Week-6
The Singly Linked List- Creating Nodes, Traversing the Nodes, searching for a Node, Prepending Nodes, Removing Nodes. Linked List Iterators.

Creation of Singly Linked List:

We can create a singly linked list in python by following the mentioned steps.
Step 1: First, we create empty head or first reference and initialize it with None
Step 2: Create a class "node". The objects of this class hold one variable to store the values of the nodes and another variable to store the reference addresses.
Step 3: Create an empty node and assign it a value. Set the reference part of this node to None.
Step 4: Since we have only one element in the linked list so far, we will link first to this node by putting in its reference address

The python code to create a new node of the linked list is given below.
```
class Node:
   def__init_(self, data = None):
      self.data = data
      self.next = None
```

The python code to create a linked list and initialize first reference.
```
class LinkedList:
   def__init_(self):
      self.first = None
```

Prepending Nodes:

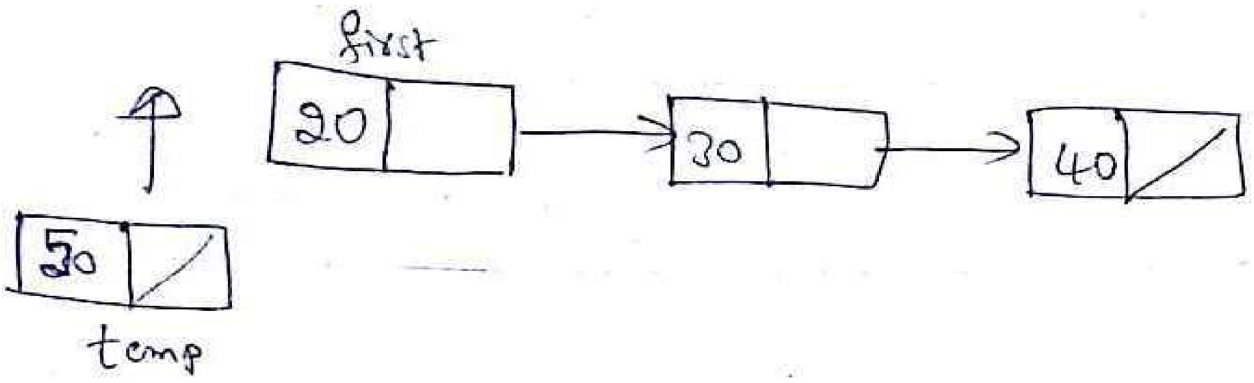
The following are the steps to be followed to insert a new node at beginning of the list.

Step 1:- Create a new node and insert the item into the new node.

Step 2:- If the list is empty then make new node as first. Go to step 4

Step 3:- Assign the “next” reference of the new node as first. Set the created node as the new first.

Step 4: Terminate.



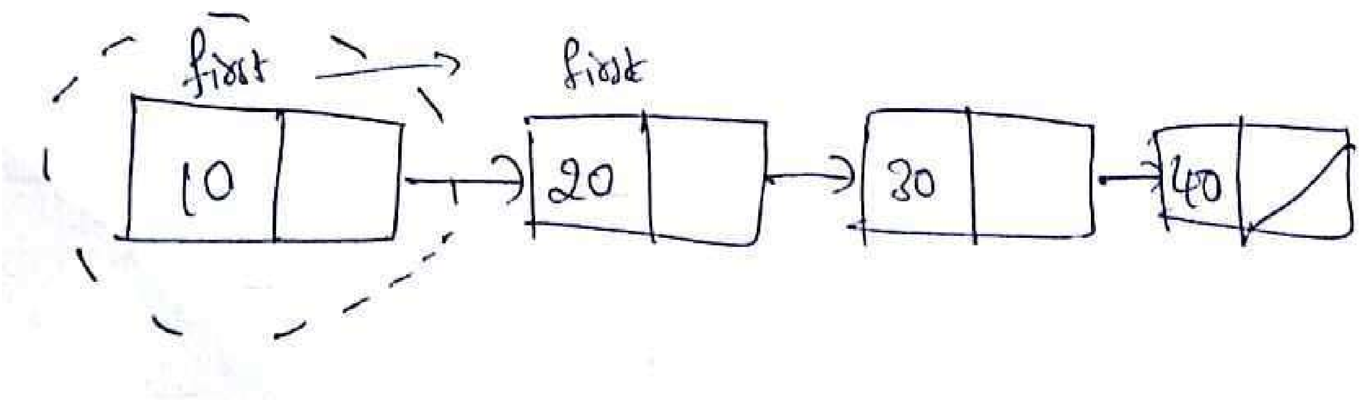
The python code to add new node to the linked list is given below.

```
def insertFirst(self, data):
        Newnode=Node(data)
        if(self.first == None):
            self.first=Newnode
       else:
           Newnode.next=self.first
           self.first=Newnode
```

Removing Nodes:

The following are the steps to be followed to remove node from the beginning of the list.

