Dr. D. Y. PATIL INSTITUTE OF TECHNOLOGY, PIMPRI, PUNE-18

Department of First Year Engineering

Programming and Problem Solving

Unit- 5 Notes

Unit V: Object Oriented Programming

5.1 Programming Paradigms

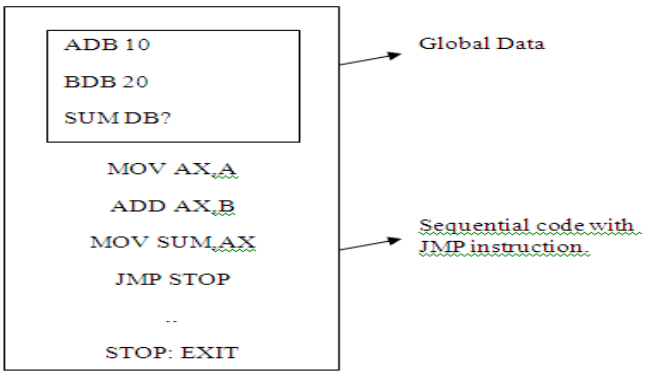
Q1. Write a note on:

1. **Monolithic Programming**
2. **Procedural Programming**
3. **Structured Programming**
4. **Object Oriented Programming**

Ans.

(i) Monolithic Programming Paradigm:

- Programs written using monolithic programming languages such as assembly language consist of global data and sequential code.
- The global data can be accessed and modified from any part of the program.
- A sequential code is one in which all instructions are executed in the specified sequence.
- In order to change the sequence of instructions, jump statements or 'goto' statements are used.
- Structure of a monolithic program:



Advantages:

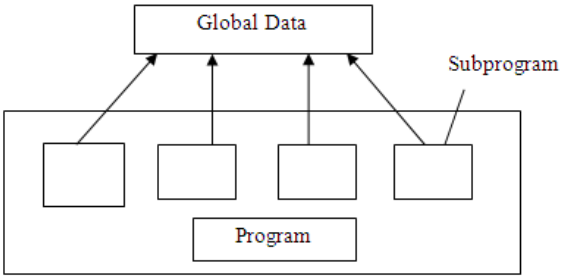
- Monolithic programming language is used only for very small and simple applications.
- Monolithic programming language is used only for applications where reusability is not a major concern.

Disadvantages:

- Monolithic programs have just one program module as monolithic programming languages do not support the concept of subroutines.
- As it is containing just one program module, all the actions required completing particular task are embedded within same application itself. This makes the size of program large.
- It is difficult to debug and maintain monolithic programs.

(ii) Procedural Programming Paradigm:

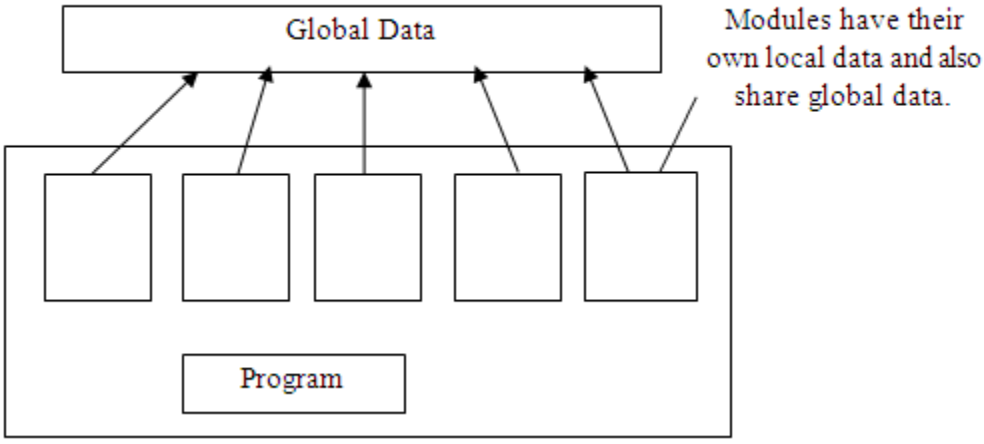
- In procedural programming language, a program is divided into n number of subroutines that access global data.
- To avoid repetition of code, each subroutine performs a well-defined task.
- A subroutine that needs the service provided by another subroutine can call that subroutine.
- Therefore, with 'jump','goto', and 'call' instructions, the sequence of execution of instructions can be altered.
- For example, FORTRAN and COBOL are two popular procedural programming languages.
- Structure of a procedural program:



