Doubly-linked list implementations

Implement a node class for a doubly-linked list. It should be able to point to both the next node in the list, and to the previous one.

Double-ended queue using a doubly-linked list

Implement the class ***DLL_Deque***.

- Implement the constructor so that it initializes the sentinel nodes (or one node) that serve as the head and tail in the list, without being actual data nodes themselves. In effect nodes number ***-1*** and ***size*** in the list.
- Implement operations to add items, get values and remove items from both ends
    - *That´s 6 functions in all, add, remove and get value from both front and back*
- Implement **__str__** that returns a string with all the items, from front to back.
- Next implement a function to print the list from back to front.
    - Don't use recursion here
        - Just walk the list from back to front, using the prev pointers.
            - This helps test that all the connections are correct
- Implement a get_size function, and use it in your tests to make sure the data is correct.

Positional list using a doubly-linked list

Implement the class ***DLL_PosList***.

- Implement the constructor in the same way as in DLL_Deque.
- Set it up so that it keeps track of a current location, by pointing to specific ***current_node*** in the list. It can also have an integer variable for the ***current_position*** if needed.
    - At the start this should be 0, and the current node should point to tail.
- Implement an ***insert*** function that only takes a value, and inserts it in a new node at the current location.
    - The new node is added in front of the current node
    - If several items are inserted in a row, each one comes in front of the last one
        - lis.insert('A')
          lis.insert('B')
          lis.insert('C')
          print(lis)
            - Output: C B A
        - Update you current node and current position variables accordingly
        - *If we imagine the locations are numbered, the current location stays at the same number location*
    - Draw this operation on a piece of paper and consult with a peer. *Don't just point to a node on your image but specifically state which variable references the node, e.g. current_node, an extra temporary variable, header, trailer, etc.* **Be exact in your preparation and your coding will come much easier.**
- Implement ***__str__***, ***get_size*** and a backward print operation to check the links.

more on next page...

- Implement two more functions:
    - ***move_to_next***
        - Moves the current location one closer to the back of the list
        - Check that it's not at the back of the list already
    - ***move_to_prev***
        - Moves the current location one closer to the front of the list
        - Check that it's not at the front of the list already
    - lis.insert('A')
      lis.insert('B')
      lis.move_to_next()
      lis.insert('C')
      lis.move_to_prev()
      lis.insert('D')
      print(lis)
        - Output: D B C A
- Implement a ***get_value*** function, that returns the value of the item at the current location.
- Implement a ***remove*** function that removes the item at the current location.
    - If several items are removed in a row, they will be removed towards the back of the list
        - print(lis)
          lis.remove()
          print(lis)
          lis.remove()
          print(lis)
            - Output:
                - C B A
                  B A
                  A
        - Update you current node and current position variables accordingly
        - *If we imagine the locations are numbered, the current location stays at the same number location*
    - Draw this operation on a piece of paper and consult with a peer. *Don't just point to a node on your image but specifically state which variable references the node, e.g. current_node, an extra temporary variable, header, trailer, etc.* **Be exact in your preparation and your coding will come much easier.**