In [1]:

```
def add(a,b):  #formal arguments
    print("a=",a)
    print("b=",b)

add(1,2,3,4,5) # actual arguments
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-add24c0949cc> in <module>
      3     print("b=",b)
      4
----> 5 add(1,2,3,4,5) # actual arguments

TypeError: add() takes 2 positional arguments but 5 were given
```

In [1]:

```
def add(a,*b):  #formal arguments
    print("a=",a)
    print("b=",b)

add(1,2,3,4,5) # actual arguments
```

```
a= 1
b= (2, 3, 4, 5)
```

In [2]:

```
def add(a,*b):
    summation = a  # summation =1
    for i in b: #b=2
        summation +=i
    print(summation)

add(1,2,3,4,8,9)
```

```
27
```

What is oops

- oops allows decomposition of a problem into a no of units called objects.
- python is an object oriented programming language.

why to use OOPS?

- PROVIDES A CLAR PROGRAM STRUCTURE.
- it makes the development and maintaince easier.
- code reusability & complexity reduced.

Class

- class is acollection of variables and methods.

```
Syntax: class classname:
            list of variables
            list of methods
```

Object

- An object is istance of class
- object is a collection data and methods/functions.

```
syntax: objectname = classname
```

In [6]:

```
# Example for class creation

class Hi:
    a,b = 10,20
    def disply():
        print("hi, i am from display method")



obj = Hi
print(obj.a)
print(obj.b)
obj.disply()
```

```
10
20
hi, i am from display method
```

In [11]:

```
class Math:
    def add(n1,n2):
        return n1+n2
    def mul(n1,n2):
        return n1+n2

obj = Math
print(obj.add(9,7))
print(obj.mul(9,6))
```

```
16
15
```

Constructor

- It's task is to intialize to the data members of a class when an object of a class is created.

```
syntax:
    class classname:
        def _init_(self): # it is a constructor
        def_init_(self,a,b):
        def_init_(a,b,self):
```

- The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

In [15]:

```
class Math:
    def _init_(self,n1,n2):
        self.n1 = n1
        self.n2 = n2
    def show(self):
        print(self.n1)
        print(self.n2)

obj = Math(2,5)
obj.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-ab62850c2c73> in <module>
      7         print(self.n2)
      8
----> 9 obj = Math(2,5)
     10 obj.show()

TypeError: Math() takes no arguments
```

In [19]:

```
class Math:
    def  _init_ (abc,n1,n2):
        abc.n1 = n1
        abc.n2 = n2
    def show(abc):
        print(abc.n1)
        print(abc.n2)

obj = Math(4)
obj.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-985bf362d461> in <module>
      7         print(abc.n2)
      8
----> 9 obj = Math(4)
     10 obj.show()

TypeError: Math() takes no arguments
```

In [16]:

```
class Myclass:
    X=5

print(Myclass)
```

```
<class '__main__.Myclass'>
```

Single inheritance

In [23]:

```
class A:
    a,b=9,8
    def display():
        print("i am a parent class")
class B(A):
    c,d = 70,89
    def show():
        print("i am a child class")

obj = B
print(obj.b)
print(obj.d)
print(obj.display())
```

```
8
89
i am a parent class
None
```

Multilevel inheritance

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

In [25]:

```
class A:
    def classA():
        print("class a")

class B(A):
    def classB():
        print(" class B")

class C(B):
    def classC():
        print(" class C")

obj = C
print(obj.classA())
obj.classB()

```

```
class a
None
 class B
```

multiple inheritance

- more than one parent class and child class.

In [ ]:

```

```