كــليـة الـعـلـوم

قســــم الانـظـمـة الـطـبـيـة الـذكـيـة

Intelligent Medical Systems Department

Lecture: (4)

Linked List in Python

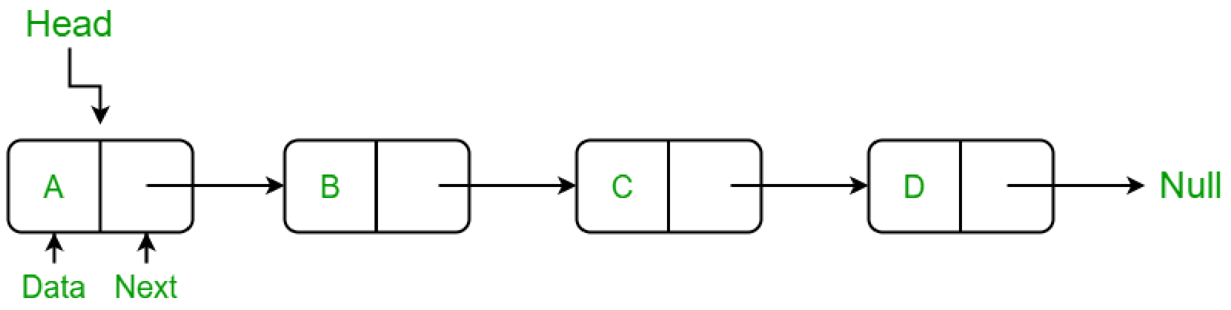
Subject: Data Structure Lab.
Class: Second
Lecturer: Asst. Prof. Mehdi Ebady Manaa
Asst. Lec. Sajjad Ibrahim Ismael

linked list

in this Lecture, we will learn about the implementation of a linked list in Python. To implement the linked list in Python, we will use classes in Python. we know that a linked list consists of nodes and nodes have two elements i.e. data and a reference to another node. Let's implement the node first.

What is Linked List in Python

A linked list is a type of linear data structure similar to arrays. It is a collection of nodes that are linked with each other. A node contains two things first is data and second is a link that connects it with another node. Below is an example of a linked list with four nodes and each node contains character data and a link to another node. Our first node is where **head** points and we can access all the elements of the linked list using the **head.**



Creating a linked list in Python

In this LinkedList class, we will use the Node class to create a linked list. In this class, we have an **__init__** method that initializes the linked list with an empty head. Next, we have created an **insertAtBegin()** method to insert a node at the beginning of the linked list, an **insertAtIndex()** method to insert a node at the given index of the linked list, and **insertAtEnd()** method inserts a node at the end of the linked list. After that, we have the **remove_node()** method which takes the data as an argument to delete that node. In the **remove_node()** method we traverse the linked list if a node is present equal to data then we delete that node from the linked list. Then we have the **sizeOfLL()** method to get the current size of the linked list and the last method of the LinkedList class is **printLL()** which traverses the linked list and prints the data of each node.

Creating a Node Class

We have created a Node class in which we have defined a **__init__** function to initialize the node with the data passed as an argument and a reference with None because if we have only one node then there is nothing in its reference.

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

Insertion in Linked List

Insertion at Beginning in Linked List

This method inserts the node at the beginning of the linked list. In this method, we create a **new_node** with the given **data** and check if the head is an empty node or not if the head is empty then we make the **new_node** as **head** and **return** else we insert the head at the next **new_node** and make the **head** equal to **new_node.**

```
def insertAtBegin(self, data):
    new_node = Node(data)
    if self.head is None:
        self.head = new_node
        return
    else:
        new_node.next = self.head
        self.head = new_node
```

Insert a Node at a Specific Position in a Linked List

This method inserts the node at the given index in the linked list. In this method, we create a **new_node** with given data , a current_node that equals to the head, and a counter **'position'** initializes with **0.** Now, if the index is equal to zero it means the node is to be inserted at begin so we called **insertAtBegin() method** else we run a while loop until the **current_node** is not equal to **None** or **(position+1)** is not equal to the index we have to at the one position back to insert at a given position to make the linking of nodes and in each iteration, we increment the position by 1 and make the **current_node** next of it. When the loop breaks and if **current_node** is not equal to **None** we insert new_node at after to the **current_node.** If **current_node** is equal to **None** it means that the index is not present in the list and we print **"Index not present".**

```
# Indexing starts from 0.
def insertAtIndex(self, data, index):
        new_node = Node(data)
        current_node = self.head
        position = 0
        if position == index:
            self.insertAtBegin(data)
        else:
            while(current_node != None and position+1 != index):
                position = position+1
                current_node = current_node.next

            if current_node != None:

                new_node.next = current_node.next
                current_node.next = new_node
            else:
                print("Index not present")
```

