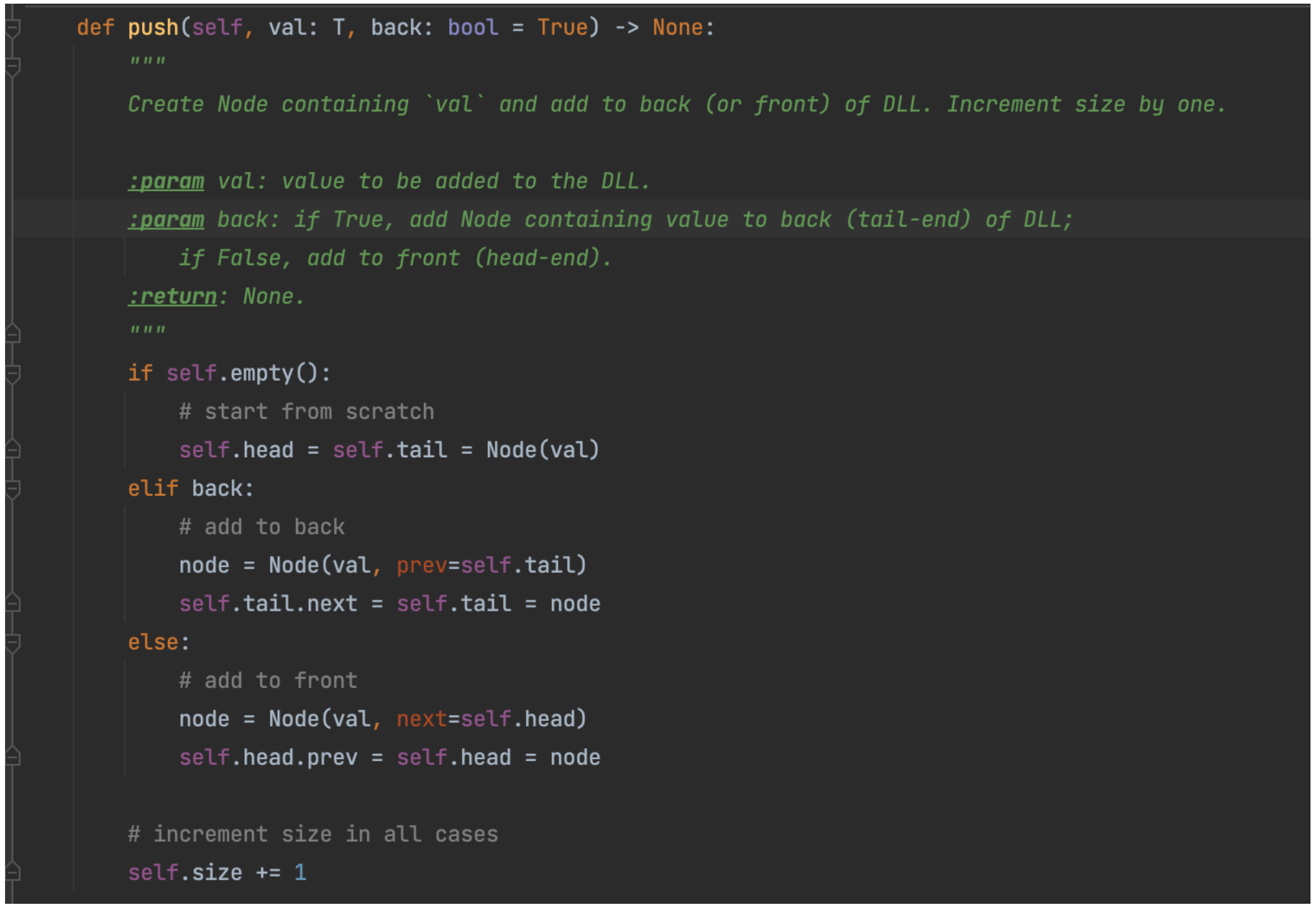
Project 01 DLL

We will share few of the essential functions required by students to complete in their project.

push function

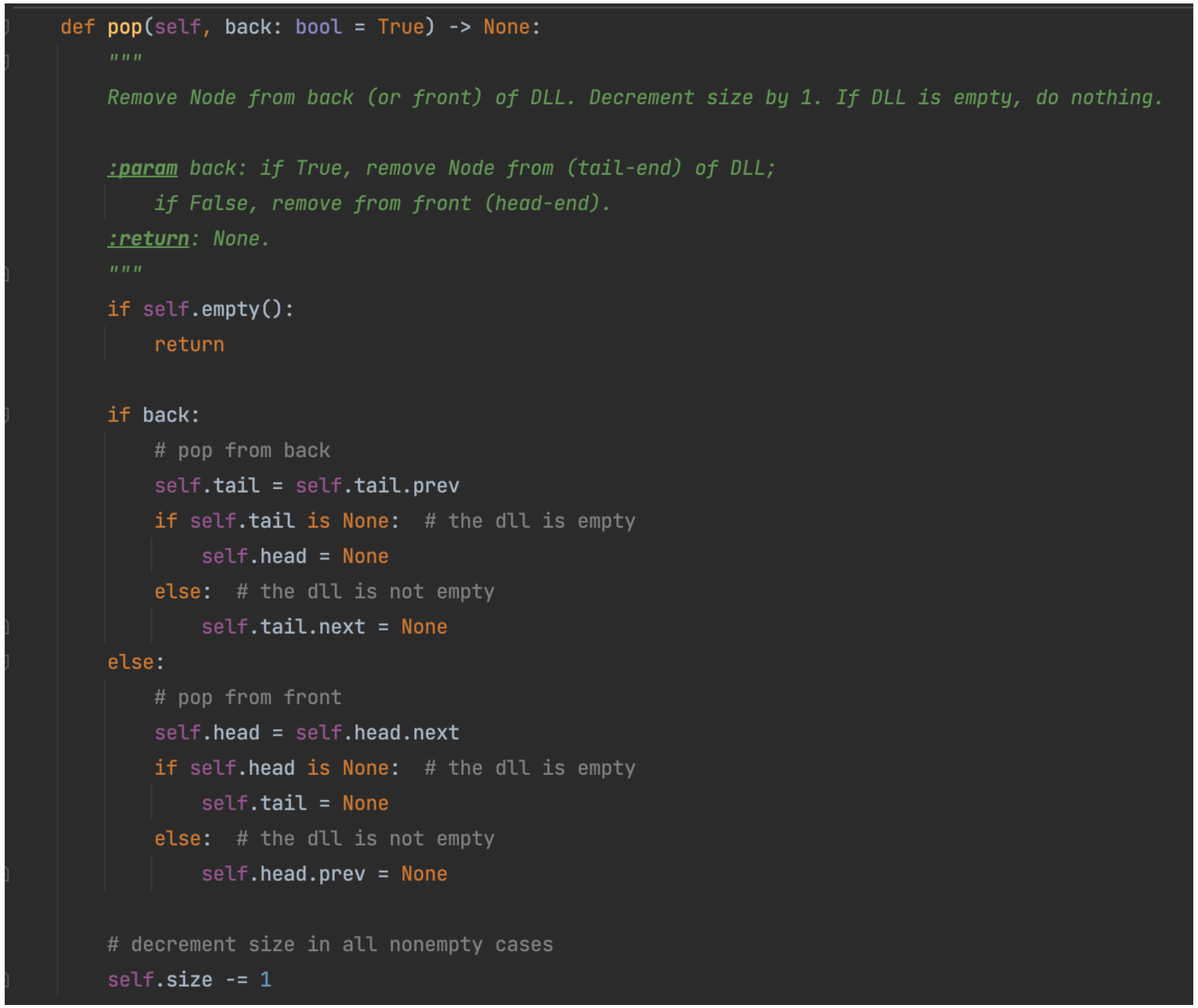


```python
def push(self, val: T, back: bool = True) -> None:
    """
    Create Node containing `val` and add to back (or front) of DLL. Increment size by one.

    :param val: value to be added to the DLL.
    :param back: if True, add Node containing value to back (tail-end) of DLL;
        if False, add to front (head-end).
    :return: None.
    """
    if self.empty():
        # start from scratch
        self.head = self.tail = Node(val)
    elif back:
        # add to back
        node = Node(val, prev=self.tail)
        self.tail.next = self.tail = node
    else:
        # add to front
        node = Node(val, next=self.head)
        self.head.prev = self.head = node

    # increment size in all cases
    self.size += 1
```

pop function



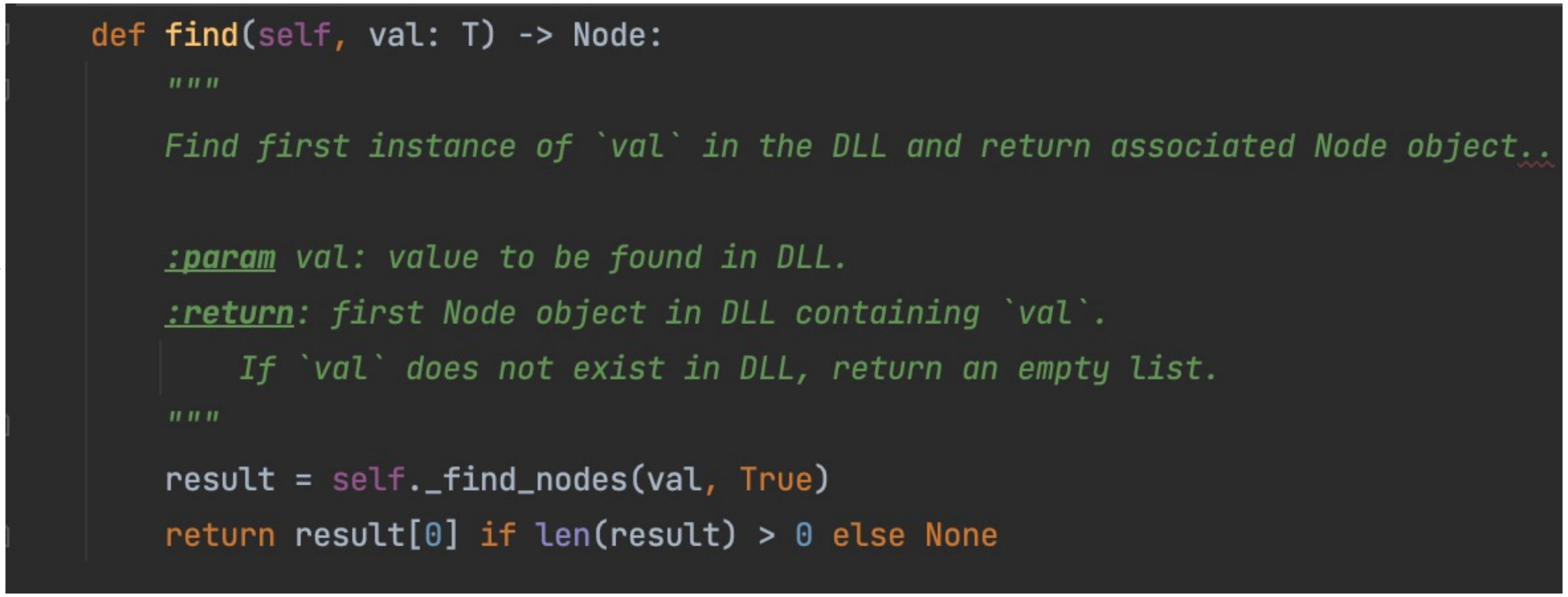
```python
def pop(self, back: bool = True) -> None:
    """
    Remove Node from back (or front) of DLL. Decrement size by 1. If DLL is empty, do nothing.

    :param back: if True, remove Node from (tail-end) of DLL;
        if False, remove from front (head-end).
    :return: None.