- Advantages:
  - This paradigm is used to write just correct programs.
  - Programs using this paradigm are easier to write as compared to monolithic programming.
- Disadvantages:
  - Writing programs is complex.
  - No concept of reusability.
  - It requires more time and effort to write programs.
  - Programs are difficult to maintain.
  - Global data is shared and therefore may get altered (mistakenly).
  -

(iii) Structured Programming Paradigm:

- Structured programming is also known as modular programming.
- By using structured programming paradigm, programs are designed to implement a logical structure on the program.
- Due to this, program becomes more efficient and easier to understand.
- Structured programming is used in large programs that require a large development team to develop different parts of the same program.
- Structured programming employs a top-down approach in which the overall program structure is broken down into separate modules.
- Due to this top-down approach, the code is loaded into memory more efficiently and also can be reused in other programs.
- Modules are coded separately and once a module is written and tested individually, it is then integrated with other modules to form the overall program structure.
- Structured programming is based on modularization. Modularization groups related statements together into modules.
- Modularization makes programming easier to write, debug and understand.

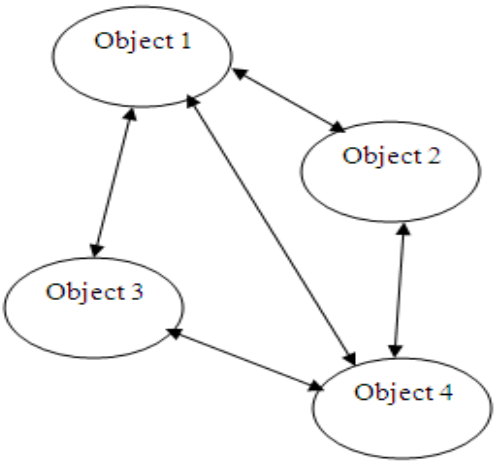


- Advantages:
  - Structured programming is used to write correct programs that are easy to understand and change.
  - Structured programming allows users to look at the big picture first and then focus on details later by using modules.
  - With modules, many programmers can work on a single, large program, with each working on a different module.
  - A structured program takes less time to be written than other programs. Also modules or procedures written for one program can be reused in other programs as well.
  - Each module performs a specific task.
  - Each module has its own local data.
  - A structured program is easy to change as well as understand because every procedure has meaningful names and has clear documentation to identify the task performed by it.
  - A structured program is easy to debug because each procedure is specialized to perform just one task and every procedure can be checked individually for the presence of any error.
  - Structured programming gives more emphasis on code and the least importance is given to the data.

- Disadvantages:
  - Structured programming is not data-centered.
  - Global data is shared and may get modified.
  - In structured programming, main focus is on functions.

(iv) Object-oriented Programming (OOP) Paradigm:

- Object-oriented programming paradigm treats data as a critical element in the program development.
- In this paradigm, all relevant data and tasks are grouped together in entities known as objects.
- Object-oriented programming is **task based** as it considers operations.
- Object-oriented programming is **data-based** as these operations are grouped with relevant data in objects.
- Every object contains some data and the operations, methods, or functions that operate on that data.
- Programs that need the object will access the object's methods through a specific interface. The interface specifies how to send a message to the object, that is, a request for a certain operation to be performed.



