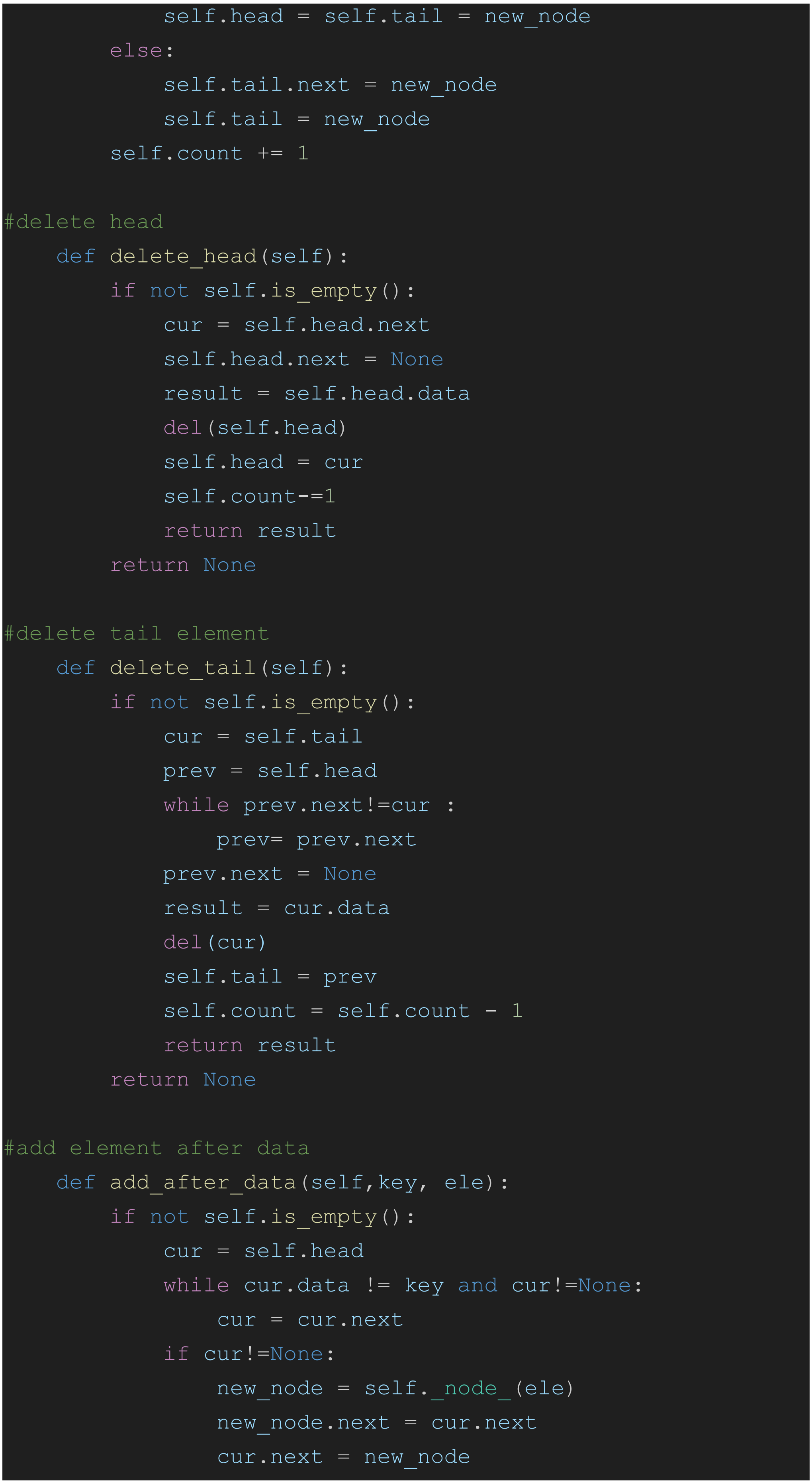
ADS LAB Assignment 2

1. Create Single Linked List class with following functionalities:

   Code:

```
class singleLinkedList:
    class _node_:
        def __init__(self, ele):
            self.data = ele
            self.next = None
    def __init__(self):
        self.head = None
        self.tail = None
        self.count = 0

    def is_empty(self):
        return self.count == 0

    def get_count(self):
        return self.count

    def displayLL(self):
        if self.is_empty():
            print("Linked list is empty")
        else:
            cur = self.head
            while cur!=None:
                print("{} ".format(cur.data), end=" ")
                cur = cur.next
#add head element
    def add_at_head(self, ele):
        new_node = self._node_(ele)
        if self.is_empty():
            self.head = self.tail = new_node
        else:
            new_node.next = self.head
            self.head = new_node
        self.count+=1

#add tail element
    def add_at_tail(self,ele):
        new_node = self._node_(ele)
        if self.is_empty():
```



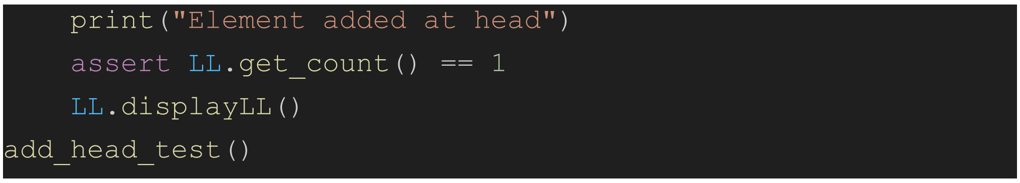
```
            self.head = self.tail = new_node
        else:
            self.tail.next = new_node
            self.tail = new_node
        self.count += 1

#delete head
    def delete_head(self):
        if not self.is_empty():
            cur = self.head.next
            self.head.next = None
            result = self.head.data
            del(self.head)
            self.head = cur
            self.count-=1
            return result
        return None

#delete tail element
    def delete_tail(self):
        if not self.is_empty():
            cur = self.tail
            prev = self.head
            while prev.next!=cur :
                prev= prev.next
            prev.next = None
            result = cur.data
            del(cur)
            self.tail = prev
            self.count = self.count - 1
            return result
        return None

#add element after data
    def add_after_data(self,key, ele):
        if not self.is_empty():
            cur = self.head
            while cur.data != key and cur!=None:
                cur = cur.next
            if cur!=None:
                new_node = self._node_(ele)
                new_node.next = cur.next
                cur.next = new_node
                if cur == self.tail:
                    self.tail = new_node
                self.count+=1
        else:
            print("Linked list is empty")

#delete element after data
    def delete_after_data(self,key):
        if not self.is_empty():
            cur = self.head
            while cur.data!=key and cur!=None:
                cur = cur.next
            if cur!=None and cur.next!=None:
                temp = cur.next
                if self.tail == temp:
                    self.tail = cur
                cur.next = temp.next
                result = temp.data
                temp.next = None
                del(temp)
                self.count-=1
                return result
            else:
                print("No element to delete")
        return None

#search element
    def search_Element(self,key):
        if not self.is_empty():
            temp = self.head
            while temp.data!=key and temp!=None:
                temp = temp.next
            if temp!=None:
                return True
        return False
```

Output:

a. Add at head

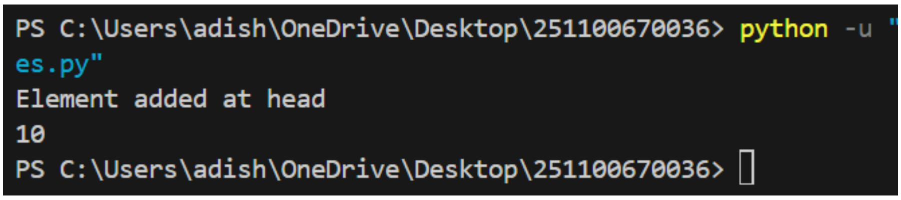
Test code

```
def add_head_test():
    LL = singleLinkedList()
    LL.add_at_head(10)
```



```
    print("Element added at head")
    assert LL.get_count() == 1
    LL.displayLL()
add_head_test()
```

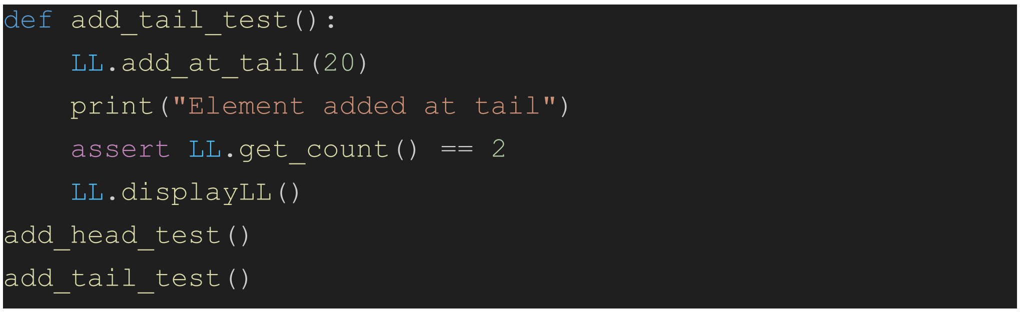
Result



```
PS C:\Users\adish\OneDrive\Desktop\251100670036> python -u "
es.py"
Element added at head
10
PS C:\Users\adish\OneDrive\Desktop\251100670036> 
```

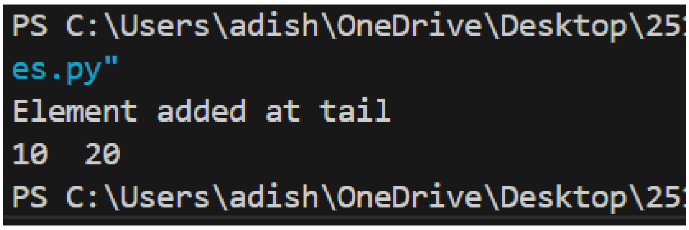
b. Add at tail

Test Code



```
def add_tail_test():
    LL.add_at_tail(20)
    print("Element added at tail")
    assert LL.get_count() == 2
    LL.displayLL()
add_head_test()
add_tail_test()
```

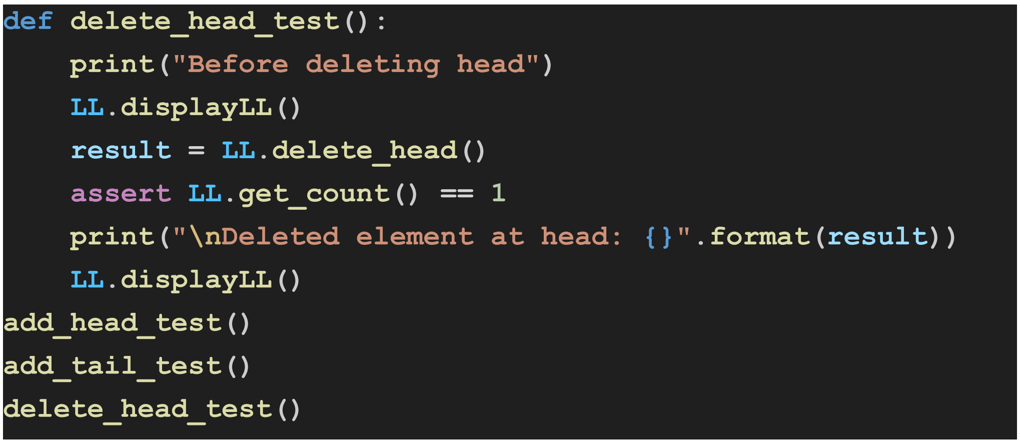
Result



```
PS C:\Users\adish\OneDrive\Desktop\25
es.py"
Element added at tail
10  20
PS C:\Users\adish\OneDrive\Desktop\25
```

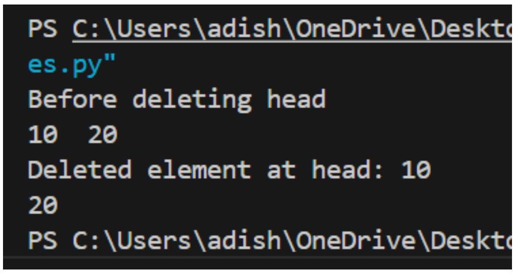
c. Delete at head

Test code



```
def delete_head_test():
    print("Before deleting head")
    LL.displayLL()
    result = LL.delete_head()
    assert LL.get_count() == 1
    print("\nDeleted element at head: {}".format(result))
    LL.displayLL()
add_head_test()
add_tail_test()
delete_head_test()
```

