LAB 01

***#Q1***

def is\_multiply(m,n):

for i in range(n,m):

if m%n==0:

return True

else:

return False

a=is\_multiply(6,2)

print(a)

***OUTPUT:***

******

***#Q2***

def isEven(n) :

return (n &amp; 1);

if(isEven(33) == 0):

print(&quot;even&quot;)

else :

print(&quot;odd&quot;)

print(isEven(8))

***OUTPUT:***



***#Q3***

def EvenList(n):

num=[]

for i in range(n+1):

if i%2==0:

num.append(i)

print(num)

EvenList(10)

***OUTPUT:***



***#Q4***

def minmax(data):

smallest = data[0]

largest = data[0]

for i in data:

if i > smallest:

smallest=i

elif i < largest:

largest = i

return smallest,largest

a=minmax([-278,898,7,2])

print(a)

***OUTPUT:***



***#Q5***

def sumsquares(n):

lst=[]

for i in range(n):

if i%2==0:

a=i\*\*2

lst.append(a)

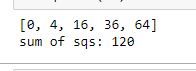
print(lst)

summ=lst[0]+lst[1]+lst[2]+lst[3]+lst[4]

print("sum of sqs:",summ)

sumsquares(10)

***OUTPUT:***

******

***#Q6***

def sumsquares(n):

lst=[]

for i in range(n):

if i%2!=0:

a=i\*\*2

lst.append(a)

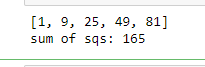
print(lst)

summ=lst[0]+lst[1]+lst[2]+lst[3]+lst[4]

print("sum of sqs:",summ)

sumsquares(10)

***OUTPUT:***

**

***#Q7***

def distinctoddpairgen(array):

for i in range(len(array)):

for j in range(len(array)):

if i != j:

product = array[i] \* array[j]

if product & 1:

return "Yes there is a distinct set of numbers whose product is odd"

return "No there is no distinct set of numbers whose product is odd"

print(distinctoddpairgen([1, 3, 5, 7, 9]))

***OUTPUT:***



***#Q8***

def Reverse(list):

a=list[::-1]

print(a)

Reverse([88,2,34,90,178])

***OUTPUT:***



***#Q9***

def Unique(lst):

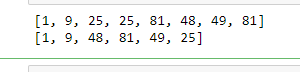
print(lst)

a=set(lst)

print(list(a))

Unique([1, 9, 25,25,81,48, 49, 81])

***OUTPUT:***

******

***#Q10***

def usernumber(list):

a=[]

for i in list:

if i%2==0:

a.append(i)

print("LAST ELEMENT=",list[-1])

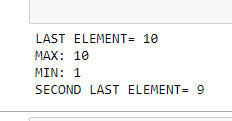
print("MAX:",max(list))

print("MIN:",min(list))

print("SECOND LAST ELEMENT=",list[-2])

usernumber([1,2,3,4,5,6,7,8,9,10])

***OUTPUT:***



***#Q11***

def showexcitement(lst):

a=lst.split()

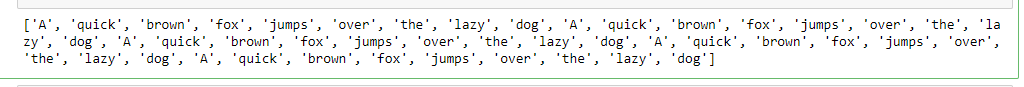
b=a\*5

return b

e=showexcitement('A quick brown fox jumps over the lazy dog')

print(e)

***OUTPUT:***

******

***#Q12***

def Greater(n1,n2,n3):

if (n1>n2 and n1>n3):

return n1

elif(n2>n1 and n2>n3):

return n2

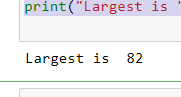
else:

return n3

a = Greater(56,82,24)

print("Largest is ",a)

***OUTPUT:***



***#Q13***

def Divide(dividend,divisor):

ans = dividend//divisor

res = dividend % divisor

return ans , res

a = Divide(10,3)

print(a)

***OUTPUT:***



***#Q14***

class Person:

def \_\_init\_\_(self,name,age):

self.name = name

self.age = age

def birthday(self):

return self.age + 1

p1 = Person("Durraiya",19)

print(p1.birthday())

***OUTPUT:***



### Home Task:

***Task 1.1***

class counter:

def \_\_init\_\_(self):

self.num = 0

def increment(self):

self.num = self.num + 1

self.show()

def reset(self):

self.num = 0

self.show()

def show(self):

print(self.num)

obj = counter()

obj.show()

obj.increment()

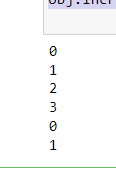
obj.increment()

obj.increment()

obj.reset()

obj.increment()

***OUTPUT:***



***Task 1.2***

class Bag:

def \_\_init\_\_(self):

self.item = set()

def lenght(self):

return len(self.item)

def add(self, item):

self.item.add(item)

def getitem(self):

return self.item.pop()

def check(self,item):

return item in self.item

def iter(self):

for i in self.item:

print(i)

obj = Bag()

obj.lenght()

obj.add("item1")

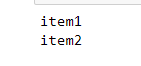
obj.add("item2")

obj.iter()

obj.check("item2")

obj.getitem()

***OUTPUT:***



***Task 1.3***

class Bag:

def \_\_init\_\_(self):

self.item = []

self.num = 0

def lenght(self):

return len(self.item)

def add(self, item):

self.item.append(item)

def remove(self,item):

return self.item.remove(item)

def check(self,item):

return item in self.item

def iter(self):

obj = iteration(self.item)

obj.looping()

def numofitem(self,item):

for i in self.item:

if item == i:

self.num +=1

return self.num

class iteration:

def \_\_init\_\_(self,lst):

self.lst = lst

self.index = 0

def looping(self):

while self.index< len(self.lst):

print(self.lst[self.index])

self.index +=1

obj = Bag()

obj.lenght()

obj.add("item1")

obj.add("item2")

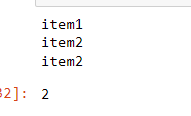
obj.add("item2")

obj.iter()

obj.check("item2")

obj.numofitem("item2")

***OUTPUT:***

******

***Task 1.4:***

class filereader:

def \_\_init\_\_(self):

self.name = "C:/Users/b/Desktop/studentrecord.txt"

def open(self):

self.file = open(self.name, "r")

def close(self):

self.file.close()

def fetch(self):

for i in range(3):

self.records()

def records(self):

self.read = self.file.readline()

if self.read == "":

return None

data=self.read.split(",")

student = Studentfilerecord()

student.idNum = int(data[0])

student.firstName = data[1]

student.lastName = data[2]

student.classCode = int(data[3])

student.gpa = float(data[4])

print("\n")

print("ID number: ",int(data[0]))

print("First name: ",data[1])

print("Last name: ",data[2])

print("Class code: ",int(data[3]))

print("GPA: ",float(data[4]))

class Studentfilerecord:

def \_\_init\_\_(self):

self.idNum = 0

self.firstName = None

self.lastName = None

self.classCode = 0

self.gpa = 0.0

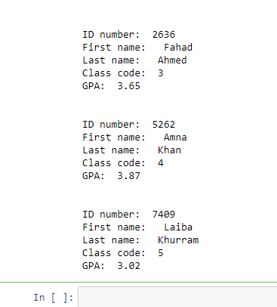
obj = filereader()

obj.open()

obj.fetch()

obj.close()

***Output:***

******

***Task 1.5:***

class filereader:

def \_\_init\_\_(self):

self.name = "C:/Users/b/Desktop/studentrecord.txt"

def open(self):

self.file = open(self.name, "r")

self.outputfile = open("outputrecords.txt","w")

def close(self):

self.file.close()

self.outputfile.close()

def fetch(self):

for i in range(3):

self.records()

def records(self):

self.read = self.file.readline()

if self.read == "":

return None

data=self.read.split(",")

student = Studentfilerecord()

student.idNum = int(data[0])

student.firstName = data[1]

student.lastName = data[2]

student.classCode = int(data[3])

student.gpa = float(data[4])

print("\n")

print("ID number: ",int(data[0]))

print("First name: ",data[1])

print("Last name: ",data[2])

print("Class code: ",int(data[3]))

print("GPA: ",float(data[4]))

self.outputfile.writelines(data)

class Studentfilerecord:

def \_\_init\_\_(self):

self.idNum = 0

self.firstName = None

self.lastName = None

self.classCode = 0

self.gpa = 0.0

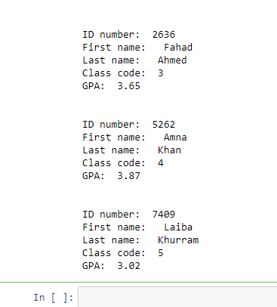
obj = filereader()

obj.open()

obj.fetch()

obj.close()

***Output:***

******

***Task 1.6***

from math import sqrt

class lineSegment:

def \_\_init\_\_(self,ptA,ptB):

self.point1 = ptA

self.point2 = ptB

def endPointA(self):

eA=("x1=",self.point1[0],"y1=",self.point1[1])

return eA

def endPointB(self):

eB=("x2=",self.point2[0],"y2=",self.point2[1])

return eB

def \_\_str\_\_(self):

return "({0} , {1}) , ({2} , {3})".format(self.point1[0],self.point1[1],self.point2[0],self.point2[1])

def lenght(self):

return sqrt(((self.point1[0]-self.point2[0])\*2)+((self.point1[1]-self.point2[1])\*2))

def isVartical(self):

if self.point1[0]==self.point2[0]:

return ("The line is vertical")

else:

return ("The line is not vertical")

def isHorizontal(self):

if self.point1[1]==self.point2[1]:

return ("The line is Horizontal")

else:

return ("The line is not Horizontal")

def midpoint(self):

return ((self.point1[0]+self.point2[0])//2,(self.point1[1]+self.point2[1])//2)

def slope(self):

return (self.point2[1]-self.point1[1])/(self.point2[0]-self.point1[0])

def isParallel(self,other):

if self.slope()==other.slope():

return True

else:

return False

def isPerpendicular(self,other):

if self.slope()==-1/other.slope():

return True

else:

return False

def bisects(self,other):

if self.isParallel(other):

return False

else:

return True

def intersection(self,other):

if self.bisects(other):

return True

else:

return False

def shift(self,x,y):

self.point1[0]+=x

self.point1[1]+=y

self.point2[0]+=x

self.point2[1]+=y

return (self.point1,self.point2)

obj1 = lineSegment([1,2],[11,5])

obj2 = lineSegment([1,2],[5,5])

print(obj1)

print(obj1.isVartical())

print(obj1.isParallel(obj2))

print(obj1.isPerpendicular(obj2))

print(obj1.intersection(obj2))

