#DSA-Exer-1

def update\_mark\_list(mark\_list, new\_element, pos):

#Write your logic here

return mark\_list

def find\_mark(mark\_list,pos1,pos2):

'''Remove pass and write your logic here to return a list containing

the marks at pos1 and pos2 respectively.'''

pass

#Provide different values for the variables and test your program

mark\_list=[89,78,99,76,77,72,88,99]

new\_element=69

pos=2

pos1=5

pos2=8

print(update\_mark\_list(mark\_list, new\_element, pos))

print(find\_mark(mark\_list, pos1, pos2))

#DSA-Exer-2

class Node:

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

self.\_\_data=data

self.\_\_next=None

def get\_data(self):

return self.\_\_data

def set\_data(self,data):

self.\_\_data=data

def get\_next(self):

return self.\_\_next

def set\_next(self,next\_node):

self.\_\_next=next\_node

class LinkedList:

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

self.\_\_head=None

self.\_\_tail=None

def get\_head(self):

return self.\_\_head

def get\_tail(self):

return self.\_\_tail

def add(self,data):

new\_node=Node(data)

if(self.\_\_head is None):

self.\_\_head=self.\_\_tail=new\_node

else:

self.\_\_tail.set\_next(new\_node)

self.\_\_tail=new\_node

def display(self):

temp=self.\_\_head

while(temp is not None):

print(temp.get\_data())

temp=temp.get\_next()

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

temp=self.\_\_head

msg=[]

while(temp is not None):

msg.append(str(temp.get\_data()))

temp=temp.get\_next()

msg=" ".join(msg)

msg="Linkedlist data(Head to Tail): "+ msg

return msg

def count\_nodes(biscuit\_list):

count=0

# Write your logic here

return count

biscuit\_list=LinkedList()

biscuit\_list.add("Goodday")

biscuit\_list.add("Bourbon")

biscuit\_list.add("Hide&Seek")

biscuit\_list.add("Nutrichoice")

print(count\_nodes(biscuit\_list))

#DSA-Assgn-1

def merge\_list(list1, list2):

merged\_data=""

#write your logic here

return resultant\_data

#Provide different values for the variables and test your program

list1=['A', 'app','a', 'd', 'ke', 'th', 'doc', 'awa']

list2=['y','tor','e','eps','ay',None,'le','n']

merged\_data=merge\_list(list1,list2)

print(merged\_data)

#DSA-Assgn-2

class Car:

def \_\_init\_\_(self,model,year,registration\_number):

self.\_\_model=model

self.\_\_year=year

self.\_\_registration\_number=registration\_number

def get\_model(self):

return self.\_\_model

def get\_year(self):

return self.\_\_year

def get\_registration\_number(self):

return self.\_\_registration\_number

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

return(self.\_\_model+" "+self.\_\_registration\_number+" "+(str)(self.\_\_year))

class Service:

def \_\_init\_\_(self,car\_list):

self.\_\_car\_list = car\_list

def get\_car\_list(self):

return self.\_\_car\_list

def find\_cars\_by\_year(self,year):

cars\_year\_wise\_list =[]

for car\_object in self.\_\_car\_list:

if car\_object.get\_year() == year:

cars\_year\_wise\_list.append(car\_object.get\_model())

if len(cars\_year\_wise\_list) == 0:

return None

return cars\_year\_wise\_list

def add\_cars(self,new\_car\_list):

self.\_\_car\_list += new\_car\_list

self.\_\_car\_list.sort(key=lambda car : car.get\_year(), reverse=False)

def remove\_cars\_from\_karnataka(self):

carlist = self.\_\_car\_list

self.\_\_car\_list=[]

for car in carlist:

if car.get\_registration\_number()[:2] != 'KA':

self.\_\_car\_list.append(car)

#Implement Service class here

car1=Car("WagonR",2010,"KA09 3056")

car2=Car("Beat", 2011, "MH10 6776")

car3=Car("Ritz", 2013,"KA12 9098")

car4=Car("Polo",2013,"GJ01 7854")

car5=Car("Amaze",2014,"KL07 4332")

#Add different values to the list and test the program

car\_list=[car1, car2, car3, car4,car5]

#Create object of Service class, invoke the methods and test your program

#DSA-Assgn-3

def check\_double(list1,list2):

new\_list=[]

for i in list1:

if 2\*i in list2:

new\_list.append(i)

return new\_list

#Provide different values for the variables and test your program

list1=[11,8,23,7,25,15]

list2=[6,33,50,31,46,78,16,34]

print(check\_double(list1, list2))

#DSA-Assgn-4

class Player:

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

self.\_\_name=name

self.\_\_experience=experience

def get\_name(self):

return self.\_\_name

def get\_experience(self):

return self.\_\_experience

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

return(self.\_\_name+" "+(str)(self.\_\_experience))

#Implement Game class here

class Game:

def \_\_init\_\_(self,players\_list):

self.\_\_players\_list = players\_list

def sort\_players\_based\_on\_experience(self):

self.\_\_players\_list.sort(key=lambda player : player.get\_experience(), reverse=True)

def shift\_player\_to\_new\_position (self,old\_index\_position, new\_index\_position):

self.\_\_players\_list.insert (new\_index\_position,(self.\_\_players\_list.pop(old\_index\_position)))

def display\_player\_details(self):

pass

player1=Player("Dhoni",15)

player2=Player("Virat",10)

player3=Player("Rohit",12)

player4=Player("Raina",11)

player5=Player("Jadeja",13)

player6=Player("Ishant",9)

player7=Player("Shikhar",8)

player8=Player("Axar",7.5)

player9=Player("Ashwin",6)

player10=Player("Stuart",7)

player11=Player("Bhuvneshwar",5)

#Add different values to the list and test the program

players\_list=[player1,player2,player3,player4,player5,player6,player7,player8,player9,player10,player11]

#Create object of Game class, invoke the methods and test your program

#DSA-Exer-3

class Node:

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

self.\_\_data=data

self.\_\_next=None

def get\_data(self):

return self.\_\_data

def set\_data(self,data):

self.\_\_data=data

def get\_next(self):

return self.\_\_next

def set\_next(self,next\_node):

self.\_\_next=next\_node

class LinkedList:

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

self.\_\_head=None

self.\_\_tail=None

def get\_head(self):

return self.\_\_head

def get\_tail(self):

return self.\_\_tail

def add(self,data):

new\_node=Node(data)

if(self.\_\_head is None):

self.\_\_head=self.\_\_tail=new\_node

else:

self.\_\_tail.set\_next(new\_node)

self.\_\_tail=new\_node

def insert(self,data,data\_before):

new\_node=Node(data)

if(data\_before==None):

new\_node.set\_next(self.\_\_head)

self.\_\_head=new\_node

if(new\_node.get\_next()==None):

self.\_\_tail=new\_node

else:

node\_before=self.find\_node(data\_before)

if(node\_before is not None):

new\_node.set\_next(node\_before.get\_next())

node\_before.set\_next(new\_node)

if(new\_node.get\_next() is None):

self.\_\_tail=new\_node

else:

print(data\_before,"is not present in the Linked list")

def display(self):

temp=self.\_\_head

while(temp is not None):

print(temp.get\_data())

temp=temp.get\_next()

def find\_node(self,data):

temp=self.\_\_head

while(temp is not None):

if(temp.get\_data()==data):

return temp

temp=temp.get\_next()

return None

def delete(self,data):

node=self.find\_node(data)

if(node is not None):

if(node==self.\_\_head):

if(self.\_\_head==self.\_\_tail):

self.\_\_tail=None

self.\_\_head=node.get\_next()

else:

temp=self.\_\_head

while(temp is not None):

if(temp.get\_next()==node):

temp.set\_next(node.get\_next())

if(node==self.\_\_tail):

self.\_\_tail=temp

node.set\_next(None)

break

temp=temp.get\_next()

else:

print(data,"is not present in Linked list")

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

temp=self.\_\_head

msg=[]

while(temp is not None):

msg.append(str(temp.get\_data()))

temp=temp.get\_next()

msg=" ".join(msg)

msg="Linkedlist data(Head to Tail): "+ msg

return msg

def find\_sum(number\_list):

sum=0

temp = number\_list.get\_head()

count = 1

while temp:

if count % 2 != 0:

sum += temp.get\_data()

count +=1

temp = temp.get\_next()

# Write your logic here

return sum

number\_list=LinkedList()

number\_list.add(10)

number\_list.add(20)

number\_list.add(30)

number\_list.add(40)

number\_list.add(50)

number\_list.add(60)

number\_list.add(70)

number\_list.add(80)

number\_list.add(90)

number\_list.add(100)

number\_list.add(110)

print(find\_sum(number\_list))

#DSA-Exer-4

class Node:

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

self.\_\_data=data

self.\_\_next=None

def get\_data(self):

return self.\_\_data

def set\_data(self,data):

self.\_\_data=data

def get\_next(self):

return self.\_\_next

def set\_next(self,next\_node):

self.\_\_next=next\_node

class LinkedList:

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

self.\_\_head=None

self.\_\_tail=None

def get\_head(self):

return self.\_\_head

def get\_tail(self):

return self.\_\_tail

def add(self,data):

new\_node=Node(data)

if(self.\_\_head is None):

self.\_\_head=self.\_\_tail=new\_node

else:

self.\_\_tail.set\_next(new\_node)

self.\_\_tail=new\_node

def insert(self,data,data\_before):

new\_node=Node(data)

if(data\_before==None):

new\_node.set\_next(self.\_\_head)

self.\_\_head=new\_node

if(new\_node.get\_next()==None):

self.\_\_tail=new\_node

else:

node\_before=self.find\_node(data\_before)

if(node\_before is not None):

new\_node.set\_next(node\_before.get\_next())

node\_before.set\_next(new\_node)

if(new\_node.get\_next() is None):

self.\_\_tail=new\_node

else:

print(data\_before,"is not present in the Linked list")

def display(self):

temp=self.\_\_head

while(temp is not None):

print(temp.get\_data())

temp=temp.get\_next()

def find\_node(self,data):

temp=self.\_\_head

while(temp is not None):

if(temp.get\_data()==data):

return temp

temp=temp.get\_next()

return None

def delete(self,data):

node=self.find\_node(data)

if(node is not None):

if(node==self.\_\_head):

if(self.\_\_head==self.\_\_tail):

self.\_\_tail=None

self.\_\_head=node.get\_next()

else:

temp=self.\_\_head

while(temp is not None):

if(temp.get\_next()==node):

temp.set\_next(node.get\_next())

if(node==self.\_\_tail):

self.\_\_tail=temp

node.set\_next(None)

break

temp=temp.get\_next()

else:

print(data,"is not present in Linked list")

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

temp=self.\_\_head

msg=[]

while(temp is not None):

msg.append(str(temp.get\_data()))

temp=temp.get\_next()

msg=" ".join(msg)

msg="Linkedlist data(Head to Tail): "+ msg

return msg

def replace\_maximum(number\_list,new\_value):

temp = number\_list.get\_head()

mx=0

while temp:

mx = max(mx,temp.get\_data())

temp = temp.get\_next()

temp = number\_list.get\_head()

while temp:

if mx == temp.get\_data():

temp.set\_data(new\_value)

break

temp = temp.get\_next()

return number\_list

#Add different values to the linked list and test your program

number\_list=LinkedList()

number\_list.add(12)

number\_list.add(95)

number\_list.add(940)

number\_list.add(110)

number\_list.add(240)

new\_value=100

number\_list=replace\_maximum(number\_list,new\_value)

number\_list.display()

#DSA-Exer-5

class Node:

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

self.\_\_data=data

self.\_\_next=None

def get\_data(self):

return self.\_\_data

def set\_data(self,data):

self.\_\_data=data

def get\_next(self):

return self.\_\_next

def set\_next(self,next\_node):

self.\_\_next=next\_node

class LinkedList:

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

self.\_\_head=None

self.\_\_tail=None

def get\_head(self):

return self.\_\_head

def get\_tail(self):

return self.\_\_tail

def add(self,data):

new\_node=Node(data)

if(self.\_\_head is None):

self.\_\_head=self.\_\_tail=new\_node

else:

self.\_\_tail.set\_next(new\_node)

self.\_\_tail=new\_node

def insert(self,data,data\_before):

new\_node=Node(data)

if(data\_before==None):

new\_node.set\_next(self.\_\_head)

self.\_\_head=new\_node

if(new\_node.get\_next()==None):

self.\_\_tail=new\_node

else:

node\_before=self.find\_node(data\_before)

if(node\_before is not None):

new\_node.set\_next(node\_before.get\_next())

node\_before.set\_next(new\_node)

if(new\_node.get\_next() is None):

self.\_\_tail=new\_node

else:

print(data\_before,"is not present in the Linked list")

def display(self):

temp=self.\_\_head

while(temp is not None):

print(temp.get\_data())

temp=temp.get\_next()

def find\_node(self,data):

temp=self.\_\_head

while(temp is not None):

if(temp.get\_data()==data):

return temp

temp=temp.get\_next()

return None

def delete(self,data):

node=self.find\_node(data)

if(node is not None):

if(node==self.\_\_head):

if(self.\_\_head==self.\_\_tail):

self.\_\_tail=None

self.\_\_head=node.get\_next()

else:

temp=self.\_\_head

while(temp is not None):

if(temp.get\_next()==node):

temp.set\_next(node.get\_next())

if(node==self.\_\_tail):

self.\_\_tail=temp

node.set\_next(None)

break

temp=temp.get\_next()

else:

print(data,"is not present in Linked list")

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

temp=self.\_\_head

msg=[]

while(temp is not None):

msg.append(str(temp.get\_data()))

temp=temp.get\_next()

msg=" ".join(msg)

msg="Linkedlist data(Head to Tail): "+ msg

return msg

class Circle:

def \_\_init\_\_(self, color,radius):

self.\_\_color=color

self.\_\_radius=radius

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

return (self.\_\_color+" "+str(self.\_\_radius))

def get\_color(self):

return self.\_\_color

def get\_radius(self):

return self.\_\_radius

class Shape:

def \_\_init\_\_(self,circle\_list):

self.\_\_circle\_list=circle\_list

def get\_circle\_list (self):

return self.\_\_circle\_list

def insert\_circle (self,new\_circle):

self.\_\_circle\_list.insert(new\_circle, None)

# Write your logic here

circle1=Circle("Red",4)

circle2=Circle("Green",5)

circle3=Circle("Purple",3.5)

new\_circle=Circle("Blue",6)

circle\_list=LinkedList()

circle\_list.add(circle1)

circle\_list.add(circle2)

circle\_list.add(circle3)

shape=Shape(circle\_list)

shape.insert\_circle(new\_circle)

shape.get\_circle\_list().display()

#DSA-Assgn-5

class Node:

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

self.\_\_data=data

self.\_\_next=None

def get\_data(self):

return self.\_\_data

def set\_data(self,data):

self.\_\_data=data

def get\_next(self):

return self.\_\_next

def set\_next(self,next\_node):

self.\_\_next=next\_node

class LinkedList:

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

self.\_\_head=None

self.\_\_tail=None

def get\_head(self):

return self.\_\_head

def get\_tail(self):

return self.\_\_tail

def add(self,data):

new\_node=Node(data)

if(self.\_\_head is None):

self.\_\_head=self.\_\_tail=new\_node

else:

self.\_\_tail.set\_next(new\_node)

self.\_\_tail=new\_node

def insert(self,data,data\_before):

new\_node=Node(data)

if(data\_before==None):

new\_node.set\_next(self.\_\_head)

self.\_\_head=new\_node

if(new\_node.get\_next()==None):

self.\_\_tail=new\_node

else:

node\_before=self.find\_node(data\_before)

if(node\_before is not None):

new\_node.set\_next(node\_before.get\_next())

node\_before.set\_next(new\_node)

if(new\_node.get\_next() is None):

self.\_\_tail=new\_node

else:

print(data\_before,"is not present in the Linked list")

def display(self):

temp=self.\_\_head

while(temp is not None):

print(temp.get\_data())

temp=temp.get\_next()

def find\_node(self,data):

temp=self.\_\_head

while(temp is not None):

if(temp.get\_data()==data):

return temp

temp=temp.get\_next()

return None

def delete(self,data):

node=self.find\_node(data)

if(node is not None):

if(node==self.\_\_head):

if(self.\_\_head==self.\_\_tail):

self.\_\_tail=None

self.\_\_head=node.get\_next()

else:

temp=self.\_\_head

while(temp is not None):

if(temp.get\_next()==node):

temp.set\_next(node.get\_next())

if(node==self.\_\_tail):

self.\_\_tail=temp

node.set\_next(None)

break

temp=temp.get\_next()

else:

print(data,"is not present in Linked list")

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

temp=self.\_\_head

msg=[]

while(temp is not None):

msg.append(str(temp.get\_data()))

temp=temp.get\_next()

msg=" ".join(msg)

msg="Linkedlist data(Head to Tail): "+ msg

return msg

def create\_new\_sentence(word\_list):

new\_sentence=""

tmp = ''

current = word\_list.get\_head()

while (current):

tmp = current.get\_data()

if tmp == "\*" or tmp == "/":

new\_sentence +=" "

elif new\_sentence[-2:] == " ":

new\_sentence += tmp.upper()

else:

new\_sentence += tmp

current = current.get\_next()

new\_sentence = new\_sentence.replace(' ', ' ')

return new\_sentence

word\_list=LinkedList()

word\_list.add("T")

word\_list.add("h")

word\_list.add("e")

word\_list.add("/")

word\_list.add("\*")

word\_list.add("s")

word\_list.add("k")

word\_list.add("y")

word\_list.add("\*")

word\_list.add("i")

word\_list.add("s")

word\_list.add("/")

word\_list.add("/")

word\_list.add("b")

word\_list.add("l")

word\_list.add("u")

word\_list.add("e")

result=create\_new\_sentence(word\_list)

print(result)

#DSA-Assgn-6

class Node:

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

self.\_\_data=data

self.\_\_next=None

def get\_data(self):

return self.\_\_data

def set\_data(self,data):

self.\_\_data=data

def get\_next(self):

return self.\_\_next

def set\_next(self,next\_node):

self.\_\_next=next\_node

class LinkedList:

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

self.\_\_head=None

self.\_\_tail=None

def get\_head(self):

return self.\_\_head

def get\_tail(self):

return self.\_\_tail

def add(self,data):

new\_node=Node(data)

if(self.\_\_head is None):

self.\_\_head=self.\_\_tail=new\_node

else:

self.\_\_tail.set\_next(new\_node)

self.\_\_tail=new\_node

def insert(self,data,data\_before):

new\_node=Node(data)

if(data\_before==None):

new\_node.set\_next(self.\_\_head)

self.\_\_head=new\_node

if(new\_node.get\_next()==None):

self.\_\_tail=new\_node

else:

node\_before=self.find\_node(data\_before)

if(node\_before is not None):

new\_node.set\_next(node\_before.get\_next())

node\_before.set\_next(new\_node)

if(new\_node.get\_next() is None):

self.\_\_tail=new\_node

else:

print(data\_before,"is not present in the Linked list")

def display(self):

temp=self.\_\_head

while(temp is not None):

print(temp.get\_data())

temp=temp.get\_next()

def find\_node(self,data):

temp=self.\_\_head

while(temp is not None):

if(temp.get\_data()==data):

return temp

temp=temp.get\_next()

return None

def delete(self,data):

node=self.find\_node(data)

if(node is not None):

if(node==self.\_\_head):

if(self.\_\_head==self.\_\_tail):

self.\_\_tail=None

self.\_\_head=node.get\_next()

else:

temp=self.\_\_head

while(temp is not None):

if(temp.get\_next()==node):

temp.set\_next(node.get\_next())

if(node==self.\_\_tail):

self.\_\_tail=temp

node.set\_next(None)

break

temp=temp.get\_next()

else:

print(data,"is not present in Linked list")

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

temp=self.\_\_head

msg=[]

while(temp is not None):

msg.append(str(temp.get\_data()))

temp=temp.get\_next()

msg=" ".join(msg)

msg="Linkedlist data(Head to Tail): "+ msg

return msg

class Child:

def \_\_init\_\_(self,name,item\_to\_perform):

self.\_\_name=name

self.\_\_item\_to\_perform=item\_to\_perform

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

return(self.\_\_name+" "+self.\_\_item\_to\_perform)

def get\_name(self):

return self.\_\_name

def get\_item\_to\_perform(self):

return self.\_\_item\_to\_perform

class Performance:

def \_\_init\_\_(self,children\_list):

self.\_\_children\_list = children\_list

def get\_children\_list (self):

return self.\_\_children\_list

def change\_position (self,child):

lnk = self.\_\_children\_list

lnk.delete (child)

current = lnk.get\_head()

length = 0

while current:

length +=1

current = current.get\_next()

before\_child = lnk.get\_head()

a=length

while (length != (a//2)+1):

before\_child = before\_child.get\_next()

length-=1

lnk.insert(child,before\_child.get\_data())

def add\_new\_child(self,child):

lnk = self.\_\_children\_list

lnk.add(child)

#Implement Performance class here

child1=Child("Rahul","solo song")

child2=Child("Sheema","Dance")

child3=Child("Gitu","Plays Flute")

child4=Child("Tarun","Gymnastics")

child5=Child("Tom","MIME")

#Add different values to the list and test the program

children\_list=LinkedList()

children\_list.add(child1)

children\_list.add(child2)

children\_list.add(child3)

children\_list.add(child4)

children\_list.add(child5)

performance=Performance(children\_list)

print("The order in which the children would perform:")

performance.get\_children\_list().display()

print()

print("After Rahul's performance, the schedule would change to:")

performance.change\_position(child1)

performance.get\_children\_list().display()

print()

child6=Child("Swetha","Vote of Thanks")

print("After Swetha has joined, the schedule is:")

performance.add\_new\_child(child6)

performance.get\_children\_list().display()

#DSA-Assgn-7

class Node:

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

self.\_\_data=data

self.\_\_next=None

def get\_data(self):

return self.\_\_data

def set\_data(self,data):

self.\_\_data=data

def get\_next(self):

return self.\_\_next

def set\_next(self,next\_node):

self.\_\_next=next\_node

class LinkedList:

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

self.\_\_head=None

self.\_\_tail=None

def get\_head(self):

return self.\_\_head

def get\_tail(self):

return self.\_\_tail

def add(self,data):

new\_node=Node(data)

if(self.\_\_head is None):

self.\_\_head=self.\_\_tail=new\_node

else:

self.\_\_tail.set\_next(new\_node)

self.\_\_tail=new\_node

def insert(self,data,data\_before):

new\_node=Node(data)

if(data\_before==None):

new\_node.set\_next(self.\_\_head)

self.\_\_head=new\_node

if(new\_node.get\_next()==None):

self.\_\_tail=new\_node

else:

node\_before=self.find\_node(data\_before)

if(node\_before is not None):

new\_node.set\_next(node\_before.get\_next())

node\_before.set\_next(new\_node)

if(new\_node.get\_next() is None):

self.\_\_tail=new\_node

else:

print(data\_before,"is not present in the Linked list")

def display(self):

temp=self.\_\_head

while(temp is not None):

print(temp.get\_data())

temp=temp.get\_next()

def find\_node(self,data):

temp=self.\_\_head

while(temp is not None):

if(temp.get\_data()==data):

return temp

temp=temp.get\_next()

return None

def delete(self,data):

node=self.find\_node(data)

if(node is not None):

if(node==self.\_\_head):

if(self.\_\_head==self.\_\_tail):

self.\_\_tail=None

self.\_\_head=node.get\_next()

else:

temp=self.\_\_head

while(temp is not None):

if(temp.get\_next()==node):

temp.set\_next(node.get\_next())

if(node==self.\_\_tail):

self.\_\_tail=temp

node.set\_next(None)

break

temp=temp.get\_next()

else:

print(data,"is not present in Linked list")

