In [6]:
```
#Write a program that uses class to store the marks of students.
#use list to store the marks tn three subject.
#also give total of marks

class Student:
    def __init__(self,name,roll_no):
        self.marks=[]
        self.name=name
        self.roll_no=roll_no
    def entermarks(self):
        for i in range(3):
           # m=int(input("Enter marks of %s in subject %d: "%(self.name,i+1)))
            m=int(input("Enter marks of {} in subject {}: ".format(self.name,i+1)
            self.marks.append(m)
    def total(self):
        return self.marks[0]+self.marks[1]+self.marks[2]
    def display(self):
        print("name=",self.name , ", roll number=", self.roll_no ,
              "got marks in three subjects" ,self.marks ,"\nTotal marks= ",self.t
```

In [7]:
```
s1=Student("pratham",123)
s1.entermarks()
s1.display()
```

```
Enter marks of pratham in subject 1: 66
Enter marks of pratham in subject 2: 75
Enter marks of pratham in subject 3: 89
name= pratham , roll number= 123 got marks in three subjects [66, 75, 89]
Total marks=  230
```

In [10]:
```
#write a program with class employee that keeps record of no. of employees in org
#also store thair data.
class Employee:
    empcount=0
    def __init__(self,name,age,salary):
        self.name=name
        self.age=age
        self.salary=salary
        Employee.empcount+=1
    def display_count_emp(self):
        print("There are",Employee.empcount,"employees")
    def display(self):
        print("Employee name is",self.name," , age is",self.age,"and salary is",s
```

In [12]:
```
emp1=Employee("s1",34,40000)

emp2=Employee("h1",36,50000)
emp2.display_count_emp()
print("Detail of employee-2")
emp2.display()
```

```
There are 4 employees
Detail of employee-2
Employee name is h1  , age is 36 and salary is 50000
```

In [2]:
```
#write a python program for library book record with oops.

class library:
    def __init__(self):
        self.title=""
        self.author=""
        self.publisher=""
    def read(self):
        self.title=input("Enter Book Title: ")
        self.author=input("Enter Book author: ")
        self.publisher=input("Enter Book Publisher: ")
    def display(self):
        print("Title:", self.title)
        print("Author:", self.author)
        print("Publisher:", self.publisher)
        print("\n")
my_book=[]
ch='y'
while(ch=='y'):
    print('''
1. Add New Book
2. Display Books
''')
    choice=int(input("Enter choice: "))
    if(choice==1):
        book=library()
        book.read()
        my_book.append(book)
    elif(choice==2):
        for i in my_book:
            i.display()
    else:
        print("Invalid choice!")
    ch=input("Do you want to continue..?")
print("Bye!")
```

```
1. Add New Book
2. Display Books

Enter choice: 1
Enter Book Title: b
Enter Book author: b
Enter Book Publisher: b
Do you want to continue..?y

1. Add New Book
2. Display Books

Enter choice: 2
Title: b
Author: b
Publisher: b


Do you want to continue..?y
```

```
1. Add New Book
2. Display Books

Enter choice: 1
Enter Book Title: j
Enter Book author: u
Enter Book Publisher: y
Do you want to continue..?y

1. Add New Book
2. Display Books

Enter choice: 2
Title: b
Author: b
Publisher: b


Title: j
Author: u
Publisher: y


Do you want to continue..?2
Bye!
```