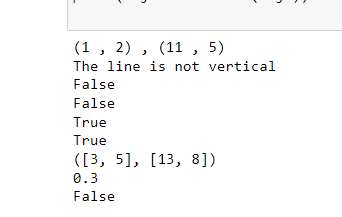
print(obj1.bisects(obj2))

print(obj1.shift(2,3))

print(obj1.slope())

print(obj1.isParallel(obj2))

***OUTPUT:***

******

LAB 02

***#Q1***

import random

import ctypes

class array:

def \_\_init\_\_(self,size):

self.size = size

self.elements = (ctypes.py\_object \* self.size)(None)

def \_\_len\_\_(self):

return self.size

def \_\_getitem\_\_(self,index):

return "{:.2f}". format(self.elements[index])

def \_\_setitem\_\_(self,index,value):

self.elements[index]=value

def \_\_clear\_\_(self,value):

for i in range(len(self.elements)):

self.elements[i]=value

def \_\_iter\_\_(self):

return Arrayiter(self.elements)

class Arrayiter:

def \_\_init\_\_(self,element):

self.element = element

self.currind= 0

def \_\_next\_\_(self):

if self.currind < len(self.element):

entry = self.element[self.currind]

self.currind+=1

return entry

else:

raise StopIteration

a1=array(7)

print("my array length is",a1.\_\_len\_\_())

for i in range(a1.\_\_len\_\_()):

item = random.random()

a1.\_\_setitem\_\_(i,item)

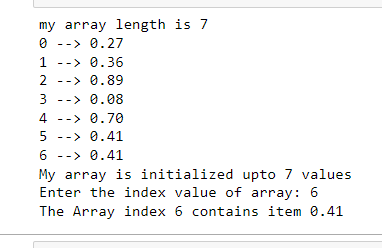
print("{} --> {:.2f}". format(i,item))

print("My array is initialized upto", a1.\_\_len\_\_() , "values")

indval = int(input("Enter the index value of array: "))

print("The Array index",indval,"contains item" ,a1.\_\_getitem\_\_(indval))

***OUTPUT:***



***#Q2***

class Array2D:

def \_\_init\_\_(self,row,column):

self.row=row

self.column=column

self.elements = (ctypes.py\_object\* (self.row\*self.column))(0)

def setvalues(self,i,j,v):

index = i \* self.row + j

self.elements[index]=v

def getvalue(self,i,j):

index = i \* self.row + j

return self.elements[index]

def printvalues(self):

row = ""

for i in range(self.row):

for j in range(self.column):

item = self.getvalue(i,j)

row = row + " " +str(item)

row = row + "\n"

print("The 2D-array is:" )

print(row)

def subvalues(self,a1,a2):

a3 = Array2D(self.row,self.column)

for i in range (self.row):

for j in range(self.column):

item1= self.getvalue(i,j)

item2 = self.getvalue(i,j)

item3=item2-item1

a3.setvalues(i,j,item3)

return a3

def Multvalues(self,a1,a2):

a3 = Array2D(self.row,self.column)

for i in range (self.row):

for j in range(self.column):

item1= self.getvalue(i,j)

item2 = self.getvalue(i,j)

item3=item2\*item1

a3.setvalues(i,j,item3)

return a3

def Transpose(self):

array = Array2D(self.column,self.row)

for i in range(self.row):

for j in range(self.column):

item = self.getvalue(i,j)

array.setvalues(j,i,item)

return array

a1 = Array2D(3,3)

a2= Array2D(3,3)

a1.setvalues(0,0,1)

a1.setvalues(0,1,4)

a1.setvalues(0,2,9)

a1.setvalues(1,0,6)

a1.setvalues(1,1,0)

a1.setvalues(1,2,2)

a1.setvalues(2,0,3)

a1.setvalues(2,1,5)

a1.setvalues(2,2,6)

a2.setvalues(0,0,7)

a2.setvalues(0,1,6)

a2.setvalues(0,2,4)

a2.setvalues(1,0,3)

a2.setvalues(1,1,9)

a2.setvalues(1,2,7)

a2.setvalues(2,0,2)

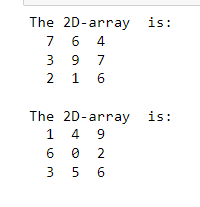
a2.setvalues(2,1,1)

a2.setvalues(2,2,6)

a2.printvalues()

a1.printvalues()

***OUTPUT:***



***#Q3***

import numpy as np

array1 = np.array([[1,2,3,4],[5,6,7,8]], dtype=np.int64)

print(array1)

x = np.ones((3,4),dtype=np.int64)

print(x)

y = np.zeros((2,3,4),dtype=np.int16)

print(y)

array2 = np.random.random((2,2))

print(array2)

array3 = np.full((3,3),7)

print(array3)

array4 = np.identity(3,dtype=np.int64)

print(array4)

add = np.add(x,y)

print(add)

diff = np.subtract(x,y)

print(diff)

mult = np.multiply(x,y)

print(mult)

div = np.divide(y,x)

print(div)

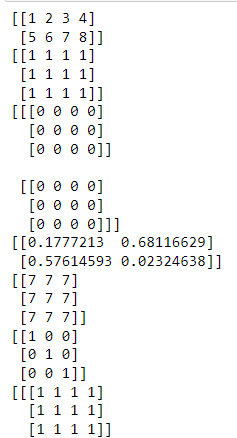
rem = np.remainder(y,x)

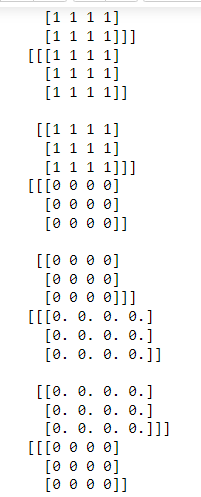
print(rem)

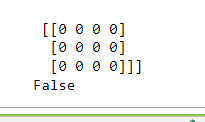
result = np.array\_equal(x,y)

print(result)

***OUTPUT:***







LAB 03

***#Q1***

class Set:

def \_\_init\_\_(self):

self.element=list()

def \_\_len\_\_(self):

return len(self.element)

def \_\_contains\_\_(self,value):

if value in self.element:

return True

else:

return False

def add(self,value):

if value not in self.element:

self.element.append(value)

def \_\_remove\_\_(self,value):

if value in self.element:

self.element.remove(value)

else:

raise Exception("The value you entered is not in the list")

def \_\_equal\_\_(self,setB):

if len(self.element) == len(setB) and setB in self.element:

print("yes,both are equal")

else:

print("both are not equal")

def \_\_issubsetof\_\_(self,setB):

for i in self.element:

for j in setB:

if i==j:

return True

else:

return False

def intersect(self,setB):

setD=[value for value in self.element if value in setB]

return setD

def diff(self,setB):

setE=[value for value in self.element if value not in setB]

return setE

def \_\_union\_\_(self,secB):

newvar=sorted(self.element + secB)

return newvar

def \_\_iter\_\_( self ):

return iter( self.element )

s1 = Set()

s1.add( "CS-112" )

s1.add( "Calculus-121" )

s1.add( "English-340" )

s1.add( "Chinese-101" )

a2 = Set()

a2.add( "Physics-101" )

a2.add( "DSA-230" )

a2.add( "CS-112" )

a2.add( "Chinese-101" )

print("Person 1 set:")

print()

for subs in s1:

print(subs)

print()

print("Person 2 Set:")

for subj in a2:

print(subj)

if s1 == a2 :

print( "Person 1 and Person 2 are taking the same courses." )

else:

same\_Courses = s1.intersect(a2)

print()

print("Intersection: ")

for course in same\_Courses :

print( course )

print()

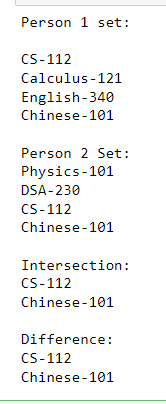
print("Difference: ")

uniqueCourses = s1.diff(a2)

for course\_names in same\_Courses :

print(course\_names)

***OUTPUT:***

****

***#Q2***

class Map:

def \_\_init\_\_(self):

self.entryList=list()

def \_\_len\_\_(self):

return len(self.entryList)

def \_\_contains\_\_(self, key):

ndx = self.findPosition(key)

return ndx is not None

def add(self, key, value):

ndx = self.findPosition(key)

if ndx is not None:

self.entryList[ndx].value = value

return False

else:

entry = \_MapEntry(key,value)

self.entryList.append(entry)

return True

def valueOf(self, key):

ndx = self.findPosition(key)

assert ndx is not None, "Invalid map key"

return self.entryList[ndx].value

def remove(self, key):

ndx = self.fndPosition(key)

assert ndx is not None, "Invalid map key."

self.entryList.pop(ndx)

def \_\_iter\_\_(self):

return iter(self.entryList)

def findPosition(self, key):

for i in range(len(self)):

if self.entryList[i].key == key:

return i

return None

#private class

class \_MapEntry:

def \_\_init\_\_(self, key, value):

self.key = key

self.value = value

b = Map()

b.add(1, 1)

b.add(2, 2)

b.add(3, 3)

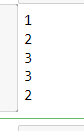
for x in iter(b):

print(x.key)

print(len(b))

print(b.valueOf(2))