Objects of a program interact by sending messages to each other

Object oriented paradigm

- Advantages:
  - OOP is used for simulating real world problems on computers because the real world is made up of objects.
  - Programs written using OOP are data centered.
  - Programs are divided in terms of objects and not procedures.
  - Functions that operate on data are tied together with the data.
  - Data is hidden and not accessible by external functions.
  - New data and functions can be easily added as and when required.
  - It follows a bottom-up approach for problem solving.
- Disadvantages:
  - It requires more data protection.
  - Inability to work with existing systems.
  - Larger program size.
  - Not suitable for all types of problems-for smaller problems it is in general not suitable.

Q2.Differentiate between Procedural and Object oriented programming.

Ans.

Index	Object-oriented Programming	Procedural Programming
1.	Object-oriented programming is the problem-solving approach and used where computation is done by using objects.	Procedural programming uses a list of instructions to do computation step by step.
2.	It makes the development and maintenance easier.	In procedural programming, It is not easy to maintain the codes when the project becomes lengthy.

3.	It simulates the real world entity. So real-world problems can be easily solved through oops.	It doesn't simulate the real world. It works on step by step instructions divided into small parts called functions.
4.	It provides data hiding. So it is more secure than procedural languages. You cannot access private data from anywhere.	Procedural language doesn't provide any proper way for data binding, so it is less secure.
5.	Example of object-oriented programming languages is C++, Java, .Net, Python, C#, etc.	Example of procedural languages are: C, Fortran, Pascal, VB etc.

5.2 Features of Object oriented programming

Q3. Explain features of OOP.

Or

Q3. Write a note on: 1.___ 2.____ 3.____

1. **Classes**
   - A class is user-defined data type used to describe something in the world, such as occurrences, things, external entities, and so on.
   - A class describes template or blueprint that describes the structure and behavior of a set of similar objects.
   - Once we have definition for a class, a specific instance of that class can be easily created.
   - For eg. Consider a class *student*. A *student* has **attributes** such a roll_no, name, course and aggregate. The **operations** that can be performed on its data may include 'getdata', 'setdata','editdata' and so on.
   - A class is collection of objects.

2. Objects

- Object is basic unit of object-oriented programming.
- Object is basic run-time entity in an object-oriented system.
- *Anything having its own properties can be considered as an object.*
- For example flower is an object having properties such as name, fragrance, etc.
- An object is a collection of data members and associated member function also known as methods shown in figure1 below:

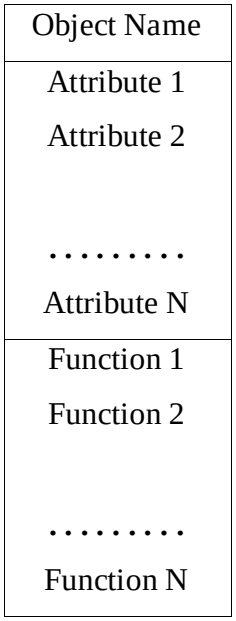


Fig1. Representation of an Object

3. Methods and Message passing

Methods

- A ***method*** is a function associated with class.
- It defines the operations that the object can execute when it receive a message.
- In object oriented language, only methods of the class can access and manipulate the data stored in an instance of the class (or object).
- Figure2 shows how a class is declared using its data members and member functions.

