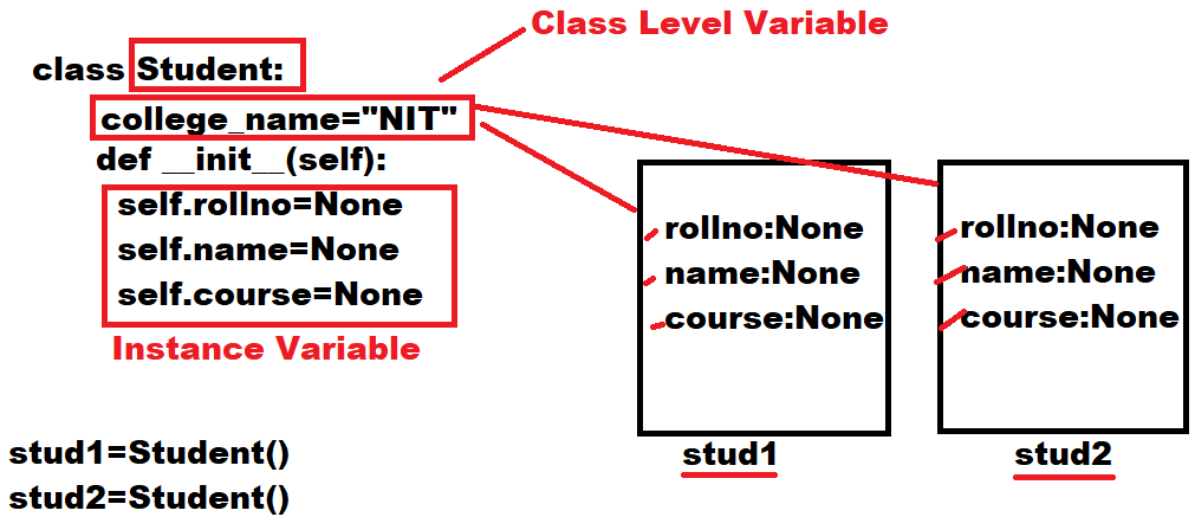
**Class level variables and Methods**

Any variable within class is bind with class name and outside the class bind with class name is called class level variable.

These variables are global variables, which are global to more than one object.

Class level variables memory is allocated within class.

This memory is allocated once; these variables are accessible without creating object.




**Syntax:**

```
class <class-name>:
        <class-level-variable>
      <class-level-variable>
      Def __init__(self):
            self.<instance variable>
            self.<instance variable>
```

**Example:**

```
class A:
   x=100  # class level variable
   def __init__(self):
      self.y=200 # instance variable/object level variable
```

```
print(A.x)


obj1=A()
print(obj1.y)
obj2=A()
print(obj2.y)
```

**Output**

```
100
200
200
```

**Example:**

```
class Circle:
    pi=3.147  # Class Level Variable
    def __init__(self,r):
        self.radius=r  # Instance Variable / Object Level Variable
    def findArea(self):
        return Circle.pi*self.radius*self.radius


circle1=Circle(1.5)
circle2=Circle(1.7)

area1=circle1.findArea()
area2=circle2.findArea()
print(f'Area of Circle1 {area1}')
print(f'Aera of Circle2 {area2}')
```

**Output**

```
Area of Circle1 7.080749999999999
Aera of Circle2 9.09483
```

**Example:**

```
class Account:
    min_balance=2000
    def __init__(self,a,c,b):
```

```
        self.__accno=a
        self.__cname=c
        self.__balance=b
    def print_account(self):
        print(f'''AccountNo {self.__accno},
CustomerName {self.__cname},
Balance {self.__balance}''')
    def deposit(self,t):
        self.__balance=self.__balance+t
    def withdraw(self,t):
        if (self.__balance-t)<Account.min_balance:
            print("Insuff Balance")
        else:
            self.__balance=self.__balance-t


cust1=Account(101,"naresh",5000)
cust2=Account(102,"suresh",8000)
cust1.print_account()
cust2.print_account()
cust1.deposit(2000)
cust1.print_account()
cust2.withdraw(2000)
cust2.print_account()
```

