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

1. #Represent a node of the singly linked list
2. **class** Node:
3. **def** \_\_init\_\_(self,data):
4. self.data = data;
5. self.next = None;
7. **class** SinglyLinkedList:
8. #Represent the head and tail of the singly linked list
9. **def** \_\_init\_\_(self):
10. self.head = None;
11. self.tail = None;
13. #addNode() will add a new node to the list
14. **def** addNode(self, data):
15. #Create a new node
16. newNode = Node(data);
18. #Checks if the list is empty
19. **if**(self.head == None):
20. #If list is empty, both head and tail will point to new node
21. self.head = newNode;
22. self.tail = newNode;
23. **else**:
24. #newNode will be added after tail such that tail's next will point to newNode
25. self.tail.next = newNode;
26. #newNode will become new tail of the list
27. self.tail = newNode;
29. #display() will display all the nodes present in the list
30. **def** display(self):
31. #Node current will point to head
32. current = self.head;
34. **if**(self.head == None):
35. **print**("List is empty");
36. **return**;
37. **print**("Nodes of singly linked list: ");
38. **while**(current != None):
39. #Prints each node by incrementing pointer
40. **print**(current.data),
41. current = current.next;
43. sList = SinglyLinkedList();
45. #Add nodes to the list
46. sList.addNode(1);
47. sList.addNode(2);
48. sList.addNode(3);
49. sList.addNode(4);
51. #Displays the nodes present in the list
52. 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.