- Step 1: If the linked list is empty, return and go to step 5.
- Step 2: If there’s only one element, delete that node and set first to none. Go to step 5.
- Step 3: Set a “current” node pointing at first.
- Step 4: Assign first as the next node. Delete the current node.
- Step 5: Terminate



The python code to remove node from the linked list is given below.

```
def removeFirst(self):
        if(self.first== None):
           print("list is empty")
        else:
           cur=self.first;
           self.first=self.first.next
           print("the deleted item is",cur.data)
```

Traversing the Nodes:

A singly Linked list can only be traversed in forward direction from the first element to the last. We get the value of the next data element by simply iterating with the help of the reference address.

- Step 1: If the linked list is empty, display the message "List is empty" and move to step 5.
- Step 2: Iterate over the linked list using the reference address for each node.
- Step 3: Print every node's data.
- Step 4: Terminate.

The python code to traversing the nodes of linked list is given below.

```
def display(self):
        if(self.first== None):
           print("list is empty")
           return
        current = self.first
        while(current):
          print(current.data, end = " ")
          current = current.next
```

Searching for a Node:

- To find a node in a given singly linked list, we use the technique of traversal. In this case as soon as we find the node, we will terminate the loop.
- Algorithm to search a given node from the linked list is given below
  - ✓ Step 1: If the linked list is empty, display the message "List is empty" and move to step 5.
  - ✓ Step 2: Iterate over the linked list using the reference address for each node.
  - ✓ Step 3: Search every node for the given value.
  - ✓ Step 4: If the element is found, break the loop. If not, return the message "Element not found".
  - ✓ Step 5: Terminate.
- The python code implementation of searching the nodes of linked list is given below.

```
def search(self,item):
        if(self.first== None):
           print("list is empty")
           return
        current = self.first
        found = False
        while current != None and not found:
           if current.data == item:
```

```
        found = True
      else:
        current = current.next
    if(found):
      print("Item is present in the linked list")
    else:
      print("Item is not present in the linked list")
```

Implementation of singly linked list (Traversing the Nodes, searching for a Node, Prepending Nodes, Removing Nodes) :

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

class SinglyLinkedList:
  def__init_(self):
    self.first = None

  def insertFirst(self, data):
    newnode= Node(data)
    if(self.first == None):
       self.first=newnode
    else:
      newnode.next=self.first
      self.first=newnode

  def removeFirst(self):
    if(self.first== None):
      print("list is empty")
    else:
      cur=self.first
      self.first=self.first.next
      print("the deleted item is",cur.data)

  def display(self):
    if(self.first== None):
      print("list is empty")
      return
    current = self.first
    while(current):
    print(current.data, end = " ")
```

```
            current = current.next


    def search(self,item):
        if(self.first== None):
            print("list is empty")
            return
        current = self.first
        found = False
        while current != None and not found:
            if current.data == item:
                found = True
            else:
                current = current.next
        if(found):
            print("Item is present in the linked list")
        else:
            print("Item is not present in the linked list")

#Singly Linked List
linklist = SinglyLinkedList()
while(True):
    c1 = int(input("\nEnter your choice 1-insert 2-delete 3-search 4-display 5-exit :"))
    if(c1 == 1):
        item = input("Enter the element to insert:")
        linklist.insertFirst(item)
    elif(c1 == 2):
        linklist.removeFirst()
    elif(c1 == 3):
        item = input("Enter the element to search:")
        linklist.search(item)
    elif(c1 == 4):
        linklist.display()
    else:
        break
```

Linked List Iterators:

- Traversals are very common operations, especially on containers.
- A python for loop is used to traverse the items in strings, lists, tuples and dictionaries as follows.

```
print("List iteration")
l1 = [1,2,3,4]
for x in l1:
    print(x)
print("tuple iteration")
t1 = (10,20,30,40)
for x in t1:
    print(x)
print("String iteration")
t1 = "Welcome to gpt Athani"
for x in t1:
    print(x)
```

- An iterators guarantee that each element is visited exactly once.
- Custom created linked list is not iterable; there is a need to add a __iter___function to traverse through the list.
- Iterator function is defined as follows

```
def___iter_(self):
    current = self.first
    while current:
        yield current.data
        current = current.next
```

Implementation of linked list Iterator

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class SingleLinkedList:
    def __init__(self):
        self.first=None
    def insertFirst(self,data):
        Newnode=Node(data)
        if self.first==None:
            self.first=Newnode
        else:
            Newnode.next=self.first
            self.first=Newnode
    def removeFirst(self):
        if self.first==None:
            print("List is empty")
        else:
            cur=self.first
            self.first=self.first.next
            print("The deleted item is", cur.data)
```