**Output**

```
AccountNo 101,
CustomerName naresh,
Balance 5000
AccountNo 102,
CustomerName suresh,
Balance 8000
AccountNo 101,
CustomerName naresh,
Balance 7000
AccountNo 102,
CustomerName suresh,
Balance 6000
```

**Example:**

```
class Product:
    product_count=0
    def __init__(self,pn,p):
        self.__pname=pn
        self.__price=p
        Product.product_count+=1
        print("Product Created...")



print(f'Product Count is {Product.product_count}')
p1=Product("keyboard",2000)
p2=Product("mouse",1200)
print(f'Product Count is {Product.product_count}')
```

**Class level method**

A method defined inside class with first parameter as “cls”, it is called class level method.
This method is bind with class name (OR) this method is called with class name without creating object.
This method defines class level operation.

**Syntax:**

```
class <class-name>:
        def instance-method-name(self,param,param,param):
                statement-1
                statement-2

        @classmethod
        def class-method-name(cls,param,param,param):
```

```
        statement-1
        statement-2
```

To define a method as class level, it is decorated using @classmethod decorator. @classmethod is predefined decorator.

Class level method cannot access instance members
Class level method access only class level members
Instance method access class level members and instance members

**Example:**

```
class A:
    y=200
    def __init__(self):
        self.x=100
    def m1(self):
        print("instance method")
        print(self.x)
        print(A.y)
    @classmethod
    def m2(cls):
        print("class method")
        print(cls.y)



A.m2()
obj1=A()
obj1.m1()
```

**Output**

```
class method
200
instance method
100
200
```

**Example:**

```
class Student:
    __college_name="NIT"
    def __init__(self,r,n):
        self.__rollno=r
        self.__name=n
    def print_student(self):
        print(f'''{self.__rollno},
{self.__name},
{Student.__college_name}''')
    @classmethod
    def set_college_name(cls,name):
        cls.__college_name=name
    @classmethod
    def get_college_name(cls):
        return cls.__college_name


print(Student.get_college_name())
stud1=Student(101,"naresh")
stud2=Student(102,"suresh")
stud1.print_student()
stud2.print_student()
Student.set_college_name("NIIT")

stud1.print_student()
stud2.print_student()
```

**Output**

```
NIT
101,
naresh,
NIT
102,
suresh,
NIT
101,
naresh,
NIIT
```

102,
suresh,
NIIT

**Static method**

A method defined inside class without implicit first argument is called static method. This method does not have first argument as “self” or “cls”.

Static methods are global methods, these methods performs global operations. This operation does not belong any class or object.

To declare a method as static @staticmethod decorator is used

**Syntax:**

```
class <class-name>:
        def instance-method(self,param,param,...):
                statement-1
                statement-2
        @classmethod
        def class_method(cls,param,param):
                statement-1
                statement-2
        @staticmethod
        Def static_method(param,param,param):
                Statement-1
                Statement-2
```

This method is bind with class name and can be invoked without creating object.

**Example:**

```
class Math:
    @staticmethod
    def power(num,p):
        return num**p
    @staticmethod
    def factorial(num):
```

```
        if num==0:
            return 1
        else:
            return num*Math.factorial(num-1)
    @staticmethod
    def isPrime(num):
        c=0
        for i in range(1,num+1):
            if num%i==0:
                c=c+1
        return c==2



res1=Math.power(5,2)
res2=Math.factorial(4)
res3=Math.isPrime(7)

print(res1,res2,res3)
```

**Output**

```
25 24 True
```

**Class reusability**

Object oriented application is a collection of classes. The content of one class can be used inside another class in different ways.

1. Composition (Has-A)
2. Aggregation (Use-A)
3. Inheritance (IS-A)