Addition using doubly linked list

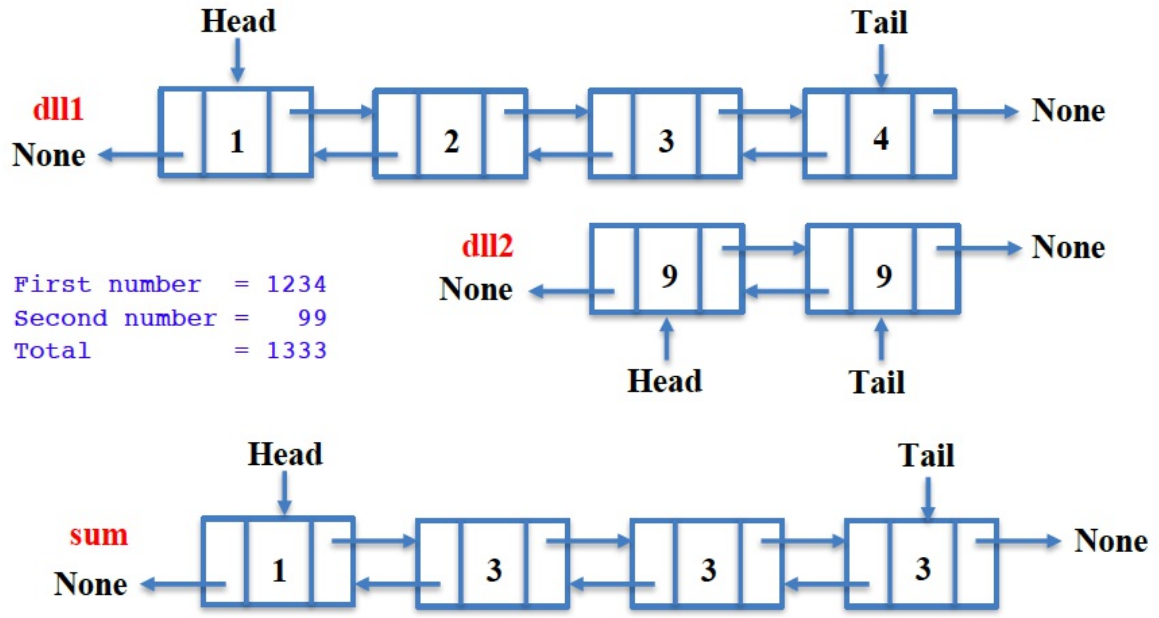
Part 1: Introduction

In this lab, we use Doubly Linked Lists to store positive integers and perform addition operations.

The purpose of this assignment is to:

1. Enhance your understanding of doubly linked lists
2. Practice how to perform operations on doubly linked lists

The goal of this assignment is to input two positive integers, store each number in a doubly linked list (where each digit is data in a node), and perform addition on them by implementing various operations of doubly linked list. For example, the first two doubly linked lists below represent `1234`, and `99` respectively. The addition of these two numbers: `1333`, is represented as the third doubly linked list.



Part 1 (In-lab): Initial work

Initializing the doubly linked list.

Define the `DoublyLinkedList` class, with methods `__init__`, `addLast`, `addFirst` and `__str__`:

• `__init__(self, string = '')`: As the linked list is initialized, each digit in the input string is added as data in a node. A `for` loop is used to iterate over the entire string. For each character in the string , the corresponding new node is added at the end of the list (calling `addLast`).

