```
self is an implicit variable which is always provided by pvm
self is always pointing to the current object
for every constructor and instace method the first arguemnt should be self.
within the class self can be  used to declare instance variables and to access instance
variables
```

```
Constructor
special method in python
__init__()
will be exeecuted at the time of object creation
purpose is to declare and initialize the instance variables
per object constructor can be executed only once

constructor has atlest one arguement(self)
not mandetory for every class but python will provide default constructor
```

In [1]:

```
class Test:
    def __init__(self):
        print('constructor executed')

t = Test()
```

```
constructor executed
```

In [5]:

```
# overloading constructor

class Test:
    def __init__(self):
        print('no args')

    def __init__(self,x):   # overloaded constructors(not available in python)
        print('one-arg')

        # when ever we are writing second constructor that is priorited and
        # first one is no more (puython will consider most recent one or last)

# t1 = Test() #try uncommenting it
t2 = Test(10)
```

```
one-arg
```

In [9]:

```
class Student:
    def __init__(self,name, rollno):
        print('constructor execution')
        self.name = name
        self.rollno = rollno

    def display(self):
        print('method execution')
        print('Hello Myself:',self.name)
        print('Roll no ', self.rollno)
a = Student('Pranav', 100)
a.display()
```

```
constructor execution
method execution
Hello Myself: Pranav
Roll no  100
```

```
purpose of the constructor is to declare and initialize the instance variables\
constructor can be called only ones for the object
purpose of the methods is to perform some business logic and code
method can be called any no of times

name of method can be any thing but name of constructor should be __init__


method will be executed if we call

constructor will be executed automatically

we can implement certain business logic in constructor but it is not a good
coding practice
```

```
Inside python class

3 types of variables

1. Instance variables / object level variabes
        seperate copy of instace variabes is created for every object
        if the value of the variabe varies from object to object it is instance
        variable
        declared using self


2. static variables / class level variables
        for all objects one copy is created




3. Local variables
        variables declared inside the method or constructor without self
```

3 Types of Methods

1. Instance Methods / object related methods
        first arguement is self

2. class methods / class related
        @classmethod decorator is mandetory for class method
        It can modify a class state that would apply across all the
        instances of the class. For example, it can modify a class variable
        that would be applicable to all the instances.

        cls is reference variable to class object
        for every class one special object will be created by PVM to
        maintain class level information, which is nothing but class livel
        object. cls is the reference variable pointing to that object

        used to point to the class variables

        can be accessed by the object and class

3. static methods / general utility method
        declared by using @staicmethod decorator
        no self, no cls

        if you not declare the decorator then you can call by using Class
        name

        can be called as

if decorators are not given they are treated as instance methods

In [23]:

```
class Student:
    college_name = ' SITS'              # class variables
    def __init__(self,name, rollno):   #here name and rollno are local variabes

        print('constructor execution')
        self.name = name              # instance variables
        self.rollno = rollno          # instance variables

    def display(self):    # Instance method
        identy_schema = 100         # local variables
        print('method execution')
        print('Hello Myself:',self.name)
        print('Roll no ', self.rollno)

    @classmethod              # decorator
    def getCollegeName(cls):
        print('College Name: ', cls.college_name)

    @staticmethod     # try commeting this
    def findAverage(x,y):
        print('Average len of words')



a = Student('Pranav', 100)
print(a.college_name)
a.display()

a.getCollegeName()
a.findAverage(10, 20)
```

```
constructor execution
 SITS
method execution
Hello Myself: Pranav
Roll no  100
College Name:   SITS

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_31540/1984987955.py in <module>
     28
     29 a.getCollegeName()
---> 30 a.findAverage(10, 20)

TypeError: findAverage() takes 2 positional arguments but 3 were given
```

In [24]:

```
Student('Pranav', 100)


Student.getCollegeName()
Student.findAverage(10, 20)
```

```
constructor execution
College Name:   SITS
Average len of words
```

