Static Method
A method without implicit first argument is called static methods
Static method defines global operations
In order to write individual methods.
In order to transform method to a static, we need to use @staticmethod decorator

```
class <class-name>:
        @staticmethod
        def <method-name>(arg1,arg2,arg3,…):
                statement-1
                statement-2
        def <method-name>(self,arg1,arg2,..):
                statement-1
                statement-2
        @classmethod
        def <method-name>(cls,arg1,arg2,arg3,…):
                statement-1
                statement-2
```

static methods bind with class name or invoked with class name

Example:

```
class Math:
   @staticmethod
   def power(num,p):
      return num**p
   @staticmethod
   def iseven(num):
      if num%2==0:
         return True
      else:
         return False
   @staticmethod
   def isodd(num):
      if num%2!=0:
         return True
      else:
         return False
```

```
def main():
    res1=Math.power(5,3)
    res2=Math.iseven(5)
    res3=Math.isodd(4)
    print(res1,res2,res3)
main()
```

Output:

```
125 False False
>>>
```

Example:

```
class ComplexOperations:
    @staticmethod
    def add_complex(c1,c2):
        return c1+c2
    @staticmethod
    def sub_complex(c1,c2):
        return c1-c2

def main():
    comp1=1+2j
    comp2=1+3j
    comp3=ComplexOperations.add_complex(comp1,comp2)
    comp4=ComplexOperations.sub_complex(comp2,comp1)
    print(comp1,comp2,comp3,comp4,sep="\n")

main()
```

Output:

```
(1+2j)
(1+3j)
(2+5j)
1j
```

Static methods can be used as factory methods
A method creates an object and return address is called factory method

Example:

```
import datetime
class Person:
    def __init__(self,n,a):
        self.__name=n
        self.__age=a
    @staticmethod
    def create_person(n,y):
        year=datetime.date.today().year
        a=year-y
        p=Person(n,a)
        return p
    def print_person(self):
        print(f'Name: {self.__name}')
        print(f'Age: {self.__age}')


def main():
    p1=Person("naresh",50)
    p2=Person.create_person("suresh",2000)
    p1.print_person()
    p2.print_person()
main()
```

Output:

```
Name: naresh
Age: 50
Name: suresh
Age: 22
>>>
```

Q: What is difference between object level method, class level method and static method?

Object level method or instance method	**Class level method**	**Static method**
A method defined inside class with first argument as "self" is called object level method or instance	A method defined inside class with first argument as "cls" is called class level method	A method defined inside class without implicit first argument called static method This method is defined

method	This method is defined with @classmethod decorator	with @staticmethod decorator
This method is bind with object name or it cannot called without creating object	This method is bind with class name	This method is bind with class name
This method is able to access object level data or variables	This method is able access class level data	This method is global method which perform global operation

Syntax:
def <method-name>(self,arg1,…):
   statement-1
	Syntax:
	@classmethod
	def <method-name>(cls,arg1,..):
	   statement-1
		Syntax:
		@staticmethod
		def <method-name>(arg1,…_):
		   statement-1

Example:

```
class A:
   p=100
   q=200
   def __init__(self):
      self.x=100
      self.y=200
   def add(self): # object level method
      return self.x+self.y
   def sub(self):
      return self.x-self.y
   @classmethod
   def m1(cls): # class level method
      print(cls.p,cls.q)
   @staticmethod
   def m2():
      print("static method")

def main():
   A.m1()
   obj1=A()
   print(obj1.add())
   A.m2()
```

```
main()
```

Output:

```
100 200
300
static method
>>>
```

setattr,getattr functions

getattr(object, name[, default])
Return the value of the named attribute of object. name must be a string. If the string is the name of one of the object's attributes, the result is the value of that attribute.

setattr(object, name, value)
This is the counterpart of getattr(). The arguments are an object, a string and an arbitrary value. The string may name an existing attribute or a new attribute. The function assigns the value to the attribute, provided the object allows it.

Example:

```
class Employee:
    pass

emp1=Employee()
setattr(emp1,"empno",1)
setattr(emp1,"ename","naresh")
setattr(emp1,"salary",6000)
print(getattr(emp1,"empno"))
print(getattr(emp1,"ename"))
print(getattr(emp1,"salary"))
emp2=Employee()
setattr(emp2,"empno",2)
setattr(emp2,"ename","suresh")
setattr(emp2,"salary",7000)
print(getattr(emp2,"empno"))
print(getattr(emp2,"ename"))
print(getattr(emp2,"salary"))
```

Output:
```
1
naresh
6000
2
suresh
7000
```

Example:
```
class Marks:
    def __init__(self,r,s1,s2):
        self.__rno=r
        self.__sub1=s1
        self.__sub2=s2
    def print_marks(self):
        print(f'{self.__rno},{self.__sub1},{self.__sub2}')
    def get_sub1(self):
        return self.__sub1
    def get_sub2(self):
        return self.__sub2

def main():
    stud1=Marks(1,60,70)
    stud1.print_marks()
    setattr(stud1,"total",stud1.get_sub1()+stud1.get_sub2())
    print(f'total marks',getattr(stud1,"total"))

main()
```

Output:
```
1,60,70
total marks 130
>>>
```

properties class