Result



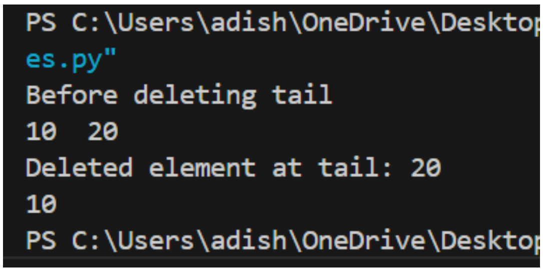
```
PS C:\Users\adish\OneDrive\Deskto
es.py"
Before deleting head
10  20
Deleted element at head: 10
20
PS C:\Users\adish\OneDrive\Deskto
```

d. Delete at tail

Test code

```
def delete_tail_test():
    print("Before deleting tail")
    LL.displayLL()
    result = LL.delete_tail()
    assert LL.get_count() == 1
    print("\nDeleted element at tail: {}".format(result))
    LL.displayLL()

add_head_test()
add_tail_test()
delete_tail_test()
```

Result



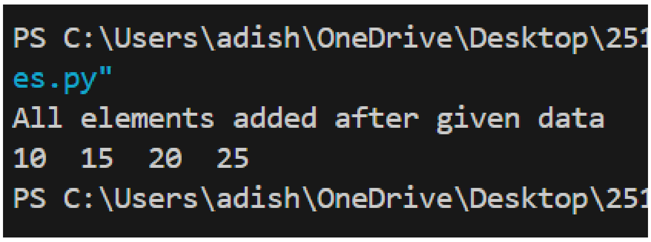
```
PS C:\Users\adish\OneDrive\Desktop
es.py"
Before deleting tail
10  20
Deleted element at tail: 20
10
PS C:\Users\adish\OneDrive\Desktop
```

e. Add after given data

Test Code

```
def after_given_data_test():
    LL.add_after_data(10,15)
    LL.add_after_data(15,20)
    LL.add_after_data(20,25)
    assert LL.get_count() == 4
    print("All elements added after given data")
    LL.displayLL()
add_head_test()
after_given_data_test()
```

Result



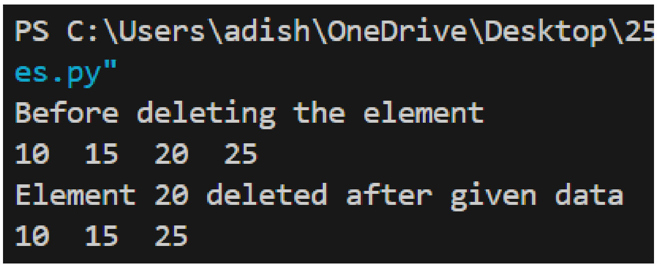
```
PS C:\Users\adish\OneDrive\Desktop\251
es.py"
All elements added after given data
10  15  20  25
PS C:\Users\adish\OneDrive\Desktop\251
```

f. Delete after given data

Test Code

```
def delete_after_data_test():
    print("Before deleting the element")
    LL.displayLL()
    result = LL.delete_after_data(15)
    assert LL.get_count() == 3
    print("\nElement {} deleted after given data".format(result))
    LL.displayLL()

add_head_test()
after_given_data_test()
delete_after_data_test()
```

Result



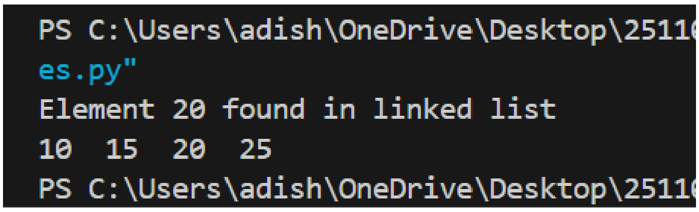
```
PS C:\Users\adish\OneDrive\Desktop\25
es.py"
Before deleting the element
10  15  20  25
Element 20 deleted after given data
10  15  25
```

g. Search an element

Test Code

```
def search_ele_test():
    element = 20
    assert LL.search_Element(element) == True
    print("Element {} found in linked list".format(element))
    LL.displayLL()
add_head_test()
after_given_data_test()
search_ele_test()
```

Result



```
PS C:\Users\adish\OneDrive\Desktop\2511
es.py"
Element 20 found in linked list
10  15  20  25
PS C:\Users\adish\OneDrive\Desktop\2511
```

2. Assume that we have two linked lists. Elements in the individual list are unique. There may be identical elements across linked lists. Create a third list which contains only common elements across first two lists.

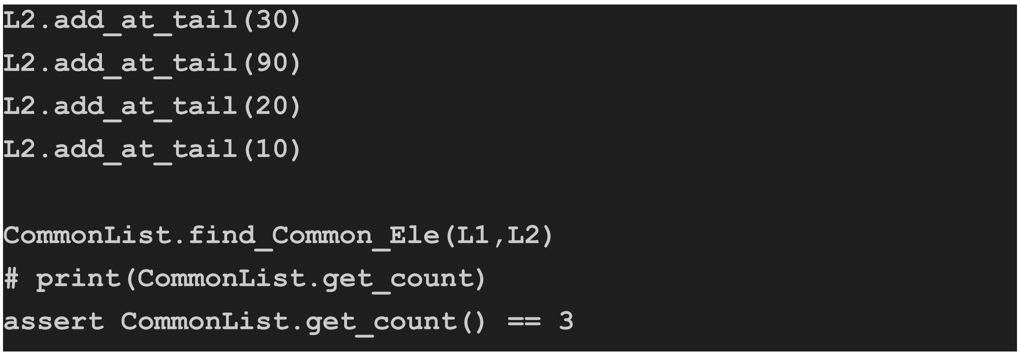
Code:

```
from Question_1 import *

class common_Elements(singleLinkedList):
    def __init__(self):
        super().__init__()

    def find_Common_Ele(self, list1, list2):
        ite = list1.head
        while ite!=None:
            if list2.search_Element(ite.data):
                self.add_at_tail(ite.data)
            ite = ite.next
        if not self.is_empty():
            print("Common Elements added:")
            self.displayLL()
```

Output:

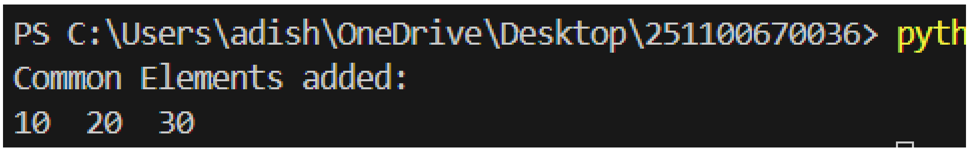
Test Code

```
from Question_1 import *
from Question_2 import *

L1 = singleLinkedList()
L2 = singleLinkedList()
CommonList = common_Elements()

L1.add_at_tail(10)
L1.add_at_tail(20)
L1.add_at_tail(30)
L1.add_at_tail(40)

L2.add_at_tail(50)
```



```
L2.add_at_tail(30)
L2.add_at_tail(90)
L2.add_at_tail(20)
L2.add_at_tail(10)

CommonList.find_Common_Ele(L1,L2)
# print(CommonList.get_count)
assert CommonList.get_count() == 3
```

