**DSA ASSIGNMENT 4**



from datetime import datetime

startTime = "2:13:57"

endTime = "11:46:13"

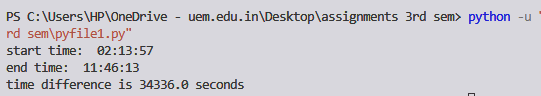
t1= datetime.strptime(startTime, "%H:%M:%S")

print("start time: ", t1.time())

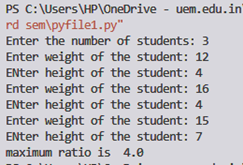
t2 = datetime.strptime(endTime, "%H:%M:%S")

print("end time: ", t2.time())

diff = t2-t1

print("time difference is", diff.total\_seconds(), "seconds")



n= int(input("Enter the number of students: "))

max= 0

for i in range(n):

w = float(input("Enter weight of the student: "))

h = float(input("ENter height of the student: "))

ratio = w/h

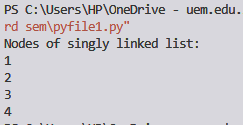
if(ratio>max):

max=ratio

print("maximum ratio is ", max)



class Node:

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

self.data = data

self.next = None

class SinglyLinkedList:

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

self.head = None

self.tail = None

def addNode(self, data):

newNode = Node(data)

if(self.head == None):

self.head = newNode

self.tail = newNode

else:

self.tail.next = newNode;

self.tail = newNode

def display(self):

current = self.head

if(self.head == None):

print("List is empty")

return

print("Nodes of singly linked list: ")

while(current != None):

print(current.data)

current = current.next

sList = SinglyLinkedList()

sList.addNode(1)

sList.addNode(2)

sList.addNode(3)

sList.addNode(4)

sList.display()



li = [1, 2]

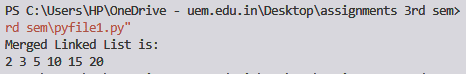
li.insert(2, 5)

li.pop(0)

print(li)



class Node:

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

self.data = data

self.next = None

class LinkedList:

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

self.head = None

def printList(self):

temp = self.head

while temp:

print(temp.data, end=" ")

temp = temp.next

def addToList(self, newData):

newNode = Node(newData)

if self.head is None:

self.head = newNode

return

last = self.head

while last.next:

last = last.next

last.next = newNode

def mergeLists(headA, headB):

dummyNode = Node(0)

tail = dummyNode

while True:

if headA is None:

tail.next = headB

break

if headB is None:

tail.next = headA

break

if headA.data <= headB.data:

tail.next = headA

headA = headA.next

else:

tail.next = headB

headB = headB.next

tail = tail.next

return dummyNode.next

listA = LinkedList()

listB = LinkedList()

listA.addToList(5)

listA.addToList(10)

listA.addToList(15)

listB.addToList(2)

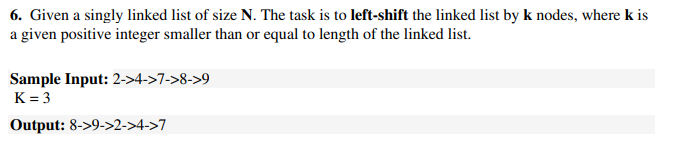
listB.addToList(3)

listB.addToList(20)

listA.head = mergeLists(listA.head, listB.head)

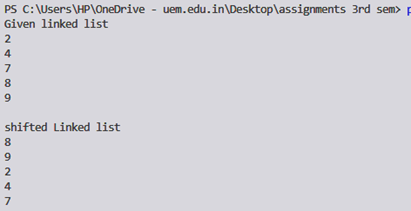
print("Merged Linked List is:")

listA.printList()



class Node:

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

 self.data = data

self.next = None

class LinkedList:

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

self.head = None

def push(self, new\_data):

new\_node = Node(new\_data)

new\_node.next = self.head

self.head = new\_node

def printList(self):

temp = self.head

while(temp):

print (temp.data),

temp = temp.next

def rotate(self, k):

if k == 0:

return

current = self.head

count = 1

while(count < k and current is not None):

current = current.next

count += 1

if current is None:

return

kthNode = current

while(current.next is not None):

current = current.next

current.next = self.head

self.head = kthNode.next

kthNode.next = None

llist = LinkedList()

llist.push(9)

llist.push(8)

llist.push(7)

llist.push(4)

llist.push(2)

print("Given linked list")

llist.printList()

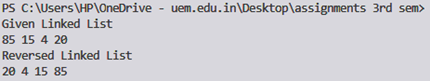
llist.rotate(3)

print("\nshifted Linked list")

llist.printList()



class Node:

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

self.data = data

self.next = None

class LinkedList:

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

self.head = None

def reverse(self):

prev = None

current = self.head

while(current is not None):

next = current.next

current.next = prev

prev = current

current = next

self.head = prev

def push(self, new\_data):

new\_node = Node(new\_data)

new\_node.next = self.head

self.head = new\_node

def printList(self):

temp = self.head

while(temp):

print (temp.data,end=" ")

temp = temp.next

llist = LinkedList()

llist.push(20)

llist.push(4)

llist.push(15)

llist.push(85)

print ("Given Linked List")

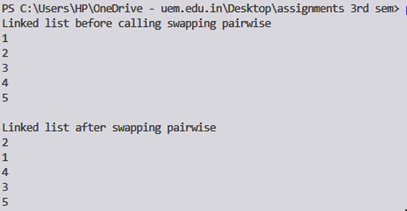
llist.printList()

llist.reverse()

print ("\nReversed Linked List")

llist.printList()



class Node:

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

self.data = data

self.next = None

class LinkedList:

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

self.head = None

def pairwiseSwap(self):

temp = self.head

if temp is None:

return

while(temp and temp.next):

if(temp.data != temp.next.data):

temp.data, temp.next.data = temp.next.data, temp.data

temp = temp.next.next

def push(self, new\_data):

new\_node = Node(new\_data)

new\_node.next = self.head

self.head = new\_node

def printList(self):

temp = self.head

while(temp):

print (temp.data),

temp = temp.next

llist = LinkedList()

llist.push(5)

llist.push(4)

llist.push(3)

llist.push(2)

llist.push(1)

print( "Linked list before calling swapping pairwise ")

llist.printList()

llist.pairwiseSwap()

print( "\nLinked list after swapping pairwise")

llist.printList()



class Node:

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

self.data = 0

self.next = None

self.flag = 0

def push(head\_ref, new\_data):

new\_node = Node();

new\_node.data = new\_data;

new\_node.flag = 0;

new\_node.next = (head\_ref);

(head\_ref) = new\_node;

return head\_ref

def detectLoop(h):

while (h != None):

if (h.flag == 1):

return True;

h.flag = 1;

h = h.next;

return False;

if \_\_name\_\_=='\_\_main\_\_':

head = None;

head = push(head, 20);

head = push(head, 4);

head = push(head, 15);

head = push( head, 10)

head.next.next.next.next = head;

if (detectLoop(head)):

print("Loop found")

else:

print("No Loop")



class Node:

def \_\_init\_\_(self, coeff, power):

self.coeff = coeff

self.power = power

self.next = None

def addPolynomials(head1, head2):

if(head1 == None and head2 == None):

return

elif(head1.power == head2.power):

print(f"{head1.coeff + head2.coeff}x^{head1.power}", end=" ")

addPolynomials(head1.next, head2.next)

elif(head1.power > head2.power):

print(f"{head1.coeff}x^{head1.power}", end=" ")

addPolynomials(head1.next, head2)

else:

print(f"{head2.coeff}x^{head2.power} ", end=" ")

addPolynomials(head1, head2.next)

def insert(head, coeff, power):

new\_node = Node(coeff, power)

while(head.next != None):

head = head.next

head.next = new\_node

def printList(head):

print("Linked List")

while(head != None):

print(f'{head.coeff}x^{head.power}', end=" ")

head = head.next

head = Node(5, 2)

insert(head, 4, 1)

head2 = Node(6, 2)

insert(head2, 4, 1)

printList(head)

print()

printList(head2)

print()

print("Addition:")

addPolynomials(head, head2)

