Delete Node

In this lesson, you will learn how to remove a node from a doubly linked list.

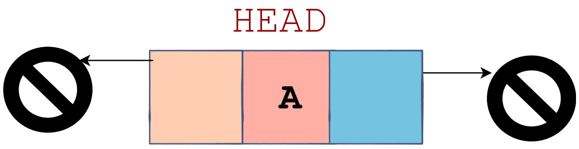
We'll cover the following

In this lesson, we consider how to delete, or, remove nodes from a doubly linked list. Once we cover the concept of how to perform this action, we follow through with a Python implementation.

We will analyze the entire implementation step by step in four parts. The following are the different cases that we can encounter while deleting a node from a doubly linked list.

Case 1: Deleting the only node present

Doubly Linked List: Delete Node -- Case 1



Delete Node A

Case 1 is where we want to delete the only node present in the linked list. As it is the single node in the linked list, then it is the head node as well. The `prev` and `next` pointer of such a node point to `None` which makes it a special case. Let’s look at the implementation for this case below.

Implementation

```
def delete(self, key):
        cur = self.head
        while cur:
            if cur.data == key and cur == self.head:
                # Case 1:
                if not cur.next:
                    cur = None
                    self.head = None
                    return
            cur = cur.next
```

delete(self, key)

Explanation

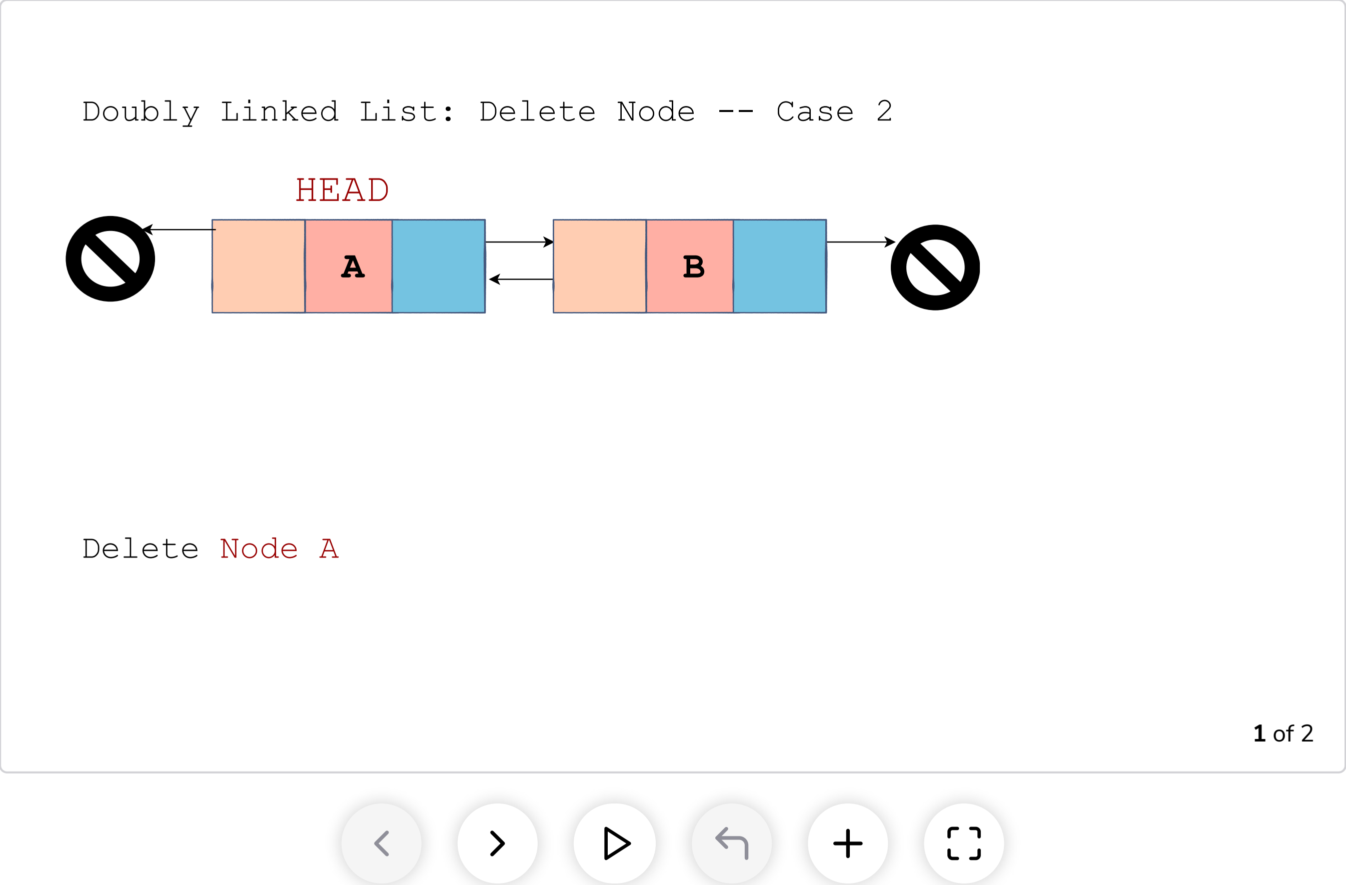
The method takes in `key` which is the key of the node to be deleted. On **line 2**, we set `cur` to `self.head` and proceed to the `while` loop on **line 3** which will run until `cur` is `None`. The execution will jump to **line 6** if the conditions on **line 4** evaluate to `True` i.e., `cur.data` is equal to `key` and the current node is also the head node. Now at this point, we have met most of the conditions for *Case 1*, and we have to do a final check. Therefore, on **line 6**, we check if the