    """
    if self.empty():
        return

    if back:
        # pop from back
        self.tail = self.tail.prev
        if self.tail is None:  # the dll is empty
            self.head = None
        else:  # the dll is not empty
            self.tail.next = None
    else:
        # pop from front
        self.head = self.head.next
        if self.head is None:  # the dll is empty
            self.tail = None
        else:  # the dll is not empty
            self.head.prev = None

    # decrement size in all nonempty cases
    self.size -= 1
```

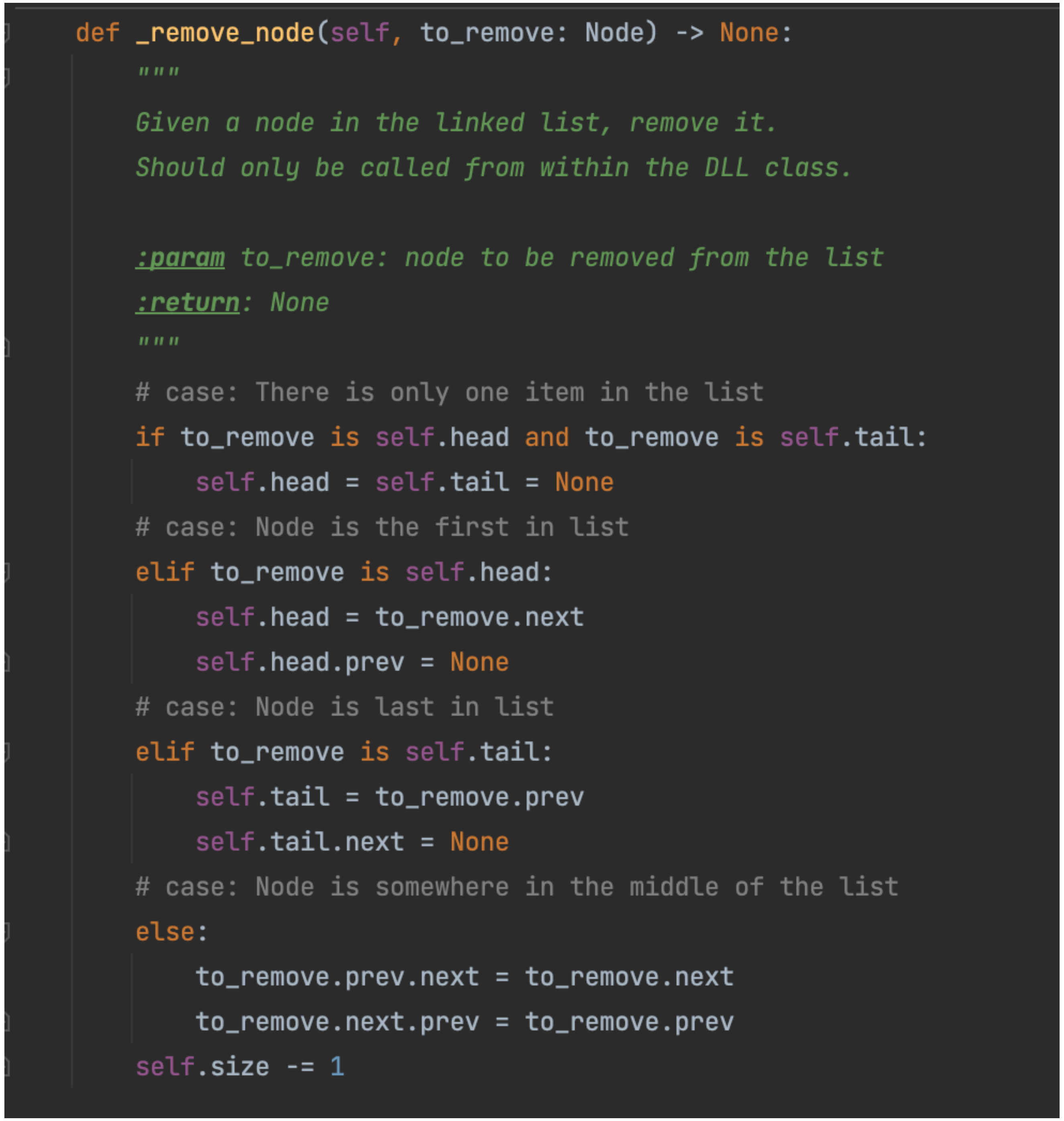
find function



```
def find(self, val: T) -> Node:
    """
    Find first instance of `val` in the DLL and return associated Node object..

    :param val: value to be found in DLL.
    :return: first Node object in DLL containing `val`.
        If `val` does not exist in DLL, return an empty list.
    """
    result = self._find_nodes(val, True)
    return result[0] if len(result) > 0 else None
```

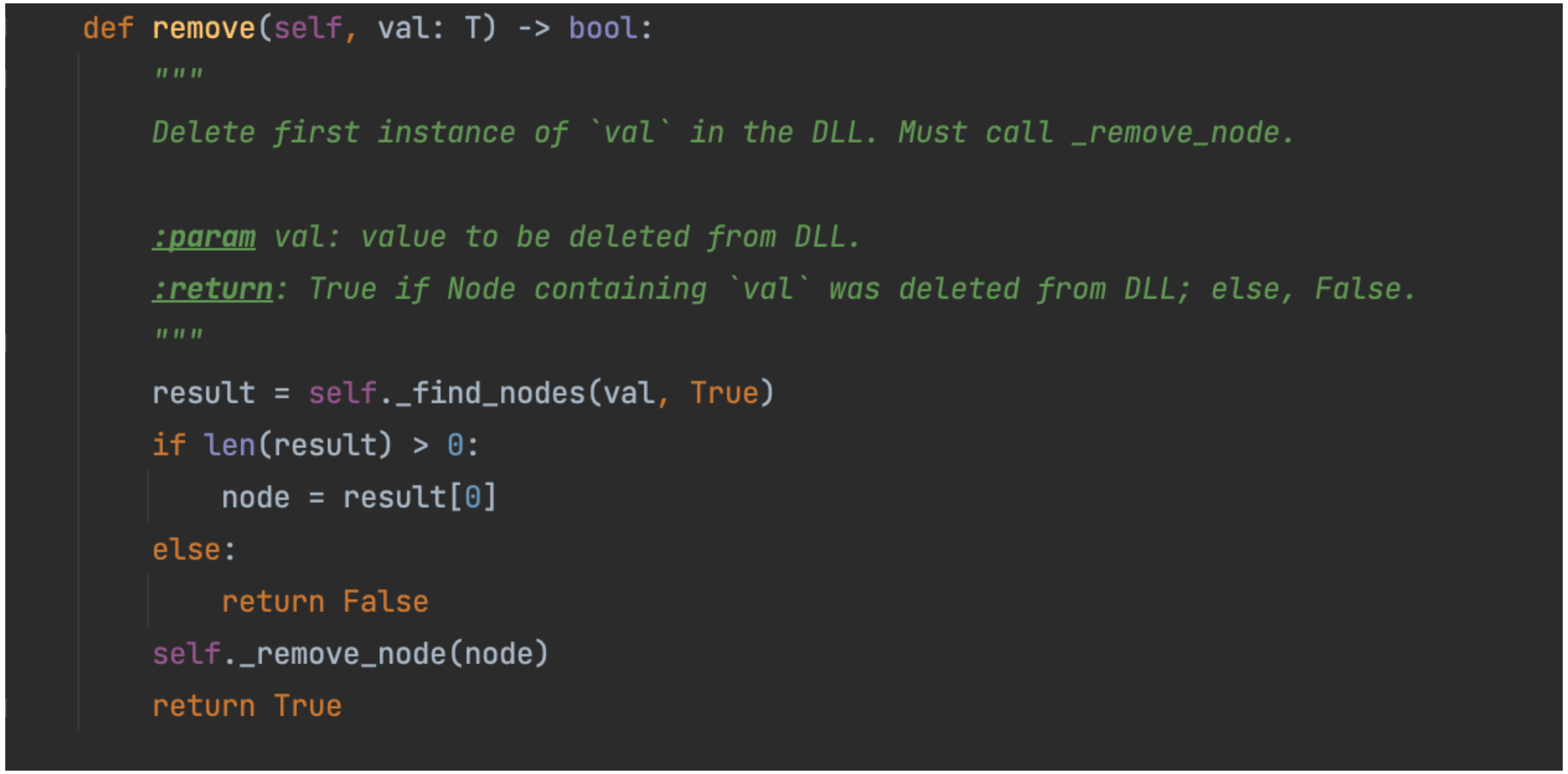
remove_node function



```
def _remove_node(self, to_remove: Node) -> None:
    """
    Given a node in the linked list, remove it.
    Should only be called from within the DLL class.

    :param to_remove: node to be removed from the list
    :return: None
    """
    # case: There is only one item in the list
    if to_remove is self.head and to_remove is self.tail:
        self.head = self.tail = None
    # case: Node is the first in list
    elif to_remove is self.head:
        self.head = to_remove.next
        self.head.prev = None
    # case: Node is last in list
    elif to_remove is self.tail:
        self.tail = to_remove.prev
        self.tail.next = None
    # case: Node is somewhere in the middle of the list
    else:
        to_remove.prev.next = to_remove.next
        to_remove.next.prev = to_remove.prev
    self.size -= 1
```

remove function



```python
def remove(self, val: T) -> bool:
    """
    Delete first instance of `val` in the DLL. Must call _remove_node.

    :param val: value to be deleted from DLL.
    :return: True if Node containing `val` was deleted from DLL; else, False.
    """
    result = self._find_nodes(val, True)
    if len(result) > 0:
        node = result[0]
    else:
        return False
    self._remove_node(node)
    return True
```