Instance variables

```
|if value of a variables is varied form object to object
every object has a seperate copy

Where we can declare instance variables

1. Inside constructor by using self
2. Inside instance method by using self
3. From outside of the class by using object refrence
```

In [32]:

```
class Student:
    def __init__(self,name, rollno):
        self.name = name
        self.rollno = rollno

    def info(self):
        self.marks= 60 # instance variable
        x=10             # local variable


s1 = Student('Durga',101)   # this object will have only two instance methoc
                            # as the instance method is not called

print(s1.__dict__)   # gives the instance variables

print("--"*35)

s1.info()         # now there willbe 3 instance variables

print(s1.__dict__)
print("--"*35)


s1.age = 24        # instance variable created outside the class

print(s1.__dict__) # dict with object gives instance variables

# now for the second object
print("--"*35)
s2= Student('Pavan',102)
s2.wife = 'Renu'
print(s2.__dict__)
```

```
{'name': 'Durga', 'rollno': 101}
----------------------------------------------------------------------
{'name': 'Durga', 'rollno': 101, 'marks': 60}
----------------------------------------------------------------------
{'name': 'Durga', 'rollno': 101, 'marks': 60, 'age': 24}
----------------------------------------------------------------------
{'name': 'Pavan', 'rollno': 102, 'wife': 'Renu'}
```

In [34]:

```
# Accessing the instance variables


# within the class by using self
# from outside by using refrence variables

class Student:
    def __init__(self,name, rollno):
        self.name = name
        self.rollno = rollno

    def display(self):
        print('hello my rollno is:', self.name)
        print('My roll no:', self.rollno)
s = Student('Durga', 101)
s.display()
print("--"*35)

print(s.name, s.rollno)
```

```
hello my rollno is: Durga
My roll no: 101
----------------------------------------------------------------------
Durga 101
```

```
how to delete the instance variables

syntax
del  self.variablename
del objectreference. variablename
```

In [42]:

```
class Test:
    def __init__(self):
        self.a = 10
        self.b = 10
        self.c = 10
        self.d = 22

    def delete(self):
        del self.b
        del self.c

t1 = Test()

print(t1.__dict__)
print("--"*35)
t1.delete()

print(t1.__dict__)
print("--"*35)

del t1.d
print(t1.__dict__)
print("--"*35)

del t1
```

```
{'a': 10, 'b': 10, 'c': 10, 'd': 22}
----------------------------------------------------------------------
{'a': 10, 'd': 22}
----------------------------------------------------------------------
{'a': 10}
----------------------------------------------------------------------
```

In [43]:

```
class Test:
    def __init__(self):
        self.a=20
        self.b=20

t1 = Test()
t2 = Test()
t1.a = 888
t1.b = 999
print(t1.a, t1.b)
print(t2.a, t2.b)
```

```
888 999
20 20
```

static variables

```
instance variable: For every object a seperate copy

static: For all objects a single copy maintained at class level
```

In [44]:

```
class Student:
    cname =  'SITS'
    def __init__(self,name,rollno):
        self.name = name
        self.rollno = rollno

s1 = Student('Narayan', 101)
s1 = Student('Narayan', 101)

print(s1.name, s1.rollno, Student.cname)
print(s2.name, s2.rollno, Student.cname)
```

```
Narayan 101 SITS
Pavan 102 SITS
```

various places to declare static variables

1. In class directly but from outside form method
2. Inside constructor by usig classname
3. Inside instance method by using classname
4. inside the classmethod by using cls or classname
5. Inside the staicmethod by using classname
6. outside the class by using class name