***OUTPUT:***

******

***#Q3***

import ctypes

class Array:

def \_\_init\_\_(self, size):

assert size > 0

self.\_size = size

PyArrayType = ctypes.py\_object \* size

self.\_elements = PyArrayType()

self.clear(None)

def \_\_len\_\_(self):

return self.\_size

def \_\_getitem\_\_(self, index):

assert index >= 0 and index < len(self), "Array subscript out of range"

return self.\_elements[index]

def \_\_setitem\_\_(self, index, value):

assert index >= 0 and index < len(self), "Array subscript out of range"

self.\_elements[index] = value

def clear(self, value):

for i in range(len(self)):

self.\_elements[i] = value

def \_\_iter\_\_(self):

return \_ArrayIterator(self.\_elements)

class \_ArrayIterator:

def \_\_init\_\_(self, theArray):

self.\_arrayRef = theArray

self.\_curNdx = 0

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

if self.\_curNdx < len(self.\_arrayRef):

entry = self.\_arrayRef[self.\_curNdx]

self.\_curNdx += 1

return entry

else:

raise StopIteration

class MultiArray :

def \_\_init\_\_( self, \*dimensions ):

assert len(dimensions) > 1, "The array must have 2 or more dimensions."

self.\_dims = dimensions

size = 1

for d in dimensions :

assert d > 0, "Dimensions must be > 0."

size \*= d

self.\_elements = Array( size )

self.\_factors = Array( len(dimensions) )

self.\_computeFactors()

def numDims( self ):

return len(self.\_dims)

def length( self, dim ):

assert dim >= 1 and dim < len(self.\_dims),"Dimension component out of range."

return self.\_dims[dim - 1]

def clear( self, value ):

self.\_elements.clear( value )

def \_\_getitem\_\_( self, ndxTuple ):

assert len(ndxTuple) == self.numDims(), "Invalid # of array subscripts."

index = self.\_computeIndex( ndxTuple )

assert index is not None, "Array subscript out of range."

return self.\_elements[index]

def \_\_setitem\_\_( self, ndxTuple, value ):

assert len(ndxTuple) == self.numDims(), "Invalid # of array subscripts."

index = self.\_computeIndex( ndxTuple )

assert index is not None, "Array subscript out of range."

self.\_elements[index] = value

def \_computeIndex( self, idx ):

offset = 0

for j in range( len(idx) ):

if idx[j] < 0 or idx[j] >= self.\_dims[j] :

return None

else :

offset += idx[j] \* self.\_factors[j]

return offset

def \_computeFactors( self ):

max\_idx = len(self.\_factors) - 1

self.\_factors[max\_idx] = 1

for i in range(max\_idx, 0 , -1):

self.\_factors[i - 1] = self.\_dims[i] \* self.\_factors[i]

m= MultiArray(3,3,3)

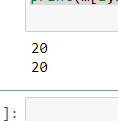
m.clear(10)

m[1,2,1] = 20

print(m[1,2,1])

print(m[1,2,1])

***OUTPUT:***



***Additional Tasks:***

***Task1(a):***

class Set:

def \_\_init\_\_(self, \*initElements):

if initElements != None:

self.\_theElements = list(initElements)

else:

self.\_theElements = list()

def \_\_len\_\_(self):

return len(self.\_theElements)

def \_\_contains\_\_(self, element):

return element in self.\_theElements

def add(self, element):

if element not in self:

self.\_theElements.append(element)

def remove (self, element):

assert element in self, "The element must be in the set."

self.\_theElements.remove(element)

def \_\_eq\_\_(self, setB):

if len(self) != len(setB):

return False

else:

return self.isSubsetOf(setB)

def isSubsetOf(self, setB):

for element in self.\_theElements:

if element not in setB:

return False

return True

def union(self, setB):

newSet = Set()

newSet.\_theElements.extend(self.\_theElements)

for element in setB:

if element not in self:

newSet.\_theElements.append(element)

return newSet

def intersection(self, setB):

newSet = Set()

for element in setB:

if element in self:

newSet.add(element)

return newSet

def difference(self, setB):

newSet = Set()

for element in setB:

if element not in self:

newSet.add(element)

return newSet

def \_\_iter\_\_(self):

return \_SetIterator(self.\_theElements)

class \_SetIterator:

def \_\_init\_\_( self, theList ):

self.\_setItems = theList

self.\_curNdx = 0

def \_\_iter\_\_( self ):

return self

def next( self ):

if self.\_curNdx < len( self.\_setItems ):

item = self.\_setItems[ self.\_curNdx ]

self.\_curNdx += 1

return item

else:

raise StopIteration

obj = Set(1,2,3,5,6,8)

print(obj.\_\_len\_\_())

***OUTPUT:***

****

***Task1(b):***

class Set:

def \_\_init\_\_(self, \*initElements):

if initElements != None:

self.\_theElements = list(initElements)

else:

self.\_theElements = list()

def \_\_len\_\_(self):

return len(self.\_theElements)

def \_\_contains\_\_(self, element):

return element in self.\_theElements

def add(self, element):

if element not in self:

self.\_theElements.append(element)

def remove (self, element):

assert element in self, "The element must be in the set."

self.\_theElements.remove(element)

def \_\_eq\_\_(self, setB):

if len(self) != len(setB):

return False

else:

return self.isSubsetOf(setB)

def isSubsetOf(self, setB):

for element in self.\_theElements:

if element not in setB:

return False

return True

def union(self, setB):

newSet = Set()

newSet.\_theElements.extend(self.\_theElements)

for element in setB:

if element not in self:

newSet.\_theElements.append(element)

return newSet

def intersection(self, setB):

newSet = Set()

for element in setB:

if element in self:

newSet.add(element)

return newSet

def difference(self, setB):

newSet = Set()

for element in setB:

if element not in self:

newSet.add(element)

return newSet

def propersubset(self,setB):

print(self.\_theElements<setB )

def \_\_iter\_\_(self):

return \_SetIterator(self.\_theElements)

class \_SetIterator:

def \_\_init\_\_( self, theList ):

self.\_setItems = theList

self.\_curNdx = 0

def \_\_iter\_\_( self ):

return self

def next( self ):

if self.\_curNdx < len( self.\_setItems ):

item = self.\_setItems[ self.\_curNdx ]

self.\_curNdx += 1

return item

else:

raise StopIteration

A = Set(1, 2, 3)

A.propersubset([1, 2, 3 , 4 ,5])

***OUTPUT:***



***Home Task 1:***

import ctypes

class Array:

# Creates an array with size elements.

def \_\_init\_\_( self, size ):

assert size > 0, "Array size must be > 0"

self.\_size = size

# Create the array structure using the ctypes module.

PyArrayType = ctypes.py\_object \* size

self.\_elements = PyArrayType()

# Initialize each element

self.clear( None )

# Returns the size of the array

def \_\_len\_\_( self ):

return self.\_size

# Gets the contents of the index element

def \_\_getitem\_\_( self, index ):

assert index >= 0 and index < len(self), "Array subscript out of range"

return self.\_elements[ index ]

# Puts the value in the array element at index position

def \_\_setitem\_\_( self, index, value):

assert index >= 0 and index < len(self), "Array subscript out of range"

self.\_elements[ index ] = value

# Clears the array by setting each element to the given value

def clear( self, value ):

for i in range( len(self) ):

self.\_elements[i] = value

# Returns the array's iterator for traversing the elements

def \_\_iter\_\_( self ):

return \_ArrayIterator( self.\_elements )

class \_ArrayIterator:

def \_\_init\_\_( self, theArray):

self.\_arrayRef = theArray

self.\_curNdx = 0

def \_\_iter\_\_( self ):

return self

def next( self ):

if self.\_curNdx < len( self.\_arrayRef ):

entry = self.\_arrayRef[ self.\_curNdx]

self.\_curNdx += 1

return entry

else:

raise StopIteration

class MultiArray:

def \_\_init\_\_( self, \*dimensions ):

assert len(dimensions) > 1, "The array must have 2 or more dimensions."

self.\_dims = dimensions

size = 1

for d in dimensions:

assert d > 0, " Dimensions must be > 0."

size \*= d

self.\_elements = Array( size )

# Create the 1-D array to store the equation factors

self.\_factors = Array( len(dimensions) )

self.\_computeFactors()

# Returns the number of dimensions in the array.

def numDims( self ):

return len(self.\_dims)

# Returns the length of the given dimension.

def length( self, dim ):

assert dim >= 1 and dim <= len(self.\_dims), \

"Dimension component out of range."

return self.\_dims[dim-1]

def clear( self, value ):

self.\_elements.clear( value )

def \_\_getitem\_\_( self, ndxTuple ):

assert len(ndxTuple) == self.numDims(), "Invalid # of array subscripts."

index = self.\_computeIndex(ndxTuple)

assert index is not None, "Array subscript out of range."

return self.\_elements[index]

def \_\_setitem\_\_( self, ndxTuple, value ):

assert len(ndxTuple) == self.numDims(), "Invalid # of array subscripts."

index = self.\_computeIndex( ndxTuple )

assert index is not None, "Array subscript out of range."

self.\_elements[index] = value

def \_computeIndex( self, idx ):

offset = 0

for j in range( len(idx) ):

if idx[j] < 0 | idx[j] >= self.dims[j] :

return None

else:

offset += idx[j] \* self.\_factors[j]

return offset

def \_computeFactors( self ):

for j in range( self.numDims()-1 ):

self.\_factors[j] = 1

for k in range( j+1, self.numDims() ):

self.\_factors[j] \*= self.\_dims[k]

self.\_factors[self.numDims()-1] = 1

class ReportGenerator:

def \_\_init\_\_(self,filename):

self.\_filename = filename

self.\_inputFO = None#file object

self.\_multiArray = MultiArray(5,20,12)

def open(self):

self.\_inputFO = open(self.\_filename, "r")

def close(self):

self.\_inputFO.close()

self.\_inputFO = None

def load\_data(self):

self.open()

for line in self.\_inputFO:

words = line.split()

if len(words) == 2:

item\_counter = 0

f\_index = int(words[1][1])-1#store index

if len(words) == 13:

if words[0]== "Item#":

pass

else:

for i in range(12):

self.\_multiArray[f\_index, item\_counter, i]=float(words[i+1])

item\_counter +=1

self.close()

return self.\_multiArray

rg = ReportGenerator('SalesData.txt')

salesData = rg.load\_data()

def main():

salesData = MultiArray(8, 100, 12)

print("Continue to add things")

def totalSalesByStore(salesData, store):

s = store - 1

total = 0.0

for i in range(salesData.length(2)):

for m in range(salesData.length(3)):

total += salesData[s, i, m]

return total

def totalSalesByMonth(salesData, month):

m = month - 1

total = 0.0

for s in range(salesData.length(1)):

for i in range(salesData.length(2)):

total += salesData[s, i, m]

return total

def totalSalesByItem(salesData, item):

i = item - 1

total = 0.0

for s in range(salesData.length(1)):

for m in range(salesData.length(3)):

total += salesData[s, i, m]

return total

def totalSalesPerMonth(salesData, store):

s = store - 1

totals = Array(12)

for m in range(salesData.length(3)):

