__hash__() method of object class

This method is used for generating hash value of object. This hash value is used by hash based data structure (set, dict) for finding key. This method called by hash() function.

Example:

```
class Employee: # creating data type
    def __init__(self,eno,en):
        self.__empno=eno
        self.__ename=en
    def __str__(self):
        return f'{self.__empno},{self.__ename}'
    def __hash__(self):
        return self.__empno
    def __eq__(self, other):
        return self.__empno==other.__empno



e1=Employee(101,"naresh")
e2=Employee(102,"suresh")
e3=Employee(103,"kishore")
e4=Employee(101,"ramesh")
print(e1,e2,e3,sep="\n")
emp_set={e1,e2,e3,e4}
for emp in emp_set:
    print(emp)
```

Output

```
101,naresh
102,suresh
103,kishore
101,naresh
102,suresh
```

103,kishore

__eq__(self,other)

This method is used to compare two objects. This method is called when two objects are compared using == operator.

```
class Triangle:
    def __init__(self,b,h):
        self.__base=b
        self.__height=h
    def __eq__(self, other):
        return self.__base==other.__base and
self.__height==other.__height


t1=Triangle(1.5,1.7)
t2=Triangle(1.5,1.7)
b=t1==t2  # t1.__eq__(t2)
print(b )
```

Output

True

Example:

```
class Complex: # UDT
    def __init__(self,r,i):
        self.__real=r
        self.__img=i
    def __str__(self):
        return f'{self.__real},{self.__img}'
    def __add__(self, other):
        c3=Complex(0,0)
        c3.__real=self.__real+other.__real
        c3.__img=self.__img+other.__img
```

```
        return c3


c1=Complex(1.2,1.5)
c2=Complex(1.7,1.8)
print(c1)
print(c2)
c3=c1+c2
print(c3)
```

Output

1.2,1.5
1.7,1.8
2.9,3.3

__new__() method of object class

Python __new__() method is static method (a.k.a magic or dunder method) that gives the programmer more control over how a specific class (cls) is instantiated.

How does the Python __new__() method work?

Python calls the __new__() method every time you instantiate a class. It does the instantiation in two steps:

1. First, it invokes the __new__() method of the class to create and return an instance (this instance is then passed to __init__() as its first argument self)

2. Next, the __init__() method is invoked to initialize the object state.

Example:

```
class Alpha:
    def __new__(cls, *args, **kwargs):
        print("inside new method")
        Alpha.__init__(cls)
    def __init__(self):
        print("inside init method")
        print(self)



a1=Alpha()
```

Output

```
inside new method
inside init method
<class '__main__.Alpha'>
```

Example:

```
class PosInteger(int):
    def __new__(cls, value):
        return int.__new__(cls,abs(value))



a=PosInteger(10)
print(a )
b=PosInteger(-30)
print(b)
```

Output

```
10
30
```

Example:

```
class PosInteger(int):
    def __new__(cls, value):
        return int.__new__(cls,abs(value))
    def __init__(self,value):
        print(self)



a=PosInteger(10)
print(a )
b=PosInteger(-30)
print(b)
```

Output

10

10

30

30

Example:

```
class UpperString(str):
    def __new__(cls, value):
        return str.__new__(cls,value.upper())



str1=UpperString("python")
print(str1)
str2=UpperString("PYTHON")
print(str2)
```

Output:

PYTHON

PYTHON

Example:

```
class A:
    def __new__(cls):
        return "A class object"

class B(A):
    def __new__(cls):
        return A.__new__(cls)+"B class Object"


objb=B()
print(objb)
```

Output

A class objectB class Object

Differences Between __new__ and __init__

__new__ is a static method, while __init__ is an instance method.
__new__ is responsible for creating and returning a new instance, while __init__ is responsible for initializing the attributes of the newly created object.
__new__ is called before __init__.
__new__ happens first, then __init__.
__new__ can return any object, while __init__ must return None.

Example:

```
class Employee:
    def __new__(cls, *args, **kwargs):
        print("Employee object is created")
        a=super().__new__(cls)
        return a
```

```
    def __init__(self,eno,en): #initial object
        self.empno=eno
        self.ename=en
    def __str__(self):
        return f'{self.empno},{self.ename}'



emp1=Employee(101,"naresh") # Employee.__new__
print(emp1)
```

Output

Employee object is created

101,naresh