In [ ]:
```
#write a python program that has class store which keeps record of code and price
#each product. Display a menu of all products to the user and prompt
#him to enter the quantity of each item required . generate a bill and display to
class Store:
    def __init__(self,n):
        self.item_code=[]
        self.price=[]
        self.n=n
    def get_data(self):
        #n=int(input("enter no of items: "))
        for i in range(self.n):
            self.item_code.append(input("enter code of item: "))
            self.price.append(int(input("enter cost of item: ")))
    def display_data(self):
        print("Item Code \t\t Price")
        for i in range(self.n):
            print(self.item_code[i],"\t\t ",self.price[i])

    def calculate_bill(self,quant):
        total_amount=0
        for i in range(self.n):
            total_amount+=(self.price[i]*quant[i])
        print("**********************Bill**********************")
        print("ITEM \t PRICE \t QUANTITY \t SUBTOTAL")
        for i in range(self.n):
            print(self.item_code[i],"\t",self.price[i],"\t",quant[i],"\t\t",
                  self.price[i]*quant[i])
        print("************************************")
        print("Total= ",total_amount)

n=int(input("enter no of items: "))
s1=Store(n)
s1.get_data()
s1.display_data()
q=[]
print("Enter quantity of each item: ")
for i in range(n):
    q.append(int(input("Enter quantity of item : " )))

s1.calculate_bill(q)
```

In [1]:
```
""" Bank Account class:
Create a Python class called BankAccount which represents a bank account,
having as attributes: accountNumber (numeric type), name
(name of the account owner as string type), balance.
Create a constructor with parameters: accountNumber, name, balance.
Create a Deposit() method which manages the deposit actions.
Create a Withdrawal() method  which manages withdrawals actions.
Create an bankFees() method to apply the bank fees with a
percentage of 5% of the balance account.
Create a display() method to display account details.
Give the complete code for the  BankAccount class.
"""

class BankAccount:
    # create the constuctor with parameters: accountNumber, name and balance
    def __init__(self,accountNumber, name, balance):
        self.accountNumber = accountNumber
        self.name = name
        self.balance = balance

    # create Deposit() method
    def Deposit(self , d ):
        self.balance = self.balance + d

    # create Withdrawal method
    def Withdrawal(self , w):
        if(self.balance < w):
            print("impossible operation! Insufficient balance !")
        else:
            self.balance = self.balance - w
    # create bankFees() method
    def bankFees(self):
        self.balance = (95/100)*self.balance

    # create display() method
    def display(self):
        print("Account Number : " , self.accountNumber)
        print("Account Name : " , self.name)
        print("Account Balance : " , self.balance , " $")

# Testing the code :
newAccount = BankAccount(2178514584, "Albert" , 2700)
# Creating Withdrawal Test
newAccount.Withdrawal(300)
# Create deposit test
newAccount.Deposit(200)
# Display account informations
newAccount.display()
```

```
Account Number :  2178514584
Account Name :  Albert
Account Balance :  2600  $
```

In [2]:
```
"""Circle class
1 - Define a Circle class allowing to create a circleC (O, r)
with center O(a, b) and radius r using the constructor:
    def __init__(self,a,b,r):
        self.a = a
        self.b = b
        self.r = r
2 - Define a Area() method of the class which calculates the area of the circle.
3 - Define a Perimeter() method of the class which allows you to
calculate the perimeter of the circle.
4 - Define a testBelongs() method of the class which allows to test whether a poi
A(x, y) belongs to the circle C(O, r) or not."""

from math import pi
class Circle:
    def __init__(self, a, b, r):
        self.a = a
        self.b = b
        self.r = r

    def perimeter (self):
        return  2 * pi * self.r


    def area (self):
        return  pi * self.r**2


    # form of the cercle equation
    def formEquation (self, x, y):
        return (x-self.a)**2 + (y-self.b)**2 - self.r**2

    # method to test if given point blong to the circle or not
    def test_belong (self, x, y):
        if (self.formEquation (x, y) == 0):
            print ("the point: (", x, y, ") belongs to the circle C")
        else:
            print ("the point: (", x, y, ") does not belong to the circle C")


# Creating of an instance object
C = Circle (1,2,1)

print ("the perimeter of the circle C is:", C.perimeter() )
print ("the area of circle C is:", C.area())
# we test if the point(1,1) belong to the circle or not
C.test_belong(1,1)
# The output is:
#the perimeter of the circle C is: 6.283185307179586
#the area of circle C is: 3.141592653589793
#the point: ( 1 1 ) belongs to the circle C
```

```
the perimeter of the circle C is: 6.283185307179586
the area of circle C is: 3.141592653589793
the point: ( 1 1 ) belongs to the circle C
```