\_sum = 0.0

for i in range(salesData.length(2)):

\_sum += salesData[s, i, m]

totals[m] = \_sum

return totals

print( totalSalesByStore(salesData, 1))

print( totalSalesByItem(salesData, 2))

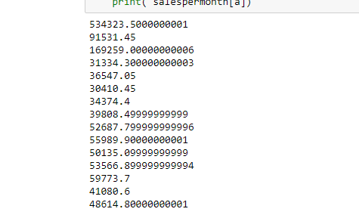
print( totalSalesByMonth(salesData, 4))

salespermonth = totalSalesPerMonth(salesData, 1)

for a in range(len(salespermonth)):

print( salespermonth[a])

***Output:***

******

***Task 2:***

import ctypes

class Array:

# Creates an array with size elements.

def \_\_init\_\_( self, size ):

assert size > 0, "Array size must be > 0"

self.\_size = size

# Create the array structure using the ctypes module.

PyArrayType = ctypes.py\_object \* size

self.\_elements = PyArrayType()

# Initialize each element

self.clear( None )

# Returns the size of the array

def \_\_len\_\_( self ):

return self.\_size

# Gets the contents of the index element

def \_\_getitem\_\_( self, index ):

assert index >= 0 and index < len(self), "Array subscript out of range"

return self.\_elements[ index ]

# Puts the value in the array element at index position

def \_\_setitem\_\_( self, index, value):

assert index >= 0 and index < len(self), "Array subscript out of range"

self.\_elements[ index ] = value

# Clears the array by setting each element to the given value

def clear( self, value ):

for i in range( len(self) ):

self.\_elements[i] = value

# Returns the array's iterator for traversing the elements

def \_\_iter\_\_( self ):

return \_ArrayIterator( self.\_elements )

class \_ArrayIterator:

def \_\_init\_\_( self, theArray):

self.\_arrayRef = theArray

self.\_curNdx = 0

def \_\_iter\_\_( self ):

return self

def next( self ):

if self.\_curNdx < len( self.\_arrayRef ):

entry = self.\_arrayRef[ self.\_curNdx]

self.\_curNdx += 1

return entry

else:

raise StopIteration

class MultiArray:

def \_\_init\_\_( self, \*dimensions ):

assert len(dimensions) > 1, "The array must have 2 or more dimensions."

self.\_dims = dimensions

size = 1

for d in dimensions:

assert d > 0, " Dimensions must be > 0."

size \*= d

self.\_elements = Array( size )

# Create the 1-D array to store the equation factors

self.\_factors = Array( len(dimensions) )

self.\_computeFactors()

# Returns the number of dimensions in the array.

def numDims( self ):

return len(self.\_dims)

# Returns the length of the given dimension.

def length( self, dim ):

assert dim >= 1 and dim <= len(self.\_dims), \

"Dimension component out of range."

return self.\_dims[dim-1]

def clear( self, value ):

self.\_elements.clear( value )

def \_\_getitem\_\_( self, ndxTuple ):

assert len(ndxTuple) == self.numDims(), "Invalid # of array subscripts."

index = self.\_computeIndex(ndxTuple)

assert index is not None, "Array subscript out of range."

return self.\_elements[index]

def \_\_setitem\_\_( self, ndxTuple, value ):

assert len(ndxTuple) == self.numDims(), "Invalid # of array subscripts."

index = self.\_computeIndex( ndxTuple )

assert index is not None, "Array subscript out of range."

self.\_elements[index] = value

def \_computeIndex( self, idx ):

offset = 0

for j in range( len(idx) ):

if idx[j] < 0 | idx[j] >= self.dims[j] :

return None

else:

offset += idx[j] \* self.\_factors[j]

return offset

def \_computeFactors( self ):

for j in range( self.numDims()-1 ):

self.\_factors[j] = 1

for k in range( j+1, self.numDims() ):

self.\_factors[j] \*= self.\_dims[k]

self.\_factors[self.numDims()-1] = 1

class ReportGenerator:

def \_\_init\_\_(self,filename):

self.\_filename = filename

self.\_inputFO = None#file object

self.\_multiArray = MultiArray(5,20,12)

def open(self):

self.\_inputFO = open(self.\_filename, "r")

def close(self):

self.\_inputFO.close()

self.\_inputFO = None

def load\_data(self):

self.open()

for line in self.\_inputFO:

words = line.split()

if len(words) == 2:

item\_counter = 0

f\_index = int(words[1][1])-1#store index

if len(words) == 13:

if words[0]== "Item#":

pass

else:

for i in range(12):

self.\_multiArray[f\_index, item\_counter, i]=float(words[i+1])

item\_counter +=1

self.close()

return self.\_multiArray

rg = ReportGenerator('SalesData.txt')

salesData = rg.load\_data()

def main():

salesData = MultiArray(8, 100, 12)

print("Continue to add things")

def totalSalesByStore(salesData, store):

s = store - 1

total = 0.0

for i in range(salesData.length(2)):

for m in range(salesData.length(3)):

total += salesData[s, i, m]

return total

def totalSalesByMonth(salesData, month):

m = month - 1

total = 0.0

for s in range(salesData.length(1)):

for i in range(salesData.length(2)):

total += salesData[s, i, m]

return total

def totalSalesByItem(salesData, item):

i = item - 1

total = 0.0

for s in range(salesData.length(1)):

for m in range(salesData.length(3)):

total += salesData[s, i, m]

return total

def totalSalesPerMonth(salesData, store):

s = store - 1

totals = Array(12)

for m in range(salesData.length(3)):

\_sum = 0.0

for i in range(salesData.length(2)):

\_sum += salesData[s, i, m]

totals[m] = \_sum

return totals

print( totalSalesByStore(salesData, 1))

print( totalSalesByItem(salesData, 2))

print( totalSalesByMonth(salesData, 4))

salespermonth = totalSalesPerMonth(salesData, 1)

for a in range(len(salespermonth)):

print( salespermonth[a])

def main():

print("welcome to the sales Report Program")

print("1.total Sales by Store")

print("2.total Sales by Item")

print("3.total Sales by Month")

print("4.Total Sales by Month")

print("Quit")

choice = int(input("Enter your choice: "))

while choice != 5:

if choice == 1:

store = int(input("Enter your store number: "))

print("Total sales for store", store , "is",totalSalesByStore(salesData, store))

elif choice == 2:

item = int(input("Enter your item number: "))

print("Total sales for store", item , "is",totalSalesByStore(salesData, item))

elif choice == 3:

month = int(input("Enter month number: "))

print("Total sales for month", month , "is",totalSalesByStore(salesData, month))

elif choice == 4:

store = int(input("Enter your store number: "))

salespermonth = totalSalesPerMonth(salesData, store)

for a in range(len(salespermonth)):

print(salespermonth[a])

else:

print("Invalid choice: ")

print("1.total Sales by Store")

print("2.total Sales by Item")

print("3.total Sales by Month")

print("4.Total Sales by Month")

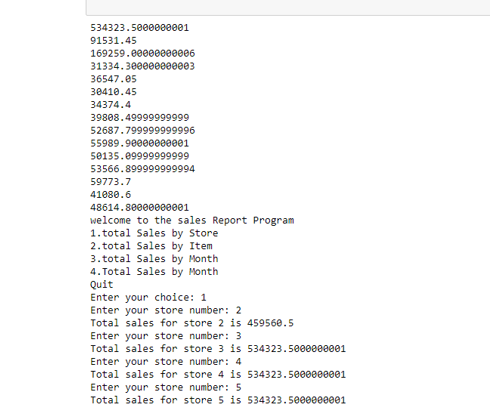
print("Quit")

choice = int(input("Enter your choice: "))

print('BYE')

main()

***Output:***

******

LAB 04

***Task 1:***

from matplotlib import pyplot as plt

import numpy as np

def version1(n):

totalSum = 0 # Version 1

matrix= np.random.randint(10, size=(n, n))

rowSum=[0]\*n

counter=3 #Counts the number of statement executed , excluding the counter updates

for i in range(0,n):

rowSum[i] = 0

counter+=2

for j in range(0, n ) :

rowSum[i] = rowSum[i] + matrix[i,j]

totalSum = totalSum + matrix[i,j]

counter+=2

return counter

def version2(n):

totalSum = 0 # Version 1

matrix= np.random.randint(10, size=(n, n))

rowSum=[0]\*n

counter=3 #Counts the number of statement excuted , excluding the counter updates

for i in range(0,n,1):

rowSum[i] = 0

counter+=1

for j in range(0, n ) :

rowSum[i] = rowSum[i] + matrix[i,j]

counter+=1

totalSum = totalSum + matrix[i,j]

counter+=2

return counter

def simulation(n):

steps\_version1=[0]\*n

steps\_version2 = [0] \* n

for i in range(0,n):

steps\_version1[i]=version1(i)

steps\_version2[i]=version2(i)

x=list(range(n))

plt.plot(x,steps\_version2)

plt.plot(x, steps\_version1)

plt.grid(which='both')

plt.xlabel('Input Size(n)')

