Deletion by Position

In this lesson, you will learn how to delete a node at a given position in a linked list.

WE'LL COVER THE FOLLOWING

We will solve this problem in a very similar way as we have done in the last lesson.

Cases to Consider

Again, we'll consider two cases while writing our code:

1. Node to be deleted is at position 0
2. Node to be deleted is not at position 0

The overall logic will stay the same as in the previous lesson except that we'll change the code a bit to cater to position rather than a key.

Implementation

Without any further ado, let's jump to the implementation:

```
def delete_node_at_pos(self, pos):
  if self.head:
    cur_node = self.head
    if pos == 0:
      self.head = cur_node.next
      cur_node = None
      return

    prev = None
    count = 0
    while cur_node and count != pos:
      prev = cur_node
```

```
        prev = cur_node
        cur_node = cur_node.next
        count += 1


    if cur_node is None:
        return

    prev.next = cur_node.next
    cur_node = None
```

delete_node_at_pos(self, pos)

Explanation

The `delete_node_at_pos` takes in `pos` as one of the input parameters.

First of all, we check on **line 2** if the linked list is an empty list or not. We only proceed to **line 3** if `self.head` is not `None`.

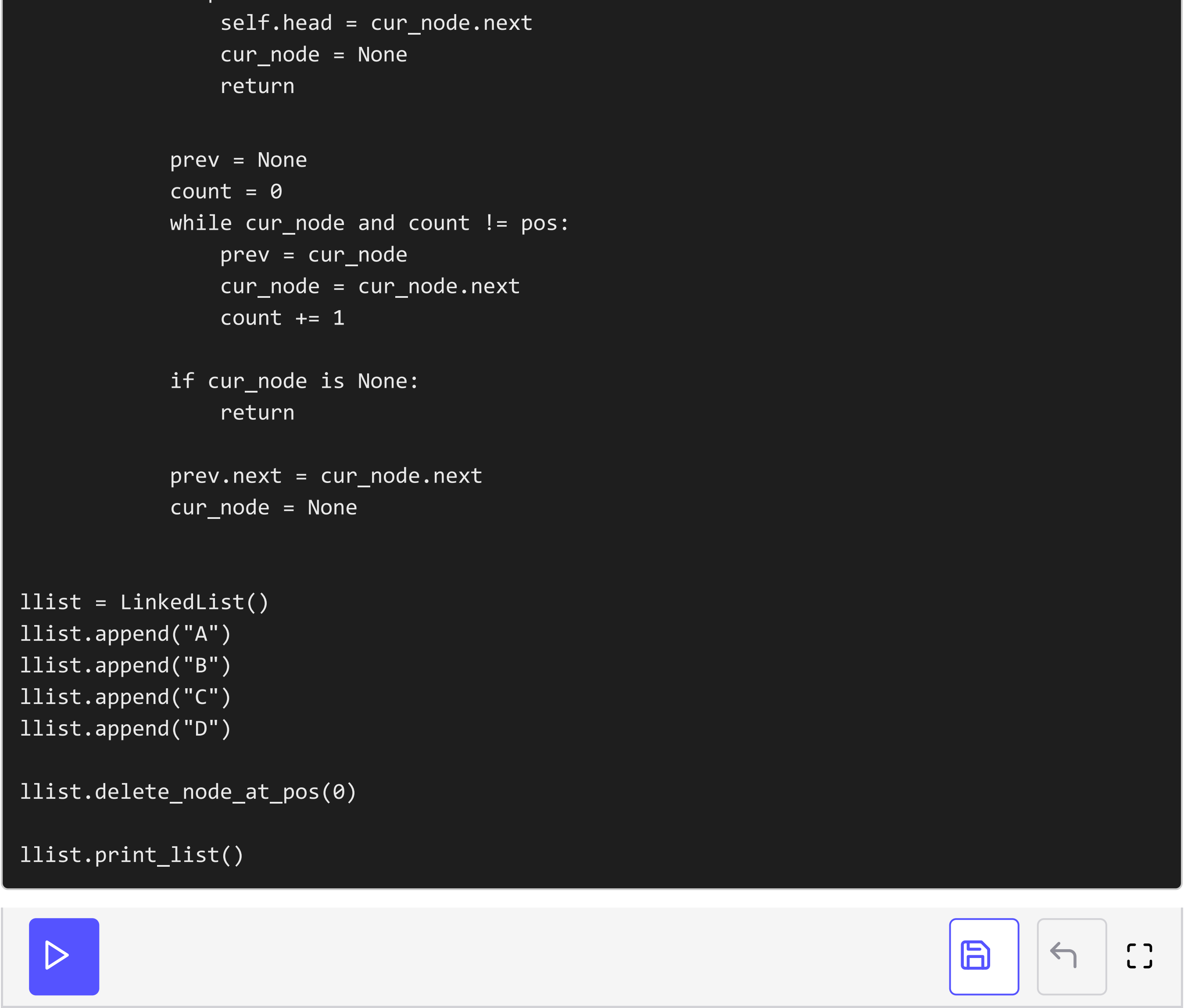
As we discussed before, we need to handle the case where `pos` will equal `0`. If `pos` equals `0`, it essentially means that we want to delete the head node.

On **line 3**, `cur_node` is initialized to the head node. Next, we check if `pos` is `0` or not. If it is, we update the head node to the next node of `cur_node`, set `cur_node` to `None`, and return from the method (**lines 5-7**). On the other hand, if we are deleting a node at a position other than the head node, we will proceed to **line 9**. We declare `prev` and set it to `None` and on **line 10**, we initialize `count` to `0`. Now we traverse the linked list by updating `prev` and `cur_node` (**lines 12-13**) and increment `count` by `1` on **line 14**. The `while` loop will terminate if `cur_node` becomes `None` or `count` becomes equal to `pos` which will imply that `cur_node` will be the node that we want to delete.

The code on **lines 16-20** is precisely the same as in the `delete_node` class method.

I hope everything's been clear up until now. You can practice this method more by playing around with it in the coding widget below:

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


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

    def print_list(self):

        cur_node = self.head
        while cur_node:
            print(cur_node.data)
            cur_node = cur_node.next

    def append(self, data):
        new_node = Node(data)

        if self.head is None:
            self.head = new_node
            return

        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    def prepend(self, data):
        new_node = Node(data)

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

    def insert_after_node(self, prev_node, data):

        if not prev_node:
            print("Previous node does not exist.")
            return

        new_node = Node(data)

        new_node.next = prev_node.next
        prev_node.next = new_node

    def delete_node(self, key):

        cur_node = self.head

        if cur_node and cur_node.data == key:
            self.head = cur_node.next
            cur_node = None
            return

        prev = None
        while cur_node and cur_node.data != key:
            prev = cur_node
            cur_node = cur_node.next

        if cur_node is None:
            return

        prev.next = cur_node.next
        cur_node = None

    def delete_node_at_pos(self, pos):
        if self.head:
            cur_node = self.head

            if pos == 0:
```



```
                self.head = cur_node.next
                cur_node = None
                return

            prev = None
            count = 0
            while cur_node and count != pos:
                prev = cur_node
                cur_node = cur_node.next
                count += 1

            if cur_node is None:
                return

            prev.next = cur_node.next
            cur_node = None


llist = LinkedList()
llist.append("A")
llist.append("B")
llist.append("C")
llist.append("D")

llist.delete_node_at_pos(0)

llist.print_list()
```

class Node and class LinkedList

That was all regarding deleting nodes from a singly linked list. In the next lesson, we'll learn how to calculate the length of a linked list.