Lesson Weekend

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Adding and Removing HTML Elements

Text Cheat sheet

Workflow Tip

- Test out code to add new elements in the DevTools console.
- Use the Elements tab of the DevTools to inspect newly created elements to verify their placement in the DOM.
- Reference documentation as needed.

Code Examples

Creating New Elements

To create new HTML elements, we'll use the `document.createElement()` (https://developer.mozilla.org/en-US/docs/Web/API/Document/createElement) method.

- Argument: the tag name as a string.
- Return value: empty tag, an HTML element object corresponding to the tag created.
- Visit the MDN documentation on the HTML DOM API (https://developer.mozilla.org/en-

US/docs/Web/API/HTML_DOM_API#html_element_interfaces_2) to see a list of all HTML element objects.

```
> const pElement = document.createElement("p");
> pElement;
<p></p>
> Object.prototype.toString.call(pElement);
"[object HTMLParagraphElement]"
```

Adding Attributes

To add, remove, or get attributes to any HTML element, we need to use these methods:

- `Element.setAttribute()` (https://developer.mozilla.org/en-US/docs/Web/API/Element/setAttribute)
- `Element.removeAttribute()` (https://developer.mozilla.org/en-US/docs/Web/API/Element/removeAttribute)
- `Element.getAttribute()` (https://developer.mozilla.org/en-US/docs/Web/API/Element/getAttribute)

Adding Text

We'll use the `Element.append()` (https://developer.mozilla.org/en-US/docs/Web/API/Element/append) method to add text to an element. We call this method on the element that we want to add text to, and the argument we pass in will be string with the text we want to add. Let's look at the code!

```
> const pElement = document.createElement("p");
> pElement;
<p></p>
> pElement.append("text");
> pElement;
<p>text</p>
> p.innerText;
"text"
```

Adding Element to DOM with `Element.append()`, `Element.prepend()`, `Element.after()`, `Element.before()`

The (`Element.append()` (https://developer.mozilla.org/en-US/docs/Web/API/Element/append)), (`Element.prepend()` (https://developer.mozilla.org/en-US/docs/Web/API/Element/prepend)), (`Element.after()` (https://developer.mozilla.org/en-US/docs/Web/API/Element/after)), and (`Element.before()` (https://developer.mozilla.org/en-US/docs/Web/API/Element/before)) methods all add elements to the DOM, but at different locations:

- the `Element.prepend()` method will add a new element inside of the element we're calling the method on, at the top.
- the `Element.append()` method will add a new element inside of the element we're calling the method on, at the bottom.
- the `Element.after()` method will add a new element after the element we're calling the method on.
- the `Element.before()` method will add a new element before the element we're calling the method on.

```
> const pElement = document.createElement("p");
> pElement.append("text");
> const firstDiv = document.querySelector("div");
> firstDiv.append(pElement);
> const liElement = document.createElement("li");
> liElement.append("text");
> const firstUl = document.querySelector("ul");
> firstUl.prepend(liElement);
```

```
> const p1 = document.createElement("p");
> p1.append("text");
> const p2 = document.createElement("p");
> p2.append("text");
> const firstH2 = document.querySelector("h2");
> firstH2.after(p1);
> firstH2.before(p2);
```

Special Note

Note that you can only add the same element once, no matter which method you are using (`Element.append()` , `Element.prepend()` , `Element.after()` , `Element.before()`). You can't do it twice. For example, the following code would only leave the paragraph after the heading:

```
> const pElement = document.createElement("p");
> pElement.append("text");
> const firstH2 = document.querySelector("h2");
> firstH2.before(pElement);
> firstH2.after(pElement);
```

And we'll see the same behavior with `Element.append()` and `Element.prepend()` .

If we want a paragraph added before and after the heading we're targeting, we'll have to create two paragraph elements:

```
> const p1 = document.createElement("p");
> const p2 = document.createElement("p");
> p1.append("text");
> p1.append("other text");
> const firstH2 = document.querySelector("h2");
> firstH2.before(p1);
> firstH2.after(p2);
```

Removing an Element

If we want to remove an element from the DOM, we simply have to get it with a `document` method and call the `Element.remove()` (https://developer.mozilla.org/en-US/docs/Web/API/Element/remove) method on it.

```
> const firstDivOnPage = document.querySelector("div");
> firstDivOnPage.remove();
```

Adding Multiple Elements to the DOM at Once

We can add multiple elements at once with with the `Element.append()`, `Element.prepend()`, `Element.after()`, and `Element.before()` methods by passing in a series of arguments, each one representing an HTML element.

Here's an example with `Element.append()`:

```
> const ul = document.createElement("ul");
> ul.setAttribute("id", "iceCream");
> document.querySelector("div").append(ul);
> const liOne = document.createElement("li");
> const liTwo = document.createElement("li");
> const liThree = document.createElement("li");
> liOne.append("Chocolate");
> liTwo.append("Vanilla");
> liThree.append("Strawberry");
> document.getElementById("iceCream").append(liOne, liTwo,
liThree);
```

Previous (/introduction-to-programming/arrays-and-looping/document-query-methods-that-return-collections)
Next (/introduction-to-programming/arrays-and-looping/debugging-in-javascript-using-a-linter)

Lesson 10 of 50
Last updated February 28, 2023

disable dark mode

Epicodus (http://www.epicodus.com)

Lesson Monday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Introduction to Looping

Text Cheat sheet

Terminology

Loop: A piece of code that repeats until a condition is met.

Callback function: A function that is passed as an argument into another functions to be executed later.

Anonymous function: An unnamed function. They can be stored using a function expression or used as a callback in another function such as `Array.prototype.forEach()`.

syntactic sugar: syntax that makes a piece of code easier to write or use.