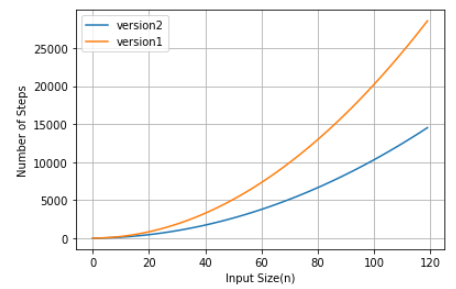
plt.ylabel('Number of Steps')

plt.legend(['version2','version1'])

plt.show()

simulation(120)

***OUTPUT:***

******

***Task 2:***

from timeit import Timer

import matplotlib.pyplot as plt

def concatenation():

l = []

for i in range(1000):

l = l + [i]

def append():

l = []

for i in range(1000):

l.append(i)

def comprehension():

l = [i for i in range(1000)]

def rangeFunction():

l = list(range(1000))

t1 = Timer("concatenation()", "from \_\_main\_\_ import concatenation")

concatTime = t1.timeit(number=1000)

print("concatination ", concatTime , "milliseconds")

t2 = Timer("append()", "from \_\_main\_\_ import append")

appendTime = t2.timeit(number=1000)

print("append ", appendTime , "milliseconds")

t3 = Timer("comprehension()", "from \_\_main\_\_ import comprehension")

compTime= t3.timeit(number=1000)

print("comprehension ", compTime , "milliseconds")

t4 = Timer("rangeFunction()", "from \_\_main\_\_ import rangeFunction")

rangeTime = t4.timeit(number=1000)

print("list range ",rangeTime , "milliseconds")

fig = plt.figure()

ax = fig.add\_axes([0,0,1,1])

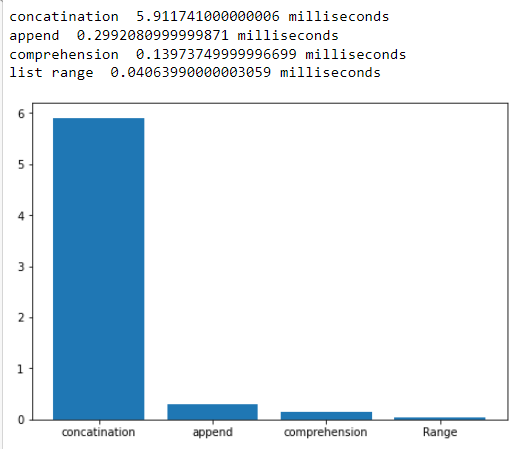
langs = ['concatination', 'append', 'comprehension', 'Range']

students = [concatTime ,appendTime ,compTime ,rangeTime]

ax.bar(langs,students)

plt.show()

***OUTPUT:***



***Task 3:***

from matplotlib import pyplot as plt

import numpy as np

def ex1(n):

count=0

for i in range(n):

count+=1

return count

def ex2(n):

count=0

for i in range(n):

count+=1

for j in range(n):

count+=1

return count

def ex3(n):

count=0

for i in range(n):

for j in range(n):

count+=1

return count

def ex4(n):

count=0

for i in range(n):

for j in range(10):

count+=1

return count

def ex5(n):

count=0

for i in range(n):

for j in range(i+1):

count+=1

return count

def ex6( n ):

count = 0

i = n

while i >= 1 :

count += 1

i = i // 2

return count

def ex7(n):

count=0

for i in range(n):

count+=ex6(n)

return count

def simulation(n):

steps\_version1=[0]\*n

steps\_version2 = [0] \* n

steps\_version3 = [0] \* n

steps\_version4 = [0] \* n

steps\_version5 = [0] \* n

steps\_version6 = [0] \* n

steps\_version7 = [0] \* n

for i in range(0,n):

steps\_version1[i]=ex1(i)

steps\_version2[i]=ex2(i)

steps\_version3[i]=ex3(i)

steps\_version4[i]=ex4(i)

steps\_version5[i]=ex5(i)

steps\_version6[i]=ex6(i)

steps\_version7[i]=ex7(i)

x=list(range(n))

plt.plot(x,steps\_version7)

plt.plot(x,steps\_version6)

plt.plot(x,steps\_version5)

plt.plot(x,steps\_version4)

plt.plot(x,steps\_version3)

plt.plot(x,steps\_version2)

plt.plot(x,steps\_version1)

plt.grid(which='both')

plt.xlabel(f'Input Size({n})')

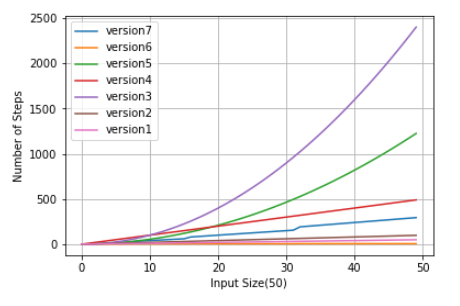
plt.ylabel('Number of Steps')

plt.legend(['version7','version6','version5','version4','version3','version2','version1'])

plt.show()

simulation(50)

***OUTPUT:***

******

***Task 4:***

import random

def lst():

lst=list()

for i in range(1000):

lst.append(i)

print(lst)

random.shuffle(lst)

for l in range(1000):

if lst[l]==50:

print(l)

if l==0:

print("best case")

elif l == 999 :

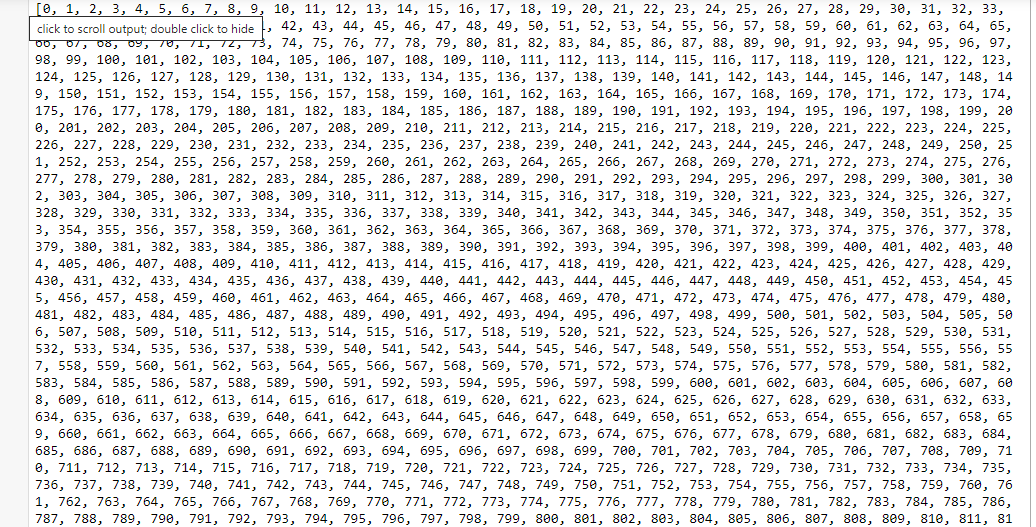
print("worst case")

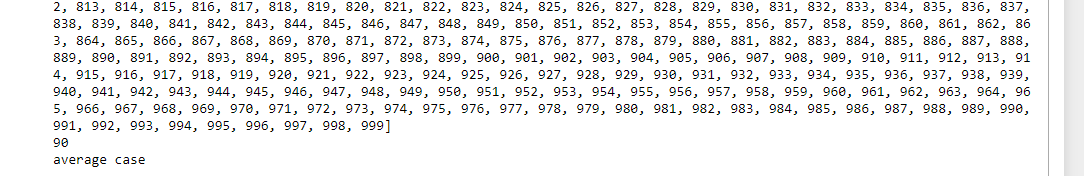
else:

print("average case")

lst()

***OUTPUT:***

******

******

LAB 05

***Exercises – Searching***

***Task 1: Linear search on unsorted sequence***

def linearSearch (theValues , target) :

n = len( theValues )

for i in range ( n ) :

# If the target is in the ith element , return True

if theValues [ i ] == target:

return True

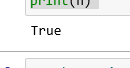
return False # If not found , return False .

lst = [1,5,6,7,4]

n = linearSearch(lst, 6)

print(n)

***OUTPUT:***

******

***Task 2: Linear search on a sorted sequence.***

***d***ef sortedLinearSearch ( theValues , item ) :

n = len( theValues )

for i in range ( n ) :

# If the target is found in the ith element , return True

if theValues [ i ] == item :

return True

elif theValues [ i ] > item :

return False

return False # The item is not in the sequence .

lst = [1,2,3,4,5]

a = linearSearch(lst, 2)

print(a)

***OUTPUT:***

******

***Task 3: The Binary Search***

def binarySearch ( theValues , target ) :

# Start with the entire sequence of elements .

low = 0

high = len( theValues ) - 1

# Repeatedly subdivide the sequence in half until the target is found .

while low <= high :

# Find the midpoint of the sequence .

mid = ( high + low ) // 2

# Does the midpoint contain the target ?

if theValues [ mid ] == target :

return True

# Or does the target precede the midpoint ?

elif target < theValues [ mid ] :

high = mid - 1

# Or does it follow the midpoint ?

else :

low = mid + 1

# If the sequence cannot be subdivided further , we ’re done .

return False

lst = [10,51,22,8,2,9]

b = binarySearch(lst, 10)

print(b)

***OUTPUT:***



***Exercises – Sorting***

***Task 1:*** ***Bubble Sort***

def bubbleSort(theSeq):

n = len(theSeq)

for i in range(n - 1):

for j in range(n - 1-i):

if theSeq[j] > theSeq[j + 1]:

tmp = theSeq[j]

theSeq[j] = theSeq[j + 1]

theSeq[j + 1] = tmp

return theSeq

seq = [10,51,38,22]

ab = bubbleSort(seq)

print(ab)

***OUTPUT:***



***Task 2:*** ***Selection Sort***

def selectionSort( theSeq ):

n = len( theSeq )

for i in range( n - 1 ):

smallNdx = i

for j in range( i + 1, n ):

if theSeq[j] < theSeq[smallNdx] :

smallNdx = j

if smallNdx != i :

tmp = theSeq[i]

theSeq[i] = theSeq[smallNdx]

theSeq[smallNdx] = tmp

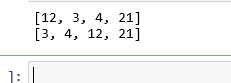
print(theSeq)

theSeq=[12,3,4,21]

print(theSeq)

selectionSort(theSeq)

