create a singly linked list and display all the nodes present in the list

```
#Represent a node of the singly linked list
class Node:
    def __init__(self,data):
        self.data = data;
        self.next = None;

class SinglyLinkedList:
    #Represent the head and tail of the singly linked list
    def __init__(self):
        self.head = None;
        self.tail = None;

    #addNode() will add a new node to the list
    def addNode(self, data):
        #Create a new node
        newNode = Node(data);

        #Checks if the list is empty
        if(self.head == None):
            #If list is empty, both head and tail will point to new node
            self.head = newNode;
            self.tail = newNode;
        else:
            #newNode will be added after tail such that tail's next will point to newNode
            self.tail.next = newNode;
            #newNode will become new tail of the list
            self.tail = newNode;

    #display() will display all the nodes present in the list
    def display(self):
        #Node current will point to head
        current = self.head;

        if(self.head == None):
            print("List is empty");
            return;
        print("Nodes of singly linked list: ");
        while(current != None):
            #Prints each node by incrementing pointer
            print(current.data),
            current = current.next;

sList = SinglyLinkedList();

#Add nodes to the list
sList.addNode(1);
sList.addNode(2);
sList.addNode(3);
sList.addNode(4);

#Displays the nodes present in the list
sList.display();
```

```
class Node:
  def __init__(self, data):
    self.data = data
    self.next = None

class my_linked_list:
  def __init__(self):
    self.head = None
    self.last_node = None

  def add_value(self, my_data):
    if self.last_node is None:
      self.head = Node(my_data)
      self.last_node = self.head
    else:
      self.last_node.next = Node(my_data)
      self.last_node = self.last_node.next

  def print_it(self):
    curr = self.head
    while curr is not None:
      print(curr.data)
      curr = curr.next

  def find_index_val(self, my_key):
    curr = self.head

    index_val = 0
    while curr:
```

```
        if curr.data == my_key:
          return index_val
        curr = curr.next
        index_val = index_val + 1
      return -1

my_instance = my_linked_list()
my_list = [67, 4, 78, 98, 32, 0, 11, 8]
for data in my_list:
  my_instance.add_value(data)
print('The linked list is : ')
my_instance.print_it()
print()

my_key = int(input('What value would you search for? '))
index_val = my_instance.find_index_val(my_key)
if index_val == -1:
  print(str(my_key) + ' was not found.')
else:
  print('Element was found at index ' + str(index_val) + '.')
n = int(input('How many elements would you wish to add ? '))
for i in range(n):
  data = int(input('Enter data : '))
  my_instance.add_value(data)
print('The linked list is : ')
my_instance.print_it()
```

Explanation

- The ‘Node’ class is created.

- Another ‘my_linked_list’ class with required attributes is created.
- It has an ‘init’ function that is used to initialize the first element, i.e the ‘head’ to ‘None’ and last node to ‘None’.
- Another method named ‘add_value’ is defined, that is used to add data to the linked list.
- Another method named ‘print_it’ is defined that is used to display the linked list data on the console.
- Another method named ‘find_index_val’ is defined that helps find the index of the element entered by the user.
- An object of the ‘my_linked_list’ class is created.
- A list is defined.
- This list is iterated over, and the methods are called on it to add data.
- This is displayed on the console using the ‘print_it’ method.
- The user input is asked for the element to be searched.
- The ‘find_index_val’ method is called on this, and output is displayed on the console.