In [45]:

```
class Test:
    a = 10
    def __init__(self):
        Test.b=20
    def m1(self):
        Test.c= 30

    @classmethod
    def m2(cls):
        cls.d = 40
        Test.e = 50

    @staticmethod
    def m3():
        Test.f = 60
Test.g = 70

print(Test.__dict__)  # dict with Classname gives static variables
```

```
{'__module__': '__main__', 'a': 10, '__init__': <function Test.__init__ at 0
x0000025B42DFA280>, 'm1': <function Test.m1 at 0x0000025B42DFA4C0>, 'm2': <c
lassmethod object at 0x0000025B45286E50>, 'm3': <staticmethod object at 0x00
00025B452869D0>, '__dict__': <attribute '__dict__' of 'Test' objects>, '__we
akref__': <attribute '__weakref__' of 'Test' objects>, '__doc__': None, 'g':
70}
```

how to access static varaiables

we can access static variable either by classname or by object refernce

within the class

classname, self, cls

outside of the class

object refrence, classname

In [47]:

```
class Test:
    a = 10

    def __init__(self):
        print('Inside constructor')
        print(Test.a)
        print(self.a)
    def m1(self):
        print('inside instance method')
        print(Test.a)
        print(self.a)
    @classmethod
    def m2(cls):
        print('inside classmethod')
        print(Test.a)
        print(cls.a)
    @staticmethod
    def m3():
        print('Inside static method')
        print(Test.a)

t = Test()
t.m1()
t.m2()
t.m3()
print('From outside the class')
print(Test.a)
print(t.a)
```

```
Inside constructor
10
10
inside instance method
10
10
inside classmethod
10
10
Inside static method
10
From outside the class
10
10
```

```
# modifying static variables
```

within the class we should use classname, cls variable

from outside of the class: only classname

In [53]:

```
class Test:
    a = 10

    def __init__(self):
        Test.a = 20               # modification

    @classmethod
    def m1(cls):
        cls.a= 30
        Test.a = 40

    @staticmethod
    def m2():
        Test.a = 50




t = Test()
t.m1()
t.m2()

Test.a = 61
print(Test.a)
print(t.a)
```

```
61
61
```

Deleting the static variables

within the class we should use classname, cls variable

from outside of the class: only classname

In [58]:

```
class Test:
    a = 10
    b= 20
    c=30

    def __init__(self):
        del Test.a

    @classmethod
    def m1(cls):
        del cls.b


    @staticmethod
    def m2():
        del Test.c


print(Test.__dict__)
t = Test()
print(Test.__dict__)
t.m1()
print(Test.__dict__)
t.m2()
print(Test.__dict__)
```

```
{'__module__': '__main__', 'a': 10, 'b': 20, 'c': 30, '__init__': <function
Test.__init__ at 0x0000025B450A1790>, 'm1': <classmethod object at 0x0000025
B42E8AC10>, 'm2': <staticmethod object at 0x0000025B42E8A5E0>, '__dict__': <
attribute '__dict__' of 'Test' objects>, '__weakref__': <attribute '__weakre
f__' of 'Test' objects>, '__doc__': None}
{'__module__': '__main__', 'b': 20, 'c': 30, '__init__': <function Test.__in
it__ at 0x0000025B450A1790>, 'm1': <classmethod object at 0x0000025B42E8AC10
>, 'm2': <staticmethod object at 0x0000025B42E8A5E0>, '__dict__': <attribute
'__dict__' of 'Test' objects>, '__weakref__': <attribute '__weakref__' of 'T
est' objects>, '__doc__': None}
{'__module__': '__main__', 'c': 30, '__init__': <function Test.__init__ at 0
x0000025B450A1790>, 'm1': <classmethod object at 0x0000025B42E8AC10>, 'm2':
<staticmethod object at 0x0000025B42E8A5E0>, '__dict__': <attribute '__dict_
_' of 'Test' objects>, '__weakref__': <attribute '__weakref__' of 'Test' obj
ects>, '__doc__': None}
{'__module__': '__main__', '__init__': <function Test.__init__ at 0x0000025B
450A1790>, 'm1': <classmethod object at 0x0000025B42E8AC10>, 'm2': <staticme
thod object at 0x0000025B42E8A5E0>, '__dict__': <attribute '__dict__' of 'Te
st' objects>, '__weakref__': <attribute '__weakref__' of 'Test' objects>, '_
_doc__': None}
```

In [ ]:

```

```