Append and Prepend

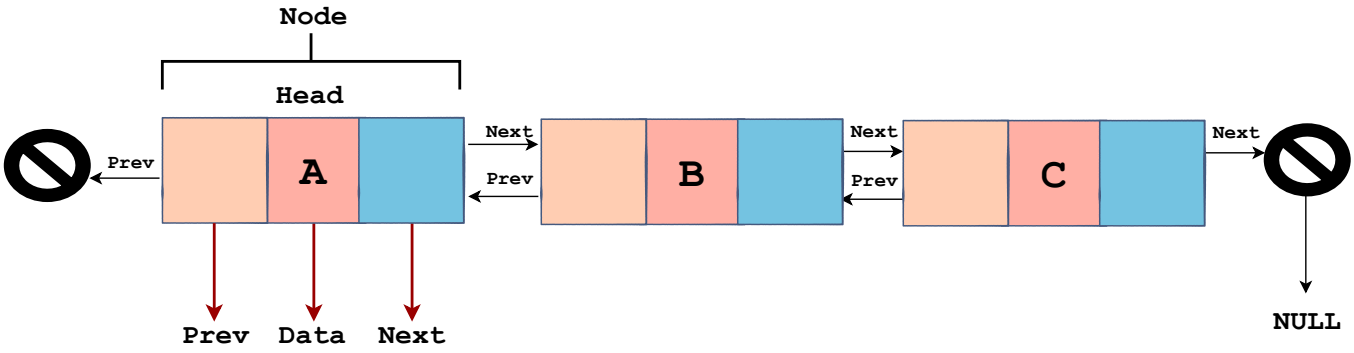
In this lesson, you will learn how to implement doubly linked lists and insert elements in it using Python.

We'll cover the following

In this lesson, we introduce the doubly linked list data structure and code up an implementation of this data structure in Python.

After that, we look at how to append (add elements to the back of the doubly linked list) and prepend (add elements to the front of the doubly linked list).

Finally, we will write a print method to verify that the append and prepend methods work as expected.



Doubly Linked List

As you can see, the doubly linked list is very similar to the singly linked list except for the difference of the previous pointer. In a doubly linked list, a node is made up of the following components:

- Data

- Next (Points to the next node)
- Prev (Points to the previous node)

As illustrated, the `prev` of the head node points to NULL while the `next` of the tail node also points to NULL.

Let's go ahead and implement the Doubly Linked List in Python:

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


class DoublyLinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        pass

    def prepend(self, data):
        pass

    def print_list(self):
        pass
```

class Node and class DoublyLinkedList

We have two classes in the code above:

1. `Node`
2. `DoublyLinkedList`

The key difference in the `Node` class of the last two chapters and of this chapter is that we have another component, `prev`, present in the `Node` class for this chapter. `prev` will keep track of the previous node in the linked list.

We also define `DoublyLinkedList` class and initialize its head to `None`. In this lesson, we will complete the implementation of the three class methods that we have specified in the code above.

append

Let's go over the implementation of `append`:

```
def append(self, data):
  if self.head is None:
    new_node = Node(data)
    new_node.prev = None
    self.head = new_node
  else:
    new_node = Node(data)
    cur = self.head
    while cur.next:
      cur = cur.next
    cur.next = new_node
    new_node.prev = cur
    new_node.next = None
```

append(self, data)

We have handled two cases of appending elements in the code above:

1. The doubly linked list is empty.
2. The doubly linked list is not empty.

If the doubly linked list is empty, i.e., `self.head` is `None`, then the execution jumps to **line 3** and we initialize `new_node` based on `data` inputted to the method. As the `new_node` will be the only node in the linked list, we will set `new_node.prev` to `None` on **line 4** and make it the head node on **line 5**.

Now let's move on to the second case. On **line 7**, we initialize `new_node` by calling the constructor of the `Node` class and passing `data` to it. Next, we need an appropriate position to locate the newly created node. Therefore, we initialize `cur` on **line 8** to the head node to traverse the doubly linked list and reach the last node in it. This is done by the `while` loop in which `cur` updates to `cur.next` and the loop runs until we reach the last node where `cur.next` will be equal to `None` (**lines 9-10**). Once the `while` loop terminates, we point the next of `cur` to `new_node` on **line 11**. As now we have appended `new_node` to the linked list, we also need to care about the previous component of `new_node` which should now point to `cur`. This step is implemented on **line 12** after which we also update `new_node.next` to `None` as it is our new last node.

`print_list`

As we need some way to test the append method, we'll go on to implement `print_list`:

```
def print_list(self):
  cur = self.head
  while cur:
    print(cur.data)
    cur = cur.next
```

print_list(self)

The `print_list` method is pretty straightforward. `cur` initially points to `self.head` (**line 2**) and then keeps printing its data (**line 4**) and updating itself to `cur.next` (**line 5**) to traverse the entire linked list. This is it!

`prepend`

It's time for us to implement the `prepend` method of `DoublyLinkedList` class. Check out the code below:

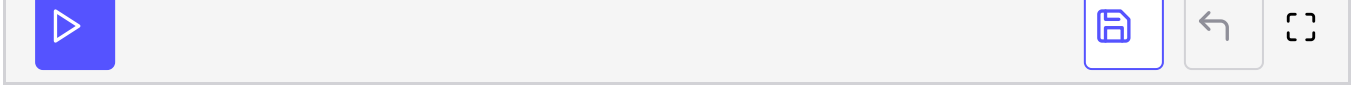
```
def prepend(self, data):
  if self.head is None:
    new_node = Node(data)
    new_node.prev = None
    self.head = new_node
  else:
    new_node = Node(data)
    self.head.prev = new_node
    new_node.next = self.head
    self.head = new_node
    new_node.prev = None
```

prepend(self, data)

The code on **lines 2-5** is pretty much the same as the code in the `append` method. Let's jump to **line 7** which will execute if we don't have an empty linked list. On **line 7**, we create `new_node` using `data` passed into the method. Now we need to position it at the start of the linked list appropriately. As the newly created node will precede the current head node and take over the head position, the previous of the current head node should point to `new_node`. This is done on **line 8**. The next of the `new_node` should then also point to the current head node so, on **line 9**, we set `new_node.next` to `self.head`. Finally, on **line 10**, we update `self.head` to `new_node` and set `new_node.prev` equal to `None` (**line 11**) as it is the new head now. This perfects our implementation to prepend an element to a doubly linked list.

Below we have the entire implementation of the `DoublyLinkedList` class with all the methods completely implemented along with a sample test case. Check it out!

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


class DoublyLinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        if self.head is None:
            new_node = Node(data)
            new_node.prev = None
            self.head = new_node
        else:
            new_node = Node(data)
            cur = self.head
            while cur.next:
                cur = cur.next
            cur.next = new_node
            new_node.prev = cur
            new_node.next = None

    def prepend(self, data):
        if self.head is None:
            new_node = Node(data)
            new_node.prev = None
            self.head = new_node
        else:
            new_node = Node(data)
            self.head.prev = new_node
            new_node.next = self.head
            self.head = new_node
            new_node.prev = None

    def print_list(self):
        cur = self.head
        while cur:
            print(cur.data)
            cur = cur.next


dllist = DoublyLinkedList()
dllist.prepend(0)
dllist.append(1)
dllist.append(2)
dllist.append(3)
dllist.append(4)
dllist.prepend(5)

dllist.print_list()
```



In the next lesson, we'll go over other methods of inserting elements in a doubly linked list. Stay tuned!