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

temp=self.\_\_head

msg=[]

while(temp is not None):

msg.append(str(temp.get\_data()))

temp=temp.get\_next()

msg=" ".join(msg)

msg="Linkedlist data(Head to Tail): "+ msg

return msg

def remove\_duplicates(duplicate\_list):

cur = duplicate\_list.get\_head()

nxt = cur.get\_next()

while nxt :

if cur.get\_data() == nxt.get\_data():

delt = cur

duplicate\_list.delete(delt.get\_data())

cur = nxt

nxt = nxt.get\_next()

else:

cur = nxt

nxt = nxt.get\_next()

return duplicate\_list

#Add different values to the linked list and test your program

duplicate\_list=LinkedList()

duplicate\_list.add(30)

duplicate\_list.add(40)

duplicate\_list.add(40)

duplicate\_list.add(40)

duplicate\_list.add(40)

remove\_duplicates(duplicate\_list)

#DSA-Assgn-8

class Node:

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

self.\_\_data=data

self.\_\_next=None

def get\_data(self):

return self.\_\_data

def set\_data(self,data):

self.\_\_data=data

def get\_next(self):

return self.\_\_next

def set\_next(self,next\_node):

self.\_\_next=next\_node

class LinkedList:

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

self.\_\_head=None

self.\_\_tail=None

def get\_head(self):

return self.\_\_head

def get\_tail(self):

return self.\_\_tail

def add(self,data):

new\_node=Node(data)

if(self.\_\_head is None):

self.\_\_head=self.\_\_tail=new\_node

else:

self.\_\_tail.set\_next(new\_node)

self.\_\_tail=new\_node

def insert(self,data,data\_before):

new\_node=Node(data)

if(data\_before==None):

new\_node.set\_next(self.\_\_head)

self.\_\_head=new\_node

if(new\_node.get\_next()==None):

self.\_\_tail=new\_node

else:

node\_before=self.find\_node(data\_before)

if(node\_before is not None):

new\_node.set\_next(node\_before.get\_next())

node\_before.set\_next(new\_node)

if(new\_node.get\_next() is None):

self.\_\_tail=new\_node

else:

print(data\_before,"is not present in the Linked list")

def display(self):

temp=self.\_\_head

while(temp is not None):

print(temp.get\_data())

temp=temp.get\_next()

def find\_node(self,data):

temp=self.\_\_head

while(temp is not None):

if(temp.get\_data()==data):

return temp

temp=temp.get\_next()

return None

def delete(self,data):

node=self.find\_node(data)

if(node is not None):

if(node==self.\_\_head):

if(self.\_\_head==self.\_\_tail):

self.\_\_tail=None

self.\_\_head=node.get\_next()

else:

temp=self.\_\_head

while(temp is not None):

if(temp.get\_next()==node):

temp.set\_next(node.get\_next())

if(node==self.\_\_tail):

self.\_\_tail=temp

node.set\_next(None)

break

temp=temp.get\_next()

else:

print(data,"is not present in Linked list")

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

temp=self.\_\_head

msg=[]

while(temp is not None):

msg.append(str(temp.get\_data()))

temp=temp.get\_next()

msg=" ".join(msg)

msg="Linkedlist data(Head to Tail): "+ msg

return msg

class BakeHouse:

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

self.\_\_occupied\_table\_list=LinkedList()

def get\_occupied\_table\_list(self):

return self.\_\_occupied\_table\_list

def allocate\_table(self):

for i in range(1,11):

if self.\_\_occupied\_table\_list.find\_node(i):

continue

else:

if (i==1):

self.\_\_occupied\_table\_list.insert(i,None)

else:

self.\_\_occupied\_table\_list.insert(i,i-1)

return i

def deallocate\_table(self,table\_number):

self.\_\_occupied\_table\_list.delete(table\_number)

bakehouse=BakeHouse()

print(bakehouse.allocate\_table())

print(bakehouse.allocate\_table())

print(bakehouse.allocate\_table())

print(bakehouse.allocate\_table())

print(bakehouse.allocate\_table())

print(bakehouse.allocate\_table())

print(bakehouse.allocate\_table())

print(bakehouse.get\_occupied\_table\_list())

print(bakehouse.deallocate\_table(2))

print(bakehouse.deallocate\_table(3))

print(bakehouse.allocate\_table())

print(bakehouse.get\_occupied\_table\_list())

#Invoke the methods of BakeHouse class and test the program

#DSA-Assgn-9

class Node:

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

self.\_\_data=data

self.\_\_next=None

def get\_data(self):

return self.\_\_data

def set\_data(self,data):

self.\_\_data=data

def get\_next(self):

return self.\_\_next

def set\_next(self,next\_node):

self.\_\_next=next\_node

class LinkedList:

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

self.\_\_head=None

self.\_\_tail=None

def get\_head(self):

return self.\_\_head

def get\_tail(self):

return self.\_\_tail

def add(self,data):

new\_node=Node(data)

if(self.\_\_head is None):

self.\_\_head=self.\_\_tail=new\_node

else:

self.\_\_tail.set\_next(new\_node)

self.\_\_tail=new\_node

def insert(self,data,data\_before):

new\_node=Node(data)

if(data\_before==None):

new\_node.set\_next(self.\_\_head)

self.\_\_head=new\_node

if(new\_node.get\_next()==None):

self.\_\_tail=new\_node

else:

node\_before=self.find\_node(data\_before)

if(node\_before is not None):

new\_node.set\_next(node\_before.get\_next())

node\_before.set\_next(new\_node)

if(new\_node.get\_next() is None):

self.\_\_tail=new\_node

else:

print(data\_before,"is not present in the Linked list")

def display(self):

temp=self.\_\_head

while(temp is not None):

print(temp.get\_data())

temp=temp.get\_next()

def find\_node(self,data):

temp=self.\_\_head

while(temp is not None):

if(temp.get\_data()==data):

return temp

temp=temp.get\_next()

return None

def delete(self,data):

node=self.find\_node(data)

if(node is not None):

if(node==self.\_\_head):

if(self.\_\_head==self.\_\_tail):

self.\_\_tail=None

self.\_\_head=node.get\_next()

else:

temp=self.\_\_head

while(temp is not None):

if(temp.get\_next()==node):

temp.set\_next(node.get\_next())

if(node==self.\_\_tail):

self.\_\_tail=temp

node.set\_next(None)

break

temp=temp.get\_next()

else:

print(data,"is not present in Linked list")

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

temp=self.\_\_head

msg=[]

while(temp is not None):

msg.append(str(temp.get\_data()))

temp=temp.get\_next()

msg=" ".join(msg)

msg="Linkedlist data(Head to Tail): "+ msg

return msg

def reverse\_linkedlist(reverse\_list):

ls=[]

cur = reverse\_list.get\_head()

while cur:

ls.append(cur.get\_data())

cur = cur.get\_next()

ls.reverse()

reverse\_list = LinkedList()

for x in ls:

reverse\_list.add(x)

return reverse\_list

#Add different values to the linked list and test your program

reverse\_list=LinkedList()

reverse\_list.add(10)

reverse\_list.add(15)

reverse\_list.add(14)

reverse\_list.add(28)

reverse\_list.add(30)

reversed\_linkedlist=reverse\_linkedlist(reverse\_list)

reversed\_linkedlist.display()

#DSA-Assgn-10

class Node:

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

self.\_\_data=data

self.\_\_next=None

def get\_data(self):

return self.\_\_data

def set\_data(self,data):

self.\_\_data=data

def get\_next(self):

return self.\_\_next

def set\_next(self,next\_node):

self.\_\_next=next\_node

class LinkedList:

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

self.\_\_head=None

self.\_\_tail=None

def get\_head(self):

return self.\_\_head

def get\_tail(self):

return self.\_\_tail

def add(self,data):

new\_node=Node(data)

if(self.\_\_head is None):

self.\_\_head=self.\_\_tail=new\_node

else:

self.\_\_tail.set\_next(new\_node)

self.\_\_tail=new\_node

def insert(self,data,data\_before):

new\_node=Node(data)

if(data\_before==None):

new\_node.set\_next(self.\_\_head)

self.\_\_head=new\_node

if(new\_node.get\_next()==None):

self.\_\_tail=new\_node

else:

node\_before=self.find\_node(data\_before)

if(node\_before is not None):

new\_node.set\_next(node\_before.get\_next())

node\_before.set\_next(new\_node)

if(new\_node.get\_next() is None):

self.\_\_tail=new\_node

else:

print(data\_before,"is not present in the Linked list")

def display(self):

temp=self.\_\_head

while(temp is not None):

print(temp.get\_data())

temp=temp.get\_next()

def find\_node(self,data):

temp=self.\_\_head

while(temp is not None):

if(temp.get\_data()==data):

return temp

temp=temp.get\_next()

return None

def delete(self,data):

node=self.find\_node(data)

if(node is not None):

if(node==self.\_\_head):

if(self.\_\_head==self.\_\_tail):

self.\_\_tail=None

self.\_\_head=node.get\_next()

else:

temp=self.\_\_head

while(temp is not None):

if(temp.get\_next()==node):

temp.set\_next(node.get\_next())

if(node==self.\_\_tail):

self.\_\_tail=temp

node.set\_next(None)

break

temp=temp.get\_next()

else:

print(data,"is not present in Linked list")

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

temp=self.\_\_head

msg=[]

while(temp is not None):

msg.append(str(temp.get\_data()))

temp=temp.get\_next()

msg=" ".join(msg)

msg="Linkedlist data(Head to Tail): "+ msg

return msg

class Compartment:

def \_\_init\_\_(self,compartment\_name,no\_of\_passengers,total\_seats):

self.\_\_compartment\_name=compartment\_name

self.\_\_no\_of\_passengers=no\_of\_passengers

self.\_\_total\_seats=total\_seats

def get\_compartment\_name(self):

return self.\_\_compartment\_name

def get\_no\_of\_passengers(self):

return self.\_\_no\_of\_passengers

def get\_total\_seats(self):

return self.\_\_total\_seats

class Train:

def \_\_init\_\_(self,train\_name,compartment\_list):

self.\_\_train\_name=train\_name

self.\_\_compartment\_list=compartment\_list

def get\_compartment\_list(self):

return self.\_\_compartment\_list

def get\_train\_name(self):

return self.\_\_train\_name

def count\_compartments (self):

cur = self.\_\_compartment\_list.get\_head()

c = 0

while cur :

c +=1

cur = cur.get\_next()

return c

def check\_vacancy(self):

cur = self.\_\_compartment\_list.get\_head()

c = 0

while cur :

pass\_count = cur.get\_data().get\_no\_of\_passengers()

seats = cur.get\_data().get\_total\_seats()

if pass\_count <(seats//2):

c +=1

cur = cur.get\_next()

return c

#Implement the remaining methods of Train class here

#Use different values for compartment and test your program

compartment1=Compartment("SL",250,400)

compartment2=Compartment("2AC",125,280)

compartment3=Compartment("3AC",120,300)

compartment4=Compartment("FC",160,300)

compartment5=Compartment("1AC",100,210)

compartment\_list=LinkedList()

compartment\_list.add(compartment1)

compartment\_list.add(compartment2)

compartment\_list.add(compartment3)

compartment\_list.add(compartment4)

compartment\_list.add(compartment5)

train1=Train("Shatabdi",compartment\_list)

count=train1.count\_compartments()

print("The number of compartments in the train:",count)

vacancy\_count=train1.check\_vacancy()

print("The number of compartments which have more than 50% vacancy:",vacancy\_count)

#DSA-Exer-8

class Queue:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_rear=-1

self.\_\_front=0

def get\_max\_size(self):

return self.\_\_max\_size

def is\_full(self):

if(self.\_\_rear==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_front>self.\_\_rear):

return True

return False

def enqueue(self,data):

if(self.is\_full()):

print("Queue is full!!!")

else:

self.\_\_rear+=1

self.\_\_elements[self.\_\_rear]=data

def dequeue(self):

if(self.is\_empty()):

print("Queue is empty!!!")

else:

data=self.\_\_elements[self.\_\_front]

self.\_\_front+=1

return data

def display(self):

for index in range(self.\_\_front, self.\_\_rear+1):

print(self.\_\_elements[index])