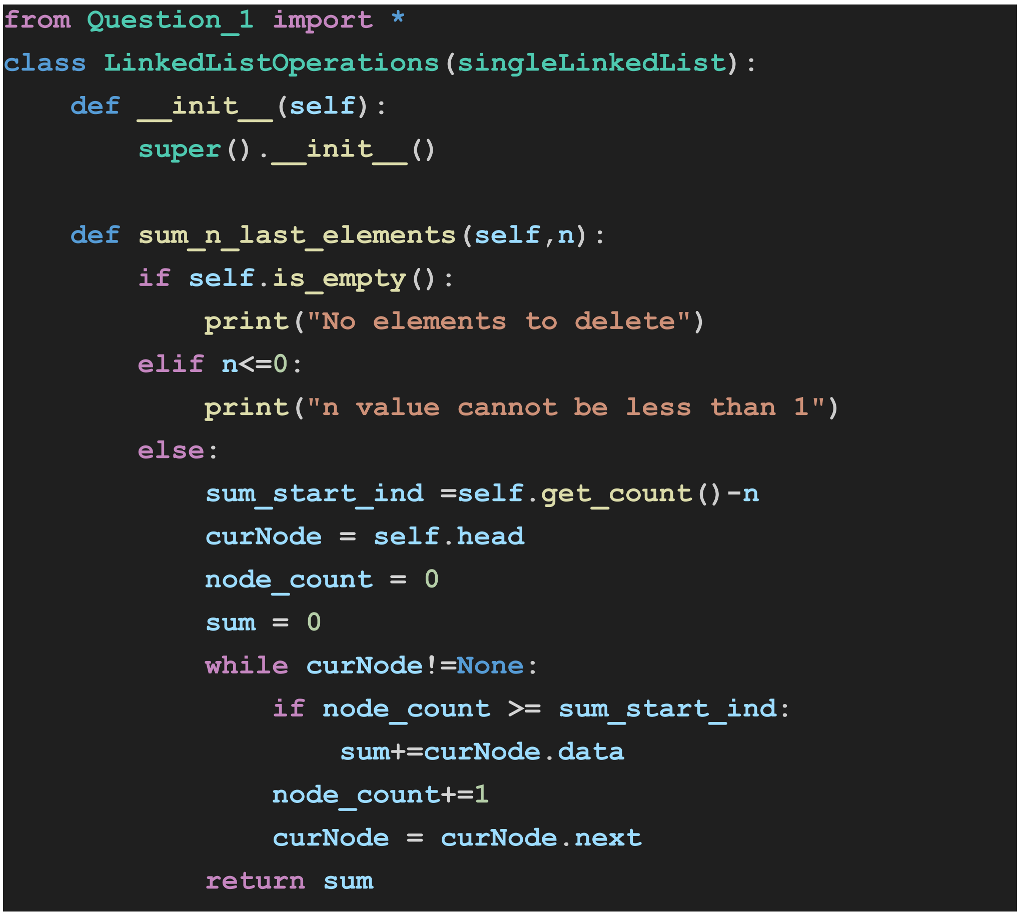
Result:



```
PS C:\Users\adish\OneDrive\Desktop\251100670036> pyth
Common Elements added:
10  20  30
```

3. Find the sum of last ‘n’ nodes in a single linked list. Where ‘n’ will be given. Sum should be calculated with one iteration.

Code:



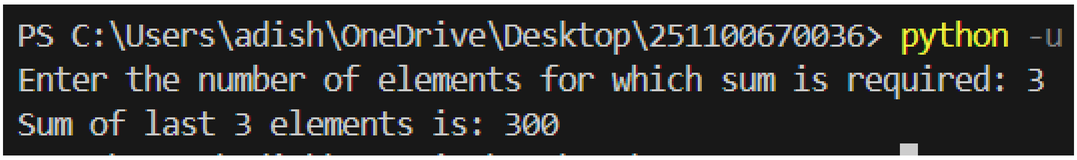
```
from Question_1 import *
class LinkedListOperations(singleLinkedList):
    def __init__(self):
        super().__init__()

    def sum_n_last_elements(self,n):
        if self.is_empty():
            print("No elements to delete")
        elif n<=0:
            print("n value cannot be less than 1")
        else:
            sum_start_ind =self.get_count()-n
            curNode = self.head
            node_count = 0
            sum = 0
            while curNode!=None:
                if node_count >= sum_start_ind:
                    sum+=curNode.data
                node_count+=1
                curNode = curNode.next
            return sum
```

Output:

Test Code

```
from Question_1 import *
from LL_Operations import *

LL = LinkedListOperations()

def add_elements():
      LL.add_at_tail(20)
      LL.add_at_tail(40)
      LL.add_at_tail(60)
      LL.add_at_tail(80)
      LL.add_at_tail(100)
      LL.add_at_tail(120)

add_elements()
n = int(input("Enter the number of elements for which sum is
required: "))
sum = LL.sum_n_last_elements(n)
assert sum == 300
print("Sum of last {} elements is: {}".format(n,sum))
```

Result:

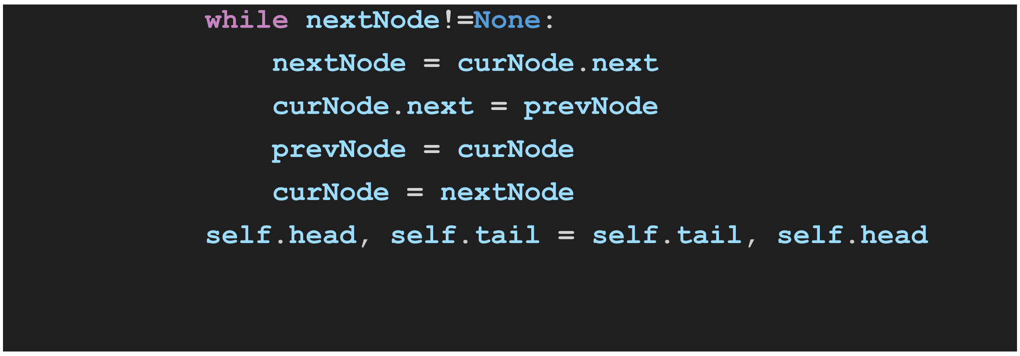


```
PS C:\Users\adish\OneDrive\Desktop\251100670036> python -u
Enter the number of elements for which sum is required: 3
Sum of last 3 elements is: 300
```

4. Reverse the single linked list.

Code:

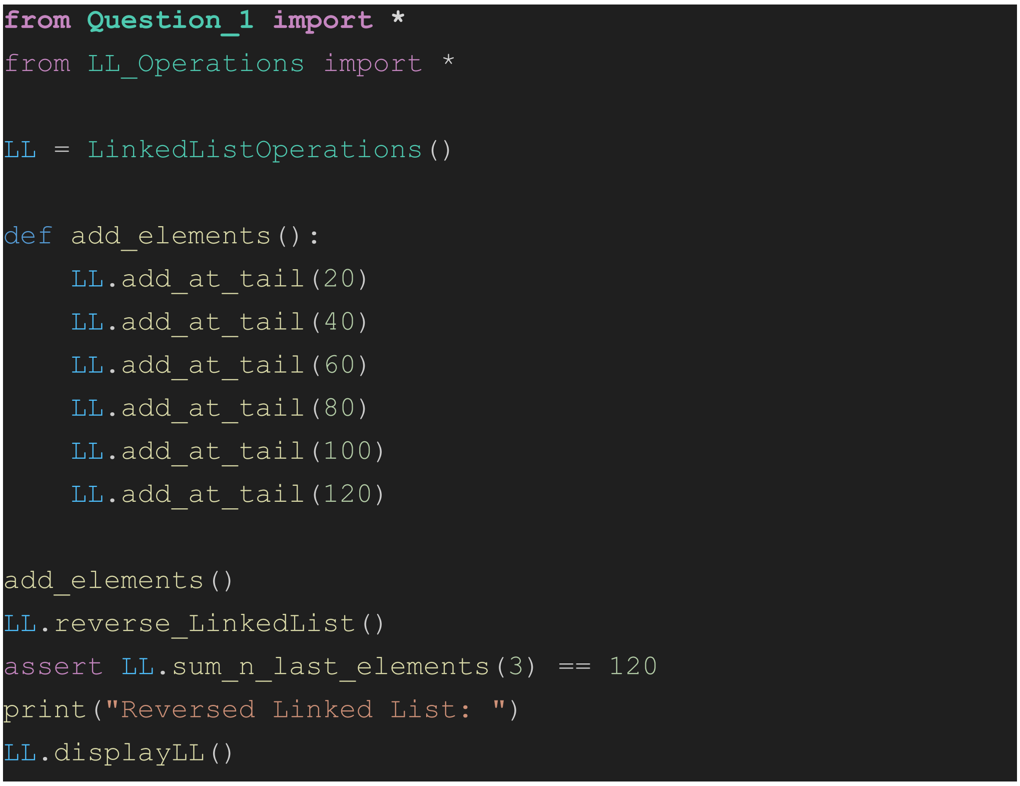
```
from Question_1 import *

class LinkedListOperations(singleLinkedList):
    def __init__(self):
        super().__init__()

    def reverse_LinkedList(self):
        if not self.is_empty():
            curNode = self.head
            nextNode = curNode
            prevNode = None
```



```
            while nextNode!=None:
                nextNode = curNode.next
                curNode.next = prevNode
                prevNode = curNode
                curNode = nextNode
            self.head, self.tail = self.tail, self.head
```

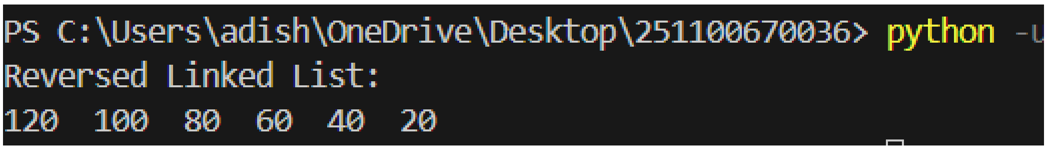
Output:

Testcode:



```
from Question_1 import *
from LL_Operations import *

LL = LinkedListOperations()

def add_elements():
    LL.add_at_tail(20)
    LL.add_at_tail(40)
    LL.add_at_tail(60)
    LL.add_at_tail(80)
    LL.add_at_tail(100)
    LL.add_at_tail(120)

add_elements()
LL.reverse_LinkedList()
assert LL.sum_n_last_elements(3) == 120
print("Reversed Linked List: ")
LL.displayLL()
```

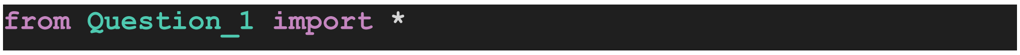
Output:



```
PS C:\Users\adish\OneDrive\Desktop\251100670036> python -u
Reversed Linked List:
120  100  80  60  40  20
```

5. Implement split() function which splits a given linked list into two separate linked lists containing alternate elements from the original list.

Code:



```
from Question_1 import *
```

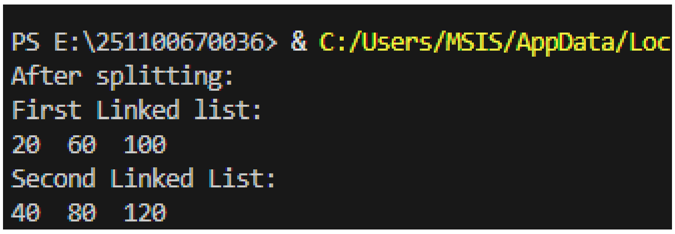
```
class LinkedListOperations(singleLinkedList):
    def __init__(self):
        super().__init__()

    def split_LinkedList(self):
        if not self.is_empty():
            # self.reverse_LinkedList()
            L1 = singleLinkedList()
            L2 = singleLinkedList()
            curNode = self.head
            NodeCount=0
            while curNode!=None:
                if NodeCount%2 == 0:
                    L1.add_at_tail(curNode.data)
                else:
                    L2.add_at_tail(curNode.data)
                curNode = curNode.next
                NodeCount+=1
            print("After splitting:")
            print("First Linked list: ")
            L1.displayLL()
            print("\nSecond Linked List: ")
            L2.displayLL()
```

Output:

Testcode:

```
from Question_1 import *
from LL_Operations import *

LL = LinkedListOperations()

def add_elements():
    LL.add_at_tail(20)
    LL.add_at_tail(40)
    LL.add_at_tail(60)
    LL.add_at_tail(80)
    LL.add_at_tail(100)
    LL.add_at_tail(120)

add_elements()
LL.split_LinkedList()
```

Result:



```
PS E:\251100670036> & C:/Users/MSIS/AppData/Loc
After splitting:
First Linked list:
20  60  100
Second Linked List:
40  80  120
```

6. Check whether the given linked list is palindrome or not.

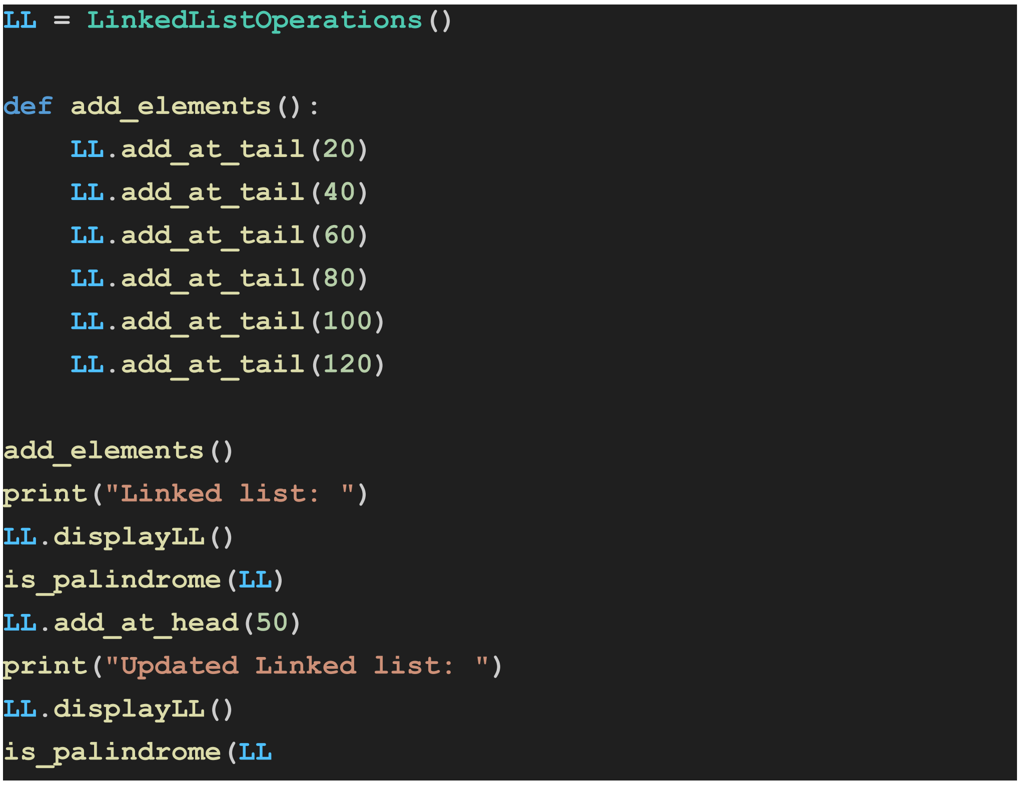
Code:

```
class LinkedListOperations(singleLinkedList):
    def __init__(self):
        super().__init__()

      def is_palindrome(self):
        if not self.is_empty():
            rev_list = singleLinkedList()
            curNode = self.head
            #creating a copy of the original linked list
            while curNode!=None:
                rev_list.add_at_tail(curNode.data)
                curNode = curNode.next
            rev_list.reverse_LinkedList()
            curNode=self.head
            curNode_2 = rev_list.head
            list_size = int(self.get_count())
            nodeCount = list_size/2 if list_size%2 == 0 else
(list_size/2)+1
            while nodeCount!=0:
                if curNode.data == curNode_2.data:
                    curNode = curNode.next
                    curNode_2 = curNode_2.next
                    nodeCount-=1
                else:
                    break
            return nodeCount == 0
```

Output:

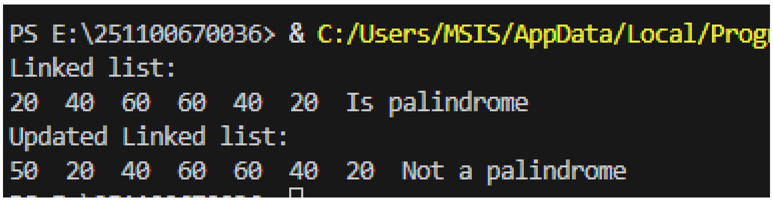
Testcode:

```
from Question_1 import *
from LL_Operations import *
```



```
LL = LinkedListOperations()

def add_elements():
    LL.add_at_tail(20)
    LL.add_at_tail(40)
    LL.add_at_tail(60)
    LL.add_at_tail(80)
    LL.add_at_tail(100)
    LL.add_at_tail(120)

add_elements()
print("Linked list: ")
LL.displayLL()
is_palindrome(LL)
LL.add_at_head(50)
print("Updated Linked list: ")
LL.displayLL()
is_palindrome(LL
```

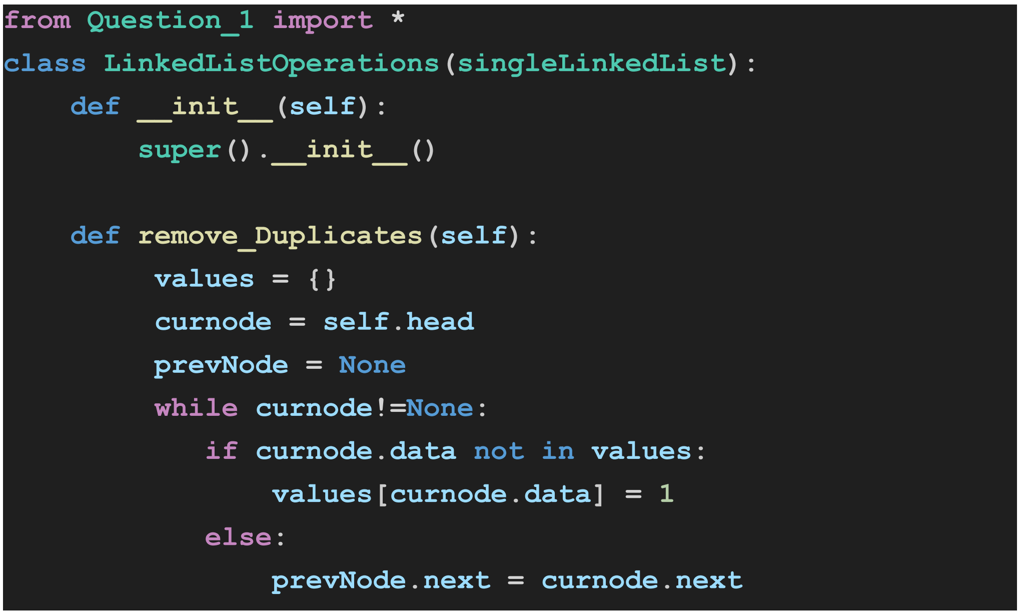
Result:



```
PS E:\251100670036> & C:/Users/MSIS/AppData/Local/Prog
Linked list:
20  40  60  60  40  20  Is palindrome
Updated Linked list:
50  20  40  60  60  40  20  Not a palindrome
```

