In [1]:

```python
def add(a,b):
    print("a=",a)
    print("b=",b)

add(1,2,3,4,5)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-62f608c577b6> in <module>
      3     print("b=",b)
      4 
----> 5 add(1,2,3,4,5)

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

In [4]:

```python
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 [5]:

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

add(1,2,3,4,5)
```

```
15
```

What is Object Oriented Programming(OOPs)?

- OOPs allows decompostion of a problem into a no of units called objects.
- Python is an object oriented programming language.

Why to use OOP?

- Provides a clear program structure.
- It makes the development and maintenance easier.
- Code reusability.

Class

- Class is a collection of variables and functions.

Syntax: class className:

```
list of variables
list of methods
```

Object

- An object is also called an instance of a class.
- An object is a collectio of data and methods

Syntax: objectname = className

In [13]:

```python
# Example for class creation

class Hi:
    a,b = 10,20
    def disply():
        print("Hi, I am from display method")
        return 9

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

```
10
20
Hi, I am from display method
9
```

In [16]:

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

obj = Math
print(obj.add(12,13))
print(obj.mul(2,3))
```

```
25
6
```

Constructor

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

```
Syntax:
        class className:
            def __init__(self): it is constructor
            def __init__(self,a,b):
            def __init__(a,b,self):
```

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

In [18]:

```python
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()
```

```
2
5
```

In [21]:

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

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

```
2
5
```

In [20]:

```python
class MyClass:
    x = 5

print(MyClass)
```

```
<class '__main__.MyClass'>
```

Single inheritance

In [25]:

```python
class A:
    a,b = 10,20
    def display():
        print('I am form class A')
class B(A):
    c,d = 13,15
    def show():
        print('I am form class B')

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

```
20
I am form class A
None
```

Out[25]:

```
13
```

Multilevel inheritance

- One or more parent classes and onr or more child classeS

In [28]:

```python
class A:
    def classA():
        print("I am from classA")
class B(A):
    def classB():
        print("I am from classB")
class C(B):
    def classB():
        print("I am from classB")

obj = C
obj.classA()
```

```
I am from classA
```

In [29]:

```python
obj.classB()
```

```
I am from classB
```

Multiple inheritance

- More than one parent class and one child class

In [ ]:

```

```