Case 1, and we have to do a final check. Therefore, on **line 6**, we check if the next node of `cur` is `None` or not. This will confirm if it's the only node in the linked list. If it is the only node in the linked list, then `cur` has met the criteria for the type of node mentioned in *Case 1*. As a result, we set `cur` and `self.head` to `None` (**lines 7-8**) and return from the method after successfully deleting the specified node and making the linked list empty.

On **line 10**, we update `cur` to `cur.next` to traverse the linked list using the `while` loop.

Case 2: Deleting Head node

Now let's have a look at another case:


Doubly Linked List: Delete Node -- Case 2

Delete Node A

Case 2 refers to deleting the head node as in *case 1*, but now the node to be deleted is not the only node in the linked list. The head node points to another node which should replace the head node after the deletion. Let's see how we handle the second case in Python.

Implementation

```
def delete(self, key):
  cur = self.head
  while cur:
    if cur.data == key and cur == self.head:
      # Case 1:
      if not cur.next:

        cur = None
        self.head = None
        return

      # Case 2:
      else:
        nxt = cur.next
        cur.next = None
        nxt.prev = None
        cur = None
        self.head = nxt
        return
    cur = cur.next
```

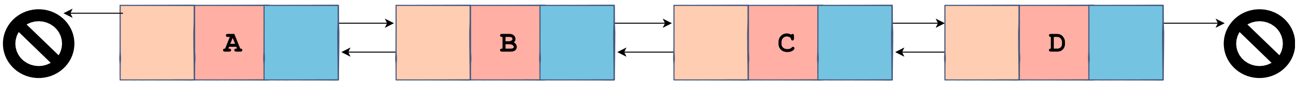
delete(self, key)

Explanation

Just like in *Case 1*, the condition on **line 4** checks for the case when the node to be deleted is the head node. As explained before, the condition on **line 6** checks if `cur` is the only node in the linked list which is a criterion for *Case 1*. If `cur.next` is not `None`, it implies that `cur` is not the only node in the linked list and the execution jumps to the `else` part (**line 12**). On **line 13**, we save the next node of `cur` in a variable named `nxt`. Now `cur` needs to be deleted, so we set `cur.next` to `None` on **line 14**. After the deletion, `nxt` will be the new head as we are deleting the current head. Therefore, we set `nxt.prev` to `None` (**line 15**) instead of `cur`, which is set to `None` in the next line. Finally, we make `nxt` the head node by setting `self.head` to `nxt` on **line 17** and return from the method in the very next line. In the code above, we have successfully deleted the head node and made the next node of the deleted head node the new head node.

