Object Oriented Programming in Python

Created By :-

The EasyLearn Academy

9662512857

Procedure-Oriented Programming…

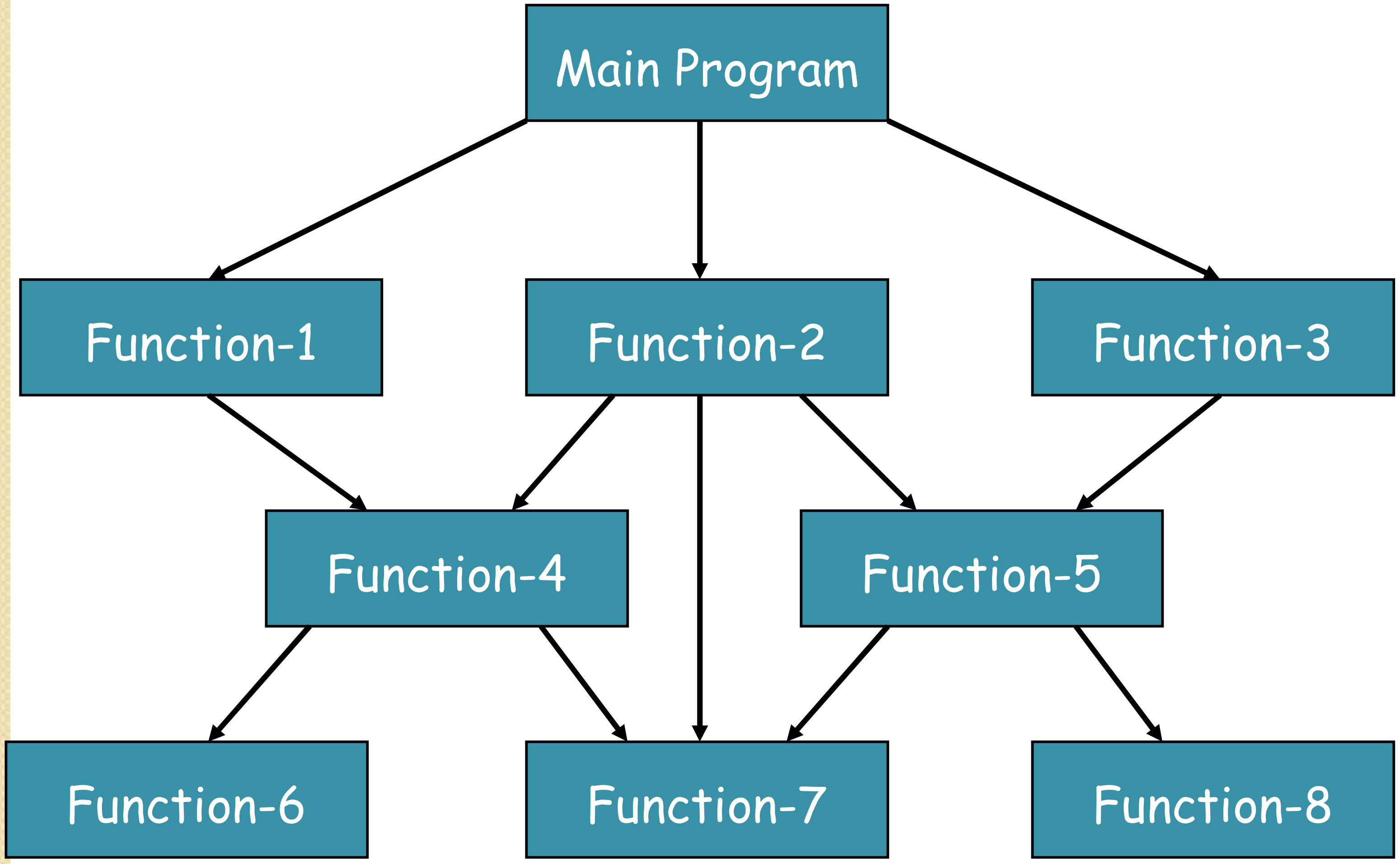
❖ Conventional programming such as high level language such C, is commonly known as procedure-oriented programming(P.O.P.).

❖ in P.O.P. we write code into blocks known as functions.

❖P.O.P. has lots of functions (with or without arguments) that can be called from any functions to perform specific task.

❖Each function may have one or more arguments that we have to pass when calling functions.

❖ therefore we can say P.O.P. code is divided into functions.