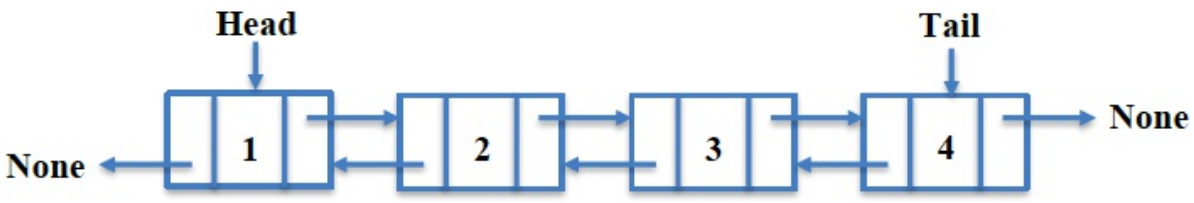
Also initialize the `head`, `tail` and `_length` attributes.

Note: `string` is an OPTIONAL argument, whose default value is `''` (empty string).

• `__str__(self)`: This method returns the string representation of the object. Each node is converted to string by traversing through the entire doubly linked list, and they are concatenated together.

Example:

```
x = DoublyLinkedList('1234')    # addLast will be called 4 times
print(x)                        # prints '1234'
```

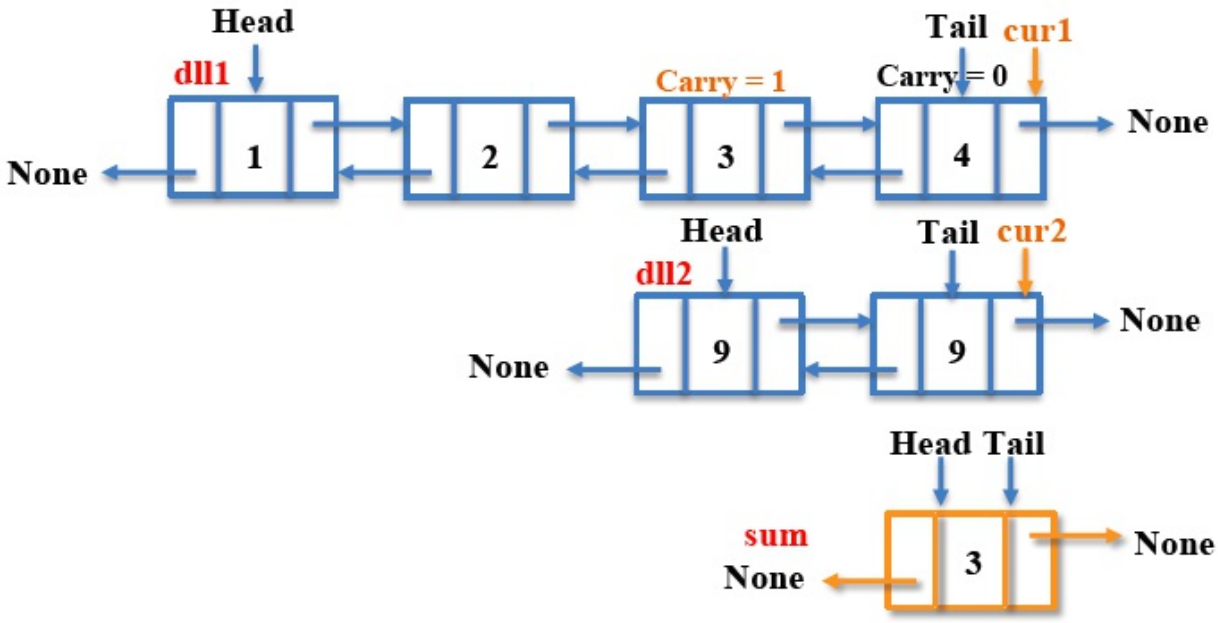


Part 2: Adding numbers

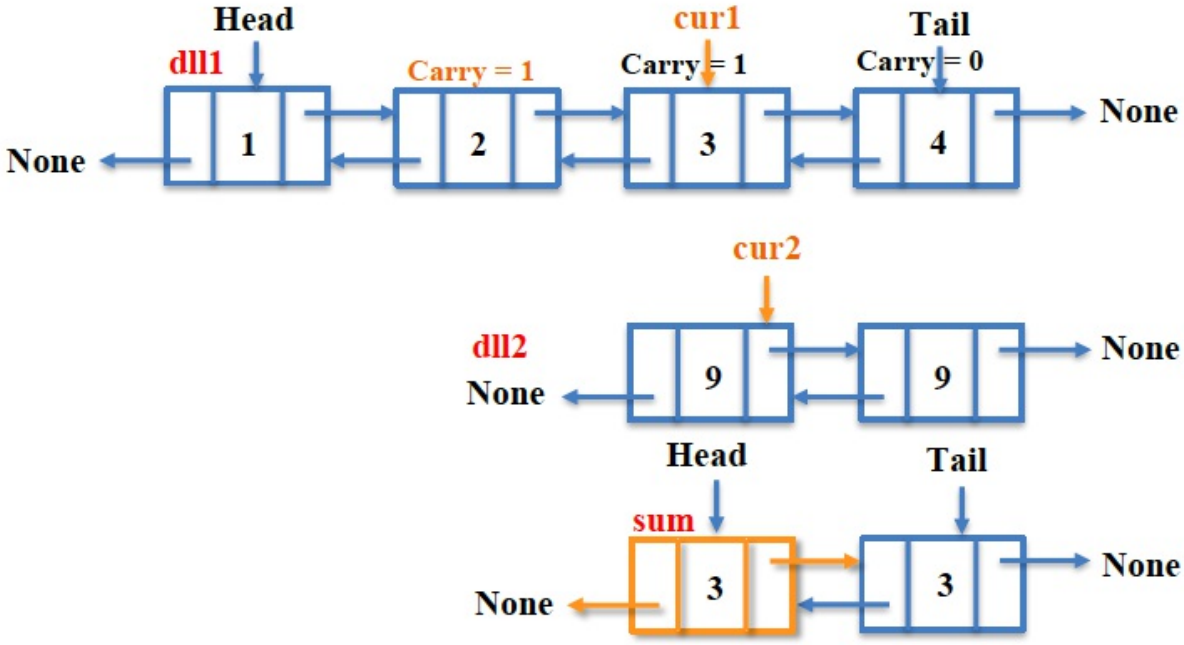
You now need to write `sumlinkednumbers(dll1, dll2)`. This function performs the addition of the two doubly linked lists and stores the sum in a third doubly linked list. This function returns the sum.

```
Example:
dll1 = DoublyLinkedList('1234')
dll2 = DoublyLinkedList('99')
dll3 = sumlinkednumbers(dll1, dll2)
print(dll3)                          # prints 1333
```

