Static variables:

- If the value of a variable is not varied from object to object, such type of variables we have to declare with in the class directly but outside of methods. Such type of variables are called Static variables.

- For total class only one copy of static variable will be created and shared by all objects of that class.

- We can access static variables either by class name or by object reference. But recommended to use class name.

- In the case of instance variables for every object a separate copy will be created, but in the case of static variables for total class only one copy will be created and shared by every object of that class.

How to access static variables:

1. inside constructor: by using either self or classname
2. inside instance method: by using either self or classname
3. inside class method: by using either cls variable or classname
4. inside static method: by using classname
5. From outside of class: by using either object reference or classnmae

Where we can modify the value of static variable:

Anywhere either with in the class or outside of class we can modify by using classname.

But inside class method, by using cls variable.

In general we can declare within the class directly but from out side of any method

```
#Static variable

class studinfo:
    section='6B'
    def __init__(self):
        self.usn=105

t=studinfo()

print('t:',t.section,  t.usn)
#you can use classname or object name to access static varialbe
print('t:',studinfo.section, t.usn)
```

```
t: 6B 105
t: 6B 105
```

EPA N P

Inside constructor by using class name

```
#To use static variable inside the constructor
class test:
    a=10
    def __init__(self):
        self.b=20
        test.c=30
t=test()
print(test.__dict__)
#only a and c are printed as b is a instance varialbe
```

```
{'__module__': '__main__', 'a': 10, '__init__': <function test.__init__ at 0x00
000000050BEA68>, '__dict__': <attribute '__dict__' of 'test' objects>, '__weakr
ef__': <attribute '__weakref__' of 'test' objects>, '__doc__': None, 'c': 30}
```

Inside constructor by using class name

```
#To use static variable inside the constructor
class test:
    a=10
    def __init__(self):
        self.b=20
        test.c=30
t=test()
print(test.__dict__)
#only a and c are printed as b is a instance varialbe
```

```
{'__module__': '__main__', 'a': 10, '__init__': <function test.__init__ at 0x00
000000050BEA68>, '__dict__': <attribute '__dict__' of 'test' objects>, '__weakr
ef__': <attribute '__weakref__' of 'test' objects>, '__doc__': None, 'c': 30}
```

Inside class method by using either class name or cls variable

```
# sttic varialbe Inside class method by using either class name or cls variable

class test:
    a=10
    def __init__(self):
        self.b=20
        test.c=30
    def m1(self):
        self.d=40
        test.e=50
    @classmethod
    def m2(cls):
        cls.f=60
        test.g=70
# cls referes to current class object, instead of cls other name canbe used.
t=test()
t.m1()
test.m2()
print(test.__dict__)
```

In [7]:

```
# sttic varialbe Inside class method by using either class name or cls varic

class test:
    a=10
    def __init__(self):
        self.b=20
        test.c=30
    def m1(self):
        self.d=40
        test.e=50
    @classmethod
    def m2(cls):
        cls.f=60
        test.g=70
# cls referes to current class object, instead of cls other name canbe used.
t=test()
t.m1()
test.m2()
print(test.__dict__)

```

```
{'__module__': '__main__', 'a': 10, '__init__': <function test.__init__ at 0x00
00000005378678>, 'm1': <function test.m1 at 0x0000000005378SE8>, 'm2': <classme
thod object at 0x0000000005376088>, '__dict__': <attribute '__dict__' of 'test'
objects>, '__weakref__': <attribute '__weakref__' of 'test' objects>, '__doc_
_': None, 'c': 30, 'e': 50, 'f': 60, 'g': 70}
```

Inside static method by using class name

```
#Static varialbe inside static method by using class name

class test:
    a=10
    def __init__(self):
        self.b=20
        test.c=30
    def m1(self):
        self.d=40
        test.e=50

    @classmethod
    def m2(cls):
        cls.f=60
        test.g=70

    @staticmethod
    def m3():
        test.h=80

# if we dont class static method also works
t=test()
t.m1()
#t.m2()
test.m2()
#t.m3()
test.m3()
print(test.__dict__)
```

Local variables:

Sometimes to meet temporary requirements of programmer, we can declare variables inside a method directly, such type of variables are called local variable or temporary variables.

Local variables will be created at the time of method execution and destroyed once method completes.

Local variables of a method cannot be accessed from outside of method.

```
#Local varialbe
class Test:
    def m1(self):
        a=1000
        print(a)
    def m2(self):
        b=2000
        print(b)
t=Test()
t.m1()
t.m2()
```

```
1000
```

Static Methods:

In general these methods are general utility methods.

Inside these methods we won't use any instance or class variables.

Here we won't provide self or cls arguments at the time of declaration.

We can declare static method explicitly by using @staticmethod decorator

We can access static methods by using classname or object reference

Note: In general we can use only instance and static methods. Inside static method we can access class level variables by using class name.

class methods are most rarely used methods in python.

Destructors:

Destructor is a special method and the name should be __del__

Just before destroying an object Garbage Collector always calls destructor to perform clean up activities (Resource deallocation activities like close database connection etc).

Once destructor execution completed then Garbage Collector automatically destroys that object.

Note: The job of destructor is not to destroy object and it is just to perform clean up activities.

Note: If the object does not contain any reference variable then only it is eligible fo GC. ie if the reference count is zero then only object eligible for GC

Garbage Collection:

In Python, We have some assistant which is always running in the background to destroy useless objects. Because this assistant the chance of failing Python program with memory problems is very less. This assistant is nothing but Garbage Collector.

Hence the main objective of Garbage Collector is to destroy useless objects.

If an object does not have any reference variable then that object eligible for Garbage Collection.

How to enable and disable Garbage Collector in our program:

By default Garbage collector is enabled, but we can disable based on our requirement. In this context we can use the following functions of gc module.

1. gc.isenabled()

Returns True if GC enabled

2. gc.disable()

To disable GC explicitly

3. gc.enable()

To enable GC explicitly

In [6]:

```
class Math:
    @staticmethod
    def add(x,y):
        print('The Sum:',x+y)

    @staticmethod
    def product(x,y):
        print('The Product:',x*y)

    @staticmethod
    def average(x,y):
        print('The average:',(x+y)/2)

Math.add(10,20)
Math.product(10,20)
Math.average(10,20)
```

```
The Sum: 30
The Product: 200
The average: 15.0
```

In [ ]: