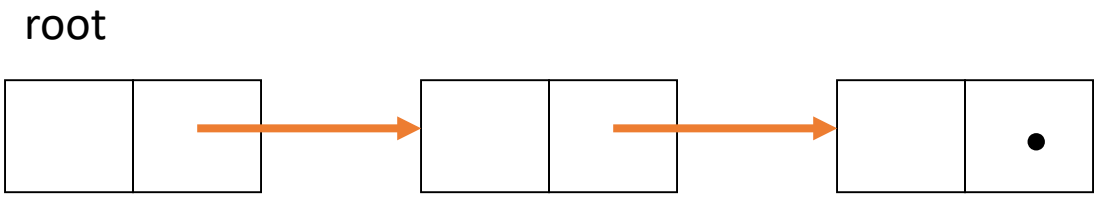
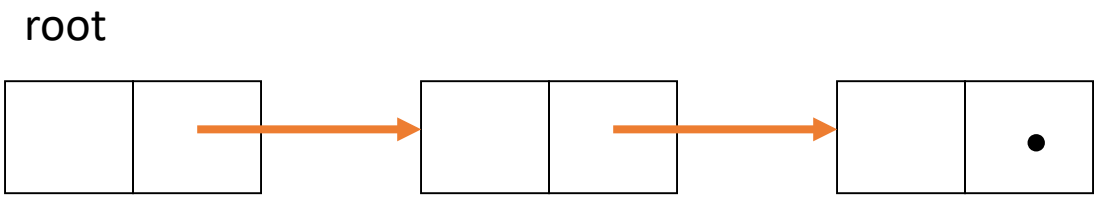
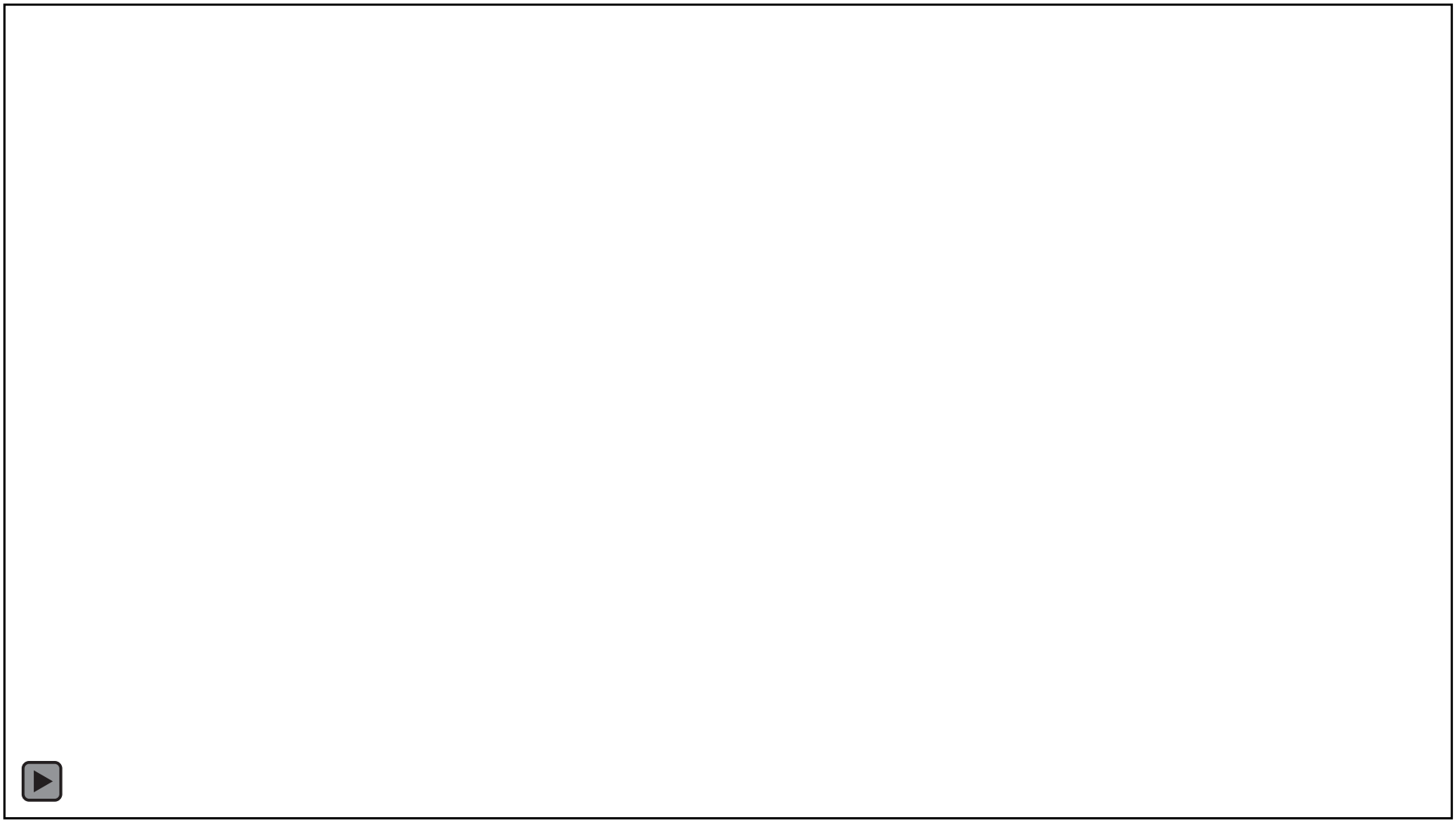
Data Structure using OOP - Linked List



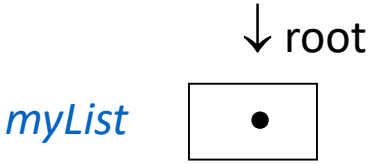
Part 1 : Creating a linked list and Adding nodes



What is a Linked List?



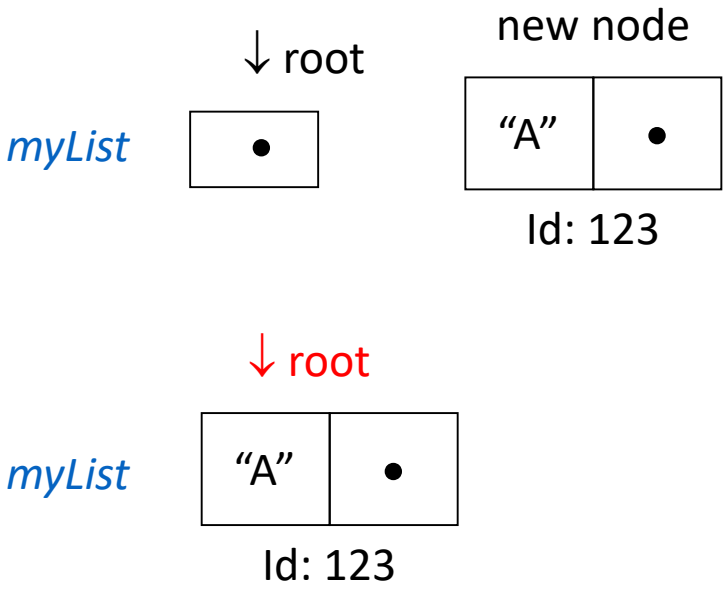
Create a new linked list



Create a class `LinkedList`:

```
class LinkedList:
    def __init__(self, root = None):
        self.root = root

>>> myList = LinkedList()
```

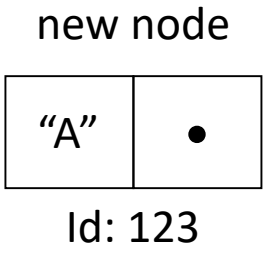
Add a first node to an ‘empty’ linked list



This is a two step process:

- Create a new node
- Point the root pointer to the newly created node

Step(1): Create a new node



Create a class called `Node`:

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

>>> new_node = Node("A")
```

Step(2): Include a method in the class `LinkedList` to add the newly created node

```
class LinkedList:
    def __init__(self, root = None):
        self.root = root
    def add(self, data):
        new_node = Node(data, self.root)
        self.root = new_node

>>> myList = LinkedList()
>>> myList.add("A")
```

What if we try to add two more nodes into the linked list?

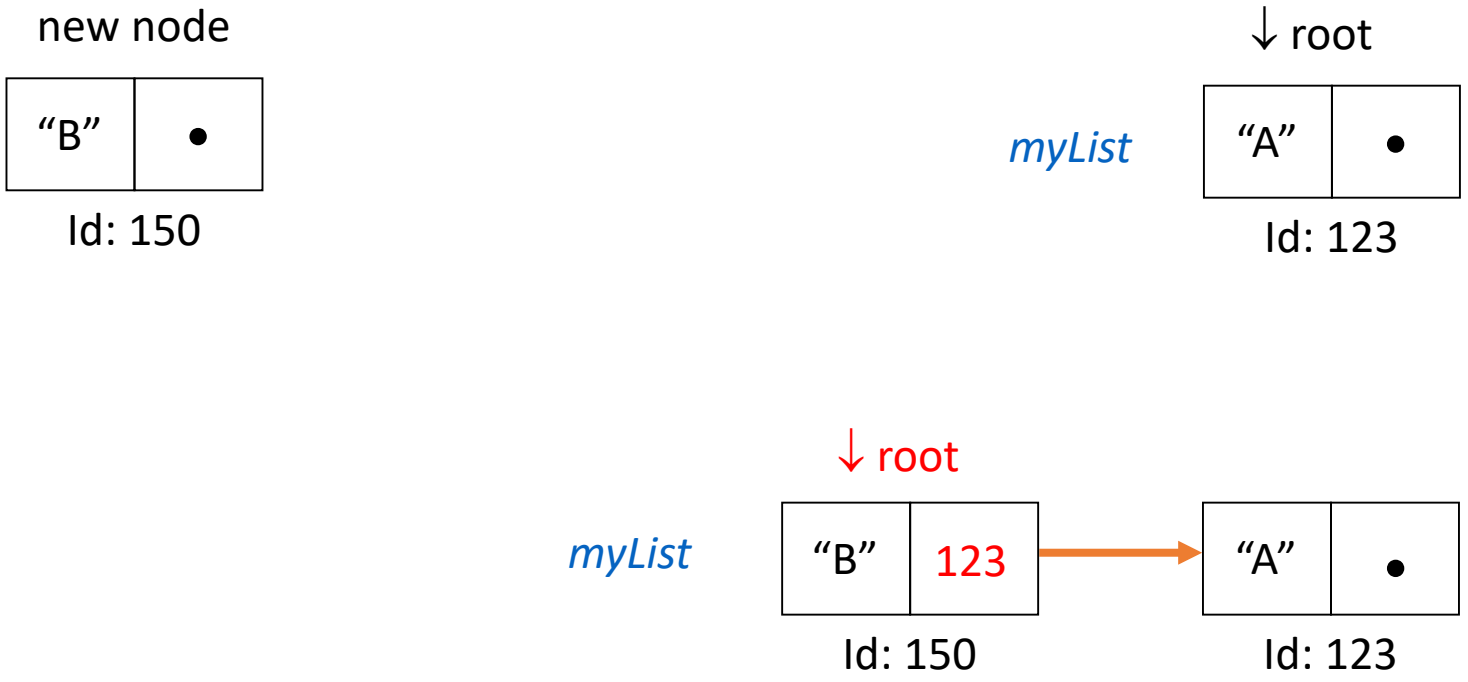
```
class LinkedList:
    def __init__(self, root = None):
        self.root = root
    def add(self, data):
        new_node = Node(data, self.root)
        self.root = new_node

>>> myList = LinkedList()
>>> myList.add("A")
>>> myList.add("B")
>>> myList.add("C")
```

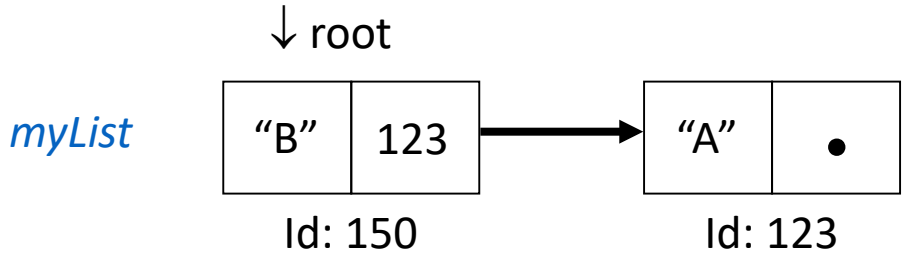
How were the nodes being added into the linked list?

>>> myList = LinkedList()

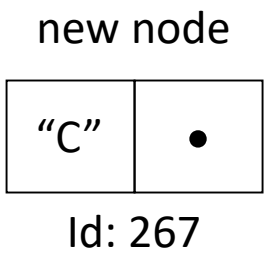
>>> myList.add("A")

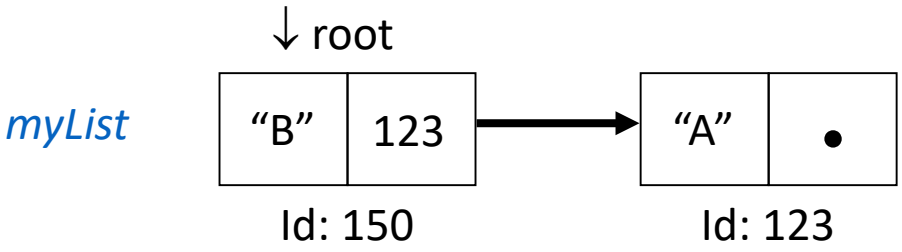
>>> myList.add("B")

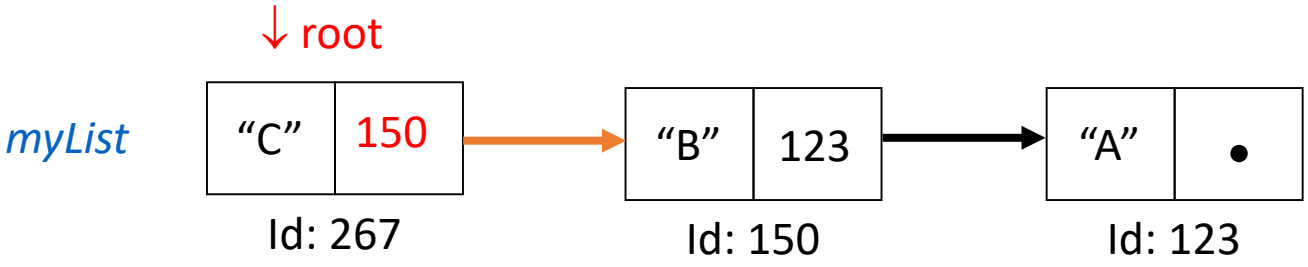



↓ root
myList
“B”
123
“A”
Id: 150
Id: 123

>>> myList.add(“C”)

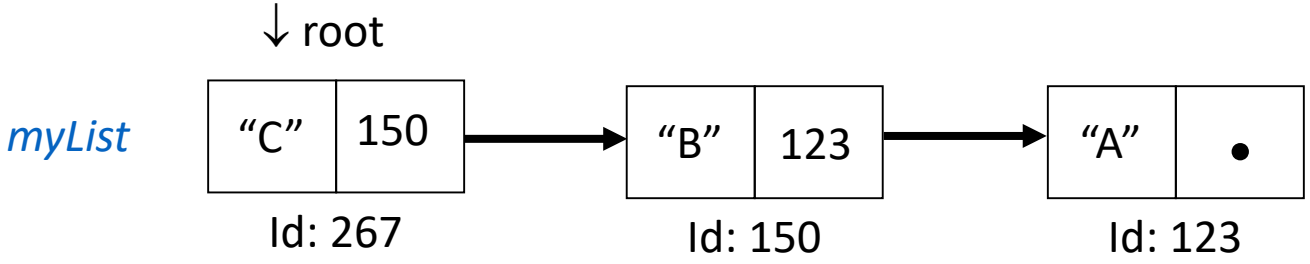

new node
“C”
Id: 267


↓ root
myList
“B”
123
“A”
Id: 150
Id: 123


↓ root
myList
“C”
150
“B”
123
“A”
Id: 267
Id: 150
Id: 123

How were the nodes being added into the linked list?

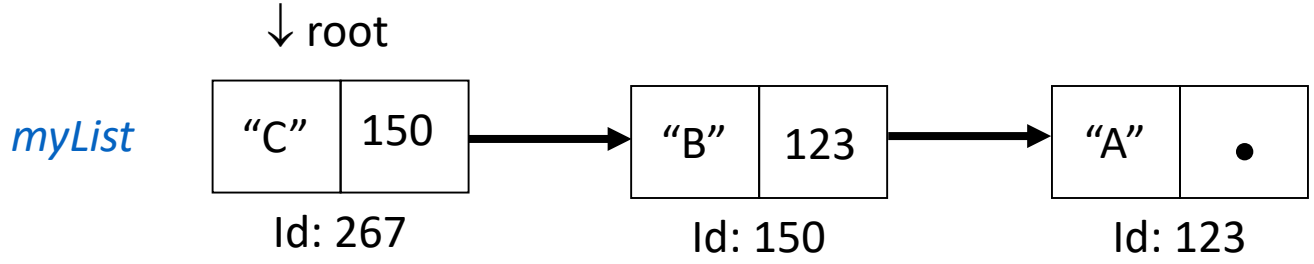
```
>>> myList = LinkedList()
>>> myList.add("A")
>>> myList.add("B")
>>> myList.add("C")
```



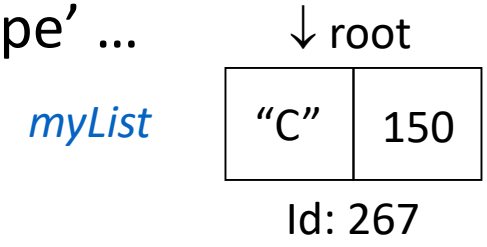
They were added to the **front** of the linked list.

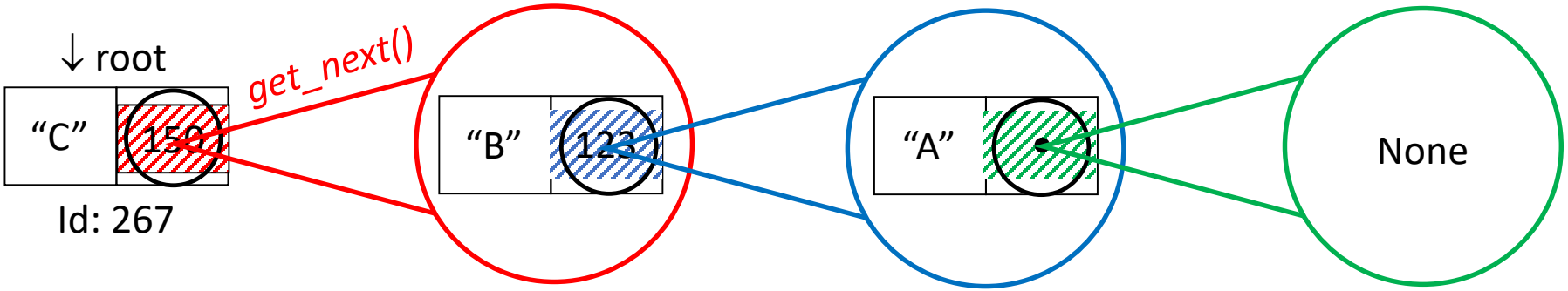
Actually, how does `myList` looks like?

Does it looks like a 'chain'?

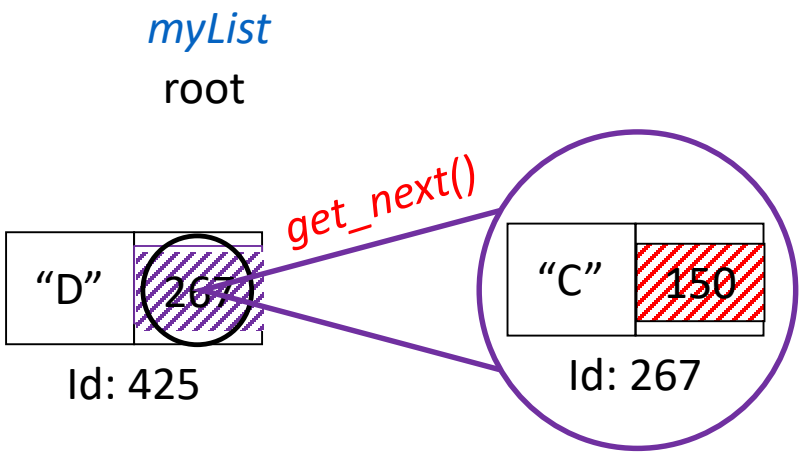


It actually looks more like an 'envelope' …

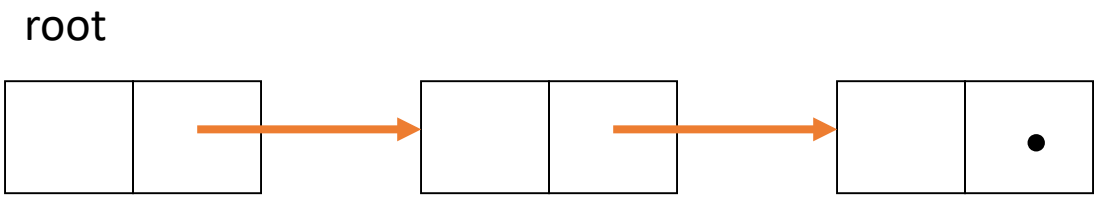




Adding a new node to the front of *myList*



Part 2 :
Traversing a linked list

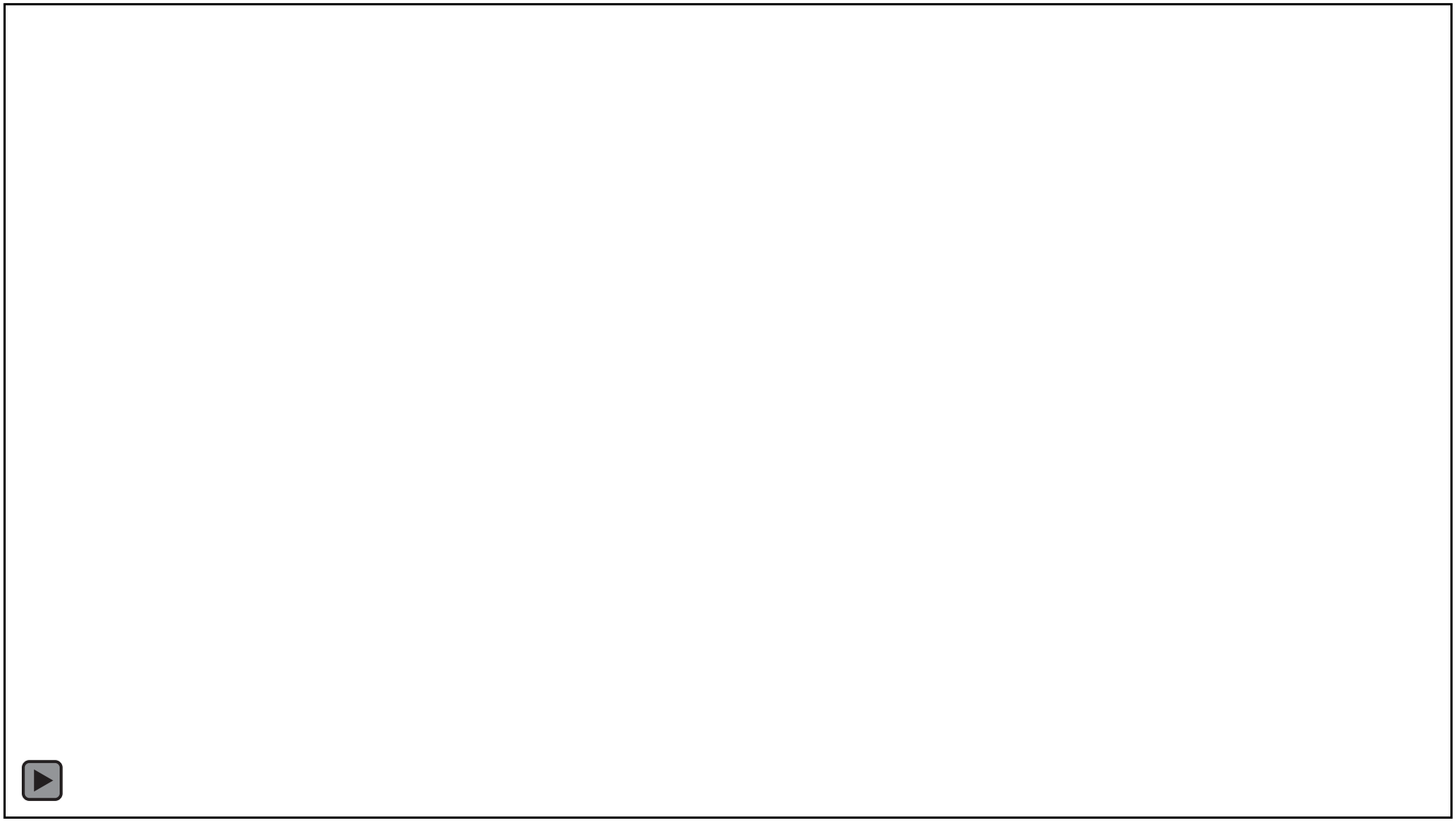


Traversing a linked list

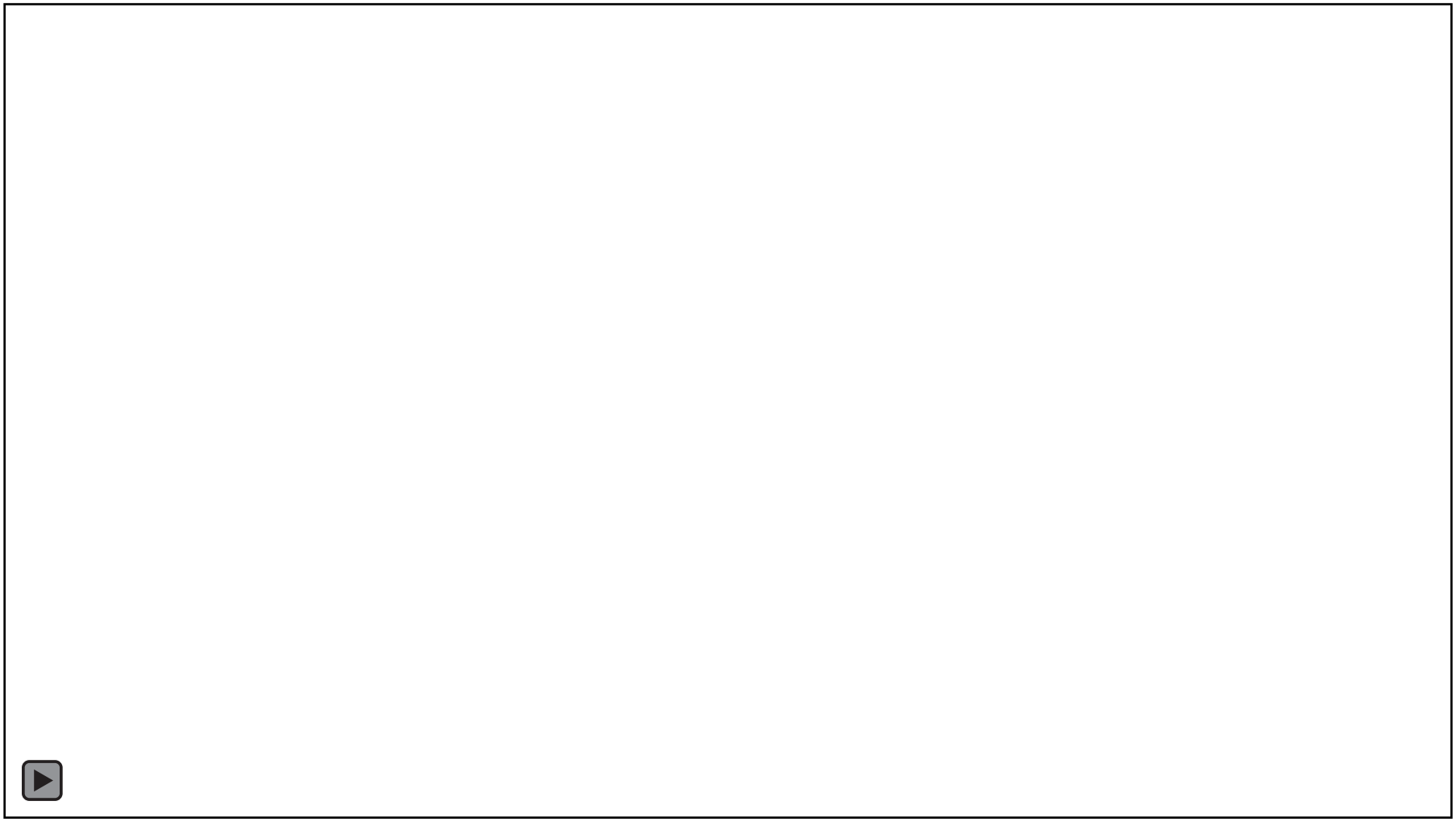
There are a few reasons why we need to traverse a linked list:

- Inserting a new node at the end of the linked list
- Searching for a node containing a particular data
- Counting number of nodes in the linked list
- Copying from one linked list to another
- Printing the ‘data’ of all the nodes in the linked list

Traversing a linked list



Traversing a linked list



Traversing a linked list for Printing

Step(1) : Check whether the linked list is empty or not

```
class LinkedList:
    ...
    def print_list(self):
        if self.root == None:
            print('The list has no node.')

>>> myList = LinkedList()
>>> myList.print_list()
```

Traversing a linked list for Printing

Step(2) : Traverse if the linked list is not empty

```
class LinkedList:
    ...
    def print_list(self):
        if self.root == None:
            print('The list has no node.')
        else:
            this_node = self.root
            while this_node:
                print(this_node.get_data())
                this_node = this_node.get_next()
```

while this_node is not None

Traversing a linked list for Printing

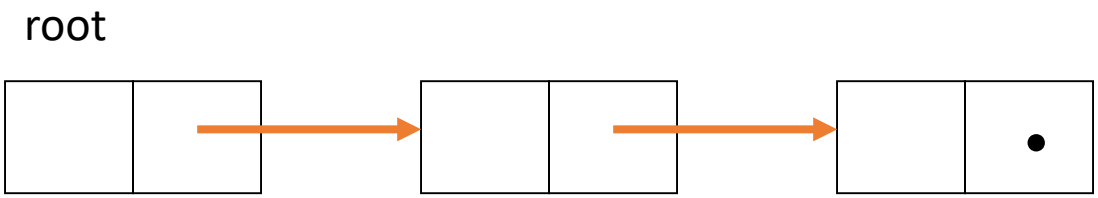
In order to traverse and print the data from the nodes, we need include two Accessors, or *Getters*, in the class `Node`:

```
class Node:
    ...
    def get_next(self):
        return self.next

    def get_data(self):
        return self.data
```

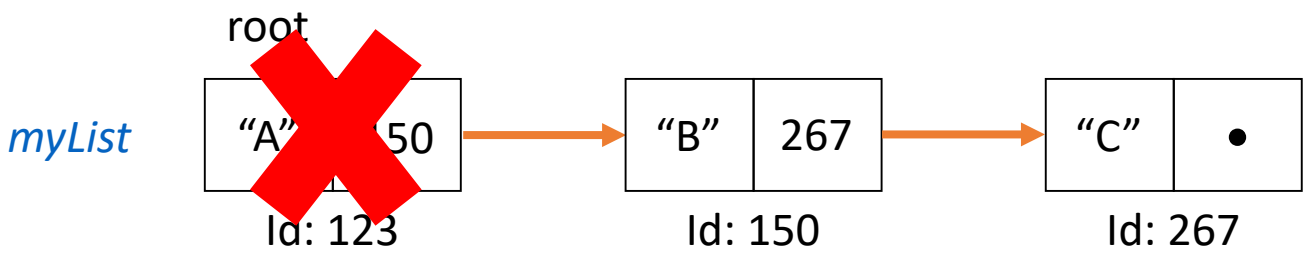
Printing a linked list

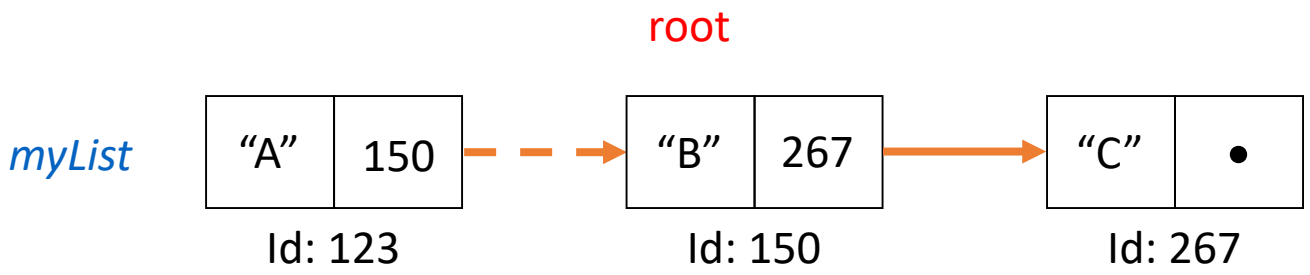
```
>>> myList = LinkedList()
>>> myList.add("A")
>>> myList.add("B")
>>> myList.add("C")
>>> myList.print_list()
```

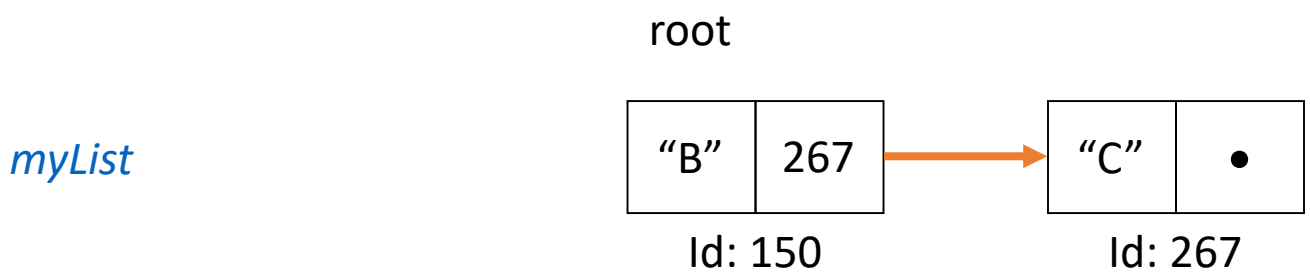
Part 3 : Removing a node from a linked list



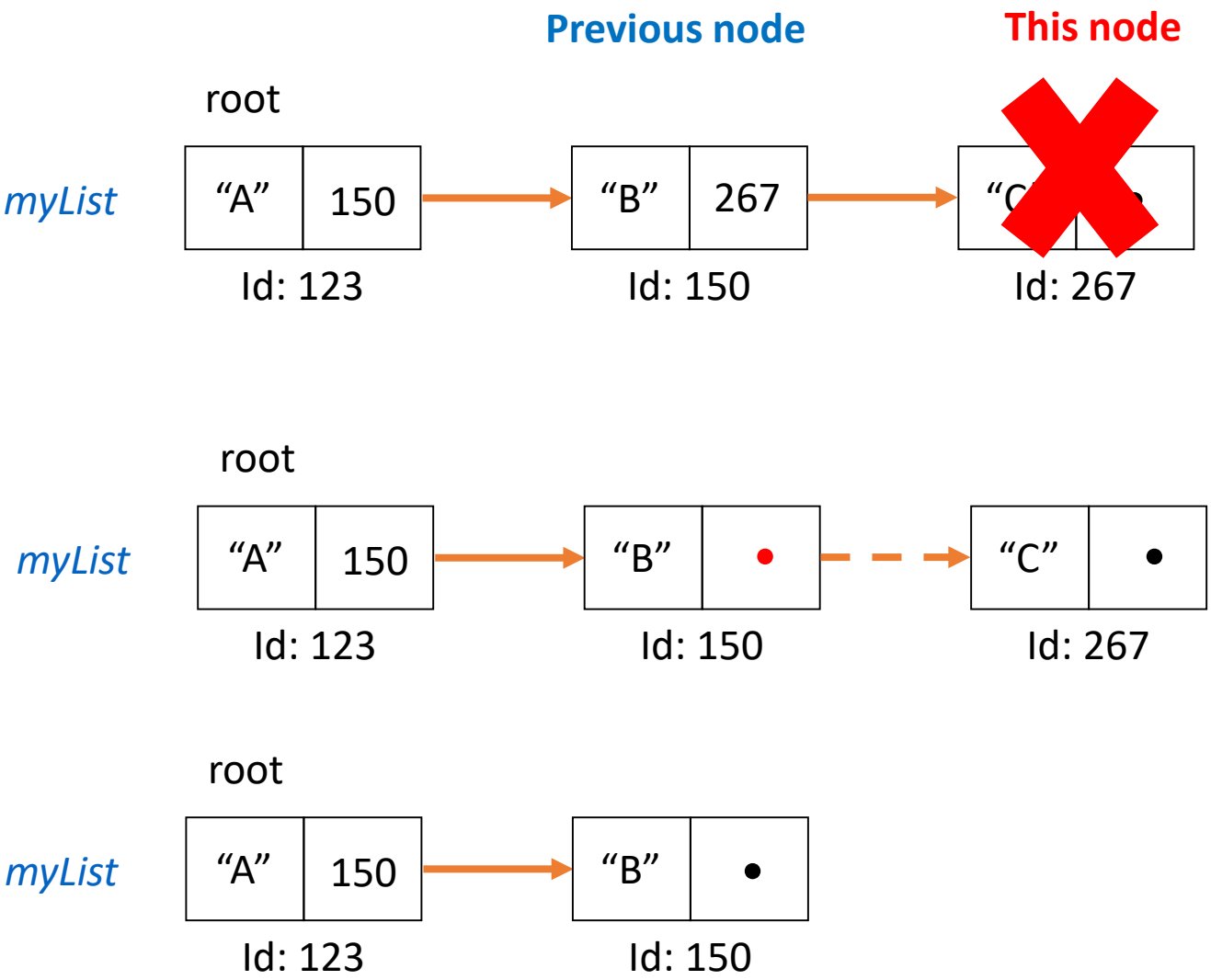
Removing the first Node



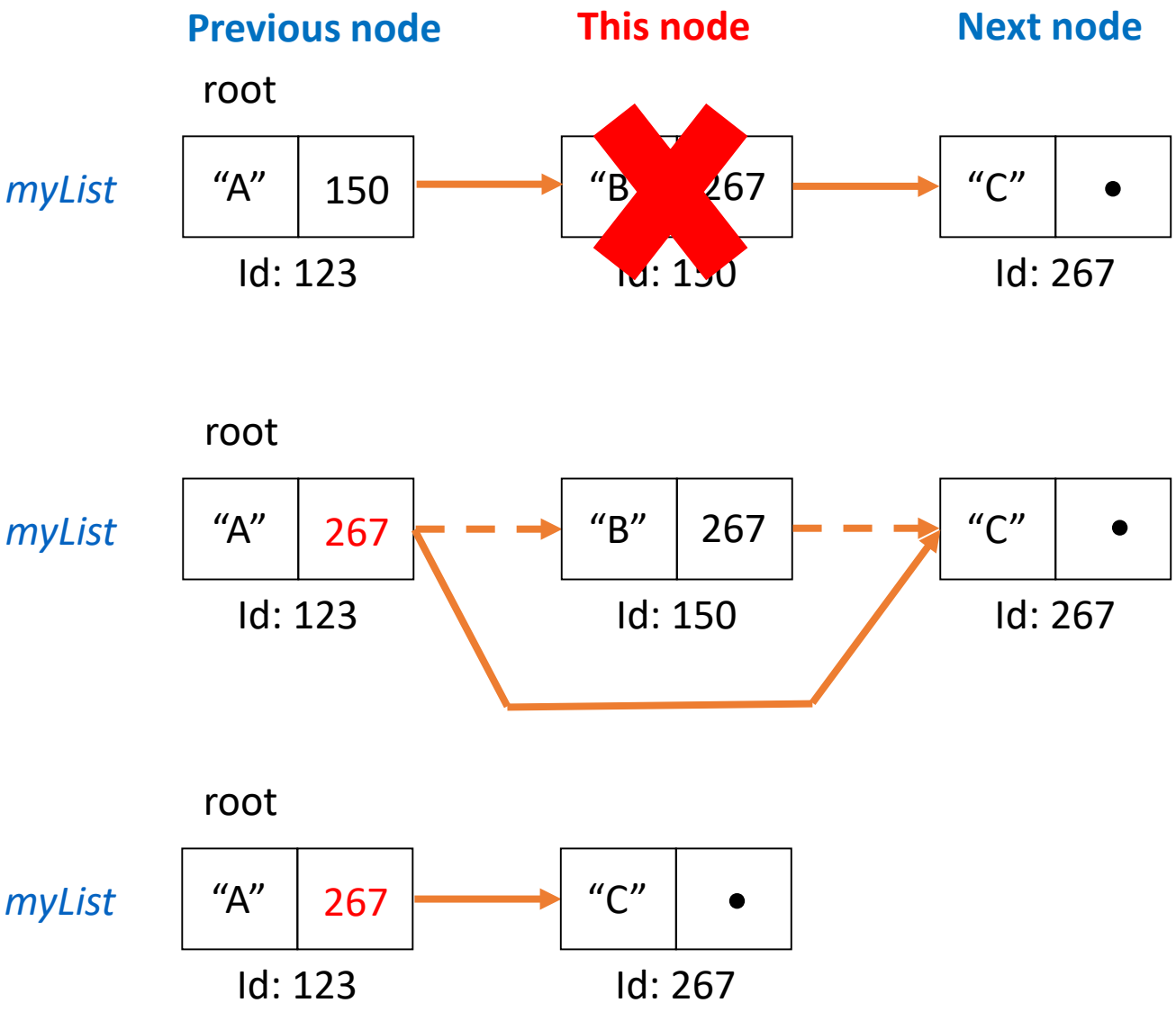




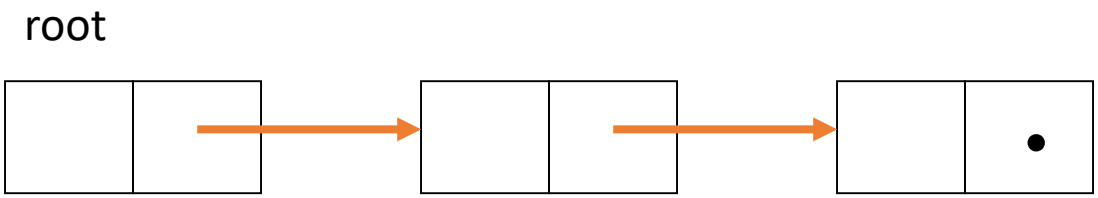
Removing the last Node



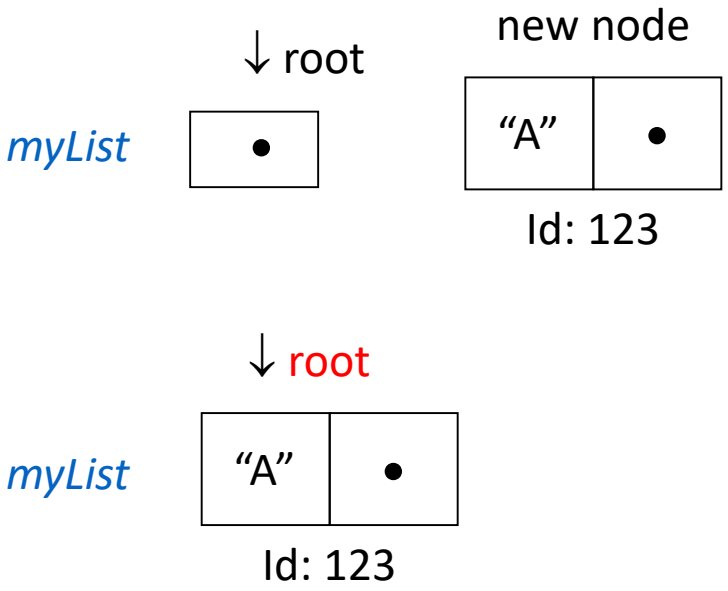
Removing a Node in between 2 nodes

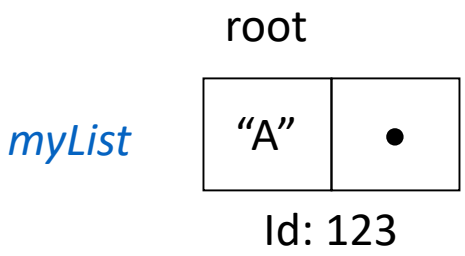


Part 4 :
Adding a node at the end of the linked list

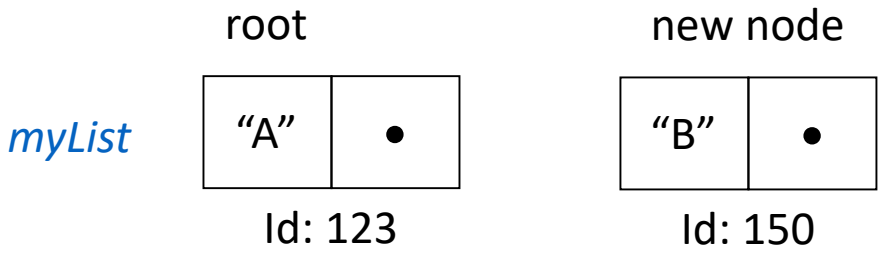


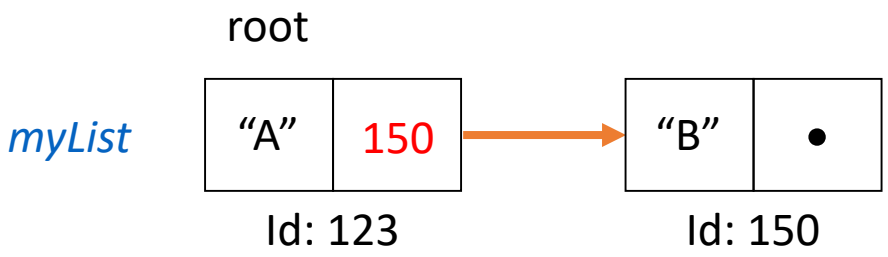
Adding more nodes to the end of the linked list

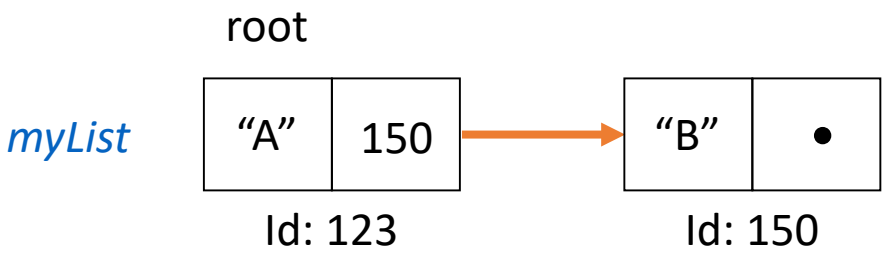




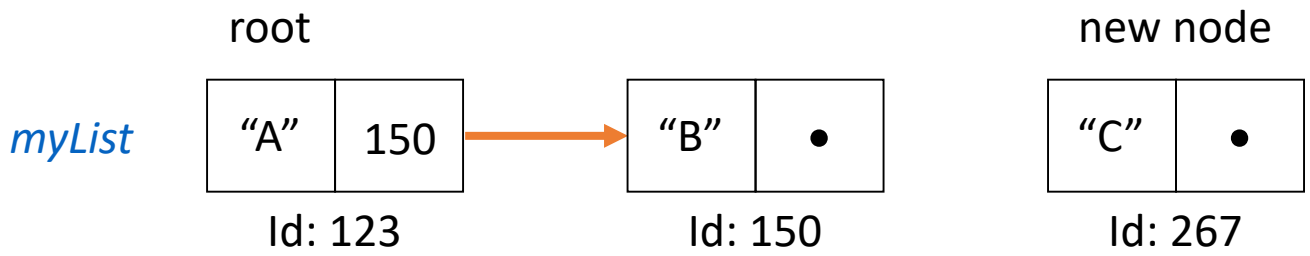
Adding a node to the end of a linked list

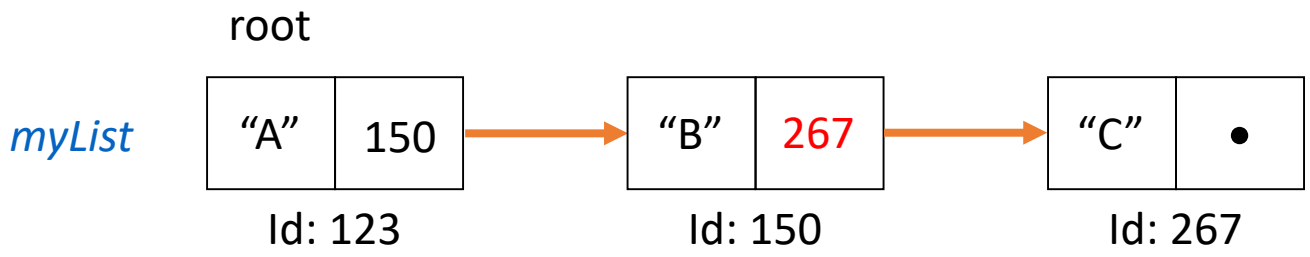




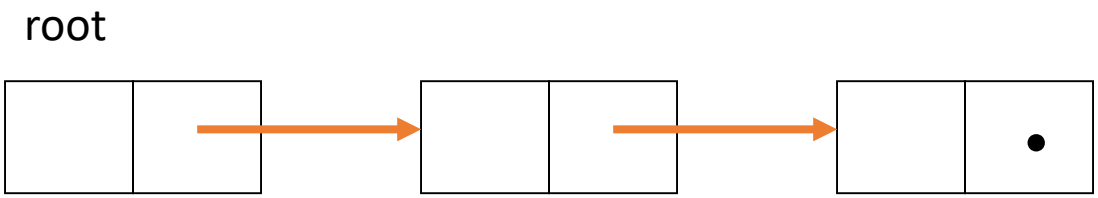


Adding another node to the end of a linked list

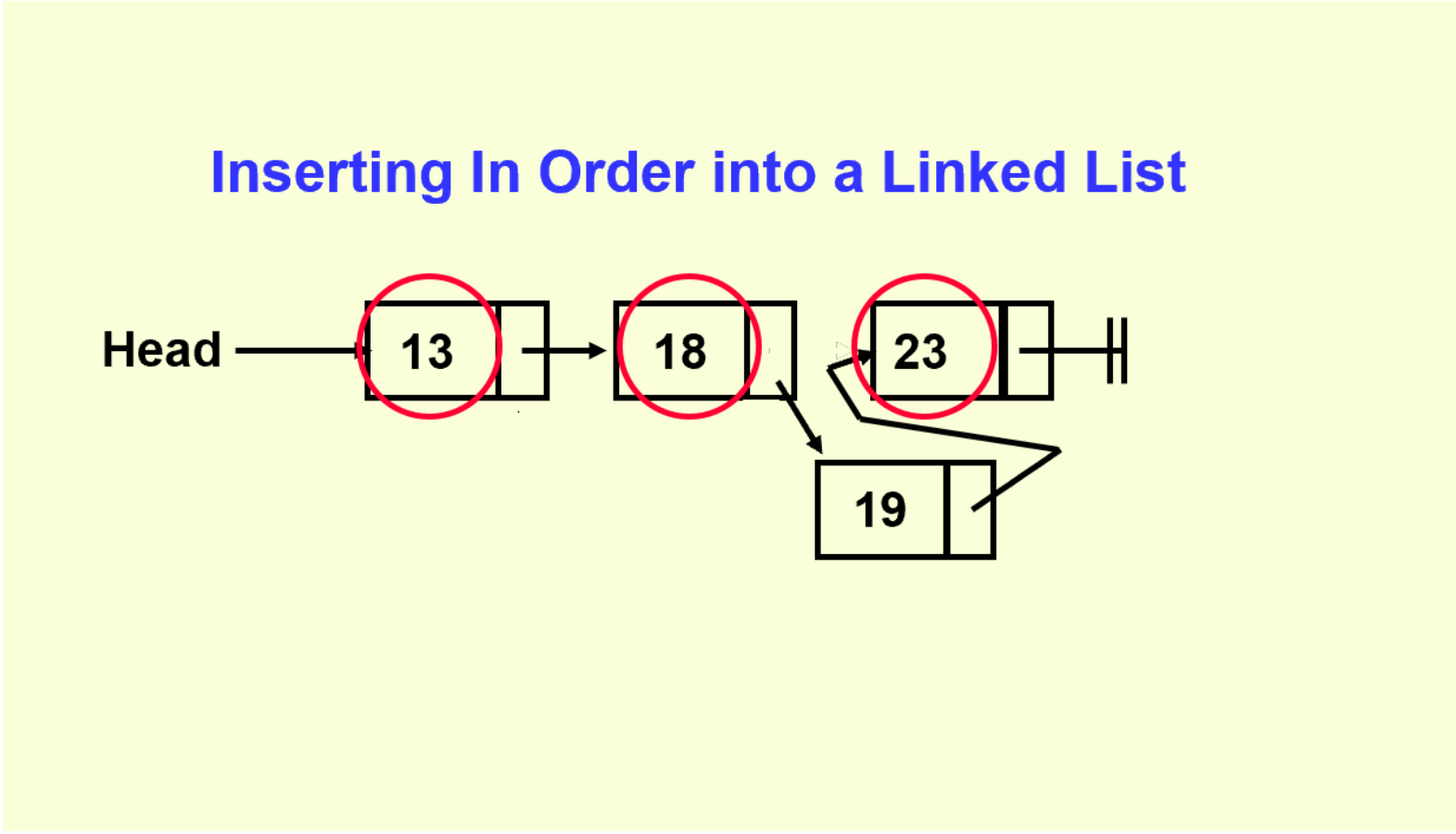




Part 5 :
Other operations on the linked list



Adding nodes in a sorted linked list



Merging two linked lists