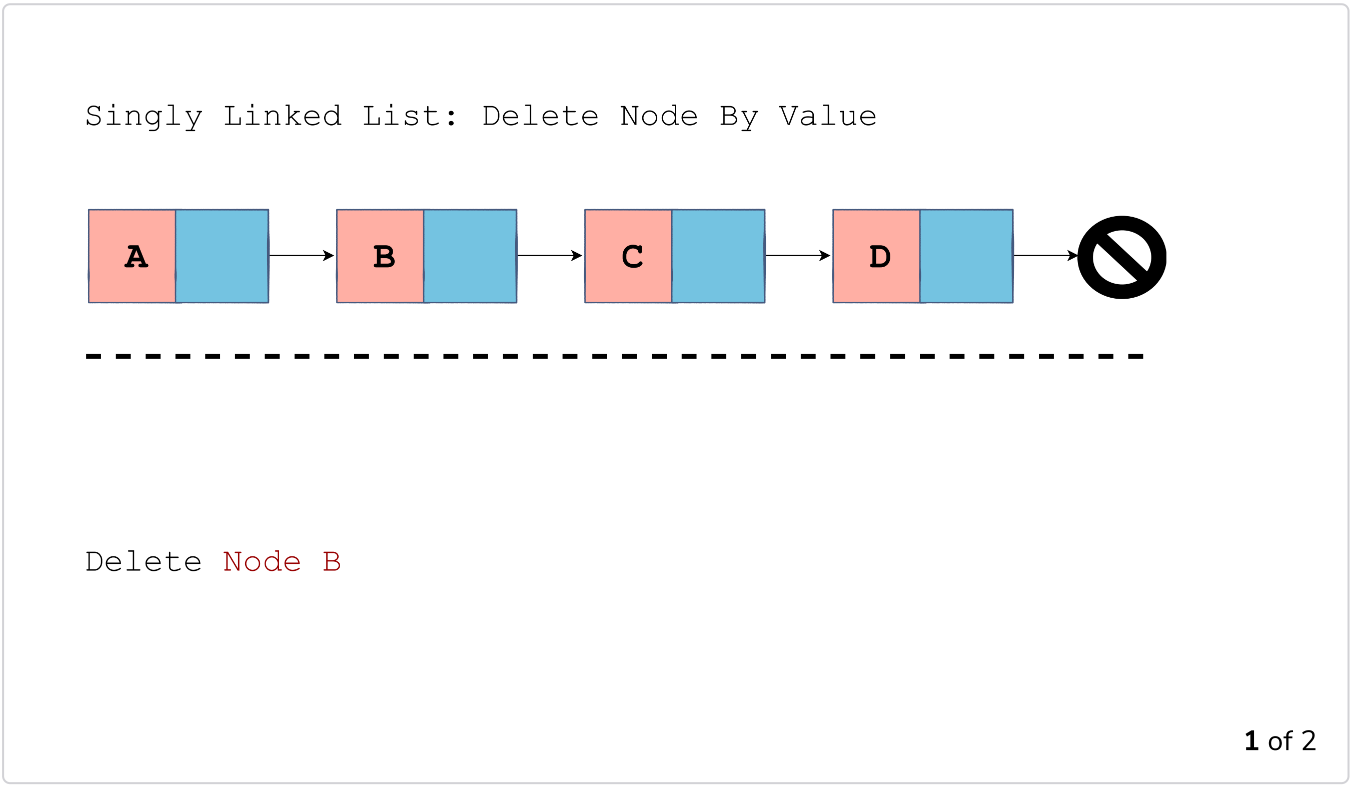
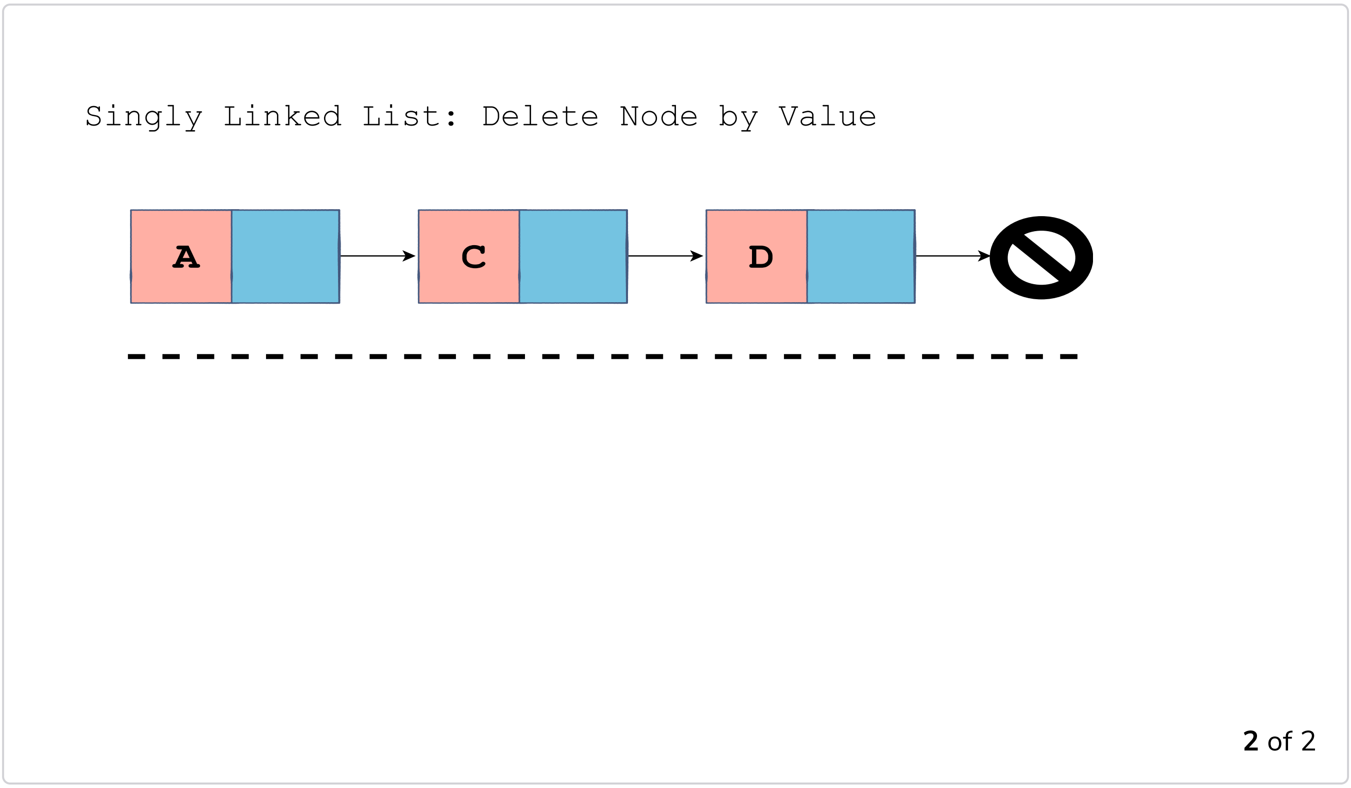
Deletion by Value