7. Write an efficient code to remove duplicate elements from a single linked list.

Code:



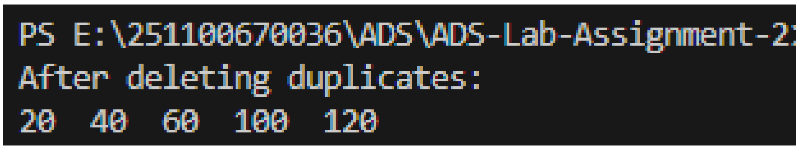
```
from Question_1 import *
class LinkedListOperations(singleLinkedList):
    def __init__(self):
        super().__init__()

    def remove_Duplicates(self):
         values = {}
         curnode = self.head
         prevNode = None
         while curnode!=None:
            if curnode.data not in values:
                values[curnode.data] = 1
            else:
                prevNode.next = curnode.next
                prevNode = curnode
                curnode = curnode.next
                prevNode.next = None
                del prevNode
                continue
            prevNode = curnode
            curnode = curnode.next
```

Output:

Testcode:

```
From Question_1 import *
from LL_Operations import *

LL = LinkedListOperations()

def add_elements():
    LL.add_at_tail(20)
    LL.add_at_tail(40)
    LL.add_at_tail(60)
    LL.add_at_tail(80)
    LL.add_at_tail(100)
    LL.add_at_tail(120)

LL.remove_Duplicates()
print(“After deleting duplicates:”)
LL.displayLL()
```

Result:



```
PS E:\251100670036\ADS\ADS-Lab-Assignment-2
After deleting duplicates:
20  40  60  100  120
```

8. Find the middle element of the linked list without iterating all elements.

Code:

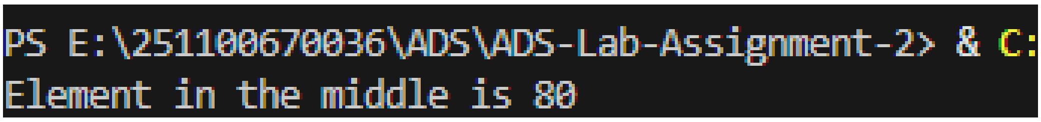
```
from Question_1 import *
class LinkedListOperations(singleLinkedList):
    def __init__(self):
        super().__init__()

    def middleElement(self):
        Elements_to_Search = self.get_count()
        curNode = self.head
        if Elements_to_Search %2 == 0:
            Elements_to_Search/=2
        else:
            Elements_to_Search =
int((Elements_to_Search/2)+1)
        while Elements_to_Search>1:
            curNode = curNode.next
            Elements_to_Search-=1
        print("Element in the middle is
{}".format(curNode.data))
```

Output:

Testcode:

```
From Question_1 import *
from LL_Operations import *

LL = LinkedListOperations()
def add_elements():
    LL.add_at_tail(20)
    LL.add_at_tail(40)
    LL.add_at_tail(60)
    LL.add_at_tail(80)
    LL.add_at_tail(90)
    LL.add_at_tail(100)
    LL.add_at_tail(120)

add_elements()
LL.middleElement()
```

Result:



```
PS E:\251100670036\ADS\ADS-Lab-Assignment-2> & C:
Element in the middle is 80
```

9. Write a method to remove odd elements from a circular single linked list.

Code:

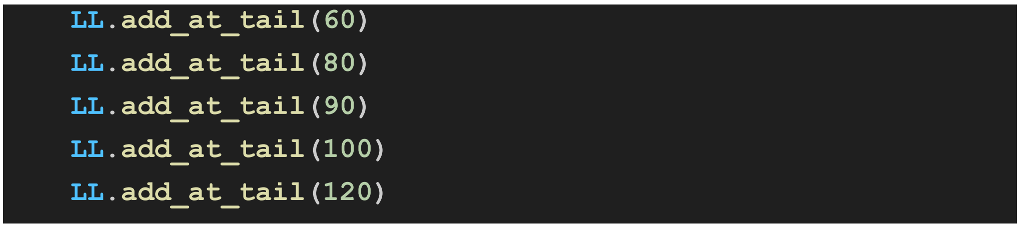
```
From Question_1 import *
class LinkedListOperations(singleLinkedList):
    def __init__(self):
        super().__init__()
      def makeCircularLL(self):
              self.tail.next = self.head

          def delete_odd_elements(self):
              if not self.head:
                  return
              NodeCount = self.get_count()
              if NodeCount == 1:
                  self.head = None
                  return
              curNode = self.head
              prevNode = None
              pos = 1
              new_head = self.head.next if self.head.next != self.head else None
              for _ in range(NodeCount):
                  nextNode = curNode.next
                  if pos % 2 != 0:
                      if prevNode:
                          prevNode.next = nextNode
                  else:
                      prevNode = curNode
                  curNode = nextNode
                  pos += 1

              self.head = new_head

              if prevNode:
                  prevNode.next = self.head
              else:
                  self.head = None
```

Output:

Testcode:

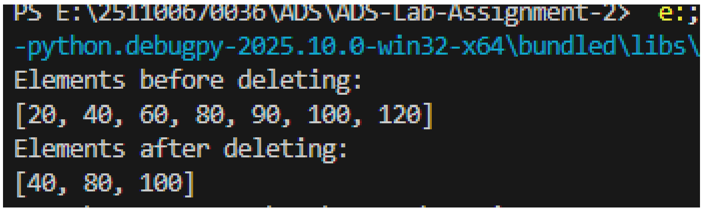
```
From Question_1 import *
```

```
from LL_Operations import *

LL = LinkedListOperations()
def add_elements():
    LL.add_at_tail(20)
    LL.add_at_tail(40)
```



```
    LL.add_at_tail(60)
    LL.add_at_tail(80)
    LL.add_at_tail(90)
    LL.add_at_tail(100)
    LL.add_at_tail(120)
```

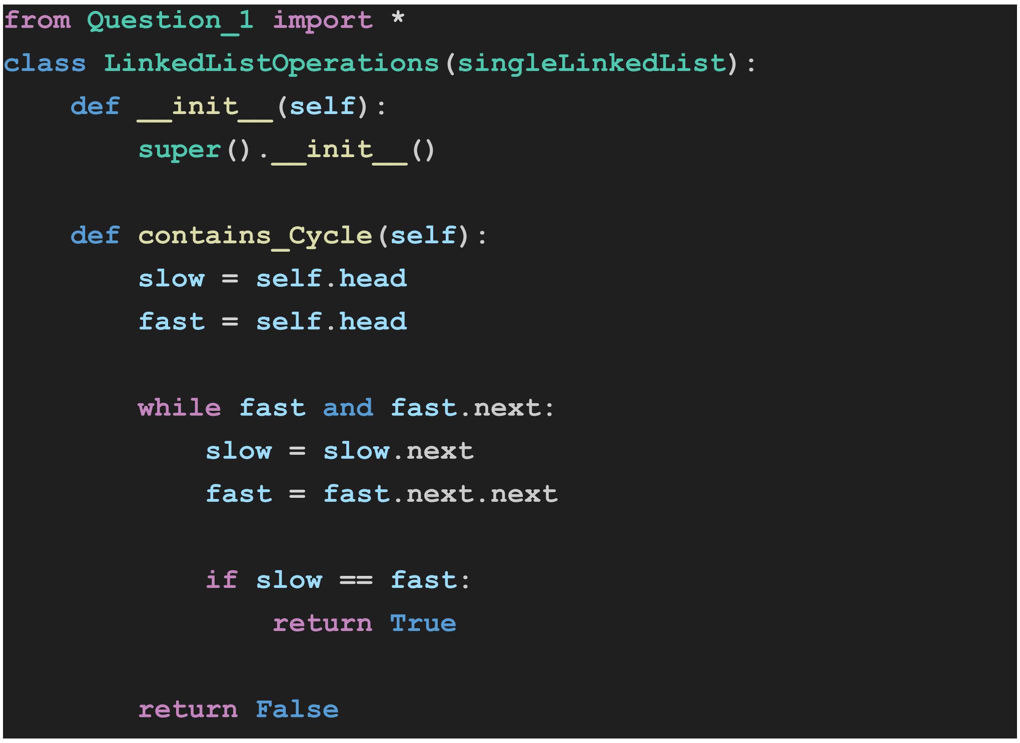
Result:



```
PS E:\25110067003б\ADS\ADS-Lab-Assignment-2>  e:;
-python.debugpy-2025.10.0-win32-x64\bundled\libs\
Elements before deleting:
[20, 40, 60, 80, 90, 100, 120]
Elements after deleting:
[40, 80, 100]
```

10. Find whether the linked list contains cycles.

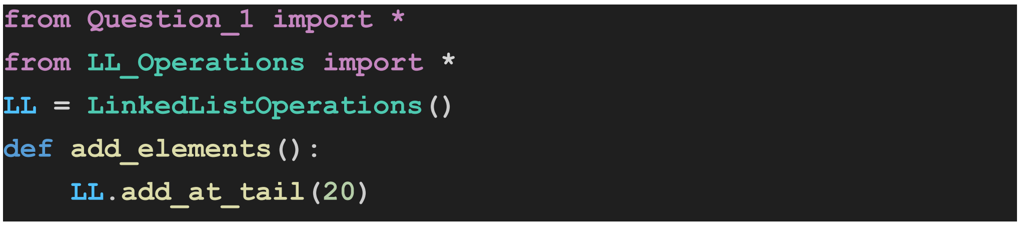
Code:



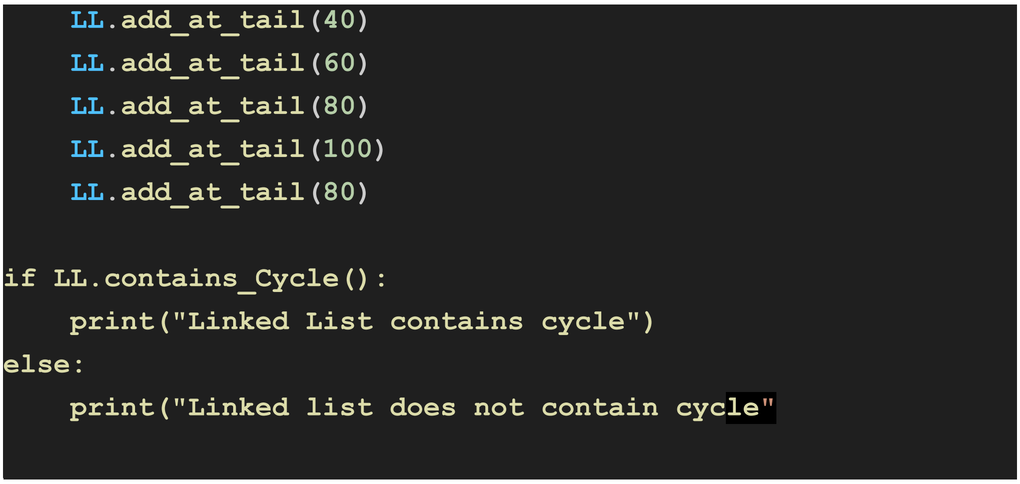
```
from Question_1 import *
class LinkedListOperations(singleLinkedList):
    def __init__(self):
        super().__init__()

    def contains_Cycle(self):
        slow = self.head
        fast = self.head

        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next

            if slow == fast:
                return True

        return False
```

Output:

Testcode:

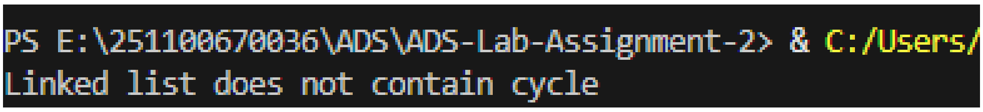


```
from Question_1 import *
from LL_Operations import *
LL = LinkedListOperations()
def add_elements():
    LL.add_at_tail(20)
```



```
    LL.add_at_tail(40)
    LL.add_at_tail(60)
    LL.add_at_tail(80)
    LL.add_at_tail(100)
    LL.add_at_tail(80)

if LL.contains_Cycle():
    print("Linked List contains cycle")
else:
    print("Linked list does not contain cycle"
```

Result:



```
PS E:\251100670036\ADS\ADS-Lab-Assignment-2> & C:/Users/
Linked list does not contain cycle
```