Insertion in Linked List at End

This method inserts the node at the end of the linked list. In this method, we create a **new_node** with the given data and check if the **head** is an empty node or not if the **head** is empty then we make the **new_node** as **head** and return **else** we make a **current_node equal** to **the**

head traverse to the last **node** of the linked list and when we get **None** after the current_node the while loop breaks and insert the **new_node** in the next of **current_node** which is the last node of linked list.

```
def inserAtEnd(self, data):
    new_node = Node(data)
    if self.head is None:
        self.head = new_node
        return

    current_node = self.head
    while(current_node.next):
        current_node = current_node.next

    current_node.next = new_node
```

Update the Node of a Linked List

This code defines a method called updateNode in a linked list class. It is used to update the value of a node at a given position in the linked list.

```
# Update node of a linked list at given position
def updateNode(self, val, index):
    current_node = self.head
    position = 0
    if position == index:
        current_node.data = val
    else:
        while(current_node != None and position != index):
            position = position+1
            current_node = current_node.next

        if current_node != None:
            current_node.data = val
        else:
            print("Index not present")
```

Delete Node in a Linked List

Remove First Node from Linked List

This method removes the first node of the linked list simply by making the second node **head** of the linked list.

```
def remove_first_node(self):
    if(self.head == None):
        return

    self.head = self.head.next
```

Remove Last Node from Linked List

In this method, we will delete the last node. First, we traverse to the second last node using the while loop, and then we make the next of that node **None** and last node will be removed.

```
def remove_last_node(self):

    if self.head is None:
        return

    current_node = self.head
    while(current_node.next.next):
        current_node = current_node.next

    current_node.next = None
```

Delete a Linked List Node at a given Position

In this method, we will remove the node at the given index, this method is similar to the **insert_at_inded()** method. In this method, if the **head** is **None** we simply **return** else we initialize a **current_node** with **self.head** and **position** with **0.** If the position is equal to the index we called the **remove_first_node()** method else we traverse to the one node before that we want to remove using the while loop. After that when we out of the while loop we check **that current_node** is equal to **None** if not then we make the next of current_node equal to the next of node that we want to remove else we print the message **“Index not present”** because **current_node** is equal to **None.**

```python
# Method to remove at given index
def remove_at_index(self, index):
        if self.head == None:
            return

        current_node = self.head
        position = 0
        if position == index:
            self.remove_first_node()
        else:
            while(current_node != None and position+1 != index):
                position = position+1
                current_node = current_node.next

            if current_node != None:
                current_node.next = current_node.next.next
            else:
                print("Index not present")
```

Delete a Linked List Node of a given Data

This method removes the node with the given data from the linked list. In this method, firstly we made a **current_node** equal to the **head** and run a **while loop** to traverse the linked list. This while loop breaks when **current_node** becomes **None** or the data next to the current node is equal to the data given in the argument. Now, After coming out of the loop if the **current_node** is equal to **None** it means that the node is not present in the data and we just return, and if the data next to the **current_node** is equal to the data given then we remove that node by making next of that removed_node to the next of current_node. And this is implemented using the if else condition.

```python
def remove_node(self, data):
    current_node = self.head

    while(current_node != None and current_node.next.data != data):
        current_node = current_node.next

    if current_node == None:
        return
    else:
        current_node.next = current_node.next.next
```

Linked List Traversal in Python

This method traverses the linked list and prints the data of each node. In this method, we made a **current_node** equal to the **head** and iterate through the linked list using a **while loop** until the **current_node** become None and print the data of **current_node** in each iteration and make the **current_node** next to it.

```python
def printLL(self):
    current_node = self.head
    while(current_node):
        print(current_node.data)
        current_node = current_node.next
```

