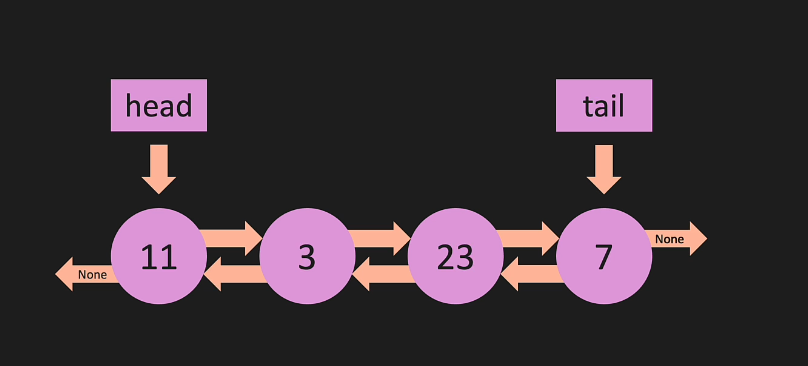
**Doubly Linked List**



**Constructor:**

1. class Node:
2. def \_\_init\_\_(self, value):
3. self.value = value
4. self.next = None
5. self.prev = None

8. class DoublyLinkedList:
9. def \_\_init\_\_(self, value):
10. new\_node = Node(value)
11. self.head = new\_node
12. self.tail = new\_node
13. self.length = 1

This code defines a **Node** class and a **DoublyLinkedList** class.

1. The **Node** class is defined with an **\_\_init\_\_** method that takes a value as its input. It sets the **value** property of the **Node** instance to the given value and initializes the **next** and **prev** properties to **None**. These **next** and **prev** properties will be used to reference the next and previous nodes in the doubly linked list.
2. The **DoublyLinkedList** class is defined with an **\_\_init\_\_** method that takes a value as its input. It creates a new **Node** instance called **new\_node** with the given value. The **head**, **tail**, and **length** properties of the **DoublyLinkedList** instance are initialized as follows:
   * The **head** property is set to **new\_node**, indicating the beginning of the list.
   * The **tail** property is set to **new\_node**, indicating the end of the list.
   * The **length** property is set to 1, as there is only one node in the list at this point.

**Append to DLL**

1. def append(self, value):
2. new\_node = Node(value)
3. if self.head is None:
4. self.head = new\_node
5. self.tail = new\_node
6. else:
7. self.tail.next = new\_node
8. new\_node.prev = self.tail
9. self.tail = new\_node
10. self.length += 1
11. return True

The **append** method is to add a new node with the given value to the end of the doubly linked list.

1. Create a new **Node** instance called **new\_node** with the given value.
2. Check if the head of the doubly linked list is **None**, which means the list is empty. a. If the list is empty, set both the **head** and **tail** of the list to **new\_node**, because it's now the only node in the list.
3. If the list is not empty, perform the following steps: a. Set the **next** property of the current **tail** (last node) to **new\_node**. This connects the new node to the end of the list. b. Set the **prev** property of **new\_node** to the current **tail**. This connects the new node to the previous node in the list. c. Update the **tail** property of the list to point to **new\_node**, as it's now the new last node in the list.
4. Increment the **length** property of the list by 1, as we've added a new node to the list.
5. Return **True** to indicate that the operation was successful.

**Pop from DLL**

1. def pop(self):
2. if self.length == 0:
3. return None
4. temp = self.tail
5. if self.length == 1:
6. self.head = None
7. self.tail = None
8. else:
9. self.tail = self.tail.prev
10. self.tail.next = None
11. temp.prev = None
12. self.length -= 1
13. return temp
14. **def pop(self):**: This line defines a method called **pop** which will be a part of a class. The **self** parameter refers to the instance of the class itself.
15. **if self.length == 0:**: This is a check to see if the linked list is empty. The attribute **length** keeps track of the number of nodes in the list.
16. **return None**: If the list is empty, the method returns **None**.
17. **temp = self.tail**: Here, we save the current tail node of the linked list in a temporary variable called **temp**.
18. **if self.length == 1:**: This checks if the list has only one node.
19. **self.head = None; self.tail = None**: If there's only one node, both the head and tail pointers are set to **None**, effectively emptying the list.
20. **else:**: This is the case where the list has more than one node.
21. **self.tail = self.tail.prev**: The tail pointer is updated to point to the node just before the current tail.
22. **self.tail.next = None; temp.prev = None**: The next pointer of the new tail is set to **None**. Also, the previous pointer of the node that we are going to return is set to **None**.
23. **self.length -= 1**: Decrease the length of the list by 1, as we are removing a node.
24. **return temp**: Finally, the method returns the node that was removed from the list.