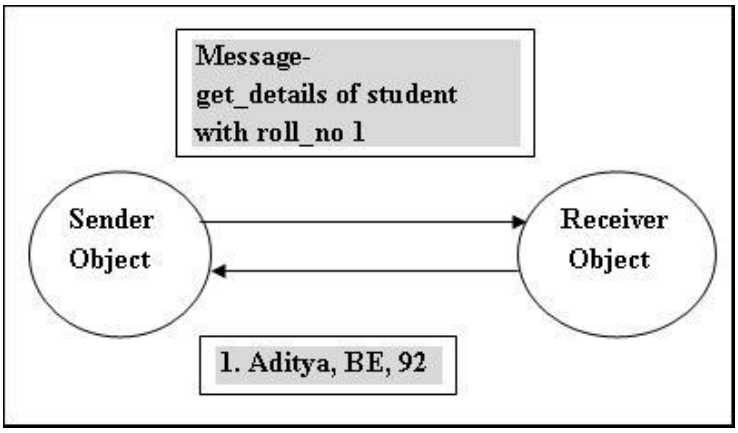


Figure2. Objects sending a message

Message Passing

- Two objects communicate with each other though messages.
- An object asks another object to involve one of its methods by sending it a message.
- In figure2 sender object is sending message to the receiver object to get details of a student. i.e. the sender is passing some specific information to the receiver so that the receiver can send the correct and precise information to the sender. The data that is transferred with the message is called *parameters.* Here, *roll_no* 1 is the parameter.
- The messages that are sent to other objects consist of three aspects-the receiver object, the name of the method that the receiver should invoke, and the parameters that must be used with the method.

4. Inheritance ('is-a' relation)

- *Inheritance* is a concept of OOP in which a new class is created from an existing class.
- The new class, often known as a subclass, contains the attributes and methods of the parent class. (the exiting class from which the new class is created).
- The new class, known as a **subclass or derived class**, inherits the **attributes and behaviour** of pre-existing class, which is referred to as **superclass or parent class**. (refer figure3.)

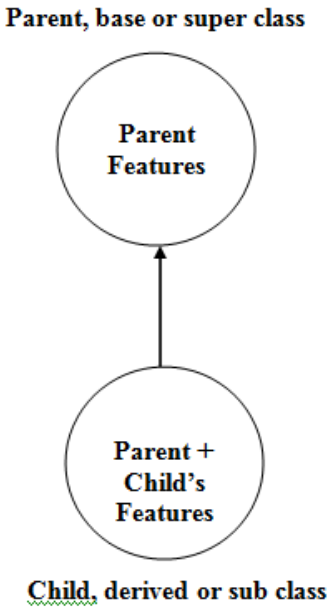


Figure3: Child, derived or subclass

- The inheritance relationship of a subclasses and superclasses generates a hierarchy. Therefore inheritance relation is also called as **'is-a' relation.**
- A subclass not only has all the states and behavior associated with superclass but has other specialized features (additional data or methods) as well.
- The main advantage is ability **to reuse the code**.
- For example, we have a class student with following members:
  Properties: rollno, name, course, marks
  Methods: getdata, setdata

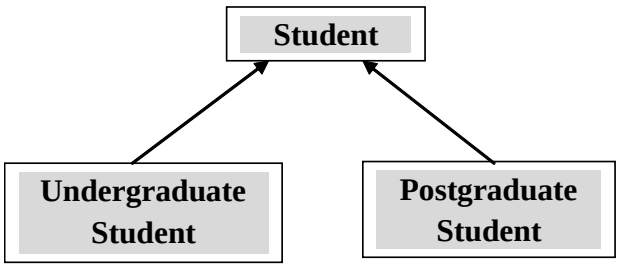


Figure4. Example of Inheritance

- We can inherit two classes from the class student- undergraduate students and postgraduate students (refer figure4). These two classes will have all the properties of class students and in addition to that it will have even more specialized members.
- When a derived class receives a message to execute a method, it finds the method in its own class. If it finds the method, then it simply executes it. If the method is not present, it searches for that method in its superclass. If the method is found, it is executed; otherwise, an error message is reported.

5. Polymorphism

- *Polymorphism* refers to having several different forms.
- While inheritance is related to classes and their hierarchy, polymorphism on other hand, is related to methods.
- Polymorphism is a concept that enables the programmers to assign a different meaning or usage to a method in different context.
- Polymorphism exist when a number of subclasses is defined which have methods of same name.
- Polymorphism can also be applied to operators.