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_front

while(index<=self.\_\_rear):

msg.append((str)(self.\_\_elements[index]))

index+=1

msg=" ".join(msg)

msg="Queue data(Front to Rear): "+msg

return msg

def split\_queue(num\_queue):

queue\_list=[]

odd = Queue(num\_queue.get\_max\_size())

even = Queue(num\_queue.get\_max\_size())

while not(num\_queue.is\_empty()):

num = num\_queue.dequeue()

if num&1 == 0:

even.enqueue(num)

else:

odd.enqueue(num)

queue\_list=[odd,even]

return queue\_list

# Enqueue different values to the queue and test your program

num\_queue=Queue(7)

num\_queue.enqueue(2)

num\_queue.enqueue(7)

num\_queue.enqueue(9)

num\_queue.enqueue(4)

num\_queue.enqueue(6)

num\_queue.enqueue(5)

num\_queue.enqueue(10)

q\_list=split\_queue(num\_queue)

q\_list[0].display()

q\_list[1].display()

#DSA-Exer-9

class Queue:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_rear=-1

self.\_\_front=0

def get\_max\_size(self):

return self.\_\_max\_size

def is\_full(self):

if(self.\_\_rear==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_front>self.\_\_rear):

return True

return False

def enqueue(self,data):

if(self.is\_full()):

print("Queue is full!!!")

else:

self.\_\_rear+=1

self.\_\_elements[self.\_\_rear]=data

def dequeue(self):

if(self.is\_empty()):

print("Queue is empty!!!")

else:

data=self.\_\_elements[self.\_\_front]

self.\_\_front+=1

return data

def display(self):

for index in range(self.\_\_front, self.\_\_rear+1):

print(self.\_\_elements[index])

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_front

while(index<=self.\_\_rear):

msg.append((str)(self.\_\_elements[index]))

index+=1

msg=" ".join(msg)

msg="Queue data(Front to Rear): "+msg

return msg

class People:

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

self.\_\_name=name

self.\_\_age=age

self.\_\_gender=gender

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

return(self.\_\_name+" "+str(self.\_\_age)+" "+self.\_\_gender)

def get\_age(self):

return self.\_\_age

def get\_gender(self):

return self.\_\_gender

def get\_name(self):

return self.\_\_name

@staticmethod

def check\_gender(people\_queue,gender):

gender\_queue = Queue(people\_queue.get\_max\_size())

while not(people\_queue.is\_empty()):

person = people\_queue.dequeue()

if person.get\_gender() == gender:

gender\_queue.enqueue(person)

return gender\_queue

people1=People("Jack",25,"Male")

people2=People("Tom",30,"Male")

people3=People("Asha",27,"Female")

people4=People("Henry",27,"Male")

people5=People("Tina",27,"Female")

people\_queue=Queue(5)

people\_queue.enqueue(people1)

people\_queue.enqueue(people2)

people\_queue.enqueue(people3)

people\_queue.enqueue(people4)

people\_queue.enqueue(***people5***)

result\_queue=People.check\_gender(people\_queue, "Male")

result\_queue.display()

#DSA-Exer-6

class Stack:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_top=-1

def get\_max\_size(self):

return self.\_\_max\_size

def is\_full(self):

if(self.\_\_top==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_top==-1):

return True

return False

def push(self,data):

if(self.is\_full()):

print("The stack is full!!")

else:

self.\_\_top+=1

self.\_\_elements[self.\_\_top]=data

def pop(self):

if(self.is\_empty()):

print("The stack is empty!!")

else:

data= self.\_\_elements[self.\_\_top]

self.\_\_top-=1

return data

def display(self):

if(self.is\_empty()):

print("The stack is empty")

else:

index=self.\_\_top

while(index>=0):

print(self.\_\_elements[index])

index-=1

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_top

while(index>=0):

msg.append((str)(self.\_\_elements[index]))

index-=1

msg=" ".join(msg)

msg="Stack data(Top to Bottom): "+msg

return msg

def find\_average(num\_list):

ls = []

for x in range (num\_list.get\_max\_size()-1):

ls.append(num\_list.pop())

ls.insert(0, sum(ls)/len(ls))

for x in range (len(ls)):

num\_list.push(ls.pop())

return num\_list

#Push different values to the stack and test your program

num\_list=Stack(7)

num\_list.push(78)

num\_list.push(65)

num\_list.push(92)

num\_list.push(46)

num\_list.push(89)

num\_list.push(71)

new\_stack=find\_average(num\_list)

new\_stack.display()

#DSA-Assgn-11

class Queue:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_rear=-1

self.\_\_front=0

def is\_full(self):

if(self.\_\_rear==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_front>self.\_\_rear):

return True

return False

def enqueue(self,data):

if(self.is\_full()):

print("Queue is full!!!")

else:

self.\_\_rear+=1

self.\_\_elements[self.\_\_rear]=data

def dequeue(self):

if(self.is\_empty()):

print("Queue is empty!!!")

else:

data=self.\_\_elements[self.\_\_front]

self.\_\_front+=1

return data

def display(self):

for index in range(self.\_\_front, self.\_\_rear+1):

print(self.\_\_elements[index])

def get\_max\_size(self):

return self.\_\_max\_size

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_front

while(index<=self.\_\_rear):

msg.append((str)(self.\_\_elements[index]))

index+=1

msg=" ".join(msg)

msg="Queue data(Front to Rear): "+msg

return msg

def merge\_queue(queue1,queue2):

if queue1.get\_max\_size() > queue2.get\_max\_size():

big = queue1

small = queue2

else:

big = queue2

small = queue1

merged\_queue = Queue(queue1.get\_max\_size()+queue2.get\_max\_size())

count = 1

for y in range (small.get\_max\_size()\*2):

if count%2 ==0:

merged\_queue.enqueue(queue2.dequeue())

else:

merged\_queue.enqueue(queue1.dequeue())

count +=1

for x in range (big.get\_max\_size()-small.get\_max\_size()):

merged\_queue.enqueue(big.dequeue())

return merged\_queue

#Enqueue different values to both the queues and test your program

queue1=Queue(3)

queue2=Queue(6)

queue1.enqueue(3)

queue1.enqueue(6)

queue1.enqueue(8)

queue2.enqueue('b')

queue2.enqueue('y')

queue2.enqueue('u')

queue2.enqueue('t')

queue2.enqueue('r')

queue2.enqueue('o')

merged\_queue=merge\_queue(queue1, queue2)

print("The elements in the merged queue are:")

merged\_queue.display()

#DSA-Assgn-12

class Stack:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_top=-1

def is\_full(self):

if(self.\_\_top==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_top==-1):

return True

return False

def push(self,data):

if(self.is\_full()):

print("The stack is full!!")

else:

self.\_\_top+=1

self.\_\_elements[self.\_\_top]=data

def pop(self):

if(self.is\_empty()):

print("The stack is empty!!")

else:

data= self.\_\_elements[self.\_\_top]

self.\_\_top-=1

return data

def display(self):

if(self.is\_empty()):

print("The stack is empty")

else:

index=self.\_\_top

while(index>=0):

print(self.\_\_elements[index])

index-=1

def get\_max\_size(self):

return self.\_\_max\_size

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_top

while(index>=0):

msg.append((str)(self.\_\_elements[index]))

index-=1

msg=" ".join(msg)

msg="Stack data(Top to Bottom): "+msg

return msg

class Queue:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_rear=-1

self.\_\_front=0

def is\_full(self):

if(self.\_\_rear==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_front>self.\_\_rear):

return True

return False

def enqueue(self,data):

if(self.is\_full()):

print("Queue is full!!!")

else:

self.\_\_rear+=1

self.\_\_elements[self.\_\_rear]=data

def dequeue(self):

if(self.is\_empty()):

print("Queue is empty!!!")

else:

data=self.\_\_elements[self.\_\_front]

self.\_\_front+=1

return data

def display(self):

for index in range(self.\_\_front, self.\_\_rear+1):

print(self.\_\_elements[index])

def get\_max\_size(self):

return self.\_\_max\_size

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_front

while(index<=self.\_\_rear):

msg.append((str)(self.\_\_elements[index]))

index+=1

msg=" ".join(msg)

msg="Queue data(Front to Rear): "+msg

return msg

class Ball:

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

self.\_\_color=color

self.\_\_name=name

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

return (self.\_\_color+" "+self.\_\_name)

def get\_color(self):

return self.\_\_color

def get\_name(self):

return self.\_\_name

class Game:

def \_\_init\_\_(self, ball\_stack):

self.ball\_container = ball\_stack

self.red\_balls\_container = Stack(2)

self.green\_balls\_container = Stack(2)

self.blue\_balls\_container = Stack(2)

self.yellow\_balls\_container = Stack(2)

def grouping\_based\_on\_color(self):

for i in range (self.ball\_container.get\_max\_size()):

ball = self.ball\_container.pop()

if ball.get\_color() == "Red":

self.red\_balls\_container.push(ball)

elif ball.get\_color() == "Green":

self.green\_balls\_container.push(ball)

elif ball.get\_color() == "Yellow":

self.yellow\_balls\_container.push(ball)

elif ball.get\_color() == "Blue":

self.blue\_balls\_container.push(ball)

def rearrange\_balls(self, color):

if color == "Red":

for x in range (2):

ball =self.red\_balls\_container.pop()

if ball.get\_name() == "A":

a = ball

else:

b = ball

self.red\_balls\_container.push(b)

self.red\_balls\_container.push(a)

elif color == "Blue":

for x in range (2):

ball =self.blue\_balls\_container.pop()

if ball.get\_name() == "A":

a = ball

else:

b = ball

self.blue\_balls\_container.push(b)

self.blue\_balls\_container.push(a)

def display\_ball\_details(self, color):

pass

#Use different values to test your program

ball1=Ball("Red","A")

ball2=Ball("Blue","B")

ball3=Ball("Yellow","B")

ball4=Ball("Blue","A")

ball5=Ball("Yellow","A")

ball6=Ball("Green","B")

ball7=Ball("Green","A")

ball8=Ball("Red","B")

#ball\_list=Stack(8)

# ball\_list.push(ball1)

# ball\_list.push(ball2)

# ball\_list.push(ball3)

# ball\_list.push(ball4)

# ball\_list.push(ball5)

# ball\_list.push(ball6)

# ball\_list.push(ball7)

# ball\_list.push(ball8)

#Create objects of Game class, invoke the methods and test the program

#DSA-Assgn-13

class Stack:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_top=-1

def is\_full(self):

if(self.\_\_top==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_top==-1):

return True

return False

def push(self,data):

if(self.is\_full()):

print("The stack is full!!")

else:

self.\_\_top+=1

self.\_\_elements[self.\_\_top]=data

def pop(self):

if(self.is\_empty()):

print("The stack is empty!!")

else:

data= self.\_\_elements[self.\_\_top]

self.\_\_top-=1

return data

def display(self):

if(self.is\_empty()):

print("The stack is empty")

else:

index=self.\_\_top

while(index>=0):

print(self.\_\_elements[index])

index-=1

def get\_max\_size(self):

return self.\_\_max\_size

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_top

while(index>=0):

msg.append((str)(self.\_\_elements[index]))

index-=1

msg=" ".join(msg)

msg="Stack data(Top to Bottom): "+msg

return msg

def change\_smallest\_value(number\_stack):

ls =[]

while not (number\_stack.is\_empty()):

pop = number\_stack.pop()

ls.append(pop)

mn = min(ls)

count = ls.count(mn)

for i in range (count):

ls.remove(mn)

number\_stack.push(mn)

for x in ls[::-1]:

number\_stack.push(x)

return number\_stack

#Add different values to the stack and test your program

number\_stack=Stack(8)

number\_stack.push(7)

number\_stack.push(8)

number\_stack.push(5)

number\_stack.push(66)

number\_stack.push(5)

print("Initial Stack:")

number\_stack.display()

change\_smallest\_value(number\_stack)

print("After the change:")

number\_stack.display()

#DSA-Assgn-14

class Queue:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_rear=-1

self.\_\_front=0

def is\_full(self):

if(self.\_\_rear==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_front>self.\_\_rear):

return True

return False

def enqueue(self,data):

if(self.is\_full()):

print("Queue is full!!!")

else:

self.\_\_rear+=1

self.\_\_elements[self.\_\_rear]=data

def dequeue(self):

if(self.is\_empty()):

print("Queue is empty!!!")