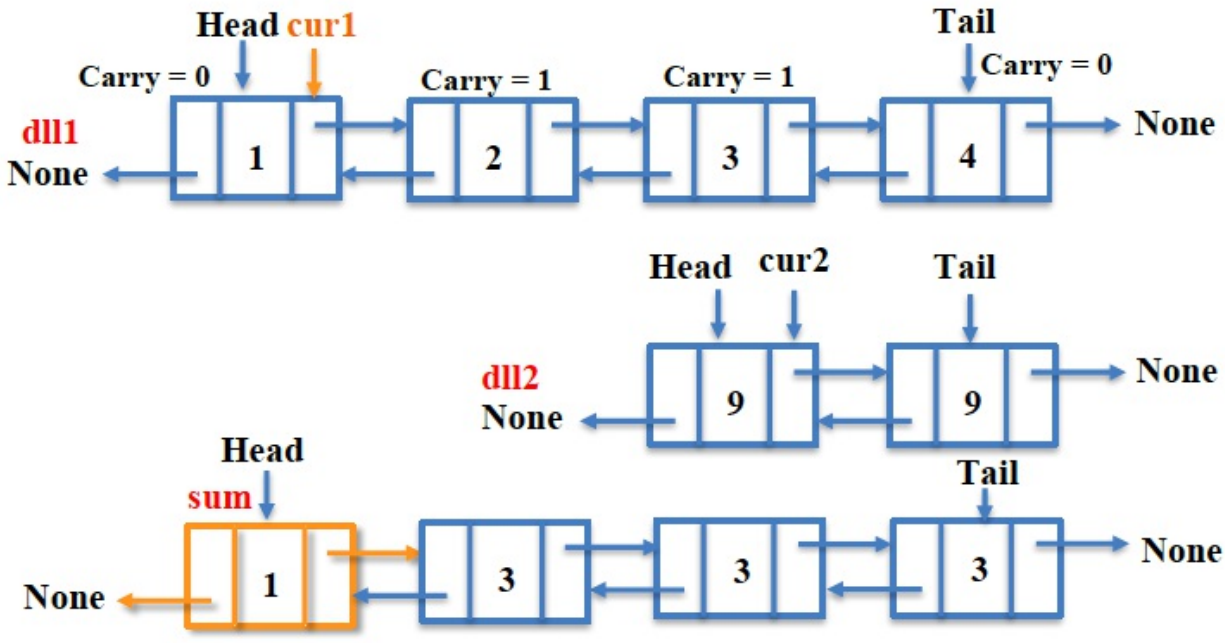
1. The first step in addition is to start from the unit's place. Therefore, in our case, we must start from the `tail` of the two doubly linked lists. To do this we use `cur1` and `cur2` which point to the `tail` of the first and second doubly linked list respectively. In each iteration, they are updated to point to the previous `node`. We must also ensure that we account for the `carry` that will be generated during this process.



2. The next step is to perform the actual addition. Create a third empty `DoublyLinkedList` object that stores the sum. While `cur1` and `cur2` still point to a node, we add the digits in those nodes along with the `carry`. The result is stored in the third doubly linked list (use `addFirst`). We must next calculate the new value of carry. Finally, the values of `cur1` and `cur2` are updated to the previous `node`.



3. There are two more cases that we have to account for. Either the first number could have more digits than the second (in which case, `cur2` would have reached the end of the list while `cur1` would still point to a valid `node`) or vice versa. In this case, we simply add the `carry` and valid `node` to generate the sum. We then calculate the new value of `carry` and update `cur1` (or `cur2`) to point to the previous `node`.



Note: The final sum shouldn't have any leading zeros.

```
dll1 = DoublyLinkedList('0023')
dll2 = DoublyLinkedList('99')
dll3 = sumlinkednumbers(dll1, dll2)
print(dll3)                # prints '122' and NOT '0122'
```

Part 3: List Reversal

Now add a method `reverse` to reverse the lists.

Example:

```
dll1 = DoublyLinkedList('1234')
dll1.reverse()
print(dll1)                              # prints '4321'
dll2 = DoublyLinkedList('5')
dll3 = sumlinkednumbers(dll1, dll2)
print(dll3)                              # prints 4326
```

Reversal will be done with a loop that traverses the list and swaps the `prev` and `next` links for each node.

Note: Also remember to update the `head` and `tail` attributes of the list!

Part 4: Fast List Reversal

Now add an `isReversed` flag (Boolean attribute with default value `False`) to your doubly linked list, and write `fastReverse()` which works in `O(1)` time. It toggles the `isReversed` flag, and swaps the `head` and `tail` attributes of the list. It does NOT traverse the list – then it would not be `O(1)`!

Now that you have this flag, you have to update the way you access the `prev` and `next` link of every `node` throughout the other methods. If the flag is `TRUE`, you have to get the opposite one! This is true whether you are just *getting* it, or if you are *setting* it.

If done correctly, the two ways of reversing will be indistinguishable to the users of your doubly linked lists, unless they start timing them over VERY large lists!

```
dll1 = DoublyLinkedList('1234')
dll1.fastReverse()
```

```
print(dll1)                    # prints ‘4321’
dll1.reverse()
print(dll1)                    # prints ‘1234’
```