Q1. Create a class called number which contains an integer. It should contain various methods to set the value of the integer, return the integer value, print the integer value, check whether the integer is negative.

In [1]:
```
class Number:
    '''Check whether the integer is negative or not'''

    def set_integer(self):
        self._num = int(input("Enter the integer : "))
    def rtn_integer(self):
        print("Return Integer is : ",str(self._num))

    def print_integer(self):
        print('Print the integer is : ',str(self._num))
    def check_integer(self):
        print('' * 60)
        if self._num == 0 or self._num >= 0:
            print(str(self._num) + ' is the positive integer.')
        else:
            print(str(self._num) + ' is the negative integer.')
        print('' * 60)
num = Number()
num.set_integer()
num.rtn_integer()
num.print_integer()
num.check_integer()
```

```
Enter the integer : -20
Return Integer is :  -20
Print the integer is :  -20
************************************************************
-20 is the negative integer.
************************************************************
```

In [ ]:

Q2.Create a class that can calculate the perimeter/ circumference and area of a regular shape. The class should also have a provision to accept the data relevant to the shape.

In [2]:
```
class square:
    '''Area and perimeter of a square.'''
    def __init__(self,a):
        self._area = a
    def area_of_square(self):
        print('Area of the square is : ',self._area * self._area)
    def perimeter_of_square(self):
        print('' * 50)
        print('Perimeter / Circumference of the square is : ', 4 * self._area)

s= square(10)
s.area_of_square()
s.perimeter_of_square()
```

```
Area of the square is :  100
**************************************************
Perimeter / Circumference of the square is :  40
```

In [ ]:

Q3.Create a class called student which contains the data members called rollno, sname and branch. The class should contain methods to initialize the data members and to set and display data members. Finally, the class should contain the **del** method to release the memory at the time of object destruction. Test your class with at least two objects of the student class

In [3]:
```
class Student:
    def __init__(self,r=0,s='',b=''):
        self._rno = r
        self._name = s
        self._branch = b

    def set_data(self,r,s,b):
        self._rno = r
        self._name = s
        self._branch = b

    def display_data(self):
        print('The student '+self._name+ ' Roll.No is '+ str(self._rno)+' and the branch is :',self._branch)
    def __del__(self):
        print('Deleting the object...',str(self))
stu = Student(26,'Aravindh','CSE')
stu.display_data()
print('*' * 50)
stu1 = Student(37,'Aravindh','CSE')
stu1.display_data()
```

```
Deleting the object.. <__main__.Student object at 0x000001968864850>
The student Aravindh Roll.No is 26 and the branch is : CSE
**************************************************
Deleting the object.. <__main__.Student object at 0x000001968864748>
The student Aravindh Roll.No is 37 and the branch is : CSE
```

In [ ]:

Q4.Modify the student class defined in Q. No. 1 in so that it now includes a class variable to count the number of student objects created from this class and also add a class function in order to display the value of the class variable. The modified class should be named as student1 class.

In [4]:
```
class Student1:
    c = 0
    def __init__(self,r=0,s='',b=''):
        Student1.count()
        self._rno = r
        self._name = s
        self._branch = b

    def set_data(self,r,s,b):
        self._rno = r
        self._name = s
        self._branch = b
    def display_data(self):
        print('The student '+self._name+ ' Roll.No is '+ str(self._rno)+' and the branch is :',self._branch)
    @classmethod
    def count(cls):
        cls.c += 1


stu = Student1(26,'Aravindh','ECE')
stu1 = Student1(37,'Gayu','ECE')
stu3 = Student1()
print("Number of student object created so far : ",Student1.c)
```

```


Deleting the object.. <__main__.Student object at 0x000001968888270>
Deleting the object.. <__main__.Student object at 0x0000019688641C0>
Number of student object created so far :  3
```

In [ ]:

Q5.Apply the vars() and dir() global function on student and student1 classes

In [11]:
```
print("Global vars function to class student")
print(vars(Student))
print('*' * 120)
print("Global vars function to class student1")
print(vars(Student1))
print('*' * 120)
print("Global dir function to class student")
print(dir(Student))
print('*' * 120)
print("Global dir function to class student1")
print(dir(Student1))
```

```
Global vars function to class student
{'__module__': '__main__', '__init__': <function Student.__init__ at 0x000001968884D70>, 'marks': <function Student.marks at 0x000001968CB720D0>, '__str__': <function Student.__str__ at 0x000001968CB7210A0>, '__doc__': None}
************************************************************************************************************************
Global vars function to class student1
{'__module__': '__main__', '__init__': <function Student1.__init__ at 0x000001968884310>, 'disp_data': <function Student1.disp_data at 0x000001968884F70>, '__doc__': None}
************************************************************************************************************************
Global dir function to class student
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'marks']
************************************************************************************************************************
Global dir function to class student1
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclasses__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'disp_data']
```

In [ ]:

Q6.Create a class called Date. It should contain data members day, month and year. It should also contain functions to set and display date, as well as function to initialize date objects. Overload the == operator to check if two dates are equal