else:

data=self.\_\_elements[self.\_\_front]

self.\_\_front+=1

return data

def display(self):

for index in range(self.\_\_front, self.\_\_rear+1):

print(self.\_\_elements[index])

def get\_max\_size(self):

return self.\_\_max\_size

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_front

while(index<=self.\_\_rear):

msg.append((str)(self.\_\_elements[index]))

index+=1

msg=" ".join(msg)

msg="Queue data(Front to Rear): "+msg

return msg

def check\_numbers(number\_queue):

solution\_queue1 = Queue(number\_queue.get\_max\_size())

while not (number\_queue.is\_empty()):

pop = number\_queue.dequeue()

count = 0

for x in range (1,11):

if pop%x == 0:

count+=1

if count == 10:

solution\_queue1.enqueue(pop)

return solution\_queue1

#Add different values to the queue and test your program

number\_queue=Queue(5)

number\_queue.enqueue(13983)

number\_queue.enqueue(10080)

number\_queue.enqueue(7113)

number\_queue.enqueue(2520)

number\_queue.enqueue(2500)

check\_numbers(number\_queue)

#DSA-Assgn-15

class Queue:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_rear=-1

self.\_\_front=0

def is\_full(self):

if(self.\_\_rear==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_front>self.\_\_rear):

return True

return False

def enqueue(self,data):

if(self.is\_full()):

print("Queue is full!!!")

else:

self.\_\_rear+=1

self.\_\_elements[self.\_\_rear]=data

def dequeue(self):

if(self.is\_empty()):

print("Queue is empty!!!")

else:

data=self.\_\_elements[self.\_\_front]

self.\_\_front+=1

return data

def display(self):

for index in range(self.\_\_front, self.\_\_rear+1):

print(self.\_\_elements[index])

def get\_max\_size(self):

return self.\_\_max\_size

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

msg=[]

index=self.\_\_front

while(index<=self.\_\_rear):

msg.append((str)(self.\_\_elements[index]))

index+=1

msg=" ".join(msg)

msg="Queue data(Front to Rear): "+msg

return msg

#Implement Job, Employee and Company classes here

class Job:

def \_\_init\_\_(self,name,time\_needed):

self.\_\_name = name

self.\_\_time\_needed = time\_needed

self.\_\_time\_elapsed = 0

def get\_name(self):

return self.\_\_name

def get\_time\_needed(self):

return self.\_\_time\_needed

def get\_time\_elapsed(self):

return self.\_\_time\_elapsed

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

return self.\_\_name+' '+str(self.\_\_time\_needed)

def elapsed\_time(self,no\_of\_mins):

self.\_\_time\_elapsed += no\_of\_mins

if self.\_\_time\_elapsed >= self.\_\_time\_needed:

return True

return False

class Company:

def \_\_init\_\_(self,emp\_list):

self.\_\_employees = emp\_list

self.\_\_pending\_jobs = Queue(10)

def get\_employees(self):

return self.\_\_employees

def get\_pending\_jobs(self):

return self.\_\_pending\_jobs

def allocate\_new\_job(self,job):

in\_job=0

for employee in self.\_\_employees:

if employee.get\_allocated\_job() is None:

employee.set\_allocated\_job(job)

break

else:

in\_job+=1

if in\_job == len(self.\_\_employees):

self.\_\_pending\_jobs.enqueue(job)

pass

def elapsed\_time(self,no\_of\_mins):

completed\_list = []

for employe in self.\_\_employees:

job\_status = employe.elapsed\_time(no\_of\_mins)

if employe.get\_allocated\_job() is None:

self.allocate\_new\_job(self.get\_pending\_jobs().dequeue())

if job\_status is not None:

completed\_list.append(job\_status)

if len(completed\_list)!=0:

return completed\_list

else:

return None

class Employee:

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

self.\_\_name = name

self.\_\_allocated\_job = None

def get\_name(self):

return self.\_\_name

def get\_allocated\_job(self):

return self.\_\_allocated\_job

def set\_allocated\_job(self, allocated\_job):

self.\_\_allocated\_job = allocated\_job

def elapsed\_time(self,no\_of\_mins):

self.\_\_allocated\_job.elapsed\_time(no\_of\_mins)

if self.\_\_allocated\_job.get\_time\_needed() <= self.\_\_allocated\_job.get\_time\_elapsed():

ret = self.\_\_allocated\_job

self.\_\_allocated\_job = None

return ret

else:

return None

#Change the values and test your programH

emp1=Employee("Ken")

emp2=Employee("Henry")

emp3=Employee("Jack")

emp4=Employee("Hen")

emp5=Employee("Jill")

emp\_list=[emp1,emp2,emp3,emp4,emp5]

company=Company(emp\_list)

job1=Job("job1",50)

job2=Job("job2",45)

job3=Job("job3",35)

job4=Job("job4",400)

job5=Job("job5",30)

job6=Job("job6",30)

job7=Job("job7",50)

job8=Job("job8",25)

company.allocate\_new\_job(job1)

company.allocate\_new\_job(job2)

company.allocate\_new\_job(job3)

company.allocate\_new\_job(job4)

company.allocate\_new\_job(job5)

company.allocate\_new\_job(job6)

company.allocate\_new\_job(job7)

company.allocate\_new\_job(job8)

print("Initial allocation:")

for emp in company.get\_employees():

print(emp.get\_name(),"is allocated",emp.get\_allocated\_job().get\_name())

print()

print("Pending Jobs:")

company.get\_pending\_jobs().display()

completed\_jobs=company.elapsed\_time(30)

'''print("Completed Jobs :")

for job in completed\_jobs:

print(job.name)'''

print("After completion:")

for emp in company.get\_employees():

print(emp.get\_name(),"needs", emp.get\_allocated\_job().get\_time\_needed()-emp.get\_allocated\_job().get\_time\_elapsed(),"more minutes for",emp.get\_allocated\_job().get\_name())

print()

print("Pending Jobs:")

company.get\_pending\_jobs().display()

completed\_jobs=company.elapsed\_time(10)

print("After completion:")

for emp in company.get\_employees():

print(emp.get\_name(),"needs", emp.get\_allocated\_job().get\_time\_needed()-emp.get\_allocated\_job().get\_time\_elapsed(),"more minutes for",emp.get\_allocated\_job().get\_name())

print()

print("Pending Jobs:")

company.get\_pending\_jobs().display()

#DSA-Tryout

import random

def find\_it(num,element\_list):

#Remove pass and write the logic to search num in element\_list using linear search algorithm

#Return the total number of guesses made

c = 1

for i in element\_list:

if i==num:

break

c+=1

return c

#Initializes a list with values 1 to n in random order and returns it

def initialize\_list\_of\_elements(n):

list\_of\_elements=[]

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

list\_of\_elements.append(i)

mid=n//2

for j in range(0,n):

index1=random.randrange(0,mid)

index2=random.randrange(mid,n)

num1=list\_of\_elements[index1]

list\_of\_elements[index1]=list\_of\_elements[index2]

list\_of\_elements[index2]=num1

return list\_of\_elements

def play(n):

# Step 1: Invoke initialize\_list\_of\_elements() by passing n

# Step 2: Generate a random number from the list of elements. The number should be between 1 and n (both inclusive)

# Step 3: Invoke find\_it() by passing the number generated at Step 2 and list generated at Step 1 and display the return value

# Remove pass and write the code to implement the above three steps.

ls = initialize\_list\_of\_elements(n)

num = random.randrange(n)

return find\_it(num,ls)

#Pass different values to play() and observe the output

play(400)

#DSA-Tryout

import random

def find\_it(num,element\_list):

#Remove pass and copy the solution earlier written for this function here

#Modify it, if required

c = 1

for i in element\_list:

if i==num:

break

c+=1

return c

def initialize\_list\_of\_elements(n):

#Modify the code to initialize the list of elements in ascending order

#Try with descending order also

list\_of\_elements=[]

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

list\_of\_elements.append(i)

mid=n//2

for j in range(0,n):

index1=random.randrange(0,mid)

index2=random.randrange(mid,n)

num1=list\_of\_elements[index1]

list\_of\_elements[index1]=list\_of\_elements[index2]

list\_of\_elements[index2]=num1

return list\_of\_elements

def play(n):

#Remove pass and copy the solution earlier written for this function here

ls = initialize\_list\_of\_elements(n)

num = random.randrange(n)

return find\_it(num,ls)

#Pass different values to play() and observe the output

play(400)

#DSA-Tryout

import random

def find\_it(num,element\_list):

#Remove pass and write the logic to search num in element\_list using binary search algorithm

#Return the total number of guesses made

i = 0

j = len(element\_list)-1

c=1

while(i<j-1):

mid = (i+j)//2

if element\_list[mid]==num:

return c

elif element\_list[mid]<num:

i = mid

else:

j = mid

c+=1

return c

#Initializes a list with values 1 to n in ascending order and returns it

def initialize\_list\_of\_elements(n):

list\_of\_elements=[]

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

list\_of\_elements.append(i)

return list\_of\_elements

def play(n):

# Step 1: Invoke initialize\_list\_of\_elements() by passing n

# Step 2: Generate a random number from the list of elements. The number should be between 1 and n (both inclusive)

# Step 3: Invoke find\_it() by passing the number generated at Step 2 and list generated at Step 1 and display the return value

# Remove pass and write the code to implement the above three steps.

ls = initialize\_list\_of\_elements(n)

num = random.randrange(n)

return find\_it(num,ls)

#Pass different values to play() and observe the output

play(400)

#DSA-Tryout

import random

from timeit import default\_timer as timer

def find\_it\_linear(num,element\_list):

#remove pass and copy the code written earlier for linear search

c = 1

for i in element\_list:

if i==num:

break

c+=1

return c

def find\_it\_binary(num,element\_list):

#remove pass and copy the code written earlier for binary search

i = 0

j = len(element\_list)-1

c=1

while(i<j-1):

mid = (i+j)//2

if element\_list[mid]==num:

return c

elif element\_list[mid]<num:

i = mid

else:

j = mid

c+=1

return c

#Initializes a list with values 1 to n in ascending order and returns it

def initialize\_list\_of\_elements(n):

list\_of\_elements=[]

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

list\_of\_elements.append(i)

return list\_of\_elements

def play(n):

list\_of\_elements=initialize\_list\_of\_elements(n)

rand\_index=random.randrange(0,len(list\_of\_elements))

rand\_num=list\_of\_elements[rand\_index]

print("Number to be guessed:",rand\_num)

start=timer()

print("No. of guesses made using linear search:",find\_it\_linear(rand\_num,list\_of\_elements))

end=timer()

print("Linear Search:Execution time in seconds:{0:.5f}".format( (end-start)))

start=timer()

print("No. of guesses made using binary search:",find\_it\_binary(rand\_num,list\_of\_elements))

end=timer()

print("Binary Search:Execution time in seconds:{0:.5f}".format( (end-start)))

#Pass different values to play() and observe the output

play(40000)

#DSA-Exer-15

def pattern\_search(text, pattern):

l1 = len(pattern)

c=0

for i in range(0,len(text)-l1):

if pattern==text[i:i+l1]:

c+=1

return c

#Use different values for text and pattern and test your program

text = "MESMERIZING MESSAGE"

pattern = "MES"

result=pattern\_search(text, pattern)

print("The given text:",text)

print("Pattern:",pattern)

print("No. of occurrences of the pattern :",result)

#DSA-Exer-16

def find\_decreasing\_start(list1,start,end):

for i in range (start,end):

if list1[i] > list1[i+1]:

return list1.index(list1[i+1])

#Use different values for list1 and test your program

list1=[1,4,7,8,9,5,4]

start=0

end=len(list1)-1

result=find\_decreasing\_start(list1,start,end)

print("The index position at which the increasing array starts decreasing is:",result)

#DSA-Assgn-16

class Queue:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_rear=-1

self.\_\_front=0

def is\_full(self):

if(self.\_\_rear==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_front>self.\_\_rear):

return True

return False

def enqueue(self,data):

if(self.is\_full()):

print("Queue is full!!!")

else:

self.\_\_rear+=1

self.\_\_elements[self.\_\_rear]=data

def dequeue(self):

if(self.is\_empty()):

print("Queue is empty!!!")

