# Evidence 1

def main():

fruits = LinkedList()

while True:

print("1. Add an item\n" + \

"2. Traverse the linked list of used nodes and output the data values\n" + \

"3. Output all pointers and data values\n" + \

"X. Exit")

option = input("Enter option: ")

if option == '1':

fruits.AddNode()

elif option == '2':

fruits.Traversal()

elif option == '3':

fruits.DisplayLinkedList()

elif option == 'X':

break

else:

print("Invalid input.\n")

# Evidence 2

class ListNode:

def \_\_init\_\_(self, data = "", pointer= 0):

self.\_\_DataValue = data

self.\_\_PointerValue = pointer

def getData(self):

return self.\_\_DataValue

def getPointer(self):

return self.\_\_PointerValue

def setData(self, newdata):

self.\_\_DataValue = newdata

def setPointer(self, newpointer):

self.\_\_PointerValue = newpointer

class LinkedList:

def \_\_init\_\_(self, size = 30): # Initialise()

self.\_\_Node = [ListNode() for i in range(size + 1)]

self.\_\_Start = 0

self.\_\_NextFree = 1

for i in range(1, 30):

self.\_\_Node[i].setPointer(i + 1) # points to next node

def AddNode(self):

pass

def RemoveNode(self):

pass

def Traversal(self):

pass

def ReverseTraversal(self):

pass

def DisplayLinkedList(self):

print("| Node | Data | Pointer |\n" + \

"-------------------------------------------------------------------")

for i in range(1, len(self.\_\_Node)):

print("|{0:^8}|{1:^44}|{2:^11}|".format(i, self.\_\_Node[i].getData(), self.\_\_Node[i].getPointer()))

print()

print("Start: {0}".format(self.\_\_Start))

print("NextFree: {0}".format(self.\_\_NextFree))

print()

print()

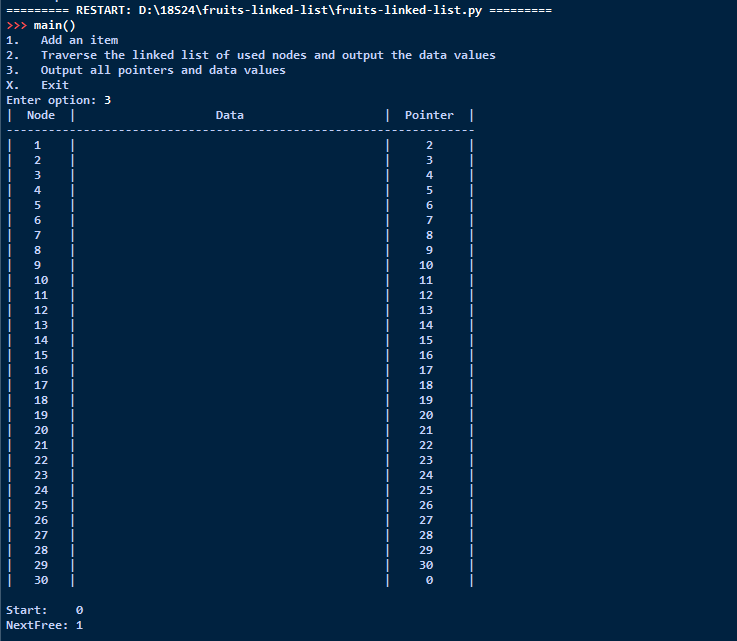
def IsEmpty(self):

return (self.\_\_Start == -1)

def IsFull(self):

pass

# Evidence 3



# Evidence 4

if self.IsFull():

print("Linked list is full, cannot insert!")