6. Containership (composition or has-a' relationship **)**

- *Containership* is ability of a class to contain object(s) of one or more classes a member data.
- For example, class Person can have an object of class Student as its data member. This would allow the object of class Person to call the public functions of class Student. Here class Person becomes the container, whereas class Student becomes the contained class.
- Containership is also called ***composition*** because as in given example, class Person is composed of class Student.
- In OOP, containership represents a **'has-a'** relationship.

7. Reusability

- *Reusability* means developing codes that can be reused either in the same program or in the different programs.
- Reusability is attained through **inheritance, containership and polymorphism**.

8. Delegation

- To provide maximum flexibility to programmers and to allow them to generate a reusable code, object oriented languages also support delegation.
- In composition, an object can be composed of other objects and thus, the object exhibits a 'has-a' relationship.
- In delegation, more than one object is involved in handling a request. The object that receives the request for a service, delegates it to another object called its delegate.
- Delegation means that one object is dependent on another object to provide functionalities.
- The property of delegation emphasizes on the ideology than a complex object is made of several simpler objects.
- For example, our body is made up of brain, heart, hands, eyes, ears, etc. The functioning of the whole body as a system rests on the correct functioning of the parts it is composed of.
- Delegation is different from inheritance in the way that two classes that participates in inheritance share 'is-a' relationship; however, in delegation, they have a 'has-a' relationship.

9. Data Abstraction

- *Data Abstraction* refers to the process by which data and functions are defined in such a way that only essential details are revealed and the implementation details are hidden.
- The main focus of data abstraction is to separate the interface and implementation of a program.

- For example, as user of television sets, we can switch it on or off, change the channel, set the volume and add external devices such as speakers and CD or DVD players without knowing the details how its functionality has been implemented. Therefore the implementation is completely hidden from external world.

10. Encapsulation (Data hiding)

- Data encapsulation, also called data hiding, is the technique of packing data and functions into a single component (class) to hide implementation details of a class from users.
- Users are allowed to execute only a restricted set of operations (class methods) on the data members of the class.
- Therefore encapsulation organizes the data and methods into a structure that prevents data access by any function (or method) that is not specified in the class. This ensures the integrity of the data contained in the object.
- Encapsulation defines three access levels for data variables and member functions of the class. These access levels specify the access rights, explained as follows:
    - Any data of function with access level as **public** can be accessed by any function belonging to any class. This is lowest level of data protection.
    - Any data of function with access level as **protected** can be accessed only by that class or by any class that is inherited from it.
    - Any data of function with access level as **private** can be accessed only by the class in which it is declared. This is highest level of data protection.

5.3 Classes and Object

Q4. Explain class and object with suitable examples.

Ans.

- A **class** creates new data type and object is an instance (or variable) of the class.
- Classes provide a blueprint or a template using which objects are created.

- In fact, *in Python, everything is an object o an instance of some class.*
- For example, all integer variables that we define in our program are actually instances of class int. We can find out the type of any object using the **type() function.**

Defining Classes

- Python has very simple syntax of defining a class. Syntax is given below:

```
class class_name:
    <statement-1>
    <statement-2>
    .
    .
    .
    <statement-N>
```

- It starts with a keyword *class* followed by the *class_name* and *colon (:)*
- The statement in the definition can be any of these—
    - Sequential instructions,
    - Decision control statements
    - Loop statements and
    - Function definition
- Variables defined in the class are called *class variables* and functions defined inside a class are called *class method*. Class variables and class methods are together known as *class members.*
- The class members can be accessed through class objects.
- Class definitions can appear anywhere in a program, but they usually written near the beginning of the program, after import statement.
- When a class definition is entered, a new namespace is created, and used as the local scope.

