hierarchical inheritance** When a single class is inherited into more then one class, it is called Hierarchical inheritance.
- **5. Hybrid inheritance**: This form combines more than one form of inheritance.

example

```
class Base1(object):
    def __init__(self):
        self.str1 = "hello"
        print "Base1 class "
class Base2(object):
    def __init__(self):
        self.str2 = "How are you"
        print "Base2 class “
class Derived(Base1, Base2):
    def __init__(self):
        # Calling constructors of Base1
        # and Base2 classes
        Base1.__init__(self)
        Base2.__init__(self)
        print "Derived“
    def printStrs(self):
        print(self.str1, self.str2)
ob = Derived()
ob.printStrs()
```

Example of multilevel inheritance

```
class Base(object):
    def __init__(self, name):
        self.name = name
    def getName(self):
        return self.name
class Child(Base):
    def __init__(self, name, age):
        Base.__init__(self, name)
        self.age = age
    def getAge(self):
        return self.age
class GrandChild(Child):
    def __init__(self, name, age, address):
        Child.__init__(self, name, age)
        self.address = address
    def getAddress(self):
        return self.address
g = GrandChild("jiya", 07, "Bhavnagar")
print(g.getName(), g.getAge(), g.getAddress())
```

Private members of parent class

- Private instance variables means variables that can be only accessed and changed by method of the same class.
- In python we try to change value of private variable outside class (means any method which is not part the class in which variable exists), then it wont be accessible or changed.
- Private variable is also not available to use in derived class.
- In python we can create private variables by adding __ before variablename. For example we want to create private variable balance in account class then it would like in next example

Example of private member

```
class account:
    def __init__(self,name,acctype,balance):
        self.name = name
        self.acctype = acctype;
        #let us create private instance variable
        self.__balance = balance
A1 = account(”ankit patel”,”current”,1000)
A1.balance = 123456789
A1.__balance = 123456789
```

Method Overloading

- Method overloading means class/python program has multiple method with same name but each method has different number of argument.
- But python does not supports method overloading.
- **We may overload the methods but can only use the latest defined method.**
- Calling the other method will produce an error.
- But we can use trick, in Python we can define a method in such a way that there are multiple ways to call it.
- Given a single method or function, we can specify the number of parameters ourself.
- Depending on the function definition, it can be called with zero, one, two or more parameters.
- **This can be one kind of method overloading.**
- **Let us see an example.**

Example of function overloading

```
class Human:
    def sayHello(self, name=None):
        if name is not None:
            print('Hello ' + name)
        else:
            print('Hello ')
obj = Human()
# Call the method
obj.sayHello()
# Call the method with a parameter
obj.sayHello('The EasyLearn Academy')
```

2nd example

```
class Compute:
    def area(self, x = None, y = None):
        if x != None and y != None:
            return x * y
        elif x != None:
            return x * x
        else:
            return 0
obj = Compute()
# zero argument
print("Area:", obj.area())
# one argument
print("Area:", obj.area(2))
# two argument
print("Area:", obj.area(4, 5))
```

Data hidding

- We know that variable of class can be declared private by adding __ before variable name.
- Such variable are private and can be accessed only by methods of the class in which it is declared.
- Let us see an example

Example of data hiding ….

```
class MyClass:
  # Hidden member of MyClass
  __hiddenVariable = 0
  def add(self, increment):
      self.__hiddenVariable += increment
      print (self.__hiddenVariable)
myObject = MyClass()
myObject.add(2)
myObject.add(5)
# This line causes error
print (myObject.__hiddenVariable)
```

Data hiding

- in the previous program, we tried to access hidden variable outside the class using object and it threw an exception.
- We can however access private variable using tricky syntax

```
class MyClass:
   # Hidden member of MyClass
   __hiddenVariable = 10
# Driver code
myObject = MyClass()
print(myObject._MyClass__hiddenVariable)
```