Assignment: Doubly-Linked Lists

Motivation: This assignment gives you practice building a linked data structure and carefully manipulating references.

In this assignment you will implement doubly-linked lists. These are similar to linked lists, but each node knows about the *previous* node in addition to the next one. This allows for various operations to be implemented more efficiently, including concatenating two lists in constant time.

The Task

Define a class `DoublyLinkedList` that represents a double-ended queue. It should implement the following methods:

```
__init__
__str__
add_front
add_back
remove_front
remove_back
concatenate
```

The unit tests below give examples of using these methods.

`__str__` is the only method that returns a value; the others simply modify the `DoublyLinked`dList on which they are called. concatenate also modifies its argument, so that after `a.concatenate(b) a` contains all of the items that were previously in *both* `DoublyLinkedLists` and the state of `b` is no longer meaningful.

`__str__` is also the only method that involves any kind of loop or recursion. The other methods should all run in constant time.

Unit Tests

test_doubly_linked.py

Hints

You will also need to define a `Node` class with attributes `item`, `prev`, and `next`.

While not strictly necessary, defining `__repr__` as well as `__str__` can make your debugging easier.

`DoublyLinkedList` should have a single attribute holding a "header node". The item of this node is irrelevant. Its `next` attribute refers to the first node in the list, while `prev` refers to the last node. If you think of the `DoublyLinkedList` as a necklace of beads, this is the clasp that holds the ends together.

No node should have `None` as its `prev` or `next` attribute. The `next` attribute of the last node, and the `prev` attribute of the first node, should be the header node.

Draw pictures! It's very easy to get your references tangled. If you draw a diagram of the data structure after each step in a method, it's much easier to keep things straight. Draw a new picture for each step rather than editing the previous picture.

Optional Challenges

If you get everything working, *have saved a copy just in case*, and want an additional challenge, try some of the following:

- Write a method `str_reverse` that returns a representation of the `DoublyLinkedList` with the elements reverse order. This method should not modify the list.
- Write a method `reverse` that modifies the `DoublyLinkedList` to put the items in reverse order. This method should not create any new Nodes; just modify the existing ones.
- Write a method `remove` that takes an argument and removes all copies of that item from the `DoublyLinkedList`. Finding the items will take linear time, but actually removing each one should only take constant time.

What to Hand in

Hand in a single file `doubly_linked.py`. It should contain your definitions of `Node` and `DoublyLinkedList`.