LEARN LINKED LISTS: JAVASCRIPT

Constructor and Adding to Head

Let's implement a linked list in JavaScript. As you might recall, a linked list is a sequential chain of nodes. Remember that a node contains two elements:

- data
- a link to the next node

We are going to use a provided `Node` class, which you can find in **Node.js**. Make sure to use the proper `Node` methods throughout the lesson instead of accessing properties directly (ex. use `someNode.getNextNode()` instead of `someNode.next`).

Depending on the end-use of the linked list, there are a variety of methods that we can define. For our use, we want to be able to:

- add a new node to the beginning (head) of the list
- add a new node to the end (tail) of the list
- remove a node from the beginning (head) of the list
- print out the nodes in the list in order from head to tail

To start, we are going to create the `LinkedList`'s constructor and `.addToHead()` method.

Ready? Let's go!

Instructions

1.

The only property we need our linked list to have is a head, which we will add in our constructor. Inside the `LinkedList` class, define the constructor. It should take no parameters, and should set the list's head to `null`.

Hint

Remember to use `this` when referencing the head: `this.head = null;`.

2.

Define an `.addToHead()` method that takes one parameter called `data`. Inside the method, create a `Node` `const` variable named `newHead`, and pass `data` as an argument.

Hint

Remember, you define a class instance using the `new` keyword:

```
const someNode = new Node(parameter);
```

3.

Still inside your `.addToHead()` method, create a `const` variable named `currentHead`, and set it equal to the list's head. Then change the list's head to equal `newHead`.

Hint

Remember, you can access the list's head using `this.head`.

4.

Finally, still in your `.addToHead()` method, check if there is a current head to the list. If there is, set the list's head's next node to `currentHead`.

Hint

To check if there is a current head to the list, you can use `if (currentHead)`. Look back at **Node.js** for a refresher on how to set the next node.

LinkedList.js

```
const Node = require('./Node');

class LinkedList {
  constructor() {
    this.head = null;
  }

  addToHead(data) {
    const newHead = new Node(data);
    const currentHead = this.head;
    this.head = newHead;
    if(currentHead) {
      this.head.setNextNode(currentHead);
    }
  }
}

module.exports = LinkedList;
```

Node.js

```
const Node = require('./Node');

class LinkedList {
  constructor() {
    this.head = null;
  }

  addToHead(data) {
    const newHead = new Node(data);
    const currentHead = this.head;
    this.head = newHead;
    if(currentHead) {
      this.head.setNextNode(currentHead);
    }
  }
}

module.exports = LinkedList;
```

Adding to Tail

Now that we can add to the head of the linked list, the next step is to be able to add to the tail. This will require a few more steps since we don't have a `tail` property in our linked list data structure.

To do this, we are going to start with a temporary tail variable that will be set equal to the list's head. If there is no head, that means that the list is empty, and we will add the node to the head of the list. Otherwise, we will iterate through the list until we find the last node. Once we've found the current tail, we will add a pointer from that node to our new tail.

Instructions

1.

Define an `.addToTail()` method for the `LinkedList` that has one parameter called `data`. Create a variable named `tail`, and set it equal to the list's head. `tail` is going to change throughout the method, so make it a `let` variable.

Hint

Remember that your `.addToTail()` method takes a parameter, and that you can access the list's head using `this.head`.

2.

Now that `tail` is equal to the head of the list, we want to check if it has any value. If `tail` has no value, then that means the list was empty, and we will be creating the head *and* tail with the `data` passed in.

To do this, check if `tail` has no value. If so, set the list's head equal to a new `Node` that takes `data` as an argument.

Hint

You can check if `tail` has no value with a simple `if` statement:

```
if (!tail) {
  // Set the list's head equal to a new node here
}
```

3.

If `tail` does have a value, that means the list is not empty. In that case, we want to iterate through the list until we find the end, so we can add `tail` to the end of the list.

To do this, create an `else` after your `if` statement. Inside it, make a `while` loop that runs while `tail` has a next node. Inside the loop, set `tail` equal to its next node.

(If you accidentally create an infinite loop and your code won't stop running, you can reload the page to stop it.)

Hint

This should all be wrapped in an `else` block that follows your `if` statement. Look back at **Node.js** for a refresher on how to access the next node.

```
if (!tail) {
  this.head = new Node(data);
} else {
  // Create your while loop here
}
```

4.

Finally, inside the same `else` block, but outside the `while` loop, set `tail`'s next node equal to a new `Node` that takes `data` as an argument.