else:

newItem = input("Enter new item here: ")

self.\_\_Node[self.\_\_NextFree].setData(newItem)

if self.\_\_Start == 0:

self.\_\_Start = self.\_\_NextFree

temp = self.\_\_Node[self.\_\_NextFree].getPointer()

self.\_\_Node[self.\_\_NextFree].setPointer(0)

self.\_\_NextFree = temp

else:

temp = self.\_\_Node[self.\_\_NextFree].getPointer()

if newItem <= self.\_\_Node[self.\_\_Start].getData():

self.\_\_Node[self.\_\_NextFree].setPointer(self.\_\_Start)

self.\_\_Start = self.\_\_NextFree

self.\_\_NextFree = temp

else:

previous = 0

current = self.\_\_Start

found = False

while (found == False and current != 0):

if newItem <= self.\_\_Node[current].getData():

self.\_\_Node[previous].setPointer(self.\_\_NextFree)

self.\_\_Node[self.\_\_NextFree].setPointer(current)

self.\_\_NextFree = temp

found = True

else:

previous = current

current = self.\_\_Node[current].getPointer()

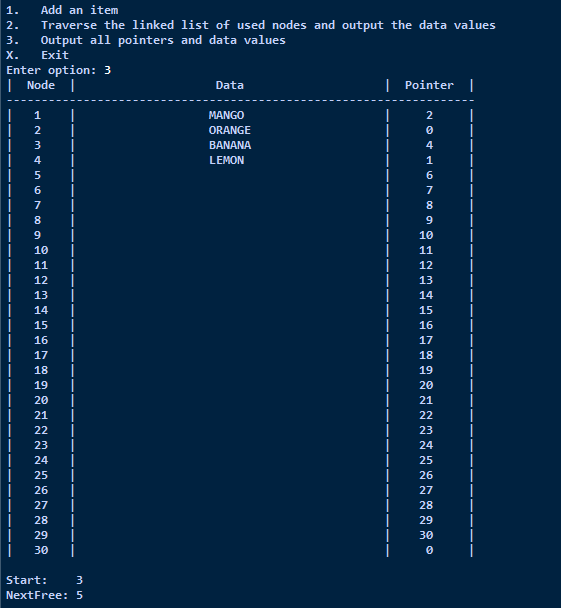
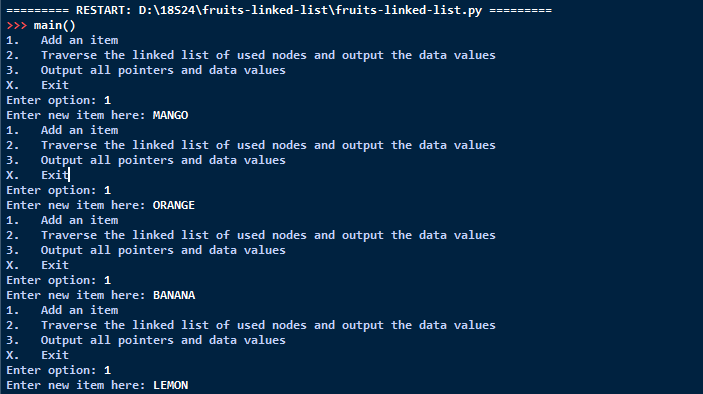
if current == 0:

self.\_\_Node[previous].setPointer(self.\_\_NextFree)

self.\_\_Node[self.\_\_NextFree].setPointer(0)

self.\_\_NextFree = temp

# Evidence 5



# Evidence 6

def Traversal(self):

index = self.\_\_Start

self.\_\_TraversalInOrder(index)

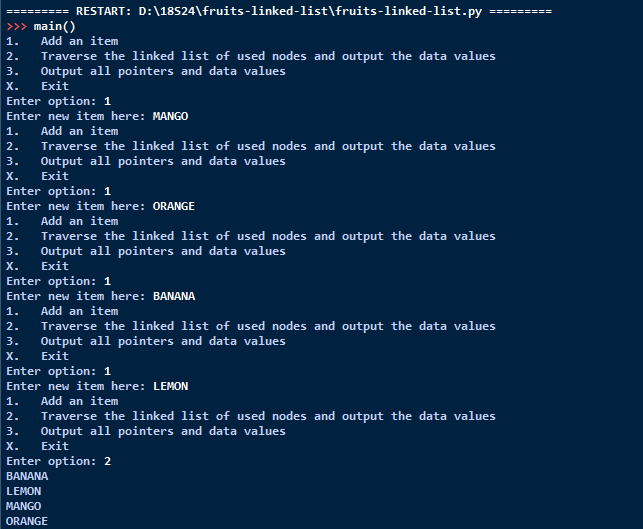
def \_\_TraversalInOrder(self, index):

if index != -1:

print(self.\_\_Node[index].getData())

self.\_\_TraversalInOrder(self.\_\_Node[index].getPointer())