In [12]:
```
class Date:
    '''Method Overloading'''
    def __init__(self,d=0,m=0,y=0):
        self._day = d
        self._month = m
        self._year = y
    def __eq__(self,other):
        if self._day == other._day and self._month == other._month and self._year == other._year:
            return 'Both Dates are equal'
        else:
            return 'Both Dates are not equal'
    def __str__(self):
        print('Equals of ' + str(self._year) + ' and ' + str(self.other))



d = Date(15,9,1998)
d1 = Date(15,9,1998)
print('Check 1 :', d == d1)
d2 = Date(19,5,1999)
d3 = Date(19,5,1998)
print('Check 2 : ', d2 == d3)
print(Date.__doc__)
```

```
Check 1 :  Both Dates are equal
Check 2 :  Both Dates are not equal
```

In [ ]:

Q7.Create a class called weather that has a list containing weather parameters. Overload the in operator that checks whether an item is present in the list or not.

In [13]:
```
class Weather:
    def __init__(self,f,c,f):
        self._temperature = t
        self._celsius = c
        self._farenheit = f
        self.list = [self._temperature,self._celsius,self._farenheit]

    def set_data(self,f,c,f):
        self._temperature = t
        self._celsius = c
        self._farenheit = f
    def weather_display(self):
        print(self.list)
    def check(self,x):
        if x in self.list:
            print("Item present in the list named as ",str(w))
        else:
            print("Item not present in the list named as , ",str(x))

    def __del__(self):
        print("Deleting the object...",str(self))


wh = Weather("Temperature","Celsius","Farenheit")
wh.weather_display()
wh.check('moisture')
print('' * 50)
wh1 = Weather("Temperature","Celsius","Moisture")
wh1.weather_display()
wh1.check('Temperature')
```

```
Deleting the object.. <__main__.Weather object at 0x000001968882850>
Item not present in the list named as ,  moisture
**************************************************
Deleting the object.. <__main__.Weather object at 0x000001968882528>
Item present in the list named as , Temperature
```

In [ ]:

Q8.Create a class called complex containing real and imaginary parts and then use it to check whether two objects are of the same type, whether their attributes are same and whether they are pointing to the same object

In [8]:
```
#id:
class Complex:
    def __init__(self,r,i='1'):
        self._real = r
        self._imaginary = i

    def __eq__(self,self1):
        if self._real == self1._real and self._imaginary == self1._imaginary and type(c1) == type(c2):
            print("Two objects,and their attributes and objects are same",type(c1),type(c2))
        else:
            print("Two objects,and their attributes and objects are not same",type(c1),type(c2))


print('Check 1:')
c1 = Complex(3,'-2')
c2 = Complex(2,'-2')
c3 = c4
print('' * 60)
print('Check 2')
c3 = Complex(2,'-2')
c4 = Complex(2,'-2')
c3 == c4
```

```


Check 1:
Two objects,and their attributes and objects are not same <class '__main__.Complex'> <class '__main__.Complex'>
************************************************************
Check 2
Two objects,and their attributes and objects are same <class '__main__.Complex'> <class '__main__.Complex'>
```

In [ ]:

Q9.Create a class called department with the attributes deptname and deptid. Modify the student class defined in Q.No. 3 such that it contains an object of department class. Name the new modified student class as student2.

In [9]:
```
class Department:
    def __init__(self,dpname,dpid):
        self._dpname = dpname
        self._dpid = dpid


    def __str__(self):
        return self._dpname + ' id is '+ str(self._dpid)


class Student1(Department):
    def __init__(self,dpname,dpid,dpyear):
        super().__init__(dpname,dpid)
        self._dpyear = dpyear

    def disp_data(self):
        print('The Department name as : '+self._dpname+ '\nDepartment id as : '+ str(self._dpid)+'\nYear of starting is :',self._dpyear)


s1 = Student1("BE",1013126,4)
s1.disp_data()
#dep.display_data()
```

```
The Department name as :  BE
Department id as :  1013126
Year of starting is :  4
```

In [ ]:

Q10.Create a class called Person. From this class, inherit another class called student. Assume suitable data members and member functions for these two classes. In both the classes, define the **str** function and demonstrate method overriding.

In [10]:
```
#id10:
class Person:
    def __init__(self,n,a):
        self._name = n
        self._age = a
    def __str__(self):
        return self._name + ' is ' + str(self._age) + ' years old'
class Student(Person):
    def __init__(self,n,a,i):
        super().__init__(n,a)
        self._id = i

    def marks(self,grade):
        if grade >= 35 :
            print(grade, ' : is a pass mark')
        else:
            print(grade,' : is a fail mark')

    def __str__(self):
        return self._name + ' has an the id as ' + str(self._id)


p = Person('Aravindh',23)
print("The person : ",p)
s = Student('Gayu',45,372)
s.marks(40)
```

```


The person :  Aravindh is 23 years old
40  : is a pass mark
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: