Review: Strings Lab

- Due No Due Date
- Points 1
- Submitting a website url

 (https://github.com/learn-co-curriculum/phase-1-review-strings-lab)  (https://github.com/learn-co-curriculum/phase-1-review-strings-lab/issues/new)

Learning Goals

- Concatenate strings with the `+` operator
- Interpolate variables and other JavaScript expressions inside template literals
- Read the MDN documentation on string methods and practice using a few

Introduction

For this lab, you've just been onboarded to the dev team working on Flatbook, the world's premier Flatiron School-based social network. At the moment, the view that our users see upon logging in is pretty generic. We'd like to improve the user experience by adding some custom greeting capabilities.

Work Through Failing Tests in a JavaScript Test Suite

If you haven't already, fork and clone this lab into your local environment. Remember to **fork** a copy into your GitHub account first, then **clone** from that copy. Navigate into its directory in the terminal, then run `code .` to open the files in Visual Studio Code.

Next, let's run `npm install` to install the dependencies then run the test suite with the `npm test` command. Our code is currently failing all of the tests, but we expected that because we haven't done anything yet. Let's get to work!

`currentUser`

The first test is telling us that `currentUser` is not defined. Let's go to `index.js` and write the following code:

```
const currentUser = 'Grace Hopper';
```

> **Note**: Generally, when the tests ask you to define something, you want to define it exactly as indicated in the test. But in this case, you don't have to write `'Grace Hopper'`, because the important part is the variable name: `currentUser`. You can use your own name, your pet's name, your favorite programmer's name — whatever you'd like.

Rerun the tests and you should see that the first one is passing.

welcomeMessage

The next failing test is similarly helpful, telling us exactly what we have to fix: `welcomeMessage contains "Welcome to Flatbook, "`.

Let's return to `index.js` and define our second variable below where we declared `currentUser`:

```
const currentUser = 'Grace Hopper';

const welcomeMessage = 'Welcome to Flatbook, ';
```

Rerun the tests; you should see a second passing test.

The third test tells us that `welcomeMessage` should contain the value stored in `currentUser`. This seems like it might contradict the second test a bit, but let's try it out. Let's erase `'Welcome to Flatbook, '` and set `welcomeMessage` equal to `currentUser` instead:

```
const currentUser = 'Grace Hopper';

const welcomeMessage = currentUser;
```

When we rerun the tests, we still have two passing. But now the first and third tests are passing instead of the first and second! That doesn't seem quite right.

It turns out that the tests want `welcomeMessage` to include *both* `'Welcome to Flatbook, '` and the value stored in `currentUser`. Maybe we can include both of them in a single string?

```
const currentUser = 'Grace Hopper';

const welcomeMessage = 'Welcome to Flatbook, currentUser';
```

If we rerun the tests, we're once again passing the second test, but we're back to failing the third test. The new error message for the third test gives us a hint about what's happening:

```
AssertionError: expected 'Welcome to Flatbook, currentUser' to contain 'Grace Hopper'
```

When JavaScript is expecting a variable to contain one thing, and it does not, that is known as an `AssertionError`. The test suite looked at the value stored in `welcomeMessage` and expected to find the string `'Grace Hopper'`, which is the value stored in `currentUser`. Instead, `welcomeMessage` contains the literal string `"currentUser"`. It's important to understand the distinction:

- `currentUser` is a *variable* that contains a string (`'Grace Hopper'` in our examples).
- `'currentUser'` is a *string*, ***not a variable***.

The JavaScript engine sees a matching pair of single quotes (`' '`), creates a new string, and assumes that *everything* in between the matching punctuation marks is part of that string. For example, if we add quotation marks around the first line of code that we wrote, it becomes a simple string consisting of 35 characters:

```
typeof "const currentUser = 'Grace Hopper';";
//=> "string"

"const currentUser = 'Grace Hopper';".length;
//=> 35

currentUser;
//=> Uncaught ReferenceError: currentUser is not defined
```

As demonstrated by the last line in that snippet, because we turned our code into a string it no longer functions as JavaScript code for declaring and assigning a `currentUser` variable.

Since we want `welcomeMessage` to contain both `'Welcome to Flatbook, '` and the value stored in `currentUser`, we have two options: **concatenation** and **interpolation**.