# Evidence 7



# Evidence 8

def ReverseTraversal(self):

index = self.\_\_Start

self.\_\_TraversalInReverseOrder(index)

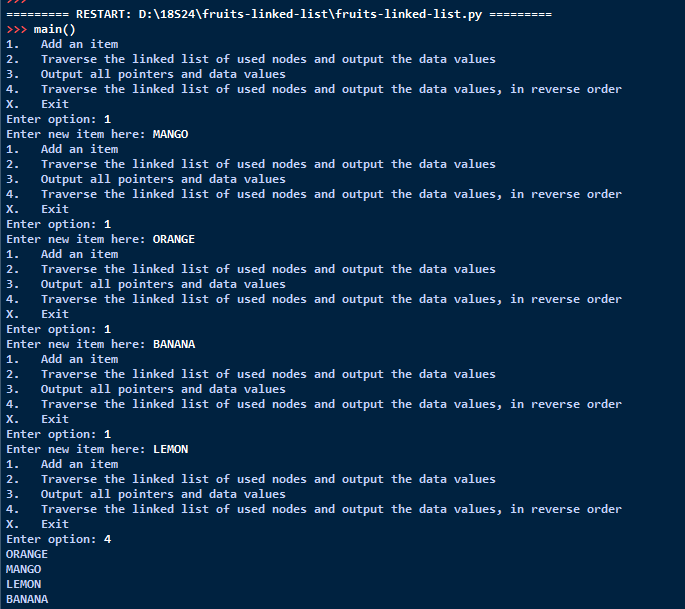
def \_\_TraversalInReverseOrder(self, index):

if index != -1:

self.\_\_TraversalInReverseOrder(self.\_\_Node[index].getPointer())

print(self.\_\_Node[index].getData())

# Evidence 9



# Evidence 10

|  |  |  |
| --- | --- | --- |
| **Identifier** | **Data type** | **Description** |
| ToRemove | STRING | The item to remove, input by the user. |
| Found | BOOLEAN | Flags to TRUE when the position at which to remove the item has been found. |
| Current | INTEGER | Current array index position during list traversal |
| Previous | INTEGER | Previous array index position during list traversal |
| Temp | INTEGER | Temporary storage of pointer value |

# Evidence 11

def RemoveNode(self):

ToRemove = input("Enter the item to remove: ")

if self.IsEmpty():

print("Cannot remove from the linked list as linked list is empty.")

else:

if self.\_\_Node[self.\_\_Start].getData() == ToRemove: # one to be removed is the first node

temp = self.\_\_NextFree

self.\_\_NextFree = self.\_\_Start

self.\_\_Start = self.\_\_Node[self.\_\_Start].getPointer()

self.\_\_Node[self.\_\_NextFree].setPointer(temp)

else:

previous = self.\_\_Start

current = self.\_\_Start

found = False

while (found == False and current != 0):

if self.\_\_Node[current].getData() == ToRemove:

temp = self.\_\_NextFree

self.\_\_Node[previous].setPointer(self.\_\_Node[current].getPointer())

self.\_\_NextFree = current

self.\_\_Node[self.\_\_NextFree].setPointer(temp)

found = True

else:

previous = current

current = self.\_\_Node[current].getPointer()

if current == 0:

print("Item not found in linked list, cannot delete.") Evidence 12

def IsEmpty(self):

return (self.\_\_Start == 0)

# Evidence 13

