CO1107
Data Structure

Linear Data Structure

- The data structure where data items are organized sequentially or linearly where data elements attached one after another is called linear data structure.
- Data elements in a liner data structure are traversed one after the other and only one element can be directly reached while traversing.
- All the data items in linear data structure can be traversed in single run.

Linear Data Structure

- There are two techniques of representing such linear structure within memory.
- The first way is to provide the linear relationships among all the elements represented using linear memory location. These linear structures are termed as arrays.
- The second technique is to provide the linear relationship among all the elements represented by using the concept of pointers or links. These linear structures are termed as linked lists.

Common Example of Linear Data Structure

- Arrays
- Queues
- Stacks
- Linked Lists

Non Linear Data Structure

- These are the data structures in which there is no sequential linking of data elements.
- Any pair or group of data elements can be linked to each other and can be accessed without a strict sequence.
- All the data elements in non linear data structure can not be traversed in single run.

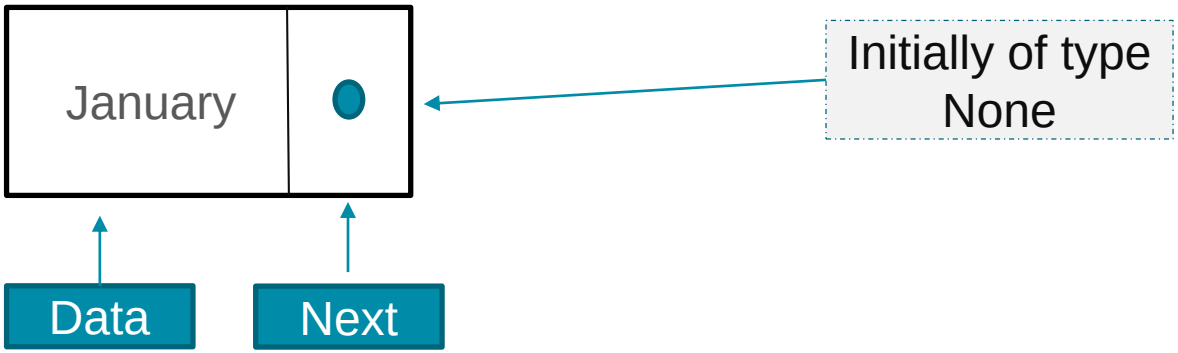
  - Binary Tree
  - Heap
  - Graph

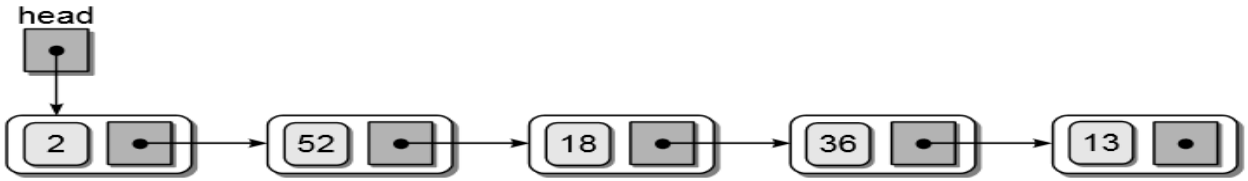
Python Specific Data Structure

- Python comes with a general set of built in data structures:
  - lists
  - tuples
  - string
  - dictionaries
  - sets
  - others...

Singly Linked List

- 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. [1]
- A singly linked list is made of series of nodes where each node consists of two fields:
  - Data Field: it has the data that we want to store
  - Next pointer: points to the address of the next node in the list





Implementation of Linked List in Python

- **Create Nodes:** a node can be defined as class which has two attributes: data and next

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


```
node1=Node("January")
```

Implementation of Linked List in Python

- **Add Nodes to linked list:** in order to add nodes to the linked list, we need to create a class called *LinkedList*


```
node1 = Node("Jan")
```

Implementation of Linked List in Python

- **Add Nodes to linked list:** in order to add nodes to the linked list, we need to create a class called *LinkedList*

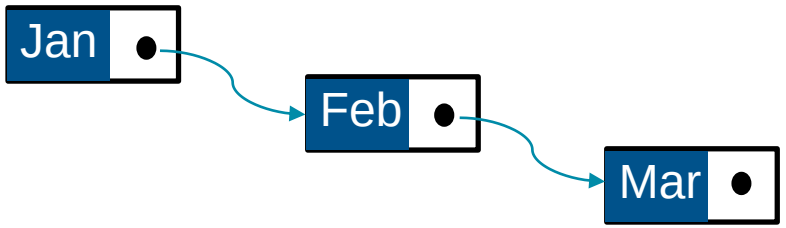
```
class Node:
    def __init__(self,data):
        self.data = data
        self.next = None
```

```
node1 = Node("Jan")
```

Implementation of Linked List in Python

- **Add Nodes to linked list:** in order to add nodes to the linked list, we need to create a class called *LinkedList*

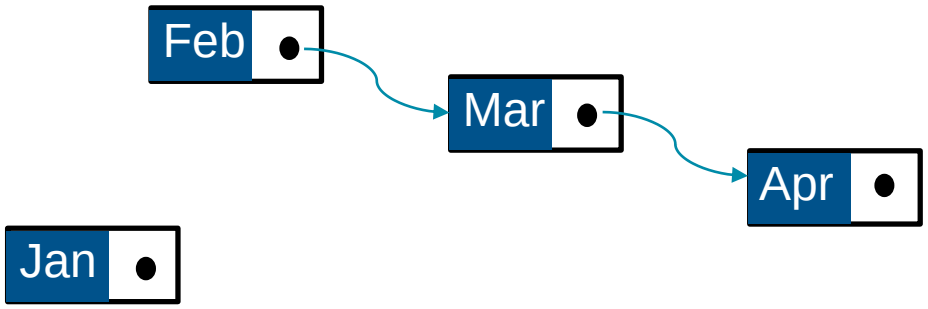
```
Class LinkedList:
    def __init__(self):
        self.head=None

    def insert(self, newNode):
        if self.head is None:
            self.head=newNode
        else:
            lastNode=self.head
            while True:
                if lastNode.next is None:
                    break
                lastNode=lastNode.next
            lastNode.next=newNode
```




```
node1 = Node("Jan")
lnklist = LinkedList()
lnklist.insert(node1)

node2 = Node("Feb")
lnklist.insert(node2)

node3 = Node("Mar")
lnklist.insert(node3)
```

Implementation of Linked List in Python

- **Add new node as a head node:**

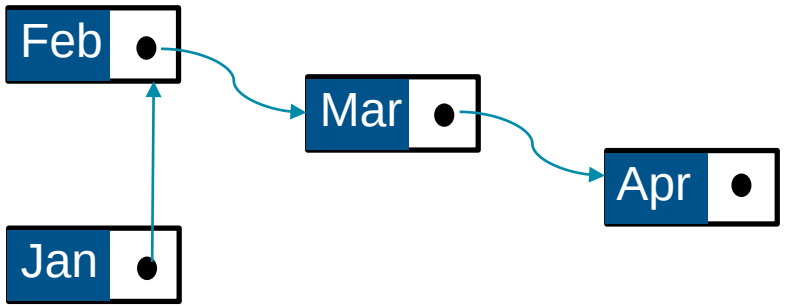


- Step 1: store the current head node into a temporary node
- Step 2: make the new node as a head node
- Step 3: make the next of your new node point to the temporary node

Implementation of Linked List in Python

- **Add new node as a head node:** in order to add nodes to the linked list, we need to create a class called *LinkedList*

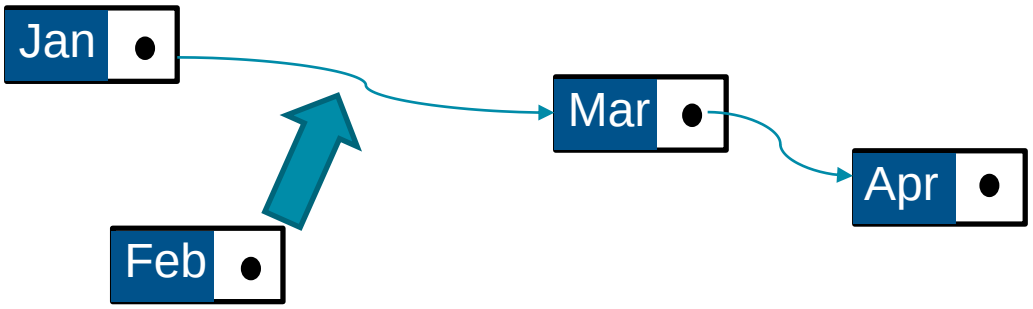
```
Class LinkedList:
    def __init__(self):
        self.head=None

    def insert(self, newNode):
        ….

    def insertHead(self, newNode):
        tempNode = self.head
        self.head = newNode
        self.head.next = tempNode
        del tempNode
```




```
node4 = Node("Jan")
lnklist.insertHead(node4)
```

Implementation of Linked List in Python

- **Add new node at specific position:**

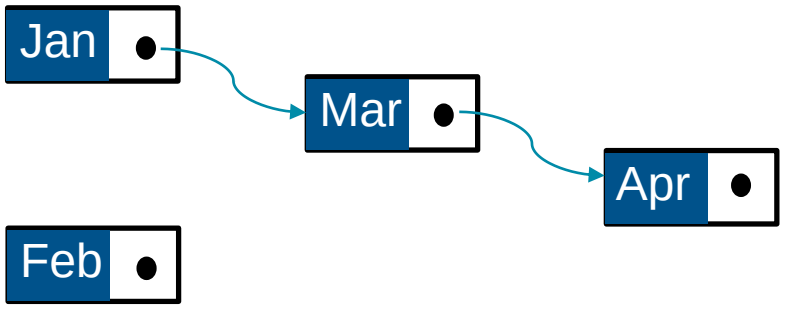


- Step 1: traverse the list till that specific position
- Step 2: store the details of the previous node
- Step 3: make a connection from the next of previous node to a new node

Implementation of Linked List in Python

- **Add new node at specific position:**

```
Class LinkedList:
    def __init__(self):
        self.head=None
    def insert(self, newNode):
        ….
    def insertHead(self, newNode):
        ….
    def insertAt(self, newNode, position):
        currentNode=self.head
        currentPosition=0
        while True:
            if currentPosition == position:
                previousNode.next=newNode
                newNode.next=currentNode
                break
            previousNode=currentNode
            currentNode=currentNode.next
            currentPosition += 1
```




```
node4 = Node("Feb")
lnklist.insertAt(node4, 1)
```

Class Activity

Traversing the node

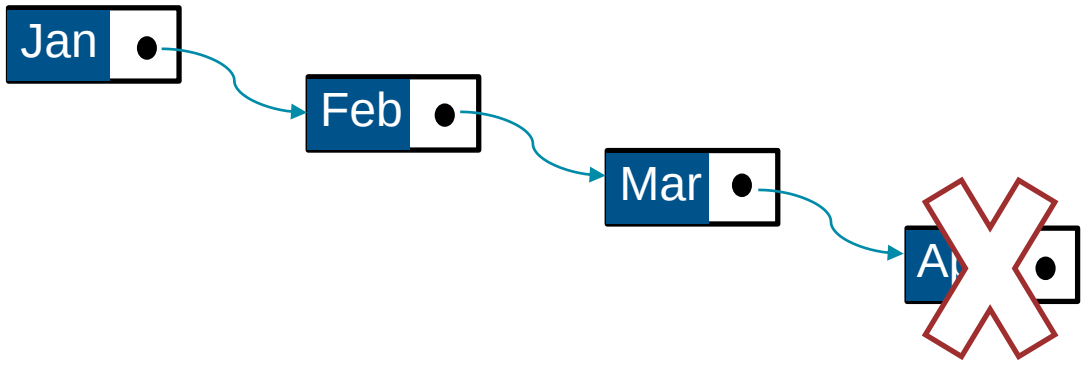
Implementation of Linked List in Python

- **List Traversal:**

```
Class LinkedList:
    def __init__(self):
        self.head=None
    def traversal(self):
        currentNode=self.head
        while currentNode is not None:
            print(currentNode.data)
            currentNode=currentNode.next
```

Implementation of Linked List in Python

- **Delete a node from the end of the list:**

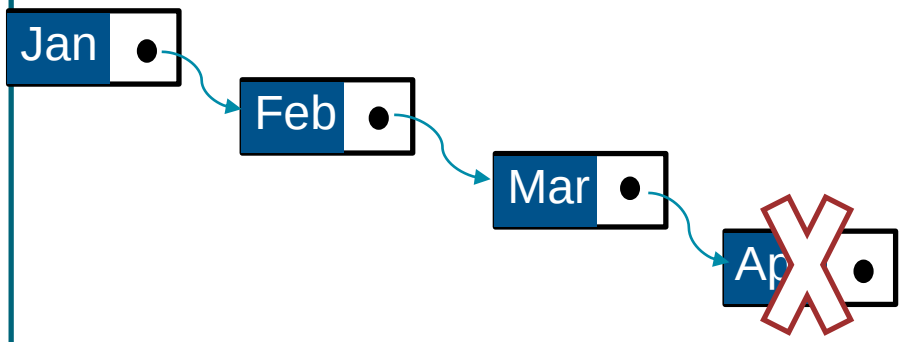


- Traverse till end of the list
- Store the last second node into a temporary node
- Delete the last node
- Make the next of temporary node points to **None**

Implementation of Linked List in Python

- **Delete a node from the end of the list:**

```
Class LinkedList:
    def __init__(self):
        self.head=None
    def insert(self, newNode):
        ….
    def insertHead(self, newNode):
        ….
    def insertAt(self, newNode, position):
         …
    def deleteEnd(self):
        lastNode=self.head
        while lastNode.next is not None:
            prevNode=lastNode
            lastNode=lastNode.next
        prevNode.next=None
```




```
lnklist.deleteEnd()
```

Class Activity

Check if the list is Empty

Implementation of Linked List in Python

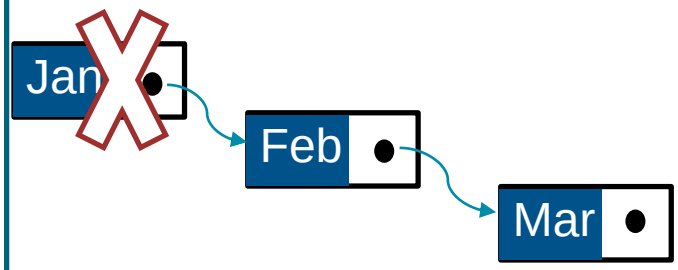
- **Check if the list is empty?**

```
Class LinkedList:
    def __init__(self):
        self.head=None

    def isEmpty(self):
        if self.head is None:
            return True
        else:
            return False
```

Implementation of Linked List in Python

- **Delete a node from the head of the list:**

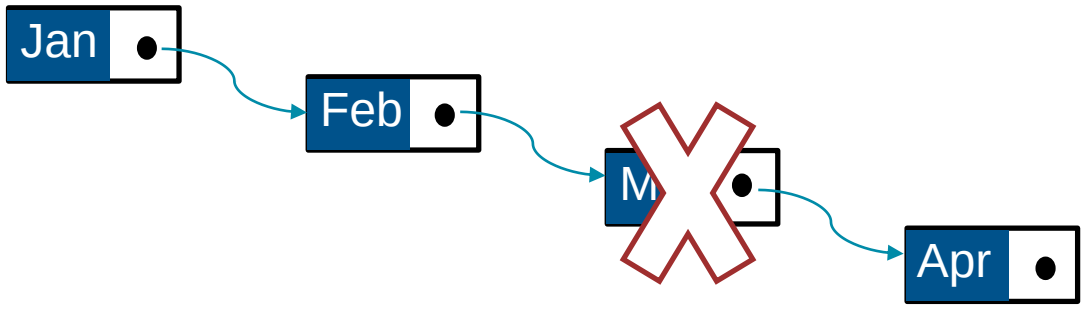
```
Class LinkedList:
    def __init__(self):
        self.head=None
    def insert(self, newNode):
        ….
    def insertHead(self, newNode):
        ….
    def insertAt(self, newNode, position):
         …
    def deleteHead(self):
        if self.isEmpty() is False:
            prevHead=self.head
            self.head=self.head.next
            prevHead.next=None
            print("The first item is deleted successfully")
        else:
            print("Linked List is empty, Delete Failed")
```



lnklist.deleteHead()

Implementation of Linked List in Python

- **Delete a node from the specific position:**

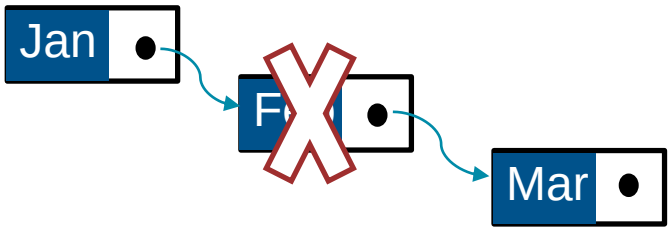


- Traverse till the node that need to be deleted
- Store the details of the previous node
- Establish a connection from the next of the previous node to the next of this node
- Make the next of this node points to **None**

Implementation of Linked List in Python

- **Delete a node from the specific position:**

```
Class LinkedList:
    def __init__(self):
        self.head=None

def deleteAt(self,position):
        currentNode=self.head
        currentPosition=0
        while True:
            if currentPosition == position:
                prevNode.next=currentNode.next
                currentNode.next=None
                break
            prevNode=currentNode
            currentNode=currentNode.next
            currentPosition +=1
```




```
lnklist.deleteEnd()
```

How about if the list is empty?

How about if the given position is not valid?