Add Node Before/After

In this lesson, you will learn how to add a node before or after a specified node.

We'll cover the following

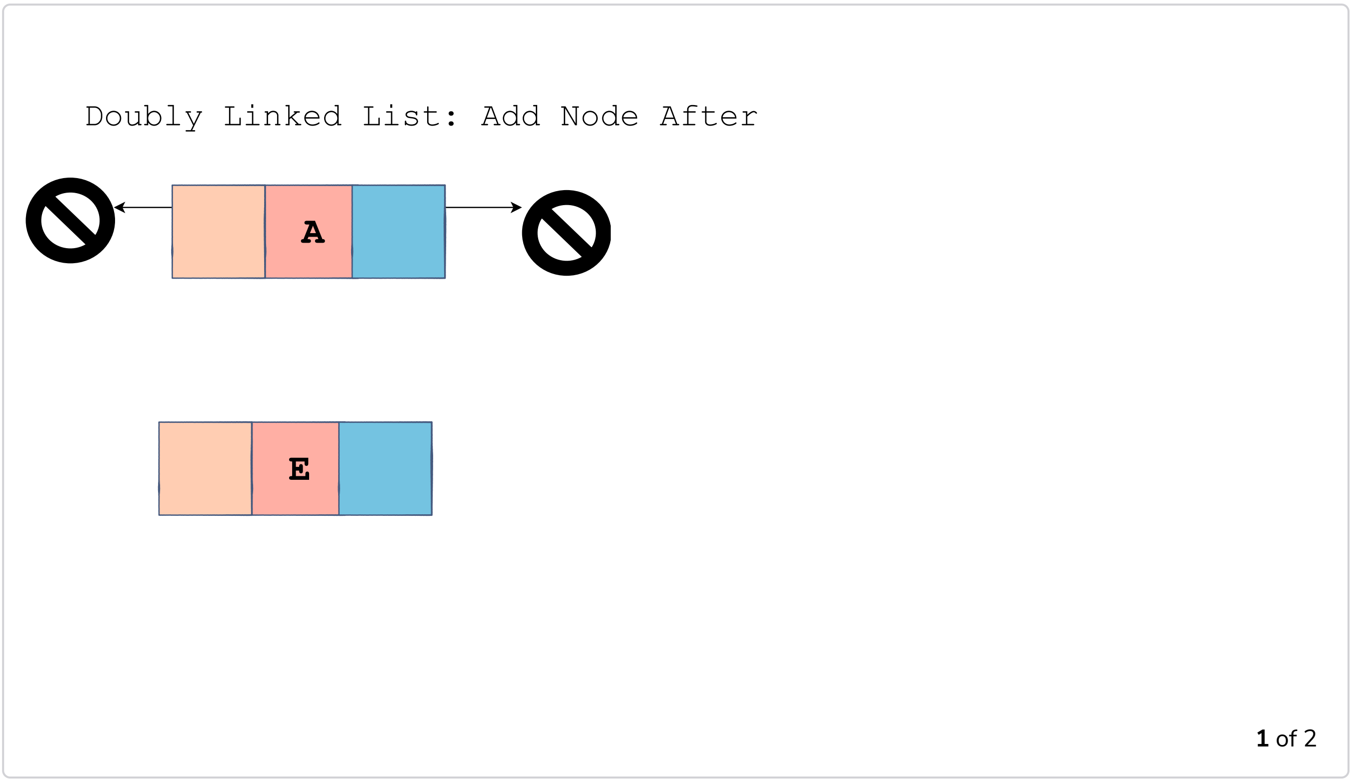
In this lesson, we consider how to add nodes either before or after a specified node in a doubly-linked list. Once we cover the concept of how to perform this action, we follow through with a Python implementation.

The general approach to solve this problem would be to traverse the doubly linked list and look for the key that we have to insert the new node after or before. Once we get to the node whose value matches the input key, we update the pointers to make room for the new node and insert the new node according to the position specified.

Add Node After

We'll break up this scenario into two cases:

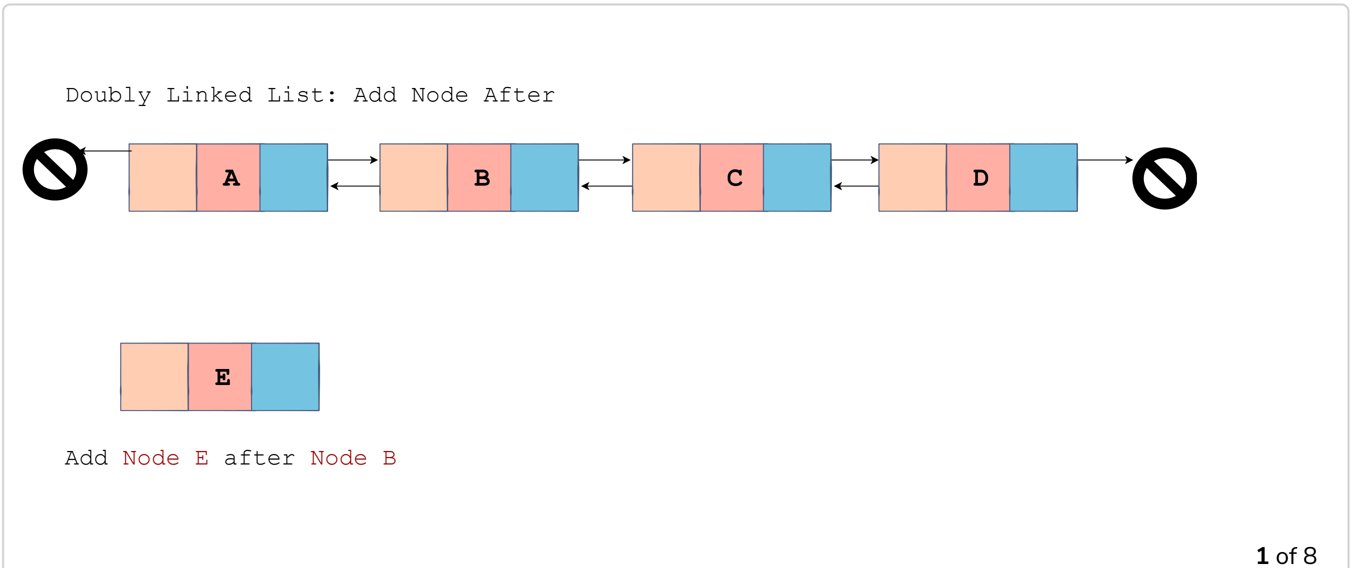
1. The node that we have to insert after the new node is the last node in the doubly linked list so that the next of that node is NULL.



In the above case, we’ll make a call to the `append` method that we implemented in the last lesson. `append` will handle everything for us and perfectly works in this scenario as we have to insert the new node after the last node, which is what `append` method does.

2. The node that we have to insert after the new node is not the last node in the doubly linked list.

Let’s look at the illustration below to check how we will handle the second case:



Hope the slides above were useful for you to understand the algorithm that we are about to code in Python. Let's go ahead and check out the code below:

```
def add_after_node(self, key, data):
  cur = self.head
  while cur:
    if cur.next is None and cur.data == key:
      self.append(data)
      return
    elif cur.data == key:
      new_node = Node(data)
      nxt = cur.next
      cur.next = new_node
      new_node.next = nxt
      new_node.prev = cur
      nxt.prev = new_node
      return
    cur = cur.next
```

add_after_node(self, key, data)

`key` passed into `add_after_node` is the key of the node after which we have to insert the new node in the linked list while `data` is for the data component of the new node to be inserted.

On **line 2**, we set `cur` to `self.head` and proceed to the `while` loop on **line 3** which runs until `cur` is `None`. As we divided our algorithm for two cases, we have divided our code into conditions to cater to these two cases in the `while` loop. First, the conditions on **line 4** check if `key` matches the data of the last node of the linked list where `cur.next` will be `None`. If it does, then we call the `append` method to append the new node to the linked list and return on **line 6**.