***OUTPUT:***

******

***Task 3:*** ***InsertionSort***

def insertionSort( theSeq ):

n = len( theSeq )

for i in range( 1, n ) :

value = theSeq[i]

pos = i

while pos > 0 and value < theSeq[pos - 1] :

theSeq[pos] = theSeq[pos - 1]

pos -= 1

theSeq[pos] = value

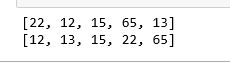
print(theSeq)

theValues=[22,12,15,65,13]

print(theValues)

insertionSort(theValues)

***OUTPUT:***



***Lab Tasks***

***Task 1: Binary Set ADT using Binary Search***

class Set:

def \_\_init\_\_(self):

self.\_list = list

def \_\_len\_\_(self):

return len(self.\_list)

def \_\_contains\_\_(self,element):

return element in self.\_list

def add(self,element):

assert element not in self.\_list, "Duplicate element"

if element not in self.\_list:

self.\_list.append(element)

def remove(self,element):

assert element not in self.\_list,"Element Dosenot exist ."

if element in self.\_list:

self.\_list.remove(element)

def intersect(self,setB):

newSet = Set()

for i in range(len(self.\_list)):

if setB[i] not in newSet.\_list and self.\_list[i] not in newSet.\_list:

newSet.\_list.append(self.\_list[i])

return newSet

def union(self,setB):

newSet = Set()

for i in range(len(self.\_list)):

if setB[i] not in newSet.\_list and self.\_list[i] not in newSet.\_list:

newSet.\_list.append(self.\_list[i])

return newSet

def binary\_search(self,item):

first = 0

last = len( self.\_list ) - 1

found = False

while ( first <= last and not found ):

midpoint = ( first + last ) // 2

if self.\_list[ midpoint ] == item :

found = True

else:

if item < self.\_list [ midpoint ] :

last = midpoint - 1

else:

first = midpoint + 1

return found

def \_\_iter\_\_(self:

return iter(self.\_list)

obj = Set()

obj.add(11)

obj.add(66)

obj.add(34)

obj.add(88)

obj.add(90)

obj.add(14)

print(obj.binary\_search(88))

***OUTPUT:***

**

***Task 2 & 3: Modify Binary Search Algorithm And Return list of negative values***

def findFirstOccurrence(nums, target):

left = 0

right = len(nums) - 1

result = -1

while left <= right:

mid = (left + right) // 2

if target == nums[mid]:

result = mid

right = mid - 1

elif target < nums[mid]:

right = mid - 1

else:

left = mid + 1

return result

print("Modified binary search:", findFirstOccurrence([5,99,1,22,3,5,2,99,100],99))

def negative\_values(theValues):

for i in range(len(theValues)):

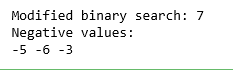
if theValues[i] < 0:

print(theValues[i], end = " ")

print("Negative values:")

negative\_values([-5,0,5,6,-6,4,-3])

***OUTPUT:***

******

***Task 4: Modify Bag ADT using Binary Search***

class Bag:

def \_\_init\_\_(self):

self.\_list = list()

def \_\_len\_\_(self):

return len(self.\_list)

def \_\_contains\_\_(self,element):

return element in self.\_list

def add(self,element):

self.\_list.append(element)

def remove(self,element):

assert element in self.\_list,"Element Dosenot exist ."

if element in self.\_list:

self.\_list.remove(element)

def \_\_iter\_\_(self):

return iter(self.\_list)

def binary\_search(self,item):

first = 0

last = len( self.\_list ) - 1

found = False

while ( first <= last and not found ):

midpoint = ( first + last ) // 2

if self.\_list[ midpoint ] == item :

found = True

else:

if item < self.\_list [ midpoint ] :

last = midpoint - 1

else:

first = midpoint + 1

return found

obj2 = Bag()

obj2.add(11)

obj2.add(66)

obj2.add(34)

obj2.add(88)

obj2.add(90)

print(obj2.binary\_search(90))

***OUTPUT:***

******

***Task 5: Modify MAP ADT using a Sortd list and Binary Search***

class Map:

def \_\_init\_\_(self):

self.\_list = list()

def \_\_len\_\_(self):

return len(self.\_list)

def \_\_contains\_\_(self,element):

return element in self.\_list

def add(self,element):

self.\_list.append(element)

def remove(self,element):

assert element in self.\_list,"Element Dosenot exist ."

if element in self.\_list:

self.\_list.remove(element)

def \_\_iter\_\_(self):

return iter(self.\_list)

def binary\_search(self,key):

first = 0

last = len( self.\_list ) - 1

found = False

while ( first <= last and not found ):

midpoint = ( first + last ) // 2

if self.\_list[ midpoint ][0] == key :

found = True

else:

if key < self.\_list [ midpoint ][0] :

last = midpoint - 1

else:

first = midpoint + 1

return found

def \_\_getitem\_\_(self,key):

for i in range(len(self.\_list)):

if self.\_list[i][0] == key:

return self.\_list[i][1]

return None

def \_\_setitem\_\_(self,key,value):

for i in range(len(self.\_list)):

if self.\_list[i][0] == key:

self.\_list[i][1] = value

return

self.\_list.append([key,value])

obj3 = Map()

obj3.add([11,22])

obj3.add([66,77])

obj3.add([34,55])

obj3.add([88,99])

print(obj3.binary\_search(88))

***OUTPUT:***

******

LAB 06

***#Q1***

**#Task\_01:**

**#Implementing Linked List**

class ListNode:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.\_head = None

def traversal(self):

curNode = self.\_head

while curNode is not None:

print(curNode.data)

curNode = curNode.next

def unorderedSearch(self, target):

curNode = self.\_head

while curNode is not None and curNode.data != target:

curNode = curNode.next

return curNode is not None

def prependNode(self, item):

newNode = ListNode(item)

newNode.next = self.\_head

self.\_head = newNode

def removeNode(self, item):

predNode = None

curNode = self.\_head

while curNode is not None and curNode.data != item:

predNode = curNode

curNode = curNode.next

assert curNode is not None, "Item must be in the Linked List!"

if curNode is self.\_head:

self.\_head = curNode.next

else:

predNode.next = curNode.next

linList = LinkedList()

linList.prependNode(1)

linList.prependNode(22)

linList.prependNode(31)

linList.prependNode(4)

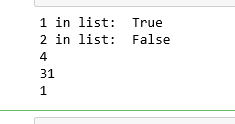
linList.removeNode(22)

print("1 in list: ", linList.unorderedSearch(1))

print("2 in list: ", linList.unorderedSearch(22))

linList.traversal()

***OUTPUT:***



***#Q2 #Implementing the Bag ADT using a singly linked list.***

*class Bag:*

*def \_\_init\_\_(self):*

*self.\_head = None*

*self.\_size = 0*

*def \_\_len\_\_(self):*

*return self.\_size*

*def \_\_contains\_\_(self, item):*

*curNode = self.\_head*

*while curNode is not None and curNode.data != item:*

*curNode = curNode.next*

*return curNode is not None*

*def add(self, item):*

*newNode = self.\_BagListNode(item)*

*newNode.next = self.\_head*

*self.\_head = newNode*

*self.\_size += 1*

*def remove(self, item):*

*predNode = None*

*curNode = self.\_head*

*while curNode is not None and curNode.data != item:*

*predNode = curNode*

*curNode = curNode.next*

*assert curNode is not None, "Item must be in the bag!"*

*self.\_size -= 1*

*if curNode is self.\_head:*

*self.\_head = curNode.next*

*else:*

*predNode.next = curNode.next*

*return curNode.data*

*def \_\_iter\_\_(self):*

*return self.\_BagIterator(self.\_head)*

*class \_BagListNode(object):*

*def \_\_init\_\_(self, data):*

*self.data = data*

*self.next = None*

*class \_BagIterator:*

*def \_\_init\_\_(self, head):*

*self.curNode = head*

*def \_\_iter\_\_(self):*

*return self*

*def \_\_next\_\_(self):*

*if self.curNode is not None:*

*curItem = self.curNode.data*

*self.curNode = self.curNode.next*

*return curItem*

*else:*

*raise StopIteration*

*def bagg():*

*bag = Bag()*

*bag.add("pencil")*

*bag.add("surf")*

*bag.add("pen")*

*bag.add("usb")*

*bag.remove("usb")*

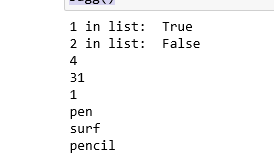
*for item in bag:*

*print(item)*

*linkk()*

*bagg()*

***OUTPUT:***

******

***Exercises:***

***Task1: #The removeAll(head)function, which accepts a head reference to a singly linked list, unlinks and remove every node individually from the list.***

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

def push(self, new\_data):

new\_node = Node(new\_data)

new\_node.next = self.head

self.head = new\_node

def removeAll(self, key):

temp = self.head

if (temp is not None):

if (temp.data == key):

self.head = temp.next

temp = None

return

while(temp is not None):

if temp.data == key:

break

prev = temp

temp = temp.next

if(temp == None):

return

prev.next = temp.next

temp = None

def printList(self):

temp = self.head

while(temp):

print (" %d" %(temp.data)),

temp = temp.next

# Driver program

llist = LinkedList()

llist.push(7)

llist.push(1)

llist.push(3)

llist.push(2)

print ("Created Linked List: ")

llist.printList()

llist.removeAll(3)

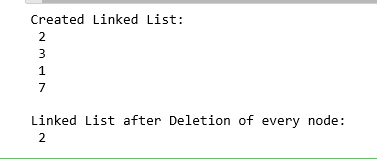
llist.removeAll(1)

llist.removeAll(7)

print ("\nLinked List after Deletion of every node:")

llist.printList()

