oops:

- 1. Python programming also support the object oriented programming.
- 2. oops stands for Object Oriented Programming System.
- 3. It is mainly used for code reusability.
- 4. In oops we can write the classes,functions,variables..

How to create a class:

- class class_name: body of class variables methods

1. class:

- It is collection of variables and methods.
- It is a blue print of object.

what is the difference between method and function.

- function:

```
- It is collection of statements.
```

- method:

```
- If a funciton is existed inside of a class is called as a method.
```

How to access the class elements:

- using basic method is .(dot) operator
- class_name.variable_name
- class_name.method_name

In [6]:
```python
# class declaration:
class Sample:
    a = 10
    b ='python'
    def display():
        print('I am from Sample class!...')
# To access the class elements:
print(Sample.a)
print(Sample.b)
Sample.display()
```

```
10
python
I am from Sample class!...
```

2. Object:

- It is a instance of a class.
- syntax:

    -object_name = class_name

In [9]:
```python
# To access the class elements through object_name.
class Arithmetic:
    def Addition(n1,n2):
        return n1+n2
    def Subtraction(n1,n2):
        return n1-n2
obj1 = Arithmetic # obj1 is object_name
print(obj1.Addition(10,20))
print(obj1.Subtraction(200,100))
```

```
30
100
```

constructor:

- It is a special method that is called when object is created.
- Two types of constructors:
- 1. default constructor
- 2. Parameterized constuctor
- syntax of constructor:

```
class class_name:
    def __init__(self):
        variables
        block of code
```

In [10]:

```python
# 1. Default Constructor:
class Default:
    def __init__(self):
        self.name='python'
    def Display(self):
        print(self.name)
obj2 = Default()
obj2.Display()
```

```
python
```

In [12]:
```python
# 1. Parameterized constructor:
class Example:
    def __init__(self,a,b):
        self.n1=a
        self.n2=b
    def printing(self):
        print(self.n1)
        print(self.n2)
obj3 = Example(10,20)
obj3.printing()
```

```
10
20
```

4. Inheritance:

- To acquaring the propertie of parent class to child class.
- Types of inheritance:
- 1. Single-Level Inheritance
- 2. Multi-Level Inheritance
- 3. Multiple Inheritance
- 4. Hirarichel Inheritance
- 5. Hybrid Inheritance

In [14]:
```python
# 1. Single-Level Inheritance:
# Only one parent class and one child class.
class Parent:
    def pdisplay():
        print('I am from parent class')
class Child(Parent):
    def cdisplay():
        print('I am from child class')
obj = Child
obj.cdisplay()
obj.pdisplay()
```

```
I am from child class
I am from parent class
```

Multi-Level Inheritance:

- Here more than one parent class and more than one child class.

In [17]:
```python
# Multi-Level Inheritance..
class Grandparent:
    def gdisplay():
        print('I am from Grand parent class')
class Parent(Grandparent):
    def pdisplay():
        print('I am from parent class')
class Child(Parent):
    def cdisplay():
        print('I am from Child Class')
ch = Child
ch.gdisplay()
ch.pdisplay()
ch.cdisplay()
```

```
I am from Grand parent class
I am from parent class
I am from Child Class
```

Multiple Inheritance:

- one or more parents class and one child class

In [19]:
```
# multuple inheritance.
class Mother:
    def mdisplay():
        print('I am from mother class')
class Father:
    def fdisplay():
        print('I am from father class')
class Child(Mother,Father):
    def cdisplay():
        print('I am from child class')
ch1 = Child
ch1.mdisplay()
ch1.fdisplay()
ch1.cdisplay()
```

```
I am from mother class
I am from father class
I am from child class
```

Hierachical Inheritance:

- Here one parent class and more than one child class.

In [23]:
```
class Parent:
    def pdisplay():
        print('I am from parent class')
class Child1(Parent):
    def c1display():
        print('I am from child1 class')
class Child2(Parent):
    def c2display():
        print('I am from child2 class')
ch3 = Child2
ch3.pdisplay()
ch3.c2display()
ch4 = Child1
ch4.c1display()
```

```
I am from parent class
I am from child2 class
I am from child1 class
```

Hybrid Inheritance:

- The combination of Multi-Level and Hierarchical inheritance.

In [29]:
```python
# hybrid inheritance
#Multi-Level: Here more than one parent class and more than one child class.
#Hire: Here one parent class and more than one child class.
class School:
    def scdisplay():
        print('I am from school class')
class stu1(School):
    def s1display():
        print('I am from student1 class')
class stu2(School):
    def s2display():
        print('I am from student2 class')
class faculty(stu1,stu2):
    def fdisplay():
        print('I am from faculty class')
obj2 = faculty
obj2.scdisplay()
obj2.fdisplay()
```

```
I am from school class
I am from faculty class
```

polymarphism:

- Polymarphism means to create a many forms.
- example:

```
'+' -> symbol
-> To find the addition of two numerical values.
-> To join two string strings.
```

- same method but implenting different ways.

In [31]:
```
# operator
s1 = 'abc'
s2 ='python'
print(s1+s2)
print(10+34)
```

```
abcpython
44
```

In [33]:
```
# method.
# len()
print(len('python'))
print(len([10,20,30]))
```

```
6
3
```

In [ ]:
```

```