Python - Linked Lists

A linked list is a sequence of data elements, which are connected together via links. Each data element contains a connection to another data element in form of a pointer. Python does not have linked lists in its standard library. We implement the concept of linked lists using the concept of nodes as discussed in the previous chapter.

We have already seen how we create a node class and how to traverse the elements of a node.In this chapter we are going to study the types of linked lists known as singly linked lists. In this type of data structure there is only one link between any two data elements. We create such a list and create additional methods to insert, update and remove elements from the list.

Creation of Linked list

A linked list is created by using the node class we studied in the last chapter. We create a Node object and create another class to use this ode object. We pass the appropriate values through the node object to point the to the next data elements. The below program creates the linked list with three data elements. In the next section we will see how to traverse the linked list.

```python
class Node:
    def __init__(self, dataval=None):
        self.dataval = dataval
        self.nextval = None

class SLinkedList:
    def __init__(self):
        self.headval = None

list1 = SLinkedList()
list1.headval = Node("Mon")
e2 = Node("Tue")
e3 = Node("Wed")
# Link first Node to second node
list1.headval.nextval = e2

# Link second Node to third node
e2.nextval = e3
```

Traversing a Linked List

Singly linked lists can be traversed in only forward direction starting form the first data element. We simply print the value of the next data element by assigning the pointer of the next node to the current data element.

Example

```python
class Node:
    def __init__(self, dataval=None):
        self.dataval = dataval
        self.nextval = None

class SLinkedList:
    def __init__(self):
        self.headval = None

    def listprint(self):
        printval = self.headval
        while printval is not None:
            print (printval.dataval)
            printval = printval.nextval

list = SLinkedList()
list.headval = Node("Mon")
e2 = Node("Tue")
e3 = Node("Wed")

# Link first Node to second node
list.headval.nextval = e2

# Link second Node to third node
e2.nextval = e3

list.listprint()
```

Output

When the above code is executed, it produces the following result −

```
Mon
Tue
Wed
```

Insertion in a Linked List

Inserting element in the linked list involves reassigning the pointers from the existing nodes to the newly inserted node. Depending on whether the new data element is getting inserted at the beginning or at the middle or at the end of the linked list, we have the below scenarios.

Inserting at the Beginning

This involves pointing the next pointer of the new data node to the current head of the linked list. So the current head of the linked list becomes the second data element and the new node becomes the head of the linked list.

Example

```python
class Node:
    def __init__(self, dataval=None):
        self.dataval = dataval
        self.nextval = None

class SLinkedList:
    def __init__(self):
        self.headval = None
# Print the linked list
    def listprint(self):
        printval = self.headval
        while printval is not None:
            print (printval.dataval)
            printval = printval.nextval
    def AtBegining(self,newdata):
        NewNode = Node(newdata)

# Update the new nodes next val to existing node
        NewNode.nextval = self.headval
        self.headval = NewNode

list = SLinkedList()
list.headval = Node("Mon")
e2 = Node("Tue")
e3 = Node("Wed")

list.headval.nextval = e2
e2.nextval = e3
```

```
    list.AtBegining("Sun")
    list.listprint()
```

Output

When the above code is executed, it produces the following result −

```
Sun
Mon
Tue
Wed
```

Inserting at the End

This involves pointing the next pointer of the the current last node of the linked list to the new data node. So the current last node of the linked list becomes the second last data node and the new node becomes the last node of the linked list.

Example

```python
class Node:
    def __init__(self, dataval=None):
        self.dataval = dataval
        self.nextval = None
class SLinkedList:
    def __init__(self):
        self.headval = None
# Function to add newnode
    def AtEnd(self, newdata):
        NewNode = Node(newdata)
        if self.headval is None:
            self.headval = NewNode
            return
        laste = self.headval
        while(laste.nextval):
            laste = laste.nextval
        laste.nextval=NewNode
# Print the linked list
    def listprint(self):
        printval = self.headval
        while printval is not None:
            print (printval.dataval)
            printval = printval.nextval
```

```python
list = SLinkedList()
list.headval = Node("Mon")
e2 = Node("Tue")
e3 = Node("Wed")

list.headval.nextval = e2
e2.nextval = e3

list.AtEnd("Thu")

list.listprint()
```

Output

When the above code is executed, it produces the following result −

```
Mon
Tue
Wed
Thu
```

Inserting in between two Data Nodes

This involves changing the pointer of a specific node to point to the new node. That is possible by passing in both the new node and the existing node after which the new node will be inserted. So we define an additional class which will change the next pointer of the new node to the next pointer of middle node. Then assign the new node to next pointer of the middle node.

```python
class Node:
    def __init__(self, dataval=None):
        self.dataval = dataval
        self.nextval = None
class SLinkedList:
    def __init__(self):
        self.headval = None

# Function to add node
    def Inbetween(self,middle_node,newdata):
        if middle_node is None:
            print("The mentioned node is absent")
            return
```

```python
        NewNode = Node(newdata)
        NewNode.nextval = middle_node.nextval
        middle_node.nextval = NewNode

# Print the linked list
    def listprint(self):
        printval = self.headval
        while printval is not None:
            print (printval.dataval)
            printval = printval.nextval

list = SLinkedList()
list.headval = Node("Mon")
e2 = Node("Tue")
e3 = Node("Thu")

list.headval.nextval = e2
e2.nextval = e3

list.Inbetween(list.headval.nextval,"Fri")

list.listprint()
```

Output

When the above code is executed, it produces the following result −

```
Mon
Tue
Fri
Thu
```

Removing an Item

We can remove an existing node using the key for that node. In the below program we locate the previous node of the node which is to be deleted.Then, point the next pointer of this node to the next node of the node to be deleted.

Example

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

```python
        self.next = None
class SLinkedList:
    def __init__(self):
        self.head = None

    def Atbegining(self, data_in):
        NewNode = Node(data_in)
        NewNode.next = self.head
        self.head = NewNode

# Function to remove node
    def RemoveNode(self, Removekey):
        HeadVal = self.head

        if (HeadVal is not None):
            if (HeadVal.data == Removekey):
                self.head = HeadVal.next
                HeadVal = None
                return
        while (HeadVal is not None):
            if HeadVal.data == Removekey:
                break
            prev = HeadVal
            HeadVal = HeadVal.next

        if (HeadVal == None):
            return

        prev.next = HeadVal.next
            HeadVal = None

    def LListprint(self):
        printval = self.head
        while (printval):
            print(printval.data),
            printval = printval.next

llist = SLinkedList()
llist.Atbegining("Mon")
llist.Atbegining("Tue")
llist.Atbegining("Wed")
llist.Atbegining("Thu")
llist.RemoveNode("Tue")
llist.LListprint()
```

Output

When the above code is executed, it produces the following result −

```
Thu
Wed
Mon
```

Useful Video Courses

Python Online Training

187 Lectures   17.5 hours

Malhar Lathkar

More Detail

Python Essentials Online Training

55 Lectures   8 hours

Arnab Chakraborty

More Detail

Learn Python Programming In 100 Easy Steps

136 Lectures   11 hours

In28Minutes Official

More Detail

Python With Data Science

75 Lectures   13 hours

Eduonix Learning Solutions

More Detail

Python 3 From Scratch To Become A Developer In Demand

70 Lectures   8.5 hours

Lets Kode It

More Detail

Python Data Science Basics With Numpy, Pandas And Matplotlib

63 Lectures   6 hours

Abhilash Nelson

More Detail