Case 3: Deleting node other than head where `cur.next` is not `None`

Doubly Linked List: Delete Node -- Case 3



Delete Node B

1 of 2

In this case, we will code for a node that is not the head node or the last node but is located somewhere in between the two nodes in the linked list.

Implementation

Have a look at the code below:

```
def delete(self, key):
  cur = self.head
  while cur:
    if cur.data == key and cur == self.head:
      # Case 1:
      if not cur.next:
        cur = None
        self.head = None
        return

      # Case 2:
      else:
        nxt = cur.next
        cur.next = None
        nxt.prev = None
        cur = None
        self.head = nxt
        return

    elif cur.data == key:
      # Case 3:
      if cur.next:
          nxt = cur.next
          prev = cur.prev
          prev.next = nxt
          nxt.prev = prev
          nxt.prev = prev
          cur.next = None
          cur.prev = None
          cur = None

          return
    cur = cur.next
```

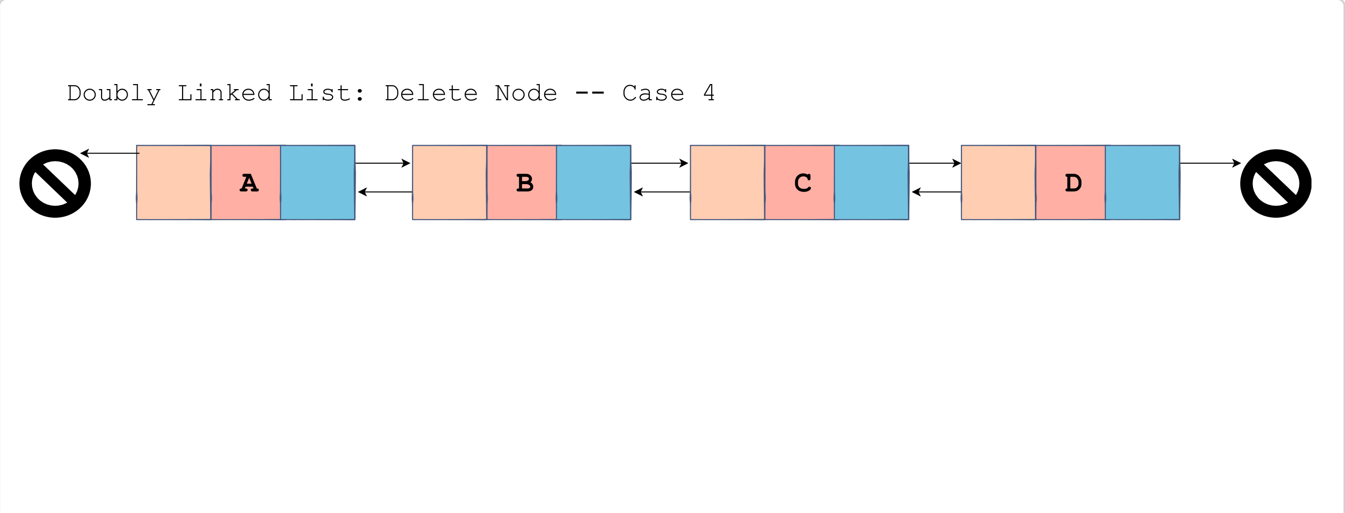
delete(self, key)