Concatenate Strings with the `+` Operator

String concatenation is a way to take two strings and add one to the other, creating a single, longer string. The easiest way to concatenate strings in JavaScript is with the `+` operator, like so:

```
"High " + "five!";
//=> "High five!"

"We" + ' ' + `can` + " " + 'concat' + `enate` + " as many strings " + 'as our heart ' + `desires.`;
//=> "We can concatenate as many strings as our heart desires."
```

Since our `currentUser` variable contains a string, we can concatenate it to the end of `'Welcome to Flatbook, '` to dynamically create a new string based on whatever value `currentUser` contains at a given moment:

```
const currentUser = 'Grace Hopper';

const welcomeMessage = 'Welcome to Flatbook, ' + currentUser;
```

If we run the test suite with our updated code, we'll see both the second and third tests passing! However, before we move on, let's talk about interpolation.

Interpolate Variables and Other JavaScript Expressions Inside Template Literals

String interpolation lets us dynamically insert values in the middle of a string. To do this, we need to use **template literals (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals)**, which are simply strings wrapped in backticks rather than single or double quotes. Template literals enable us to interpolate the value of a variable into a string by wrapping the variable in curly braces

preceded by a dollar sign: `${yourVariable}`. The `${}`, when inside backticks, tells the JavaScript engine that it needs to *interpret the value* of `yourVariable` and insert that value into the string. If you forget to use the backticks and use single or double quotes instead, the dollar sign, curly braces and variable name will all be inserted into the string instead of the variable's value.

Unlike string concatenation, template literals will also allow you to use multi-line strings. Wrapping the string in backticks preserves any new lines when the string is returned or output.

```
const myString = 'template literal';

const myNumber = 10;

const myBoolean = false;

`Saying that interpolation with ${myString}s is better than concatenation ${90 + myNumber}% of the time is simply ${myBo

Beware that new lines inside of a ${myString} will be preserved as new lines in the resulting ${typeof myString}!`;
//=> "Saying that interpolation with template literals is better than concatenation 100% of the time is simply false. Bu

// Beware that new lines inside of a template literal will be preserved as new lines in the resulting string!"
```

Note that, in the example above, one of the things we interpolated into our string is an arithmetic expression: `${90 + myNumber}`. We aren't limited to interpolating just variables — we can use *any expression* inside the curly braces.

While, for most purposes, the choice of whether to use concatenation or string interpolation is primarily a matter of personal preference, JavaScript programmers tend to use string interpolation for all but the simplest of cases.

Let's rewrite our `welcomeMessage` to use a template literal:

```
const currentUser = 'Grace Hopper';

const welcomeMessage = `Welcome to Flatbook, ${currentUser}`;
```

The first three tests are still passing, but the fourth wants our `welcomeMessage` to end with an exclamation point. The fix is as simple as adding a `!` as the last character in the template literal:

```
const currentUser = 'Grace Hopper';

const welcomeMessage = `Welcome to Flatbook, ${currentUser}!`;
```

Four tests down, six to go!

Read the MDN Documentation on String Methods and Practice Using a Few

`excitedWelcomeMessage`

Sometimes we get so excited when someone logs into their Flatbook account that we just want to shout out loud. We *could* copy over most of the code from `welcomeMessage` and then change every character to its uppercase equivalent, but as developers we try not to repeat ourselves. Instead, let's use the `.toUpperCase()` string method:

```
const currentUser = 'Grace Hopper';

const welcomeMessage = `Welcome to Flatbook, ${currentUser}!`;

const excitedWelcomeMessage = welcomeMessage.toUpperCase();
```

All strings in JavaScript have access to the same set of default methods, which are common operations like changing a string and returning the new version, searching through a string for specific character(s) and returning the match, and so on. For example, we can use `.toUpperCase()` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/toUpperCase) and `.toLowerCase()` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/toLowerCase) on a string to make the entire string uppercase or lowercase. There are lots of other **string methods** (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String#Methods_2) that you'll find useful at various points throughout your JavaScript programming career.

Rerun the tests; you should see the first seven tests passing. Woohoo!

shortGreeting

The mobile team at Flatbook is busy redesigning the site for smaller devices, and they're a bit concerned about how much real estate the `welcomeMessage` takes up on the screen. They want us to create a shorter version that truncates the `currentUser` 's name into just their first initial.

If you take a look at the first error, you'll see that the JavaScript engine is telling us that it can't find `shortGreeting` :

```
shortGreeting
  contains "Welcome, "
    ReferenceError: shortGreeting is not defined
```

Once we define it in `index.js` :

```
...

const shortGreeting = '';
```

we see a new error from the test suite:

```
shortGreeting
  contains "Welcome, "
    AssertionError: expected '' to contain 'Welcome, '
```

It expected `shortGreeting` to contain the string `"Welcome, "` , but `shortGreeting` is currently an empty string, `''` . We can fix that now:

```
...

const shortGreeting = 'Welcome, ';
```

Next up is another `AssertionError` , this one checking that `shortGreeting` contains the first letter from `currentUser` :

```
shortGreeting
  contains the first initial of the name stored in the 'currentUser' variable
```

```
    AssertionError: expected 'Welcome, ' to contain 'G'
```

To get a sense of how specific the tests are, let's start by adding the entirety of `currentUser` to `shortGreeting`:

```
const currentUser = 'Grace Hopper';

...

const shortGreeting = `Welcome, ${currentUser}`;
```

Notice that we changed the single quotes to backticks, which allows us to interpolate with `${ }`.

The new error reads as follows:

```
shortGreeting
  contains the first initial of the name stored in the 'currentUser' variable
    AssertionError: expected 'Welcome, Grace Hopper' to not contain 'race Hopper'
```

The test suite checks that `shortGreeting` contains the first character in `currentUser` (`G` in our example) and that it *doesn't* contain the rest of the string (`race Hopper`).

There are a few different ways we could get just the first character of `currentUser`. The easiest would be to use **bracket notation or the `.charAt()` method (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String#Character_access)** to grab the character at index `0`:

```
'Edsger Dijkstra'[0];
//=> "E"

'Edsger Dijkstra'.charAt(0);
//=> "E"
```

However, it's a good practice to make our code flexible and future-proof it a bit. What if our product team decides it would be better to shorten `currentName` to two characters instead of one? Or three characters?

For the added flexibility, we're going to use `.slice()`, but you can always explore the **MDN documentation on string methods (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String#Methods_2)** to pick out your own strategy.

`.slice()`

If you take a look at the documentation for **`.slice()` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/slice)**, you will see the following description:

> The **slice()** method extracts a section of a string and returns it as a new string, without modifying the original string.

The method takes two arguments: the index at which the extraction should begin and the index *before which* it should end. When we talk about indexes of a string, we're talking about how to access specific characters at various points within the string. Recall that computers start counting with 0. Because we start at index `0` instead of `1`, the index of each character in a string is always one less than the character's place in the string. The second character is at index `1`, the fifth at index `4`, the twelfth at index `11`, and so on. The index of the last character is always one less than the **length (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/length)** of the string:

```
'Edsger Dijkstra'.length;
//=> 15

'Edsger Dijkstra'[15];
//=> undefined

'Edsger Dijkstra'[14];
//=> "a"
```

If we omit both arguments, `.slice()` will return a full copy of the original string:

```
'Edsger Dijkstra'.slice();
//=> "Edsger Dijkstra"
```

If we provide a single argument, `.slice()` will return a copy from that index to the end of the string. For example, to grab Dijkstra's last name, we could start the slice on index `7`:

```
'Edsger Dijkstra'.slice(7);
//=> "Dijkstra"
```

If we wanted the first three characters of Dijkstra's name, we would specify `0` as the first argument, the index at which to start, and `3` as the second argument, the index before which to end:

```
'Edsger Dijkstra'.slice(0, 3);
//=> "Eds"
```

To satisfy our team's current specifications for `shortGreeting`, we need to start our slice at index `0` and end it before index `1`:

```
currentUser.slice(0, 1);
```

Now, when our product team asks us to use the first two characters of `currentUser`, the change is as simple as `currentUser.slice(0, 1)` → `currentUser.slice(0, 2)`.

Add an exclamation point to the end, and the entire test suite should be passing:

```
const currentUser = 'Grace Hopper';

...

const shortGreeting = `Welcome, ${currentUser.slice(0, 1)}!`;
```

After you have all the tests passing, remember to commit and push your changes up to GitHub, then submit your work to Canvas using CodeGrade. If you need a reminder, go back to the **Completing and Submitting Assignments with CodeGrade (https://github.com/learn-co-curriculum/phase-1-completing-assignments-with-codegrade)** lesson to review the process.

Great work!

Resources

- StackExchange – How to open the JavaScript console (https://webmasters.stackexchange.com/questions/8525/how-do-i-open-the-javascript-console-in-different-browsers/77337#77337)
- MDN — Template literals (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals)
- DRY — Don't Repeat Yourself (https://en.wikipedia.org/wiki/Don%27t_repeat_yourself)
- MDN — String — `.length` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/length)
- MDN — String — Character access (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String#Character_access)
- MDN — String — Methods (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String#Methods_2)
  - MDN — `.toUpperCase()` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/toUpperCase)
  - MDN — `.toLowerCase()` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/toLowerCase)
  - MDN — `.slice()` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/slice)