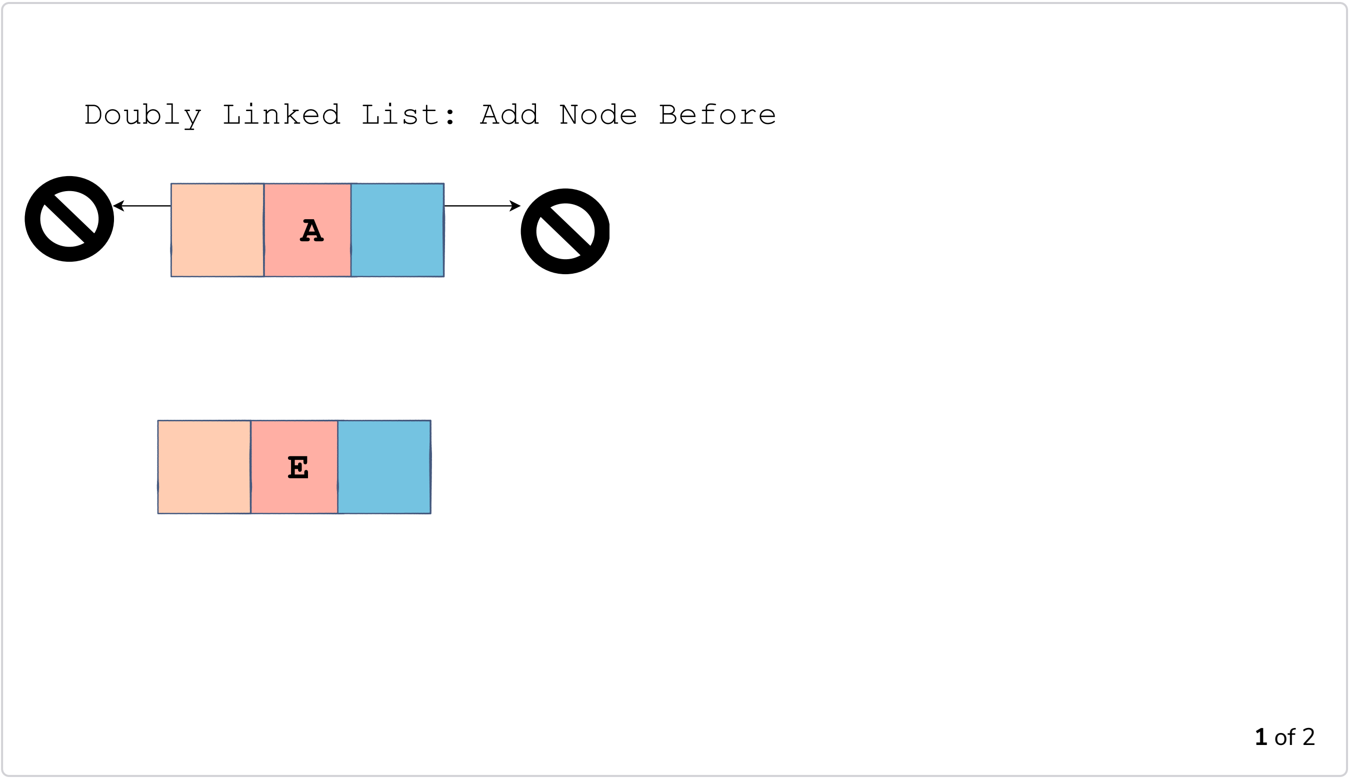
On the other hand, the condition on **line 7** will handle the second case. If we find the node after which we have to insert the new node, and it is not the last node in the linked list, then execution jumps to **line 8**. `new_node` is created based on the `data` argument passed to the method. Next, we store the next node of `cur` in a temporary variable `nxt` on **line 9**. As we have to insert `new_node` after `cur`, `cur.next` is set to `new_node` on **line 10**. On **line 11**, `new_node.next` is set to `nxt` which was previously the next node of `cur`. At this point, we have set all the next pointers, now let's update the previous pointers as we are dealing with a doubly linked list.

As we have inserted `new_node` after `cur`, the `prev` of `new_node` points to `cur` on **line 12**. `nxt` is the next node to the `new_node` so we update the `prev` of `nxt` to `new_node` on **line 13**. After this step, we return from the method on **line 14**. `cur` updates itself to `cur.next` outside the if conditions to help us traverse the linked list (**line 15**).

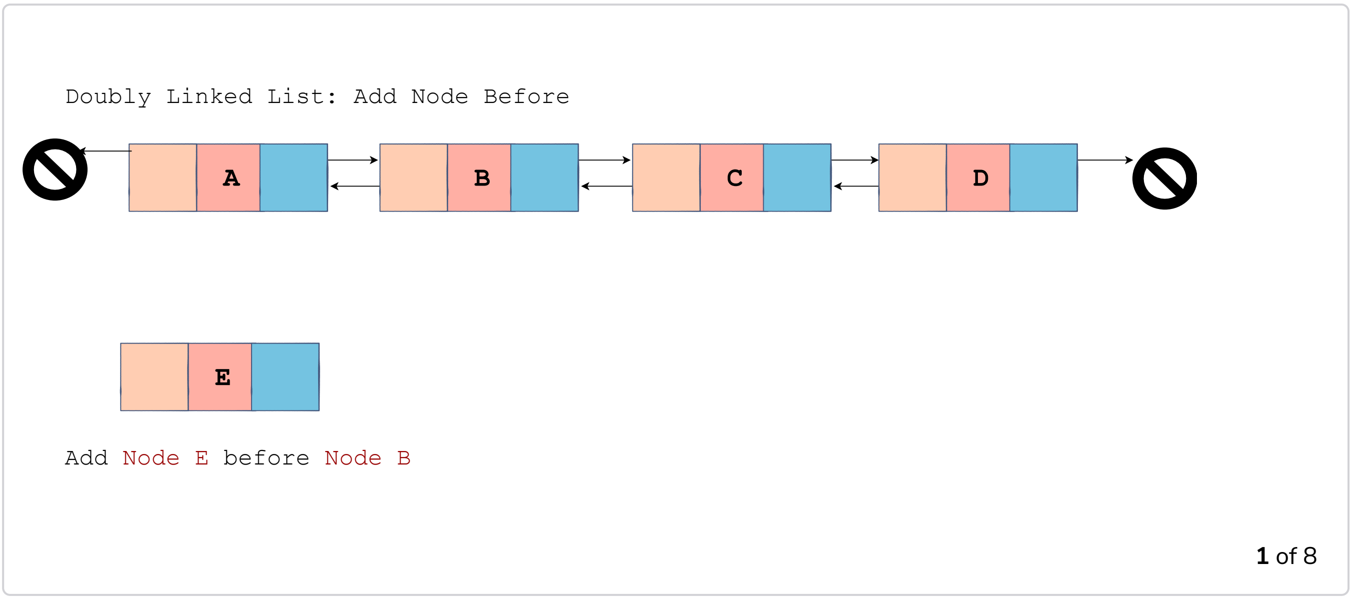
Add Node Before

Now let's discuss how to add nodes before a specified node. We will also divide this problem into two scenarios:

1. We have to insert a new node before the head node.



2. We have to insert a new node before a node that is NOT the head node.



Now let’s jump to the programming part in Python:

```
def add_before_node(self, key, data):
  cur = self.head
  while cur:
    if cur.prev is None and cur.data == key:
      self.prepend(data)
      return
    elif cur.data == key:
      new_node = Node(data)
      prev = cur.prev
      prev.next = new_node
      cur.prev = new_node
      new_node.next = cur
      new_node.prev = prev
      return
    cur = cur.next
```

add_before_node(self, key, data)

The same arguments (`self`, `key`, `data`) have been passed to `add_before_node` method as were given to `add_after_node` method. `key` is the key of the node before which we will insert the new node that we will create using `data`.

`cur` is initialized to `self.head` on **line 2**. Next, we have a `while` loop on **line 3** in which `cur` updates to `cur.next` (**line 15**) until `cur` becomes `None`. Now, we check whether or not the previous node of `cur` is `None` and whether if we want to insert our new node before `cur`. If true, then we call the `prepend` method that we implemented in the last lesson which will do the necessary steps to insert the new node before the head node, after which we return (**lines 5-6**). However, if we don't want to insert before the head node and find

the node that we want to insert before (which is not the head node), the execution jumps to **line 8**. The first thing we do is create the `new_node` by passing in `data` to the constructor from the `Node` class. On **line 9**, we store the previous node of the current node in a temporary variable `prev`. Now we need to update all the pointers to make the insertion successful. So, on **line 10**, we update the next node of `prev` to be `new_node` and as we have to insert `new_node` before `cur`, we set `cur.prev` to `new_node` on **line 11**. Let's also update the previous and the next pointers of `new_node`. As `new_node` is before `cur`, the `next` of `new_node` points to `cur` (**line 12**) and as `prev` is the previous node to `new_node` now, `new_node.prev` points to `prev` on **line 13**. On **line 14**, we return from the method after we have successfully inserted the `data`.

That was all regarding the insertion of elements in a doubly linked list. We have a test case for you in the code widget below so that you can play around will all the insertion implementations:

```
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.next = self.head
        self.head = new_node
    else:
        new_node = Node(data)
        self.head.prev = new_node

        new_node.next = self.head
        self.head = new_node

  def add_after_node(self, key, data):
    cur = self.head
    while cur:
      if cur.next is None and cur.data == key:
        self.append(data)
        return
      elif cur.data == key:
        new_node = Node(data)
        nxt = cur.next
        cur.next = new_node
        new_node.next = nxt
        new_node.prev = cur
        nxt.prev = new_node
        return
      cur = cur.next

  def add_before_node(self, key, data):
    cur = self.head
    while cur:
      if cur.prev is None and cur.data == key:
        self.prepend(data)
        return
      elif cur.data == key:
        new_node = Node(data)
        prev = cur.prev
        prev.next = new_node
        cur.prev = new_node
        new_node.next = cur
        new_node.prev = prev
        return
      cur = cur.next

  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.add_after_node(3,6)
dllist.add_before_node(4,9)

dllist.print_list()
```

So far, we have covered many different methods to insert elements in a doubly linked list. Let’s learn how to remove elements from a doubly linked list in the next lesson.