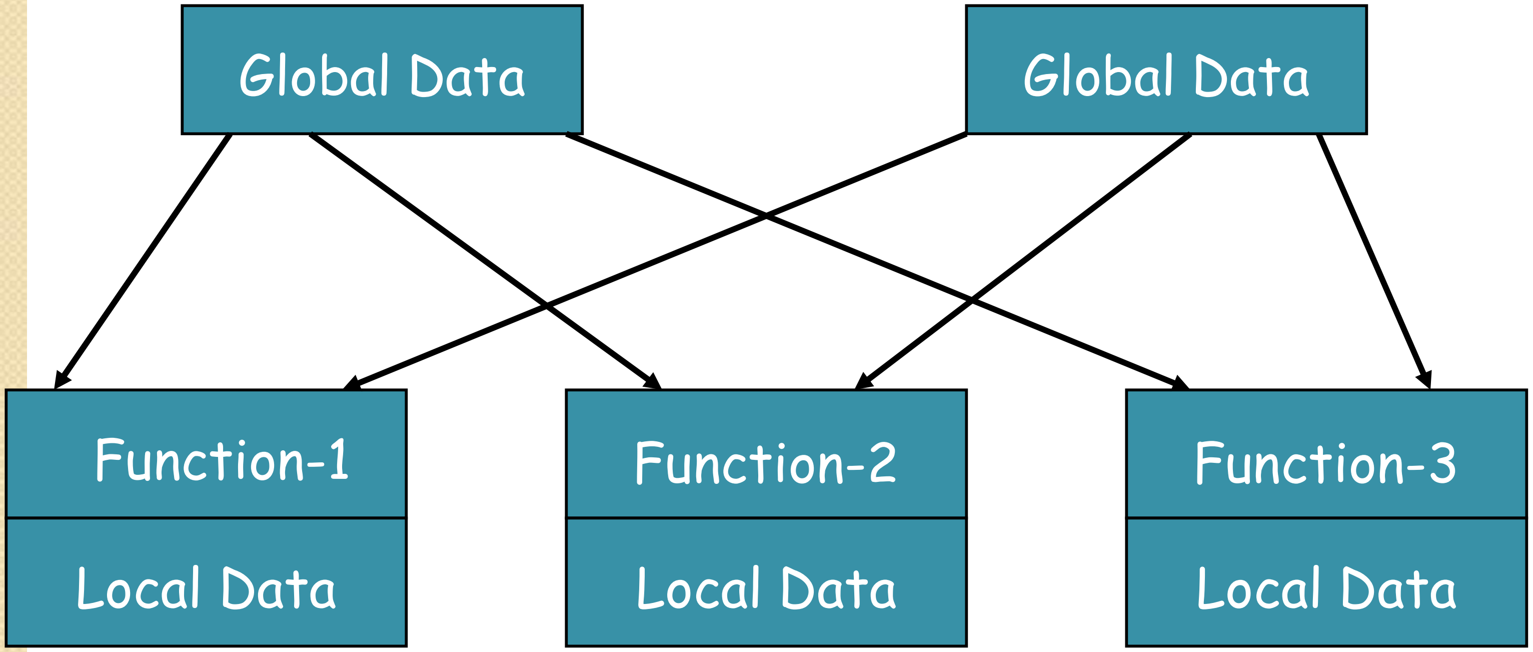


Typical Structure of Procedure-Oriented Programming

Procedure-Oriented Programming

- ❖ In big project, there can be thousands of functions in program which has to be called to multiple times.
- ❖ it is very difficult for developer to remember each and every function it’s argument and sequence of argument.

Procedure-Oriented Programming



Relationship of data and functions in procedural programming

Procedure-Oriented Programming

❖ In a multi-function program, many important data items are placed as global so that they may be accessed by all the functions. Each function may have its own local data.

Characteristics of P.O.P.

- ❖ Emphasis is on doing things(algorithms)
- ❖ Large programs are divided are into smaller programs known as **functions**.
- ❖ Most of the functions share **global data**.
- ❖ Data move openly around the system from function to function.
- ❖ Functions transform data from one form to another.
- ❖ it use top-down approach in program design.