Creating Objects

- Once a class is defined, the next job is to create an object (or instance) of that class. The object can then access class variables and class methods using **dot operator (.)**
- The syntax to create an object is given as:

```
Object_name=class_name()
```

- Creating an object or instance of a class is known as class instantiation. Class instantiation uses function notation.
- The syntax for accessing a class member through the class object is:

```
Object_name.class_member_name
```

- **Program to access class variable using class object**

```
class ABC:
    var=10
obj=ABC()                #Object obj is created of class ABC
print(obj.var)
OUTPUT
10
```

- In the above program, we have defined a class ABC which has var having a value of 10. The object of the class is created and used to access the class variable using dot operator.

5.4 Data Abstraction and Hiding through Classes

Q5. What is data abstraction and hiding through classes?

Ans.

Data Abstraction

- *Data Abstraction* refers to the process by which data and functions are defined in such a way that only essential details are provided to the outside world and the implementation details are hidden.
- The main focus of data abstraction is to separate the interface and implementation of a program.

Encapsulation (Data hiding)

- Data encapsulation, also called data hiding, is the technique of packing data and functions into a single component (class) to hide implementation details of a class from users.
- Users are allowed to execute only a restricted set of operations (class methods) on the data members of the class.
- Therefore encapsulation organizes the data and methods into a structure that prevents data access by any function (or method) that is not specified in the class. This ensures the integrity of the data contained in the object.
- Encapsulation defines three access levels for data variables and member functions of the class. These access levels specify the access rights, explained as follows:
    - Any data of function with access level as **public** can be accessed by any function belonging to any class. This is lowest level of data protection.
    - Any data of function with access level as **protected** can be accessed only by that class or by any class that is inherited from it.
    - Any data of function with access level as **private** can be accessed only by the class in which it is declared. This is highest level of data protection.

5.5 Class Method and Self objects

Q6. What does the self argument signify in the class methods?

Ans.

- Class Method definition is nothing but Function Definitions, exactly same as ordinary functions.
- Syntax:

```
Class Class_name:
    def function_name(self):
        Statement 1
        Statement 2
        .
```

.

Statement n

- Class method must have the first argument named as self. Moreover we do not pass any value for this parameter.
- When we call the method, python provide its value automatically. Self-argument refers to the object itself.
- This means that even that method that takes no arguments, it should be defined to accept the self.
- self is just a parameter in function and user can use any parameter name in place of it. But it is advisable to use self because it increase the readability of code.
- Example:

```
class student:                                  #Class Definition
        rollno=10                               # Class Variable
        def display(self):                      # Class Method
                print("Roll no of Student is",self.rollno)
```

```
obj=student()                                   #Object Creation
obj.display()                                   #Method call
```

Output:

Roll no of Student is 10

5.6 The __init__() Method (The class constructor)

Q7. Explain the significance of __init__( ) method.

Ans.

- Python supports a very unique method named as __init__() method. This method gets executed when instance of a class gets created (i.e.- when objects get created).

- It is also called as an initializer method or constructor in object oriented programming.
- It is useful to initialize the variables of the class objects.
- First parameter of this method is called as self, means instance/object of the class.
- This method is prefixed as well as suffixed with double underscore symbol
- Syntax:

```
Class Class_name:
    def __init__(self [,argument list]):
        Statement 1
        Statement 2
        .
        .
        Statement n
```

```
class student:                                  #class Defination
    def __init__(self, marks):                  # Class Method (Constructor)
        self.marks=marks
        print ("Marks of a Student is",self.marks)
```

```
obj1=student(70)                                #Object 1 Created
obj2=student(80)                                #Object 2 Created
```

5.7 Class variables and Object variables

Q8. Write a note on class variables and object variables with example.

Ans.

Class Variable:

- Class variables are used to declare within a class. These variables have same value across all instances (objects). These variables are also called as static variables.
- All the objects/Instance of the class will share the class variable.