else:

data=self.\_\_elements[self.\_\_front]

self.\_\_front+=1

return data

def display(self):

for index in range(self.\_\_front, self.\_\_rear+1):

print(self.\_\_elements[index])

def get\_max\_size(self):

return self.\_\_max\_size

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_front

while(index<=self.\_\_rear):

msg.append((str)(self.\_\_elements[index]))

index+=1

msg=" ".join(msg)

msg="Queue data(Front to Rear): "+msg

return msg

class Stack:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_top=-1

def is\_full(self):

if(self.\_\_top==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_top==-1):

return True

return False

def push(self,data):

if(self.is\_full()):

print("The stack is full!!")

else:

self.\_\_top+=1

self.\_\_elements[self.\_\_top]=data

def pop(self):

if(self.is\_empty()):

print("The stack is empty!!")

else:

data= self.\_\_elements[self.\_\_top]

self.\_\_top-=1

return data

def display(self):

if(self.is\_empty()):

print("The stack is empty")

else:

index=self.\_\_top

while(index>=0):

print(self.\_\_elements[index])

index-=1

def get\_max\_size(self):

return self.\_\_max\_size

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_top

while(index>=0):

msg.append((str)(self.\_\_elements[index]))

index-=1

msg=" ".join(msg)

msg="Stack data(Top to Bottom): "+msg

return msg

def separate\_boxes(box\_stack):

#def separate\_boxes(box\_stack):

que = Queue(box\_stack.get\_max\_size())

lst = []

while not (box\_stack.is\_empty()):

poped = box\_stack.pop()

if poped in ["Red","Blue","Green"]:

lst.append(poped)

else:

que.enqueue(poped)

while len(lst)!=0:

box\_stack.push(lst.pop())

return que

#Remove pass and write your logic here

#Use different values for stack and test your program

box\_stack=Stack(8)

box\_stack.push("Red")

box\_stack.push("Magenta")

box\_stack.push("Yellow")

box\_stack.push("Red")

box\_stack.push("Orange")

box\_stack.push("Green")

box\_stack.push("White")

box\_stack.push("Purple")

print("Boxes in the stack:")

box\_stack.display()

result=separate\_boxes(box\_stack)

print()

print("Boxes in the stack after modification:")

box\_stack.display()

print("Boxes in the queue:")

result.display()

#DSA-Assgn-17

def find\_matches(country\_name):

match = []

for detail in match\_list:

if (detail[:3]) == country\_name:

match.append(detail)

return match

def max\_wins():

cham,wor,t20,tmp,ret = [],[],[],[],dict()

for detail in match\_list:

split\_detail = detail.split(":")

if split\_detail[1] == "CHAM":

cham.append(split\_detail[0])

tmp.append(split\_detail[3])

if len(tmp) !=0:

while (min(tmp)!=max(tmp)):

cham.pop(tmp.index(min(tmp)))

tmp.pop(tmp.index(min(tmp)))

ret ["CHAM"] = cham

tmp = []

for detail in match\_list:

split\_detail = detail.split(":")

if split\_detail[1] == "T20":

t20.append(split\_detail[0])

tmp.append(split\_detail[3])

if len(tmp) !=0:

while (min(tmp)!=max(tmp)):

t20.pop(tmp.index(min(tmp)))

tmp.pop(tmp.index(min(tmp)))

ret ["T20"] = t20

tmp = []

for detail in match\_list:

split\_detail = detail.split(":")

if split\_detail[1] == "WOR":

wor.append(split\_detail[0])

tmp.append(split\_detail[3])

if len(tmp) !=0:

while (min(tmp)!=max(tmp)):

wor.pop(tmp.index(min(tmp)))

tmp.pop(tmp.index(min(tmp)))

ret ["WOR"] = wor

return ret

def find\_winner(country1,country2):

c1=c2 = 0

for detail in match\_list:

split\_detail = detail.split(":")

print (split\_detail[1])

if split\_detail[0] == country1:

c1+=int(split\_detail[3])

elif split\_detail[0] == country2:

c2+=int(split\_detail[3])

if c1 > c2:

return country1

elif c2 > c1:

return country2

return "Tie"

#Consider match\_list to be a global variable

match\_list=["AUS:CHAM:5:2","AUS:WOR:2:1","ENG:WOR:2:0","IND:T20:5:3","IND:WOR:2:1","PAK:WOR:2:0","PAK:T20:5:1","SA:WOR:2:0","SA:CHAM:5:1","SA:T20:5:0"]

#Pass different values to each function and test your program

print("The match status list details are:")

print(match\_list)

print()

#DSA-Assgn-18

def find\_unknown\_words(text,vocabulary):

#Remove pass and write your logic here

text = text.split(" ")

ls = []

for i in text:

if i not in vocabulary:

ls.append(i)

if len(ls)==0:

return -1

return set(ls)

#Pass different values of text and vocabulary to the function and test your program

text="The sun rises in the east and sets in the west."

vocabulary = ["sun","in","rises","the","east"]

unknown\_words=find\_unknown\_words(text,vocabulary)

print("The unknown words in the file are:",unknown\_words)

#DSA-Assgn-19

def last\_instance( num\_list, start, end, key):

num\_list = num\_list[::-1]

try:

i = len(num\_list)-num\_list.index(key)-1

except ValueError:

i = -1

return i

num\_list=[1,1,2,2,3,4,5,5,5,5]

start=0

end=len(num\_list)-1

key=5 #Number to be searched

#Pass different values for num\_list, start,end and key and test your program

result=last\_instance(num\_list, start,end,key)

if(result!=-1):

print("The index position of the last occurrence of the number:",result)

else:

print("Number not found")

#DSA-Exer-17

def swap(num\_list, first\_index, second\_index):

#Remove pass and write your logic here

#As python lists are mutable, num\_list need not be returned after swapping

temp = num\_list[first\_index]

num\_list[first\_index] = num\_list[second\_index]

num\_list[second\_index] = temp

#Pass different values to the function and test your program

num\_list=[2,3,89,45,67]

print("List before swapping:",num\_list)

swap(num\_list, 1, 2)

print("List after swapping:",num\_list)

#DSA-Exer-18

def find\_next\_min(num\_list,start\_index):

#Remove pass and write the logic to find the minimum element in a sub-list and return the index of the identified element in the num\_list.

#start\_index indicates the start index of the sub-list

return num\_list.index(min(num\_list[start\_index:]))

#Pass different values to the function and test your program

num\_list=[10,2,100,67]

start\_index=1

print("Index of the next minimum element is", find\_next\_min(num\_list,start\_index))

#DSA-Exer-19

def swap(num\_list, first\_index, second\_index):

temp = num\_list[first\_index]

num\_list[first\_index] = num\_list[second\_index]

num\_list[second\_index] = temp

def find\_next\_min(num\_list,start\_index):

return num\_list.index(min(num\_list[start\_index:]))

def selection\_sort(num\_list):

for x in range(len(num\_list)):

swap (num\_list,x,find\_next\_min(num\_list, x))

#Pass different values to the function and test your program

num\_list=[8,2,19,34,23, 67, 91]

print("Before sorting;",num\_list)

selection\_sort(num\_list)

print("After sorting:",num\_list)

#DSA-Exer-20

def swap(num\_list, first\_index, second\_index):

temp = num\_list[first\_index]

num\_list[first\_index] = num\_list[second\_index]

num\_list[second\_index] = temp

def find\_next\_min(num\_list,start\_index):

return num\_list.index(min(num\_list[start\_index:]))

def selection\_sort(num\_list):

for x in range(len(num\_list)):

swap (num\_list,x,find\_next\_min(num\_list, x))

return (len(num\_list)-1)

def bubble\_sort(num\_list):

total\_no\_of\_passes=0

end\_index=len(num\_list)

for index1 in range(0, end\_index-1):

swapped=False

total\_no\_of\_passes+=1

for index2 in range(0, (end\_index-index1-1)):

if(num\_list[index2]>num\_list[index2+1]):

swap(num\_list, index2, index2+1)

swapped=True;

if(swapped==False):

break

return total\_no\_of\_passes

num\_list=[8,2,19,34,23, 67, 91]

#num\_list=[91,8,19,23,34,67,2]

print("Selection Sort - No. of passes:",selection\_sort(num\_list))

num\_list=[8,2,19,34,23, 67, 91]

#num\_list=[91,8,19,23,34,67,2]

print("Bubble Sort - No. of passes:",bubble\_sort(num\_list))’

‘

#DSA-Exer-21

def merge\_sort(num\_list):

high = len(num\_list)

low = 0

if high-1 == low:

return num\_list

else:

mid = high//2

sorted\_list = merge (merge\_sort(num\_list[:mid]),merge\_sort(num\_list[mid:]))

return sorted\_list

# Remove pass and write the logic here to return the sorted list

def merge(left\_list,right\_list):

i,j = 0,0

sorted\_list = []

while i < len(left\_list) and j < len(right\_list):

if left\_list[i] <= right\_list[j]:

sorted\_list.append(left\_list[i])

i+=1

else:

sorted\_list.append (right\_list[j])

j+=1

for i in left\_list:

if i not in sorted\_list:

sorted\_list.append (i)

for j in right\_list:

if j not in sorted\_list :

sorted\_list.append (j)

return sorted\_list

# Remove pass and write the logic to merge the elements in the left\_list and right\_list and return the sorted list

num\_list=[34, 67, 8, 19, 2, 23, 1, 91]

print("Before sorting:",num\_list)

sorted\_list = merge\_sort(num\_list)

print("After sorting:",sorted\_list)

#DSA-Exer-22

def order\_heights(student\_list,height\_list):

for y in range (2):

for x in range (len(student\_list)-1):

if height\_list[x] > height\_list[x+1]:

height\_list[x],height\_list[x+1] = height\_list[x+1], height\_list[x]

student\_list[x],student\_list[x+1] = student\_list[x+1],student\_list[x]

return[student\_list,height\_list]

#Pass different values to the function and test your program

student\_list=["Santa","Tris","Arun","Rachel","John"]

height\_list=[132.7,129.2,135,130.6,140]

print("Initial student details :")

print("The students:",student\_list)

print("Their heights:",height\_list)

print()

result=order\_heights(student\_list,height\_list)

print("After arranging the students in the order of their height:")

print("The students :",result[0])

print("Their heights:",result[1])

#DSA-Exer-23

def arrange\_tickets(tickets\_list):

ls = [0]\*20

for num in tickets\_list:

ls.pop(int(num[1:])-1)

ls.insert(int(num[1:])-1,num)

for x in range (len(ls)):

if ls[x]==0:

ls[x]="V"

first\_ten = ls[:10]

next\_ten = ls[10:]

while "V" in next\_ten:

next\_ten.remove("V")

for i in range (len(first\_ten)):

if first\_ten[i] == "V":

first\_ten.pop(i)

first\_ten.insert(i,next\_ten.pop(0))

return first\_ten

tickets\_list = ['T5','T7','T1','T2','T8','T15','T17','T19','T6','T12','T13']

print("Ticket ids of all the available students :")

print(tickets\_list)

result=arrange\_tickets(tickets\_list)

print()

print("Ticket ids of the ten students in Group-1:")

print(result)

#DSA-Assgn-20

#Implement Item class here

class Item:

def \_\_init\_\_(self,item\_name,author\_name,published\_year):

self.\_\_item\_name = item\_name

self.\_\_author\_name = author\_name

self.\_\_published\_year = published\_year

def get\_author\_name(self):

return self.\_\_author\_name

def get\_item\_name(self):

return self.\_\_item\_name

def get\_published\_year(self):

return self.\_\_published\_year

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

return (self.\_\_item\_name+'\n\t'+self.\_\_author\_name+'\n\t\t'+str(self.\_\_published\_year))

class Library:

def \_\_init\_\_(self,item\_list):

self.\_\_item\_list = item\_list

def get\_item\_list(self):

return self.\_\_item\_list

def add\_new\_items(self,new\_item\_list):

self.\_\_item\_list=self.sort\_item\_list\_by\_author(self.\_\_item\_list+new\_item\_list)

def sort\_item\_list\_by\_author(self,new\_item\_list):

new\_item\_list.sort(key=lambda author:author.get\_author\_name().replace(" ",""), reverse=False)

return new\_item\_list

def sort\_items\_by\_published\_year(self):

self.\_\_item\_list.sort(key=lambda author:(author.get\_published\_year(),author.get\_author\_name().replace(" ","")))