***OUTPUT:***

******

***#Task\_02:***

*class Node:*

*def \_\_init\_\_(self, data, next = None):*

*self.data = data*

*self.next = None*

*class LinkedList:*

*def \_\_init\_\_(self):*

*self.head = None*

*def splitInHalf(self, listt1, listt2):*

*first = self.head*

*second = first.next*

*while (first is not None and*

*second is not None and*

*first.next is not None):*

*self.MoveNode(listt1, first)*

*self.MoveNode(listt2, second)*

*first = first.next.next*

*if first is None:*

*break*

*second = first.next*

*def MoveNode(self, dest, node):*

*new\_node = Node(node.data)*

*if dest.head is None:*

*dest.head = new\_node*

*else:*

*new\_node.next = dest.head*

*dest.head = new\_node*

*def push(self, data):*

*new\_node = Node(data)*

*new\_node.next = self.head*

*self.head = new\_node*

*def printList(self):*

*temp = self.head*

*while temp:*

*print (temp.data)*

*temp = temp.next*

*print("")*

*# Driver Code*

*if \_\_name\_\_ == "\_\_main\_\_":*

*# Start with empty list*

*llist = LinkedList()*

*listt1 = LinkedList()*

*listt2 = LinkedList()*

*llist.push(5)*

*llist.push(4)*

*llist.push(3)*

*llist.push(2)*

*llist.push(1)*

*llist.push(0)*

*llist.splitInHalf(listt1,listt2)*

*print("Original Linked List: ")*

*llist.printList()*

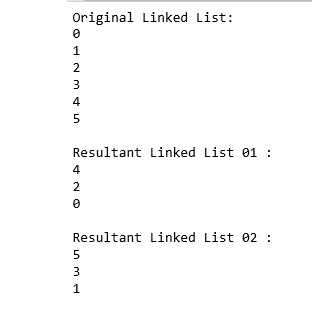
*print ("Resultant Linked List 01 : ")*

*listt1.printList()*

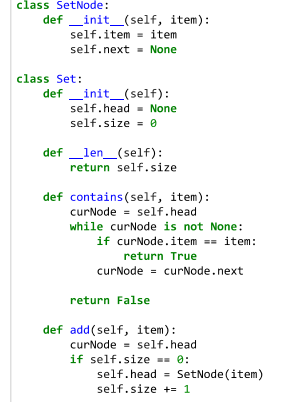
*print ("Resultant Linked List 02 : ")*

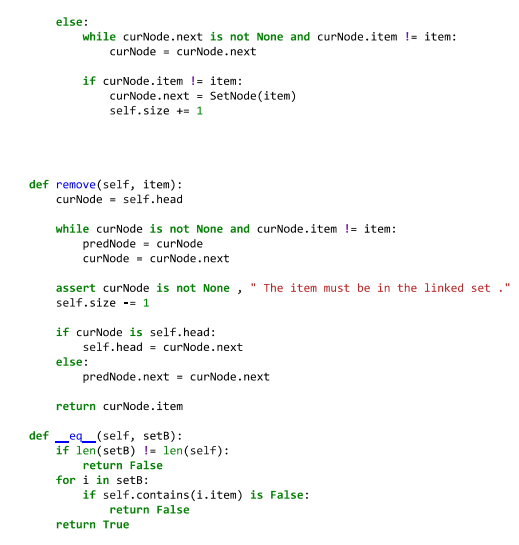