Hint

Be careful where you place your code:

```
else {
  while (tail.getNextNode() !== null) {
    tail = tail.getNextNode();
  }
  // Set the tail's next node here
}
```

LinkedList.js

```
const Node = require('./Node');

class LinkedList {

  constructor() {
    this.head = null;
  }

  addToHead(data) {
    const newHead = new Node(data);
    const currentHead = this.head;
    this.head = newHead;
    if (currentHead) {
      this.head.setNextNode(currentHead);
    }
  }

  addToTail(data) {
    let tail = this.head;
    if (!tail) {
      this.head = new Node(data);
    } else {
      while(tail.getNextNode()) {
        tail = tail.getNextNode();
      }
      tail.setNextNode(new Node(data));
    }
  }
```

```
}

module.exports = LinkedList;
```

Node.js

```
class Node {
  constructor(data) {
    this.data = data;
    this.next = null;
  }

  setNextNode(node) {
    if (!(node instanceof Node)) {
      throw new Error('Next node must be a member of the Node class');
    }
    this.next = node;
  }

  getNextNode() {
    return this.next;
  }
}

module.exports = Node;
```

Removing the Head

So far we can:

- create a new `LinkedList` using its constructor
- add to the head of the list using `.addToHead()`
- add to the tail of the list using `.addToTail()`

Now we're going to learn how to remove from the head of the list. To do this, we are first going to check to see if the list has a head. If it doesn't, there is nothing to return. If there is a head, we will remove it by setting the list's head equal to the original head's next node, and then return that original head.

Instructions

1.

Define a `.removeHead()` method that takes no parameters. Inside the method, create a `const` variable named `removedHead` and set it equal to the list's head. We will use this to keep track of the original head of the list.

Hint

Remember that you can access the list's head using `this.head`.

2.

If `removedHead` has no value, `return` to end execution of the `.removeHead()` method.

Hint

You can check if `removedHead` has no value using an `if` statement:

```
if (!removedHead) {
  // Return statement goes here
}
```

3.

Outside the `if` statement, set the list's head equal to `removedHead`'s next node.

Hint

Remember to use `Node`'s `.getNextNode()` method to check if a node has a next node.

4.

Finally, return `removedHead`'s data.

Hint

You can access a node's data by using `someNode.data`.

LinkedList.js

```
const Node = require('./Node');

class LinkedList {

  constructor() {
    this.head = null;
  }

  addToHead(data) {
```

```
    const newHead = new Node(data);
    const currentHead = this.head;
    this.head = newHead;
    if (currentHead) {
      this.head.setNextNode(currentHead);
    }
  }

  addToTail(data) {
    let tail = this.head;
    if (!tail) {
      this.head = new Node(data);
    } else {
      while (tail.getNextNode() !== null) {
        tail = tail.getNextNode();
      }
      tail.setNextNode(new Node(data));
    }
  }

  removeHead() {
    const removedHead = this.head;
    if (!removedHead) {
      return;
    }
    this.head = removedHead.getNextNode();
    return removedHead.data;
  }
}

module.exports = LinkedList;
```

Node.js

```
class Node {
  constructor(data) {
    this.data = data;
    this.next = null;
  }
```

```
  setNextNode(node) {
    if (!(node instanceof Node)) {
      throw new Error('Next node must be a member of the Node class');
    }
    this.next = node;
  }

  getNextNode() {
    return this.next;
  }
}

module.exports = Node;
```

Printing

Nice! Now we have a bunch of helpful `LinkedList` methods under our belt. Our next step is to create a `.printList()` method so we can see our list as it changes.

While it's possible to just use `console.log()` on the list (try it!), we want to print it in a more understandable and readable way. `console.log()` will print the pointers of each node as well as the data, but we're just going to print the data while maintaining the order of the list.

To do this, we will create a `String` that holds the data from every node in the list. We'll start at the list's head and iterate through the list, adding to our `String` as we go.

For example, if we had a list for the days of the week, starting with Sunday, `.printList()` would print it as follows:

```
<head> Sunday Monday Tuesday Wednesday Thursday Friday Saturday
<tail>
```

Instructions

1.

Define a method named `.printList()`. Inside it, create:

- A `let` variable named `currentNode`, and set it equal to the list's head
- Another `let` variable named `output`, and set it equal to `'<head> '`

Then, log `output` to the console

Stuck? Get a hint

2.

While `currentNode` doesn't equal `null`, add its `data` and a `' '` to `output`. Then update `currentNode` to be its next node. Do this above your `console.log()` statement.

Hint

You can add to a `String` using the `+=` operator:

```
let someString = 'Hello';
someString += ' World';
// someString now equals 'Hello World'
```

3.

Finally, outside of the `while` loop, but before your `console.log()`, set `output` equal to itself concatenated with `'<tail>'`.