In [3]:
```
""" Computation class:
1 - Create a Coputation class with a default constructor
(without parameters) allowing to perform
various calculations on integers numbers.
2 - Create a method called Factorial() which allows to calculate the factorial of
integer. Test the method by instantiating the class.
3 - Create a method called Sum() allowing to calculate the sum of the first n
integers 1 + 2 + 3 + .. + n. Test this method.
4 - Create a method called testPrim() in  the Calculation
class to test the primality of a given integer. Test this method.
4 - Create  a method called testPrims() allowing to test if
two numbers are prime between them.
5 - Create a tableMult() method which creates and displays
the multiplication table of a given integer.
Then create an allTablesMult() method to display all the
integer multiplication tables 1, 2, 3, ..., 9.
6 - Create a static listDiv() method that gets all
the divisors of a given integer on new list called  Ldiv.
Create another listDivPrim() method that gets all
the prime divisors of a given integer."""


class Computation:
    def __init__ (self):
        pass
# --- Factorial ------------
    def factorial(self, n):
        j = 1
        for i in range (1, n + 1):
            j = j * i
        return j

# --- Sum of the first n numbers ----
    def sum (self, n):
        j = 1
        for i in range (1, n + 1):
            j = j + i
        return j

# --- Primality test of a number ------------
    def testPrim (self, n):
        j = 0
        for i in range (1, n + 1):
            if (n% i == 0):
                j = j + 1
        if (j == 2):
            return True
        else:
            return False

# --- Primality test of two integers ------------
    def testprims (self, n, m):

        # initialize the number of commons divisors
        commonDiv = 0
        for i in range (1, n + 1):
```

```
            if (n% i == 0 and m% i == 0):
                commonDiv = commonDiv + 1
        if commonDiv == 1:
            print ("The numbers", n, "and", m, "are co-primes")
        else:
            print ("The numbers", n, "and", m, "are not co-primes")

#---Multiplication table-------------
    def tableMult (self, k):
        for i in range (1,10):
            print (i, "x", k, "=", i * k)

# --- All multiplication tables of the numbers 1, 2, .., 9
    def allTables (self):
        for k in range (1,10):
            print ("\nthe multiplication table of:", k, "is:")
            for i in range (1,10):
                print (i, "x", k, "=", i * k)

# ----- list of divisors of an integer
    def listDiv (self, n):
        # initialization of the list of divisors
        lDiv = []
        for i in range (1, n + 1):
            if (n% i == 0):
                lDiv.append (i)
        return lDiv

# ------ list of prime divisors of an integer ----------------
    def listDivPrim (self, n):
        # initialization of the list of divisors
        lDiv = []
        for i in range (1, n + 1):
            if (n% i == 0 and self.testPrim (i)):
                lDiv.append (i)
        return lDiv

# Instantiation example
Comput= Computation ()
Comput.testprims (13, 7)
print ("List of divisors of 18:", Comput.listDiv (18))
print ("List of prime divisors of 18:", Comput.listDivPrim (18))
Comput.allTables ()
```

```
The numbers 13 and 7 are co-primes
List of divisors of 18: [1, 2, 3, 6, 9, 18]
List of prime divisors of 18: [2, 3]

the multiplication table of: 1 is:
1 x 1 = 1
2 x 1 = 2
3 x 1 = 3
4 x 1 = 4
5 x 1 = 5
6 x 1 = 6
7 x 1 = 7
8 x 1 = 8
9 x 1 = 9
```

```
the multiplication table of: 2 is:
1 x 2 = 2
2 x 2 = 4
3 x 2 = 6
4 x 2 = 8
5 x 2 = 10
6 x 2 = 12
7 x 2 = 14
8 x 2 = 16
9 x 2 = 18

the multiplication table of: 3 is:
1 x 3 = 3
2 x 3 = 6
3 x 3 = 9
4 x 3 = 12
5 x 3 = 15
6 x 3 = 18
7 x 3 = 21
8 x 3 = 24
9 x 3 = 27

the multiplication table of: 4 is:
1 x 4 = 4
2 x 4 = 8
3 x 4 = 12
4 x 4 = 16
5 x 4 = 20
6 x 4 = 24
7 x 4 = 28
8 x 4 = 32
9 x 4 = 36

the multiplication table of: 5 is:
1 x 5 = 5
2 x 5 = 10
3 x 5 = 15
4 x 5 = 20
5 x 5 = 25
6 x 5 = 30
7 x 5 = 35
8 x 5 = 40
9 x 5 = 45

the multiplication table of: 6 is:
1 x 6 = 6
2 x 6 = 12
3 x 6 = 18
4 x 6 = 24
5 x 6 = 30
6 x 6 = 36
7 x 6 = 42
8 x 6 = 48
9 x 6 = 54

the multiplication table of: 7 is:
```

```
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63

the multiplication table of: 8 is:
1 x 8 = 8
2 x 8 = 16
3 x 8 = 24
4 x 8 = 32
5 x 8 = 40
6 x 8 = 48
7 x 8 = 56
8 x 8 = 64
9 x 8 = 72

the multiplication table of: 9 is:
1 x 9 = 9
2 x 9 = 18
3 x 9 = 27
4 x 9 = 36
5 x 9 = 45
6 x 9 = 54
7 x 9 = 63
8 x 9 = 72
9 x 9 = 81
```