In this lesson, we will learn how to delete a node based on a value from a linked list.

WE'LL COVER THE FOLLOWING

In this lesson, we will investigate singly-linked lists by focusing on how one might delete a node in the linked list. In summary, to delete a node, we'll first find the node to be deleted by traversing the linked list. Then, we'll delete that node and update the rest of the pointers. That's it!





Algorithm

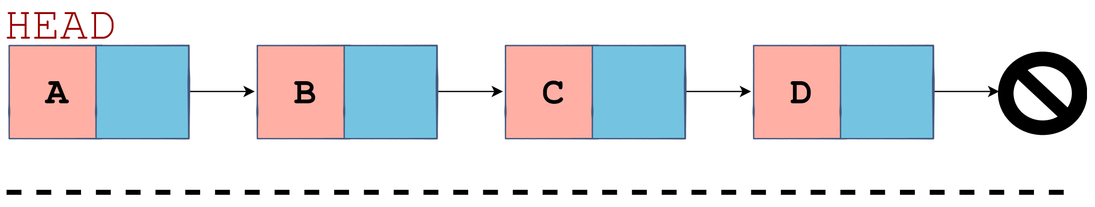
To solve this problem, we need to handle two cases:

1. Node to be deleted is head.
2. Node to be deleted is not head.

Case of Deleting Head

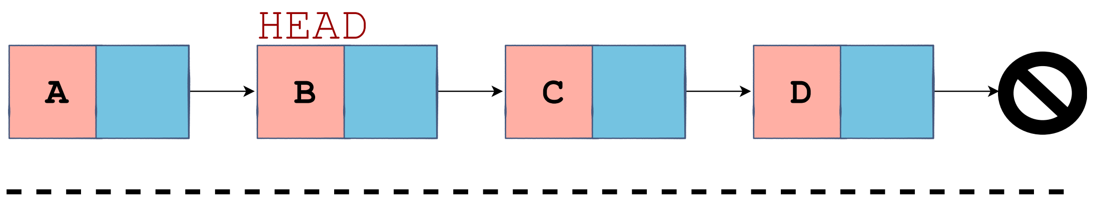
Let's look at the illustration below to get a fair idea of the steps that we are going to follow while writing the code.