#Implement Library class here

#Use different values for item and test your program

item1=Item("A Mission In Kashmir","Andrew Whitehead",1995)

item2=Item("A Passage of India","E.M.Forster",2012)

item3=Item("A new deal for Asia","Mahathir Mohammad",1999)

item4=Item("A Voice of Freedom","Nayantara Sehgal",2001)

item5=Item("A pair of blue eyes","Thomas Hardy",1998 )

item\_list=[item1,item2,item3,item4,item5]

library=Library(item\_list)

print("The current items in the library are:")

for item in library.get\_item\_list():

print(item)

item11=Item("Broken Wing","Sarojini Naidu",2012)

item12=Item("Guide","R.K.Narayanan",2001)

item13=Item("Indian Summers","John Mathews",2001)

item14=Item("Innocent in Death","J.D.Robb",2010)

item15=Item("Life of Pi","Yann Martel",2010 )

item16=Item("Sustainability","Johny",2016)

item17=Item("Look Ahead","E.M.Freddy",2012 )

new\_item\_list=[item11,item12,item13,item14,item15,item16,item17]

print()

print("The new items to be added are:")

for item in new\_item\_list:

print(item)

new\_item\_list\_sorted=library.sort\_item\_list\_by\_author(new\_item\_list)

print()

print("The new items after sorting based on the author name are:")

for item in new\_item\_list\_sorted:

print(item)

library.add\_new\_items(new\_item\_list\_sorted)

print()

print("The final set of items after adding the new item list are:")

for item in library.get\_item\_list():

print(item)

library.sort\_items\_by\_published\_year()

print()

print("The items sorted based on the increasing order of their published year:")

for item in library.get\_item\_list():

print(item)

#DSA-Assgn-21

class Queue:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_rear=-1

self.\_\_front=0

def is\_full(self):

if(self.\_\_rear==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_front>self.\_\_rear):

return True

return False

def enqueue(self,data):

if(self.is\_full()):

print("Queue is full!!!")

else:

self.\_\_rear+=1

self.\_\_elements[self.\_\_rear]=data

def dequeue(self):

if(self.is\_empty()):

print("Queue is empty!!!")

else:

data=self.\_\_elements[self.\_\_front]

self.\_\_front+=1

return data

def display(self):

for index in range(self.\_\_front, self.\_\_rear+1):

print(self.\_\_elements[index])

def get\_max\_size(self):

return self.\_\_max\_size

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_front

while(index<=self.\_\_rear):

msg.append((str)(self.\_\_elements[index]))

index+=1

msg=" ".join(msg)

msg="Queue data(Front to Rear): "+msg

return msg

class Stack:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_top=-1

def is\_full(self):

if(self.\_\_top==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_top==-1):

return True

return False

def push(self,data):

if(self.is\_full()):

print("The stack is full!!")

else:

self.\_\_top+=1

self.\_\_elements[self.\_\_top]=data

def pop(self):

if(self.is\_empty()):

print("The stack is empty!!")

else:

data= self.\_\_elements[self.\_\_top]

self.\_\_top-=1

return data

def display(self):

if(self.is\_empty()):

print("The stack is empty")

else:

index=self.\_\_top

while(index>=0):

print(self.\_\_elements[index])

index-=1

def get\_max\_size(self):

return self.\_\_max\_size

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_top

while(index>=0):

msg.append((str)(self.\_\_elements[index]))

index-=1

msg=" ".join(msg)

msg="Stack data(Top to Bottom): "+msg

return msg

#Global variables

flight\_details=["AI890:BAN:MUM:1400","AI678:BAN:LON:1200","AI345:BAN:CAN:1410","AF780:BAN:AGF:1340","AI001:BAN:AUS:1500","AI404:BAN:NY:1220"]

passenger\_details\_dict=\

{"LW101":["Amanda","AI678","C7",25],"LW103":["John","AI345","A2",10],"LW107":["Alex","AI678","G5",12],\

"TW700":["Hary","AF780","D2",26],"LW167":["Kate","AI001","G3",25],"LT890":["Wade","AI404","G3",25],\

"TW677":["Preet","AF780","D3",25],"LA106":["Henry","AI001","B5",25.5],"LA104":["Ajay","AI001","A7",23],\

"LW202":["Amy","AI345","C3",24.5],"LT673":["Susan","AI404","J8",5],"TW709":["Tris","AF780","H5",22.5],\

"LA188":["Cameron","AI890","H4",22],"LA902":["Scofield","AI678","G4",23],"TW767":["Pom","AF780","H4",2],\

"LW787":["Burrows","AI890","B4",29],"LW898":["Sara","AI678","E4",14],"LW104":["Williams","AI890","C4",10] }

def find\_flights(flight\_time):

ls = []

for i in flight\_details:

if int(i[14:])>=flight\_time and flight\_time<=int(i[14:])+200:

ls.append(i)

return ls

def sort\_flight\_list(flight\_list):

flight\_list.sort(key = lambda x:x[14:])

return flight\_list

def get\_passenger\_details(flight\_detail):

#Remove pass and write your logic here

ls = []

for key in passenger\_details\_dict:

if passenger\_details\_dict[key][1] == flight\_detail[:5]:

ls.append(key)

return ls

def security\_check(passenger\_pnr\_list):

#Remove pass and write your logic here

ls = []

for key in passenger\_pnr\_list:

if passenger\_details\_dict[key][3]<=25 and passenger\_details\_dict[key][3]>=0:

ls.append(key)

return ls

def sort\_passengers(passenger\_pnr\_list):

passenger\_pnr\_list.sort(key = lambda x: passenger\_details\_dict[x][2])

return passenger\_pnr\_list

def boarding(passenger\_pnr\_list):

#Remove pass and write your logic here

pass

def seating(passenger\_queue):

#Remove pass and write your logic here

pass

print(find\_flights(1400))

print("The flight details :")

print(flight\_details)

print()

print("The passenger details at the airport:")

print(passenger\_details\_dict)

print()

time=1130

print("Details of the flight between the timings",time,"and",time+200,"are:")

flight\_list=find\_flights(time)

print(flight\_list)

flight\_list=sort\_flight\_list(flight\_list)

print(flight\_list)

print()

print("Details of the passengers boarding the flights between the timings ",time,"and",(time+200),"are:")

print(flight\_list)

# for i in range(0,len(flight\_list)):

# flight\_data=flight\_list[i].split(':')

# flight\_name=flight\_data[0]

# passenger\_pnr\_list=get\_passenger\_details(flight\_list[i])

# print("PNR details of the passengers boarding the flight",flight\_name,":")

# print(passenger\_pnr\_list)

# print()

# updated\_passenger\_pnr\_list=security\_check(passenger\_pnr\_list)

# print("PNR details of the passengers of flight",flight\_name," whose baggage has been cleared:")

# print(updated\_passenger\_pnr\_list)

# sorted\_passenger\_pnr\_list=sort\_passengers(updated\_passenger\_pnr\_list)

# print("PNR details of the passengers of flight",flight\_name," sorted based on seating number:")

# print(sorted\_passenger\_pnr\_list)

# print()

# print("The PNR details of the passengers at the queue",flight\_name,":")

# passenger\_queue=boarding(updated\_passenger\_pnr\_list)

# passenger\_queue.display()

# print()

# seating\_stack=seating(passenger\_queue)

# print("The PNR details of the passengers in the flight",flight\_name,":")

# seating\_stack.display()

#DSA-Assgn-22

class Stack:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_top=-1

def is\_full(self):

if(self.\_\_top==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_top==-1):

return True

return False

def push(self,data):

if(self.is\_full()):

print("The stack is full!!")

else:

self.\_\_top+=1

self.\_\_elements[self.\_\_top]=data

def pop(self):

if(self.is\_empty()):

print("The stack is empty!!")

else:

data= self.\_\_elements[self.\_\_top]

self.\_\_top-=1

return data

def display(self):

if(self.is\_empty()):

print("The stack is empty")

else:

index=self.\_\_top

while(index>=0):

print(self.\_\_elements[index])

index-=1

def get\_max\_size(self):

return self.\_\_max\_size

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_top

while(index>=0):

msg.append((str)(self.\_\_elements[index]))

index-=1

msg=" ".join(msg)

msg="Stack data(Top to Bottom): "+msg

return msg

def merge\_stack(stack1,stack2):

array = Stack(stack1.get\_max\_size()+stack2.get\_max\_size())

ls=[]

while not(stack1.is\_empty()):

ls.append(stack1.pop())

while not(stack2.is\_empty()):

ls.append(stack2.pop())

ls.sort()

for x in ls:

array.push(x)

return array

#Pass different values to the function and test your program

stack2=Stack(3)

stack2.push(9)

stack2.push(11)

stack2.push(15)

stack1=Stack(4)

stack1.push(3)

stack1.push(7)

stack1.push(10)

stack1.push(21)

print("The elements in stack1 are:")

stack1.display()

print("The elements in stack2 are:")

stack2.display()

print()

output\_stack=merge\_stack(stack1, stack2)

print("The elements in the output stack are:")

output\_stack.display()

#DSA-Exer-24

def make\_change(denomination\_list, amount):

denomination\_list.sort(reverse = True)

c=0

for i in denomination\_list:

if i>amount:

continue

c += amount//i

amount = amount%i

#print(amount,i)

if amount<=0:

break

if amount!=0:

return -1

return c

#Pass different values to the function and test your program

amount= 30

denomination\_list = [20,2,10]

print(make\_change(denomination\_list, amount))

#DSA-Exer-25

def find\_maximum\_activities(activity\_list,start\_time\_list, finish\_time\_list):

#Remove pass and write your logic here

ans = []

ls = []

l = len(activity\_list)

for i in range(l):

ls.append([activity\_list[i],start\_time\_list[i],finish\_time\_list[i]])

ls.sort(key = lambda x : x[2])

ans.append(ls[0][0])

last = ls[0][2]

for i in range(1,l):

if ls[i][1]>=last:

ans.append(ls[i][0])

last = ls[i][1]

#ans.sort()

return ans

#Pass different values to the function and test your program

activity\_list=[1,2,3,4,5,6,7]

start\_time\_list=[1,4,2,3,6,8,6]

finish\_time\_list=[2,6,4,5,7,10,9]

print("Activities:",activity\_list)

print("Start time of the activities:",start\_time\_list)

print("Finishing time of the activities:", finish\_time\_list)

result=find\_maximum\_activities(activity\_list,start\_time\_list, finish\_time\_list)

print("The maximum set of activities that can be completed:",result)

#DSA-Tryout

import sys

sys.setrecursionlimit(10000) #This is to overcome default python recursion limit

def fibonacci(num):

memo[0] = memo[1] = 1

for i in range(2,num+1):

memo[i] = memo[i-1]+memo[i-2]

return memo[num]

memo={} #global dictionary to store the fibonacci number already computed

print("Fibonacci number:",fibonacci(3))

#DSA-Exer-27

def max\_sum\_is(num\_list):

#Remove pass and write your logic here

n = len(num\_list)

max = 0

ls = [0 for x in range(n)]

for i in range(n):

ls[i] = num\_list[i]

for i in range(1, n):

for j in range(i):

if (num\_list[i] > num\_list[j] and

ls[i] < ls[j] + num\_list[i]):

ls[i] = ls[j] + num\_list[i]

for i in range(n):

if max < ls[i]:

max = ls[i]

return max

#Pass different values to the function and test your program

num\_list=[1, 101, 2, 3, 100, 4, 5]

print("Sum of the maximum sum increasing subsequence is :" ,max\_sum\_is(num\_list))

#DSA-Assgn-24

def count\_decoding(digit\_list):

n=len(digit\_list)

count = [0]\*(n + 1)

count[0] = 1

count[1] = 1

for i in range(2, n + 1):

if (digit\_list[i-1] > 0):

count[i] = count[i-1]

if (digit\_list[i-2] == 1 or (digit\_list[i-2] == 2 and digit\_list[i-1] < 7)):

count[i] += count[i - 2]

return count[n]

#Pass different values to the function and test your program

digit\_list=[9,8,1,5]

print("Number of possible decodings for the given sequence is:",count\_decoding(digit\_list))

#DSA-Assgn-25

def find\_number\_of\_platforms(arrival\_time\_list,departure\_time\_list):