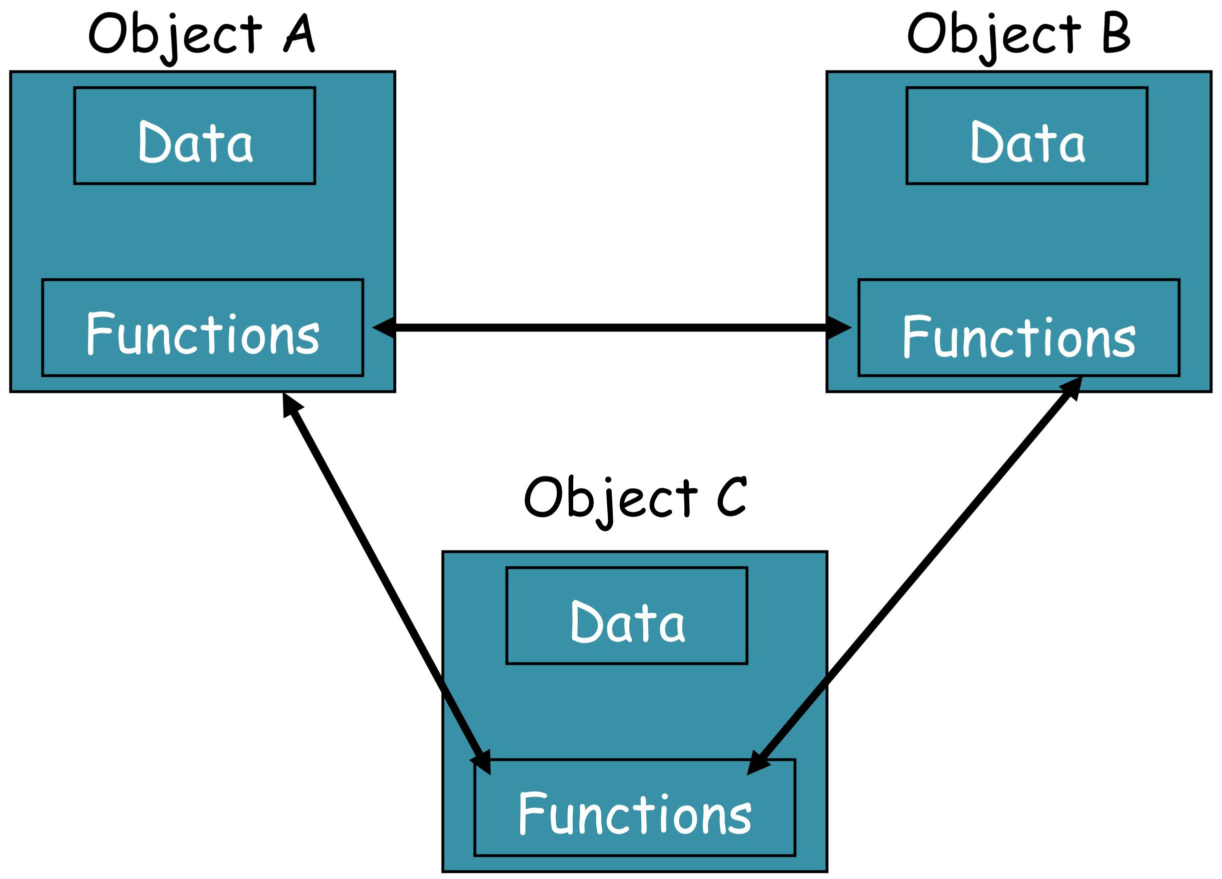
Characteristics of O.O.P.

- ❖ full form of OOP is object oriented programming.
- ❖in O.O.P. more focus is on data rather than functions.
- ❖ Programs are divided into objects.
- ❖Object is made of two elements, variables and methods.
- ❖ Functions that operate on the data of an object are tied together in the data structure known as object.

Characteristics of OOP

- ❖ variable in object can not be directly accessed or changed by external functions accidently.
- ❖ Objects may communicate with each other through by each other’s functions.
- ❖ New variable and functions can be easily added whenever necessary.
- ❖ it is called bottom-up approach in program design.