Hint

Be careful of where you place your code:

```
while (currentNode !== null) {
  output += currentNode.data + ' ';
  currentNode = currentNode.next;
}
// Update output here:

console.log(output);
```

LinkedList.js

```
const Node = require('./Node');

class LinkedList {

  constructor() {
    this.head = null;
  }

  addToHead(data) {
    const newHead = new Node(data);
```

```
    const currentHead = this.head;
    this.head = newHead;
    if (currentHead) {
      this.head.setNextNode(currentHead);
    }
  }

  addToTail(data) {
    let tail = this.head;
    if (!tail) {
      this.head = new Node(data);
    } else {
      while (tail.getNextNode() !== null) {
        tail = tail.getNextNode();
      }
      tail.setNextNode(new Node(data));
    }
  }

  removeHead() {
    const removedHead = this.head;
    if (!removedHead) {
      return;
    }
    this.head = removedHead.getNextNode();
    return removedHead.data;
  }

  printList() {
    let currentNode = this.head;
    let output = '<head> ';
    while (currentNode !== null) {
      output += currentNode.data + ' ';
      currentNode = currentNode.getNextNode();
    }
    output += '<tail>';
    console.log(output);
  }

}
```

```
module.exports = LinkedList;
```

Node.js

```
class Node {
  constructor(data) {
    this.data = data;
    this.next = null;
  }

  setNextNode(node) {
    if (!(node instanceof Node)) {
      throw new Error('Next node must be a member of the Node class');
    }
    this.next = node;
  }

  getNextNode() {
    return this.next;
  }
}

module.exports = Node;
```

Using the Linked List

You finished your `LinkedList` class! Now we're going to create an instance of that class and create a linked list of the seasons. We will add to it, remove from it, and finally print it out to check what we've done.

Instructions

1.

In **seasons.js**, define a `LinkedList` named `seasons`. Print it out – what do you expect to see?

Hint

Remember to use the `new` keyword when creating a class instance, and to use your `.printList()` method to print the list. Make sure to do this in **seasons.js**!

2.

Add `'summer'` to the head of the `seasons`, then add `'spring'` to the head. Print the list again.

Hint

Use your `.addToHead()` and `.printList()` methods.

3.

Add `'fall'` to the tail of `seasons`. Then add `'winter'` to the tail and print the list again.

Hint

Use your `.addToTail()` and `.printList()` methods.

4.

Finally, remove the element at the head of the list. Print the list to check that the correct element was removed.

Hint

Use your `.removeHead()` and `.printList()` methods.

seasons.js

```
const LinkedList = require('./LinkedList');

const seasons = new LinkedList();
seasons.printList();
seasons.addToHead('summer');
seasons.addToHead('spring');
seasons.printList();
seasons.addToTail('fall');
seasons.addToTail('winter');
seasons.printList();
seasons.removeHead();
seasons.printList();
```

LinkedList.js

```
const Node = require('./Node');

class LinkedList {
  constructor() {
    this.head = null;
```

```
}

addToHead(data) {
  const newHead = new Node(data);
  const currentHead = this.head;
  this.head = newHead;
  if (currentHead) {
    this.head.setNextNode(currentHead);
  }
}

addToTail(data) {
  let tail = this.head;
  if (!tail) {
    this.head = new Node(data);
  } else {
    while (tail.getNextNode() !== null) {
      tail = tail.getNextNode();
    }
    tail.setNextNode(new Node(data));
  }
}

removeHead() {
  const removedHead = this.head;
  if (!removedHead) {
    return;
  }
  this.head = removedHead.getNextNode();
  return removedHead.data;
}

printList() {
  let currentNode = this.head;
  let output = '<head> ';
  while (currentNode !== null) {
    output += currentNode.data + ' ';
    currentNode = currentNode.getNextNode();
  }
  output += '<tail>';
```

```
    console.log(output);
  }

}

module.exports = LinkedList;
```

Node.js

```
class Node {
  constructor(data) {
    this.data = data;
    this.next = null;
  }

  setNextNode(node) {
    if (!(node instanceof Node)) {
      throw new Error('Next node must be a member of the Node class');
    }
    this.next = node;
  }

  getNextNode() {
    return this.next;
  }
}

module.exports = Node;
```

Linked List Review

Congratulations, you have created and implemented a linked list class in JavaScript!

We did this by:

- Using our `Node` class to hold the data and links between nodes
- Implementing a `LinkedList` class to handle external operations on the list, like adding and removing nodes
- Creating an instance of our list, and using our `.printList()` method to track the changes we made

Instructions

Feel free to play around a bit with your code. Here are some ideas:

- Create a few nodes and add them to both ends of a new linked list
- Print your linked list out using your `.printList()` method
- Use `console.log()` on your list to see how it's different from your `.printList()` method
- Remove your linked list's head node
- Print your list again — was your original head node removed?
- So far you've built a method to remove the head of the list. How do you think you would remove a node that has a specific data? Try building a method to do that!