#Remove pass and test your program

arrival\_time\_list.sort()

departure\_time\_list.sort()

plat\_needed = 1

result = 1

n = len(arrival\_time\_list)

i = 1

j = 0

while (i < n and j < n):

if (arrival\_time\_list[i] < departure\_time\_list[j]):

plat\_needed += 1

i += 1

if (plat\_needed > result):

result = plat\_needed

else:

plat\_needed -= 1

j += 1

return result

#Pass different values to the function and test your program

arrival\_time\_list = [800,810]

departure\_time\_list = [2300,2000]

print("The arrival time of the trains:", arrival\_time\_list)

print("The departure time of the trains:",departure\_time\_list)

print("Minimum number of platforms required :",find\_number\_of\_platforms(arrival\_time\_list,departure\_time\_list))

#DSA-Assgn-26

def count\_strings(number):

#Remove pass and write your logic here

a = [0 ]\*number

b = [0 ]\*number

a[0] = b[0] = 1

for i in range(1, number):

a[i] = a[i - 1] + b[i - 1]

b[i] = a[i - 1]

return a[number - 1] + b[number - 1]

#Pass different values to the function and test your program

number=3

print("The number of strings that can be made are:",count\_strings(number))

#DSA-Assgn-28

class Queue:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_rear=-1

self.\_\_front=0

def is\_full(self):

if(self.\_\_rear==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_front>self.\_\_rear):

return True

return False

def enqueue(self,data):

if(self.is\_full()):

print("Queue is full!!!")

else:

self.\_\_rear+=1

self.\_\_elements[self.\_\_rear]=data

def dequeue(self):

if(self.is\_empty()):

print("Queue is empty!!!")

else:

data=self.\_\_elements[self.\_\_front]

self.\_\_front+=1

return data

def display(self):

for index in range(self.\_\_front, self.\_\_rear+1):

print(self.\_\_elements[index])

def get\_max\_size(self):

return self.\_\_max\_size

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_front

while(index<=self.\_\_rear):

msg.append((str)(self.\_\_elements[index]))

index+=1

msg=" ".join(msg)

msg="Queue data(Front to Rear): "+msg

return msg

class Stack:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_top=-1

def is\_full(self):

if(self.\_\_top==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_top==-1):

return True

return False

def push(self,data):

if(self.is\_full()):

print("The stack is full!!")

else:

self.\_\_top+=1

self.\_\_elements[self.\_\_top]=data

def pop(self):

if(self.is\_empty()):

print("The stack is empty!!")

else:

data= self.\_\_elements[self.\_\_top]

self.\_\_top-=1

return data

def display(self):

if(self.is\_empty()):

print("The stack is empty")

else:

index=self.\_\_top

while(index>=0):

print(self.\_\_elements[index])

index-=1

def get\_max\_size(self):

return self.\_\_max\_size

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_top

while(index>=0):

msg.append((str)(self.\_\_elements[index]))

index-=1

msg=" ".join(msg)

msg="Stack data(Top to Bottom): "+msg

return msg

def check\_triangle\_number(queue1):

st = Stack(queue1.get\_max\_size())

i=0

while !queue1.is\_empty():

if i&1 == 0:

last = queue1.pop()

else:

cur = queue1.pop()

if ((cur)\*(cur+1))//2 == last:

st.push(cur)

return st

#Pass different values of queue to the function and test your program

queue1=Queue(10)

queue1.enqueue(7)

queue1.enqueue(28)

queue1.enqueue(8)

queue1.enqueue(35)

queue1.enqueue(3)

queue1.enqueue(6)

queue1.enqueue(5)

queue1.enqueue(15)

queue1.enqueue(2)

queue1.enqueue(3)

print("The elements in the queue are:")

queue1.display()

check\_triangle\_number(queue1)

#DSA-Assgn-28

class Queue:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_rear=-1

self.\_\_front=0

def is\_full(self):

if(self.\_\_rear==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_front>self.\_\_rear):

return True

return False

def enqueue(self,data):

if(self.is\_full()):

print("Queue is full!!!")

else:

self.\_\_rear+=1

self.\_\_elements[self.\_\_rear]=data

def dequeue(self):

if(self.is\_empty()):

print("Queue is empty!!!")

else:

data=self.\_\_elements[self.\_\_front]

self.\_\_front+=1

return data

def display(self):

for index in range(self.\_\_front, self.\_\_rear+1):

print(self.\_\_elements[index])

def get\_max\_size(self):

return self.\_\_max\_size

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_front

while(index<=self.\_\_rear):

msg.append((str)(self.\_\_elements[index]))

index+=1

msg=" ".join(msg)

msg="Queue data(Front to Rear): "+msg

return msg

class Stack:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_top=-1

def is\_full(self):

if(self.\_\_top==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_top==-1):

return True

return False

def push(self,data):

if(self.is\_full()):

print("The stack is full!!")

else:

self.\_\_top+=1

self.\_\_elements[self.\_\_top]=data

def pop(self):

if(self.is\_empty()):

print("The stack is empty!!")

else:

data= self.\_\_elements[self.\_\_top]

self.\_\_top-=1

return data

def display(self):

if(self.is\_empty()):

print("The stack is empty")

else:

index=self.\_\_top

while(index>=0):

print(self.\_\_elements[index])

index-=1

def get\_max\_size(self):

return self.\_\_max\_size

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_top

while(index>=0):

msg.append((str)(self.\_\_elements[index]))

index-=1

msg=" ".join(msg)

msg="Stack data(Top to Bottom): "+msg

return msg

def check\_triangle\_number(queue1):

st = Stack(queue1.get\_max\_size())

i=0

while not queue1.is\_empty():

if i&1 == 0:

last = queue1.dequeue()

else:

cur = queue1.dequeue()

#print(cur)

if ((last)\*(last+1))//2 == cur:

st.push(cur)

#print(cur)

i+=1

return st

#Pass different values of queue to the function and test your program

queue1=Queue(11)

queue1.enqueue(10)

queue1.enqueue(12)

queue1.enqueue(78)

queue1.enqueue(8)

queue1.enqueue(37)

queue1.enqueue(6)

queue1.enqueue(21)

queue1.enqueue(3)

queue1.enqueue(3)

queue1.enqueue(6)

queue1.enqueue(15)

#[4, 10, 12, 78, 8, 37, 6, 21, 3, 3, 6, 15]

#print("The elements in the queue are:")

#queue1.display()

check\_triangle\_number(queue1)

#DSA-Assgn-29

class Queue:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_rear=-1

self.\_\_front=0

def is\_full(self):

if(self.\_\_rear==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_front>self.\_\_rear):

return True

return False

def enqueue(self,data):

if(self.is\_full()):

print("Queue is full!!!")

else:

self.\_\_rear+=1

self.\_\_elements[self.\_\_rear]=data

def dequeue(self):

if(self.is\_empty()):

print("Queue is empty!!!")

else:

data=self.\_\_elements[self.\_\_front]

self.\_\_front+=1

return data

def display(self):

for index in range(self.\_\_front, self.\_\_rear+1):

print(self.\_\_elements[index])

def get\_max\_size(self):

return self.\_\_max\_size

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_front

while(index<=self.\_\_rear):

msg.append((str)(self.\_\_elements[index]))

index+=1

msg=" ".join(msg)

msg="Queue data(Front to Rear): "+msg

return msg

class Stack:

def \_\_init\_\_(self,max\_size):

self.\_\_max\_size=max\_size

self.\_\_elements=[None]\*self.\_\_max\_size

self.\_\_top=-1

def is\_full(self):

if(self.\_\_top==self.\_\_max\_size-1):

return True

return False

def is\_empty(self):

if(self.\_\_top==-1):

return True

return False

def push(self,data):

if(self.is\_full()):

print("The stack is full!!")

else:

self.\_\_top+=1

self.\_\_elements[self.\_\_top]=data

def pop(self):

if(self.is\_empty()):

print("The stack is empty!!")

else:

data= self.\_\_elements[self.\_\_top]

self.\_\_top-=1

return data

def display(self):

if(self.is\_empty()):

print("The stack is empty")

else:

index=self.\_\_top

while(index>=0):

print(self.\_\_elements[index])

index-=1

def get\_max\_size(self):

return self.\_\_max\_size

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

msg=[]

index=self.\_\_top

while(index>=0):

msg.append((str)(self.\_\_elements[index]))

index-=1

msg=" ".join(msg)

msg="Stack data(Top to Bottom): "+msg

return msg

def stack\_operation(input\_stack):

#Remove pass and write your logic here

ls = []

updated\_stack = Stack(input\_stack.get\_max\_size())

while not input\_stack.is\_empty():

ls.append(input\_stack.pop())

l = len(ls)

ls1 = []

for i in range(l-3):

ls1.append(ls[l-4-i])

updated\_stack.push(ls[l-4-i])

for i in range(3):

ls1.append(ls[-i-1])

updated\_stack.push(ls[-i-1])

#print(ls1)

return updated\_stack

#Pass different values of stack to the function and test your program

input\_stack=Stack(5)

input\_stack.push('E')

input\_stack.push('D')

input\_stack.push('C')

input\_stack.push('B')

input\_stack.push('A')

print("The elements in input stack are:")

input\_stack.display()

print()

updated\_stack=stack\_operation(input\_stack)

print("The elements in the updated stack are:")

updated\_stack.display()

#DSA-Assgn-30

class Node:

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

self.\_\_data=data

self.\_\_next=None

def get\_data(self):

return self.\_\_data

def set\_data(self,data):

self.\_\_data=data

def get\_next(self):

return self.\_\_next

def set\_next(self,next\_node):

self.\_\_next=next\_node

class LinkedList:

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

self.\_\_head=None

self.\_\_tail=None

def get\_head(self):

return self.\_\_head

def get\_tail(self):

return self.\_\_tail

def add(self,data):

new\_node=Node(data)

if(self.\_\_head is None):

self.\_\_head=self.\_\_tail=new\_node

else:

self.\_\_tail.set\_next(new\_node)

self.\_\_tail=new\_node

def insert(self,data,data\_before):

new\_node=Node(data)

if(data\_before==None):

new\_node.set\_next(self.\_\_head)

self.\_\_head=new\_node

if(new\_node.get\_next()==None):

self.\_\_tail=new\_node

else:

node\_before=self.find\_node(data\_before)

if(node\_before is not None):

new\_node.set\_next(node\_before.get\_next())

node\_before.set\_next(new\_node)

if(new\_node.get\_next() is None):

self.\_\_tail=new\_node

else:

print(data\_before,"is not present in the Linked list")

def display(self):

temp=self.\_\_head

while(temp is not None):

print(temp.get\_data())

temp=temp.get\_next()

def find\_node(self,data):

temp=self.\_\_head

while(temp is not None):

if(temp.get\_data()==data):

return temp

temp=temp.get\_next()

return None

def delete(self,data):

node=self.find\_node(data)

if(node is not None):

if(node==self.\_\_head):

if(self.\_\_head==self.\_\_tail):

self.\_\_tail=None

self.\_\_head=node.get\_next()

else:

temp=self.\_\_head

while(temp is not None):

if(temp.get\_next()==node):

temp.set\_next(node.get\_next())

if(node==self.\_\_tail):

self.\_\_tail=temp

node.set\_next(None)

break

temp=temp.get\_next()

else:

print(data,"is not present in Linked list")

#You can use the below \_\_str\_\_() to print the elements of the DS object while debugging

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

temp=self.\_\_head

msg=[]

while(temp is not None):

msg.append(str(temp.get\_data()))

temp=temp.get\_next()

msg=" ".join(msg)

msg="Linkedlist data(Head to Tail): "+ msg

return msgPythonCopy

def separate\_chaining(list\_of\_numbers,size\_of\_hash\_table):

#Remove pass and write your logic here

hash\_table = {}

for i in range(size\_of\_hash\_table):

hash\_table[i] = LinkedList()

for i in list\_of\_numbers:

hash\_table[i%5].add(i)

return hash\_table

#Pass different values to the function and test your program

list\_of\_numbers=[81, 20, 34, 42, 21, 45,33,99]

size\_of\_hash\_table=5

result\_hash\_table=separate\_chaining(list\_of\_numbers, size\_of\_hash\_table)

count=0

for k in result\_hash\_table:

print("Values in hash table at index",count)

count+=1

k.display()

