In [2]:

```python
#Python Program to Create a Class and Compute the Area and the Perimeter of the Circle
import math
class circle():
    def __init__(self,radius):
        self.radius=radius
    def area(self):
        return math.pi*(self.radius**2)
    def perimeter(self):
        return 2*math.pi*self.radius
r=int(input("Enter radius of circle: "))
obj=circle(r)
print("Area of circle:",round(obj.area(),2))
print("Perimeter of circle:",round(obj.perimeter(),2))
```

```
Enter radius of circle: 2
Area of circle: 12.57
Perimeter of circle: 12.57
```

In [5]:

```python
#Creating simple class and objects for counting the number of employees
#defining class
class Employee:
    #'Common base class for all employees'
    empCount = 0
#defining the constructor
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1
#defining the member functions
    def displayCount(self):
        print("Total Employee %d" % Employee.empCount)
    def displayEmployee(self):
        print ("Name : ", self.name, ", Salary: ", self.salary)
#"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
#"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)

emp1.displayEmployee()
emp2.displayEmployee()
print("Total Employee %d" % Employee.empCount)
```

```
Name :  Zara , Salary:  2000
Name :  Manni , Salary:  5000
Total Employee 2
```

In [11]:

```python
# A Python program to demonstrate inheritance
class Person(object):
# Constructor
    def __init__(self, name):
        self.name = name
# To get name
    def getName(self):
        return self.name
# To check if this person is an employee
    def isEmployee(self):
        return False

# Inherited or Subclass (Note Person in bracket)
class Employee(Person):
# Here we return true
    def isEmployee(self):
        return True

# Driver code
emp = Person("Ram") # An Object of Person
print(emp.getName(), emp.isEmployee())
emp = Employee("Raj") # An Object of Employee
print(emp.getName(), emp.isEmployee())
```

```
Ram False
Raj True
```

In [12]:

```python
# Accessing public members of the class
class Person:
    def __init__(self, name, age=0):
        self.name = name
        self.age = age
    def display(self):

        print(self.name)
        print(self.age)
person = Person('Dev', 30)
#accessing using class method
person.display()
#accessing directly from outside
print(person.name)
print(person.age)
```

```
Dev
30
Dev
30
```

In [13]:

```python
# Accessing protected members of the class using single underscore
class Person:
    def __init__(self, name, age=0):
        self.name = name
        self._age = age
    def display(self):
        print(self.name)
        print(self._age)
person = Person('Dev', 30)
#accessing using class method
person.display()
#accessing directly from outside
print(person.name)
print(person._age)
```

```
Dev
30
Dev
30
```

In [14]:

```python
# Accessing private members of the class using double underscore
class Person:
    def __init__(self, name, age=0):
        self.name = name
        self.__age = age
    def display(self):
        print(self.name)
        print(self.__age)
person = Person('Dev', 30)

#accessing using class method
person.display()
#accessing directly from outside
print('Trying to access variables from outside the class ')
print(person.name)
print(person.__age)
```

```
Dev
30
Trying to access variables from outside the class
Dev

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-14-030ca028073c> in <module>
     14 print('Trying to access variables from outside the class ')
     15 print(person.name)
---> 16 print(person.__age)

AttributeError: 'Person' object has no attribute '__age'
```

In [15]:

```python
#Using Getter and Setter methods to access private variables
class Person:
    def __init__(self, name, age=0):
        self.name = name
        self.__age = age
    def display(self):
        print(self.name)
        print(self.__age)
    def getAge(self):
        print(self.__age)
    def setAge(self, age):
        self.__age = age
person = Person('Dev', 30)
#accessing using class method
person.display()
#changing age using setter
person.setAge(35)
person.getAge()
```

```
Dev
30
35
```

In [16]:

```python
class Family:
    def show_family(self):
        print("This is our family:")
# Father class inherited from Family

class Father(Family):
    fathername = ""
    def show_father(self):
        print(self.fathername)
# Mother class inherited from Family
class Mother(Family):
    mothername = ""
    def show_mother(self):
        print(self.mothername)
# Son class inherited from Father and Mother classes
class Son(Father, Mother):
    def show_parent(self):
        print("Father :", self.fathername)
        print("Mother :", self.mothername)
s1 = Son() # Object of Son class
s1.fathername = "Mark"
s1.mothername = "Sonia"
s1.show_family()
s1.show_parent()
```

```
This is our family:
Father : Mark
Mother : Sonia
```