Example

```
const languages = ['HTML', 'CSS', 'JavaScript'];
languages.forEach(function(language) {
  alert('I love ' + language + '!');
});
```

Previous (/introduction-to-programming/arrays-and-looping/practice-javascript-arrays)
Next (/introduction-to-programming/arrays-and-looping/foreach-loops)

Lesson 17 of 50
Last updated March 24, 2023

disable dark mode

Epicodus (http://www.epicodus.com)

Lesson Monday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / forEach Loops

Text Cheat sheet

Examples

Logging Values to the Console:

```
const array = [0,1,2,3,4,5];
array.forEach(function(number) {
  console.log(number * 2);
});
```

Creating a New Array with Modified Elements

```
const array = [0,1,2,3,4,5];
let doubledArray = [];
array.forEach(function(element) {
  doubledArray.push(element * 2);
});
```

Using a Loop to Sum Numbers

```
const array = [0,1,2,3,4,5];
let sum = 0;
array.forEach(function(element) {
  sum += element;
});
```

Using a Loop to Make a String

```
let thingsILike = "I like...";
const arrayOfThingsILike = ["bubble baths", "kittens", "goo
d books", "clean code"];
arrayOfThingsILike.forEach(function(thing) {
  thingsILike = thingsILike.concat(" " + thing + "!");
});
```

Using A Loop to Add Elements to the DOM

```
const arrayOfThingsILike = ["bubble baths", "kittens", "goo
d books", "clean code"];
const ul = document.querySelector("ul#likable-things");
arrayOfThingsILike.forEach(function(thing) {
  const li = document.createElement("li");
  li.append(thing);
  ul.append(li);
});
```

Previous (/introduction-to-programming/arrays-and-looping/introduction-to-looping)
Next (/introduction-to-programming/arrays-and-looping/practice-looping)

Lesson 18 of 50
Last updated March 24, 2023

disable dark mode

Epicodus (http://www.epicodus.com)

Lesson Monday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Gathering Data with Checkboxes

Text Cheat sheet

Examples

If we had the following form containing a group of checkboxes:

transportation_survey/index.html

```
<!DOCTYPE html>
<html lang="en-US">
<head>
  <title>Transportation Survey</title>
  <link href="css/styles.css" rel="stylesheet" type="text/c
ss">
  <script src="js/scripts.js"></script>
</head>
<body>
  <h2>Transportation Survey</h2>
  <form id="transportation_survey">
    <p>In the past year, I have used the following modes of
transportation to travel to work or school:</p>
    <label>
      <input type="checkbox" name="transportation-option" v
alue="bike">
      Riding a bike.
    </label><br />
    <label>
      <input type="checkbox" name="transportation-option" v
alue="car">
      Driving a car.
    </label><br />
    <label>
      <input type="checkbox" name="transportation-option" v
alue="carpool">
      Carpooling with others.
    </label><br />
    <label>
      <input type="checkbox" name="transportation-option" v
alue="walk">
      Walking.
    </label><br />
    <label>
      <input type="checkbox" name="transportation-option" v
alue="bus">
      Riding the bus.
    </label><br />
    <label>
      <input type="checkbox" name="transportation-option" v
alue="train">
```

```
      Riding the train.
    </label><br />
    <label>
      <input type="checkbox" name="transportation-option" v
alue="streetcar">
      Riding the streetcar.
    </label><br />
    <label>
      <input type="checkbox" name="transportation-option" v
alue="taxi">
      Taking a taxi.
    </label><br />
    <label>
      <input type="checkbox" name="transportation-option" v
alue="rideshare">
      Using a rideshare service like Lyft or Uber.
    </label><br />
    <label>
      <input type="checkbox" name="transportation-option" v
alue="skateboard">
      Skateboarding.
    </label><br />
    <label>
      <input type="checkbox" name="transportation-option" v
alue="rollerblade">
      Rollerblading.
    </label><br />
    <label>
      <input type="checkbox" name="transportation-option" v
alue="scooter">
      Riding a scooter.
    </label><br />
    <label>
      <input type="checkbox" name="transportation-option" v
alue="other">
      Another mode of transportation not listed here.
    </label><br />
    <button type="submit">Submit Survey</button>
  </form>
</body>
</html>
```

This JavaScript would retrieve all selected checkboxes, create a heading for the survey results, and append their values to a span in our HTML:

transportation_survey/js/scripts.js

```
function handleForm(event) {
  event.preventDefault();
  const userSelections = document.querySelectorAll("input[n
ame=transportation-option]:checked");
  const userSelectionsArray = Array.from(userSelections);

  const resultsHeading = document.createElement("h2");
  resultsHeading.append("You use the following methods of t
ransportation to travel to work or school:");
  document.body.append(resultsHeading);

  userSelectionsArray.forEach(function(element) {
    const paragraph = document.createElement("p");
    paragraph.append(element.value);
    document.body.append(paragraph);
  });
}

window.addEventListener("load", function() {
  document.querySelector("form#transportation_survey").addE
ventListener("submit", handleForm);
});
```

- `event.preventDefault()` is added to prevent the 'submit' event's default action of refreshing the page.
- `const userSelections = document.querySelectorAll("input[name=transportation-option]:checked");` is added to get all of the checkbox inputs that have been checked. We could instead pass in these arguments to the `document.querySelectorAll()` method:
  - `"[name=transportation-option]:checked"`
  - `"input:checked"`

    - Whatever you choose really depends on what's easiest to read so that your code is easy to understand, and what other inputs and forms you have in your webpage.
- We turn the variable `userSelections` which is a `NodeList` object into an array with: `const userSelectionsArray = Array.from(userSelections);`
- We call `Array.prototype.forEach()` on the `userSelectionsArray` variable, passing in a callback function as the argument to the method.

The Callback Function

The callback function is as follows:

```
function(element) {
  const paragraph = document.createElement("p");
  paragraph.append(element.value);
  document.body.append(paragraph);
}
```

The `element` parameter is the placeholder for the actual array element. Each element in our array is an `HTMLInputElement` with a `value` property that gets the value of the input. So, for every element in our array, we:

- Create a new HTML paragraph element, with `const paragraph = document.createElement("p");`.
- Add the input's value (`element.value`) as the text to that paragraph element, with `paragraph.append(element.value);`.
- Add the new paragraph to the inside/end of our document's body tag with `document.body.append(paragraph);`.

An Alternate Way of Organizing Checkboxes

The following organization of checkboxes connects the label and the input by giving the same value to the `for` (on the label) and `id` (on the input) attributes:

```
<label for="bike">Riding a bike.</label><br />
<input type="checkbox" name="transportation-option" value
="car" id="car">
<label for="car">Driving a car.</label><br />
<input type="checkbox" name="transportation-option" value
="carpool" id="carpool">
<label for="carpool">Carpooling with others.</label><br />
<input type="checkbox" name="transportation-option" value
="walk" id="walk">
<label for="walk">Walking.</label><br />
<input type="checkbox" name="transportation-option" value
="bus" id="bus">
<label for="bus">Riding the bus.</label><br />
<input type="checkbox" name="transportation-option" value
="train" id="train">
<label for="train">Riding the train.</label><br />
<input type="checkbox" name="transportation-option" value
="streetcar" id="streetcar">
<label for="streetcar">Riding the streetcar.</label><br />
<input type="checkbox" name="transportation-option" value
="taxi" id="taxi">
<label for="taxi">Taking a taxi.</label><br />
<input type="checkbox" name="transportation-option" value
="rideshare" id="rideshare">
<label for="rideshare">Using a rideshare service like Lyft
or Uber.</label><br />
<input type="checkbox" name="transportation-option" value
="skateboard" id="skateboard">
<label for="skateboard">Skateboarding.</label><br />
<input type="checkbox" name="transportation-option" value
="rollerblade" id="rollerblade">
<label for="rollerblade">Rollerblading.</label><br />
<input type="checkbox" name="transportation-option" value
="scooter" id="scooter">
<label for="scooter">Riding a scooter.</label><br />
<input type="checkbox" name="transportation-option" value
="other" id="other">
<label for="other">Another mode of transportation not liste
d here.</label><br />
```

Previous (/introduction-to-programming/arrays-and-looping/practice-looping)
Next (/introduction-to-programming/arrays-and-looping/practice-foreach-loops)

Lesson 20 of 50
Last updated March 24, 2023

disable dark mode

Epicodus (http://www.epicodus.com)

Lesson Tuesday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Overview of Test-Driven Development (TDD) with Text Analyzer

Text Cheat sheet

Terminology

- **Test-Driven Development**: A workflow used by developers across coding languages. In TDD, we write tests that describe our application's behavior. Then we write the minimal amount of code we need to get the test passing. The goal is to break larger problems into more manageable steps and also to make sure our code is working correctly.
- **Tests**, **Specifications** or **Specs**: Examples of small, isolated behaviors a program should demonstrate, including input and output examples. "Spec" and "test" are interchangeable terms.

Example

Here's an example of a pseudocode test:

```
Describe: wordCounter()

Test: "It should return 1 if a passage has just one word."
Code:
const text = "hello";
wordCounter(text);
Expected Output: 1
```

Parts of a Test

- **Describe** is used to organize a group of tests, such as all tests for a specific function.
- **Test** is a description, in plain English, of what the test will do.
- **Code** is any code that needs to run in order to get the expected output.
- **Expected Output** is the output we expect for the test to correctly pass.

Previous (/introduction-to-programming/arrays-and-looping/building-a-text-analyzer)
Next (/introduction-to-programming/arrays-and-looping/text-analyzer-with-tdd-wordcounter)

Lesson 24 of 50
Last updated March 24, 2023

disable dark mode

(http://www.epicodus.com)

Lesson Tuesday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Text Analyzer with TDD: wordCounter()

Text Cheat sheet

Tests

Here are all the tests we wrote in this lesson. They should provide a good sense both of what a pseudocode test should look like and a general progression from simplest behavior to more complex behavior.

```
Describe: wordCounter()

Test: "It should return 1 if a passage has just one word."
Code:
const text = "hello";
wordCounter(text);
Expected Output: 1

Test: "It should return 2 if a passage has two words."
Code:
const text = "hello there";
wordCounter(text);
Expected Output: 2

Test: "It should return 0 for an empty string."
Code: wordCounter("");
Expected Output: 0

Test: "It should return 0 for a string that is only space
s."
Code: wordCounter("            ");
Expected Output: 0

Test: "It should not count numbers as words."
Code: wordCounter("hi there 77 19");
Expected Output: 2
```

Previous (/introduction-to-programming/arrays-and-looping/overview-of-test-driven-development-tdd-with-text-analyzer)
Next (/introduction-to-programming/arrays-and-looping/text-analyzer-with-tdd-numberofoccurrencesintext)

Lesson 25 of 50
Last updated February 28, 2023

disable dark mode

Epicodus (http://www.epicodus.com)

Lesson Tuesday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Text Analyzer with TDD: numberOfOccurrencesInText()

Text Cheat sheet

Tests

Here are all the tests we wrote for both `wordCounter()` and `numberOfOccurrencesInText()`. They should provide a good sense both of what a pseudocode test should look like and a general progression from simplest behavior to more complex behavior.

```
Describe: wordCounter()

Test: "It should return 1 if a passage has just one word."
Code:
const text = "hello";
wordCounter(text);
Expected Output: 1

Test: "It should return 2 if a passage has two words."
Code:
const text = "hello there";
wordCounter(text);
Expected Output: 2

Test: "It should return 0 for an empty string."
Code: wordCounter("");
Expected Output: 0

Test: "It should return 0 for a string that is only space
s."
Code: wordCounter("            ");
Expected Output: 0

Test: "It should not count numbers as words."
Code: wordCounter("hi there 77 19");
Expected Output: 2


Describe: numberOfOccurrencesInText()

Test: "It should return 0 occurrences of a word for an empt
y string."
Code:
const text = "";
const word = "red";
numberOfOccurrencesInText(word, text);
Expected Output: 0

Test: "It should return 1 occurrence of a word when the wor
d and the text are the same."
Code:
```

```
const text = "red";
const word = "red";
numberOfOccurrencesInText(word, text);
Expected Output: 1

Test: "It should return 0 occurrences of a word when the wo
rd and the text are different."
Code:
const text = "red";
const word = "blue";
numberOfOccurrencesInText(word, text);
Expected Output: 0

Test: "It should return the number of occurrences of a wor
d."
Code:
const text = "red blue red red red green";
const word = "red";
numberOfOccurrencesInText(word, text);
Expected Output: 4

Test: "It should return a word match regardless of case."
Code:
const text = "red RED Red green Green GREEN";
const word = "Red";
numberOfOccurrencesInText(word, text);
Expected Output: 3

Test: "It should return a word match regardless of punctuat
ion."
Code:
const text = "Red! Red. I like red, green, and yellow.";
const word = "Red";
numberOfOccurrencesInText(word, text);
Expected Output: 3
```

Previous (/introduction-to-programming/arrays-and-looping/text-analyzer-with-tdd-wordcounter)
Next (/introduction-to-programming/arrays-and-looping/practice-extending-text-analyzer-business-logic-with-tdd)

Lesson 26 of 50
Last updated February 28, 2023

disable dark mode

(http://www.epicodus.com)

Lesson Tuesday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Separation of Logic: Fixing a Bug in Text Analyzer

Text Cheat sheet

Examples of Bad Separation of Logic

Don't Access/Manipulate the DOM from your Business Logic

The following code is bad:

```
// This is bad! Don't put any logic to alter the DOM in you
r function!

function numberOfOccurrencesInText(word, text) {
  if (word.trim().length === 0) {
    // I'm directly altering the DOM from my business logi
c! This is bad.
    document.getElementById("total-count").innerText = 0;
  }
  const textArray = text.split(" ");
  let wordCount = 0;
  textArray.forEach(function(element) {
    if (element.toLowerCase().includes(word.toLowerCase()))
{
      wordCount++;
    }
  });
  // I'm directly altering the DOM from my business logic!
This is bad.
  document.getElementById("total-count").innerText = wordCo
unt;
}
```

Don't Include a Message to the User in the Return Values of Your Business Logic Functions

Instead, you should be returning just the value, without a message to the user. Any messages to the user should be in the UI logic. Don do this:

```
// Not so good either, but for less obvious reasons.

function numberOfOccurrencesInText(word, text) {
  if (word.trim().length === 0) {
    return "You need to enter a word!";
  }
  const textArray = text.split(" ");
  let wordCount = 0;
  textArray.forEach(function(element) {
    if (element.toLowerCase().includes(word.toLowerCase()))
{
      wordCount++;
    }
  });
  return "There are " + wordCount + " total matches!";
}
```

Tests

Here are all the tests we wrote for both `wordCounter()` and `numberOfOccurrencesInText()`. **This set of tests now includes the test we wrote to fix the bug that was caused when the `word` variable was an empty string `""`.**

These should provide a good sense both of what a pseudocode test should look like and a general progression from simplest behavior to more complex behavior.

```
Describe: wordCounter()

Test: "It should return 1 if a passage has just one word."
Code:
const text = "hello";
wordCounter(text);
Expected Output: 1

Test: "It should return 2 if a passage has two words."
Code:
const text = "hello there";
wordCounter(text);
Expected Output: 2

Test: "It should return 0 for an empty string."
Code: wordCounter("");
Expected Output: 0

Test: "It should return 0 for a string that is only space
s."
Code: wordCounter("            ");
Expected Output: 0

Test: "It should not count numbers as words."
Code: wordCounter("hi there 77 19");
Expected Output: 2


Describe: numberOfOccurrencesInText()

Test: "It should return 0 occurrences of a word for an empt
y string."
Code:
const text = "";
const word = "red";
numberOfOccurrencesInText(word, text);
Expected Output: 0

Test: "It should return 1 occurrence of a word when the wor
d and the text are the same."
Code:
```

```
const text = "red";
const word = "red";
numberOfOccurrencesInText(word, text);
Expected Output: 1

Test: "It should return 0 occurrences of a word when the wo
rd and the text are different."
Code:
const text = "red";
const word = "blue";
numberOfOccurrencesInText(word, text);
Expected Output: 0

Test: "It should return the number of occurrences of a wor
d."
Code:
const text = "red blue red red red green";
const word = "red";
numberOfOccurrencesInText(word, text);
Expected Output: 4

Test: "It should return a word match regardless of case."
Code:
const text = "red RED Red green Green GREEN";
const word = "Red";
numberOfOccurrencesInText(word, text);
Expected Output: 3

Test: "It should return a word match regardless of punctuat
ion."
Code:
const text = "Red! Red. I like red, green, and yellow.";
const word = "Red";
numberOfOccurrencesInText(word, text);
Expected Output: 3

Test: "If an empty string is passed in as a word, it should
return 0."
Code:
const word = "";
const text = "red RED Red!";
```

```
numberOfOccurrencesInText(word, text);
Expected Output: 0
```

Previous (/introduction-to-programming/arrays-and-looping/separation-of-logic-adding-a-ui-to-text-analyzer)
Next (/introduction-to-programming/arrays-and-looping/separation-of-concerns-in-text-analyzer-boldpassage-ui-function)
Lesson 29 of 50

Last updated February 28, 2023

disable dark mode

Epicodus (http://www.epicodus.com)

Lesson Tuesday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Separation of Concerns in Text Analyzer: boldPassage() UI Function

Text Cheat sheet

Terminology

Separation of concerns: A key programming design pattern that dictates that each function should only be responsible for doing one thing, and not know about anything else in the application. In this context, a **concern** is a responsibility. So when we apply separation of concerns to our code, we're separating the functionality of our webpage into multiple different functions, each of which has a single responsibility. For instance, one function might be 'concerned' about one thing (adding two numbers together) while another function might be 'concerned' with returning those numbers to the user.

Using an `index` with `Array.prototype.forEach()`

We can pass in an `index` as the second parameter of the callback we pass into `Array.prototype.forEach()`. This allows us to get the index of the current iteration of the loop. The index always starts at 0. Here's an example:

```
const string = "I like cats!";
string.split(" ").forEach(function(element, index) {
  console.log(element, index);
});
```

Tests and Code for `boldPassage()`

Remember that we'll be writing tests for business logic only. The following tests for the user interface function `boldPassage()` is for TDD demonstration purposes only.

```
Describe: boldPassage()

Test: "It should return null if no word or text is entere
d."
Code:
const text = "";
const word = "";
boldPassage(word, text);
Expected Output: null

Test: "It should return a non-matching word in a p tag."
Code:
const word = "hello";
const text = "yo";
boldPassage(word, text);
Expected Output: <p>yo</p>

Test: "It should return a matching word in a strong tag."
Code:
const word = "hello";
const text = "hello";
boldPassage(word, text);
Expected Output: <p><strong>hello</strong></p>

Test: "It should wrap words that match in strong tags but n
ot words that don't."
Code:
const word = "hello";
const text = "hello there";
boldPassage(word, text);
Expected Output: <p><strong>hello</strong> there</p>
```

```
function boldPassage(word, text) {
  if ((text.trim().length === 0) || (word.trim().length ===
0)) {
    return null;
  }
  const p = document.createElement("p");
  let textArray = text.split(" ");
  textArray.forEach(function(element, index) {
    if (word === element) {
      const bold = document.createElement("strong");
      bold.append(element);
      p.append(bold);
    } else {
      p.append(element);
    }
    if (index !== (textArray.length - 1)) {
      p.append(" ");
    }
  });
  return p;
}
```

Previous (/introduction-to-programming/arrays-and-looping/separation-of-logic-fixing-a-bug-in-text-analyzer)
Next (/introduction-to-programming/arrays-and-looping/drying-code-and-completing-the-text-analyzer-ui)

Lesson 30 of 50
Last updated February 28, 2023

disable dark mode

Epicodus (http://www.epicodus.com)

Lesson Tuesday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / DRYing Code and Completing the Text Analyzer UI

Text Cheat sheet

Terminology

Don't Repeat Yourself: Known as **DRY** for short. This is another essential programming concept. We should avoid repeating code where possible. There are a lot of good reasons not to repeat yourself:

- It results in unnecessary repeated code.
- It's harder to read and reason about because there's extra repeated code to deal with and read.
- If the code breaks or it needs to be updated, we have to change it in multiple places, not just one.

Helper/Utility function: A function, often small, with reusable code. We can use helper functions to DRY up our code by removing repeated code, moving it into the helper function, and then calling that helper function wherever that code is needed. Much less repetition!

Code

HTML

```
<!DOCTYPE html>
<html lang="en-US">
<head>
  <title>Text Analyzer</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/
dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha
384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kad
q2F9CUG65" crossorigin="anonymous">
  <link href="css/styles.css" rel="stylesheet" type="text/c
ss">
  <script src="js/scripts.js"></script>
</head>
<body>
  <div class="container">
    <h2>Text Analyzer</h2>
    <form id="word-counter">
      <div class="form-group">
        <p>Input a text passage to get a total word count:
</p>
        <textarea id="text-passage" name="text-passage" cla
ss="form-control"></textarea>
        <p>Optionally enter a word to count the number of t
imes it occurs in the passage:</p>
        <input type="text" id="word" name="word" class="for
m-control">
        <br />
        <button type="submit" class="btn btn-success">Submi
t Survey</button>
      </div>
    </form>
    <p>Total Word Count: <span id="total-count"></span></p>
    <br />
    <p>Selected Word Count: <span id="selected-count"></spa
n></p>
    <div id="bolded-passage">

    </div>
  </div>
```

```
</body>
</html>
```

JS

```
// Utility Logic

function isEmpty(testString) {
  return (testString.trim().length === 0);
}

// Business Logic

function wordCounter(text) {
  if (isEmpty(text)) {
    return 0;
  }
  let wordCount = 0;
  const textArray = text.split(" ");
  textArray.forEach(function(element) {
    if (!Number(element)) {
      wordCount++;
    }
  });
  return wordCount;
}

function numberOfOccurrencesInText(word, text) {
  if (isEmpty(word)) {
    return 0;
  }
  const textArray = text.split(" ");
  let wordCount = 0;
  textArray.forEach(function(element) {
    if (element.toLowerCase().includes(word.toLowerCase()))
{
      wordCount++;
    }
  });
  return wordCount;
}

// UI Logic
```

```
function boldPassage(word, text) {
  if (isEmpty(word) || isEmpty(text)) {
    return null;
  }
  const p = document.createElement("p");
  let textArray = text.split(" ");
  textArray.forEach(function(element, index) {
    if (word === element) {
      const bold = document.createElement("strong");
      bold.append(element);
      p.append(bold);
    } else {
      p.append(element);
    }
    if (index !== (textArray.length - 1)) {
      p.append(" ");
    }
  });
  return p;
}

function handleFormSubmission() {
  event.preventDefault();
  const passage = document.getElementById("text-passage").v
alue;
  const word = document.getElementById("word").value;
  const wordCount = wordCounter(passage);
  const occurrencesOfWord = numberOfOccurrencesInText(word,
passage);
  document.getElementById("total-count").innerText = wordCo
unt;
  document.getElementById("selected-count").innerText = occ
urrencesOfWord;
  let boldedPassage = boldPassage(word, passage);
  if (boldedPassage) {
    document.querySelector("div#bolded-passage").append(bol
dedPassage);
  } else {
    document.querySelector("div#bolded-passage").innerText
= null;
  }
}
```

```
window.addEventListener("load", function() {
  document.querySelector("form#word-counter").addEventListener("submit", handleFormSubmission);
});
```

Previous (/introduction-to-programming/arrays-and-looping/separation-of-concerns-in-text-analyzer-boldpassage-ui-function)
Next (/introduction-to-programming/arrays-and-looping/practice-using-tdd-with-text-analyzer)

Lesson 31 of 50
Last updated March 24, 2023

disable dark mode

Epicodus (http://www.epicodus.com)

Lesson Wednesday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Rewriting Git History with Rebase

Text Cheat sheet

Terminology

- **Rebase**: The act of moving a branch to a new base commit (essentially just moving a branch from one commit to another).
- **Squashing**: The act of combining multiple commits together into one.

Commands

- `$ git log --oneline`: Returns a repository's log of commits, each condensed to a single line.
- `$ git commit --amend -m "updated commit message here.`: Modifies a repository's most recent commit message with the new message provided.
- `$ git rebase -i`: Changes commit messages and combines multiple commits by "squashing" them together:
    - rewrite commit messages by changing `pick` to `reword`

- squash one or more commit messages by changing `pick` to `squash`

Examples

See a project's commit history with the `git log` command. Add the `--oneline` flag to display the history in an easier to read format:

```
$ git log --oneline
d7e33de something css related
32ccc0b do something with html
9db82b6 update readme
79e9de2 add readme
827ad58 add initial files
```

Amend the most recent Git commit:

```
$ git commit --amend -m "add styling to main page"
```

Previous (/introduction-to-programming/arrays-and-looping/practice-using-tdd-with-text-analyzer)
Next (/introduction-to-programming/arrays-and-looping/practice-rewriting-git-history-with-rebase)
Lesson 33 of 50
Last updated March 24, 2023

disable dark mode

Epicodus (http://www.epicodus.com)

Lesson Wednesday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Looping with for

Text Cheat sheet

Terminology

- **For loop**: The "traditional" way to write loops in JavaScript. Creates a loop without needing to call a method on an array.
- **Initialization parameter**: Initializes a variable with the denoted value, which usually is a number. This only happens once, when the loop is first triggered. It needs to be initialized with `let`. Usually this variable is called `index` or `i`, but it could be called anything, depending on what the loop is used for and what's most descriptive.
- **Condition parameter**: Tells the loop under what conditions to *continue* running the loop. As long as the condition is true, the loop will continue running.
- **Final expression parameter**: Usually changes the initial value in some way, often by incrementing or decrementing it.

Examples

Example `for` loop:

```
sfor (let index = 1; index <= 3; index += 1) {
  console.log(index);
}
```

In the code above...

- `let index = 1;` is the **initialization parameter**.
- `index <=3;` is the **condition**.
- `index += 1` is the **final expression**.

Example of using a `for` loop with an array:

```
const languages = ['HTML', 'CSS', 'Javascript'];
for (let index = 0; index < languages.length; index += 1) {
  console.log('I love ' + languages[index] + '!');
}
```

Previous (/introduction-to-programming/arrays-and-looping/printing-an-array-to-a-webpage)
Next (/introduction-to-programming/arrays-and-looping/practice-looping-with-for)

Lesson 36 of 50
Last updated March 24, 2023

disable dark mode

Epicodus (http://www.epicodus.com)

Lesson Wednesday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / When to use for

Text Cheat sheet

When to Use `for` Loops

- **When we need to break out of the loop.** For instance, we might want to break out of a loop when we verify that an array contains a result. Instead of looping through the rest of the array, we can use the `break;` or `return` keyword to end the loop.
- **When we're not looping through an array.** `Array.prototype.forEach()` only works with arrays. When we are using other data types, `for` loops can be more convenient.

Previous (/introduction-to-programming/arrays-and-looping/practice-looping-with-for)
Next (/introduction-to-programming/arrays-and-looping/for-loops-with-text-analyzer)

Lesson 38 of 50
Last updated February 28, 2023

disable dark mode

Epicodus (http://www.epicodus.com)

Lesson Wednesday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / for Loops with Text Analyzer

Text Cheat sheet

JavaScript `arguments` Object

- JavaScript provides an `arguments` object in all functions. It is an "array-like" object that stores *all* of the arguments passed into a function. We can use bracket notation to access arguments individually.

Here's an example:

```
function isEmpty() {
  for (let i=0; i < arguments.length; i++) {
    if (arguments[i].trim().length === 0) {
      return true;
    }
  }
  return false;
}
```

We use a `for` loop and bracket notation to access each argument in the `arguments` object because `arguments` is not an array, which means we can't use `Array.prototype.forEach()` to loop through its arguments.

Previous (/introduction-to-programming/arrays-and-looping/when-to-use-for)
Next (/introduction-to-programming/arrays-and-looping/practice-pig-latin)

Lesson 39 of 50
Last updated February 28, 2023

disable dark mode

Epicodus (http://www.epicodus.com)

Lesson Wednesday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Further Exploration: Introduction to Regular Expressions

Text Cheat sheet

Terminology

Regular expression: Also known as a **regex**, a regular expression is a set of characters we can use to find patterns in a string. The set of characters is enclosed in `/ /` and may include flags after the second slash.

Methods That Use Regular Expressions

- `String.prototype.replace()` : Takes two arguments — the first is a regular expression, the second is what the pattern should be replaced by.
- `String.prototype.match()` : Takes a regular expression as an argument and then returns an array with all matches.
- `RegExp.prototype.test()` : Takes a string as an argument — the regular expression is the receiver — and returns a boolean if the string contains the pattern.

Regex Characters

- `\d` : Numbers
- `\D` : Not numbers
- `\w` : Matches any alphanumeric character (including underscores) — so numbers and letters
- `\W` : Matches any character that's not a number, letter or underscore
- `\s` : Matches a whitespace character
- `\S` : Matches any non-whitespace character
- `.` : Any single character (wildcard)
- `^` : *Not* this pattern

Regex Flags

Regex flags come after the second slash in a regular expression. For instance: `/cat/gi`.

- `g` is the global flag. Without this flag, regular expressions usually just find the first matching pattern in the string. With this flag, the regex will find *all* matching patterns in the string.
- `i` is the case insensitivity flag. When it's added, the regular expression will ignore case sensitivity.

Regex Groups and Ranges

- `[ ]` denotes that all characters inside the brackets should be considered a matching pattern. For instance, the pattern `/[aieou]/` will match any vowels in a string.
- `-` denotes a range of characters. For instance, the pattern `/[0-9]/` denotes all numerical digits. `[A-Z]` and `[a-z]` are other common ranges.

Regex Quantifiers

- `+` : Match the preceding character one or more times
- `*` : Match the preceding character zero or more times
- `?` : Match the preceding character zero or one times

- `{x}` : Match the pattern `x` number of times
- `{x,}` : Match the pattern at least `x` times
- `{x,y}` : Match the pattern at least `x` but no more than `y` times

Other Helpful Regex Symbols

- `|` : Represents or. For example, `/cat|dog/` states match either `"cat"` *or* `"dog"`
- `\b` : Denotes a pattern boundary. Can be used at beginning or end of a pattern. For example, `/\bcat\b/` represents an *exact* match with "cat" — and doesn't match with "cathedral".

Documentation

- Regular expression syntax cheatsheet. (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions/Cheatsheet)
- MDN guide to regular expressions. (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions)
- Regex Crossword (https://regexcrossword.com/) is a fun site for learning about regular expressions.
- Finally, it's very common to use regex generators that make it easier to get the regex we need to get the job done. A quick Google search will reveal many out there! Here's just a few to optionally check out:
  - https://regexr.com/ (https://regexr.com/)
  - https://regex-generator.olafneumann.org/ (https://regex-generator.olafneumann.org/)
  - https://regex101.com/ (https://regex101.com/)

Previous (/introduction-to-programming/arrays-and-looping/practice-pig-latin)
Next (/introduction-to-programming/arrays-and-looping/further-exploration-regular-expressions-with-text-analyzer)

Lesson 41 of 50
Last updated March 24, 2023

disable dark mode

Epicodus (http://www.epicodus.com)

Lesson Wednesday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Further Exploration: Regular Expressions with Text Analyzer

Text Cheat sheet

Code

We can **create**s a `RegExp` object like this:

```
> const word = "red";
> const regex = new RegExp(word, "gi");
```

This is the best way to pass a variable into a regular expression.

We can **use** the `RegExp` object like this:

```
> const text = "RED red red! Green GREEN green.";
> const word = "red";
> const regex = new RegExp(word, "gi");
> text.match(regex);
["RED", "red", "red"]
```

`numberOfOccurrencesInText()` with regex:

```
function numberOfOccurrencesInText(word, text) {
  if (isEmpty(word)) {
    return 0;
  }
  const regex = new RegExp(word, "gi");
  return text.match(regex).length;
}
```

Previous (/introduction-to-programming/arrays-and-looping/further-exploration-introduction-to-regular-expressions)
Next (/introduction-to-programming/arrays-and-looping/array-mapping)

Lesson 42 of 50
Last updated February 28, 2023

disable dark mode

Epicodus (http://www.epicodus.com)

Lesson Thursday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Array Mapping

Text Cheat sheet

`Array.prototype.map()`

This is perhaps the most powerful looping method in JavaScript because it loops through a collection *and* also transforms it.

Here's an example:

```
const array = [0,1,2,3,4,5];
const doubledArray = array.map(function(element) {
  return element * 2;
});
doubledArray;
(6) [1, 2, 4, 6, 8, 10]
```

- Like `Array.prototype.forEach()`, `Array.prototype.map()` takes a function as an argument, which we call a callback function.
- With `Array.prototype.map()`, we *must* use a `return` statement (unlike `Array.prototype.forEach()`).

- Each time we iterate with `Array.prototype.map()`, we are specifying how the element should be transformed. In the example above, we specify that we should `return element * 2`.
- `Array.prototype.map()` returns a transformed array. We can't use it to return another data type such as a number or string.

Documentation on MDN

For more information, check out the Mozilla documentation on `Array.prototype.map()`. (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map)

Previous (/introduction-to-programming/arrays-and-looping/further-exploration-regular-expressions-with-text-analyzer)
Next (/introduction-to-programming/arrays-and-looping/practice-array-mapping)

Lesson 43 of 50
Last updated February 28, 2023

disable dark mode

Epicodus (http://www.epicodus.com)

Lesson Thursday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Further Exploration: Looping with for...of

Text Cheat sheet

Terminology

- **`for...of`** : A technique we can use to loop through arrays, strings, and other types of iterable objects.

Examples

Example Using an Array

```
const array = [0,1,2,3,4,5];
let doubledArray = [];
for (const element of array) {
  doubledArray.push(element * 2);
}
doubledArray;
(6) [0, 2, 4, 6, 8, 10]
```

Example Using a String

```
const consonantString = "bdfmxtgl"
let vowelizedString = "";
for (const letter of consonantString) {
  vowelizedString = vowelizedString.concat(letter + "a");
}
vowelizedString;
"badafamaxatagala"
```

Previous (/introduction-to-programming/arrays-and-looping/practice-credit-card-validator-roman-numerals-or-cryptosquare)
Next (/introduction-to-programming/arrays-and-looping/further-exploration-while-loops)

Lesson 46 of 50
Last updated March 24, 2023

disable dark mode

(http://www.epicodus.com)

Lesson Thursday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Further Exploration: While Loops

Text Cheat sheet

Terminology

- `while` : A type of loop that runs until a condition is met.

Examples

Example of `while` Loop

```
let number = 10;
while (number > 0) {
  console.log(number);
  number --;
}
console.log("Blast off!")
```

Example of `do...while` Loop

```
let number = 10;
do {
  console.log(number);
    number --;
} while (number > 0);
console.log("Blast off!");
```

Previous (/introduction-to-programming/arrays-and-looping/further-exploration-looping-with-for-of)
Next (/introduction-to-programming/arrays-and-looping/optional-review-which-loop-should-i-use)
Lesson 47 of 50
Last updated March 24, 2023

disable dark mode

Epicodus (http://www.epicodus.com)

Lesson Weekend

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Bracket Notation

Text Cheat sheet

Terminology and Examples

Index: The index of an element in an array is its numerical position. The first element has an index of 0.

OBOE: An off-by-one error. Watch out for these!

bracket notation: we can access array elements using square brackets `[]` and entering in the index:

```
> const letters = ['a', 'b', 'c'];
> letters[0];
'a'
```

Start counting elements at 0.

the `length` property: you can check the length of an array by accessing its `length` property:

```
> letters.length
3
```

You can get elements from the end of an array like this:

```
> letters[letters.length - 1]
'c'
```

Previous (/introduction-to-programming/arrays-and-looping/introduction-to-arrays)
Next (/introduction-to-programming/arrays-and-looping/array-methods)

Lesson 6 of 50
Last updated March 24, 2023

disable dark mode

(http://www.epicodus.com)

Lesson Weekend

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Array Methods

Text Cheat sheet

Terminology

- **Destructive** methods modify the receiver (the thing they are called on).
- **Non-destructive** methods don't modify the receiver, they create a brand new array. For non-destructive methods, you'll need to store the return value of the method in a variable.

Methods

- `Array.prototype.push()` : Push elements to the end of an array.
- `Array.prototype.concat()` : Concatenate elements to the end of an array. Similar to `Array.prototype.push()` except it doesn't modify the original array.
- `Array.prototype.unshift()` : Add an element to the beginning of an array.
- `Array.prototype.shift()` : Remove an element from the beginning of an array.

- `Array.prototype.pop()` : Remove an element from the end of an array.
- `Array.prototype.join()` : Turn an array into a string. You can pass an optional separator in as an argument. `""` is a common separator.
- `Array.prototype.slice()` : Slice elements from the beginning and (optionally) the end of an array.

Modify Elements in an Array with Bracket Notation

```
> let array = [1,2,3];
> array[0] = "We just modified the array at position zer
o.";
> array;
["We just modified the array at position zero.",2,3]
```

Documentation on MDN

See the list of array methods in the left-hand pane of the Mozilla array documentation (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array).

Previous (/introduction-to-programming/arrays-and-looping/bracket-notation)
Next (/introduction-to-programming/arrays-and-looping/comparing-and-cloning-arrays)

Lesson 7 of 50
Last updated March 24, 2023

disable dark mode

(http://www.epicodus.com)

Lesson Weekend

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Comparing and Cloning Arrays

Text Cheat sheet

Terminology

- **Pointer**: A reference to an object in memory but not the object itself; for example, a variable that is set to an array does not contain the array itself but rather a pointer to the saved array.
- **Edge case**: An edge case in computer programming is a possible outcome of an operation that leads to unexpected or inconsistent results.

Tips

- No two arrays are the same even if they have the exact same contents inside!
- Arrays cannot be compared with the `===` operator. However, they may be transformed into strings with `.toString()`, and those strings may be compared with `===`.
- Arrays cannot be cloned by setting a new variable name to the original array (i.e.: `let cloneArray = originalArray;`). This will only create a pointer to the original array.

Examples

To properly clone array (i.e.: not simply create a pointer to existing array):

```
const cloneArray = originalArray.slice();
```

To compare arrays by transforming them into strings:

```
const a = [1,2,3];
const b = [1,2,3];

a.toString() === b.toString();
```

Additional Resources

For more details on how the `Array.prototype.slice()` method works, check out MDN's JavaScript documentation (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/slice).

Previous (/introduction-to-programming/arrays-and-looping/array-methods)
Next (/introduction-to-programming/arrays-and-looping/document-query-methods-that-return-collections)
Lesson 8 of 50
Last updated March 24, 2023

disable dark mode

(http://www.epicodus.com)

Lesson Weekend

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Document Query Methods that Return Collections

Text Cheat sheet

document.querySelectorAll()

Just like `document.querySelector()`, we can input any valid CSS selector into `document.querySelectorAll()` (https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelectorAll) to get HTML element objects returned to us.

```
> document.querySelectorAll("span");
NodeList(9) [span#person1a, span#person2a, span#animal, spa
n#exclamation, span#person1b, span#verb, span#person1c, spa
n#person2b, span#noun]
```

```
> document.querySelectorAll("span:nth-child(4)");
NodeList [span#exclamation]
```

If we want to get one item from the `NodeList` collection, we'll use bracket notation, passing in the index of the element we want (starting from 0). Just like with arrays, if we pass in 0, we'll get the first element returned:

```
> document.querySelectorAll("span:nth-child(odd)")[0];
<span id="person1a">_________</span>
```

`document.getElementsByClassName()`

The `document.getElementsByClassName()` (https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementsByClassName) method gets all elements that have the same value for their `class` attribute. Since the Mad Libs project doesn't use any classes, the return is an empty collection.

```
> document.getElementsByClassName('x');
HTMLCollection []length: 0[[Prototype]]: HTMLCollection
```

If we want to get an element from the `HTMLCollection`, we'll also use bracket notation and pass in the index (starting at 0) of the element that we want to get.

```
> document.getElementsByClassName('x')[0];
undefined
```

Since our collection is empty, we get `undefined` returned to us.

`document.getElementsByTagName()`

The `document.getElementsByTagName()` (https://developer.mozilla.org/en-US/docs/Web/API/Element/getElementsByTagName) method gets all elements by their tag name. The same tag name that's returned from the `Element.tagName` (https://developer.mozilla.org/en-US/docs/Web/API/Element/tagName) property.

When we use `document.getElementsByTagName()` , we don't have to capitalize the tag name, even though that's how tag names are returned to us from accessing the `Element.tagName` property.

```
> document.getElementsByTagName("h1");
HTMLCollection(2) [h1, h1]
```

To get a single element from the collection, we also use bracket notation. Here, we're getting the second element:

```
> const secondH1 = document.getElementsByTagName("h1")[1];
> secondH1;
<h1>A fantastical adventure</h1>
> Object.prototype.toString.call(secondH1);
"[object HTMLHeadingElement]"
```

`document.getElementsByName()`

The method `document.getElementsByName()` (https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementsByName) will get all elements that have the same value for their `name` attribute. Here's an example:

```
> document.getElementsByName("person1Input");
NodeList [input#person1Input]
```

This method would likely be more useful for radio buttons or checkboxes (as we'll learn), since all inputs of those types must share the same `name` attribute for them to function properly.

Just like in previous examples, if we want to get a single element from the list, we'll use bracket notation.

```
> document.getElementsByName("person1Input")[0];
<input id="person1Input" type="text" name="person1Input">
```

Objects that Look and Act like Arrays

Objects can be structured to look and act like an array. An array-like object:

- Has a `length` property.
- Has properties indexed from zero.
- Do not have access to JavaScript Array methods.
- Uses bracket notation (passing in an index number) to access individual properties.
- Have names; since they are an object, they can be named to be a specific type of object.
- We're working with two array-like objects called `NodeList` and `HTMLCollection`, both of which are Web APIs.

These exist in Web APIs as well as JS proper, and we'll encounter more of these. At this time, we don't need to understand why these exist, or anything deeper about how they are set up. We just need to know how to use them!

Turning `NodeList` and `HTMLCollection` Objects into Arrays

```
> const headingCollection = document.getElementsByTagName
("h1");
> headingCollection;
HTMLCollection(2) [h1, h1]
> const headingArray = Array.from(headingCollection);
> headingArray;
(2) [h1, h1]
> Object.prototype.toString.call(headingArray);
'[object Array]'
```

Notably the `Array.from()` method is called on the array object type. We know it's the array object type, because `Array` is capitalized and we don't use `prototype` in the method's name. We'll revisit this type of method in an upcoming lesson.

Take note that there are some limitations for using `Array.from()` in older browsers, but that's true for a lot of JavaScript! As always, if you run into any issues, visit documentation like MDN.

MDN Documentation Links

The `NodeList` and `HTMLCollection` objects are two of the many objects that make up the DOM. We won't explore them in depth like we did in the last course section with `Element`, `HTMLElement`, and other objects that also are a part of the DOM.

Here are direct links to the objects and methods we learned about in this lesson:

- `NodeList` (https://developer.mozilla.org/en-US/docs/Web/API/NodeList)
- `HTMLCollection` (https://developer.mozilla.org/en-US/docs/Web/API/HTMLCollection)

- `document` object methods (https://developer.mozilla.org/en-US/docs/Web/API/document)
- `Array.from()` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/from)
- Reference for CSS Selectors (https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors))

Previous (/introduction-to-programming/arrays-and-looping/comparing-and-cloning-arrays)
Next (/introduction-to-programming/arrays-and-looping/adding-and-removing-html-elements)

Lesson 9 of 50
Last updated February 28, 2023

disable dark mode

(http://www.epicodus.com)