*listt2.printList()*

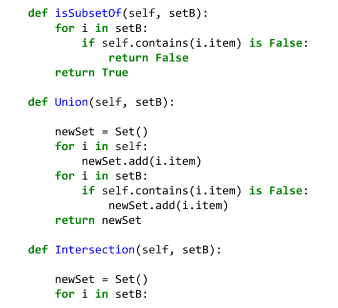
***OUTPUT:***

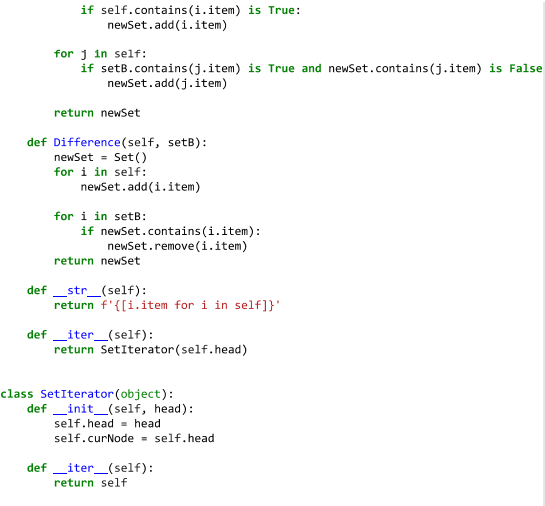


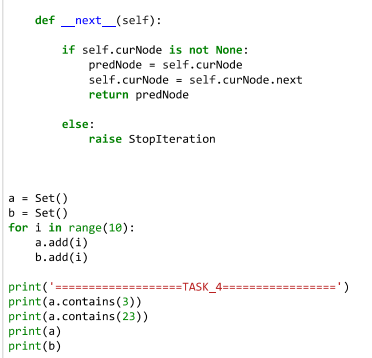
***TASK 03 Unsorted linked list:***

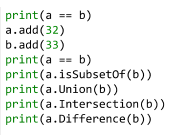
******

******

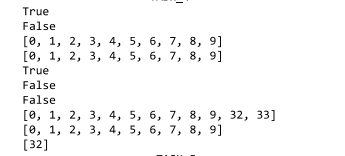
******

******

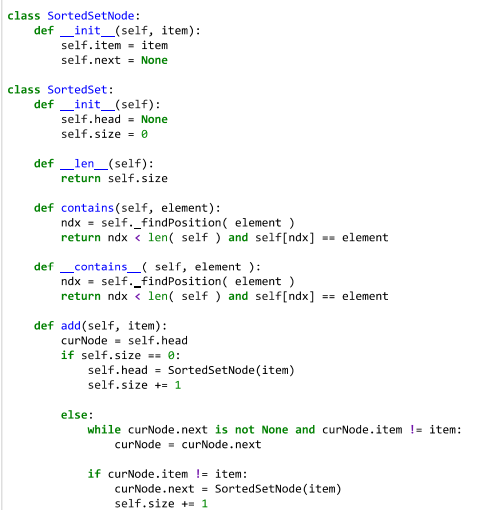
******

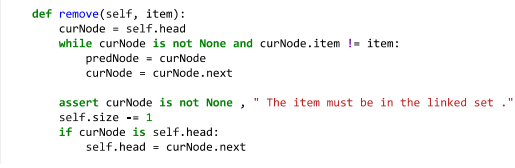
******

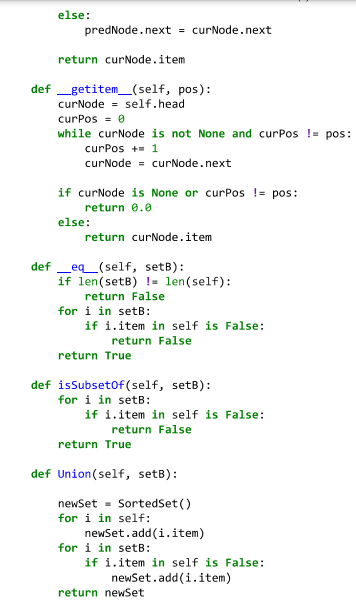
***OUTPUT:***

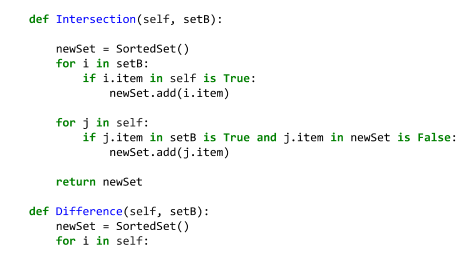
******

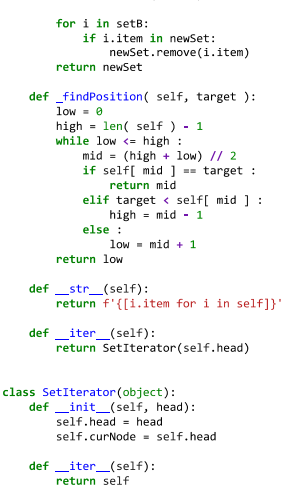
***TASK 04 Sorted linked list:***

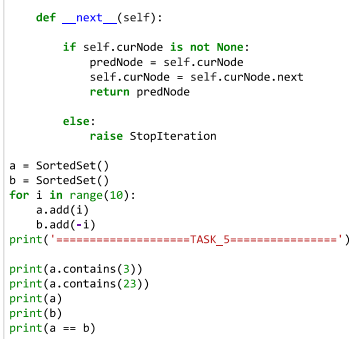
******

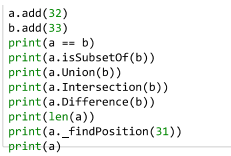
******

******

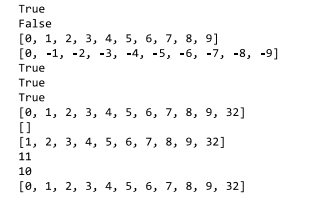
******

******

******

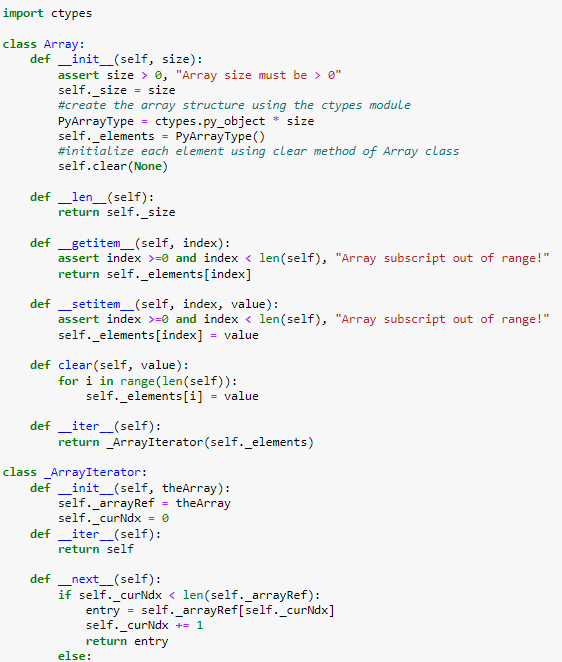
******

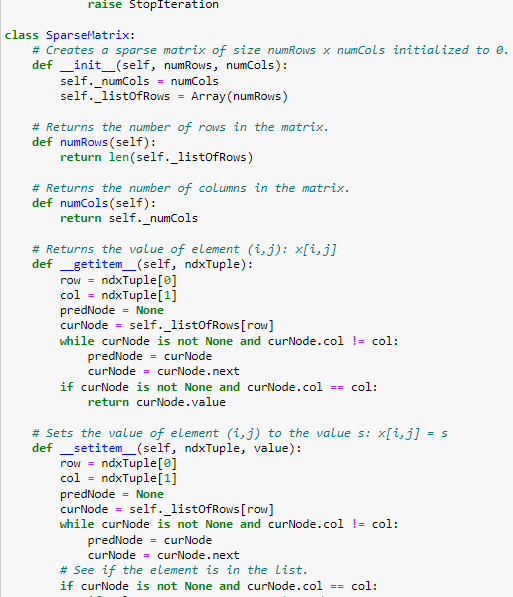
***OUTPUT:***

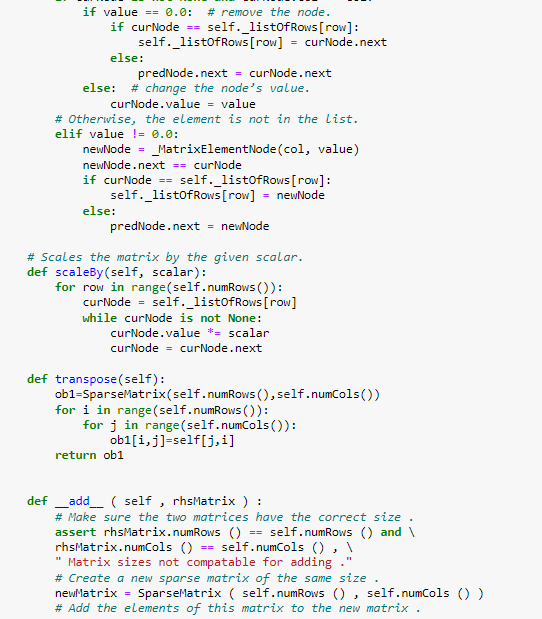
******

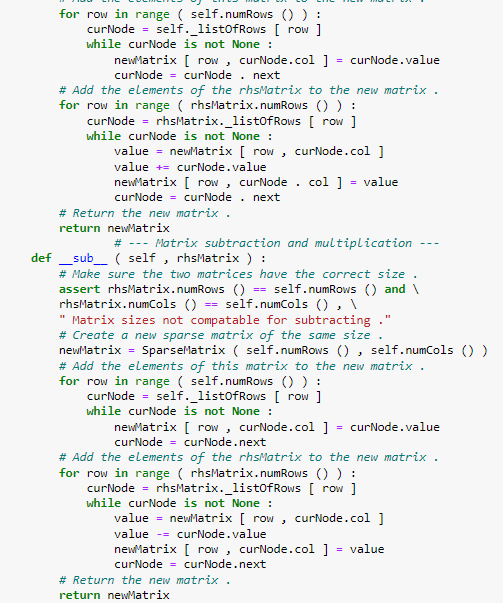
LAB 07

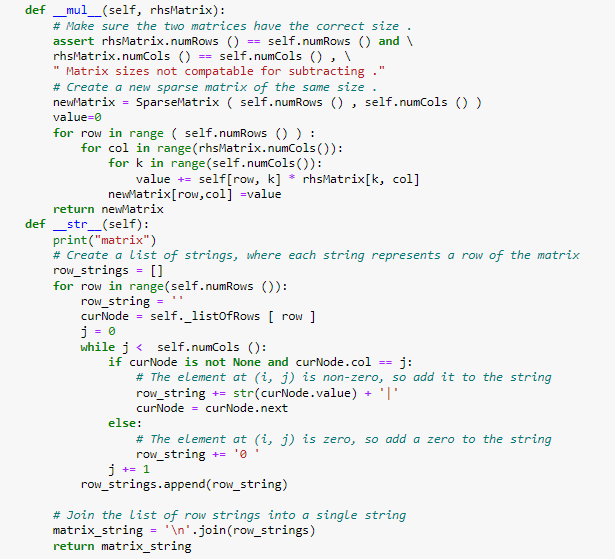
**

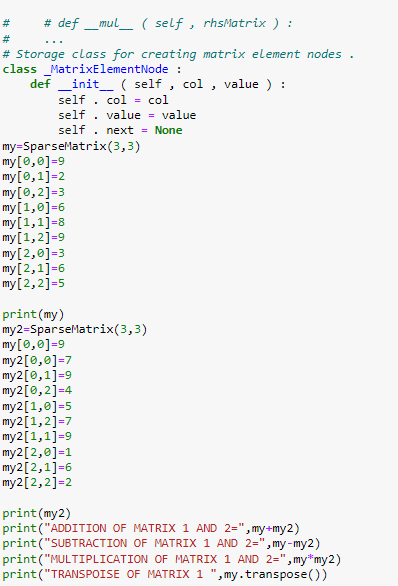
**

**

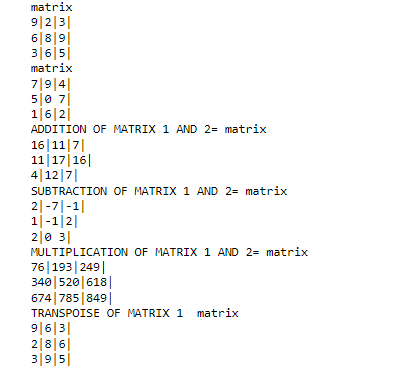
**

**

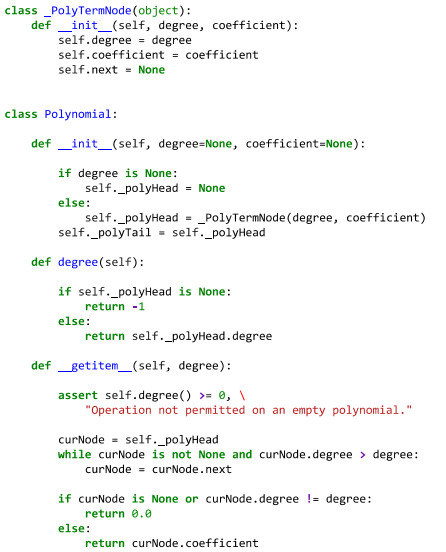
**

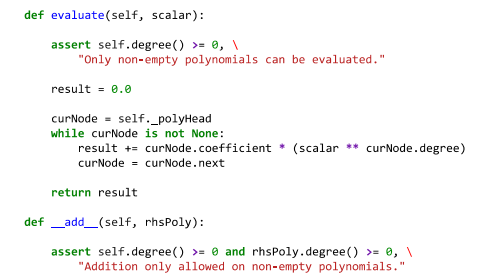
**

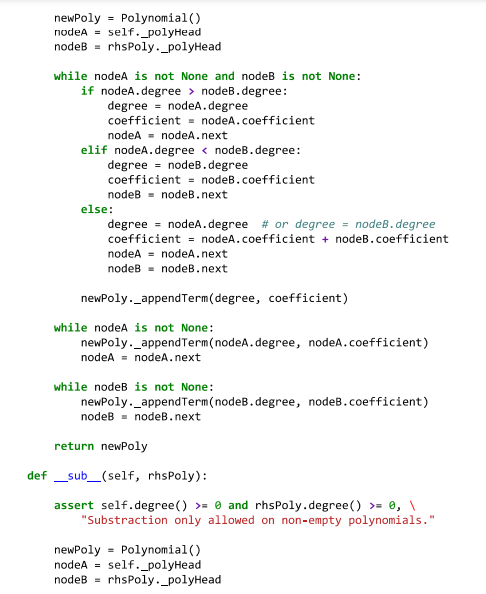
***OUTPUT:***

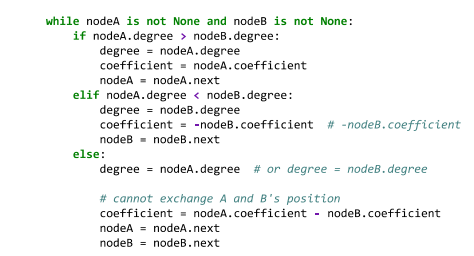
**

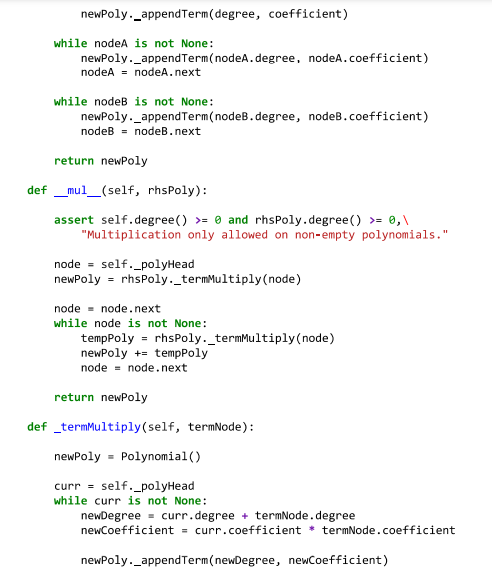
**

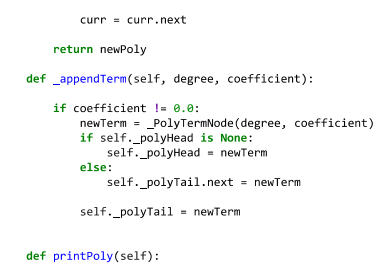
**

**

**

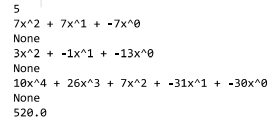
**

**

**

**

***OUTPUT:***

**