Get Length of a Linked List in Python

This method returns the size of the linked list. In this method, we have initialized a counter **'size'** with 0, and then if the head is not equal to None we traverse the linked list using a **while loop** and increment the **size** with 1 in each iteration and return the size when **current_node** becomes **None else** we return 0.

```python
def sizeOfLL(self):
    size = 0
    if(self.head):
        current_node = self.head
        while(current_node):
            size = size+1
            current_node = current_node.next
        return size
    else:
        return 0
```

Example of the Linked list in Python

In this example, After defining the Node and LinkedList class we have created a linked list named **“llist”** using the linked list class and then insert four nodes with character data **‘a’**, **‘b’**, **‘c’**, **‘d’** and **‘g’** in the linked list then we print the linked list using **printLL()** method linked list class after that we have removed some nodes using remove methods and then print the linked list again and we can see in the output that node is deleted successfully. After that, we also print the size of the linked list.

Python3

```
# Create a Node class to create a node
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

# Create a LinkedList class
  class LinkedList:
    def __init__(self):
        self.head = None

    # Method to add a node at begin of LL
    def insertAtBegin(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        else:
            new_node.next = self.head
            self.head = new_node

    # Method to add a node at any index ,Indexing starts from 0.
    def insertAtIndex(self, data, index):
        new_node = Node(data)
        current_node = self.head
        position = 0
        if position == index:
            self.insertAtBegin(data)
        else:
            while(current_node != None and position+1 != index):
                position = position+1
                current_node = current_node.next

            if current_node != None:
                new_node.next = current_node.next
                current_node.next = new_node
            else:
                print("Index not present")

    # Method to add a node at the end of LL

    def insertAtEnd(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return

        current_node = self.head
        while(current_node.next):
            current_node = current_node.next

        current_node.next = new_node

    # Update node of a linked list at given position
    def updateNode(self, val, index):
        current_node = self.head
        position = 0
        if position == index:
            current_node.data = val
        else:
            while(current_node != None and position != index):
                position = position+1
                current_node = current_node.next

            if current_node != None:
                current_node.data = val
            else:
                print("Index not present")

    # Method to remove first node of linked list

    def remove_first_node(self):
        if(self.head == None):
            return

        self.head = self.head.next

    # Method to remove last node of linked list
    def remove_last_node(self):

        if self.head is None:
            return

        current_node = self.head
        while(current_node.next.next):
            current_node = current_node.next

        current_node.next = None

    # Method to remove at given index
    def remove_at_index(self, index):
        if self.head == None:
            return

        current_node = self.head
        position = 0
        if position == index:
            self.remove_first_node()
        else:
            while(current_node != None and position+1 != index):
                position = position+1
                current_node = current_node.next

            if current_node != None:
                current_node.next = current_node.next.next
            else:
                print("Index not present")

    # Method to remove a node from linked list
    def remove_node(self, data):
        current_node = self.head

        while(current_node != None and current_node.next.data != data):
            current_node = current_node.next

        if current_node == None:
            return
        else:
            current_node.next = current_node.next.next

    # Print the size of linked list
    def sizeOfLL(self):
        size = 0
        if(self.head):
            current_node = self.head
            while(current_node):
                size = size+1
                current_node = current_node.next
            return size
        else:
            return 0

    # print method for the linked list
    def printLL(self):
        current_node = self.head
        while(current_node):
            print(current_node.data)
            current_node = current_node.next


# create a new linked list
llist = LinkedList()

# add nodes to the linked list
llist.insertAtEnd('a')
llist.insertAtEnd('b')
llist.insertAtBegin('c')
llist.insertAtEnd('d')
llist.insertAtIndex('g', 2)

# print the linked list
print("Node Data")
llist.printLL()

# remove a nodes from the linked list
print("\nRemove First Node")
llist.remove_first_node()
print("Remove Last Node")
llist.remove_last_node()
print("Remove Node at Index 1")
llist.remove_at_index(1)


# print the linked list again
```

```python
print("\nLinked list after removing a node:")
llist.printLL()

print("\nUpdate node Value")
llist.updateNode('z', 0)
llist.printLL()

print("\nSize of linked list :", end=" ")
print(llist.sizeOfLL())
```

Output:

```
Node Data

c

a

g

b

d

Remove First Node

Remove Last Node

Remove Node at Index 1


Linked list after removing a node:

a

b

Update node Value

z

b

Size of linked list : 2
```