In [18]:

```
#Python Program to Create a Class which Performs Basic Calculator Operations
class cal():
    def __init__(self,a,b):
        self.a=a
        self.b=b
    def add(self):
        return self.a+self.b
    def mul(self):
        return self.a*self.b
    def div(self):
        return self.a/self.b
    def sub(self):
        return self.a-self.b
a=int(input("Enter first number: "))
b=int(input("Enter second number: "))
obj=cal(a,b)
choice=1
while choice!=0:
    print("0. Exit")
    print("1. Add")
    print("2. Subtraction")
    print("3. Multiplication")
    print("4. Division")
    choice=int(input("Enter choice: "))
    if choice==1:

        print("Result: ",obj.add())
    elif choice==2:
        print("Result: ",obj.sub())
    elif choice==3:
        print("Result: ",obj.mul())
    elif choice==4:
        print("Result: ",round(obj.div(),2))
    elif choice==0:
        print("Exiting!")
    else:
        print("Invalid choice!!")
```

```
Enter first number: 2
Enter second number: 3
0. Exit
1. Add
2. Subtraction
3. Multiplication
4. Division
Enter choice: 1
Result:  5
0. Exit
1. Add
2. Subtraction
3. Multiplication
4. Division
Enter choice: 0
Exiting!
```

In [19]:

```
# Python Program to Append, Delete and Display Elements of a List Using Classes
class check():
    def __init__(self):
        self.n=[]
    def add(self,a):
        self.n.append(a)
    def remove(self,b):
        self.n.remove(b)
    def dis(self):
        return (self.n)
obj=check()
choice=1
while choice!=0:
    print("0. Exit")
    print("1. Add")
    print("2. Delete")
    print("3. Display")
    choice=int(input("Enter choice: "))
    if choice==1:
        n=int(input("Enter number to append: "))
        obj.add(n)
        print("List: ",obj.dis())
    elif choice==2:
        n=int(input("Enter number to remove: "))
        obj.remove(n)
        print("List: ",obj.dis())
    elif choice==3:
        print("List: ",obj.dis())
    elif choice==0:
        print("Exiting!")
    else:
        print("Invalid choice!!")
```

```
0. Exit
1. Add
2. Delete
3. Display
Enter choice: 1
Enter number to append: 2
List:  [2]
0. Exit
1. Add
2. Delete
3. Display
Enter choice: 0
Exiting!
```

In [24]:

```python
# linked list using class
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class LinkedList:
    def __init__(self):
        self.head = None
        self.last_node = None
    def append(self, data):
        if self.last_node is None:
            self.head = Node(data)
            self.last_node = self.head
        else:
            self.last_node.next = Node(data)
            self.last_node = self.last_node.next
    def display(self):
        current = self.head
        while current is not None:
            print(current.data, end = ' ')
            current = current.next
a_llist = LinkedList()
n = int(input('How many elements would you like to add? '))
for i in range(n):
    data = int(input('Enter data item: '))
    a_llist.append(data)
print('The linked list: ', end = '')
a_llist.display()
```

```
How many elements would you like to add? 2
Enter data item: 1
Enter data item: 2
The linked list: 1 2
```

In [23]:

```
# operator overloading example program
class Vector:
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def __str__(self):
        return 'Vector (%d, %d)' % (self.a, self.b)
    def __add__(self,other):
        return Vector(self.a + other.a, self.b + other.b)
    def __sub__(self,other):
        return Vector(self.a - other.a, self.b - other.b)
    def __mul__(self,other):
        return Vector(self.a * other.a, self.b * other.b)
    def __truediv__(self,other):
        return Vector(float(self.a) /other.a, float(self.b) / other.b)
    def __floordiv__(self,other):

        return Vector(float(self.a) //other.a, float(self.b) //other.b)
v1 = Vector(5,10)
v2 = Vector(2,-2)
print (v1 + v2)
print (v1 - v2)
print (v1 * v2)
print (v1 / v2)
print (v1 // v2)
```

```
Vector (7, 8)
Vector (3, 12)
Vector (10, -20)
Vector (2, -5)
Vector (2, -5)
```

In [ ]:

```

```