JavaScript Add Two Numbers

Challenge

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number `0` itself.

1st Example

```
Input: l1 = [2,4,3], l2 = [5,6,4]
Output: [7,0,8]
Explanation: 342 + 465 = 807.
```

2nd Example

```
Input: l1 = [0], l2 = [0]
Output: [0]
```

3rd Example

```
Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]
Output: [8,9,9,9,0,0,0,1]
```

Constraints

- `0 <= Node.val <= 9`
- The number of nodes in each linked list is in the range `[1, 100]`.
- It is guaranteed that the list represents a number that does not have leading zeros.

Solution

```
const addTwoNumbers = (l1, l2) => {
    let list  = new ListNode(0),
        head  = list,
        sum   = 0,
        carry = 0;

    while (l1 !== null || l2 !== null || sum > 0) {
        if (l1 !== null) {
            sum = sum + l1.val;
            l1 = l1.next;
        }

        if (l2 !== null) {
            sum = sum + l2.val;
            l2 = l2.next;
        }

        if (sum >= 10) {
            carry = 1;
            sum = sum - 10;
        }
```

Solution continues on next page...

```
        head.next = new ListNode(sum);
        head = head.next;
        sum = carry;
        carry = 0;
    }

    return list.next;
};
```

Explanation

I've written a function called `addTwoNumbers` that takes in two linked lists as parameters (`l1` and `l2`). The purpose of this function is to add the numbers represented by the linked lists and return a new linked list representing the sum.

Inside the function, several variables are initialized: `list`, `head`, `sum`, and `carry`. `list` and `head` are set to a new ListNode object with a value of `0`, which will be the head of the resulting linked list. `sum` and `carry` are set to 0 to keep track of the current sum and any carry values.

The function enters a while loop that continues until both `l1` and `l2` are null and the `sum` is greater than `0`. This ensures that all digits of both linked lists are processed, as well as any remaining carry values.

Inside the loop, it checks if `l1` is not null. If it is not null, the value of `l1` is added to the `sum`, and `l1` is moved to the next node.

Similarly, it checks if `l2` is not null. If it is not null, the value of `l2`

is added to the `sum`, and `l2` is moved to the next node.

After adding the values, it checks if the `sum` is greater than or equal to `10`. If it is, the `carry` is set to `1`, and `10` is subtracted from the `sum` to keep the value within a single digit.

It then creates a new ListNode with the value of `sum` and assigns it to `head.next`, effectively adding a new node to the resulting linked list.

The `head` is moved to the next node to prepare for the next iteration.

The `sum` is updated with the value of `carry`, and the `carry` is reset to `0` for the next calculation.

The loop continues until the while loop condition is no longer true, ensuring all digits and carry values are processed.

Finally, the function returns the next node of `list`, which is the head of the resulting linked list.

In summary, this function performs digit-by-digit addition of two linked lists, taking into account carry values, and returns a new linked list representing the sum.

Author: Trevor Morin