In [ ]:

```
"""1. Define a Book class with the following attributes: Title,
Author (Full name), Price.
2. Define a constructor used to initialize the attributes of
the method with values entered by the user.
3. Set the View() method to display information for the current book.
4. Write a program to testing the Book class.
Solution
"""
class Book:
     # Question 2
     def __init__(self , Title , Author , Price):
          self.Title    =  Title
          self.Author   =  Author
          self.Price    =  Price

     # Question 3
     def view(self ):
          return ("Book Title: " , self.Title ,  "Book Author: " ,
                  self.Author, "Book Price: " , self.Price)

# Question 4
MyBook = Book("Python Crash Course" , "Eric Matthes" , "23 $")
print( MyBook.view())
# The output: ('Book Title: ', 'Python Crash Course', 'Book Author: ',
'Eric Matthes', 'Book Price: ', '23 $')
```

In [16]:
```
"""size: return the number of rows, and number of columns
get_cell: that take the number of rows, the number of columns as parameters,
and returns the content of cell corresponding to row number col number
set_cell: that take the number of rows, the number of columns as parameters,
and a value and set the value val in cell specified by row number x column number
to_str: return a string representation of the matrix
mult: that take a scalar and return a new matrix which is the scalar product of m

class Matrix(object):

    def __init__(self, row, col, val=2):
        self._row = row
        self._col = col
        self._matrix = []
        for i in range(row):
            c = [val] * col
            self._matrix.append(c)

    def size(self):
        return self._row, self._col

    def get_cell(self, row, col):
        self._check_index(row, col)
        return self._matrix[i][j]

    def matrix_set(self, row, col, val):
        self._check_index(row, col)
        self._matrix[row][col] = val

    def __str__(self):
        s = ''
        for i in range(self._row):
            s += self._matrix[i]
            s += '\n'
        return s

    def _check_index(self, row, col):
        if not (0 < row <= self._row) or not (0 < col <= self._col):
            raise IndexError("matrix index out of range")

m1=Matrix(3,3)
m1.size()
#m1.get_cell(2,1)
m1.matrix_set(2,2,2)
m1.get_cell(2,2)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-16-8fc813117857> in <module>
     43 #m1.get_cell(2,1)
     44 m1.matrix_set(2,2,2)
---> 45 m1.get_cell(2,2)

<ipython-input-16-8fc813117857> in get_cell(self, row, col)
     22     def get_cell(self, row, col):
     23         self._check_index(row, col)
---> 24         return self._matrix[i][j]
```

```
25
26     def matrix_set(self, row, col, val):

NameError: name 'j' is not defined
```