Explanation

For the explanation, we’ll only focus on the code between **line 20** and **line 30** inclusive. The condition on **line 20** will evaluate to `True` if `cur.data` in any iteration of the `while` loop becomes equal to the `key` of the node to be deleted. If `cur` is the head node, execution jumps to **line 6** but if it’s not the head node, execution jumps to **line 22**. Here, we check whether `cur` is the last node in the linked list or not. If it is not, then `cur` perfectly matches the description for *Case 3*. From **line 23** onwards comes the deletion part where we remove `cur` and make the previous node of `cur` point to the next node of `cur` and vice versa. We save the next and the previous nodes of `cur` in the variables `nxt` and `prev` (**lines 23-24**). To remove `cur` in-between `prev` and `next`, we set `prev.next` to `nxt` (**line 25**) which was previously set to `cur`, and `nxt.prev` updates to `prev` from `cur` on **line 26**. On **lines 27-29**, we set `cur`, `cur.next`, and `cur.prev` to `None` to remove `cur` completely and then return from the method on **line 30**.

Case 4: Deleting node other than head where `cur.next` is `None`

Finally, we have come to the last case where we are deleting a node that is not the head node and where the next node points to `None`. This is essentially the last node in the doubly linked list.



1 of 2

Implementation

Let’s look at the Python implementation for the case illustrated above:

```
def delete(self, key):
  cur = self.head
  while cur:
    if cur.data == key and cur == self.head:
      # Case 1:
      if not cur.next:
        cur = None
        self.head = None
        return

      # Case 2:
      else:
        nxt = cur.next
        cur.next = None
        nxt.prev = None
        cur = None
        self.head = nxt
        return

    elif cur.data == key:
      # Case 3:
      if cur.next:
          nxt = cur.next
          prev = cur.prev
          prev.next = nxt
          nxt.prev = prev
          cur.next = None
          cur.prev = None
          cur = None
          return

        # Case 4:
      else:
          prev = cur.prev
          prev.next = None
          cur.prev = None
          cur = None
          return
    cur = cur.next
```

delete(self, key)

Explanation

This case is pretty straightforward, and we jump to its code on lines **33-38** if the following conditions are met in any iteration of the `while` loop:

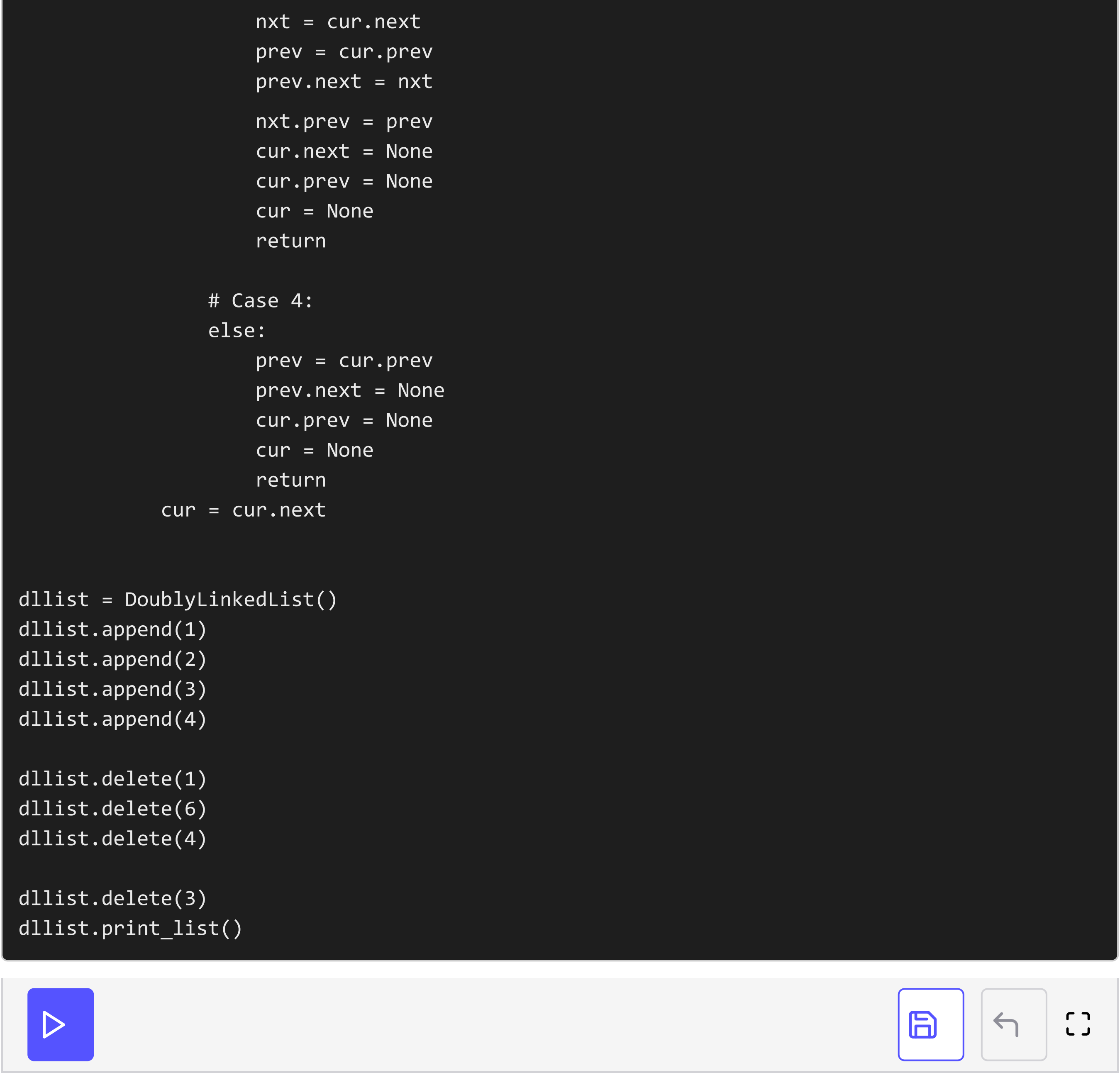
- `cur.data` is equal to `key`
- `cur` is not equal to `self.head`
- `cur.next` is not `None`

In this case, we need to care about the previous node of `cur` which will be the new last node and will now point to `None` instead of `cur`. Therefore, we set `prev` equal to `cur.prev` on **line 34** and then set its `next` to `None` on **line 35**. In the next lines (**lines 36-37**), we just set `cur.prev` and `cur` equal to `None` to remove them out of the linked list and return from the method on **line 38**.

That was all about deleting a node in the doubly linked list. I hope you were able to understand the implementation and explanation for each case.

In the code widget below, we test the `delete` method with a sample test case. Go ahead and play around with the implementation!

```
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

    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 delete(self, key):
        cur = self.head
        while cur:
            if cur.data == key and cur == self.head:
                # Case 1:
                if not cur.next:
                    cur = None
                    self.head = None
                    return

                # Case 2:
                else:
                    nxt = cur.next
                    cur.next = None
                    nxt.prev = None
                    cur = None
                    self.head = nxt
                    return

            elif cur.data == key:
                # Case 3:
                if cur.next:
```



```
                    nxt = cur.next
                    prev = cur.prev
                    prev.next = nxt
                    nxt.prev = prev
                    cur.next = None
                    cur.prev = None
                    cur = None
                    return

                # Case 4:
                else:
                    prev = cur.prev
                    prev.next = None
                    cur.prev = None
                    cur = None
                    return
            cur = cur.next


dllist = DoublyLinkedList()
dllist.append(1)
dllist.append(2)
dllist.append(3)
dllist.append(4)

dllist.delete(1)
dllist.delete(6)
dllist.delete(4)

dllist.delete(3)
dllist.print_list()
```

In the next lesson, we’ll have a look at how to reverse a doubly linked list.