Singly Linked List: Delete Head Node



1 of 4

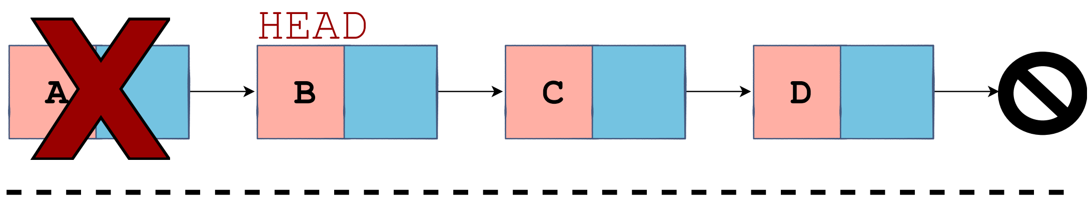
Singly Linked List: Delete Head Node



Update head to the next node of the previous head

2 of 4

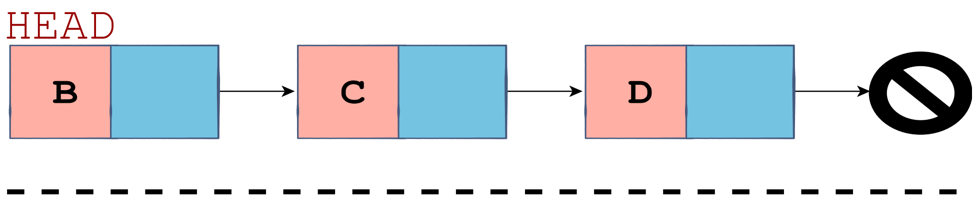
Singly Linked List: Delete Head Node



Delete the previous head node

3 of 4

Singly Linked List: Delete Node



4 of 4

Now let’s go ahead and implement the case illustrated above in Python.

```
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
    cur_node = None
    return
```

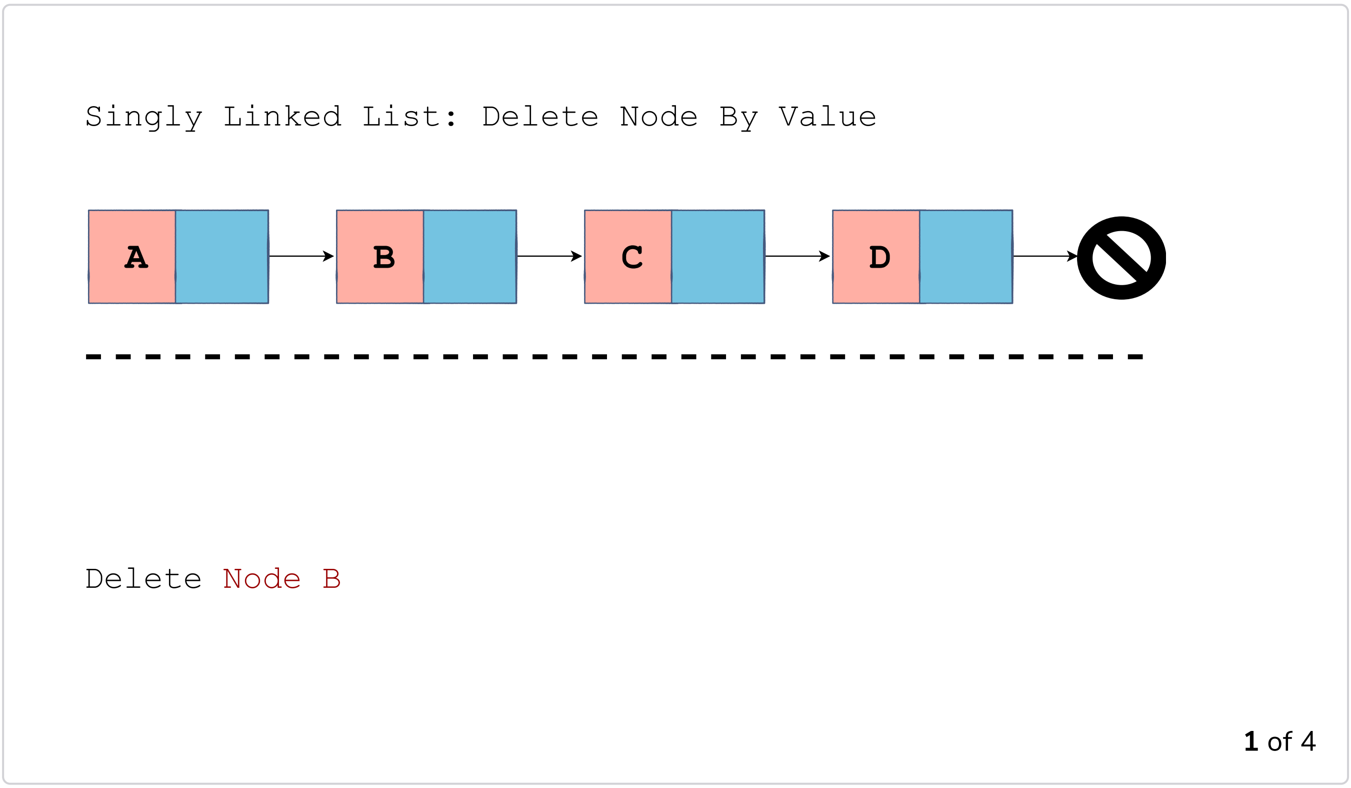
delete_node(self, key)

The class method `delete_node` takes `key` as an input parameter. On **line 3**, we'll declare `cur_node` as `self.head` to have a starting point to traverse the linked list. To handle the case of deleting the head, we'll check if `cur_node` is not `None` and if the data in `cur_node` is equal to `key` on **line 5**. Note that `cur_node` is pointing to the head of the linked list at this point. If `key` matches `cur_node.data`, we'll update the head of the linked list (`self.head`) to `cur_node.next`, i.e., the next node of the previous head node (**line 6**). Once we have updated the head of the linked list, we'll set the node to be deleted (`cur_node`) to `None` (**line 7**) and return from the method.

Now that we have written the code for deleting the head node let's move on to the other case of deletion, deleting a node from a linked list which is not the head node.

Case of Deleting Node Other Than the Head

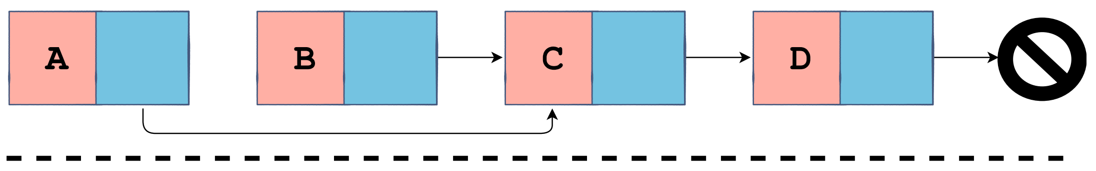
In this case, we'll traverse the linked list to find the node that we are supposed to delete. As we move along, we also need to keep track of the previous node of the node to be removed. Below is a slide for you to visualize the process.


Singly Linked List: Delete Node By Value

Delete Node B

1 of 4

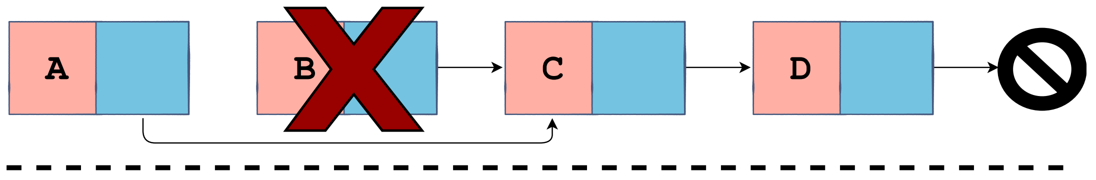
Singly Linked List: Delete Node By Value



Previous node of Node B will point to the next node of Node B

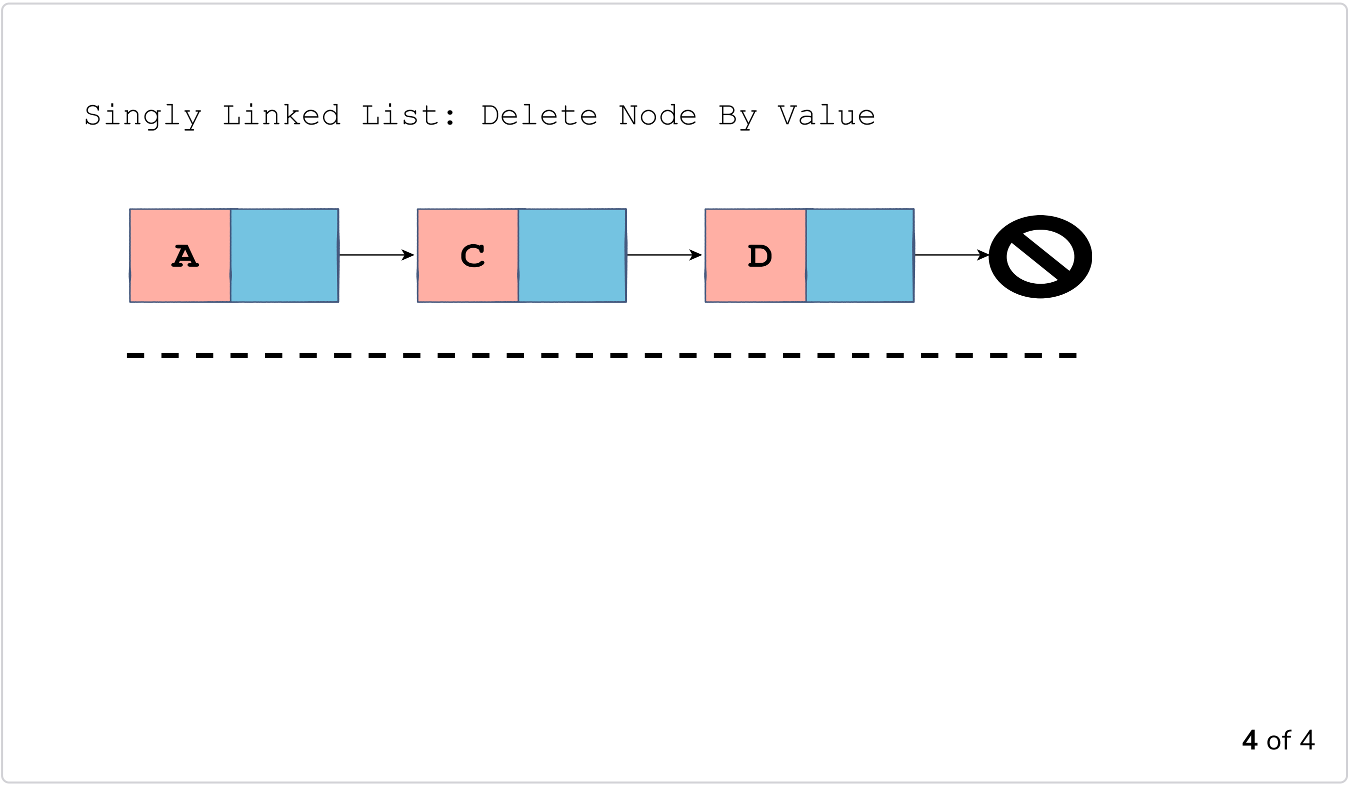
2 of 4

Singly Linked List: Delete Node By Value



Delete Node B

3 of 4



Now, let's turn to the code part. Check it out below:

```
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
```

delete_node(self, key)

We have already discussed the code present in **lines 2-8**. Now we'll concern ourselves with the code starting from **line 10** where `prev` is declared and set to `None`. This will keep track of the previous node of the node to be deleted.

On the next line (**line 11**), we have a `while` loop which will run until `cur_node` becomes `None`. This implies that `key` doesn't exist in our linked list. Another occurrence that would stop the loop would be the `cur_node.data` equaling `key` which refers to the case where we have found the node to be deleted. In the `while` loop, we set `prev` to `cur_node` on **line 12** to keep track of the previous node while `cur_node` is updated to the next node in the next line.

When the `while` loop terminates, we check using an if-condition on **line 15** whether or not it's because of `cur_node` being `None`, which will imply that `key` did not match any of the data of the current node. If it's true, then we return from the method as there is no node to delete. However, if `cur_node` is not `None` and its data matches with the `key`, we proceed to **line 18**. As we kept track of the previous node of the node to be deleted (`prev`), we now want to point it to the next node of the node to be deleted instead of pointing it to the node that we want to delete (`cur_node`). Therefore, we set `prev.next` to `cur_node.next`. In this way, we have removed the link of the `cur.node` that was previously in between. Finally, we set the node to be deleted (`cur_node`) equal to `None` on **line 19**.

Deleting the node based on a value from a linked list was as simple as that! The `delete_node` has been made a class method in the code widget below. Go ahead and play around with it by making more test cases!

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


class LinkedList:
    def __init__(self):
        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:
```

```
        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


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

llist.delete_node("B")
llist.delete_node("E")

llist.print_list()
```

delete_node(self, key)

In the next lesson, we'll consider another case of deleting a node from a linked list, i.e., we will delete a node based on position rather than value. Let's find list, i.e., we will delete a node based on position rather than value. Let's find out more in the next lesson!