Object-Oriented Programming



Organization of data and functions in OOP

What is class

- The class is at the heart of OOP language.
- class is collection of **variable**, and **methods**.
- First we create class which has variables and methods. And then we use class to create objects.
- Once class is created we can create many object of the class type.
- So class is an template to create 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 Animal:
    # A sample method
    def eat(self):
           print("I can eat")
    def run(self):
           print("I can run")
    def sleep(self):
           print("I can sleep")
#create object
#object = classname()
Tiger = Animal() #creating object of animal class
Tiger.eat()
Tiger.run()
Tiger.sleep()
```

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 eat() in our first example.

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 type (object) occupy separate memory location.

Instance variables

- Variables declared inside function of class using self keyword are known as instance variable.
- Each and every instance variable has different value & memory space for each object.
- We can access/change/remove instance variables from the method of the same class.
- For this we don’t have to pass instance variables as arguments inside methods of the same class.

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 **never** return value.
- **Name of the constructor method must be __init__**
- You can not give any other name to constructor.
- There can be only one constructor in class.

Class Variables

- Class variables are defined within the class body but outside the class methods.
- Class variables are owned by the class itself.
- It can be accessed by all instances(object) of the class.
- They **generally** have the same value for every instance(object) **unless** you are use the object to initialize or change class variable.

example

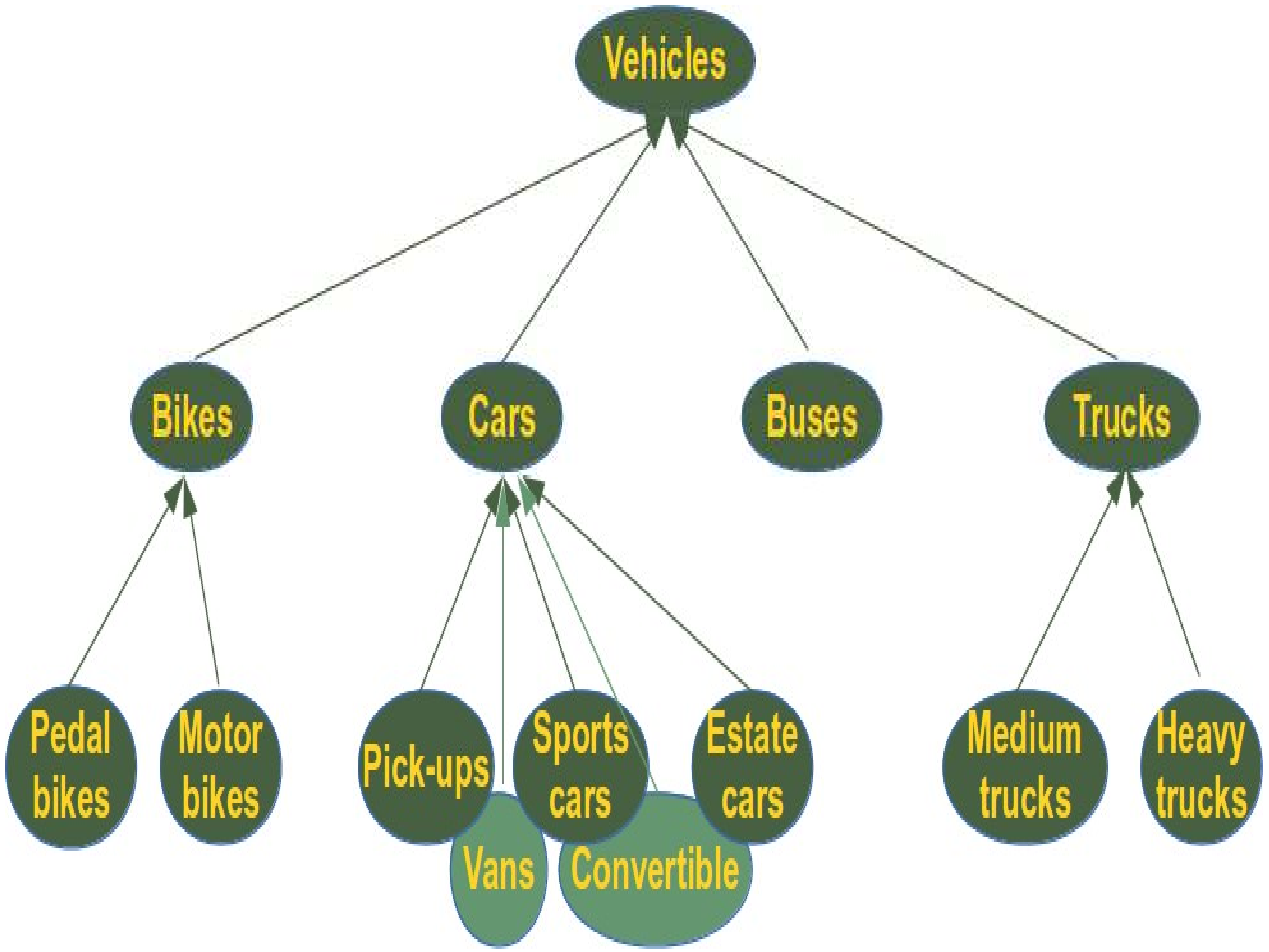
```
# Class for Computer Science Student
class Student:
   # Class Variable /shared variables
   InstituteName= “the easylearn academy”
   # The init method/constructor
   def __init__(self, rollno,name):
         self.rollno = rollno # Instance Variable
         self.name = name #Instance variable
# Objects of Student class
a = Student(101,”Ankit”)
b = Student(102,”Jiya”)