In [1]:
```
class flashcard:
    def __init__(self, word, meaning):
        self.word = word
        self.meaning = meaning
    def __str__(self):

        #we will return a string
        return self.word+' ( '+self.meaning+' )'

flash = []
print("welcome to flashcard application")

#the following loop will be repeated until
#user stops to add the flashcards
while(True):
    word = input("enter the name you want to add to flashcard : ")
    meaning = input("enter the meaning of the word : ")

    flash.append(flashcard(word, meaning))
    option = int(input("enter 0 , if you want to add another flashcard : "))

    if(option):
        break

# printing all the flashcards
print("\nYour flashcards")
for i in flash:
    print(">", i)
```

```
welcome to flashcard application
enter the name you want to add to flashcard : fcsp
enter the meaning of the word : fundamental of computer science python
enter 0 , if you want to add another flashcard : 1

Your flashcards
> fcsp ( fundamental of computer science python )
```

In [5]:
```
# Create class Student
class Student:
    def __init__(self, name, roll, s1, s2):
        self.name = name
        self.roll = roll
        self.s1 = s1
        self.s2 = s2
    # Function to create and append students
    def accept(self, Name, Roll, score1, score2):
        obj = Student(Name, Roll, score1, score2)
        ls.append(obj)
    # Display student details
    def display(self, obj):
        print("Name : ", obj.name)
        print("RollNo : ", obj.roll)
        print("Score1 : ", obj.s1)
        print("Score2 : ", obj.s2)
        print("\n")
    # Search Function
    def search(self, rn):
        for i in range(ls.__len__()):
            if (ls[i].roll == rn):
                return i
            # Deletion
    def delete(self, rn):
        i = obj1.search(rn)
        del ls[i]
    # Update Function
    def update(self, rn, No):
        i = obj1.search(rn)
        rolln = No
        ls[i].roll = rolln;
ls = []
# Object of class
obj1 = Student('', 0, 0, 0)
print("\nOperations used, ")
print("\n1.Accept Student details\n"
      "2.Display Student Details\n"
       "3.Search Details of a Student\n"
        "4.Delete Details of Student"
      "\n5.Update Student Details\n6.Exit")
obj1.accept("A", 1, 100, 100)
obj1.accept("B", 2, 90, 90)
obj1.accept("C", 3, 80, 80)
print("\n")
print("\nList of Students\n")
for i in range(ls.__len__()):
    obj1.display(ls[i])
print("\n Student Found, ")
s = obj1.search(2)
obj1.display(ls[s])
obj1.delete(2)
print(ls.__len__())
print("List after deletion")
for i in range(ls.__len__()):
    obj1.display(ls[i])
```

```
obj1.update(3, 2)
print(ls.__len__())
print("List after updation")
for i in range(ls.__len__()):
    obj1.display(ls[i])
print("Thank You !")
```

```
Operations used,

1.Accept Student details
2.Display Student Details
3.Search Details of a Student
4.Delete Details of Student
5.Update Student Details
6.Exit



List of Students

Name :  A
RollNo :  1
Score1 :  100
Score2 :  100


Name :  B
RollNo :  2
Score1 :  90
Score2 :  90


Name :  C
RollNo :  3
Score1 :  80
Score2 :  80



 Student Found,
Name :  B
RollNo :  2
Score1 :  90
Score2 :  90


2
List after deletion
Name :  A
RollNo :  1
Score1 :  100
Score2 :  100


Name :  C
RollNo :  3
Score1 :  80
```

```
Score2 :  80


2
List after updation
Name :  A
RollNo :  1
Score1 :  100
Score2 :  100


Name :  C
RollNo :  2
Score1 :  80
Score2 :  80


Thank You !
```

In [ ]: