Checkpoint 4

Strings

Remember when you first started learning about JavaScript data types? Now that you're familiar with some of the basics, you're ready to dive deeper into one specific data type: the `string`.

In this checkpoint, you'll learn more about how to write strings in JavaScript and explore new ways to access different parts of a string. You'll also learn how to transform a string into an array.

By the end of this checkpoint, you will be able to do the following:

- Access parts of a string with bracket notation and methods
- Create an array from a string based on particular criteria, and join arrays into strings
- Write strings that embed expressions using template literals

Accessing strings

As you know, a string data type is used to represent text. Strings are set off in quotes, and they can contain a variety of characters, such as letters, numbers, and symbols. So when it comes to accessing strings, you'll essentially be targeting individual characters in order to perform a specific task.

Take a look at the following string. What do you see?

```
const pangram = "The Five Boxing Wizards Jump Quickl
```

As you might've noticed, the above `pangram` string uses every letter of the alphabet. The string is also written in *Title Case*—in other words, the first letter of every word is uppercase (or capitalized). Imagine you want to update this string so that it's in *Sentence case*, which means only the first letter of the sentence is uppercase. How might you do that? You'd follow these general steps:

1. Make sure that the first letter of the sentence is uppercase.
2. Make all other letters lowercase.

To do this, you'll need to access individual characters in the string.

Characters: A refresher

Before moving forward, you might need a quick refresher on characters. As mentioned above, characters in strings can be alphanumeric characters, punctuation marks, spaces between words, or other types of symbols that increase the length of the string. Consider the ple from above:

```
const pangram = "The Five Boxing Wizards Jump Quickl
pangram.length; //> 37
```

The `pangram` is 37 characters long. That includes the period `.` and empty spaces between words, but not the double quotes (`"` and `"`) enclosing the string. You'll learn a bit more about how to work with quotes in strings below.

Bracket notation

Back to the task at hand: changing this string from title case into sentence case. One way you could do this is by using *bracket notation*. Just like with arrays, bracket notation can be used to access individual characters at any given index of a string. See below:

Outline

```
const word = "Wizards";
word[0]; //> "W"
word[2]; //> "z"
word[9]; //> undefined
```

Now, take a look at the following function, which will solve the case problem you've been given. What do you notice?

```
function sentenceCase(sentence) {
  const firstCharacter = sentence[0];
  let result = firstCharacter.toUpperCase();

  for (let i = 1; i < sentence.length; i++) {
    const character = sentence[i];
    result += character.toLowerCase();
  }
```

```
    return result;
  }
```

Take a moment to walk through the above code. Here's what it's doing:

1. It creates a new variable called `firstCharacter`. It sets `firstCharacter` to be equal to the first character of the inputted `sentence` string, using bracket notation.
2. It creates a new variable called `result`. It sets that variable to be equal to the `firstCharacter` variable, set to uppercase with the `toUpperCase()` method.
3. It begins a `for` loop, which starts at an index of `1`, thus skipping over the first character of the sentence.
4. In the `for` loop, the variable `character` is created, which is set to be equal to `sentence[i]`. With bracket notation, you can access each character of a string, just like you would each item of an array.
5. The code adds the character to the `result` variable, setting it to lowercase at the same time.
6. Finally, `return result` will return the result for you.

The `substr()` method

But bracket notation is just one approach. You can also solve your casing problem using a built-in JavaScript method, called the `substr()` method, which will actually make this process a litt sier. Check out the code sample below:

```
function sentenceCase(sentence) {
  const first = sentence.substr(0, 1);
  const rest = sentence.substr(1);

  return first.toUpperCase() + rest.toLowerCase();
}
```

The `substr()` method, or substring method, allows you to extract a specific section of characters in a string. It takes two arguments:

1. The index of the first character to include in the substring
2. The number of characters to extract

The above function works as follows:

1. It sets the `first` variable to be equal to *only* the first character. In other words, `substr(0, 1)` means that the substring will begin on index `0` and only include `1` character.
2. It sets the `rest` variable to be equal to every character starting from the first index. If you do not include a second argument, like in `substr(1)`, the substring will consist of every character following the given index.
3. It joins the two strings together, using `toUpperCase()` and `toLowerCase()` as appropriate.

Do this

Use `substr()`

Time to try it yourself. Take a look at the following examples of `substr()`. Before running the code on your own, evaluate the code in your head and predict what will happen.

```
const title = "Guards! Guards!";
title.substr(3); //> ?
title.substr(6, 4); //> ?
title.substr(25); //> ?
```

Splitting and joining strings

Now that you've looked at a couple of ways to access string characters, take a step back to analyze the functions at work here.

As the name suggests, the `sentenceCase()` function could be described as one that capitalizes the first character in a string and sets all other characters to lowercase. That's what you were trying to do above. But you could also reuse this function to create a `titleize()` function in order to turn phrases into titles (with title case formatting). For example, you could expect the `titleize()` function to work like this:

```
const title = "the light FANTASTIC";
titleize(title); //> "The Light Fantastic";
```

There are a few ways to accomplish this task. Below is one approach that makes use of the `split()` and `join()` methods, seen below.

```
function titleize(title) {
  const words = title.split(" ");
  let result = [];
```

```
    for (let i = 0; i < words.length; i++) {
      const capitalized = sentenceCase(words[i]);
      result.push(capitalized);
    }

    return result.join(" ");
  }

  titleize("the light FANTASTIC"); //> "The Light Fant
```

In the code sample above, the `split()` method separates the string based on the string given to it as an argument. Here's an example:

```
  const title = "the light FANTASTIC";
  title.split(" "); //> [ "the", "light", "FANTASTIC"
```

Outline

The argument given to `split()` is a string with a space. Therefore, an array is created in which each new item is separated by the string. Notice that in the resulting array above, all of the spaces have been removed.

Now, take a look at the `join()` method. The `join()` method is called on an array and joins all the elements in that array together with the supplied argument. In the case of the `titleize()` function, the `join()` joins all the capitalized words into a single string, separated by spaces.

```
  const result = ["The", "Light", "Fantastic"];
  result.join(" "); //> "The Light Fantastic";
```

Check out the Mozilla Developer Network, or MDN, in the links provided here to learn more about both the split() method and the join() method.

Do this

Use `split()` and `join()`

Take a look at the following examples of `split()` and `join()`. Before running the code on your own, evaluate the code in your head and predict what will happen.

```
const title = "Guards! Guards!";
title.split("!"); //> ?
title.split(""); //> ?
title.split("guards"); //> ?

const titleArr = ["The", "Light", "Fantastic"];
titleArr.join("-"); //> ?
titleArr.join("_-_"); //> ?
titleArr.join(); //> ?
```

Template literals

Joining together multiple strings can end up looking pretty messy. Previously, you've had to concatenate multiple parts of a string with the `+` operator to accomplish this task. Take a look at how the following function concatenates strings.

```
function bookSale(title, priceInCents) {
  const price = (priceInCents / 100).toFixed(2)
  return titleize(title) + " is on sale for $"
}
```

```
bookSale("the light fantastic", 950); //> "The Light
```

The above function works, but it doesn't look that great. Fortunately, JavaScript has a feature called *template literals*, which can help make this function look cleaner and clearer. Check it out:

```
function bookSale(title, priceInCents) {
  const price = (priceInCents / 100).toFixed(2);
  return `${titleize(title)} is on sale for $${price}
}
```

Template literals allow you to embed expressions and avoid using multiple `+` operators just to join a string. The syntax of a template literal is seen here: `` `${titleize(title)} is on sale for $${price}.` `` Here are the key pieces:

- Begin and end your string using backticks, which look like this `` ` ``. They are accessible on the tilde `~` key on your keyboard.
- Place variables or expressions inside of curly braces `{}`, which should be preceded by a dollar sign `$`. It should look like this: `${}`.

Do this

Use template literals

Given the variables below, construct a string using template literals that results in the following sentence.

```
The price of 'Interesting Times' by Terry Pratchett i
```

```
const title = "Interesting Times";
const author = "Terry Pratchett";
const price = 8.99;
```

If you're having trouble, you can peek at the answer below.

```
`The price of '${title}' by ${author} is $${price}.`;
```

Escaping strings

When reviewing older code, you may see examples like the snippet below:

Outline

```
const firstSentence = "Will tugged at his mother's ha
```

You likely noticed that this string was created with double quotes, but that there are also double quotes *in* the string. The backslash `\` you see here is used to *escape* the string. Escaping a string means you're providing a backslash to allow for the following character to be seen as part of the string rather than part of the syntax. In the case above, the `\` tells JavaScript that it should treat this double quote as part of the string, not as the closing quotation.

The sample string above is just fine. But as you learned with te e literals, there may be a better way to write it. Take a look:

```
const firstSentence = `Will tugged at his mother's ha
```

Checkpoint

This checkpoint will be autograded. Please click the link below to open your assignment in a new tab. Once you complete the assignment, you will see a button allowing you to submit your answers and move on to the next checkpoint.

Your work

03.22.21 **Approved**

Outline

Completed **Next checkpoint**

How would you rate this content?

Report a typo or other issue

Go to Overview

Outline