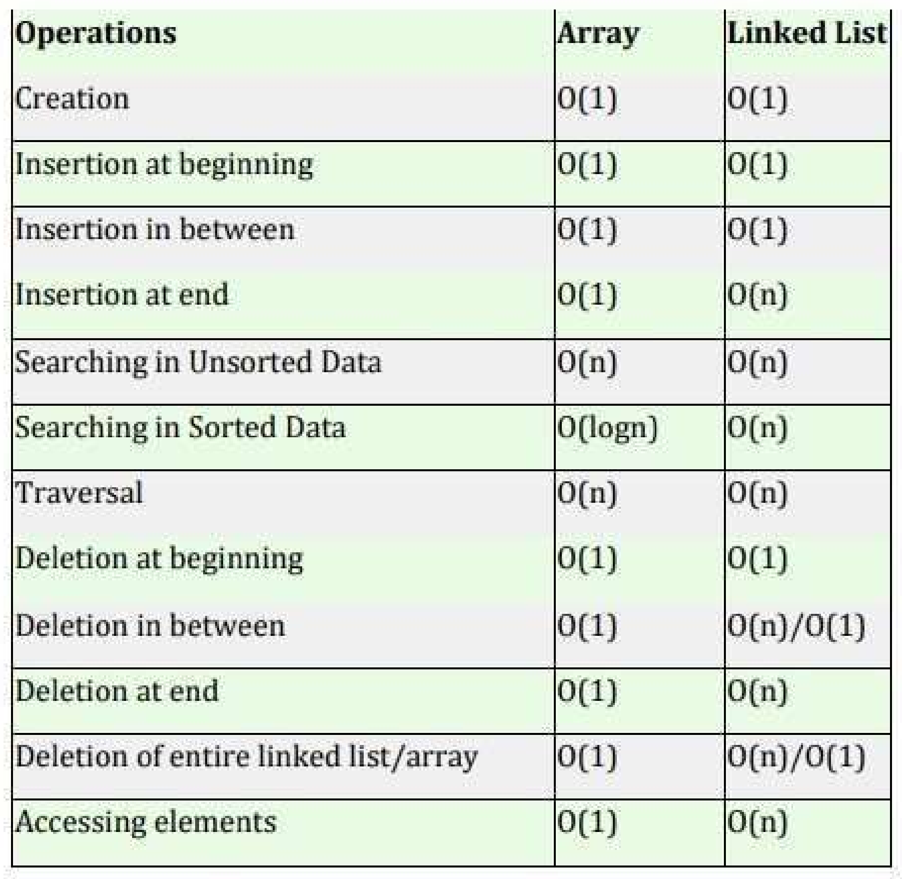
```
def removeend(self):
    if self.first==None:
        print("List is empty")
    else:
        cur=self.first
        prev=None
        while(cur.next):
            prev=cur
            cur=cur.next
        prev.next=None
        print("The deleted item is",cur.data)

def search(self,key):
    if self.first==None:
        print("List is empty")
        return
    cur=self.first
    found=False
    while cur!=None and not found:
        if cur.data==key:
            found=True
        else:
            cur=cur.next
    if(found):
        print("The item is present")
    else:
        print("The item not found")

 #Making linked list as iterable object
  def __iter__(self):
        current=self.first
        while current:
            yield current.data
            current=current.next
```

```
linklist=SingleLinkedList()
while(True):
   choice=int(input("Enter the choice : 1-insert 2-delete 3-display 4-search 5-removeend 
6-exit"))
   if(choice==1):
       item=input("Enter the data to be inserted")
       linklist.insertFirst(item)
   elif(choice==2):
      linklist.removeFirst()
   elif(choice==3):
      for x in linklist:
         print(x)
   elif(choice==4):
      key=input("Enter the key to be searched:")
      linklist.search(key)
   elif(choice==5):
         linklist.removeend()
   else:
      break
```

Time Complexity of Linked List Vs Array:



Operations	Array	Linked List
Creation	O(1)	O(1)
Insertion at beginning	O(1)	O(1)
Insertion in between	O(1)	O(1)
Insertion at end	O(1)	O(n)
Searching in Unsorted Data	O(n)	O(n)
Searching in Sorted Data	O(logn)	O(n)
Traversal	O(n)	O(n)
Deletion at beginning	O(1)	O(1)
Deletion in between	O(1)	O(n)/O(1)
Deletion at end	O(1)	O(n)
Deletion of entire linked list/array	O(1)	O(n)/O(1)
Accessing elements	O(1)	O(n)