print(a.InstituteName) # prints " the easylearn academy "
print(b.InstituteName) # prints " the easylearn academy "
print(a.roll,a.name) # prints 101 Ankit
print(b.roll,b.name) # prints 102 Jiya

# Class variables can be accessed using class
# name also
print(Student.InstituteName) # prints "the easylearn academy"
Student.InstituteName = “T.E.L"
print(Student.InstituteName) # prints “T.E.L"
```

What is Inheritance?

- Inheritance is a powerful feature in object oriented programming.
- It refers to defining a new class with little 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.
- It means that derived class can use instance variables and methods of the parent class as if these variables and methods are their own.
- 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:
  variables……
  methods……
class DerivedClass(BaseClass):
  parent class variable….
  parent class methods….
  derived class variable…..
  derived class methods…..
```

example

```
class Person: #parent/super/base class
    def walk(self):
        print("I can walk")
    def talk(self):
        print("I can talk")
    def eat(self):
        print("I can eat")
class Student(Person): #child/sub/dervied class
    def read(self):
        print("I can read")
    def write(self):
        print("I can write")
    def WhatICanDo(self):
        #calling parent class method
        super().walk();
        super().talk();
        super().eat();
        self.read();
        self.write();
#creating object of parent class
p1 = Person()
p1.walk()
p1.talk()
p1.eat()
    #creating object of child class
s1 = Student()
s1.WhatICanDo()
s1.walk()
s1.read()
```

Example of Inheritance with constructor

```
class Person():
    # __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):
        super().__init__(name, idnumber) # invoking the __init__ of the parent class
        self.salary = salary
        self.post = post
    def display(self): #method overriding
        super().display() #calling parent class method display
        print(self.salary)
        print(self.post)
p = Person('jiya', 1234)  # parent class object
p.display()  # calling a function of the class Person using its instance
e = Employee("Rahul", 101, 125000, 'Developer')  # child class object
e.display()
```

Different forms of Inheritance:

- **1. Single inheritance**: when we create one new class from only one existing class, it is called single level inheritance.
- **2. Multiple inheritance**: when we create one new class using **more then one existing class**, it is called multiple inheritance.
- **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 derived into more then one class, it is called Hierarchical inheritance. It means one class is parent class of more then one class
- **5. Hybrid inheritance**: This form combines more than one form of inheritance.

example

```
class Base1:
    def __init__(self):  #constructor
            self.name = “apple“ #instance variable
            print  "Base1 class constructor called….."
class Base2:
    def __init__(self): #constructor
            self.surname = “fruit"
            print "Base2 class constructor called…“
class Derived(Base1, Base2):  #multiple inheritance.
    def __init__(self): #constructor
            # Calling constructors of Base1
            # and Base2 classes
            Base1.__init__(self) #calling parent class
            Base2.__init__(self) #calling parent class
            print "Derived class constructor called“
    def printStrs(self):
            print(self.name, self.surname)
d1 = Derived()
d1.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 #public variable
        self.acctype = acctype; #public variable
        #let us create private instance variable
        self.__balance = balance #private variable
ACC = account(”ankit patel”,”current”,1000)
ACC.name = “Jiya Patel” #will work
#ACC.__balance = 123456789  #will be ignored
```

