object-oriented program(oop)

what is object oriented programing

- object oriented programing(oop) allows decompostion of a problem into a number of units called objects.
- it allows us to devlop application using objects and classes.

why to use oop?

- python was designed with an object - oriented approach.oop offers the following advantages.
  - provides a clear program structure , which makes it easy to map real world problems and their solutions.
  - it makes the development and maintenance easier.
  - imparts code reusbility

class

- class is collection of variables and methods

```
syntax :-
        class ClassName:
        list of variables | attributes
        list of methods
```

difference b/w methods and function

function : is a collection of statements

method:

- if function is defined inside the class then we will call this is a method
- a name given to function which is defined inside a class

object an obj is simply a collection of data(variables) and methods (fun)that act those data

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [1]:

```
class Hi:
    a,b=10,13
    def display():
        print('Hi i am from display method')
obj = Hi
print(obj.a)
print(obj.b)
obj.display()

```

```
10
13
Hi i am from display method
```

In [9]:

```
class Math:
    def add(num1,num2):
        return num1+num2
    def mul(num1,num2):
        return num1*num2
    def isEven(num1):
        if num1%2 ==0:
            return True
        else:
            return False
obj = Math
print(obj.add(12,13))
print(obj.mul(2,4))
print(obj.isEven(12))
```

```
25
8
True
```

constructor

- use to instatite ann object.
- it's task is to intialize (assign values) to the data members of a class when object of a class is created.

```
    syntax :
class ClassName:
      def__init__(self):
      def__init__(self,a,b):
      def__init(a,b,self):
- init is a default constructor
```

In [11]:

```
class Math:
    def  __init__(self,val1,val2):
        self.val1 = val1
        self.val2 = val2
        print('i am calling without using object')
    def show(self):
        print(self.val1+self.val2)
obj = Math(12,6)
obj.show()
```

```
i am calling without using object
18
```

In [29]:

```
### example of user defined constructor

class cons:
    def  __mycons__(self,val):
        self.val = val
        print('My constructor')
    def isEven(self):
        if self.val%2==0:
            return True
        else:
            return False
obj = cons()
obj.__mycons__(5)
obj.isEven()
```

```
My constructor
```

Out[29]:

```
False
```

In [33]:

```
class Math:
    def __init__(abc,val1,val2):
        abc.val1 = val1
        abc.val2 = val2
    def add(abc):
        print(abc.val1+abc.val2)
    def sub(abc):
        print(abc.val1-abc.val2)

obj = Math(10,5)
obj.add()

```

```
15
```

In [34]:

```
obj.sub()
```

```
5
```

inheritance

- inheritance allows us to define a class that inherits all the methods and properties from anthor class. -parent class is the class begin inherited from,also called class. -child class is the class that inherits from another class,also called dervied class.

```
syntax : class childclass(parentClass):
```

In [40]:

```
class ClassA:
    a,b = 2,3
    def display():
        print('I am from ClassA')
class ClassB(ClassA):
    c,d = 12,13
    def show():
        print('I am from classb')
obj = ClassB
print(obj.a)
obj.display()
```

```
2
I am from ClassA
```

In [42]:

```
class ClassA:
    a,b = 2,3
    def display():
        print('I am from ClassA')
class ClassB(ClassA):
    c = 12
    def add():

        print('I am from classb')
obj = ClassB
print(obj.a)
obj.display()
obj.add()
```

```
2
I am from ClassA
I am from classb
```

In [43]:

```
class ClassA:
    a,b = 2,3
    def display():
        print('I am from ClassA')
class ClassB(ClassA):
    c = 12
    def add(a,b):
        print(a+b)
obj = ClassB
print(obj.a)
obj.display()
obj.add(obj.a,obj.b)
```

```
2
I am from ClassA
5
```

multilevel inheritance

-one or more parent classes and one or .more child classes

In [52]:

```
class A:
    def classA():
        print('i am from classA')
class B(A):
    def classB():
        print('i am from classB')
class C(B):
    def classC():
        print('i am from classC')

obj = C
obj.classC()
obj.classA()
obj.classB()
```

```
i am from classC
i am from classA
i am from classB
```

Multiple inheritance

more than one or more parent class and one child class

In [54]:

```
class A:
    def classA():
        print('i am from classA')
class B():
    def classB():
        print('i am from classB')
class C(A,B):
    def classC():
        print('i am from classC')

obj = C
obj.classC()
obj.classA()
obj.classB()
```

```
i am from classC
i am from classA
i am from classB
```

In [ ]:

```

```