- Since there exist only one copy of the class variable, any changes made to the class variable by an object will be reflected in all other objects/instances.
- Syntax to access class variable:

  className. Class_variable
- here className is the name of the class and class_variable is the class variable defined in the class
- Example1:

```
Class student:                  # Class definition
    rollno =10                  # Class Variable

print("class variable value is ", student.rollno)     #printed class variable
```

- Class variables are created and assigned values within declaration itself.
- You can also access class variables with an object created within a class
- Example 2:

```
Class student:          # Class definition
    rollno =10          # Clas Variable
    def display(self):
        print("class variable value is ", self.rollno)        #printed class variable

obj= student()
```

Object Variable:

- Object variable or instance variable have different value for each new instance.
- If a class has N instances/Objects, then there will be a N separate copies of the object variable as each object/instances have its own object variable.
- Object variables are not shared between other objects.

- A change made to the object variable by one object, will not be replicated in other objects.
- Example1:

```
class student:                                  #class Defination
        serial_No=0                             # Class Variable
        def  __init__(self, marks):             # Class Method (Constructor)
                student.serial_No+=1
                self.marks=marks
                print(("Serial number is" , self.serial_No)
                print ("Marks of a Student is",self.marks)
```

```
obj1=student(70)                                #Object 1 Created
obj2=student(80)                                #Object 2 Created
```

- Example 2:

```
class student:                                  #class Defination
        serial_No=0                             # Class Variable
        def  display(self, marks):              # Class Method
                student.serial_No+=1
                self.marks=marks
                print(("Serial number is", self.serial_No)
                print ("Marks of a Student is",self.marks)
```

```
obj1=student()                                  #Object 1 Created
obj2=student()                                  #Object 2 Created
obj1.display(70)                                #Method call
obj2.dispaly(80)                                #Method call
```

Q9. Differentiate between class variables and instance variables.

Ans: We have seen that a Class can have variables defined in it. Basically these variables are of two types:

a) **Class Variables**

b) **Instance Variables**

As the name suggests the class variables are owned by the class and Instance (Object) variables are owned by each object.

Important points are as follows:

- If a class has n objects, then there will be n separate copies of the object variable as each object will have its own object variable.
- The object variable is not shared between objects.
- A change made to the object variable by one object will not be reflected in other objects.
- If the class has one class variable, then there will be one copy only for that variable. All the objects of that class will share the class variable.
- Since there exists only one copy of class variable, any change made to the class variable will be reflected in all other objects.

Program to differentiate between object and class variables

```
class Car:
    wheels = 4                      #Class Variable
    def  __init__(self, comp):
        self.company = comp         # Object/Instance Variable
        print("The company(Object Var) : ", self.company)
        print("The Wheels(Class var) : ", Car.wheels)
obj1 = Car("Mercedez")
obj2 = Car("BMW")
```

Output:

```
The company(Object Var) :  Mercedez
The Wheels(Class var) :  4
The company(Object Var) :  BMW
The Wheels(Class var) :  4
```

In the above example, following points should be noted:

- The class Variable **wheels** is having default value 4, which is same for all types of Cars. Such variables are declared as class variable, which is common to all.
- Whereas the Company of car is depends on the particular car and it may varies for different cars. Such variables are declared as a Instance Variables.
- The Object or Instance variables are declared/ initialized inside __init__() or member function of a class
- The class variable are declared / Initialized outside of any function in class.

5.8 The __del__() method

Q10. Explain __del__() method with example.

Ans:

- We saw __init__() method which initializes object.
- Similar to __init__() method we have __del__() method which does just the opposite work.
- The __del__() method is automatically called when the object goes out of scope.
- This is the time when object will no longer used and its occupied resources are return back to system
- Returned resources can be used for some other work.
- You can explicitly delete the object by **del** keyword, so that __del__() method will get called.

```
class ABC:
    def __init__(self, var):
        self.var = var
        print("The object value is =", var)
    def __del__(self):
        print("Object with value %d is going out of scope"%self.var)

obj1 = ABC(10)          # Creating obj1, __init__() get called
obj2 = ABC(20)          #Creating obj2, __init__() get called
```

```
del obj1          # Deleting obj1 explicitly, __del__() get called
del obj2          # Deleting obj2 explicitly, __del__() get called
```

Output:

```
The object value is = 10
The object value is = 20
Object with value 10 is going out of scope
Object with value 20 is going out of scope
```

- In the above program __init__() method is automatically called when object is created.
- __del__() method is also automatically called when object is deleted.
- We are deleting object explicitly, by **del objectName.** So while object is getting deleted __del__() method is called.