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)
```

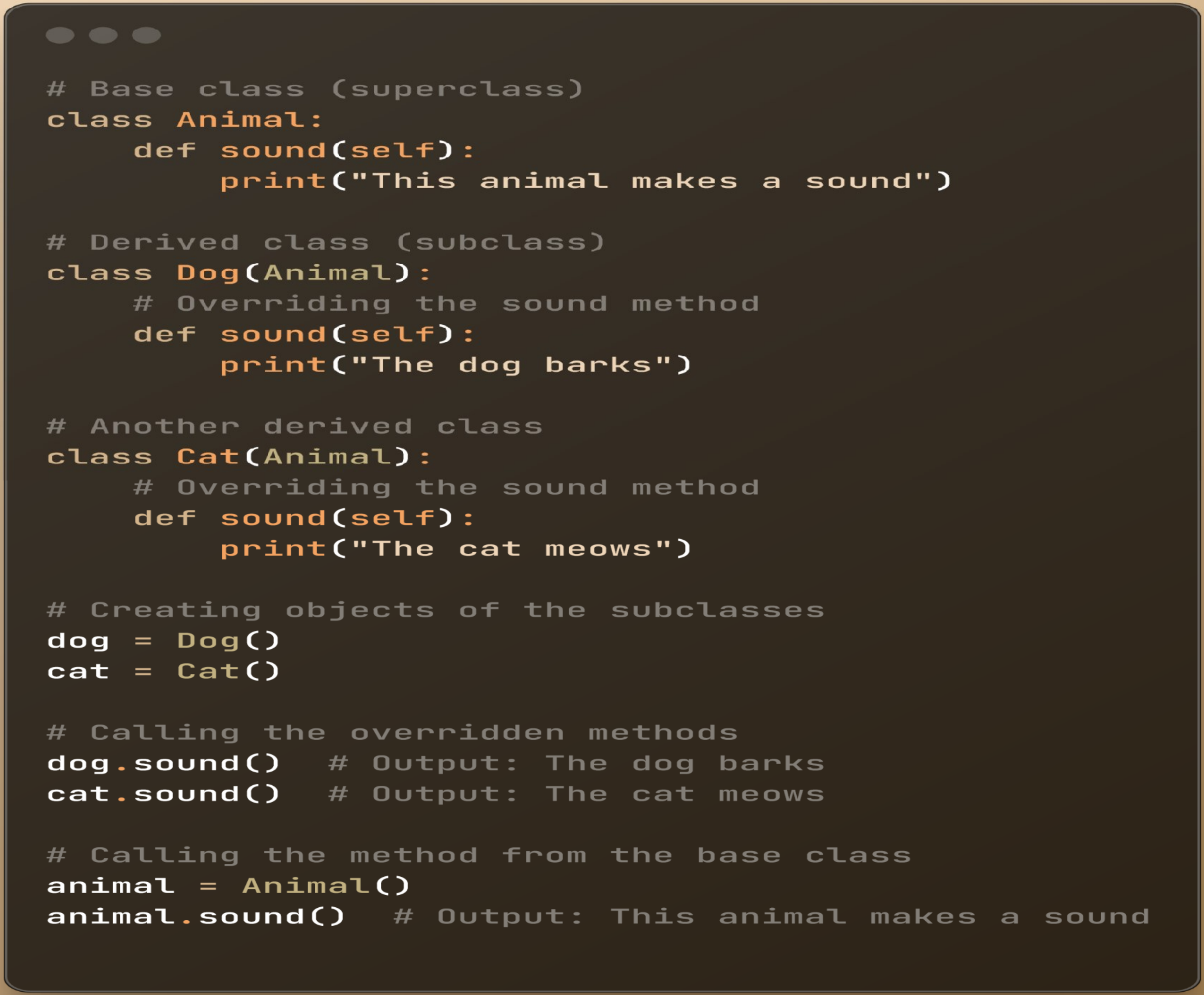
Data hiding

- 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)
```

Method Overriding

- When we create two same method, one in parent class and one in child class, it is called method overriding.
- Both method must have same name, same no of arguments.
- Now if use child class object and call such method (Overrided) then it will call child class method. And if we use parent class object then it will call parent class method.



```python
# Base class (superclass)
class Animal:
    def sound(self):
        print("This animal makes a sound")

# Derived class (subclass)
class Dog(Animal):
    # Overriding the sound method
    def sound(self):
        print("The dog barks")

# Another derived class
class Cat(Animal):
    # Overriding the sound method
    def sound(self):
        print("The cat meows")

# Creating objects of the subclasses
dog = Dog()
cat = Cat()

# Calling the overridden methods
dog.sound()  # Output: The dog barks
cat.sound()  # Output: The cat meows

# Calling the method from the base class
animal = Animal()
animal.sound()  # Output: This animal makes a sound
```