Doubly Linked List.

```
class DoublyLinkedList:
    """Implementation of a doubly linked list."""
    def __init__(self):
        # Initially, the list is empty: Pointers to
        # left and right ends are None.
        self.end = {'left' : None, 'right' : None}
        self.length = 0

    def apply(self, the_func, *additional_arguments):
        """Traverse the list from left to right and
        apply function the_func to the data stored;
        return list of results."""
        # Beachte die Möglichkeit, der Funktion the_func
        # eine _variable_ Zahl zusätzlicher Argumente
        # zu übergeben (0 ist auch möglich!)
        results = []
        pointer = self.end['left']
        while pointer:
            results+= [the_func(pointer, *additional_arguments)]
            pointer = pointer.neighbour['right']
        return results

    # Beispiel, wie "apply" verwendet werden kann:
    def list_of_nodes(self):
        """Return the doubly linked list as a "normal" python
        list of nodes"""
        return self.apply(lambda x : x)

    # Noch ein Beispiel:
    def __str__(self):
        """String representation of the list"""
        data_list = self.apply(lambda x : f'{x.contents}')
        return f"DLL of length ({self.length}): (" + ", ".join(data_list)+")"

    # Anfügen bzw. Entfernen von Listen heißt "traditionell"
    # "push" bzw. "pop": Bei einer doppelt verketteten Liste
    # können wir links oder rechts anfügen bzw. entfernen.
    def push(self, data, where='right'):
        """Push (i.e., append) new data at list's end "where"."""
        if self.end[where]:
            new_end = self.end[where].squeeze_in(data, where)
        else:
            new_end = DLLNode(data)
            self.end[mirror[where]] = new_end

        self.length+= 1
        self.end[where] = new_end

def pop(self, where='right'):
    """Remove node and return its data at list's end "where"."""
    if self.end[where]:
        # Funktion "detach" gibt den entfernten Knoten zurück:
        return self.end[where].detach().contents
        self.length-= 1
    # Implicit else:
    print('list is empty!!!')
    return None
```