5.9 Public and Private Members

Q11. Explain public and private members with suitable example.

Ans.

Ans:

- Public variables are the variables which are declared inside class and can be accessed from anywhere in the program using the dot (.) operator.
- It means public variables can be accessed from within the class as well as from outside the class in which it is defined.
- Private Variables are defined in the class with double underscore prefix (__).
- Private variables are accessible from only within the class and from nowhere outside the class.

```
class ABC:
   def __init__(self, val1, val2):
      self.var1 = val1          # var1 is public variable
      self.__var2= val2         # __var2 is private variable
```

```
    def display(self):
        print("From class method var1=", self.var1)      # var1 is accessible within class
        print("From class method var2=", self.__var2)   # __var2 is accessible within class
obj=ABC(10,20)
obj.display()
print("From main module, var1 =", obj.var1)       # var1 is public, accessible outside class
print("From main module, var2 =", obj.__var2)     #Will give Error  as __var2 is private
```

Output:

From class method var1= 10

From class method var2= 20

From main module, var1 = 10

Error Message displayed

- As a good programming habit, you should never try to access a private variable from anywhere outside the class.
- But if for some reason, you want to access the private variables outside the class, use following Syntax:

  objectName**.**_className__privatevariable
- So to remove error from above code, you shall write last statement as

  print("From main module, var2 =", obj._ABC__var2)

Private Methods:

- We know that private attributes should not be accessed from anywhere outside the class.
- Similarly, you can have private methods in your class. Which are accessible only within the class.

```
class ABC:
    def __init__(self, var1, var2):
        self.var1 = var1      # var1 is public variable
        self.__var2= var2     # var2 is private variable
```

```
    def update(self, var1):     #Public Method
        self.var1 = var1        # var1 is accessible within class
        self.__display()        #Accessing private method within the class

    def __display(self):        #Private Method
        print("From class method var1=", self.var1)
        print("From class method var2=", self.__var2)

obj=ABC(10,20)                  # Creating obj as object of class ABC
obj.update(30)                  # Accessing public method outside the class
obj._ABC__display()             #Accessing private method outside the class
```

Output:

```
From class method var1= 30
From class method var2= 20
From class method var1= 30
From class method var2= 20
```

5.10 Calling a class method from another class method

Q12 How to call class method from another class method.

Ans: You can call one class method from another class method by using self. Below program shows how can we do it.

```
class ABC():
    def method1(self):
        print("This is from method1")

    def method2(self):
        print("Executed method 2")
        self.method1()                  #method1() called in method2
```

```
obj1=ABC()
obj1.method2()
```

Output:

```
From method 2
This is from method1
```

Note: In the above program we have called method2 only. Method 2 consist a call for method 1. So method 1 will also get executed.

5.11 Class Methods

Q13. With the help of an example explain the concept of class methods.

Ans.

- Class methods are a little different from ordinary methods.
- Class methods are called by a class (not by instance of a class).
- The first argument of class method is cls, not the self.
- Class methods are widely used for factory methods, which instantiate an instance of a class, using different parameters from those usually passed to the class constructor.
- Class methods are marked with a classmethod decorator.

Example:

```
class Rectangle:
    def __init__(self,length,breadth):
        self.length=length
        self.breadth=breadth

    def area(self):
        return self.length*self.breadth
```

```
    @classmethod
    def Square(cls,side):
        return cls(side,side)

S=Rectangle.Square(10)
print("Area=",S.area())
```

Output:

Area= 100