

Responsive Web Design Certification

Basic HTML and HTML5

1. Say Hello to HTML Elements

Description

Welcome to freeCodeCamp's HTML coding challenges. These will walk you through web development step-by-step. First, you'll start by building a simple web page using HTML. You can edit `code` in your `code editor`, which is embedded into this web page. Do you see the code in your code editor that says `<h1>Hello</h1>`? That's an HTML element. Most HTML elements have an `opening tag` and a `closing tag`. Opening tags look like this: `<h1>`. Closing tags look like this: `</h1>`. The only difference between opening and closing tags is the forward slash after the opening bracket of a closing tag. Each challenge has tests you can run at any time by clicking the "Run tests" button. When you pass all tests, you'll be prompted to submit your solution and go to the next coding challenge.

Instructions

To pass the test on this challenge, change your `h1` element's text to say "Hello World".

Challenge Seed

```
<h1>Hello</h1>
```

Solution

```
<h1>Hello World</h1>
```

2. Headline with the h2 Element

Description

Over the next few lessons, we'll build an HTML5 cat photo web app piece-by-piece. The `h2` element you will be adding in this step will add a level two heading to the web page. This element tells the browser about the structure of your website. `h1` elements are often used for main headings, while `h2` elements are generally used for subheadings. There are also `h3`, `h4`, `h5` and `h6` elements to indicate different levels of subheadings.

Instructions

Add an `h2` tag that says "CatPhotoApp" to create a second HTML element below your "Hello World" `h1` element.

Challenge Seed

```
<h1>Hello World</h1>
```

Solution

```
<h1>Hello World</h1>
<h2>CatPhotoApp</h2>
```

3. Inform with the Paragraph Element

Description

`p` elements are the preferred element for paragraph text on websites. `p` is short for "paragraph". You can create a paragraph element like this: `<p>I'm a p tag!</p>`

Instructions

Create a `p` element below your `h2` element, and give it the text "Hello Paragraph". Note: As a convention, all HTML tags are written in lowercase, for example `<p></p>` and not `<P></P>`.

Challenge Seed

```
<h1>Hello World</h1>
<h2>CatPhotoApp</h2>
```

Solution

```
// solution required
```

4. Fill in the Blank with Placeholder Text

Description

Web developers traditionally use `lorem ipsum` text as placeholder text. The 'lorem ipsum' text is randomly scraped from a famous passage by Cicero of Ancient Rome. Lorem ipsum text has been used as placeholder text by typesetters since the 16th century, and this tradition continues on the web. Well, 5 centuries is long enough. Since we're building a CatPhotoApp, let's use something called `kitty ipsum` text.

Instructions

Replace the text inside your `p` element with the first few words of this kitty ipsum text: `Kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff.`

Challenge Seed

```
<h1>Hello World</h1>
<h2>CatPhotoApp</h2>
<p>Hello Paragraph</p>
```

Solution

```
// solution required
```

5. Uncomment HTML

Description

Commenting is a way that you can leave comments for other developers within your code without affecting the resulting output that is displayed to the end user. Commenting is also a convenient way to make code inactive without having to delete it entirely. Comments in HTML starts with `<!--`, and ends with a `-->`

Instructions

Uncomment your `h1`, `h2` and `p` elements.

Challenge Seed

```
<!--  
<h1>Hello World</h1>  
  
<h2>CatPhotoApp</h2>  
  
<p>Kitty ipsum dolor sit amet, shed everywhere stretch attack your ankles chase the  
red dot, hairball run catnip eat the grass sniff.</p>  
-->
```

Solution

```
<h1>Hello World</h1>  
  
<h2>CatPhotoApp</h2>  
  
<p>Kitty ipsum dolor sit amet, shed everywhere stretch attack your ankles chase the  
red dot, hairball run catnip eat the grass sniff.</p>
```

6. Comment out HTML

Description

Remember that in order to start a comment, you need to use `<!--` and to end a comment, you need to use `-->`. Here you'll need to end the comment before your `h2` element begins.

Instructions

Comment out your `h1` element and your `p` element, but not your `h2` element.

Challenge Seed

```
<!--  
<h1>Hello World</h1>  
  
<h2>CatPhotoApp</h2>  
  
<p>Kitty ipsum dolor sit amet, shed everywhere stretch attack your ankles chase the  
red dot, hairball run catnip eat the grass sniff.</p>  
-->
```

Solution

```
<!--<h1>Hello World</h1>-->  
<h2>CatPhotoApp</h2>
```

```
<!--<p>Kitty ipsum dolor sit amet, shed everywhere stretching attack your ankles chase  
the red dot, hairball run catnip eat the grass sniff.</p> -->
```

7. Delete HTML Elements

Description

Our phone doesn't have much vertical space. Let's remove the unnecessary elements so we can start building our CatPhotoApp.

Instructions

Delete your `h1` element so we can simplify our view.

Challenge Seed

```
<h1>Hello World</h1>  
  
<h2>CatPhotoApp</h2>  
  
<p>Kitty ipsum dolor sit amet, shed everywhere stretching attack your ankles chase the  
red dot, hairball run catnip eat the grass sniff.</p>
```

Solution

```
var code = "<h2>CatPhotoApp</h2><p>Kitty ipsum dolor sit amet, shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff.</p>"
```

8. Introduction to HTML5 Elements

Description

HTML5 introduces more descriptive HTML tags. These include `header`, `footer`, `nav`, `video`, `article`, `section` and others. These tags make your HTML easier to read, and also help with Search Engine Optimization (SEO) and accessibility. The `main` HTML5 tag helps search engines and other developers find the main content of your page.

Note

Many of the new HTML5 tags and their benefits are covered in the Applied Accessibility section.

Instructions

Create a second `p` element after the existing `p` element with the following kitty ipsum text: Purr jump eat the grass rip the couch scratched sunbathe, shed everywhere rip the couch sleep in the sink fluffy fur catnip scratched. Wrap the paragraphs with an opening and closing `main` tag.

Challenge Seed

```
<h2>CatPhotoApp</h2>  
  
<p>Kitty ipsum dolor sit amet, shed everywhere stretching attack your ankles chase the  
red dot, hairball run catnip eat the grass sniff.</p>
```

Solution

```
// solution required
```

9. Add Images to Your Website

Description

You can add images to your website by using the `img` element, and point to a specific image's URL using the `src` attribute. An example of this would be: `` Note that `img` elements are self-closing. All `img` elements **must** have an `alt` attribute. The text inside an `alt` attribute is used for screen readers to improve accessibility and is displayed if the image fails to load. Note: If the image is purely decorative, using an empty `alt` attribute is a best practice. Ideally the `alt` attribute should not contain special characters unless needed. Let's add an `alt` attribute to our `img` example above: ``

Instructions

Let's try to add an image to our website: Insert an `img` tag, after the `main` element. Now set the `src` attribute so that it points to this url: `https://bit.ly/fcc-relaxing-cat` Finally don't forget to give your image an `alt` text.

Challenge Seed

```
<h2>CatPhotoApp</h2>
<main>

  <p>Kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff.</p>
  <p>Purr jump eat the grass rip the couch scratched sunbathe, shed everywhere rip the couch sleep in the sink fluffy fur catnip scratched.</p>
</main>
```

Solution

```
<h2>CatPhotoApp</h2>
<main>
  <a href="#"></a>
  <p>Kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff.</p>
  <p>Purr jump eat the grass rip the couch scratched sunbathe, shed everywhere rip the couch sleep in the sink fluffy fur catnip scratched.</p>
</main>
```

10. Link to External Pages with Anchor Elements

Description

You can use `anchor` elements to link to content outside of your web page. `anchor` elements need a destination web address called an `href` attribute. They also need anchor text. Here's an example: `this links to freecodecamp.org` Then your browser will display the text "this links to freecodecamp.org" as a link you can click. And that link will take you to the web address `https://www.freecodecamp.org`.

Instructions

Create an `a` element that links to `http://freecatphotoapp.com` and has "cat photos" as its anchor text .

Challenge Seed

```
<h2>CatPhotoApp</h2>
<main>

  <p>Kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff.</p>
  <p>Purr jump eat the grass rip the couch scratched sunbathe, shed everywhere rip the couch sleep in the sink fluffy fur catnip scratched.</p>
</main>
```

Solution

```
// solution required
```

11. Link to Internal Sections of a Page with Anchor Elements

Description

anchor elements can also be used to create internal links to jump to different sections within a webpage. To create an internal link, you assign a link's href attribute to a hash symbol # plus the value of the id attribute for the element that you want to internally link to, usually further down the page. You then need to add the same id attribute to the element you are linking to. An id is an attribute that uniquely describes an element. Below is an example of an internal anchor link and its target element:

```
<a href="#contacts-header">Contacts</a>
...
<h2 id="contacts-header">Contacts</h2>
```

When users click the Contacts link, they'll be taken to the section of the webpage with the **Contacts** header element.

Instructions

Change your external link to an internal link by changing the href attribute to "#footer" and the text from "cat photos" to "Jump to Bottom". Remove the target="_blank" attribute from the anchor tag since this causes the linked document to open in a new window tab. Then add an id attribute with a value of "footer" to the <footer> element at the bottom of the page.

Challenge Seed

```
<h2>CatPhotoApp</h2>
<main>

  <a href="http://freecatphotoapp.com" target="_blank">cat photos</a>

  <p>Kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff. Purr jump eat the grass rip the couch scratched sunbathe, shed everywhere rip the couch sleep in the sink fluffy fur catnip scratched. Kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff.</p>
  <p>Purr jump eat the grass rip the couch scratched sunbathe, shed everywhere rip the couch sleep in the sink fluffy fur catnip scratched. Kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff. Purr jump eat the grass rip the couch scratched sunbathe, shed everywhere rip the couch sleep in the sink fluffy fur
```

```

catnip scratched.</p>
<p>Meowwww loved it, hated it, loved it yet spill litter box, scratch at owner, destroy all furniture, especially couch or lay on arms while you're using the keyboard. Missing until dinner time toy mouse squeak roll over. With tail in the air lounge in doorway. Man running from cops stops to pet cats, goes to jail.</p>
<p>Intently stare at the same spot poop in the plant pot but kitten is playing with dead mouse. Get video posted to internet for chasing red dot leave fur on owners clothes meow to be let out and mesmerizing birds leave fur on owners clothes or favor packaging over toy so purr for no reason. Meow to be let out play time intently sniff hand run outside as soon as door open yet destroy couch.</p>

</main>

<footer>Copyright Cat Photo App</footer>
```

Solution

```

<h2>CatPhotoApp</h2>
<main>

  <a href="#footer">Jump to Bottom</a>

  <p>Kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff. Purr jump eat the grass rip the couch scratched sunbathe, shed everywhere rip the couch sleep in the sink fluffy fur catnip scratched. Kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff.</p>
  <p>Purr jump eat the grass rip the couch scratched sunbathe, shed everywhere rip the couch sleep in the sink fluffy fur catnip scratched. Kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff. Purr jump eat the grass rip the couch scratched sunbathe, shed everywhere rip the couch sleep in the sink fluffy fur catnip scratched.</p>
  <p>Meowwww loved it, hated it, loved it yet spill litter box, scratch at owner, destroy all furniture, especially couch or lay on arms while you're using the keyboard. Missing until dinner time toy mouse squeak roll over. With tail in the air lounge in doorway. Man running from cops stops to pet cats, goes to jail.</p>
  <p>Intently stare at the same spot poop in the plant pot but kitten is playing with dead mouse. Get video posted to internet for chasing red dot leave fur on owners clothes meow to be let out and mesmerizing birds leave fur on owners clothes or favor packaging over toy so purr for no reason. Meow to be let out play time intently sniff hand run outside as soon as door open yet destroy couch.</p>

</main>

<footer id="footer">Copyright Cat Photo App</footer>
```

12. Nest an Anchor Element within a Paragraph

Description

You can nest links within other text elements.

```
<p>
  Here's a <a target="_blank" href="http://freecodecamp.org"> link to freecodecamp.org</a> for you to follow.
</p>
```

Let's break down the example: Normal text is wrapped in the `p` element:

```
<p> Here's a ... for you to follow. </p> Next is the anchor element <a> (which requires a closing tag </a>):<a> ... </a> target is an anchor tag attribute that specifies where to open the link and the value "_blank" specifies to open the link in a new tab href is an anchor tag attribute that contains the URL address of the link:<a href="http://freecodecamp.org"> ... </a> The text, "link to freecodecamp.org", within the anchor element called anchor text , will display a link to click:<a href=" ... ">link to freecodecamp.org</a> The final output of the example will look like this:Here's a link to freecodecamp.org for you to follow.
```

Instructions

Now nest the existing `a` element within a new `p` element (just after the existing `main` element). The new paragraph should have text that says "View more cat photos", where "cat photos" is a link, and the rest of the text is plain text.

Challenge Seed

```
<h2>CatPhotoApp</h2>
<main>

  <a href="http://freecatphotoapp.com" target="_blank">cat photos</a>

  <p>Kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff.</p>
  <p>Purr jump eat the grass rip the couch scratched sunbathe, shed everywhere rip the couch sleep in the sink fluffy fur catnip scratched.</p>
</main>
```

Solution

```
// solution required
```

13. Make Dead Links Using the Hash Symbol

Description

Sometimes you want to add `a` elements to your website before you know where they will link. This is also handy when you're changing the behavior of a link using `JavaScript`, which we'll learn about later.

Instructions

The current value of the `href` attribute is a link that points to "<http://freecatphotoapp.com>". Replace the `href` attribute value with a `#`, also known as a hash symbol, to create a dead link. For example: `href="#"`

Challenge Seed

```
<h2>CatPhotoApp</h2>
<main>

  <p>Click here to view more <a href="http://freecatphotoapp.com" target="_blank">cat photos</a>.</p>

  <p>Kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff.</p>
  <p>Purr jump eat the grass rip the couch scratched sunbathe, shed everywhere rip the couch sleep in the sink fluffy fur catnip scratched.</p>
</main>
```

Solution

```
// solution required
```

14. Turn an Image into a Link

Description

You can make elements into links by nesting them within an `a` element. Nest your image within an `a` element. Here's an example: `` Remember to use `#` as your `a` element's `href` property in order to turn it into a dead link.

Instructions

Place the existing image element within an `anchor` element. Once you've done this, hover over your image with your cursor. Your cursor's normal pointer should become the link clicking pointer. The photo is now a link.

Challenge Seed

```
<h2>CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>

  <p>Kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff.</p>
  <p>Purr jump eat the grass rip the couch scratched sunbathe, shed everywhere rip the couch sleep in the sink fluffy fur catnip scratched.</p>
</main>
```

Solution

```
// solution required
```

15. Create a Bulleted Unordered List

Description

HTML has a special element for creating `unordered lists`, or bullet point style lists. Unordered lists start with an opening `` element, followed by any number of `` elements. Finally, unordered lists close with a ``. For example:

```
<ul>
  <li>milk</li>
  <li>cheese</li>
</ul>
```

would create a bullet point style list of "milk" and "cheese".

Instructions

Remove the last two `p` elements and create an unordered list of three things that cats love at the bottom of the page.

Challenge Seed

```
<h2>CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <p>Kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff.</p>
  <p>Purr jump eat the grass rip the couch scratched sunbathe, shed everywhere rip the couch sleep in the sink fluffy fur catnip scratched.</p>
</main>
```

Solution

```
// solution required
```

16. Create an Ordered List

Description

HTML has another special element for creating ordered lists , or numbered lists. Ordered lists start with an opening `` element, followed by any number of `` elements. Finally, ordered lists close with a ``. For example:

```
<ol>
  <li>Garfield</li>
  <li>Sylvester</li>
</ol>
```

would create a numbered list of "Garfield" and "Sylvester".

Instructions

Create an ordered list of the top 3 things cats hate the most.

Challenge Seed

```
<h2>CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <p>Things cats love:</p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>

</main>
```

Solution

```
// solution required
```

17. Create a Text Field

Description

Now let's create a web form. `input` elements are a convenient way to get input from your user. You can create a text input like this: `<input type="text">`. Note that `input` elements are self-closing.

Instructions

Create an `input` element of type `text` below your lists.

Challenge Seed

```
<h2>CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <p>Things cats love:</p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>

</main>
```

Solution

```
<h2>CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <p>Things cats love:</p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
  <form>
    <input type="text">
  </form>
</main>
```

18. Add Placeholder Text to a Text Field

Description

Placeholder text is what is displayed in your `input` element before your user has inputted anything. You can create placeholder text like so: `<input type="text" placeholder="this is placeholder text">`

Instructions

Set the `placeholder` value of your text `input` to "cat photo URL".

Challenge Seed

```

<h2>CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <p>Things cats love:</p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
  <input type="text">
</main>

```

Solution

```
// solution required
```

19. Create a Form Element

Description

You can build web forms that actually submit data to a server using nothing more than pure HTML. You can do this by specifying an action on your `form` element. For example: `<form action="/url-where-you-want-to-submit-form-data"></form>`

Instructions

Nest your text field inside a `form` element, and add the `action="/submit-cat-photo"` attribute to the form element.

Challenge Seed

```

<h2>CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <p>Things cats love:</p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
  <input type="text" placeholder="cat photo URL">
</main>

```

Solution

```
// solution required
```

20. Add a Submit Button to a Form

Description

Let's add a `submit` button to your form. Clicking this button will send the data from your form to the URL you specified with your form's `action` attribute. Here's an example submit button: `<button type="submit">this button submits the form</button>`

Instructions

Add a button as the last element of your `form` element with a type of `submit`, and "Submit" as its text.

Challenge Seed

```
<h2>CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <p>Things cats love:</p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
  <form action="/submit-cat-photo">
    <input type="text" placeholder="cat photo URL">
  </form>
</main>
```

Solution

```
<h2>CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <p>Things cats love:</p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
  <form action="/submit-cat-photo">
    <input type="text" placeholder="cat photo URL">
    <button type="submit">Submit</button>
  </form>
</main>
```

21. Use HTML5 to Require a Field

Description

You can require specific form fields so that your user will not be able to submit your form until he or she has filled them out. For example, if you wanted to make a text input field required, you can just add the attribute `required` within your `input` element, like this: `<input type="text" required>`

Instructions

Make your text `input` a required field, so that your user can't submit the form without completing this field. Then try to submit the form without inputting any text. See how your HTML5 form notifies you that the field is required?

Challenge Seed

```
<h2>CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <p>Things cats love:</p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
  <form action="/submit-cat-photo">
    <input type="text" placeholder="cat photo URL">
    <button type="submit">Submit</button>
  </form>
</main>
```

Solution

```
// solution required
```

22. Create a Set of Radio Buttons

Description

You can use `radio` button s for questions where you want the user to only give you one answer out of multiple options. Radio buttons are a type of `input`. Each of your radio buttons can be nested within its own `label` element. By wrapping an `input` element inside of a `label` element it will automatically associate the radio button input with the label element surrounding it. All related radio buttons should have the same `name` attribute to create a radio button group. By creating a radio group, selecting any single radio button will automatically deselect the other buttons within the same group ensuring only one answer is provided by the user. Here's an example of a radio button:

```
<label>
  <input type="radio" name="indoor-outdoor">Indoor
</label>
```

It is considered best practice to set a `for` attribute on the `label` element, with a value that matches the value of the `id` attribute of the `input` element. This allows assistive technologies to create a linked relationship between the `label` and the child `input` element. For example:

```
<label for="indoor">
  <input id="indoor" type="radio" name="indoor-outdoor">Indoor
</label>
```

Instructions

Add a pair of radio buttons to your form, each nested in its own `label` element. One should have the option of `indoor` and the other should have the option of `outdoor`. Both should share the `name` attribute of `indoor-outdoor` to create a radio group.

Challenge Seed

```
<h2>CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <p>Things cats love:</p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
  <form action="/submit-cat-photo">
    <input type="text" placeholder="cat photo URL" required>
    <button type="submit">Submit</button>
  </form>
</main>
```

Solution

```
// solution required
```

23. Create a Set of Checkboxes

Description

Forms commonly use `checkboxes` for questions that may have more than one answer. Checkboxes are a type of `input`. Each of your checkboxes can be nested within its own `label` element. By wrapping an `input` element inside of a `label` element it will automatically associate the checkbox input with the `label` element surrounding it. All related checkbox inputs should have the same `name` attribute. It is considered best practice to explicitly define the relationship between a checkbox `input` and its corresponding `label` by setting the `for` attribute on the `label` element to match the `id` attribute of the associated `input` element. Here's an example of a checkbox: `<label for="loving"><input id="loving" type="checkbox" name="personality"> Loving</label>`

Instructions

Add to your form a set of three checkboxes. Each checkbox should be nested within its own `label` element. All three should share the `name` attribute of `personality`.

Challenge Seed

```
<h2>CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <p>Things cats love:</p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
  <form action="/submit-cat-photo">
    <label for="indoor"><input id="indoor" type="radio" name="indoor-outdoor"> Indoor</label>
    <label for="outdoor"><input id="outdoor" type="radio" name="indoor-outdoor"> Outdoor</label><br>
    <input type="text" placeholder="cat photo URL" required>
    <button type="submit">Submit</button>
  </form>
</main>
```

Solution

```
// solution required
```

24. Check Radio Buttons and Checkboxes by Default

Description

You can set a checkbox or radio button to be checked by default using the `checked` attribute. To do this, just add the word "checked" to the inside of an input element. For example: `<input type="radio" name="test-name" checked>`

Instructions

Set the first of your `radio` buttons and the first of your `checkbox`s to both be checked by default.

Challenge Seed

```
<h2>CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <p>Things cats love:</p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
```

```

<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor"> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
  <label><input type="checkbox" name="personality"> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Energetic</label><br>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</main>

```

Solution

```

<h2>CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <p>Things cats love:</p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
  <form action="/submit-cat-photo">
    <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
    <label><input type="checkbox" name="personality" checked> Loving</label>
    <label><input type="checkbox" name="personality"> Lazy</label>
    <label><input type="checkbox" name="personality"> Energetic</label><br>
    <input type="text" placeholder="cat photo URL" required>
    <button type="submit">Submit</button>
  </form>
</main>

```

25. Nest Many Elements within a Single div Element

Description

The `div` element, also known as a division element, is a general purpose container for other elements. The `div` element is probably the most commonly used HTML element of all. Just like any other non-self-closing element, you can open a `div` element with `<div>` and close it on another line with `</div>`.

Instructions

Nest your "Things cats love" and "Things cats hate" lists all within a single `div` element. Hint: Try putting your opening `div` tag above your "Things cats love" `p` element and your closing `div` tag after your closing `ol` tag so that both of your lists are within one `div`.

Challenge Seed

```

<h2>CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

```

```

<p>Things cats love:</p>
<ul>
  <li>cat nip</li>
  <li>laser pointers</li>
  <li>lasagna</li>
</ul>
<p>Top 3 things cats hate:</p>
<ol>
  <li>flea treatment</li>
  <li>thunder</li>
  <li>other cats</li>
</ol>

<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
  <label><input type="checkbox" name="personality" checked> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Energetic</label><br>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</main>

```

Solution

```
// solution required
```

26. Declare the Doctype of an HTML Document

Description

The challenges so far have covered specific HTML elements and their uses. However, there are a few elements that give overall structure to your page, and should be included in every HTML document. At the top of your document, you need to tell the browser which version of HTML your page is using. HTML is an evolving language, and is updated regularly. Most major browsers support the latest specification, which is HTML5. However, older web pages may use previous versions of the language. You tell the browser this information by adding the `<!DOCTYPE ...>` tag on the first line, where the "..." part is the version of HTML. For HTML5, you use `<!DOCTYPE html>`. The `!` and uppercase `DOCTYPE` is important, especially for older browsers. The `html` is not case sensitive. Next, the rest of your HTML code needs to be wrapped in `html` tags. The opening `<html>` goes directly below the `<!DOCTYPE html>` line, and the closing `</html>` goes at the end of the page. Here's an example of the page structure:

```

<!DOCTYPE html>
<html>
  <!-- Your HTML code goes here -->
</html>

```

Instructions

Add a `DOCTYPE` tag for HTML5 to the top of the blank HTML document in the code editor. Under it, add opening and closing `html` tags, which wrap around an `h1` element. The heading can include any text.

Challenge Seed

Solution

```

<!DOCTYPE html>
<html>

```

```
<h1> Hello world </h1>
</html>
```

27. Define the Head and Body of an HTML Document

Description

You can add another level of organization in your HTML document within the `html` tags with the `head` and `body` elements. Any markup with information about your page would go into the `head` tag. Then any markup with the content of the page (what displays for a user) would go into the `body` tag. Metadata elements, such as `link`, `meta`, `title`, and `style`, typically go inside the `head` element. Here's an example of a page's layout:

```
<!DOCTYPE html>
<html>
  <head>
    <!-- metadata elements -->
  </head>
  <body>
    <!-- page contents -->
  </body>
</html>
```

Instructions

Edit the markup so there's a `head` and a `body`. The `head` element should only include the `title`, and the `body` element should only include the `h1` and `p`.

Challenge Seed

```
<!DOCTYPE html>
<html>
  <title>The best page ever</title>

  <h1>The best page ever</h1>
  <p>Cat ipsum dolor sit amet, jump launch to pounce upon little yarn mouse, bare fangs at toy run hide in litter box until treats are fed. Go into a room to decide you didn't want to be in there anyway. I like big cats and i can not lie kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff. Meow i could pee on this if i had the energy for slap owner's face at 5am until human fills food dish yet scamper. Knock dish off table head butt cant eat out of my own dish scratch the furniture. Make meme, make cute face. Sleep in the bathroom sink chase laser but pee in the shoe. Paw at your fat belly licks your face and eat grass, throw it back up kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff.</p>

</html>
```

Solution

```
<!DOCTYPE html>
<html>
  <head>
    <title>The best page ever</title>
  </head>
  <body>
    <h1>The best page ever</h1>
    <p>Cat ipsum dolor sit amet, jump launch to pounce upon little yarn mouse, bare fangs at toy run hide in litter box until treats are fed. Go into a room to decide you didn't want to be in there anyway. I like big cats and i can not lie kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff. Meow i could pee on this if i had the energy for slap owner's face at 5am until human fills food dish yet scamper. Knock dish off table head butt cant eat out of my own dish scratch the furniture. Make meme, make cute face. Sleep in the bathroom sink chase laser but pee in the shoe. Paw at your fat belly licks your face and eat grass, throw it back up kitty ipsum dolor sit amet, shed everywhere shed everywhere stretching attack your ankles chase the red dot, hairball run catnip eat the grass sniff.</p>
```

```
ankles chase the red dot, hairball run catnip eat the grass sniff.</p>
</body>
</html>
```

Basic CSS

1. Change the Color of Text

Description

Now let's change the color of some of our text. We can do this by changing the `style` of your `h2` element. The property that is responsible for the color of an element's text is the `color` `style` property. Here's how you would set your `h2` element's text color to blue: `<h2 style="color: blue;">CatPhotoApp</h2>` Note that it is a good practice to end inline `style` declarations with a `;`.

Instructions

Change your `h2` element's style so that its text color is red.

Challenge Seed

```
<h2>CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <div>
    <p>Things cats love:</p>
    <ul>
      <li>cat nip</li>
      <li>laser pointers</li>
      <li>lasagna</li>
    </ul>
    <p>Top 3 things cats hate:</p>
    <ol>
      <li>flea treatment</li>
      <li>thunder</li>
      <li>other cats</li>
    </ol>
  </div>

  <form action="/submit-cat-photo">
    <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
    <label><input type="checkbox" name="personality" checked> Loving</label>
    <label><input type="checkbox" name="personality"> Lazy</label>
    <label><input type="checkbox" name="personality"> Energetic</label><br>
    <input type="text" placeholder="cat photo URL" required>
    <button type="submit">Submit</button>
  </form>
</main>
```

Solution

```
<h2 style="color: red;">CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <div>
    <p>Things cats love:</p>
```

```

<ul>
  <li>cat nip</li>
  <li>laser pointers</li>
  <li>lasagna</li>
</ul>
<p>Top 3 things cats hate:</p>
<ol>
  <li>flea treatment</li>
  <li>thunder</li>
  <li>other cats</li>
</ol>
</div>

<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
  <label><input type="checkbox" name="personality" checked> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Energetic</label><br>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</main>

```

2. Use CSS Selectors to Style Elements

Description

With CSS, there are hundreds of CSS properties that you can use to change the way an element looks on your page. When you entered `<h2 style="color: red">CatPhotoApp</h2>`, you were styling that individual `h2` element with inline CSS, which stands for Cascading Style Sheets. That's one way to specify the style of an element, but there's a better way to apply CSS. At the top of your code, create a `style` block like this:

```

<style>
</style>

```

Inside that style block, you can create a CSS selector for all `h2` elements. For example, if you wanted all `h2` elements to be red, you would add a style rule that looks like this:

```

<style>
  h2 {color: red;}
</style>

```

Note that it's important to have both opening and closing curly braces (`{` and `}`) around each element's style rule(s). You also need to make sure that your element's style definition is between the opening and closing style tags. Finally, be sure to add a semicolon to the end of each of your element's style rules.

Instructions

Delete your `h2` element's style attribute, and instead create a CSS `style` block. Add the necessary CSS to turn all `h2` elements blue.

Challenge Seed

```

<h2 style="color: red">CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>
  <a href="#"></a>
  <div>
    <p>Things cats love:</p>
    <ul>
      <li>cat nip</li>
      <li>laser pointers</li>
      <li>lasagna</li>
    </ul>
  </div>

```

```

<p>Top 3 things cats hate:</p>
<ol>
  <li>flea treatment</li>
  <li>thunder</li>
  <li>other cats</li>
</ol>
</div>

<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
  <label><input type="checkbox" name="personality" checked> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Energetic</label><br>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</main>

```

Solution

```
// solution required
```

3. Use a CSS Class to Style an Element

Description

Classes are reusable styles that can be added to HTML elements. Here's an example CSS class declaration:

```

<style>
  .blue-text {
    color: blue;
  }
</style>

```

You can see that we've created a CSS class called `blue-text` within the `<style>` tag. You can apply a class to an HTML element like this: `<h2 class="blue-text">CatPhotoApp</h2>`. Note that in your CSS `style` element, class names start with a period. In your HTML elements' `class` attribute, the class name does not include the period.

Instructions

Inside your `style` element, change the `h2` selector to `.red-text` and update the color's value from `blue` to `red`. Give your `h2` element the `class` attribute with a value of `'red-text'`.

Challenge Seed

```

<style>
  h2 {
    color: blue;
  }
</style>

<h2>CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>.</p>
  <a href="#"></a>
  <div>
    <p>Things cats love:</p>
    <ul>
      <li>cat nip</li>
      <li>laser pointers</li>
      <li>lasagna</li>
    </ul>
  </div>

```

```

</ul>
<p>Top 3 things cats hate:</p>
<ol>
  <li>flea treatment</li>
  <li>thunder</li>
  <li>other cats</li>
</ol>
</div>

<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
  <label><input type="checkbox" name="personality" checked> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Energetic</label><br>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</main>

```

Solution

```
// solution required
```

4. Style Multiple Elements with a CSS Class

Description

Classes allow you to use the same CSS styles on multiple HTML elements. You can see this by applying your `red-text` class to the first `p` element.

Instructions

Challenge Seed

```

<style>
  .red-text {
    color: red;
  }
</style>

<h2 class="red-text">CatPhotoApp</h2>
<main>
  <p>Click here to view more <a href="#">cat photos</a>. </p>
  <a href="#"></a>

  <div>
    <p>Things cats love:</p>
    <ul>
      <li>cat nip</li>
      <li>laser pointers</li>
      <li>lasagna</li>
    </ul>
    <p>Top 3 things cats hate:</p>
    <ol>
      <li>flea treatment</li>
      <li>thunder</li>
      <li>other cats</li>
    </ol>
  </div>

  <form action="/submit-cat-photo">
    <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>

```

```

<label><input type="checkbox" name="personality" checked> Loving</label>
<label><input type="checkbox" name="personality"> Lazy</label>
<label><input type="checkbox" name="personality"> Energetic</label><br>
<input type="text" placeholder="cat photo URL" required>
<button type="submit">Submit</button>
</form>
</main>

```

Solution

```
// solution required
```

5. Change the Font Size of an Element

Description

Font size is controlled by the `font-size` CSS property, like this:

```

h1 {
  font-size: 30px;
}

```

Instructions

Inside the same `<style>` tag that contains your `red-text` class, create an entry for `p` elements and set the `font-size` to 16 pixels (`16px`).

Challenge Seed

```

<style>
  .red-text {
    color: red;
  }
</style>

<h2 class="red-text">CatPhotoApp</h2>
<main>
  <p class="red-text">Click here to view more <a href="#">cat photos</a>.</p>
  <a href="#"></a>
  <div>
    <p>Things cats love:</p>
    <ul>
      <li>cat nip</li>
      <li>laser pointers</li>
      <li>lasagna</li>
    </ul>
    <p>Top 3 things cats hate:</p>
    <ol>
      <li>flea treatment</li>
      <li>thunder</li>
      <li>other cats</li>
    </ol>
  </div>

  <form action="/submit-cat-photo">
    <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
    <label><input type="checkbox" name="personality" checked> Loving</label>
    <label><input type="checkbox" name="personality"> Lazy</label>
    <label><input type="checkbox" name="personality"> Energetic</label><br>
    <input type="text" placeholder="cat photo URL" required>
    <button type="submit">Submit</button>
  </form>
</main>

```

```
</form>
</main>
```

Solution

```
// solution required
```

6. Set the Font Family of an Element

Description

You can set which font an element should use, by using the `font-family` property. For example, if you wanted to set your `h2` element's font to `sans-serif`, you would use the following CSS:

```
h2 {
  font-family: sans-serif;
}
```

Instructions

Make all of your `p` elements use the `monospace` font.

Challenge Seed

```
<style>
  .red-text {
    color: red;
  }

  p {
    font-size: 16px;
  }
</style>

<h2 class="red-text">CatPhotoApp</h2>
<main>
  <p class="red-text">Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <div>
    <p>Things cats love:</p>
    <ul>
      <li>cat nip</li>
      <li>laser pointers</li>
      <li>lasagna</li>
    </ul>
    <p>Top 3 things cats hate:</p>
    <ol>
      <li>flea treatment</li>
      <li>thunder</li>
      <li>other cats</li>
    </ol>
  </div>

  <form action="/submit-cat-photo">
    <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
    <label><input type="checkbox" name="personality" checked> Loving</label>
    <label><input type="checkbox" name="personality"> Lazy</label>
    <label><input type="checkbox" name="personality"> Energetic</label><br>
    <input type="text" placeholder="cat photo URL" required>
    <button type="submit">Submit</button>
  </form>
</main>
```

Solution

```
// solution required
```

7. Import a Google Font

Description

In addition to specifying common fonts that are found on most operating systems, we can also specify non-standard, custom web fonts for use on our website. There are many sources for web fonts on the Internet. For this example we will focus on the Google Fonts library. [Google Fonts](#) is a free library of web fonts that you can use in your CSS by referencing the font's URL. So, let's go ahead and import and apply a Google font (note that if Google is blocked in your country, you will need to skip this challenge). To import a Google Font, you can copy the font(s) URL from the Google Fonts library and then paste it in your HTML. For this challenge, we'll import the `Lobster` font. To do this, copy the following code snippet and paste it into the top of your code editor (before the opening `style` element): `<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">` Now you can use the `Lobster` font in your CSS by using `Lobster` as the `FAMILY_NAME` as in the following example:

`font-family: FAMILY_NAME, GENERIC_NAME;` . The `GENERIC_NAME` is optional, and is a fallback font in case the other specified font is not available. This is covered in the next challenge. Family names are case-sensitive and need to be wrapped in quotes if there is a space in the name. For example, you need quotes to use the "Open Sans" font, but not to use the `Lobster` font.

Instructions

Create a `font-family` CSS rule that uses the `Lobster` font, and ensure that it will be applied to your `h2` element.

Challenge Seed

```
<style>
  .red-text {
    color: red;
  }

  p {
    font-size: 16px;
    font-family: monospace;
  }
</style>

<h2 class="red-text">CatPhotoApp</h2>
<main>
  <p class="red-text">Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <div>
    <p>Things cats love:</p>
    <ul>
      <li>cat nip</li>
      <li>laser pointers</li>
      <li>lasagna</li>
    </ul>
    <p>Top 3 things cats hate:</p>
    <ol>
      <li>flea treatment</li>
      <li>thunder</li>
      <li>other cats</li>
    </ol>
  </div>

  <form action="/submit-cat-photo">
    <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
  </form>
</main>
```

```

<label><input type="checkbox" name="personality" checked> Loving</label>
<label><input type="checkbox" name="personality"> Lazy</label>
<label><input type="checkbox" name="personality"> Energetic</label><br>
<input type="text" placeholder="cat photo URL" required>
<button type="submit">Submit</button>
</form>
</main>

```

Solution

```
// solution required
```

8. Specify How Fonts Should Degrade

Description

There are several default fonts that are available in all browsers. These generic font families include `monospace`, `serif` and `sans-serif`. When one font isn't available, you can tell the browser to "degrade" to another font. For example, if you wanted an element to use the `Helvetica` font, but degrade to the `sans-serif` font when `Helvetica` isn't available, you will specify it as follows:

```

p {
  font-family: Helvetica, sans-serif;
}

```

Generic font family names are not case-sensitive. Also, they do not need quotes because they are CSS keywords.

Instructions

To begin with, apply the `monospace` font to the `h2` element, so that it now has two fonts - `Lobster` and `monospace`. In the last challenge, you imported the `Lobster` font using the `link` tag. Now comment out that import of the `Lobster` font (using the HTML comments you learned before) from Google Fonts so that it isn't available anymore. Notice how your `h2` element degrades to the `monospace` font. **Note**

If you have the `Lobster` font installed on your computer, you won't see the degradation because your browser is able to find the font.

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster;
  }

  p {
    font-size: 16px;
    font-family: monospace;
  }
</style>

<h2 class="red-text">CatPhotoApp</h2>
<main>
  <p class="red-text">Click here to view more <a href="#">cat photos</a>.</p>
  <a href="#"></a>
  <div>
    <p>Things cats love:</p>
    <ul>

```

```

<li>cat nip</li>
<li>laser pointers</li>
<li>lasagna</li>
</ul>
<p>Top 3 things cats hate:</p>
<ol>
  <li>flea treatment</li>
  <li>thunder</li>
  <li>other cats</li>
</ol>
</div>

<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
  <label><input type="checkbox" name="personality" checked> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Energetic</label><br>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</main>

```

Solution

```
// solution required
```

9. Size Your Images

Description

CSS has a property called `width` that controls an element's width. Just like with fonts, we'll use `px` (pixels) to specify the image's width. For example, if we wanted to create a CSS class called `larger-image` that gave HTML elements a width of 500 pixels, we'd use:

```

<style>
.larger-image {
  width: 500px;
}
</style>

```

Instructions

Create a class called `smaller-image` and use it to resize the image so that it's only 100 pixels wide. **Note**
Due to browser implementation differences, you may need to be at 100% zoom to pass the tests on this challenge.

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
.red-text {
  color: red;
}

h2 {
  font-family: Lobster, monospace;
}

p {
  font-size: 16px;
  font-family: monospace;
}
</style>

```

```

<h2 class="red-text">CatPhotoApp</h2>
<main>
  <p class="red-text">Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <div>
    <p>Things cats love:</p>
    <ul>
      <li>cat nip</li>
      <li>laser pointers</li>
      <li>lasagna</li>
    </ul>
    <p>Top 3 things cats hate:</p>
    <ol>
      <li>flea treatment</li>
      <li>thunder</li>
      <li>other cats</li>
    </ol>
  </div>

  <form action="/submit-cat-photo">
    <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
    <label><input type="checkbox" name="personality" checked> Loving</label>
    <label><input type="checkbox" name="personality"> Lazy</label>
    <label><input type="checkbox" name="personality"> Energetic</label><br>
    <input type="text" placeholder="cat photo URL" required>
    <button type="submit">Submit</button>
  </form>
</main>

```

Solution

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, monospace;
  }

  p {
    font-size: 16px;
    font-family: monospace;
  }

  .smaller-image {
    width: 100px;
  }
</style>

<h2 class="red-text">CatPhotoApp</h2>
<main>
  <p class="red-text">Click here to view more <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  <div>
    <p>Things cats love:</p>
    <ul>
      <li>cat nip</li>
      <li>laser pointers</li>
      <li>lasagna</li>
    </ul>
    <p>Top 3 things cats hate:</p>
    <ol>
      <li>flea treatment</li>
      <li>thunder</li>
      <li>other cats</li>
    </ol>
  </div>

```

```

</div>

<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
  <label><input type="checkbox" name="personality" checked> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Energetic</label><br>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</main>

```

10. Add Borders Around Your Elements

Description

CSS borders have properties like `style`, `color` and `width`. For example, if we wanted to create a red, 5 pixel border around an HTML element, we could use this class:

```

<style>
.thin-red-border {
  border-color: red;
  border-width: 5px;
  border-style: solid;
}
</style>

```

Instructions

Create a class called `thick-green-border`. This class should add a 10px, solid, green border around an HTML element. Apply the class to your cat photo. Remember that you can apply multiple classes to an element using its `class` attribute, by separating each class name with a space. For example: ``

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
.red-text {
  color: red;
}

h2 {
  font-family: Lobster, monospace;
}

p {
  font-size: 16px;
  font-family: monospace;
}

.smaller-image {
  width: 100px;
}
</style>

<h2 class="red-text">CatPhotoApp</h2>
<main>
  <p class="red-text">Click here to view more <a href="#">cat photos</a>.</p>
  <a href="#"></a>
  <div>
    <p>Things cats love:</p>
    <ul>
      <li>cat nip</li>

```

```

<li>laser pointers</li>
<li>lasagna</li>
</ul>
<p>Top 3 things cats hate:</p>
<ol>
  <li>flea treatment</li>
  <li>thunder</li>
  <li>other cats</li>
</ol>
</div>

<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
  <label><input type="checkbox" name="personality" checked> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Energetic</label><br>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</main>

```

Solution

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, monospace;
  }

  p {
    font-size: 16px;
    font-family: monospace;
  }

  .smaller-image {
    width: 100px;
  }

  .thick-green-border {
    border-width: 10px;
    border-color: green;
    border-style: solid;
  }
</style>

<h2 class="red-text">CatPhotoApp</h2>
<main>
  <p class="red-text">Click here to view more <a href="#">cat photos</a>.</p>
  <a href="#"></a>

  <div>
    <p>Things cats love:</p>
    <ul>
      <li>cat nip</li>
      <li>laser pointers</li>
      <li>lasagna</li>
    </ul>
    <p>Top 3 things cats hate:</p>
    <ol>
      <li>flea treatment</li>
      <li>thunder</li>
      <li>other cats</li>
    </ol>
  </div>

  <form action="/submit-cat-photo">
    <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>

```

```

<label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
<label><input type="checkbox" name="personality" checked> Loving</label>
<label><input type="checkbox" name="personality"> Lazy</label>
<label><input type="checkbox" name="personality"> Energetic</label><br>
<input type="text" placeholder="cat photo URL" required>
<button type="submit">Submit</button>
</form>
</main>

```

11. Add Rounded Corners with border-radius

Description

Your cat photo currently has sharp corners. We can round out those corners with a CSS property called `border-radius`.

Instructions

You can specify a `border-radius` with pixels. Give your cat photo a `border-radius` of `10px`. Note: this challenge allows for multiple possible solutions. For example, you may add `border-radius` to either the `.thick-green-border` class or the `.smaller-image` class.

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, monospace;
  }

  p {
    font-size: 16px;
    font-family: monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
  }

  .smaller-image {
    width: 100px;
  }
</style>

<h2 class="red-text">CatPhotoApp</h2>
<main>
  <p class="red-text">Click here to view more <a href="#">cat photos</a>.</p>
  <a href="#"></a>

  <div>
    <p>Things cats love:</p>
    <ul>
      <li>cat nip</li>
      <li>laser pointers</li>
      <li>lasagna</li>
    </ul>
    <p>Top 3 things cats hate:</p>
    <ol>
      <li>flea treatment</li>

```

```

<li>thunder</li>
<li>other cats</li>
</ol>
</div>

<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
  <label><input type="checkbox" name="personality" checked> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Energetic</label><br>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</main>

```

Solution

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, monospace;
  }

  p {
    font-size: 16px;
    font-family: monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
  }

  .smaller-image {
    width: 100px;
    border-radius: 10px;
  }
</style>

<h2 class="red-text">CatPhotoApp</h2>
<main>
  <p class="red-text">Click here to view more <a href="#">cat photos</a>.</p>
  <a href="#"></a>

  <div>
    <p>Things cats love:</p>
    <ul>
      <li>cat nip</li>
      <li>laser pointers</li>
      <li>lasagna</li>
    </ul>
    <p>Top 3 things cats hate:</p>
    <ol>
      <li>flea treatment</li>
      <li>thunder</li>
      <li>other cats</li>
    </ol>
  </div>

  <form action="/submit-cat-photo">
    <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
    <label><input type="checkbox" name="personality" checked> Loving</label>
    <label><input type="checkbox" name="personality"> Lazy</label>
    <label><input type="checkbox" name="personality"> Energetic</label><br>
    <input type="text" placeholder="cat photo URL" required>
  </form>
</main>

```

```
<button type="submit">Submit</button>
</form>
</main>
```

12. Make Circular Images with a border-radius

Description

In addition to pixels, you can also specify the `border-radius` using a percentage.

Instructions

Give your cat photo a `border-radius` of 50%.

Challenge Seed

```
<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, monospace;
  }

  p {
    font-size: 16px;
    font-family: monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 10px;
  }

  .smaller-image {
    width: 100px;
  }
</style>

<h2 class="red-text">CatPhotoApp</h2>
<main>
  <p class="red-text">Click here to view more <a href="#">cat photos</a>.</p>
  <a href="#"></a>

  <div>
    <p>Things cats love:</p>
    <ul>
      <li>cat nip</li>
      <li>laser pointers</li>
      <li>lasagna</li>
    </ul>
    <p>Top 3 things cats hate:</p>
    <ol>
      <li>flea treatment</li>
      <li>thunder</li>
      <li>other cats</li>
    </ol>
  </div>

  <form action="/submit-cat-photo">
    <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
```

```

<label><input type="checkbox" name="personality" checked> Loving</label>
<label><input type="checkbox" name="personality"> Lazy</label>
<label><input type="checkbox" name="personality"> Energetic</label><br>
<input type="text" placeholder="cat photo URL" required>
<button type="submit">Submit</button>
</form>
</main>

```

Solution

```
// solution required
```

13. Give a Background Color to a div Element

Description

You can set an element's background color with the `background-color` property. For example, if you wanted an element's background color to be `green`, you'd put this within your `style` element:

```
.green-background {
  background-color: green;
}
```

Instructions

Create a class called `silver-background` with the `background-color` of silver. Assign this class to your `div` element.

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, monospace;
  }

  p {
    font-size: 16px;
    font-family: monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }

  .smaller-image {
    width: 100px;
  }
</style>

<h2 class="red-text">CatPhotoApp</h2>
<main>
  <p class="red-text">Click here to view more <a href="#">cat photos</a>.</p>
  <a href="#"></a>
  <div>
    <p>Things cats love:</p>

```

```

<ul>
  <li>cat nip</li>
  <li>laser pointers</li>
  <li>lasagna</li>
</ul>
<p>Top 3 things cats hate:</p>
<ol>
  <li>flea treatment</li>
  <li>thunder</li>
  <li>other cats</li>
</ol>
</div>

<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
  <label><input type="checkbox" name="personality" checked> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Energetic</label><br>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</main>

```

Solution

```
// solution required
```

14. Set the id of an Element

Description

In addition to classes, each HTML element can also have an `id` attribute. There are several benefits to using `id` attributes: You can use an `id` to style a single element and later you'll learn that you can use them to select and modify specific elements with JavaScript. `id` attributes should be unique. Browsers won't enforce this, but it is a widely agreed upon best practice. So please don't give more than one element the same `id` attribute. Here's an example of how you give your `h2` element the `id` of `cat-photo-app` : `<h2 id="cat-photo-app">`

Instructions

Give your `form` element the `id` `cat-photo-form`.

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, monospace;
  }

  p {
    font-size: 16px;
    font-family: monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }

```

```

}

.smaller-image {
  width: 100px;
}

.silver-background {
  background-color: silver;
}
</style>

<h2 class="red-text">CatPhotoApp</h2>
<main>
  <p class="red-text">Click here to view more <a href="#">cat photos</a>.</p>
  <a href="#"></a>

  <div class="silver-background">
    <p>Things cats love:</p>
    <ul>
      <li>cat nip</li>
      <li>laser pointers</li>
      <li>lasagna</li>
    </ul>
    <p>Top 3 things cats hate:</p>
    <ol>
      <li>flea treatment</li>
      <li>thunder</li>
      <li>other cats</li>
    </ol>
  </div>

  <form action="/submit-cat-photo">
    <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
    <label><input type="checkbox" name="personality" checked> Loving</label>
    <label><input type="checkbox" name="personality"> Lazy</label>
    <label><input type="checkbox" name="personality"> Energetic</label><br>
    <input type="text" placeholder="cat photo URL" required>
    <button type="submit">Submit</button>
  </form>
</main>

```

Solution

```
// solution required
```

15. Use an id Attribute to Style an Element

Description

One cool thing about `id` attributes is that, like classes, you can style them using CSS. However, an `id` is not reusable and should only be applied to one element. An `id` also has a higher specificity (importance) than a class so if both are applied to the same element and have conflicting styles, the styles of the `id` will be applied. Here's an example of how you can take your element with the `id` attribute of `cat-photo-element` and give it the background color of green. In your `style` element:

```
#cat-photo-element {
  background-color: green;
}
```

Note that inside your `style` element, you always reference classes by putting a `.` in front of their names. You always reference ids by putting a `#` in front of their names.

Instructions

Try giving your form, which now has the `id` attribute of `cat-photo-form`, a green background.

Challenge Seed

```
<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, monospace;
  }

  p {
    font-size: 16px;
    font-family: monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }

  .smaller-image {
    width: 100px;
  }

  .silver-background {
    background-color: silver;
  }
</style>

<h2 class="red-text">CatPhotoApp</h2>
<main>
  <p class="red-text">Click here to view more <a href="#">cat photos</a>.</p>
  <a href="#"></a>

  <div class="silver-background">
    <p>Things cats love:</p>
    <ul>
      <li>cat nip</li>
      <li>laser pointers</li>
      <li>lasagna</li>
    </ul>
    <p>Top 3 things cats hate:</p>
    <ol>
      <li>flea treatment</li>
      <li>thunder</li>
      <li>other cats</li>
    </ol>
  </div>

  <form action="/submit-cat-photo" id="cat-photo-form">
    <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
    <label><input type="checkbox" name="personality" checked> Loving</label>
    <label><input type="checkbox" name="personality"> Lazy</label>
    <label><input type="checkbox" name="personality"> Energetic</label><br>
    <input type="text" placeholder="cat photo URL" required>
    <button type="submit">Submit</button>
  </form>
</main>
```

Solution

```
// solution required
```

16. Adjust the Padding of an Element

Description

Now let's put our Cat Photo App away for a little while and learn more about styling HTML. You may have already noticed this, but all HTML elements are essentially little rectangles. Three important properties control the space that surrounds each HTML element: `padding`, `margin`, and `border`. An element's `padding` controls the amount of space between the element's content and its `border`. Here, we can see that the blue box and the red box are nested within the yellow box. Note that the red box has more `padding` than the blue box. When you increase the blue box's `padding`, it will increase the distance (`padding`) between the text and the border around it.

Instructions

Change the `padding` of your blue box to match that of your red box.

Challenge Seed

```
<style>
.injected-text {
  margin-bottom: -25px;
  text-align: center;
}

.box {
  border-style: solid;
  border-color: black;
  border-width: 5px;
  text-align: center;
}

.yellow-box {
  background-color: yellow;
  padding: 10px;
}

.red-box {
  background-color: crimson;
  color: #fff;
  padding: 20px;
}

.blue-box {
  background-color: blue;
  color: #fff;
  padding: 10px;
}
</style>
<h5 class="injected-text">margin</h5>

<div class="box yellow-box">
  <h5 class="box red-box">padding</h5>
  <h5 class="box blue-box">padding</h5>
</div>
```

Solution

```
// solution required
```

17. Adjust the Margin of an Element

Description

An element's `margin` controls the amount of space between an element's `border` and surrounding elements. Here, we can see that the blue box and the red box are nested within the yellow box. Note that the red box has a bigger `margin` than the blue box, making it appear smaller. When you increase the blue box's `margin`, it will increase the distance between its border and surrounding elements.

Instructions

Change the `margin` of the blue box to match that of the red box.

Challenge Seed

```
<style>
.injected-text {
  margin-bottom: -25px;
  text-align: center;
}

.box {
  border-style: solid;
  border-color: black;
  border-width: 5px;
  text-align: center;
}

.yellow-box {
  background-color: yellow;
  padding: 10px;
}

.red-box {
  background-color: crimson;
  color: #fff;
  padding: 20px;
  margin: 20px;
}

.blue-box {
  background-color: blue;
  color: #fff;
  padding: 20px;
  margin: 10px;
}
</style>
<h5 class="injected-text">margin</h5>

<div class="box yellow-box">
  <h5 class="box red-box">padding</h5>
  <h5 class="box blue-box">padding</h5>
</div>
```

Solution

```
<style>
.injected-text {
  margin-bottom: -25px;
  text-align: center;
}

.box {
  border-style: solid;
  border-color: black;
  border-width: 5px;
  text-align: center;
}

.yellow-box {
  background-color: yellow;
  padding: 10px;
}

.red-box {
  background-color: crimson;
  color: #fff;
  padding: 20px;
  margin: 20px;
}

.blue-box {
  background-color: blue;
  color: #fff;
  padding: 20px;
  margin: 20px;
}
</style>
```

```
padding: 10px;
}

.red-box {
  background-color: crimson;
  color: #fff;
  padding: 20px;
  margin: 20px;
}

.blue-box {
  background-color: blue;
  color: #fff;
  padding: 20px;
  margin: 20px;
}
</style>
<h5 class="injected-text">margin</h5>

<div class="box yellow-box">
  <h5 class="box red-box">padding</h5>
  <h5 class="box blue-box">padding</h5>
</div>
```

18. Add a Negative Margin to an Element

Description

An element's `margin` controls the amount of space between an element's `border` and surrounding elements. If you set an element's `margin` to a negative value, the element will grow larger.

Instructions

Try to set the `margin` to a negative value like the one for the red box. Change the `margin` of the blue box to `-15px`, so it fills the entire horizontal width of the yellow box around it.

Challenge Seed

```
<style>
.injected-text {
  margin-bottom: -25px;
  text-align: center;
}

.box {
  border-style: solid;
  border-color: black;
  border-width: 5px;
  text-align: center;
}

.yellow-box {
  background-color: yellow;
  padding: 10px;
}

.red-box {
  background-color: crimson;
  color: #fff;
  padding: 20px;
  margin: -15px;
}

.blue-box {
  background-color: blue;
  color: #fff;
  padding: 20px;
  margin: 20px;
}</style>
```

```

        }
    </style>

<div class="box yellow-box">
    <h5 class="box red-box">padding</h5>
    <h5 class="box blue-box">padding</h5>
</div>

```

Solution

```

<style>
    .injected-text {
        margin-bottom: -25px;
        text-align: center;
    }

    .box {
        border-style: solid;
        border-color: black;
        border-width: 5px;
        text-align: center;
    }

    .yellow-box {
        background-color: yellow;
        padding: 10px;
    }

    .red-box {
        background-color: crimson;
        color: #fff;
        padding: 20px;
        margin: -15px;
    }

    .blue-box {
        background-color: blue;
        color: #fff;
        padding: 20px;
        margin: 20px;
        margin-top: -15px;
    }
</style>

<div class="box yellow-box">
    <h5 class="box red-box">padding</h5>
    <h5 class="box blue-box">padding</h5>
</div>

```

19. Add Different Padding to Each Side of an Element

Description

Sometimes you will want to customize an element so that it has different amounts of padding on each of its sides. CSS allows you to control the padding of all four individual sides of an element with the `padding-top`, `padding-right`, `padding-bottom`, and `padding-left` properties.

Instructions

Give the blue box a padding of 40px on its top and left side, but only 20px on its bottom and right side.

Challenge Seed

```

<style>
    .injected-text {

```

```

        margin-bottom: -25px;
        text-align: center;
    }

    .box {
        border-style: solid;
        border-color: black;
        border-width: 5px;
        text-align: center;
    }

    .yellow-box {
        background-color: yellow;
        padding: 10px;
    }

    .red-box {
        background-color: crimson;
        color: #fff;
        padding-top: 40px;
        padding-right: 20px;
        padding-bottom: 20px;
        padding-left: 40px;
    }

    .blue-box {
        background-color: blue;
        color: #fff;
    }

```

</style>

<h5 class="injected-text">margin</h5>

<div class="box yellow-box">

<h5 class="box red-box">padding</h5>

<h5 class="box blue-box">padding</h5>

</div>

Solution

```
// solution required
```

20. Add Different Margins to Each Side of an Element

Description

Sometimes you will want to customize an element so that it has a different `margin` on each of its sides. CSS allows you to control the `margin` of all four individual sides of an element with the `margin-top`, `margin-right`, `margin-bottom`, and `margin-left` properties.

Instructions

Give the blue box a `margin` of `40px` on its top and left side, but only `20px` on its bottom and right side.

Challenge Seed

```

<style>
    .injected-text {
        margin-bottom: -25px;
        text-align: center;
    }

    .box {
        border-style: solid;
        border-color: black;
        border-width: 5px;

```

```
        text-align: center;
    }

    .yellow-box {
        background-color: yellow;
        padding: 10px;
    }

    .red-box {
        background-color: crimson;
        color: #fff;
        margin-top: 40px;
        margin-right: 20px;
        margin-bottom: 20px;
        margin-left: 40px;
    }

    .blue-box {
        background-color: blue;
        color: #fff;
    }
</style>
<h5 class="injected-text">margin</h5>

<div class="box yellow-box">
    <h5 class="box red-box">padding</h5>
    <h5 class="box blue-box">padding</h5>
</div>
```

Solution

```
<style>
    .injected-text {
        margin-bottom: -25px;
        text-align: center;
    }

    .box {
        border-style: solid;
        border-color: black;
        border-width: 5px;
        text-align: center;
    }

    .yellow-box {
        background-color: yellow;
        padding: 10px;
    }

    .red-box {
        background-color: crimson;
        color: #fff;
        margin-top: 40px;
        margin-right: 20px;
        margin-bottom: 20px;
        margin-left: 40px;
    }

    .blue-box {
        background-color: blue;
        color: #fff;
        margin-top: 40px;
        margin-right: 20px;
        margin-bottom: 20px;
        margin-left: 40px;
    }
</style>
<h5 class="injected-text">margin</h5>

<div class="box yellow-box">
    <h5 class="box red-box">padding</h5>
    <h5 class="box blue-box">padding</h5>
</div>
```

21. Use Clockwise Notation to Specify the Padding of an Element

Description

Instead of specifying an element's `padding-top`, `padding-right`, `padding-bottom`, and `padding-left` properties individually, you can specify them all in one line, like this: `padding: 10px 20px 10px 20px;` These four values work like a clock: top, right, bottom, left, and will produce the exact same result as using the side-specific padding instructions.

Instructions

Use Clockwise Notation to give the ".blue-box" class a `padding` of `40px` on its top and left side, but only `20px` on its bottom and right side.

Challenge Seed

```
<style>
.injected-text {
  margin-bottom: -25px;
  text-align: center;
}

.box {
  border-style: solid;
  border-color: black;
  border-width: 5px;
  text-align: center;
}

.yellow-box {
  background-color: yellow;
  padding: 20px 40px 20px 40px;
}

.red-box {
  background-color: crimson;
  color: #fff;
  padding: 20px 40px 20px 40px;
}

.blue-box {
  background-color: blue;
  color: #fff;
}
</style>
<h5 class="injected-text">margin</h5>

<div class="box yellow-box">
  <h5 class="box red-box">padding</h5>
  <h5 class="box blue-box">padding</h5>
</div>
```

Solution

```
// solution required
```

22. Use Clockwise Notation to Specify the Margin of an Element

Description

Let's try this again, but with `margin` this time. Instead of specifying an element's `margin-top`, `margin-right`, `margin-bottom`, and `margin-left` properties individually, you can specify them all in one line, like this: `margin: 10px 20px 10px 20px;` These four values work like a clock: top, right, bottom, left, and will produce the exact same result as using the side-specific margin instructions.

Instructions

Use Clockwise Notation to give the element with the `blue-box` class a margin of `40px` on its top and left side, but only `20px` on its bottom and right side.

Challenge Seed

```
<style>
.injected-text {
  margin-bottom: -25px;
  text-align: center;
}

.box {
  border-style: solid;
  border-color: black;
  border-width: 5px;
  text-align: center;
}

.yellow-box {
  background-color: yellow;
  padding: 20px 40px 20px 40px;
}

.red-box {
  background-color: crimson;
  color: #fff;
  margin: 20px 40px 20px 40px;
}

.blue-box {
  background-color: blue;
  color: #fff;
}
</style>
<h5 class="injected-text">margin</h5>

<div class="box yellow-box">
  <h5 class="box red-box">padding</h5>
  <h5 class="box blue-box">padding</h5>
</div>
```

Solution

```
<style>
.injected-text {
  margin-bottom: -25px;
  text-align: center;
}

.box {
  border-style: solid;
  border-color: black;
  border-width: 5px;
  text-align: center;
}

.yellow-box {
  background-color: yellow;
  padding: 20px 40px 20px 40px;
}
```

```
.red-box {  
  background-color: crimson;  
  color: #fff;  
  margin: 20px 40px 20px 40px;  
}  
  
.blue-box {  
  background-color: blue;  
  color: #fff;  
  margin: 40px 20px 20px 40px;  
}  
/<style>  
<h5 class="injected-text">margin</h5>  
  
<div class="box yellow-box">  
  <h5 class="box red-box">padding</h5>  
  <h5 class="box blue-box">padding</h5>  
</div>
```

23. Use Attribute Selectors to Style Elements

Description

You have been given `id` or `class` attributes to elements that you wish to specifically style. These are known as ID and class selectors. There are other CSS Selectors you can use to select custom groups of elements to style. Let's bring out CatPhotoApp again to practice using CSS Selectors. For this challenge, you will use the `[attr=value]` attribute selector to style the checkboxes in CatPhotoApp. This selector matches and styles elements with a specific attribute value. For example, the below code changes the margins of all elements with the attribute `type` and a corresponding value of `radio`:

```
[type='radio'] {  
  margin: 20px 0px 20px 0px;  
}
```

Instructions

Using the `type` attribute selector, try to give the checkboxes in CatPhotoApp a top margin of 10px and a bottom margin of 15px.

Challenge Seed

```
<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">  
<style>  
  .red-text {  
    color: red;  
  }  
  
  h2 {  
    font-family: Lobster, monospace;  
  }  
  
  p {  
    font-size: 16px;  
    font-family: monospace;  
  }  
  
  .thick-green-border {  
    border-color: green;  
    border-width: 10px;  
    border-style: solid;  
    border-radius: 50%;  
  }  
  
  .smaller-image {  
    width: 100px;  
  }</style>
```

```

.silver-background {
  background-color: silver;
}
</style>

<h2 class="red-text">CatPhotoApp</h2>
<main>
  <p class="red-text">Click here to view more <a href="#">cat photos</a>.</p>
  <a href="#"></a>

  <div class="silver-background">
    <p>Things cats love:</p>
    <ul>
      <li>cat nip</li>
      <li>laser pointers</li>
      <li>lasagna</li>
    </ul>
    <p>Top 3 things cats hate:</p>
    <ol>
      <li>flea treatment</li>
      <li>thunder</li>
      <li>other cats</li>
    </ol>
  </div>

  <form action="/submit-cat-photo" id="cat-photo-form">
    <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>
    <label><input type="checkbox" name="personality" checked> Loving</label>
    <label><input type="checkbox" name="personality"> Lazy</label>
    <label><input type="checkbox" name="personality"> Energetic</label><br>
    <input type="text" placeholder="cat photo URL" required>
    <button type="submit">Submit</button>
  </form>
</main>

```

Solution

```
// solution required
```

24. Understand Absolute versus Relative Units

Description

The last several challenges all set an element's margin or padding with pixels (`px`). Pixels are a type of length unit, which tells the browser how to size or space an item. In addition to `px`, CSS has a number of different length unit options that you can use. The two main types of length units are absolute and relative. Absolute units tie to physical units of length. For example, `in` and `mm` refer to inches and millimeters, respectively. Absolute length units approximate the actual measurement on a screen, but there are some differences depending on a screen's resolution. Relative units, such as `em` or `rem`, are relative to another length value. For example, `em` is based on the size of an element's font. If you use it to set the `font-size` property itself, it's relative to the parent's `font-size`. **Note** There are several relative unit options that are tied to the size of the viewport. They are covered in the Responsive Web Design Principles section.

Instructions

Add a `padding` property to the element with class `red-box` and set it to `1.5em`.

Challenge Seed

```
<style>
.injected-text {
  margin-bottom: -25px;
  text-align: center;
}

.box {
  border-style: solid;
  border-color: black;
  border-width: 5px;
  text-align: center;
}

.yellow-box {
  background-color: yellow;
  padding: 20px 40px 20px 40px;
}

.red-box {
  background-color: red;
  margin: 20px 40px 20px 40px;
}

.green-box {
  background-color: green;
  margin: 20px 40px 20px 40px;
}
</style>
<h5 class="injected-text">margin</h5>

<div class="box yellow-box">
  <h5 class="box red-box">padding</h5>
  <h5 class="box green-box">padding</h5>
</div>
```

Solution

```
// solution required
```

25. Style the HTML Body Element

Description

Now let's start fresh and talk about CSS inheritance. Every HTML page has a `body` element.

Instructions

We can prove that the `body` element exists here by giving it a `background-color` of black. We can do this by adding the following to our `style` element:

```
body {
  background-color: black;
}
```

Challenge Seed

```
<style>
</style>
```

Solution

```
// solution required
```

26. Inherit Styles from the Body Element

Description

Now we've proven that every HTML page has a `body` element, and that its `body` element can also be styled with CSS. Remember, you can style your `body` element just like any other HTML element, and all your other elements will inherit your `body` element's styles.

Instructions

First, create a `h1` element with the text `Hello World`. Then, let's give all elements on your page the color of `green` by adding `color: green;` to your `body` element's style declaration. Finally, give your `body` element the font-family of `monospace` by adding `font-family: monospace;` to your `body` element's style declaration.

Challenge Seed

```
<style>
  body {
    background-color: black;
  }

</style>
```

Solution

```
// solution required
```

27. Prioritize One Style Over Another

Description

Sometimes your HTML elements will receive multiple styles that conflict with one another. For example, your `h1` element can't be both green and pink at the same time. Let's see what happens when we create a class that makes text pink, then apply it to an element. Will our class *override* the `body` element's `color: green;` CSS property?

Instructions

Create a CSS class called `pink-text` that gives an element the color pink. Give your `h1` element the class of `pink-text`.

Challenge Seed

```
<style>
  body {
    background-color: black;
    font-family: monospace;
    color: green;
  }

</style>
<h1>Hello World!</h1>
```

Solution

```
<style>
  body {
    background-color: black;
    font-family: monospace;
    color: green;
  }
  .pink-text {
    color: pink;
  }
</style>
<h1 class="pink-text">Hello World!</h1>
```

28. Override Styles in Subsequent CSS

Description

Our "pink-text" class overrode our `body` element's CSS declaration! We just proved that our classes will override the `body` element's CSS. So the next logical question is, what can we do to override our `pink-text` class?

Instructions

Create an additional CSS class called `blue-text` that gives an element the color blue. Make sure it's below your `pink-text` class declaration. Apply the `blue-text` class to your `h1` element in addition to your `pink-text` class, and let's see which one wins. Applying multiple class attributes to a HTML element is done with a space between them like this: `class="class1 class2"`. Note: It doesn't matter which order the classes are listed in the HTML element. However, the order of the `class` declarations in the `<style>` section are what is important. The second declaration will always take precedence over the first. Because `.blue-text` is declared second, it overrides the attributes of `.pink-text`.

Challenge Seed

```
<style>
  body {
    background-color: black;
    font-family: monospace;
    color: green;
  }
  .pink-text {
    color: pink;
  }
</style>
<h1 class="pink-text">Hello World!</h1>
```

Solution

```
// solution required
```

29. Override Class Declarations by Styling ID Attributes

Description

We just proved that browsers read CSS from top to bottom in order of their declaration. That means that, in the event of a conflict, the browser will use whichever CSS declaration came last. Notice that if we even had put `blue-text` before `pink-text` in our `h1` element's classes, it would still look at the declaration order and not the order of their use! But we're not done yet. There are other ways that you can override CSS. Do you remember id attributes? Let's

override your `pink-text` and `blue-text` classes, and make your `h1` element orange, by giving the `h1` element an `id` and then styling that `id`.

Instructions

Give your `h1` element the `id` attribute of `orange-text`. Remember, `id` styles look like this: `<h1 id="orange-text">` Leave the `blue-text` and `pink-text` classes on your `h1` element. Create a CSS declaration for your `orange-text` `id` in your `style` element. Here's an example of what this looks like:

```
#brown-text {
  color: brown;
}
```

Note: It doesn't matter whether you declare this CSS above or below `pink-text` class, since `id` attribute will always take precedence.

Challenge Seed

```
<style>
  body {
    background-color: black;
    font-family: monospace;
    color: green;
  }
  .pink-text {
    color: pink;
  }
  .blue-text {
    color: blue;
  }
</style>
<h1 class="pink-text blue-text">Hello World!</h1>
```

Solution

```
// solution required
```

30. Override Class Declarations with Inline Styles

Description

So we've proven that `id` declarations override class declarations, regardless of where they are declared in your `style` element CSS. There are other ways that you can override CSS. Do you remember inline styles?

Instructions

Use an `inline style` to try to make our `h1` element white. Remember, `in line styles` look like this: `<h1 style="color: green;">` Leave the `blue-text` and `pink-text` classes on your `h1` element.

Challenge Seed

```
<style>
  body {
    background-color: black;
    font-family: monospace;
    color: green;
  }
  #orange-text {
    color: orange;
  }
```

```
.pink-text {
  color: pink;
}
.blue-text {
  color: blue;
}
</style>
<h1 id="orange-text" class="pink-text blue-text">Hello World!</h1>
```

Solution

```
// solution required
```

31. Override All Other Styles by using Important

Description

Yay! We just proved that inline styles will override all the CSS declarations in your `style` element. But wait. There's one last way to override CSS. This is the most powerful method of all. But before we do it, let's talk about why you would ever want to override CSS. In many situations, you will use CSS libraries. These may accidentally override your own CSS. So when you absolutely need to be sure that an element has specific CSS, you can use `!important`. Let's go all the way back to our `pink-text` class declaration. Remember that our `pink-text` class was overridden by subsequent class declarations, id declarations, and inline styles.

Instructions

Let's add the keyword `!important` to your `pink-text` element's color declaration to make 100% sure that your `h1` element will be pink. An example of how to do this is: `color: red !important;`

Challenge Seed

```
<style>
  body {
    background-color: black;
    font-family: monospace;
    color: green;
  }
  #orange-text {
    color: orange;
  }
  .pink-text {
    color: pink;
  }
  .blue-text {
    color: blue;
  }
</style>
<h1 id="orange-text" class="pink-text blue-text" style="color: white">Hello World!</h1>
```

Solution

```
// solution required
```

32. Use Hex Code for Specific Colors

Description

Did you know there are other ways to represent colors in CSS? One of these ways is called hexadecimal code, or hex code for short. We usually use decimals, or base 10 numbers, which use the symbols 0 to 9 for each digit. Hexadecimals (or hex) are base 16 numbers. This means it uses sixteen distinct symbols. Like decimals, the symbols 0-9 represent the values zero to nine. Then A,B,C,D,E,F represent the values ten to fifteen. Altogether, 0 to F can represent a digit in hexadecimal, giving us 16 total possible values. You can find more information about [hexadecimal numbers here](#). In CSS, we can use 6 hexadecimal digits to represent colors, two each for the red (R), green (G), and blue (B) components. For example, #000000 is black and is also the lowest possible value. You can find more information about the [RGB color system here](#).

```
body {
  color: #000000;
}
```

Instructions

Replace the word `black` in our `body` element's `background-color` with its `hex` code representation, `#000000`.

Challenge Seed

```
<style>
  body {
    background-color: black;
  }
</style>
```

Solution

```
// solution required
```

33. Use Hex Code to Mix Colors

Description

To review, hex codes use 6 hexadecimal digits to represent colors, two each for red (R), green (G), and blue (B) components. From these three pure colors (red, green, and blue), we can vary the amounts of each to create over 16 million other colors! For example, orange is pure red, mixed with some green, and no blue. In hex code, this translates to being `#FFA500`. The digit `0` is the lowest number in hex code, and represents a complete absence of color. The digit `F` is the highest number in hex code, and represents the maximum possible brightness.

Instructions

Replace the color words in our `style` element with their correct hex codes.

Color	Hex Code
Dodger Blue	#1E90FF
Green	#00FF00
Orange	#FFA500
Red	#FF0000

Challenge Seed

```
<style>
  .red-text {
    color: black;
  }

```

```

.green-text {
  color: black;
}
.dodger-blue-text {
  color: black;
}
.orange-text {
  color: black;
}

```

</style>

```

<h1 class="red-text">I am red!</h1>

<h1 class="green-text">I am green!</h1>

<h1 class="dodger-blue-text">I am dodger blue!</h1>

<h1 class="orange-text">I am orange!</h1>

```

Solution

```
// solution required
```

34. Use Abbreviated Hex Code

Description

Many people feel overwhelmed by the possibilities of more than 16 million colors. And it's difficult to remember hex code. Fortunately, you can shorten it. For example, red's hex code `#FF0000` can be shortened to `#F00`. This shortened form gives one digit for red, one digit for green, and one digit for blue. This reduces the total number of possible colors to around 4,000. But browsers will interpret `#FF0000` and `#F00` as exactly the same color.

Instructions

Go ahead, try using the abbreviated hex codes to color the correct elements.

Color	Short Hex Code
Cyan	#0FF
Green	#0F0
Red	#F00
Fuchsia	#F0F

Challenge Seed

```

<style>
.red-text {
  color: #000000;
}
.fuchsia-text {
  color: #000000;
}
.cyan-text {
  color: #000000;
}
.green-text {
  color: #000000;
}

```

</style>

```

<h1 class="red-text">I am red!</h1>

```

```
<h1 class="fuchsia-text">I am fuchsia!</h1>
<h1 class="cyan-text">I am cyan!</h1>
<h1 class="green-text">I am green!</h1>
```

Solution

```
// solution required
```

35. Use RGB values to Color Elements

Description

Another way you can represent colors in CSS is by using `RGB` values. The `RGB` value for black looks like this: `rgb(0, 0, 0)`. The `RGB` value for white looks like this: `rgb(255, 255, 255)`. Instead of using six hexadecimal digits like you do with hex code, with `RGB` you specify the brightness of each color with a number between 0 and 255. If you do the math, the two digits for one color equal 16 times 16, which gives us 256 total values. So `RGB`, which starts counting from zero, has the exact same number of possible values as hex code. Here's an example of how you'd change the body background to orange using its `RGB` code.

```
body {
  background-color: rgb(255, 165, 0);
}
```

Instructions

Let's replace the hex code in our `body` element's background color with the `RGB` value for black: `rgb(0, 0, 0)`

Challenge Seed

```
<style>
  body {
    background-color: #F00;
  }
</style>
```

Solution

```
// solution required
```

36. Use RGB to Mix Colors

Description

Just like with hex code, you can mix colors in `RGB` by using combinations of different values.

Instructions

Replace the hex codes in our `style` element with their correct `RGB` values.

Color	RGB
Blue	<code>rgb(0, 0, 255)</code>

Red	rgb(255, 0, 0)
Orchid	rgb(218, 112, 214)
Sienna	rgb(160, 82, 45)

Challenge Seed

```
<style>
.red-text {
  color: #000000;
}
.orchid-text {
  color: #000000;
}
.sienna-text {
  color: #000000;
}
.blue-text {
  color: #000000;
}
</style>

<h1 class="red-text">I am red!</h1>
<h1 class="orchid-text">I am orchid!</h1>
<h1 class="sienna-text">I am sienna!</h1>
<h1 class="blue-text">I am blue!</h1>
```

Solution

```
<style>
.red-text {
  color: rgb(255, 0, 0);
}
.orchid-text {
  color: rgb(218, 112, 214);
}
.sienna-text {
  color: rgb(160, 82, 45);
}
.blue-text {
  color:rgb(0, 0, 255);
}
</style>

<h1 class="red-text">I am red!</h1>
<h1 class="orchid-text">I am orchid!</h1>
<h1 class="sienna-text">I am sienna!</h1>
<h1 class="blue-text">I am blue!</h1>
```

37. Use CSS Variables to change several elements at once

Description

CSS Variables are a powerful way to change many CSS style properties at once by changing only one value. Follow the instructions below to see how changing just three values can change the styling of many elements.

Instructions

In the penguin class, change the black value to gray , the gray value to white , and the yellow value to orange .

Challenge Seed

```
<style>
.penguin {

    /* change code below */
    --penguin-skin: black;
    --penguin-belly: gray;
    --penguin-beak: yellow;
    /* change code above */

    position: relative;
    margin: auto;
    display: block;
    margin-top: 5%;
    width: 300px;
    height: 300px;
}

.penguin-top {
    top: 10%;
    left: 25%;
    background: var(--penguin-skin, gray);
    width: 50%;
    height: 45%;
    border-radius: 70% 70% 60% 60%;
}

.penguin-bottom {
    top: 40%;
    left: 23.5%;
    background: var(--penguin-skin, gray);
    width: 53%;
    height: 45%;
    border-radius: 70% 70% 100% 100%;
}

.right-hand {
    top: 0%;
    left: -5%;
    background: var(--penguin-skin, gray);
    width: 30%;
    height: 60%;
    border-radius: 30% 30% 120% 30%;
    transform: rotate(45deg);
    z-index: -1;
}

.left-hand {
    top: 0%;
    left: 75%;
    background: var(--penguin-skin, gray);
    width: 30%;
    height: 60%;
    border-radius: 30% 30% 30% 120%;
    transform: rotate(-45deg);
    z-index: -1;
}

.right-cheek {
    top: 15%;
    left: 35%;
    background: var(--penguin-belly, white);
    width: 60%;
    height: 70%;
    border-radius: 70% 70% 60% 60%;
}

.left-cheek {
    top: 15%;
    left: 5%;
    background: var(--penguin-belly, white);
    width: 60%;
```

```
height: 70%;  
border-radius: 70% 70% 60% 60%;  
}  
  
.belly {  
top: 60%;  
left: 2.5%;  
background: var(--penguin-belly, white);  
width: 95%;  
height: 100%;  
border-radius: 120% 120% 100% 100%;  
}  
  
.right-feet {  
top: 85%;  
left: 60%;  
background: var(--penguin-beak, orange);  
width: 15%;  
height: 30%;  
border-radius: 50% 50% 50% 50%;  
transform: rotate(-80deg);  
z-index: -2222;  
}  
  
.left-feet {  
top: 85%;  
left: 25%;  
background: var(--penguin-beak, orange);  
width: 15%;  
height: 30%;  
border-radius: 50% 50% 50% 50%;  
transform: rotate(80deg);  
z-index: -2222;  
}  
  
.right-eye {  
top: 45%;  
left: 60%;  
background: black;  
width: 15%;  
height: 17%;  
border-radius: 50%;  
}  
  
.left-eye {  
top: 45%;  
left: 25%;  
background: black;  
width: 15%;  
height: 17%;  
border-radius: 50%;  
}  
  
.sparkle {  
top: 25%;  
left: 15%;  
background: white;  
width: 35%;  
height: 35%;  
border-radius: 50%;  
}  
  
.blush-right {  
top: 65%;  
left: 15%;  
background: pink;  
width: 15%;  
height: 10%;  
border-radius: 50%;  
}  
  
.blush-left {  
top: 65%;  
left: 70%;  
background: pink;  
width: 15%;  
height: 10%;  
}
```

```

    border-radius: 50%;
}

.beak-top {
  top: 60%;
  left: 40%;
  background: var(--penguin-beak, orange);
  width: 20%;
  height: 10%;
  border-radius: 50%;
}

.beak-bottom {
  top: 65%;
  left: 42%;
  background: var(--penguin-beak, orange);
  width: 16%;
  height: 10%;
  border-radius: 50%;
}

body {
  background:#c6faf1;
}

.penguin * {
  position: absolute;
}
</style>
<div class="penguin">
  <div class="penguin-bottom">
    <div class="right-hand"></div>
    <div class="left-hand"></div>
    <div class="right-feet"></div>
    <div class="left-feet"></div>
  </div>
  <div class="penguin-top">
    <div class="right-cheek"></div>
    <div class="left-cheek"></div>
    <div class="belly"></div>
    <div class="right-eye">
      <div class="sparkle"></div>
    </div>
    <div class="left-eye">
      <div class="sparkle"></div>
    </div>
    <div class="blush-right"></div>
    <div class="blush-left"></div>
    <div class="beak-top"></div>
    <div class="beak-bottom"></div>
  </div>
</div>

```

Solution

```
var code = ".penguin {--penguin-skin: gray; --penguin-belly: white; --penguin-beak: orange;}"
```

38. Create a custom CSS Variable

Description

To create a CSS Variable, you just need to give it a name with two dashes in front of it and assign it a value like this:

```
--penguin-skin: gray;
```

This will create a variable named `--penguin-skin` and assign it the value of `gray`. Now you can use that variable elsewhere in your CSS to change the value of other elements to `gray`.

Instructions

In the penguin class, create a variable name --penguin-skin and give it a value of gray

Challenge Seed

```
<style>
.penguin {

    /* add code below */

    /* add code above */
    position: relative;
    margin: auto;
    display: block;
    margin-top: 5%;
    width: 300px;
    height: 300px;
}

.penguin-top {
    top: 10%;
    left: 25%;
    background: black;
    width: 50%;
    height: 45%;
    border-radius: 70% 70% 60% 60%;
}

.penguin-bottom {
    top: 40%;
    left: 23.5%;
    background: black;
    width: 53%;
    height: 45%;
    border-radius: 70% 70% 100% 100%;
}

.right-hand {
    top: 0%;
    left: -5%;
    background: black;
    width: 30%;
    height: 60%;
    border-radius: 30% 30% 120% 30%;
    transform: rotate(45deg);
    z-index: -1;
}

.left-hand {
    top: 0%;
    left: 75%;
    background: black;
    width: 30%;
    height: 60%;
    border-radius: 30% 30% 30% 120%;
    transform: rotate(-45deg);
    z-index: -1;
}

.right-cheek {
    top: 15%;
    left: 35%;
    background: white;
    width: 60%;
    height: 70%;
    border-radius: 70% 70% 60% 60%;
}

.left-cheek {
    top: 15%;
    left: 5%;
    background: white;
    width: 60%;
```

```
height: 70%;  
border-radius: 70% 70% 60% 60%;  
}  
  
.belly {  
top: 60%;  
left: 2.5%;  
background: white;  
width: 95%;  
height: 100%;  
border-radius: 120% 120% 100% 100%;  
}  
  
.right-feet {  
top: 85%;  
left: 60%;  
background: orange;  
width: 15%;  
height: 30%;  
border-radius: 50% 50% 50% 50%;  
transform: rotate(-80deg);  
z-index: -2222;  
}  
  
.left-feet {  
top: 85%;  
left: 25%;  
background: orange;  
width: 15%;  
height: 30%;  
border-radius: 50% 50% 50% 50%;  
transform: rotate(80deg);  
z-index: -2222;  
}  
  
.right-eye {  
top: 45%;  
left: 60%;  
background: black;  
width: 15%;  
height: 17%;  
border-radius: 50%;  
}  
  
.left-eye {  
top: 45%;  
left: 25%;  
background: black;  
width: 15%;  
height: 17%;  
border-radius: 50%;  
}  
  
.sparkle {  
top: 25%;  
left: 15%;  
background: white;  
width: 35%;  
height: 35%;  
border-radius: 50%;  
}  
  
.blush-right {  
top: 65%;  
left: 15%;  
background: pink;  
width: 15%;  
height: 10%;  
border-radius: 50%;  
}  
  
.blush-left {  
top: 65%;  
left: 70%;  
background: pink;  
width: 15%;  
height: 10%;  
}
```

```

border-radius: 50%;

}

.beak-top {
  top: 60%;
  left: 40%;
  background: orange;
  width: 20%;
  height: 10%;
  border-radius: 50%;
}

.beak-bottom {
  top: 65%;
  left: 42%;
  background: orange;
  width: 16%;
  height: 10%;
  border-radius: 50%;
}

body {
  background:#c6faf1;
}

.penguin * {
  position: absolute;
}
</style>
<div class="penguin">
  <div class="penguin-bottom">
    <div class="right-hand"></div>
    <div class="left-hand"></div>
    <div class="right-feet"></div>
    <div class="left-feet"></div>
  </div>
  <div class="penguin-top">
    <div class="right-cheek"></div>
    <div class="left-cheek"></div>
    <div class="belly"></div>
    <div class="right-eye">
      <div class="sparkle"></div>
    </div>
    <div class="left-eye">
      <div class="sparkle"></div>
    </div>
    <div class="blush-right"></div>
    <div class="blush-left"></div>
    <div class="beak-top"></div>
    <div class="beak-bottom"></div>
  </div>
</div>

```

Solution

```
var code = ".penguin {--penguin-skin: gray;}"
```

39. Use a custom CSS Variable

Description

After you create your variable, you can assign its value to other CSS properties by referencing the name you gave it.

```
background: var(--penguin-skin);
```

This will change the background of whatever element you are targeting to gray because that is the value of the `--penguin-skin` variable. Note that styles will not be applied unless the variable names are an exact match.

Instructions

Apply the `--penguin-skin` variable to the `background` property of the `penguin-top`, `penguin-bottom`, `right-hand` and `left-hand` classes.

Challenge Seed

```
<style>
  .penguin {
    --penguin-skin: gray;
    position: relative;
    margin: auto;
    display: block;
    margin-top: 5%;
    width: 300px;
    height: 300px;
  }

  .penguin-top {
    top: 10%;
    left: 25%;

    /* change code below */
    background: black;
    /* change code above */

    width: 50%;
    height: 45%;
    border-radius: 70% 70% 60% 60%;
  }

  .penguin-bottom {
    top: 40%;
    left: 23.5%;

    /* change code below */
    background: black;
    /* change code above */

    width: 53%;
    height: 45%;
    border-radius: 70% 70% 100% 100%;
  }

  .right-hand {
    top: 0%;
    left: -5%;

    /* change code below */
    background: black;
    /* change code above */

    width: 30%;
    height: 60%;
    border-radius: 30% 30% 120% 30%;
    transform: rotate(45deg);
    z-index: -1;
  }

  .left-hand {
    top: 0%;
    left: 75%;

    /* change code below */
    background: black;
    /* change code above */

    width: 30%;
    height: 60%;
    border-radius: 30% 30% 30% 120%;
    transform: rotate(-45deg);
    z-index: -1;
  }
</style>
```

```
.right-cheek {  
    top: 15%;  
    left: 35%;  
    background: white;  
    width: 60%;  
    height: 70%;  
    border-radius: 70% 70% 60% 60%;  
}  
  
.left-cheek {  
    top: 15%;  
    left: 5%;  
    background: white;  
    width: 60%;  
    height: 70%;  
    border-radius: 70% 70% 60% 60%;  
}  
  
.belly {  
    top: 60%;  
    left: 2.5%;  
    background: white;  
    width: 95%;  
    height: 100%;  
    border-radius: 120% 120% 100% 100%;  
}  
  
.right-feet {  
    top: 85%;  
    left: 60%;  
    background: orange;  
    width: 15%;  
    height: 30%;  
    border-radius: 50% 50% 50% 50%;  
    transform: rotate(-80deg);  
    z-index: -2222;  
}  
  
.left-feet {  
    top: 85%;  
    left: 25%;  
    background: orange;  
    width: 15%;  
    height: 30%;  
    border-radius: 50% 50% 50% 50%;  
    transform: rotate(80deg);  
    z-index: -2222;  
}  
  
.right-eye {  
    top: 45%;  
    left: 60%;  
    background: black;  
    width: 15%;  
    height: 17%;  
    border-radius: 50%;  
}  
  
.left-eye {  
    top: 45%;  
    left: 25%;  
    background: black;  
    width: 15%;  
    height: 17%;  
    border-radius: 50%;  
}  
  
.sparkle {  
    top: 25%;  
    left: 15%;  
    background: white;  
    width: 35%;  
    height: 35%;  
    border-radius: 50%;  
}  
  
.blush-right {
```

```
top: 65%;  
left: 15%;  
background: pink;  
width: 15%;  
height: 10%;  
border-radius: 50%;  
}  
  
.blush-left {  
top: 65%;  
left: 70%;  
background: pink;  
width: 15%;  
height: 10%;  
border-radius: 50%;  
}  
  
.beak-top {  
top: 60%;  
left: 40%;  
background: orange;  
width: 20%;  
height: 10%;  
border-radius: 50%;  
}  
  
.beak-bottom {  
top: 65%;  
left: 42%;  
background: orange;  
width: 16%;  
height: 10%;  
border-radius: 50%;  
}  
  
body {  
background:#c6faf1;  
}  
  
.penguin * {  
position: absolute;  
}  
</style>  
<div class="penguin">  
 <div class="penguin-bottom">  
   <div class="right-hand"></div>  
   <div class="left-hand"></div>  
   <div class="right-feet"></div>  
   <div class="left-feet"></div>  
 </div>  
 <div class="penguin-top">  
   <div class="right-cheek"></div>  
   <div class="left-cheek"></div>  
   <div class="belly"></div>  
   <div class="right-eye">  
     <div class="sparkle"></div>  
   </div>  
   <div class="left-eye">  
     <div class="sparkle"></div>  
   </div>  
   <div class="blush-right"></div>  
   <div class="blush-left"></div>  
   <div class="beak-top"></div>  
   <div class="beak-bottom"></div>  
 </div>  
</div>
```

Solution

```
var code = ".penguin-top {background: var(--penguin-skin);} .penguin-bottom {background: var(--penguin-skin);} .right-hand {background: var(--penguin-skin);} .left-hand {background: var(--penguin-skin);}"
```

40. Attach a Fallback value to a CSS Variable

Description

When using your variable as a CSS property value, you can attach a fallback value that your browser will revert to if the given variable is invalid. **Note:** This fallback is not used to increase browser compatibility, and it will not work on IE browsers. Rather, it is used so that the browser has a color to display if it cannot find your variable. Here's how you do it:

```
background: var(--penguin-skin, black);
```

This will set background to black if your variable wasn't set. Note that this can be useful for debugging.

Instructions

It looks like there is a problem with the variables supplied to the `.penguin-top` and `.penguin-bottom` classes. Rather than fix the typo, add a fallback value of `black` to the `background` property of the `.penguin-top` and `.penguin-bottom` classes.

Challenge Seed

```
<style>
  .penguin {
    --penguin-skin: black;
    --penguin-belly: gray;
    --penguin-beak: yellow;
    position: relative;
    margin: auto;
    display: block;
    margin-top: 5%;
    width: 300px;
    height: 300px;
  }

  .penguin-top {
    top: 10%;
    left: 25%;

    /* change code below */
    background: var(--penguin-skin);
    /* change code above */

    width: 50%;
    height: 45%;
    border-radius: 70% 70% 60% 60%;
  }

  .penguin-bottom {
    top: 40%;
    left: 23.5%;

    /* change code below */
    background: var(--penguin-skin);
    /* change code above */

    width: 53%;
    height: 45%;
    border-radius: 70% 70% 100% 100%;
  }

  .right-hand {
    top: 0%;
    left: -5%;
    background: var(--penguin-skin, black);
    width: 30%;
    height: 60%;
    border-radius: 30% 30% 120% 30%;
    transform: rotate(45deg);
    z-index: -1;
  }
</style>
```

```
.left-hand {
  top: 0%;
  left: 75%;
  background: var(--penguin-skin, black);
  width: 30%;
  height: 60%;
  border-radius: 30% 30% 30% 120%;
  transform: rotate(-45deg);
  z-index: -1;
}

.right-cheek {
  top: 15%;
  left: 35%;
  background: var(--penguin-belly, white);
  width: 60%;
  height: 70%;
  border-radius: 70% 70% 60% 60%;
}

.left-cheek {
  top: 15%;
  left: 5%;
  background: var(--penguin-belly, white);
  width: 60%;
  height: 70%;
  border-radius: 70% 70% 60% 60%;
}

.belly {
  top: 60%;
  left: 2.5%;
  background: var(--penguin-belly, white);
  width: 95%;
  height: 100%;
  border-radius: 120% 120% 100% 100%;
}

.right-feet {
  top: 85%;
  left: 60%;
  background: var(--penguin-beak, orange);
  width: 15%;
  height: 30%;
  border-radius: 50% 50% 50% 50%;
  transform: rotate(-80deg);
  z-index: -2222;
}

.left-feet {
  top: 85%;
  left: 25%;
  background: var(--penguin-beak, orange);
  width: 15%;
  height: 30%;
  border-radius: 50% 50% 50% 50%;
  transform: rotate(80deg);
  z-index: -2222;
}

.right-eye {
  top: 45%;
  left: 60%;
  background: black;
  width: 15%;
  height: 17%;
  border-radius: 50%;
}

.left-eye {
  top: 45%;
  left: 25%;
  background: black;
  width: 15%;
  height: 17%;
  border-radius: 50%;
```

```
}

.sparkle {
  top: 25%;
  left: 15%;
  background: white;
  width: 35%;
  height: 35%;
  border-radius: 50%;
}

.blush-right {
  top: 65%;
  left: 15%;
  background: pink;
  width: 15%;
  height: 10%;
  border-radius: 50%;
}

.blush-left {
  top: 65%;
  left: 70%;
  background: pink;
  width: 15%;
  height: 10%;
  border-radius: 50%;
}

.beak-top {
  top: 60%;
  left: 40%;
  background: var(--penguin-beak, orange);
  width: 20%;
  height: 10%;
  border-radius: 50%;
}

.beak-bottom {
  top: 65%;
  left: 42%;
  background: var(--penguin-beak, orange);
  width: 16%;
  height: 10%;
  border-radius: 50%;
}

body {
  background:#c6faf1;
}

.penguin * {
  position: absolute;
}

</style>
<div class="penguin">
  <div class="penguin-bottom">
    <div class="right-hand"></div>
    <div class="left-hand"></div>
    <div class="right-feet"></div>
    <div class="left-feet"></div>
  </div>
  <div class="penguin-top">
    <div class="right-cheek"></div>
    <div class="left-cheek"></div>
    <div class="belly"></div>
    <div class="right-eye">
      <div class="sparkle"></div>
    </div>
    <div class="left-eye">
      <div class="sparkle"></div>
    </div>
    <div class="blush-right"></div>
    <div class="blush-left"></div>
    <div class="beak-top"></div>
    <div class="beak-bottom"></div>
```

```
</div>
</div>
```

Solution

```
var code = ".penguin-top {background: var(--pengiun-skin, black);} .penguin-bottom {background: var(--pengiun-skin, black);}"
```

41. Improve Compatibility with Browser Fallbacks

Description

When working with CSS you will likely run into browser compatibility issues at some point. This is why it's important to provide browser fallbacks to avoid potential problems. When your browser parses the CSS of a webpage, it ignores any properties that it doesn't recognize or support. For example, if you use a CSS variable to assign a background color on a site, Internet Explorer will ignore the background color because it does not support CSS variables. In that case, the browser will use whatever value it has for that property. If it can't find any other value set for that property, it will revert to the default value, which is typically not ideal. This means that if you do want to provide a browser fallback, it's as easy as providing another more widely supported value immediately before your declaration. That way an older browser will have something to fall back on, while a newer browser will just interpret whatever declaration comes later in the cascade.

Instructions

It looks like a variable is being used to set the background color of the `.red-box` class. Let's improve our browser compatibility by adding another `background` declaration right before the existing declaration and set its value to red.

Challenge Seed

```
<style>
:root {
  --red-color: red;
}
.red-box {

  background: var(--red-color);
  height: 200px;
  width: 200px;
}
</style>
<div class="red-box"></div>
```

Solution

```
var code=".red-box {background: red; background: var(--red-color);}"
```

42. Cascading CSS variables

Description

When you create a variable, it is available for you to use inside the element in which you create it. It also is available for any elements nested within it. This effect is known as cascading. Because of cascading, CSS variables are often defined in the `:root` element. `:root` is a pseudo-class selector that matches the root element of the document, usually

the `:root` element. By creating your variables in `:root`, they will be available globally and can be accessed from any other selector later in the style sheet.

Instructions

Define a variable named `--penguin-belly` in the `:root` selector and give it the value of `pink`. You can then see how the value will cascade down to change the value to pink, anywhere that variable is used.

Challenge Seed

```
<style>
:root {
    /* add code below */

    /* add code above */
}

body {
    background: var(--penguin-belly, #c6faf1);
}

.penguin {
    --penguin-skin: gray;
    --penguin-beak: orange;
    position: relative;
    margin: auto;
    display: block;
    margin-top: 5%;
    width: 300px;
    height: 300px;
}

.right-cheek {
    top: 15%;
    left: 35%;
    background: var(--penguin-belly, white);
    width: 60%;
    height: 70%;
    border-radius: 70% 70% 60% 60%;
}

.left-cheek {
    top: 15%;
    left: 5%;
    background: var(--penguin-belly, white);
    width: 60%;
    height: 70%;
    border-radius: 70% 70% 60% 60%;
}

.belly {
    top: 60%;
    left: 2.5%;
    background: var(--penguin-belly, white);
    width: 95%;
    height: 100%;
    border-radius: 120% 120% 100% 100%;
}

.penguin-top {
    top: 10%;
    left: 25%;
    background: var(--penguin-skin, gray);
    width: 50%;
    height: 45%;
    border-radius: 70% 70% 60% 60%;
}

.penguin-bottom {
    top: 40%;
    left: 23.5%;
    background: var(--penguin-skin, gray);
```

```
width: 53%;  
height: 45%;  
border-radius: 70% 70% 100% 100%;  
}  
  
.right-hand {  
top: 0%;  
left: -5%;  
background: var(--penguin-skin, gray);  
width: 30%;  
height: 60%;  
border-radius: 30% 30% 120% 30%;  
transform: rotate(45deg);  
z-index: -1;  
}  
  
.left-hand {  
top: 0%;  
left: 75%;  
background: var(--penguin-skin, gray);  
width: 30%;  
height: 60%;  
border-radius: 30% 30% 30% 120%;  
transform: rotate(-45deg);  
z-index: -1;  
}  
  
.right-feet {  
top: 85%;  
left: 60%;  
background: var(--penguin-beak, orange);  
width: 15%;  
height: 30%;  
border-radius: 50% 50% 50% 50%;  
transform: rotate(-80deg);  
z-index: -2222;  
}  
  
.left-feet {  
top: 85%;  
left: 25%;  
background: var(--penguin-beak, orange);  
width: 15%;  
height: 30%;  
border-radius: 50% 50% 50% 50%;  
transform: rotate(80deg);  
z-index: -2222;  
}  
  
.right-eye {  
top: 45%;  
left: 60%;  
background: black;  
width: 15%;  
height: 17%;  
border-radius: 50%;  
}  
  
.left-eye {  
top: 45%;  
left: 25%;  
background: black;  
width: 15%;  
height: 17%;  
border-radius: 50%;  
}  
  
.sparkle {  
top: 25%;  
left: 15%;  
background: white;  
width: 35%;  
height: 35%;  
border-radius: 50%;  
}  
  
.blush-right {
```

```
top: 65%;  
left: 15%;  
background: pink;  
width: 15%;  
height: 10%;  
border-radius: 50%;  
}  
  
.blush-left {  
top: 65%;  
left: 70%;  
background: pink;  
width: 15%;  
height: 10%;  
border-radius: 50%;  
}  
  
.beak-top {  
top: 60%;  
left: 40%;  
background: var(--penguin-beak, orange);  
width: 20%;  
height: 10%;  
border-radius: 50%;  
}  
  
.beak-bottom {  
top: 65%;  
left: 42%;  
background: var(--penguin-beak, orange);  
width: 16%;  
height: 10%;  
border-radius: 50%;  
}  
  
.penguin * {  
position: absolute;  
}  
/>
```

```
</style>  
<div class="penguin">  
  <div class="penguin-bottom">  
    <div class="right-hand"></div>  
    <div class="left-hand"></div>  
    <div class="right-feet"></div>  
    <div class="left-feet"></div>  
  </div>  
  <div class="penguin-top">  
    <div class="right-cheek"></div>  
    <div class="left-cheek"></div>  
    <div class="belly"></div>  
    <div class="right-eye">  
      <div class="sparkle"></div>  
    </div>  
    <div class="left-eye">  
      <div class="sparkle"></div>  
    </div>  
    <div class="blush-right"></div>  
    <div class="blush-left"></div>  
    <div class="beak-top"></div>  
    <div class="beak-bottom"></div>  
  </div>  
</div>
```

Solution

```
var code = ":root {--penguin-belly: pink;}"
```

43. Change a variable for a specific area

Description

When you create your variables in `:root` they will set the value of that variable for the whole page. You can then over-write these variables by setting them again within a specific element.

Instructions

Change the value of `--penguin-belly` to `white` in the `penguin` class.

Challenge Seed

```
<style>
  :root {
    --penguin-skin: gray;
    --penguin-belly: pink;
    --penguin-beak: orange;
  }

  body {
    background: var(--penguin-belly, #c6faf1);
  }

  .penguin {

    /* add code below */

    /* add code above */

    position: relative;
    margin: auto;
    display: block;
    margin-top: 5%;
    width: 300px;
    height: 300px;
  }

  .right-cheek {
    top: 15%;
    left: 35%;
    background: var(--penguin-belly, pink);
    width: 60%;
    height: 70%;
    border-radius: 70% 70% 60% 60%;
  }

  .left-cheek {
    top: 15%;
    left: 5%;
    background: var(--penguin-belly, pink);
    width: 60%;
    height: 70%;
    border-radius: 70% 70% 60% 60%;
  }

  .belly {
    top: 60%;
    left: 2.5%;
    background: var(--penguin-belly, pink);
    width: 95%;
    height: 100%;
    border-radius: 120% 120% 100% 100%;
  }

  .penguin-top {
    top: 10%;
    left: 25%;
    background: var(--penguin-skin, gray);
    width: 50%;
    height: 45%;
    border-radius: 70% 70% 60% 60%;
  }
```

```
.penguin-bottom {
  top: 40%;
  left: 23.5%;
  background: var(--penguin-skin, gray);
  width: 53%;
  height: 45%;
  border-radius: 70% 70% 100% 100%;
}

.right-hand {
  top: 0%;
  left: -5%;
  background: var(--penguin-skin, gray);
  width: 30%;
  height: 60%;
  border-radius: 30% 30% 120% 30%;
  transform: rotate(45deg);
  z-index: -1;
}

.left-hand {
  top: 0%;
  left: 75%;
  background: var(--penguin-skin, gray);
  width: 30%;
  height: 60%;
  border-radius: 30% 30% 30% 120%;
  transform: rotate(-45deg);
  z-index: -1;
}

.right-feet {
  top: 85%;
  left: 60%;
  background: var(--penguin-beak, orange);
  width: 15%;
  height: 30%;
  border-radius: 50% 50% 50% 50%;
  transform: rotate(-80deg);
  z-index: -2222;
}

.left-feet {
  top: 85%;
  left: 25%;
  background: var(--penguin-beak, orange);
  width: 15%;
  height: 30%;
  border-radius: 50% 50% 50% 50%;
  transform: rotate(80deg);
  z-index: -2222;
}

.right-eye {
  top: 45%;
  left: 60%;
  background: black;
  width: 15%;
  height: 17%;
  border-radius: 50%;
}

.left-eye {
  top: 45%;
  left: 25%;
  background: black;
  width: 15%;
  height: 17%;
  border-radius: 50%;
}

.sparkle {
  top: 25%;
  left: 15%;
  background: white;
  width: 35%;
  height: 35%;
```

```
border-radius: 50%;  
}  
  
.blush-right {  
  top: 65%;  
  left: 15%;  
  background: pink;  
  width: 15%;  
  height: 10%;  
  border-radius: 50%;  
}  
  
.blush-left {  
  top: 65%;  
  left: 70%;  
  background: pink;  
  width: 15%;  
  height: 10%;  
  border-radius: 50%;  
}  
  
.beak-top {  
  top: 60%;  
  left: 40%;  
  background: var(--penguin-beak, orange);  
  width: 20%;  
  height: 10%;  
  border-radius: 50%;  
}  
  
.beak-bottom {  
  top: 65%;  
  left: 42%;  
  background: var(--penguin-beak, orange);  
  width: 16%;  
  height: 10%;  
  border-radius: 50%;  
}  
  
.penguin * {  
  position: absolute;  
}  
/<style>  
<div class="penguin">  
  <div class="penguin-bottom">  
    <div class="right-hand"></div>  
    <div class="left-hand"></div>  
    <div class="right-feet"></div>  
    <div class="left-feet"></div>  
  </div>  
  <div class="penguin-top">  
    <div class="right-cheek"></div>  
    <div class="left-cheek"></div>  
    <div class="belly"></div>  
    <div class="right-eye">  
      <div class="sparkle"></div>  
    </div>  
    <div class="left-eye">  
      <div class="sparkle"></div>  
    </div>  
    <div class="blush-right"></div>  
    <div class="blush-left"></div>  
    <div class="beak-top"></div>  
    <div class="beak-bottom"></div>  
  </div>  
</div>
```

Solution

```
var code = ".penguin {--penguin-belly: white;}"
```

44. Use a media query to change a variable

Description

CSS Variables can simplify the way you use media queries. For instance, when your screen is smaller or larger than your media query break point, you can change the value of a variable, and it will apply its style wherever it is used.

Instructions

In the `:root` selector of the `media query`, change it so `--penguin-size` is redefined and given a value of `200px`. Also, redefine `--penguin-skin` and give it a value of `black`. Then resize the preview to see this change in action.

Challenge Seed

```
<style>
:root {
  --penguin-size: 300px;
  --penguin-skin: gray;
  --penguin-belly: white;
  --penguin-beak: orange;
}

@media (max-width: 350px) {
  :root {

    /* add code below */

    /* add code above */

  }
}

.penguin {
  position: relative;
  margin: auto;
  display: block;
  margin-top: 5%;
  width: var(--penguin-size, 300px);
  height: var(--penguin-size, 300px);
}

.right-cheek {
  top: 15%;
  left: 35%;
  background: var(--penguin-belly, white);
  width: 60%;
  height: 70%;
  border-radius: 70% 70% 60% 60%;
}

.left-cheek {
  top: 15%;
  left: 5%;
  background: var(--penguin-belly, white);
  width: 60%;
  height: 70%;
  border-radius: 70% 70% 60% 60%;
}

.belly {
  top: 60%;
  left: 2.5%;
  background: var(--penguin-belly, white);
  width: 95%;
  height: 100%;
  border-radius: 120% 120% 100% 100%;
}

.penguin-top {
  top: 10%;
```

```
left: 25%;  
background: var(--penguin-skin, gray);  
width: 50%;  
height: 45%;  
border-radius: 70% 70% 60% 60%;  
}  
  
.penguin-bottom {  
top: 40%;  
left: 23.5%;  
background: var(--penguin-skin, gray);  
width: 53%;  
height: 45%;  
border-radius: 70% 70% 100% 100%;  
}  
  
.right-hand {  
top: 5%;  
left: 25%;  
background: var(--penguin-skin, black);  
width: 30%;  
height: 60%;  
border-radius: 30% 30% 120% 30%;  
transform: rotate(130deg);  
z-index: -1;  
animation-duration: 3s;  
animation-name: wave;  
animation-iteration-count: infinite;  
transform-origin: 0% 0%;  
animation-timing-function: linear;  
}  
  
 @keyframes wave {  
 10% {  
    transform: rotate(110deg);  
  }  
 20% {  
    transform: rotate(130deg);  
  }  
 30% {  
    transform: rotate(110deg);  
  }  
 40% {  
    transform: rotate(130deg);  
  }  
}  
  
.left-hand {  
top: 0%;  
left: 75%;  
background: var(--penguin-skin, gray);  
width: 30%;  
height: 60%;  
border-radius: 30% 30% 30% 120%;  
transform: rotate(-45deg);  
z-index: -1;  
}  
  
.right-feet {  
top: 85%;  
left: 60%;  
background: var(--penguin-beak, orange);  
width: 15%;  
height: 30%;  
border-radius: 50% 50% 50% 50%;  
transform: rotate(-80deg);  
z-index: -2222;  
}  
  
.left-feet {  
top: 85%;  
left: 25%;  
background: var(--penguin-beak, orange);  
width: 15%;  
height: 30%;  
border-radius: 50% 50% 50% 50%;  
transform: rotate(80deg);  
}
```

```
z-index: -2222;
}

.right-eye {
  top: 45%;
  left: 60%;
  background: black;
  width: 15%;
  height: 17%;
  border-radius: 50%;
}

.left-eye {
  top: 45%;
  left: 25%;
  background: black;
  width: 15%;
  height: 17%;
  border-radius: 50%;
}

.sparkle {
  top: 25%;
  left:-23%;
  background: white;
  width: 150%;
  height: 100%;
  border-radius: 50%;
}

.blush-right {
  top: 65%;
  left: 15%;
  background: pink;
  width: 15%;
  height: 10%;
  border-radius: 50%;
}

.blush-left {
  top: 65%;
  left: 70%;
  background: pink;
  width: 15%;
  height: 10%;
  border-radius: 50%;
}

.beak-top {
  top: 60%;
  left: 40%;
  background: var(--penguin-beak, orange);
  width: 20%;
  height: 10%;
  border-radius: 50%;
}

.beak-bottom {
  top: 65%;
  left: 42%;
  background: var(--penguin-beak, orange);
  width: 16%;
  height: 10%;
  border-radius: 50%;
}

body {
  background:#c6faf1;
}

.penguin * {
  position: absolute;
}
</style>
<div class="penguin">
  <div class="penguin-bottom">
    <div class="right-hand"></div>
```

```

<div class="left-hand"></div>
<div class="right-feet"></div>
<div class="left-feet"></div>
</div>
<div class="penguin-top">
  <div class="right-cheek"></div>
  <div class="left-cheek"></div>
  <div class="belly"></div>
  <div class="right-eye">
    <div class="sparkle"></div>
  </div>
  <div class="left-eye">
    <div class="sparkle"></div>
  </div>
  <div class="blush-right"></div>
  <div class="blush-left"></div>
  <div class="beak-top"></div>
  <div class="beak-bottom"></div>
</div>
</div>

```

Solution

```
var code = "@media (max-width: 350px) {:root {--penguin-size: 200px; --penguin-skin: black;}}"
```

Applied Visual Design

1. Create Visual Balance Using the text-align Property

Description

This section of the curriculum focuses on Applied Visual Design. The first group of challenges build on the given card layout to show a number of core principles. Text is often a large part of web content. CSS has several options for how to align it with the `text-align` property. `text-align: justify;` causes all lines of text except the last line to meet the left and right edges of the line box. `text-align: center;` centers the text. `text-align: right;` right-aligns the text. And `text-align: left;` (the default) left-aligns the text.

Instructions

Align the `h4` tag's text, which says "Google", to the center. Then justify the paragraph tag which contains information about how Google was founded.

Challenge Seed

```

<style>
  h4 {
  }

  p {
  }

  .links {
    margin-right: 20px;
  }

  .fullCard {
    border: 1px solid #ccc;
    border-radius: 5px;
    margin: 10px 5px;
    padding: 4px;
  }

  .cardContent {
  }

```

```
padding: 10px;
}
</style>
<div class="fullCard">
  <div class="cardContent">
    <div class="cardText">
      <h4>Google</h4>
      <p>Google was founded by Larry Page and Sergey Brin while they were Ph.D. students at Stanford University.</p>
    </div>
    <div class="cardLinks">
      <a href="https://en.wikipedia.org/wiki/Larry_Page" target="_blank" class="links">Larry Page</a>
      <a href="https://en.wikipedia.org/wiki/Sergey_Brin" target="_blank" class="links">Sergey Brin</a>
    </div>
  </div>
</div>
```

Solution

```
// solution required
```

2. Adjust the Width of an Element Using the width Property

Description

You can specify the width of an element using the `width` property in CSS. Values can be given in relative length units (such as em), absolute length units (such as px), or as a percentage of its containing parent element. Here's an example that changes the width of an image to 220px:

```
img {
  width: 220px;
}
```

Instructions

Add a `width` property to the entire card and set it to an absolute value of 245px. Use the `fullCard` class to select the element.

Challenge Seed

```
<style>
  h4 {
    text-align: center;
  }
  p {
    text-align: justify;
  }
  .links {
    margin-right: 20px;
    text-align: left;
  }
  .fullCard {
    border: 1px solid #ccc;
    border-radius: 5px;
    margin: 10px 5px;
    padding: 4px;
  }
  .cardContent {
    padding: 10px;
  }
</style>
<div class="fullCard">
```

```
<div class="cardContent">
  <div class="cardText">
    <h4>Google</h4>
    <p>Google was founded by Larry Page and Sergey Brin while they were Ph.D. students at Stanford University.</p>
  </div>
  <div class="cardLinks">
    <a href="https://en.wikipedia.org/wiki/Larry_Page" target="_blank" class="links">Larry Page</a>
    <a href="https://en.wikipedia.org/wiki/Sergey_Brin" target="_blank" class="links">Sergey Brin</a>
  </div>
</div>
</div>
```

Solution

```
<style>
  h4 {
    text-align: center;
  }
  p {
    text-align: justify;
  }
  .links {
    margin-right: 20px;
    text-align: left;
  }
  .fullCard {
    width: 245px;
    border: 1px solid #ccc;
    border-radius: 5px;
    margin: 10px 5px;
    padding: 4px;
  }
  .cardContent {
    padding: 10px;
  }
</style>
<div class="fullCard">
  <div class="cardContent">
    <div class="cardText">
      <h4>Google</h4>
      <p>Google was founded by Larry Page and Sergey Brin while they were Ph.D. students at Stanford University.</p>
    </div>
    <div class="cardLinks">
      <a href="https://en.wikipedia.org/wiki/Larry_Page" target="_blank" class="links">Larry Page</a>
      <a href="https://en.wikipedia.org/wiki/Sergey_Brin" target="_blank" class="links">Sergey Brin</a>
    </div>
  </div>
</div>
```

3. Adjust the Height of an Element Using the height Property

Description

You can specify the height of an element using the `height` property in CSS, similar to the `width` property. Here's an example that changes the height of an image to 20px:

```
img {
  height: 20px;
}
```

Instructions

Add a `height` property to the `h4` tag and set it to 25px.

Challenge Seed

```

<style>
  h4 {
    text-align: center;
  }
  p {
    text-align: justify;
  }
  .links {
    margin-right: 20px;
    text-align: left;
  }
  .fullCard {
    width: 245px;
    border: 1px solid #ccc;
    border-radius: 5px;
    margin: 10px 5px;
    padding: 4px;
  }
  .CardContent {
    padding: 10px;
  }
</style>
<div class="fullCard">
  <div class="CardContent">
    <div class="cardText">
      <h4>Google</h4>
      <p>Google was founded by Larry Page and Sergey Brin while they were Ph.D. students at Stanford University.</p>
    </div>
    <div class="cardLinks">
      <a href="https://en.wikipedia.org/wiki/Larry_Page" target="_blank" class="links">Larry Page</a>
      <a href="https://en.wikipedia.org/wiki/Sergey_Brin" target="_blank" class="links">Sergey Brin</a>
    </div>
  </div>
</div>

```

Solution

```
// solution required
```

4. Use the strong Tag to Make Text Bold

Description

To make text bold, you can use the `strong` tag. This is often used to draw attention to text and symbolize that it is important. With the `strong` tag, the browser applies the CSS of `font-weight: bold;` to the element.

Instructions

Wrap a `strong` tag around "Stanford University" inside the `p` tag (do not include the period).

Challenge Seed

```

<style>
  h4 {
    text-align: center;
    height: 25px;
  }
  p {
    text-align: justify;
  }

```

```

}
.links {
  text-align: left;
  color: black;
}
.fullCard {
  width: 245px;
  border: 1px solid #ccc;
  border-radius: 5px;
  margin: 10px 5px;
  padding: 4px;
}
.cardContent {
  padding: 10px;
}
.cardText {
  margin-bottom: 30px;
}
</style>
<div class="fullCard">
  <div class="cardContent">
    <div class="cardText">
      <h4>Google</h4>
      <p>Google was founded by Larry Page and Sergey Brin while they were Ph.D. students at Stanford University.</p>
    </div>
    <div class="cardLinks">
      <a href="https://en.wikipedia.org/wiki/Larry_Page" target="_blank" class="links">Larry Page</a>
    <br><br>
      <a href="https://en.wikipedia.org/wiki/Sergey_Brin" target="_blank" class="links">Sergey Brin</a>
    </div>
  </div>
</div>

```

Solution

```
// solution required
```

5. Use the u Tag to Underline Text

Description

To underline text, you can use the `u` tag. This is often used to signify that a section of text is important, or something to remember. With the `u` tag, the browser applies the CSS of `text-decoration: underline;` to the element.

Instructions

Wrap the `u` tag only around the text "Ph.D. students". **Note**

Try to avoid using the `u` tag when it could be confused for a link. Anchor tags also have a default underlined formatting.

Challenge Seed

```

<style>
  h4 {
    text-align: center;
    height: 25px;
  }
  p {
    text-align: justify;
  }
  .links {
    text-align: left;
    color: black;
  }

```

```

.fullCard {
  width: 245px;
  border: 1px solid #ccc;
  border-radius: 5px;
  margin: 10px 5px;
  padding: 4px;
}
.cardContent {
  padding: 10px;
}
.cardText {
  margin-bottom: 30px;
}
</style>
<div class="fullCard">
  <div class="cardContent">
    <div class="cardText">
      <h4>Google</h4>
      <p>Google was founded by Larry Page and Sergey Brin while they were Ph.D. students at
      <strong>Stanford University</strong>.</p>
      </div>
      <div class="cardLinks">
        <a href="https://en.wikipedia.org/wiki/Larry_Page" target="_blank" class="links">Larry Page</a>
      <br><br>
        <a href="https://en.wikipedia.org/wiki/Sergey_Brin" target="_blank" class="links">Sergey Brin</a>
      </div>
      </div>
    </div>
  </div>

```

Solution

```
// solution required
```

6. Use the em Tag to Italicize Text

Description

To emphasize text, you can use the `em` tag. This displays text as italicized, as the browser applies the CSS of `font-style: italic;` to the element.

Instructions

Wrap an `em` tag around the contents of the paragraph tag to give it emphasis.

Challenge Seed

```

<style>
  h4 {
    text-align: center;
    height: 25px;
  }
  p {
    text-align: justify;
  }
  .links {
    text-align: left;
    color: black;
  }
  .fullCard {
    width: 245px;
    border: 1px solid #ccc;
    border-radius: 5px;
    margin: 10px 5px;
    padding: 4px;
  }

```

```

.cardContent {
  padding: 10px;
}
.cardText {
  margin-bottom: 30px;
}
</style>
<div class="fullCard">
  <div class="cardContent">
    <div class="cardText">
      <h4>Google</h4>
      <p>Google was founded by Larry Page and Sergey Brin while they were <u>Ph.D. students</u> at
      <strong>Stanford University</strong>.</p>
    </div>
    <div class="cardLinks">
      <a href="https://en.wikipedia.org/wiki/Larry_Page" target="_blank" class="links">Larry Page</a>
    <br><br>
      <a href="https://en.wikipedia.org/wiki/Sergey_Brin" target="_blank" class="links">Sergey Brin</a>
    </div>
  </div>
</div>

```

Solution

```
// solution required
```

7. Use the s Tag to Strikethrough Text

Description

To strikethrough text, which is when a horizontal line cuts across the characters, you can use the `s` tag. It shows that a section of text is no longer valid. With the `s` tag, the browser applies the CSS of `text-decoration: line-through;` to the element.

Instructions

Wrap the `s` tag around "Google" inside the `h4` tag and then add the word Alphabet beside it, which should not have the strikethrough formatting.

Challenge Seed

```

<style>
  h4 {
    text-align: center;
    height: 25px;
  }
  p {
    text-align: justify;
  }
  .links {
    text-align: left;
    color: black;
  }
  .fullCard {
    width: 245px;
    border: 1px solid #ccc;
    border-radius: 5px;
    margin: 10px 5px;
    padding: 4px;
  }
  .cardContent {
    padding: 10px;
  }
  .cardText {
    margin-bottom: 30px;
  }
</style>
<div class="fullCard">
  <div class="cardContent">
    <div class="cardText">
      <h4>Google</h4>
      <p>Google was founded by Larry Page and Sergey Brin while they were <u>Ph.D. students</u> at
      <strong>Stanford University</strong>.</p>
    </div>
    <div class="cardLinks">
      <a href="https://en.wikipedia.org/wiki/Larry_Page" target="_blank" class="links">Larry Page</a>
    <br><br>
      <a href="https://en.wikipedia.org/wiki/Sergey_Brin" target="_blank" class="links">Sergey Brin</a>
    </div>
  </div>
</div>

```

```

        }
    </style>
<div class="fullCard">
    <div class="cardContent">
        <div class="cardText">
            <h4>Google</h4>
            <p><em>Google was founded by Larry Page and Sergey Brin while they were <u>Ph.D. students</u> at <strong>Stanford University</strong>.</em></p>
        </div>
        <div class="cardLinks">
            <a href="https://en.wikipedia.org/wiki/Larry_Page" target="_blank" class="links">Larry Page</a>
        <br><br>
            <a href="https://en.wikipedia.org/wiki/Sergey_Brin" target="_blank" class="links">Sergey Brin</a>
        </div>
    </div>
</div>

```

Solution

```

<style>
    h4 {
        text-align: center;
        height: 25px;
    }
    p {
        text-align: justify;
    }
    .links {
        text-align: left;
        color: black;
    }
    .fullCard {
        width: 245px;
        border: 1px solid #ccc;
        border-radius: 5px;
        margin: 10px 5px;
        padding: 4px;
    }
    .cardContent {
        padding: 10px;
    }
    .cardText {
        margin-bottom: 30px;
    }
</style>
<div class="fullCard">
    <div class="cardContent">
        <div class="cardText">
            <h4><s>Google</s> Alphabet</h4>
            <p><em>Google was founded by Larry Page and Sergey Brin while they were <u>Ph.D. students</u> at <strong>Stanford University</strong>.</em></p>
        </div>
        <div class="cardLinks">
            <a href="https://en.wikipedia.org/wiki/Larry_Page" target="_blank" class="links">Larry Page</a>
        <br><br>
            <a href="https://en.wikipedia.org/wiki/Sergey_Brin" target="_blank" class="links">Sergey Brin</a>
        </div>
    </div>
</div>

```

8. Create a Horizontal Line Using the hr Element

Description

You can use the `hr` tag to add a horizontal line across the width of its containing element. This can be used to define a change in topic or to visually separate groups of content.

Instructions

Add an `hr` tag underneath the `h4` which contains the card title. **Note**

In HTML, `hr` is a self-closing tag, and therefore doesn't need a separate closing tag.

Challenge Seed

```

<style>
  h4 {
    text-align: center;
    height: 25px;
  }
  p {
    text-align: justify;
  }
  .links {
    text-align: left;
    color: black;
  }
  .fullCard {
    width: 245px;
    border: 1px solid #ccc;
    border-radius: 5px;
    margin: 10px 5px;
    padding: 4px;
  }
  .CardContent {
    padding: 10px;
  }
  .cardText {
    margin-bottom: 30px;
  }
</style>
<div class="fullCard">
  <div class="CardContent">
    <div class="cardText">
      <h4><s>Google</s>Alphabet</h4>

      <p><em>Google was founded by Larry Page and Sergey Brin while they were <u>Ph.D. students</u> at
      <strong>Stanford University</strong>.</em></p>
    </div>
    <div class="cardLinks">
      <a href="https://en.wikipedia.org/wiki/Larry_Page" target="_blank" class="links">Larry Page</a>
    <br><br>
      <a href="https://en.wikipedia.org/wiki/Sergey_Brin" target="_blank" class="links">Sergey Brin</a>
    </div>
    </div>
  </div>

```

Solution

```
// solution required
```

9. Adjust the background-color Property of Text

Description

Instead of adjusting your overall background or the color of the text to make the foreground easily readable, you can add a `background-color` to the element holding the text you want to emphasize. This challenge uses `rgba()` instead of hex codes or normal `rgb()`.

`rgba` stands for:

r = red

g = green

b = blue

a = alpha/level of opacity

The RGB values can range from 0 to 255. The alpha value can range from 1, which is fully opaque or a solid color, to 0, which is fully transparent or clear. `rgba()` is great to use in this case, as it allows you to adjust the opacity. This means you don't have to completely block out the background. You'll use `background-color: rgba(45, 45, 45, 0.1)` for this challenge. It produces a dark gray color that is nearly transparent given the low opacity value of 0.1.

Instructions

To make the text stand out more, adjust the `background-color` of the `h4` element to the given `rgba()` value. Also for the `h4`, remove the `height` property and add padding of 10px.

Challenge Seed

```
<style>
  h4 {
    text-align: center;
    height: 25px;
  }

  p {
    text-align: justify;
  }

  .links {
    text-align: left;
    color: black;
  }

  .fullCard {
    width: 245px;
    border: 1px solid #ccc;
    border-radius: 5px;
    margin: 10px 5px;
    padding: 4px;
  }

  .CardContent {
    padding: 10px;
  }

  .cardText {
    margin-bottom: 30px;
  }
</style>
<div class="fullCard">
  <div class="CardContent">
    <div class="cardText">
      <h4>Alphabet</h4>
      <hr>
      <p><em>Google was founded by Larry Page and Sergey Brin while they were <u>Ph.D. students</u> at Stanford University</em>.</p>
    </div>
    <div class="cardLinks">
      <a href="https://en.wikipedia.org/wiki/Larry_Page" target="_blank" class="links">Larry Page</a>
    <br><br>
      <a href="https://en.wikipedia.org/wiki/Sergey_Brin" target="_blank" class="links">Sergey Brin</a>
    </div>
  </div>
</div>
```

Solution

```
// solution required
```

10. Adjust the Size of a Header Versus a Paragraph Tag

Description

The font size of header tags (`h1` through `h6`) should generally be larger than the font size of paragraph tags. This makes it easier for the user to visually understand the layout and level of importance of everything on the page. You use the `font-size` property to adjust the size of the text in an element.

Instructions

To make the heading significantly larger than the paragraph, change the `font-size` of the `h4` tag to 27 pixels.

Challenge Seed

```
<style>
  h4 {
    text-align: center;
    background-color: rgba(45, 45, 45, 0.1);
    padding: 10px;
  }
  p {
    text-align: justify;
  }
  .links {
    text-align: left;
    color: black;
  }
  .fullCard {
    width: 245px;
    border: 1px solid #ccc;
    border-radius: 5px;
    margin: 10px 5px;
    padding: 4px;
  }
  .cardContent {
    padding: 10px;
  }
  .cardText {
    margin-bottom: 30px;
  }
</style>
<div class="fullCard">
  <div class="cardContent">
    <div class="cardText">
      <h4>Alphabet</h4>
      <br>
      <p><em>Google was founded by Larry Page and Sergey Brin while they were <u>Ph.D. students</u> at <strong>Stanford University</strong>.</em></p>
    </div>
    <div class="cardLinks">
      <a href="https://en.wikipedia.org/wiki/Larry_Page" target="_blank" class="links">Larry Page</a>
      <br><br>
      <a href="https://en.wikipedia.org/wiki/Sergey_Brin" target="_blank" class="links">Sergey Brin</a>
    </div>
  </div>
</div>
```

Solution

```
// solution required
```

11. Add a box-shadow to a Card-like Element

Description

The `box-shadow` property applies one or more shadows to an element. The `box-shadow` property takes values for `offset-x` (how far to push the shadow horizontally from the element), `offset-y` (how far to push the shadow

vertically from the element), `blur-radius`, `spread-radius` and a color value, in that order. The `blur-radius` and `spread-radius` values are optional. Here's an example of the CSS to create multiple shadows with some blur, at mostly-transparent black colors:

```
box-shadow: 0 10px 20px rgba(0,0,0,0.19), 0 6px 6px rgba(0,0,0,0.23);
```

Instructions

The element now has an id of `thumbnail`. With this selector, use the example CSS values above to place a `box-shadow` on the card.

Challenge Seed

```
<style>
  h4 {
    text-align: center;
    background-color: rgba(45, 45, 45, 0.1);
    padding: 10px;
    font-size: 27px;
  }
  p {
    text-align: justify;
  }
  .links {
    text-align: left;
    color: black;
  }

  .fullCard {
    width: 245px;
    border: 1px solid #ccc;
    border-radius: 5px;
    margin: 10px 5px;
    padding: 4px;
  }
  .CardContent {
    padding: 10px;
  }
  .cardText {
    margin-bottom: 30px;
  }
</style>
<div class="fullCard" id="thumbnail">
  <div class="CardContent">
    <div class="cardText">
      <h4>Alphabet</h4>
      <hr>
      <p><em>Google was founded by Larry Page and Sergey Brin while they were <u>Ph.D. students</u> at <strong>Stanford University</strong>.</em></p>
    </div>
    <div class="cardLinks">
      <a href="https://en.wikipedia.org/wiki/Larry_Page" target="_blank" class="links">Larry Page</a>
    <br><br>
      <a href="https://en.wikipedia.org/wiki/Sergey_Brin" target="_blank" class="links">Sergey Brin</a>
    </div>
    </div>
  </div>
</div>
```

Solution

```
var code = "#thumbnail {box-shadow: 0 10px 20px rgba(0,0,0,0.19), 0 6px 6px rgba(0,0,0,0.23);}"
```

12. Decrease the Opacity of an Element

Description

The `opacity` property in CSS is used to adjust the opacity, or conversely, the transparency for an item.

A value of 1 is opaque, which isn't transparent at all.

A value of 0.5 is half see-through.

A value of 0 is completely transparent.

The value given will apply to the entire element, whether that's an image with some transparency, or the foreground and background colors for a block of text.

Instructions

Set the `opacity` of the anchor tags to 0.7 using `links` class to select them.

Challenge Seed

```
<style>
  h4 {
    text-align: center;
    background-color: rgba(45, 45, 45, 0.1);
    padding: 10px;
    font-size: 27px;
  }
  p {
    text-align: justify;
  }
  .links {
    text-align: left;
    color: black;
  }
  #thumbnail {
    box-shadow: 0 10px 20px rgba(0,0,0,0.19), 0 6px 6px rgba(0,0,0,0.23);
  }
  .fullCard {
    width: 245px;
    border: 1px solid #ccc;
    border-radius: 5px;
    margin: 10px 5px;
    padding: 4px;
  }
  .cardContent {
    padding: 10px;
  }
  .cardText {
    margin-bottom: 30px;
  }
</style>
<div class="fullCard" id="thumbnail">
  <div class="cardContent">
    <div class="cardText">
      <h4>Alphabet</h4>
      <hr>
      <p><em>Google was founded by Larry Page and Sergey Brin while they were <u>Ph.D. students</u> at <strong>Stanford University</strong>.</em></p>
      </div>
      <div class="cardLinks">
        <a href="https://en.wikipedia.org/wiki/Larry_Page" target="_blank" class="links">Larry Page</a>
      <br><br>
        <a href="https://en.wikipedia.org/wiki/Sergey_Brin" target="_blank" class="links">Sergey Brin</a>
      </div>
    </div>
  </div>
</div>
```

Solution

```
// solution required
```

13. Use the text-transform Property to Make Text Uppercase

Description

The `text-transform` property in CSS is used to change the appearance of text. It's a convenient way to make sure text on a webpage appears consistently, without having to change the text content of the actual HTML elements. The following table shows how the different `text-transform` values change the example text "Transform me".

Value	Result
lowercase	"transform me"
uppercase	"TRANSFORM ME"
capitalize	"Transform Me"
initial	Use the default value
inherit	Use the <code>text-transform</code> value from the parent element
none	Default: Use the original text

Instructions

Transform the text of the `h4` to be uppercase using the `text-transform` property.

Challenge Seed

```
<style>
h4 {
  text-align: center;
  background-color: rgba(45, 45, 45, 0.1);
  padding: 10px;
  font-size: 27px;
}

p {
  text-align: justify;
}
.links {
  text-align: left;
  color: black;
  opacity: 0.7;
}
#thumbnail {
  box-shadow: 0 10px 20px rgba(0,0,0,0.19), 0 6px 6px rgba(0,0,0,0.23);
}
.fullCard {
  width: 245px;
  border: 1px solid #ccc;
  border-radius: 5px;
  margin: 10px 5px;
  padding: 4px;
}
.cardContent {
  padding: 10px;
}
.cardText {
  margin-bottom: 30px;
}
</style>
<div class="fullCard" id="thumbnail">
  <div class="cardContent">
    <div class="cardText">
      <h4>Alphabet</h4>
      <hr>
      <p><em>Google was founded by Larry Page and Sergey Brin while they were <u>Ph.D. students</u> at

```

```
<strong>Stanford University</strong>.</em></p>
</div>
<div class="cardLinks">
  <a href="https://en.wikipedia.org/wiki/Larry_Page" target="_blank" class="links">Larry Page</a>
<br><br>
  <a href="https://en.wikipedia.org/wiki/Sergey_Brin" target="_blank" class="links">Sergey Brin</a>
</div>
</div>
```

Solution

```
// solution required
```

14. Set the font-size for Multiple Heading Elements

Description

The `font-size` property is used to specify how large the text is in a given element. This rule can be used for multiple elements to create visual consistency of text on a page. In this challenge, you'll set the values for all `h1` through `h6` tags to balance the heading sizes.

Instructions

- Set the `font-size` of the `h1` tag to 68px.
- Set the `font-size` of the `h2` tag to 52px.
- Set the `font-size` of the `h3` tag to 40px.
- Set the `font-size` of the `h4` tag to 32px.
- Set the `font-size` of the `h5` tag to 21px.
- Set the `font-size` of the `h6` tag to 14px.

Challenge Seed

```
<style>

</style>
<h1>This is h1 text</h1>
<h2>This is h2 text</h2>
<h3>This is h3 text</h3>
<h4>This is h4 text</h4>
<h5>This is h5 text</h5>
<h6>This is h6 text</h6>
```

Solution

```
// solution required
```

15. Set the font-weight for Multiple Heading Elements

Description

You set the `font-size` of each heading tag in the last challenge, here you'll adjust the `font-weight`. The `font-weight` property sets how thick or thin characters are in a section of text.

Instructions

- Set the `font-weight` of the `h1` tag to 800.
- Set the `font-weight` of the `h2` tag to 600.
- Set the `font-weight` of the `h3` tag to 500.
- Set the `font-weight` of the `h4` tag to 400.
- Set the `font-weight` of the `h5` tag to 300.
- Set the `font-weight` of the `h6` tag to 200.

Challenge Seed

```
<style>
  h1 {
    font-size: 68px;
  }
  h2 {
    font-size: 52px;
  }
  h3 {
    font-size: 40px;
  }
  h4 {
    font-size: 32px;
  }
  h5 {
    font-size: 21px;
  }
  h6 {
    font-size: 14px;
  }
</style>
<h1>This is h1 text</h1>
<h2>This is h2 text</h2>
<h3>This is h3 text</h3>
<h4>This is h4 text</h4>
<h5>This is h5 text</h5>
<h6>This is h6 text</h6>
```

Solution

```
// solution required
```

16. Set the font-size of Paragraph Text

Description

The `font-size` property in CSS is not limited to headings, it can be applied to any element containing text.

Instructions

Change the value of the `font-size` property for the paragraph to `16px` to make it more visible.

Challenge Seed

```
<style>
  p {
    font-size: 10px;
  }
</style>
<p>
  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
</p>
```

Solution

```
// solution required
```

17. Set the line-height of Paragraphs

Description

CSS offers the `line-height` property to change the height of each line in a block of text. As the name suggests, it changes the amount of vertical space that each line of text gets.

Instructions

Add a `line-height` property to the `p` tag and set it to `25px`.

Challenge Seed

```
<style>
  p {
    font-size: 16px;
  }
</style>
<p>
  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
</p>
```

Solution

```
// solution required
```

18. Adjust the Hover State of an Anchor Tag

Description

This challenge will touch on the usage of pseudo-classes. A pseudo-class is a keyword that can be added to selectors, in order to select a specific state of the element. For example, the styling of an anchor tag can be changed for its

hover state using the `:hover` pseudo-class selector. Here's the CSS to change the `color` of the anchor tag to red during its hover state:

```
a:hover {  
  color: red;  
}
```

Instructions

The code editor has a CSS rule to style all `a` tags black. Add a rule so that when the user hovers over the `a` tag, the `color` is blue.

Challenge Seed

```
<style>  
a {  
  color: #000;  
}  
  
</style>  
<a href="http://freecatphotoapp.com/" target="_blank">CatPhotoApp</a>
```

Solution

```
<style>  
a {  
  color: #000;  
}  
a:hover {  
  color: rgba(0,0,255,1);  
}  
  
</style>  
<a href="http://freecatphotoapp.com/" target="_blank">CatPhotoApp</a>
```

19. Change an Element's Relative Position

Description

CSS treats each HTML element as its own box, which is usually referred to as the `CSS Box Model`. Block-level items automatically start on a new line (think headings, paragraphs, and divs) while inline items sit within surrounding content (like images or spans). The default layout of elements in this way is called the `normal flow` of a document, but CSS offers the `position` property to override it. When the position of an element is set to `relative`, it allows you to specify how CSS should move it *relative* to its current position in the normal flow of the page. It pairs with the CSS offset properties of `left` or `right`, and `top` or `bottom`. These say how many pixels, percentages, or ems to move the item *away* from where it is normally positioned. The following example moves the paragraph 10 pixels away from the bottom:

```
p {  
  position: relative;  
  bottom: 10px;  
}
```

Changing an element's position to `relative` does not remove it from the normal flow - other elements around it still behave as if that item were in its default position. **Note**

Positioning gives you a lot of flexibility and power over the visual layout of a page. It's good to remember that no matter the position of elements, the underlying HTML markup should be organized and make sense when read from

top to bottom. This is how users with visual impairments (who rely on assistive devices like screen readers) access your content.

Instructions

Change the position of the `h2` to `relative`, and use a CSS offset to move it 15 pixels away from the top of where it sits in the normal flow. Notice there is no impact on the positions of the surrounding `h1` and `p` elements.

Challenge Seed

```
<style>
  h2 {  

  }  

</style>
<body>
  <h1>On Being Well-Positioned</h1>
  <h2>Move me!</h2>
  <p>I still think the h2 is where it normally sits.</p>
</body>
```

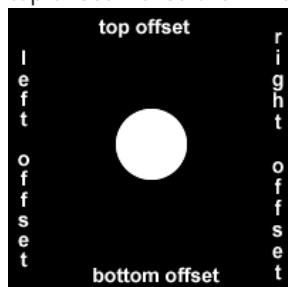
Solution

```
// solution required
```

20. Move a Relatively Positioned Element with CSS Offsets

Description

The CSS offsets of `top` or `bottom`, and `left` or `right` tell the browser how far to offset an item relative to where it would sit in the normal flow of the document. You're offsetting an element away from a given spot, which moves the element away from the referenced side (effectively, the opposite direction). As you saw in the last challenge, using the `top` offset moved the `h2` downwards. Likewise, using a `left` offset moves an item to the right.



Instructions

Use CSS offsets to move the `h2` 15 pixels to the right and 10 pixels up.

Challenge Seed

```
<head>
<style>
  h2 {  

    position: relative;  

  }  

</style>
```

```
</style>
</head>
<body>
  <h1>On Being Well-Positioned</h1>
  <h2>Move me!</h2>
  <p>I still think the h2 is where it normally sits.</p>
</body>
```

Solution

```
// solution required
```

21. Lock an Element to its Parent with Absolute Positioning

Description

The next option for the CSS `position` property is `absolute`, which locks the element in place relative to its parent container. Unlike the `relative` position, this removes the element from the normal flow of the document, so surrounding items ignore it. The CSS offset properties (`top` or `bottom` and `left` or `right`) are used to adjust the position. One nuance with absolute positioning is that it will be locked relative to its closest *positioned* ancestor. If you forget to add a position rule to the parent item, (this is typically done using `position: relative;`), the browser will keep looking up the chain and ultimately default to the body tag.

Instructions

Lock the `#searchbar` element to the top-right of its `section` parent by declaring its `position` as `absolute`. Give it `top` and `right` offsets of 50 pixels each.

Challenge Seed

```
<style>
  #searchbar {
    position: absolute;
    top: 50px;
    right: 50px;
  }
  section {
    position: relative;
  }
</style>
<body>
  <h1>Welcome!</h1>
  <section>
    <form id="searchbar">
      <label for="search">Search:</label>
      <input type="search" id="search" name="search">
      <input type="submit" name="submit" value="Go!">
    </form>
  </section>
</body>
```

Solution

```
// solution required
```

22. Lock an Element to the Browser Window with Fixed Positioning

Description

The next layout scheme that CSS offers is the `fixed` position, which is a type of absolute positioning that locks an element relative to the browser window. Similar to absolute positioning, it's used with the CSS offset properties and also removes the element from the normal flow of the document. Other items no longer "realize" where it is positioned, which may require some layout adjustments elsewhere. One key difference between the `fixed` and `absolute` positions is that an element with a `fixed` position won't move when the user scrolls.

Instructions

The navigation bar in the code is labeled with an id of `navbar`. Change its position to `fixed`, and offset it 0 pixels from the `top` and 0 pixels from the `left`. Notice the (lack of) impact to the `h1` position, it hasn't been pushed down to accommodate the navigation bar and would need to be adjusted separately.

Challenge Seed

```
<style>
  #navbar {
    width: 100%;
    background-color: #767676;
  }
  nav ul {
    margin: 0px;
    padding: 5px 0px 5px 30px;
  }
  nav li {
    display: inline;
    margin-right: 20px;
  }
  a {
    text-decoration: none;
  }
</style>
<body>
  <header>
    <h1>Welcome!</h1>
    <nav id="navbar">
      <ul>
        <li><a href="">Home</a></li>
        <li><a href="">Contact</a></li>
      </ul>
    </nav>
  </header>
  <p>I shift up when the #navbar is fixed to the browser window.</p>
</body>
```

Solution

```
// solution required
```

23. Push Elements Left or Right with the float Property

Description

The next positioning tool does not actually use `position`, but sets the `float` property of an element. Floating elements are removed from the normal flow of a document and pushed to either the `left` or `right` of their containing parent element. It's commonly used with the `width` property to specify how much horizontal space the floated element requires.

Instructions

The given markup would work well as a two-column layout, with the `section` and `aside` elements next to each other. Give the `#left` item a `float` of `left` and the `#right` item a `float` of `right`.

Challenge Seed

```
<head>
  <style>
    #left {
      width: 50%;
    }
    #right {
      width: 40%;
    }
    aside, section {
      padding: 2px;
      background-color: #ccc;
    }
  </style>
</head>
<body>
  <header>
    <h1>Welcome!</h1>
  </header>
  <section id="left">
    <h2>Content</h2>
    <p>Good stuff</p>
  </section>
  <aside id="right">
    <h2>Sidebar</h2>
    <p>Links</p>
  </aside>
</body>
```

Solution

```
// solution required
```

24. Change the Position of Overlapping Elements with the `z-index` Property

Description

When elements are positioned to overlap (i.e. using `position: absolute` | `relative` | `fixed` | `sticky`), the element coming later in the HTML markup will, by default, appear on the top of the other elements. However, the `z-index` property can specify the order of how elements are stacked on top of one another. It must be an integer (i.e. a whole number and not a decimal), and higher values for the `z-index` property of an element move it higher in the stack than those with lower values.

Instructions

Add a `z-index` property to the element with the class name of `first` (the red rectangle) and set it to a value of 2 so it covers the other element (blue rectangle).

Challenge Seed

```
<style>
  div {
    width: 60%;
    height: 200px;
    margin-top: 20px;
  }

  .first {
    background-color: red;
    position: absolute;
  }

  .second {
    background-color: blue;
    position: absolute;
    left: 40px;
    top: 50px;
    z-index: 1;
  }
</style>

<div class="first"></div>
<div class="second"></div>
```

Solution

```
// solution required
```

25. Center an Element Horizontally Using the margin Property

Description

Another positioning technique is to center a block element horizontally. One way to do this is to set its `margin` to a value of `auto`. This method works for images, too. Images are inline elements by default, but can be changed to block elements when you set the `display` property to `block`.

Instructions

Center the `div` on the page by adding a `margin` property with a value of `auto`.

Challenge Seed

```
<style>
  div {
    background-color: blue;
    height: 100px;
    width: 100px;
  }
</style>
<div></div>
```

Solution

```
var code = "div {background-color: blue; height: 100px; width: 100px; margin: auto;}"
```

26. Learn about Complementary Colors

Description

Color theory and its impact on design is a deep topic and only the basics are covered in the following challenges. On a website, color can draw attention to content, evoke emotions, or create visual harmony. Using different combinations of colors can really change the look of a website, and a lot of thought can go into picking a color palette that works with your content. The color wheel is a useful tool to visualize how colors relate to each other - it's a circle where similar hues are neighbors and different hues are farther apart. When two colors are opposite each other on the wheel, they are called complementary colors. They have the characteristic that if they are combined, they "cancel" each other out and create a gray color. However, when placed side-by-side, these colors appear more vibrant and produce a strong visual contrast. Some examples of complementary colors with their hex codes are:

```
red (#FF0000) and cyan (#00FFFF)  
green (#00FF00) and magenta (#FF00FF)  
blue (#0000FF) and yellow (#FFFF00)
```

This is different than the outdated RYB color model that many of us were taught in school, which has different primary and complementary colors. Modern color theory uses the additive RGB model (like on a computer screen) and the subtractive CMY(K) model (like in printing). Read [here](#) for more information on this complex subject. There are many color picking tools available online that have an option to find the complement of a color. **Note**

For all color challenges: Using color can be a powerful way to add visual interest to a page. However, color alone should not be used as the only way to convey important information because users with visual impairments may not understand that content. This issue will be covered in more detail in the Applied Accessibility challenges.

Instructions

Change the `background-color` property of the `blue` and `yellow` classes to their respective colors. Notice how the colors look different next to each other than they do compared against the white background.

Challenge Seed

```
<style>  
body {  
    background-color: #FFFFFF;  
}  
.blue {  
    background-color: #000000;  
}  
.yellow {  
    background-color: #000000;  
}  
div {  
    display: inline-block;  
    height: 100px;  
    width: 100px;  
}  
</style>  
<div class="blue"></div>  
<div class="yellow"></div>
```

Solution

```
// solution required
```

27. Learn about Tertiary Colors

Description

Computer monitors and device screens create different colors by combining amounts of red, green, and blue light. This is known as the RGB additive color model in modern color theory. Red (R), green (G), and blue (B) are called primary colors. Mixing two primary colors creates the secondary colors cyan ($G + B$), magenta ($R + B$) and yellow ($R + G$). You saw these colors in the Complementary Colors challenge. These secondary colors happen to be the complement to the primary color not used in their creation, and are opposite to that primary color on the color wheel. For example, magenta is made with red and blue, and is the complement to green. Tertiary colors are the result of combining a primary color with one of its secondary color neighbors. For example, within the RGB color model, red (primary) and yellow (secondary) make orange (tertiary). This adds six more colors to a simple color wheel for a total of twelve. There are various methods of selecting different colors that result in a harmonious combination in design. One example that can use tertiary colors is called the split-complementary color scheme. This scheme starts with a base color, then pairs it with the two colors that are adjacent to its complement. The three colors provide strong visual contrast in a design, but are more subtle than using two complementary colors. Here are three colors created using the split-complement scheme:

Color	Hex Code
orange	#FF7F00
cyan	#00FFFF
raspberry	#FF007F

Instructions

Change the `background-color` property of the `orange`, `cyan`, and `raspberry` classes to their respective colors. Make sure to use the hex codes and not the color names.

Challenge Seed

```
<style>
  body {
    background-color: #FFFFFF;
  }

  .orange {
    background-color: #000000;
  }

  .cyan {
    background-color: #000000;
  }

  .raspberry {
    background-color: #000000;
  }

  div {
    height: 100px;
    width: 100px;
    margin-bottom: 5px;
  }
</style>

<div class="orange"></div>
<div class="cyan"></div>
<div class="raspberry"></div>
```

Solution

```
// solution required
```

28. Adjust the Color of Various Elements to Complementary Colors

Description

The Complementary Colors challenge showed that opposite colors on the color wheel can make each other appear more vibrant when placed side-by-side. However, the strong visual contrast can be jarring if it's overused on a website, and can sometimes make text harder to read if it's placed on a complementary-colored background. In practice, one of the colors is usually dominant and the complement is used to bring visual attention to certain content on the page.

Instructions

This page will use a shade of teal (#09A7A1) as the dominant color, and its orange (#FF790E) complement to visually highlight the sign-up buttons. Change the `background-color` of both the `header` and `footer` from black to the teal color. Then change the `h2` `text color` to teal as well. Finally, change the `background-color` of the button to the orange color.

Challenge Seed

```
<style>
  body {
    background-color: white;
  }
  header {
    background-color: black;
    color: white;
    padding: 0.25em;
  }
  h2 {
    color: black;
  }
  button {
    background-color: white;
  }
  footer {
    background-color: black;
    color: white;
    padding: 0.5em;
  }
</style>
<header>
  <h1>Cooking with FCC!</h1>
</header>
<main>
  <article>
    <h2>Machine Learning in the Kitchen</h2>
    <p>Join this two day workshop that walks through how to implement cutting-edge snack-getting algorithms with a command line interface. Coding usually involves writing exact instructions, but sometimes you need your computer to execute flexible commands, like <code>fetch Pringles</code>.</p>
    <button>Sign Up</button>
  </article>
  <article>
    <h2>Bisection Vegetable Chopping</h2>
    <p>This week-long retreat will level-up your coding ninja skills to actual ninja skills. No longer is the humble bisection search limited to sorted arrays or coding interview questions, applying its concepts in the kitchen will have you chopping carrots in  $O(\log n)$  time before you know it.</p>
    <button>Sign Up</button>
  </article>
</main>
<br>
<footer>&copy; 2018 FCC Kitchen</footer>
```

Solution

```
// solution required
```

29. Adjust the Hue of a Color

Description

Colors have several characteristics including hue, saturation, and lightness. CSS3 introduced the `hsl()` property as an alternative way to pick a color by directly stating these characteristics. **Hue** is what people generally think of as 'color'. If you picture a spectrum of colors starting with red on the left, moving through green in the middle, and blue on right, the hue is where a color fits along this line. In `hsl()`, hue uses a color wheel concept instead of the spectrum, where the angle of the color on the circle is given as a value between 0 and 360. **Saturation** is the amount of gray in a color. A fully saturated color has no gray in it, and a minimally saturated color is almost completely gray. This is given as a percentage with 100% being fully saturated. **Lightness** is the amount of white or black in a color. A percentage is given ranging from 0% (black) to 100% (white), where 50% is the normal color. Here are a few examples of using `hsl()` with fully-saturated, normal lightness colors:

Color	HSL
red	<code>hsl(0, 100%, 50%)</code>
yellow	<code>hsl(60, 100%, 50%)</code>
green	<code>hsl(120, 100%, 50%)</code>
cyan	<code>hsl(180, 100%, 50%)</code>
blue	<code>hsl(240, 100%, 50%)</code>
magenta	<code>hsl(300, 100%, 50%)</code>

Instructions

Change the `background-color` of each `div` element based on the class names (`green`, `cyan`, or `blue`) using `hsl()`. All three should have full saturation and normal lightness.

Challenge Seed

```
<style>
  body {
    background-color: #FFFFFF;
  }

  .green {
    background-color: #000000;
  }

  .cyan {
    background-color: #000000;
  }

  .blue {
    background-color: #000000;
  }

  div {
    display: inline-block;
    height: 100px;
    width: 100px;
  }
</style>

<div class="green"></div>
<div class="cyan"></div>
<div class="blue"></div>
```

Solution

```
// solution required
```

30. Adjust the Tone of a Color

Description

The `hsl()` option in CSS also makes it easy to adjust the tone of a color. Mixing white with a pure hue creates a tint of that color, and adding black will make a shade. Alternatively, a tone is produced by adding gray or by both tinting and shading. Recall that the 's' and 'l' of `hsl()` stand for saturation and lightness, respectively. The saturation percent changes the amount of gray and the lightness percent determines how much white or black is in the color. This is useful when you have a base hue you like, but need different variations of it.

Instructions

All elements have a default `background-color` of `transparent`. Our `nav` element currently appears have a `cyan` background, because the element behind it has a `background-color` set to `cyan`. Add a `background-color` to the `nav` element so it uses the same `cyan` hue, but has `80%` saturation and `25%` lightness values to change its tone and shade.

Challenge Seed

```
<style>
  header {
    background-color: hsl(180, 90%, 35%);
    color: #FFFFFF;
  }

  nav {
  }

  h1 {
    text-indent: 10px;
    padding-top: 10px;
  }

  nav ul {
    margin: 0px;
    padding: 5px 0px 5px 30px;
  }

  nav li {
    display: inline;
    margin-right: 20px;
  }

  a {
    text-decoration: none;
    color: inherit;
  }
</style>

<header>
  <h1>Cooking with FCC!</h1>
  <nav>
    <ul>
      <li><a href="#">Home</a></li>
      <li><a href="#">Classes</a></li>
      <li><a href="#">Contact</a></li>
    </ul>
  </nav>
</header>
```

Solution

```
var code = "nav {background-color: hsl(180, 80%, 25%);}"
```

31. Create a Gradual CSS Linear Gradient

Description

Applying a color on HTML elements is not limited to one flat hue. CSS provides the ability to use color transitions, otherwise known as gradients, on elements. This is accessed through the `background` property's `linear-gradient()` function. Here is the general syntax: `background: linear-gradient(gradient_direction, color 1, color 2, color 3, ...);` The first argument specifies the direction from which color transition starts - it can be stated as a degree, where `90deg` makes a vertical gradient and `45deg` is angled like a backslash. The following arguments specify the order of colors used in the gradient. Example: `background: linear-gradient(90deg, red, yellow, rgb(204, 204, 255));`

Instructions

Use a `linear-gradient()` for the `div` element's `background`, and set it from a direction of 35 degrees to change the color from `#CCFFFF` to `#FFCCCC`. **Note**

While there are other ways to specify a color value, like `rgb()` or `hsl()`, use hex values for this challenge.

Challenge Seed

```
<style>
div{
  border-radius: 20px;
  width: 70%;
  height: 400px;
  margin: 50px auto;
}

</style>

<div></div>
```

Solution

```
var code = "<style> div{border-radius: 20px; width: 70%; height: 400px; margin: 50px auto; background: linear-gradient(35deg, #cff, #fcc);}</style><div></div>"
```

32. Use a CSS Linear Gradient to Create a Striped Element

Description

The `repeating-linear-gradient()` function is very similar to `linear-gradient()` with the major difference that it repeats the specified gradient pattern. `repeating-linear-gradient()` accepts a variety of values, but for simplicity, you'll work with an angle value and color stop values in this challenge. The angle value is the direction of the gradient. Color stops are like width values that mark where a transition takes place, and are given with a percentage or a number of pixels. In the example demonstrated in the code editor, the gradient starts with the color `yellow` at 0 pixels which blends into the second color `blue` at 40 pixels away from the start. Since the next color stop is also at 40 pixels, the gradient immediately changes to the third color `green`, which itself blends into the fourth color value `red`.

as that is 80 pixels away from the beginning of the gradient. For this example, it helps to think about the color stops as pairs where every two colors blend together. 0px [yellow -- blend -- blue] 40px [green -- blend -- red] 80px If every two color stop values are the same color, the blending isn't noticeable because it's between the same color, followed by a hard transition to the next color, so you end up with stripes.

Instructions

Make stripes by changing the `repeating-linear-gradient()` to use a gradient angle of `45deg`, then set the first two color stops to `yellow`, and finally the second two color stops to `black`.

Challenge Seed

```
<style>

div{
    border-radius: 20px;
    width: 70%;
    height: 400px;
    margin: 50 auto;
    background: repeating-linear-gradient(
        90deg,
        yellow 0px,
        blue 40px,
        green 40px,
        red 80px
    );
}

</style>

<div></div>
```

Solution

```
var code = "background: repeating-linear-gradient(45deg, yellow 0px, yellow 40px, black 40px, black 80px);"
```

33. Create Texture by Adding a Subtle Pattern as a Background Image

Description

One way to add texture and interest to a background and have it stand out more is to add a subtle pattern. The key is balance, as you don't want the background to stand out too much, and take away from the foreground. The `background` property supports the `url()` function in order to link to an image of the chosen texture or pattern. The link address is wrapped in quotes inside the parentheses.

Instructions

Using the url of <https://i.imgur.com/MJAkxbh.png>, set the `background` of the whole page with the `body` selector.

Challenge Seed

```
<style>
body {

}

</style>
```

Solution

```
var code = "body {background: url('https://i.imgur.com/MJAkxbh.png')}"
```

34. Use the CSS Transform scale Property to Change the Size of an Element

Description

To change the scale of an element, CSS has the `transform` property, along with its `scale()` function. The following code example doubles the size of all the paragraph elements on the page:

```
p {  
    transform: scale(2);  
}
```

Instructions

Increase the size of the element with the id of `ball2` to 1.5 times its original size.

Challenge Seed

```
<style>  
.ball {  
    width: 40px;  
    height: 40px;  
    margin: 50 auto;  
    position: fixed;  
    background: linear-gradient(  
        35deg,  
        #ccffff,  
        #ffcccc  
    );  
    border-radius: 50%;  
}  
#ball1 {  
    left: 20%;  
}  
#ball2 {  
    left: 65%;  
}  
  
</style>  
<div class="ball" id= "ball1"></div>  
<div class="ball" id= "ball2"></div>
```

Solution

```
var code = "#ball2 {left: 65%; transform: scale(1.5);}"
```

35. Use the CSS Transform scale Property to Scale an Element on Hover

Description

The `transform` property has a variety of functions that lets you scale, move, rotate, skew, etc., your elements. When used with pseudo-classes such as `:hover` that specify a certain state of an element, the `transform` property can easily add interactivity to your elements. Here's an example to scale the paragraph elements to 2.1 times their original size when a user hovers over them:

```
p:hover {  
  transform: scale(2.1);  
}
```

NOTE: Applying a transform to a `div` element will also affect any child elements contained in the `div`.

Instructions

Add a CSS rule for the `hover` state of the `div` and use the `transform` property to scale the `div` element to 1.1 times its original size when a user hovers over it.

Challenge Seed

```
<style>  
div {  
  width: 70%;  
  height: 100px;  
  margin: 50px auto;  
  background: linear-gradient(  
    53deg,  
    #ccfffc,  
    #ffcccc  
  );  
}  
  
</style>  
<div></div>
```

Solution

```
var code = "div:hover {transform: scale(1.1);}"
```

36. Use the CSS Transform Property `skewX` to Skew an Element Along the X-Axis

Description

The next function of the `transform` property is `skewX()`, which skews the selected element along its X (horizontal) axis by a given degree. The following code skews the paragraph element by -32 degrees along the X-axis.

```
p {  
  transform: skewX(-32deg);  
}
```

Instructions

Skew the element with the id of `bottom` by 24 degrees along the X-axis by using the `transform` property.

Challenge Seed

```

<style>
  div {
    width: 70%;
    height: 100px;
    margin: 50px auto;
  }
  #top {
    background-color: red;
  }
  #bottom {
    background-color: blue;
  }
</style>

<div id="top"></div>
<div id="bottom"></div>

```

Solution

```
var code = "#bottom {background-color: blue; transform: skewX(24deg);}"
```

37. Use the CSS Transform Property skewY to Skew an Element Along the Y-Axis

Description

Given that the `skewX()` function skews the selected element along the X-axis by a given degree, it is no surprise that the `skewY()` property skews an element along the Y (vertical) axis.

Instructions

Skew the element with the id of `top` -10 degrees along the Y-axis by using the `transform` property.

Challenge Seed

```

<style>
  div {
    width: 70%;
    height: 100px;
    margin: 50px auto;
  }
  #top {
    background-color: red;
  }
  #bottom {
    background-color: blue;
    transform: skewX(24deg);
  }
</style>

<div id="top"></div>
<div id="bottom"></div>

```

Solution

```
var code = "#top {background-color: red; transform: skewY(-10deg);}"
```

38. Create a Graphic Using CSS

Description

By manipulating different selectors and properties, you can make interesting shapes. One of the easier ones to try is a crescent moon shape. For this challenge you need to work with the `box-shadow` property that sets the shadow of an element, along with the `border-radius` property that controls the roundness of the element's corners. You will create a round, transparent object with a crisp shadow that is slightly offset to the side - the shadow is actually going to be the moon shape you see. In order to create a round object, the `border-radius` property should be set to a value of 50%. You may recall from an earlier challenge that the `box-shadow` property takes values for `offset-x`, `offset-y`, `blur-radius`, `spread-radius` and a color value in that order. The `blur-radius` and `spread-radius` values are optional.

Instructions

Manipulate the square element in the editor to create the moon shape. First, change the `background-color` to transparent, then set the `border-radius` property to 50% to make the circular shape. Finally, change the `box-shadow` property to set the `offset-x` to 25px, the `offset-y` to 10px, `blur-radius` to 0, `spread-radius` to 0, and color to blue.

Challenge Seed

```
<style>
.center
{
  position: absolute;
  margin: auto;
  top: 0;
  right: 0;
  bottom: 0;
  left: 0;
  width: 100px;
  height: 100px;

  background-color: blue;
  border-radius: 0px;
  box-shadow: 25px 10px 10px 10px green;
}

</style>
<div class="center"></div>
```

Solution

```
var code = ".center {background-color: transparent; border-radius: 50%; box-shadow: 25px 10px 0px 0px blue;}"
```

39. Create a More Complex Shape Using CSS and HTML

Description

One of the most popular shapes in the world is the heart shape, and in this challenge you'll create one using pure CSS. But first, you need to understand the `::before` and `::after` pseudo-elements. These pseudo-elements are used to add something before or after a selected element. In the following example, a `::before` pseudo-element is used to add a rectangle to an element with the class `heart`:

```
.heart::before {
  content: "";
  background-color: yellow;
```

```
border-radius: 25%;  
position: absolute;  
height: 50px;  
width: 70px;  
top: -50px;  
left: 5px;  
}
```

For the `::before` and `::after` pseudo-elements to function properly, they must have a defined `content` property. This property is usually used to add things like a photo or text to the selected element. When the `::before` and `::after` pseudo-elements are used to make shapes, the `content` property is still required, but it's set to an empty string. In the above example, the element with the class of `heart` has a `::before` pseudo-element that produces a yellow rectangle with `height` and `width` of `50px` and `70px`, respectively. This rectangle has round corners due to its `25%` border radius and is positioned absolutely at `5px` from the `left` and `50px` above the `top` of the element.

Instructions

Transform the element on the screen to a heart. In the `heart::after` selector, change the `background-color` to pink and the `border-radius` to `50%`. Next, target the element with the class `heart` (just `heart`) and fill in the `transform` property. Use the `rotate()` function with `-45` degrees. Finally, in the `heart::before` selector, set its `content` property to an empty string.

Challenge Seed

```
<style>  
.heart {  
  position: absolute;  
  margin: auto;  
  top: 0;  
  right: 0;  
  bottom: 0;  
  left: 0;  
  background-color: pink;  
  height: 50px;  
  width: 50px;  
  transform: ;  
}  
.heart::after {  
  background-color: blue;  
  content: "";  
  border-radius: 25%;  
  position: absolute;  
  width: 50px;  
  height: 50px;  
  top: 0px;  
  left: 25px;  
}  
.heart::before {  
  content: ;  
  background-color: pink;  
  border-radius: 50%;  
  position: absolute;  
  width: 50px;  
  height: 50px;  
  top: -25px;  
  left: 0px;  
}  
</style>  
<div class = "heart"></div>
```

Solution

```
<style>  
.heart {  
  position: absolute;  
  margin: auto;  
  top: 0;
```

```

    right: 0;
    bottom: 0;
    left: 0;
    background-color: pink;
    height: 50px;
    width: 50px;
    transform: rotate(-45deg);
}
.heart::after {
    background-color: pink;
    content: "";
    border-radius: 50%;
    position: absolute;
    width: 50px;
    height: 50px;
    top: 0px;
    left: 25px;
}
.heart::before {
    content: "";
    background-color: pink;
    border-radius: 50%;
    position: absolute;
    width: 50px;
    height: 50px;
    top: -25px;
    left: 0px;
}

```

</style>

<div class = "heart"></div>

40. Learn How the CSS @keyframes and animation Properties Work

Description

To animate an element, you need to know about the animation properties and the `@keyframes` rule. The animation properties control how the animation should behave and the `@keyframes` rule controls what happens during that animation. There are eight animation properties in total. This challenge will keep it simple and cover the two most important ones first: `animation-name` sets the name of the animation, which is later used by `@keyframes` to tell CSS which rules go with which animations. `animation-duration` sets the length of time for the animation. `@keyframes` is how to specify exactly what happens within the animation over the duration. This is done by giving CSS properties for specific "frames" during the animation, with percentages ranging from 0% to 100%. If you compare this to a movie, the CSS properties for 0% is how the element displays in the opening scene. The CSS properties for 100% is how the element appears at the end, right before the credits roll. Then CSS applies the magic to transition the element over the given duration to act out the scene. Here's an example to illustrate the usage of `@keyframes` and the animation properties:

```

#anim {
    animation-name: colorful;
    animation-duration: 3s;
}
@keyframes colorful {
    0% {
        background-color: blue;
    }
    100% {
        background-color: yellow;
    }
}

```

For the element with the `anim` id, the code snippet above sets the `animation-name` to `colorful` and sets the `animation-duration` to 3 seconds. Then the `@keyframes` rule links to the animation properties with the name `colorful`. It sets the color to blue at the beginning of the animation (0%) which will transition to yellow by the end of

the animation (100%). You aren't limited to only beginning-end transitions, you can set properties for the element for any percentage between 0% and 100%.

Instructions

Create an animation for the element with the id `rect`, by setting the `animation-name` to `rainbow` and the `animation-duration` to 4 seconds. Next, declare a `@keyframes` rule, and set the `background-color` at the beginning of the animation (0%) to blue, the middle of the animation (50%) to green, and the end of the animation (100%) to yellow.

Challenge Seed

```
<style>
div {
  height: 40px;
  width: 70%;
  background: black;
  margin: 50px auto;
  border-radius: 5px;
}

#rect {
```



```
</style>
<div id="rect"></div>
```

Solution

```
// solution required
```

41. Use CSS Animation to Change the Hover State of a Button

Description

You can use CSS `@keyframes` to change the color of a button in its hover state. Here's an example of changing the width of an image on hover:

```
<style>
img:hover {
  animation-name: width;
  animation-duration: 500ms;
}

@keyframes width {
  100% {
    width: 40px;
  }
}
</style>


```

Instructions

Note that `ms` stands for milliseconds, where 1000ms is equal to 1s. Use CSS `@keyframes` to change the background-color of the button element so it becomes `#4791d0` when a user hovers over it. The `@keyframes` rule should only have an entry for `100%`.

Challenge Seed

```
<style>
button {
  border-radius: 5px;
  color: white;
  background-color: #0F5897;
  padding: 5px 10px 8px 10px;
}

button:hover {
  animation-name: background-color;
  animation-duration: 500ms;
}

</style>

<button>Register</button>
```

Solution

```
// solution required
```

42. Modify Fill Mode of an Animation

Description

That's great, but it doesn't work right yet. Notice how the animation resets after `500ms` has passed, causing the button to revert back to the original color. You want the button to stay highlighted. This can be done by setting the `animation-fill-mode` property to `forwards`. The `animation-fill-mode` specifies the style applied to an element when the animation has finished. You can set it like so: `animation-fill-mode: forwards;`

Instructions

Set the `animation-fill-mode` property of `button:hover` to `forwards` so the button stays highlighted when a user hovers over it.

Challenge Seed

```
<style>
button {
  border-radius: 5px;
  color: white;
  background-color: #0F5897;
  padding: 5px 10px 8px 10px;
}

button:hover {
  animation-name: background-color;
  animation-duration: 500ms;
  /* add your code below this line */

  /* add your code above this line */
}

@keyframes background-color {
  100% {
    background-color: #4791d0;
  }
}
```

```

    }
</style>
<button>Register</button>
```

Solution

```
var code = "button:hover {animation-name: background-color; animation-duration: 500ms; animation-fill-mode: forwards;}"
```

43. Create Movement Using CSS Animation

Description

When elements have a specified position , such as fixed or relative , the CSS offset properties right , left , top , and bottom can be used in animation rules to create movement. As shown in the example below, you can push the item downwards then upwards by setting the top property of the 50% keyframe to 50px, but having it set to 0px for the first (0%) and the last (100%) keyframe.

```
@keyframes rainbow {
  0% {
    background-color: blue;
    top: 0px;
  }
  50% {
    background-color: green;
    top: 50px;
  }
  100% {
    background-color: yellow;
    top: 0px;
  }
}
```

Instructions

Add a horizontal motion to the div animation. Using the left offset property, add to the @keyframes rule so rainbow starts at 0 pixels at 0% , moves to 25 pixels at 50% , and ends at -25 pixels at 100% . Don't replace the top property in the editor - the animation should have both vertical and horizontal motion.

Challenge Seed

```
<style>
  div {
    height: 40px;
    width: 70%;
    background: black;
    margin: 50px auto;
    border-radius: 5px;
    position: relative;
  }

  #rect {
    animation-name: rainbow;
    animation-duration: 4s;
  }

  @keyframes rainbow {
    0% {
      background-color: blue;
      top: 0px;
      left: 0px;
    }
    50% {
      background-color: green;
      top: 50px;
      left: 25px;
    }
    100% {
      background-color: yellow;
      top: 0px;
      left: -25px;
    }
  }
</style>
```

```

}
50% {
  background-color: green;
  top: 50px;

}
100% {
  background-color: yellow;
  top: 0px;

}
}

</style>

<div id="rect"></div>

```

Solution

```
var code = "@keyframes rainbow {0% {background-color: blue; top: 0px; left: 0px;} 50% {background-color: green; top: 50px; left: 25px;} 100% {background-color: yellow; top: 0px; left:-25px;}}"
```

44. Create Visual Direction by Fading an Element from Left to Right

Description

For this challenge, you'll change the `opacity` of an animated element so it gradually fades as it reaches the right side of the screen. In the displayed animation, the round element with the gradient background moves to the right by the 50% mark of the animation per the `@keyframes` rule.

Instructions

Target the element with the id of `ball` and add the `opacity` property set to `0.1` at `50%`, so the element fades as it moves to the right.

Challenge Seed

```

<style>

#ball {
  width: 70px;
  height: 70px;
  margin: 50px auto;
  position: fixed;
  left: 20%;
  border-radius: 50%;
  background: linear-gradient(
    35deg,
    #ccffff,
    #ffcccc
  );
  animation-name: fade;
  animation-duration: 3s;
}

@keyframes fade {
  50% {
    left: 60%;

  }
}

</style>

```

```
<div id="ball"></div>
```

Solution

```
var code = "@keyframes fade {50% { left: 60%; opacity: 0.1;}}"
```

45. Animate Elements Continually Using an Infinite Animation Count

Description

The previous challenges covered how to use some of the animation properties and the `@keyframes` rule. Another animation property is the `animation-iteration-count`, which allows you to control how many times you would like to loop through the animation. Here's an example: `animation-iteration-count: 3;` In this case the animation will stop after running 3 times, but it's possible to make the animation run continuously by setting that value to infinite.

Instructions

To keep the ball bouncing on the right on a continuous loop, change the `animation-iteration-count` property to `infinite`.

Challenge Seed

```
<style>
#ball {
  width: 100px;
  height: 100px;
  margin: 50px auto;
  position: relative;
  border-radius: 50%;
  background: linear-gradient(
    35deg,
    #ccffff,
    #ffcccc
  );
  animation-name: bounce;
  animation-duration: 1s;
  animation-iteration-count: 3;
}

@keyframes bounce{
  0% {
    top: 0px;
  }
  50% {
    top: 249px;
    width: 130px;
    height: 70px;
  }
  100% {
    top: 0px;
  }
}
</style>
<div id="ball"></div>
```

Solution

```
// solution required
```

46. Make a CSS Heartbeat using an Infinite Animation Count

Description

Here's one more continuous animation example with the `animation-iteration-count` property that uses the heart you designed in a previous challenge. The one-second long heartbeat animation consists of two animated pieces. The `heart` elements (including the `:before` and `:after` pieces) are animated to change size using the `transform` property, and the background `div` is animated to change its color using the `background` property.

Instructions

Keep the heart beating by adding the `animation-iteration-count` property for both the `back` class and the `heart` class and setting the value to infinite. The `heart:before` and `heart:after` selectors do not need any animation properties.

Challenge Seed

```
<style>
.back {
  position: fixed;
  padding: 0;
  margin: 0;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: white;
  animation-name: backdiv;
  animation-duration: 1s;
}

.heart {
  position: absolute;
  margin: auto;
  top: 0;
  right: 0;
  bottom: 0;
  left: 0;
  background-color: pink;
  height: 50px;
  width: 50px;
  transform: rotate(-45deg);
  animation-name: beat;
  animation-duration: 1s;
}

.heart:after {
  background-color: pink;
  content: "";
  border-radius: 50%;
  position: absolute;
  width: 50px;
  height: 50px;
  top: 0px;
  left: 25px;
}
.heart:before {
  background-color: pink;
  content: "";
  border-radius: 50%;
  position: absolute;
```

```

width: 50px;
height: 50px;
top: -25px;
left: 0px;
}

@keyframes backdiv {
  50% {
    background: #ffe6f2;
  }
}

@keyframes beat {
  0% {
    transform: scale(1) rotate(-45deg);
  }
  50% {
    transform: scale(0.6) rotate(-45deg);
  }
}

</style>
<div class="back"></div>
<div class="heart"></div>

```

Solution

```
// solution required
```

47. Animate Elements at Variable Rates

Description

There are a variety of ways to alter the animation rates of similarly animated elements. So far, this has been achieved by applying an `animation-iteration-count` property and setting `@keyframes` rules. To illustrate, the animation on the right consists of two "stars" that each decrease in size and opacity at the 20% mark in the `@keyframes` rule, which creates the twinkle animation. You can change the `@keyframes` rule for one of the elements so the stars twinkle at different rates.

Instructions

Alter the animation rate for the element with the class name of `star-1` by changing its `@keyframes` rule to 50%.

Challenge Seed

```

<style>
  .stars {
    background-color: white;
    height: 30px;
    width: 30px;
    border-radius: 50%;
    animation-iteration-count: infinite;
  }

  .star-1 {
    margin-top: 15%;
    margin-left: 60%;
    animation-name: twinkle-1;
    animation-duration: 1s;
  }

  .star-2 {
    margin-top: 25%;
    margin-left: 25%;
  }

```

```

        animation-name: twinkle-2;
        animation-duration: 1s;
    }

    @keyframes twinkle-1 {
        20% {
            transform: scale(0.5);
            opacity: 0.5;
        }
    }

    @keyframes twinkle-2 {
        20% {
            transform: scale(0.5);
            opacity: 0.5;
        }
    }

    #back {
        position: fixed;
        padding: 0;
        margin: 0;
        top: 0;
        left: 0;
        width: 100%;
        height: 100%;
        background: linear-gradient(black, #000099, #66c2ff, #ffcccc, #ffeee6);
    }

```

</style>

```

<div id="back"></div>
<div class="star-1 stars"></div>
<div class="star-2 stars"></div>

```

Solution

```
var code = "@keyframes twinkle-1 {50% {transform: scale(0.5); opacity: 0.5;}}"
```

48. Animate Multiple Elements at Variable Rates

Description

In the previous challenge, you changed the animation rates for two similarly animated elements by altering their `@keyframes` rules. You can achieve the same goal by manipulating the `animation-duration` of multiple elements. In the animation running in the code editor, there are three "stars" in the sky that twinkle at the same rate on a continuous loop. To make them twinkle at different rates, you can set the `animation-duration` property to different values for each element.

Instructions

Set the `animation-duration` of the elements with the classes `star-1`, `star-2`, and `star-3` to `1s`, `0.9s`, and `1.1s`, respectively.

Challenge Seed

```

<style>
    .stars {
        background-color: white;
        height: 30px;
        width: 30px;
        border-radius: 50%;
        animation-iteration-count: infinite;
    }

```

```

.star-1 {
  margin-top: 15%;
  margin-left: 60%;
  animation-duration: 1s;
  animation-name: twinkle;
}

.star-2 {
  margin-top: 25%;
  margin-left: 25%;
  animation-duration: 1s;
  animation-name: twinkle;
}

.star-3 {
  margin-top: 10%;
  margin-left: 50%;
  animation-duration: 1s;
  animation-name: twinkle;
}

@keyframes twinkle {
  20% {
    transform: scale(0.5);
    opacity: 0.5;
  }
}

#back {
  position: fixed;
  padding: 0;
  margin: 0;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: linear-gradient(black, #000099, #66c2ff, #ffcccc, #ffeee6);
}

```

</style>

```

<div id="back"></div>
<div class="star-1 stars"></div>
<div class="star-2 stars"></div>
<div class="star-3 stars"></div>

```

Solution

```
// solution required
```

49. Change Animation Timing with Keywords

Description

In CSS animations, the `animation-timing-function` property controls how quickly an animated element changes over the duration of the animation. If the animation is a car moving from point A to point B in a given time (your `animation-duration`), the `animation-timing-function` says how the car accelerates and decelerates over the course of the drive. There are a number of predefined keywords available for popular options. For example, the default value is `ease`, which starts slow, speeds up in the middle, and then slows down again in the end. Other options include `ease-out`, which is quick in the beginning then slows down, `ease-in`, which is slow in the beginning, then speeds up at the end, or `linear`, which applies a constant animation speed throughout.

Instructions

For the elements with id of `ball1` and `ball2`, add an `animation-timing-function` property to each, and set `#ball1` to `linear`, and `#ball2` to `ease-out`. Notice the difference between how the elements move during the animation

but end together, since they share the same `animation-duration` of 2 seconds.

Challenge Seed

```
<style>

.balls {
  border-radius: 50%;
  background: linear-gradient(
    35deg,
    #ccffff,
    #ffcccc
  );
  position: fixed;
  width: 50px;
  height: 50px;
  margin-top: 50px;
  animation-name: bounce;
  animation-duration: 2s;
  animation-iteration-count: infinite;
}
#ball1 {
  left: 27%;
}

#ball2 {
  left: 56%;
}

@keyframes bounce {
  0% {
    top: 0px;
  }
  100% {
    top: 249px;
  }
}

</style>

<div class="balls" id="ball1"></div>
<div class="balls" id="ball2"></div>
```

Solution

```
// solution required
```

50. Learn How Bezier Curves Work

Description

The last challenge introduced the `animation-timing-function` property and a few keywords that change the speed of an animation over its duration. CSS offers an option other than keywords that provides even finer control over how the animation plays out, through the use of Bezier curves. In CSS animations, Bezier curves are used with the `cubic-bezier` function. The shape of the curve represents how the animation plays out. The curve lives on a 1 by 1 coordinate system. The X-axis of this coordinate system is the duration of the animation (think of it as a time scale), and the Y-axis is the change in the animation. The `cubic-bezier` function consists of four main points that sit on this 1 by 1 grid: `p0`, `p1`, `p2`, and `p3`. `p0` and `p3` are set for you - they are the beginning and end points which are always located respectively at the origin (0, 0) and (1, 1). You set the x and y values for the other two points, and where you place them in the grid dictates the shape of the curve for the animation to follow. This is done in CSS by declaring the x and y values of the `p1` and `p2` "anchor" points in the form: `(x1, y1, x2, y2)`. Pulling it all together, here's an example of a Bezier curve in CSS code: `animation-timing-function: cubic-bezier(0.25, 0.25, 0.75, 0.75);` In the example above, the x and y values are equivalent for each point ($x1 = 0.25 = y1$ and $x2 = 0.75 = y2$), which if you

remember from geometry class, results in a line that extends from the origin to point (1, 1). This animation is a linear change of an element during the length of an animation, and is the same as using the `linear` keyword. In other words, it changes at a constant speed.

Instructions

For the element with the id of `ball1`, change the value of the `animation-timing-function` property from `linear` to its equivalent `cubic-bezier` function value. Use the point values given in the example above.

Challenge Seed

```
<style>

.balls{
    border-radius: 50%;
    background: linear-gradient(
        35deg,
        #ccffff,
        #ffcccc
    );
    position: fixed;
    width: 50px;
    height: 50px;
    margin-top: 50px;
    animation-name: bounce;
    animation-duration: 2s;
    animation-iteration-count: infinite;
}
#ball1 {
    left: 27%;
    animation-timing-function: linear;
}
#ball2 {
    left: 56%;
    animation-timing-function: ease-out;
}

@keyframes bounce {
    0% {
        top: 0px;
    }
    100% {
        top: 249px;
    }
}

</style>

<div class="balls" id="ball1"></div>
<div class="balls" id="ball2"></div>
```

Solution

```
// solution required
```

51. Use a Bezier Curve to Move a Graphic

Description

A previous challenge discussed the `ease-out` keyword that describes an animation change that speeds up first and then slows down at the end of the animation. On the right, the difference between the `ease-out` keyword (for the blue element) and `linear` keyword (for the red element) is demonstrated. Similar animation progressions to the `ease-out` keyword can be achieved by using a custom cubic Bezier curve function. In general, changing the `p1` and

`p2` anchor points drives the creation of different Bezier curves, which controls how the animation progresses through time. Here's an example of a Bezier curve using values to mimic the ease-out style: `animation-timing-function: cubic-bezier(0, 0, 0.58, 1);` Remember that all `cubic-bezier` functions start with `p0` at (0, 0) and end with `p3` at (1, 1). In this example, the curve moves faster through the Y-axis (starts at 0, goes to `p1` y value of 0, then goes to `p2` y value of 1) than it moves through the X-axis (0 to start, then 0 for `p1`, up to 0.58 for `p2`). As a result, the change in the animated element progresses faster than the time of the animation for that segment. Towards the end of the curve, the relationship between the change in x and y values reverses - the y value moves from 1 to 1 (no change), and the x values move from 0.58 to 1, making the animation changes progress slower compared to the animation duration.

Instructions

To see the effect of this Bezier curve in action, change the `animation-timing-function` of the element with id of `red` to a `cubic-bezier` function with `x1, y1, x2, y2` values set respectively to 0, 0, 0.58, 1. This will make both elements progress through the animation similarly.

Challenge Seed

```
<style>
.balls{
  border-radius: 50%;
  position: fixed;
  width: 50px;
  height: 50px;
  margin-top: 50px;
  animation-name: bounce;
  animation-duration: 2s;
  animation-iteration-count: infinite;
}
#red {
  background: red;
  left: 27%;
  animation-timing-function: linear;
}
#blue {
  background: blue;
  left: 56%;
  animation-timing-function: ease-out;
}
@keyframes bounce {
  0% {
    top: 0px;
  }
  100% {
    top: 249px;
  }
}
</style>
<div class="balls" id= "red"></div>
<div class="balls" id= "blue"></div>
```

Solution

```
// solution required
```

52. Make Motion More Natural Using a Bezier Curve

Description

This challenge animates an element to replicate the movement of a ball being juggled. Prior challenges covered the linear and ease-out cubic Bezier curves, however neither depicts the juggling movement accurately. You need to customize a Bezier curve for this. The `animation-timing-function` automatically loops at every keyframe when the

`animation-iteration-count` is set to infinite. Since there is a keyframe rule set in the middle of the animation duration (at 50%), it results in two identical animation progressions at the upward and downward movement of the ball. The following cubic Bezier curve simulates a juggling movement: `cubic-bezier(0.3, 0.4, 0.5, 1.6)`; Notice that the value of `y2` is larger than 1. Although the cubic Bezier curve is mapped on an 1 by 1 coordinate system, and it can only accept `x` values from 0 to 1, the `y` value can be set to numbers larger than one. This results in a bouncing movement that is ideal for simulating the juggling ball.

Instructions

Change value of the `animation-timing-function` of the element with the id of `green` to a `cubic-bezier` function with `x1`, `y1`, `x2`, `y2` values set respectively to 0.311, 0.441, 0.444, 1.649.

Challenge Seed

```
<style>
.balls {
  border-radius: 50%;
  position: fixed;
  width: 50px;
  height: 50px;
  top: 60%;
  animation-name: jump;
  animation-duration: 2s;
  animation-iteration-count: infinite;
}
#red {
  background: red;
  left: 25%;
  animation-timing-function: linear;
}
#blue {
  background: blue;
  left: 50%;
  animation-timing-function: ease-out;
}
#green {
  background: green;
  left: 75%;
  animation-timing-function: cubic-bezier(0.69, 0.1, 1, 0.1);
}

@keyframes jump {
  50% {
    top: 10%;
  }
}
</style>


</div>


</div>


</div>


```

Solution

```
// solution required
```

Applied Accessibility

1. Add a Text Alternative to Images for Visually Impaired Accessibility

Description

It's likely that you've seen an `alt` attribute on an `img` tag in other challenges. Alt text describes the content of the image and provides a text-alternative for it. This helps in cases where the image fails to load or can't be seen by a user. It's also used by search engines to understand what an image contains to include it in search results. Here's an example: `` People with visual impairments rely on screen readers to convert web content to an audio interface. They won't get information if it's only presented visually. For images, screen readers can access the `alt` attribute and read its contents to deliver key information. Good alt text provides the reader a brief description of the image. You should always include an `alt` attribute on your image. Per HTML5 specification, this is now considered mandatory.

Instructions

Camper Cat happens to be both a coding ninja and an actual ninja, who is building a website to share his knowledge. The profile picture he wants to use shows his skills and should be appreciated by all site visitors. Add an `alt` attribute in the `img` tag, that explains Camper Cat is doing karate. (The image `src` doesn't link to an actual file, so you should see the `alt` text in the display.)

Challenge Seed

```

```

Solution

```

```

2. Know When Alt Text Should be Left Blank

Description

In the last challenge, you learned that including an `alt` attribute on `img` tags is mandatory. However, sometimes images are grouped with a caption already describing them, or are used for decoration only. In these cases `alt` text may seem redundant or unnecessary. In situations when an image is already explained with text content, or does not add meaning to a page, the `img` still needs an `alt` attribute, but it can be set to an empty string. Here's an example: `` Background images usually fall under the 'decorative' label as well. However, they are typically applied with CSS rules, and therefore not part of the markup screen readers process. **Note** For images with a caption, you may still want to include `alt` text, since it helps search engines catalog the content of the image.

Instructions

Camper Cat has coded a skeleton page for the blog part of his website. He's planning to add a visual break between his two articles with a decorative image of a samurai sword. Add an `alt` attribute to the `img` tag and set it to an empty string. (Note that the image `src` doesn't link to an actual file - don't worry that there are no swords showing in the display.)

Challenge Seed

```
<h1>Deep Thoughts with Master Camper Cat</h1>
<article>
  <h2>Defeating your Foe: the Red Dot is Ours!</h2>
  <p>To Come...</p>
</article>



<article>
  <h2>Is Chuck Norris a Cat Person?</h2>
```

```
<p>To Come...</p>
</article>
```

Solution

```
// solution required
```

3. Use Headings to Show Hierarchical Relationships of Content

Description

Headings (`h1` through `h6` elements) are workhorse tags that help provide structure and labeling to your content. Screen readers can be set to read only the headings on a page so the user gets a summary. This means it is important for the heading tags in your markup to have semantic meaning and relate to each other, not be picked merely for their size values. *Semantic meaning* means that the tag you use around content indicates the type of information it contains. If you were writing a paper with an introduction, a body, and a conclusion, it wouldn't make much sense to put the conclusion as a subsection of the body in your outline. It should be its own section. Similarly, the heading tags in a webpage need to go in order and indicate the hierarchical relationships of your content. Headings with equal (or higher) rank start new implied sections, headings with lower rank start subsections of the previous one. As an example, a page with an `h2` element followed by several subsections labeled with `h4` tags would confuse a screen reader user. With six choices, it's tempting to use a tag because it looks better in a browser, but you can use CSS to edit the relative sizing. One final point, each page should always have one (and only one) `h1` element, which is the main subject of your content. This and the other headings are used in part by search engines to understand the topic of the page.

Instructions

Camper Cat wants a page on his site dedicated to becoming a ninja. Help him fix the headings so his markup gives semantic meaning to the content, and shows the proper parent-child relationships of his sections. Change all the `h5` tags to the proper heading level to indicate they are subsections of the `h2` ones. Use `h3` tags for the purpose.

Challenge Seed

```
<h1>How to Become a Ninja</h1>
<main>
  <h2>Learn the Art of Moving Stealthily</h2>
  <h5>How to Hide in Plain Sight</h5>
  <h5>How to Climb a Wall</h5>

  <h2>Learn the Art of Battle</h2>
  <h5>How to Strengthen your Body</h5>
  <h5>How to Fight like a Ninja</h5>

  <h2>Learn the Art of Living with Honor</h2>
  <h5>How to Breathe Properly</h5>
  <h5>How to Simplify your Life</h5>
</main>
```

Solution

```
// solution required
```

4. Jump Straight to the Content Using the `main` Element

Description

HTML5 introduced a number of new elements that give developers more options while also incorporating accessibility features. These tags include `main`, `header`, `footer`, `nav`, `article`, and `section`, among others. By default, a browser renders these elements similarly to the humble `div`. However, using them where appropriate gives additional meaning in your markup. The tag name alone can indicate the type of information it contains, which adds semantic meaning to that content. Assistive technologies can access this information to provide better page summary or navigation options to their users. The `main` element is used to wrap (you guessed it) the main content, and there should be only one per page. It's meant to surround the information that's related to the central topic of your page. It's not meant to include items that repeat across pages, like navigation links or banners. The `main` tag also has an embedded landmark feature that assistive technology can use to quickly navigate to the main content. If you've ever seen a "Jump to Main Content" link at the top of a page, using a `main` tag automatically gives assistive devices that functionality.

Instructions

Camper Cat has some big ideas for his ninja weapons page. Help him set up his markup by adding opening and closing `main` tags between the `header` and `footer` (covered in other challenges). Keep the `main` tags empty for now.

Challenge Seed

```
<header>
  <h1>Weapons of the Ninja</h1>
</header>

<main></main>

<footer></footer>
```

Solution

```
// solution required
```

5. Wrap Content in the `article` Element

Description

`article` is another one of the new HTML5 elements that adds semantic meaning to your markup. `Article` is a sectioning element, and is used to wrap independent, self-contained content. The tag works well with blog entries, forum posts, or news articles. Determining whether content can stand alone is usually a judgement call, but there are a couple simple tests you can use. Ask yourself if you removed all surrounding context, would that content still make sense? Similarly for text, would the content hold up if it were in an RSS feed? Remember that folks using assistive technologies rely on organized, semantically meaningful markup to better understand your work. **Note about `section` and `div`**

The `section` element is also new with HTML5, and has a slightly different semantic meaning than `article`. An `article` is for standalone content, and a `section` is for grouping thematically related content. They can be used within each other, as needed. For example, if a book is the `article`, then each chapter is a `section`. When there's no relationship between groups of content, then use a `div`.

```
<div> - groups content
<section> - groups related content
<article> - groups independent, self-contained content
```

Instructions

Camper Cat used `article` tags to wrap the posts on his blog page, but he forgot to use them around the top one. Change the `div` tag to use an `article` tag instead.

Challenge Seed

```
<h1>Deep Thoughts with Master Camper Cat</h1>
<main>
  <div>
    <h2>The Garfield Files: Lasagna as Training Fuel?</h2>
    <p>The internet is littered with varying opinions on nutritional paradigms, from catnip paleo to hairball cleanses. But let's turn our attention to an often overlooked fitness fuel, and examine the protein-carb-NOM trifecta that is lasagna...</p>
  </div>

  <article>
    <h2>Defeating your Foe: the Red Dot is Ours!</h2>
    <p>Felines the world over have been waging war on the most persistent of foes. This red nemesis combines both cunning stealth and lightening speed. But chin up, fellow fighters, our time for victory may soon be near...</p>
  </article>

  <article>
    <h2>Is Chuck Norris a Cat Person?</h2>
    <p>Chuck Norris is widely regarded as the premier martial artist on the planet, and it's a complete coincidence anyone who disagrees with this fact mysteriously disappears soon after. But the real question is, is he a cat person?...</p>
  </article>
</main>
```

Solution

```
// solution required
```

6. Make Screen Reader Navigation Easier with the header Landmark

Description

The next HTML5 element that adds semantic meaning and improves accessibility is the `header` tag. It's used to wrap introductory information or navigation links for its parent tag and works well around content that's repeated at the top on multiple pages. `header` shares the embedded landmark feature you saw with `main`, allowing assistive technologies to quickly navigate to that content. **Note**

`header` is meant for use in the `body` tag of your HTML document. This is different than the `head` element, which contains the page's title, meta information, etc.

Instructions

Camper Cat is writing some great articles about ninja training, and wants to add a page for them to his site. Change the top `div` that currently contains the `h1` to a `header` tag instead.

Challenge Seed

```
<body>
  <div>
    <h1>Training with Camper Cat</h1>
```

```

</div>

<main>
  <section id="stealth">
    <h2>Stealth & Agility Training</h2>
    <article><h3>Climb foliage quickly using a minimum spanning tree approach</h3></article>
    <article><h3>No training is NP-complete without parkour</h3></article>
  </section>
  <section id="combat">
    <h2>Combat Training</h2>
    <article><h3>Dispatch multiple enemies with multithreaded tactics</h3></article>
    <article><h3>Goodbye world: 5 proven ways to knock out an opponent</h3></article>
  </section>
  <section id="weapons">
    <h2>Weapons Training</h2>
    <article><h3>Swords: the best tool to literally divide and conquer</h3></article>
    <article><h3>Breadth-first or depth-first in multi-weapon training?</h3></article>
  </section>
</main>
</body>

```

Solution

```
// solution required
```

7. Make Screen Reader Navigation Easier with the nav Landmark

Description

The `nav` element is another HTML5 item with the embedded landmark feature for easy screen reader navigation. This tag is meant to wrap around the main navigation links in your page. If there are repeated site links at the bottom of the page, it isn't necessary to markup those with a `nav` tag as well. Using a `footer` (covered in the next challenge) is sufficient.

Instructions

Camper Cat included navigation links at the top of his training page, but wrapped them in a `div`. Change the `div` to a `nav` tag to improve the accessibility on his page.

Challenge Seed

```

<body>
  <header>
    <h1>Training with Camper Cat</h1>

    <div>
      <ul>
        <li><a href="#stealth">Stealth & Agility</a></li>
        <li><a href="#combat">Combat</a></li>
        <li><a href="#weapons">Weapons</a></li>
      </ul>
    </div>

  </header>
  <main>
    <section id="stealth">
      <h2>Stealth & Agility Training</h2>
      <article><h3>Climb foliage quickly using a minimum spanning tree approach</h3></article>
      <article><h3>No training is NP-complete without parkour</h3></article>
    </section>
    <section id="combat">

```

```

<h2>Combat Training</h2>
<article><h3>Dispatch multiple enemies with multithreaded tactics</h3></article>
<article><h3>Goodbye world: 5 proven ways to knock out an opponent</h3></article>
</section>
<section id="weapons">
  <h2>Weapons Training</h2>
  <article><h3>Swords: the best tool to literally divide and conquer</h3></article>
  <article><h3>Breadth-first or depth-first in multi-weapon training?</h3></article>
</section>
</main>
</body>

```

Solution

```
// solution required
```

8. Make Screen Reader Navigation Easier with the footer Landmark

Description

Similar to `header` and `nav`, the `footer` element has a built-in landmark feature that allows assistive devices to quickly navigate to it. It's primarily used to contain copyright information or links to related documents that usually sit at the bottom of a page.

Instructions

Camper Cat's training page is making good progress. Change the `div` he used to wrap his copyright information at the bottom of the page to a `footer` element.

Challenge Seed

```

<body>
  <header>
    <h1>Training</h1>
    <nav>
      <ul>
        <li><a href="#stealth">Stealth & Agility</a></li>
        <li><a href="#combat">Combat</a></li>
        <li><a href="#weapons">Weapons</a></li>
      </ul>
    </nav>
  </header>
  <main>
    <section id="stealth">
      <h2>Stealth & Agility Training</h2>
      <article><h3>Climb foliage quickly using a minimum spanning tree approach</h3></article>
      <article><h3>No training is NP-complete without parkour</h3></article>
    </section>
    <section id="combat">
      <h2>Combat Training</h2>
      <article><h3>Dispatch multiple enemies with multithreaded tactics</h3></article>
      <article><h3>Goodbye world: 5 proven ways to knock out an opponent</h3></article>
    </section>
    <section id="weapons">
      <h2>Weapons Training</h2>
      <article><h3>Swords: the best tool to literally divide and conquer</h3></article>
      <article><h3>Breadth-first or depth-first in multi-weapon training?</h3></article>
    </section>
  </main>

  <div>&copy; 2018 Camper Cat</div>

```

```
</body>
```

Solution

```
// solution required
```

9. Improve Accessibility of Audio Content with the audio Element

Description

HTML5's `audio` element gives semantic meaning when it wraps sound or audio stream content in your markup. Audio content also needs a text alternative to be accessible to people who are deaf or hard of hearing. This can be done with nearby text on the page or a link to a transcript. The `audio` tag supports the `controls` attribute. This shows the browser default play, pause, and other controls, and supports keyboard functionality. This is a boolean attribute, meaning it doesn't need a value, its presence on the tag turns the setting on. Here's an example:

```
<audio id="meowClip" controls>
  <source src="audio/meow.mp3" type="audio/mpeg" />
  <source src="audio/meow.ogg" type="audio/ogg" />
</audio>
```

Note

Multimedia content usually has both visual and auditory components. It needs synchronized captions and a transcript so users with visual and/or auditory impairments can access it. Generally, a web developer is not responsible for creating the captions or transcript, but needs to know to include them.

Instructions

Time to take a break from Camper Cat and meet fellow camper Zersiax (@zersiax), a champion of accessibility and a screen reader user. To hear a clip of his screen reader in action, add an `audio` element after the `p`. Include the `controls` attribute. Then place a `source` tag inside the `audio` tags with the `src` attribute set to "<https://s3.amazonaws.com/freecodecamp/screen-reader.mp3>" and `type` attribute set to "audio/mpeg". **Note** The audio clip may sound fast and be difficult to understand, but that is a normal speed for screen reader users.

Challenge Seed

```
<body>
  <header>
    <h1>Real Coding Ninjas</h1>
  </header>
  <main>
    <p>A sound clip of Zersiax's screen reader in action.</p>
    </main>
</body>
```

Solution

```
// solution required
```

10. Improve Chart Accessibility with the figure Element

Description

HTML5 introduced the `figure` element, along with the related `figcaption`. Used together, these items wrap a visual representation (like an image, diagram, or chart) along with its caption. This gives a two-fold accessibility boost by both semantically grouping related content, and providing a text alternative that explains the `figure`. For data visualizations like charts, the caption can be used to briefly note the trends or conclusions for users with visual impairments. Another challenge covers how to move a table version of the chart's data off-screen (using CSS) for screen reader users. Here's an example - note that the `figcaption` goes inside the `figure` tags and can be combined with other elements:

```
<figure>
  
  <br>
  <figcaption>
    Master Camper Cat demonstrates proper form of a roundhouse kick.
  </figcaption>
</figure>
```

Instructions

Camper Cat is hard at work creating a stacked bar chart showing the amount of time per week to spend training in stealth, combat, and weapons. Help him structure his page better by changing the `div` tag he used to a `figure` tag, and the `p` tag that surrounds the caption to a `figcaption` tag.

Challenge Seed

```
<body>
  <header>
    <h1>Training</h1>
    <nav>
      <ul>
        <li><a href="#stealth">Stealth & Agility</a></li>
        <li><a href="#combat">Combat</a></li>
        <li><a href="#weapons">Weapons</a></li>
      </ul>
    </nav>
  </header>
  <main>
    <section>

      <!-- Add your code below this line -->
      <div>
        <!-- Stacked bar chart will go here -->
        <br>
        <p>Breakdown per week of time to spend training in stealth, combat, and weapons.</p>
      </div>
      <!-- Add your code above this line -->

    </section>
    <section id="stealth">
      <h2>Stealth & Agility Training</h2>
      <article><h3>Climb foliage quickly using a minimum spanning tree approach</h3></article>
      <article><h3>No training is NP-complete without parkour</h3></article>
    </section>
    <section id="combat">
      <h2>Combat Training</h2>
      <article><h3>Dispatch multiple enemies with multithreaded tactics</h3></article>
      <article><h3>Goodbye world: 5 proven ways to knock out an opponent</h3></article>
    </section>
    <section id="weapons">
      <h2>Weapons Training</h2>
      <article><h3>Swords: the best tool to literally divide and conquer</h3></article>
      <article><h3>Breadth-first or depth-first in multi-weapon training?</h3></article>
    </section>
  </main>
  <footer>&copy; 2018 Camper Cat</footer>
</body>
```

Solution

```
// solution required
```

11. Improve Form Field Accessibility with the label Element

Description

Improving accessibility with semantic HTML markup applies to using both appropriate tag names as well as attributes. The next several challenges cover some important scenarios using attributes in forms. The `label` tag wraps the text for a specific form control item, usually the name or label for a choice. This ties meaning to the item and makes the form more readable. The `for` attribute on a `label` tag explicitly associates that `label` with the form control and is used by screen readers. You learned about radio buttons and their labels in a lesson in the Basic HTML section. In that lesson, we wrapped the radio button input element inside a `label` element along with the label text in order to make the text clickable. Another way to achieve this is by using the `for` attribute as explained in this lesson. The value of the `for` attribute must be the same as the value of the `id` attribute of the form control. Here's an example:

```
<form>
  <label for="name">Name:</label>
  <input type="text" id="name" name="name">
</form>
```

Instructions

Camper Cat expects a lot of interest in his thoughtful blog posts and wants to include an email sign up form. Add a `for` attribute on the email `label` that matches the `id` on its `input` field.

Challenge Seed

```
<body>
  <header>
    <h1>Deep Thoughts with Master Camper Cat</h1>
  </header>
  <section>
    <form>
      <p>Sign up to receive Camper Cat's blog posts by email here!</p>

      <label>Email:</label>
      <input type="text" id="email" name="email">

      <input type="submit" name="submit" value="Submit">
    </form>
  </section>
  <article>
    <h2>The Garfield Files: Lasagna as Training Fuel?</h2>
    <p>The internet is littered with varying opinions on nutritional paradigms, from catnip paleo to hairball cleanses. But let's turn our attention to an often overlooked fitness fuel, and examine the protein-carb-NOM trifecta that is lasagna...</p>
  </article>
  
  <article>
    <h2>Defeating your Foe: the Red Dot is Ours!</h2>
    <p>Felines the world over have been waging war on the most persistent of foes. This red nemesis combines both cunning stealth and lightening speed. But chin up, fellow fighters, our time for victory may soon be near...</p>
  </article>
  
  <article>
    <h2>Is Chuck Norris a Cat Person?</h2>
    <p>Chuck Norris is widely regarded as the premier martial artist on the planet, and it's a complete
```

```
coincidence anyone who disagrees with this fact mysteriously disappears soon after. But the real
question is, is he a cat person?...</p>
</article>
<footer>&copy; 2018 Camper Cat</footer>
</body>
```

Solution

```
// solution required
```

12. Wrap Radio Buttons in a fieldset Element for Better Accessibility

Description

The next form topic covers accessibility of radio buttons. Each choice is given a `label` with a `for` attribute tying to the `id` of the corresponding item as covered in the last challenge. Since radio buttons often come in a group where the user must choose one, there's a way to semantically show the choices are part of a set. The `fieldset` tag surrounds the entire grouping of radio buttons to achieve this. It often uses a `legend` tag to provide a description for the grouping, which is read by screen readers for each choice in the `fieldset` element. The `fieldset` wrapper and `legend` tag are not necessary when the choices are self-explanatory, like a gender selection. Using a `label` with the `for` attribute for each radio button is sufficient. Here's an example:

```
<form>
  <fieldset>
    <legend>Choose one of these three items:</legend>
    <input id="one" type="radio" name="items" value="one">
    <label for="one">Choice One</label><br>
    <input id="two" type="radio" name="items" value="two">
    <label for="two">Choice Two</label><br>
    <input id="three" type="radio" name="items" value="three">
    <label for="three">Choice Three</label>
  </fieldset>
</form>
```

Instructions

Camper Cat wants information about the ninja level of his users when they sign up for his email list. He's added a set of radio buttons and learned from our last lesson to use `label` tags with `for` attributes for each choice. Go Camper Cat! However, his code still needs some help. Change the `div` tag surrounding the radio buttons to a `fieldset` tag, and change the `p` tag inside it to a `legend`.

Challenge Seed

```
<body>
  <header>
    <h1>Deep Thoughts with Master Camper Cat</h1>
  </header>
  <section>
    <form>
      <p>Sign up to receive Camper Cat's blog posts by email here!</p>
      <label for="email">Email:</label>
      <input type="text" id="email" name="email">

      <!-- Add your code below this line -->
      <div>
        <p>What level ninja are you?</p>
        <input id="newbie" type="radio" name="levels" value="newbie">
        <label for="newbie">Newbie Kitten</label><br>
```

```

<input id="intermediate" type="radio" name="levels" value="intermediate">
<label for="intermediate">Developing Student</label><br>
<input id="master" type="radio" name="levels" value="master">
<label for="master">Master</label>
</div>
<!-- Add your code above this line --&gt;

&lt;input type="submit" name="submit" value="Submit"&gt;
&lt;/form&gt;
&lt;/section&gt;
&lt;article&gt;
&lt;h2&gt;The Garfield Files: Lasagna as Training Fuel?&lt;/h2&gt;
&lt;p&gt;The internet is littered with varying opinions on nutritional paradigms, from catnip paleo to hairball cleanses. But let's turn our attention to an often overlooked fitness fuel, and examine the protein-carb-NOM trifecta that is lasagna...&lt;/p&gt;
&lt;/article&gt;
&lt;img src="samuraiSwords.jpeg" alt=""&gt;
&lt;article&gt;
&lt;h2&gt;Defeating your Foe: the Red Dot is Ours!&lt;/h2&gt;
&lt;p&gt;Felines the world over have been waging war on the most persistent of foes. This red nemesis combines both cunning stealth and lightening speed. But chin up, fellow fighters, our time for victory may soon be near...&lt;/p&gt;
&lt;/article&gt;
&lt;img src="samuraiSwords.jpeg" alt=""&gt;
&lt;article&gt;
&lt;h2&gt;Is Chuck Norris a Cat Person?&lt;/h2&gt;
&lt;p&gt;Chuck Norris is widely regarded as the premier martial artist on the planet, and it's a complete coincidence anyone who disagrees with this fact mysteriously disappears soon after. But the real question is, is he a cat person?...&lt;/p&gt;
&lt;/article&gt;
&lt;footer&gt;&amp;copy; 2018 Camper Cat&lt;/footer&gt;
&lt;/body&gt;
</pre>

```

Solution

```
// solution required
```

13. Add an Accessible Date Picker

Description

Forms often include the `input` field, which can be used to create several different form controls. The `type` attribute on this element indicates what kind of input will be created. You may have noticed the `text` and `submit` input types in prior challenges, and HTML5 introduced an option to specify a `date` field. Depending on browser support, a date picker shows up in the `input` field when it's in focus, which makes filling in a form easier for all users. For older browsers, the type will default to `text`, so it helps to show users the expected date format in the label or as placeholder text just in case. Here's an example:

```
<label for="input1">Enter a date:</label>
<input type="date" id="input1" name="input1">
```

Instructions

Camper Cat is setting up a Mortal Kombat tournament and wants to ask his competitors to see what date works best. Add an `input` tag with a `type` attribute of "date", an `id` attribute of "pickdate", and a `name` attribute of "date".

Challenge Seed

```

<body>
<header>
  <h1>Tournaments</h1>
</header>
<main>
```

```

<section>
  <h2>Mortal Kombat Tournament Survey</h2>
  <form>
    <p>Tell us the best date for the competition</p>
    <label for="pickdate">Preferred Date:</label>

    <!-- Add your code below this line -->

    <!-- Add your code above this line -->

    <input type="submit" name="submit" value="Submit">
  </form>
</section>
</main>
<footer>&copy; 2018 Camper Cat</footer>
</body>

```

Solution

```

<body>
  <header>
    <h1>Tournaments</h1>
  </header>
  <main>
    <section>
      <h2>Mortal Kombat Tournament Survey</h2>
      <form>
        <p>Tell us the best date for the competition</p>
        <label for="pickdate">Preferred Date:</label>
        <input type="date" id="pickdate" name="date">
        <input type="submit" name="submit" value="Submit">
      </form>
    </section>
  </main>
  <footer>&copy; 2018 Camper Cat</footer>
</body>

```

14. Standardize Times with the HTML5 datetime Attribute

Description

Continuing with the date theme, HTML5 also introduced the `time` element along with a `datetime` attribute to standardize times. This is an inline element that can wrap a date or time on a page. A valid format of that date is held by the `datetime` attribute. This is the value accessed by assistive devices. It helps avoid confusion by stating a standardized version of a time, even if it's written in an informal or colloquial manner in the text. Here's an example:

```
<p>Master Camper Cat officiated the cage match between Goro and Scorpion <time datetime="2013-02-13">last Wednesday</time>, which ended in a draw.</p>
```

Instructions

Camper Cat's Mortal Kombat survey results are in! Wrap a `time` tag around the text "Thursday, September 15th" and add a `datetime` attribute to it set to "2016-09-15".

Challenge Seed

```

<body>
  <header>
    <h1>Tournaments</h1>
  </header>
  <article>

```

```

<h2>Mortal Kombat Tournament Survey Results</h2>

<!-- Add your code below this line -->

<p>Thank you to everyone for responding to Master Camper Cat's survey. The best day to host the
vaunted Mortal Kombat tournament is Thursday, September 15<sup>th</sup>. May the best ninja win!</p>

<!-- Add your code above this line -->

<section>
  <h3>Comments:</h3>
  <article>
    <p>Posted by: Sub-Zero on <time datetime="2016-08-13T20:01Z">August 13<sup>th</sup></time></p>
    <p>Johnny Cage better be there, I'll finish him!</p>
  </article>
  <article>
    <p>Posted by: Doge on <time datetime="2016-08-15T08:12Z">August 15<sup>th</sup></time></p>
    <p>Wow, much combat, so mortal.</p>
  </article>
  <article>
    <p>Posted by: The Grim Reaper on <time datetime="2016-08-16T00:00Z">August 16<sup>th</sup>
    </time></p>
    <p>Looks like I'll be busy that day.</p>
  </article>
</section>
</article>
<footer>&copy; 2018 Camper Cat</footer>
</body>

```

Solution

```
// solution required
```

15. Make Elements Only Visible to a Screen Reader by Using Custom CSS

Description

Have you noticed that all of the applied accessibility challenges so far haven't used any CSS? This is to show the importance of a logical document outline, and using semantically meaningful tags around your content before introducing the visual design aspect. However, CSS's magic can also improve accessibility on your page when you want to visually hide content meant only for screen readers. This happens when information is in a visual format (like a chart), but screen reader users need an alternative presentation (like a table) to access the data. CSS is used to position the screen reader-only elements off the visual area of the browser window. Here's an example of the CSS rules that accomplish this:

```
.sr-only {
  position: absolute;
  left: -1000px;
  width: 1px;
  height: 1px;
  top: auto;
  overflow: hidden;
}
```

Note

The following CSS approaches will NOT do the same thing:

- `display: none;` or `visibility: hidden;` hides content for everyone, including screen reader users
- Zero values for pixel sizes, such as `width: 0px;` `height: 0px;` removes that element from the flow of your document, meaning screen readers will ignore it

Instructions

Camper Cat created a really cool stacked bar chart for his training page, and put the data into a table for his visually impaired users. The table already has an `sr-only` class, but the CSS rules aren't filled in yet. Give the position an absolute value, the left a -10000px value, and the width and height both 1px values.

Challenge Seed

```
<head>
  <style>
    .sr-only {
      position: ;
      left: ;
      width: ;
      height: ;
      top: auto;
      overflow: hidden;
    }
  </style>
</head>
<body>
  <header>
    <h1>Training</h1>
    <nav>
      <ul>
        <li><a href="#stealth">Stealth & Agility</a></li>
        <li><a href="#combat">Combat</a></li>
        <li><a href="#weapons">Weapons</a></li>
      </ul>
    </nav>
  </header>
  <section>
    <h2>Master Camper Cat's Beginner Three Week Training Program</h2>
    <figure>
      <!-- Stacked bar chart of weekly training-->
      <p>[Stacked bar chart]</p>
      <br />
      <figcaption>Breakdown per week of time to spend training in stealth, combat, and weapons.</figcaption>
    </figure>
    <table class="sr-only">
      <caption>Hours of Weekly Training in Stealth, Combat, and Weapons</caption>
      <thead>
        <tr>
          <th></th>
          <th scope="col">Stealth & Agility</th>
          <th scope="col">Combat</th>
          <th scope="col">Weapons</th>
          <th scope="col">Total</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <th scope="row">Week One</th>
          <td>3</td>
          <td>5</td>
          <td>2</td>
          <td>10</td>
        </tr>
        <tr>
          <th scope="row">Week Two</th>
          <td>4</td>
          <td>5</td>
          <td>3</td>
          <td>12</td>
        </tr>
        <tr>
          <th scope="row">Week Three</th>
          <td>4</td>
          <td>6</td>
          <td>3</td>
          <td>13</td>
        </tr>
      </tbody>
    </table>
  </section>
```

```

<section id="stealth">
  <h2>Stealth & Agility Training</h2>
  <article><h3>Climb foliage quickly using a minimum spanning tree approach</h3></article>
  <article><h3>No training is NP-complete without parkour</h3></article>
</section>
<section id="combat">
  <h2>Combat Training</h2>
  <article><h3>Dispatch multiple enemies with multithreaded tactics</h3></article>
  <article><h3>Goodbye, world: 5 proven ways to knock out an opponent</h3></article>
</section>
<section id="weapons">
  <h2>Weapons Training</h2>
  <article><h3>Swords: the best tool to literally divide and conquer</h3></article>
  <article><h3>Breadth-first or depth-first in multi-weapon training?</h3></article>
</section>
<footer>&copy; 2018 Camper Cat</footer>
</body>

```

Solution

// solution required

16. Improve Readability with High Contrast Text

Description

Low contrast between the foreground and background colors can make text difficult to read. Sufficient contrast improves the readability of your content, but what exactly does "sufficient" mean? The Web Content Accessibility Guidelines (WCAG) recommend at least a 4.5 to 1 contrast ratio for normal text. The ratio is calculated by comparing the relative luminance values of two colors. This ranges from 1:1 for the same color, or no contrast, to 21:1 for white against black, the strongest contrast. There are many contrast checking tools available online that calculate this ratio for you.

Instructions

Camper Cat's choice of light gray text on a white background for his recent blog post has a 1.5:1 contrast ratio, making it hard to read. Change the `color` of the text from the current gray (`#D3D3D3`) to a darker gray (`#636363`) to improve the contrast ratio to 6:1.

Challenge Seed

```

<head>
  <style>
    body {
      color: #D3D3D3;
      background-color: #FFF;
    }
  </style>
</head>
<body>
  <header>
    <h1>Deep Thoughts with Master Camper Cat</h1>
  </header>
  <article>
    <h2>A Word on the Recent Catnip Doping Scandal</h2>
    <p>The influence that catnip has on feline behavior is well-documented, and its use as an herbal supplement in competitive ninja circles remains controversial. Once again, the debate to ban the substance is brought to the public's attention after the high-profile win of Kittytron, a long-time proponent and user of the green stuff, at the Claw of Fury tournament.</p>
    <p>As I've stated in the past, I firmly believe a true ninja's skills must come from within, with no external influences. My own catnip use shall continue as purely recreational.</p>
  </article>
</body>

```

Solution

```
// solution required
```

17. Avoid Colorblindness Issues by Using Sufficient Contrast

Description

Color is a large part of visual design, but its use introduces two accessibility issues. First, color alone should not be used as the only way to convey important information because screen reader users won't see it. Second, foreground and background colors need sufficient contrast so colorblind users can distinguish them. Previous challenges covered having text alternatives to address the first issue. The last challenge introduced contrast checking tools to help with the second. The WCAG-recommended contrast ratio of 4.5:1 applies for color use as well as gray-scale combinations. Colorblind users have trouble distinguishing some colors from others - usually in hue but sometimes lightness as well. You may recall the contrast ratio is calculated using the relative luminance (or lightness) values of the foreground and background colors. In practice, the 4.5:1 contrast ratio can be reached by shading (adding black to) the darker color and tinting (adding white to) the lighter color. Darker shades on the color wheel are considered to be shades of blues, violets, magentas, and reds, whereas lighter tinted colors are oranges, yellows, greens, and blue-greens.

Instructions

Camper Cat is experimenting with using color for his blog text and background, but his current combination of a greenish `background-color` with maroon text `color` has a 2.5:1 contrast ratio. You can easily adjust the lightness of the colors since he declared them using the CSS `hsl()` property (which stands for hue, saturation, lightness) by changing the third argument. Increase the `background-color` lightness value from 35% to 55%, and decrease the `color` lightness value from 20% to 15%. This improves the contrast to 5.9:1.

Challenge Seed

```
<head>
  <style>
    body {
      color: hsl(0, 55%, 20%);
      background-color: hsl(120, 25%, 35%);
    }
  </style>
</head>
<body>
  <header>
    <h1>Deep Thoughts with Master Camper Cat</h1>
  </header>
  <article>
    <h2>A Word on the Recent Catnip Doping Scandal</h2>
    <p>The influence that catnip has on feline behavior is well-documented, and its use as an herbal supplement in competitive ninja circles remains controversial. Once again, the debate to ban the substance is brought to the public's attention after the high-profile win of Kittytron, a long-time proponent and user of the green stuff, at the Claw of Fury tournament.</p>
    <p>As I've stated in the past, I firmly believe a true ninja's skills must come from within, with no external influences. My own catnip use shall continue as purely recreational.</p>
  </article>
</body>
```

Solution

```
var code = "body {color: hsl(0, 55%, 15%); background-color: hsl(120, 25%, 55%);}"
```

18. Avoid Colorblindness Issues by Carefully Choosing Colors that Convey Information

Description

There are various forms of colorblindness. These can range from a reduced sensitivity to a certain wavelength of light to the inability to see color at all. The most common form is a reduced sensitivity to detect greens. For example, if two similar green colors are the foreground and background color of your content, a colorblind user may not be able to distinguish them. Close colors can be thought of as neighbors on the color wheel, and those combinations should be avoided when conveying important information. **Note**

Some online color picking tools include visual simulations of how colors appear for different types of colorblindness. These are great resources in addition to online contrast checking calculators.

Instructions

Camper Cat is testing different styles for an important button, but the yellow (#FFFF33) background-color and the green (#33FF33) text color are neighboring hues on the color wheel and virtually indistinguishable for some colorblind users. (Their similar lightness also fails the contrast ratio check). Change the text color to a dark blue (#003366) to solve both problems.

Challenge Seed

```
<head>
  <style>
    button {
      color: #33FF33;
      background-color: #FFFF33;
      font-size: 14px;
      padding: 10px;
    }
  </style>
</head>
<body>
  <header>
    <h1>Danger!</h1>
  </header>
  <button>Delete Internet</button>
</body>
```

Solution

```
// solution required
```

19. Give Links Meaning by Using Descriptive Link Text

Description

Screen reader users have different options for what type of content their device reads. This includes skipping to (or over) landmark elements, jumping to the main content, or getting a page summary from the headings. Another option is to only hear the links available on a page. Screen readers do this by reading the link text, or what's between the anchor (a) tags. Having a list of "click here" or "read more" links isn't helpful. Instead, you should use brief but descriptive text within the a tags to provide more meaning for these users.

Instructions

The link text that Camper Cat is using is not very descriptive without the surrounding context. Move the anchor () tags so they wrap around the text "information about batteries" instead of "Click here".

Challenge Seed

```
<body>
  <header>
    <h1>Deep Thoughts with Master Camper Cat</h1>
  </header>
  <article>
    <h2>Defeating your Foe: the Red Dot is Ours!</h2>
    <p>Felines the world over have been waging war on the most persistent of foes. This red nemesis combines both cunning stealth and lightening speed. But chin up, fellow fighters, our time for victory may soon be near. <a href="">Click here</a> for information about batteries</p>
  </article>
</body>
```

Solution

```
// solution required
```

21. Use tabindex to Add Keyboard Focus to an Element

Description

The HTML `tabindex` attribute has three distinct functions relating to an element's keyboard focus. When it's on a tag, it indicates that element can be focused on. The value (an integer that's positive, negative, or zero) determines the behavior. Certain elements, such as links and form controls, automatically receive keyboard focus when a user tabs through a page. It's in the same order as the elements come in the HTML source markup. This same functionality can be given to other elements, such as `div`, `span`, and `p`, by placing a `tabindex="0"` attribute on them. Here's an example: `<div tabindex="0">I need keyboard focus!</div>` **Note**
A negative `tabindex` value (typically `-1`) indicates that an element is focusable, but is not reachable by the keyboard. This method is generally used to bring focus to content programmatically (like when a `div` used for a pop-up window is activated), and is beyond the scope of these challenges.

Instructions

Camper Cat created a new survey to collect information about his users. He knows input fields automatically get keyboard focus, but he wants to make sure his keyboard users pause at the instructions while tabbing through the items. Add a `tabindex` attribute to the `p` tag and set its value to `"0"`. Bonus - using `tabindex` also enables the CSS pseudo-class `:focus` to work on the `p` tag.

Challenge Seed

```
<head>
  <style>
    p:focus {
      background-color: yellow;
    }
  </style>
</head>
<body>
  <header>
    <h1>Ninja Survey</h1>
  </header>
  <section>
    <form>

      <p>Instructions: Fill in ALL your information then click <b>Submit</b></p>
```

```

<label for="username">Username:</label>
<input type="text" id="username" name="username"><br>
<fieldset>
  <legend>What level ninja are you?</legend>
  <input id="newbie" type="radio" name="levels" value="newbie">
  <label for="newbie">Newbie Kitten</label><br>
  <input id="intermediate" type="radio" name="levels" value="intermediate">
  <label for="intermediate">Developing Student</label><br>
  <input id="master" type="radio" name="levels" value="master">
  <label for="master">9th Life Master</label>
</fieldset>
<br>
<fieldset>
  <legend>Select your favorite weapons:</legend>
  <input id="stars" type="checkbox" name="weapons" value="stars">
  <label for="stars">Throwing Stars</label><br>
  <input id="nunchucks" type="checkbox" name="weapons" value="nunchucks">
  <label for="nunchucks">Nunchucks</label><br>
  <input id="sai" type="checkbox" name="weapons" value="sai">
  <label for="sai">Sai Set</label><br>
  <input id="sword" type="checkbox" name="weapons" value="sword">
  <label for="sword">Sword</label>
</fieldset>
<br>
  <input type="submit" name="submit" value="Submit">
</form><br>
</section>
<footer>&copy; 2018 Camper Cat</footer>
</body>

```

Solution

```
// solution required
```

22. Use tabindex to Specify the Order of Keyboard Focus for Several Elements

Description

The `tabindex` attribute also specifies the exact tab order of elements. This is achieved when the value of the attribute is set to a positive number of 1 or higher. Setting a `tabindex="1"` will bring keyboard focus to that element first. Then it cycles through the sequence of specified `tabindex` values (2, 3, etc.), before moving to default and `tabindex="0"` items. It's important to note that when the tab order is set this way, it overrides the default order (which uses the HTML source). This may confuse users who are expecting to start navigation from the top of the page. This technique may be necessary in some circumstances, but in terms of accessibility, take care before applying it. Here's an example:

```
<div tabindex="1">I get keyboard focus, and I get it first!</div> <div tabindex="2">I get keyboard focus, and I get it second!</div>
```

Instructions

Camper Cat has a search field on his Inspirational Quotes page that he plans to position in the upper right corner with CSS. He wants the search `input` and submit `input` form controls to be the first two items in the tab order. Add a `tabindex` attribute set to "1" to the search `input`, and a `tabindex` attribute set to "2" to the submit `input`.

Challenge Seed

```

<body>
  <header>
    <h1>Even Deeper Thoughts with Master Camper Cat</h1>
    <nav>

```

```

<ul>
  <li><a href="">Home</a></li>
  <li><a href="">Blog</a></li>
  <li><a href="">Training</a></li>
</ul>
</nav>
</header>
<form>
  <label for="search">Search:</label>

  <input type="search" name="search" id="search">
  <input type="submit" name="submit" value="Submit" id="submit">

</form>
<h2>Inspirational Quotes</h2>
<blockquote>
  <p>&ldquo;There's no Theory of Evolution, just a list of creatures I've allowed to live.&rdquo;<br>
    - Chuck Norris</p>
</blockquote>
<blockquote>
  <p>&ldquo;Wise men say forgiveness is divine, but never pay full price for late pizza.&rdquo;<br>
    - TMNT</p>
</blockquote>
<footer>&copy; 2018 Camper Cat</footer>
</body>

```

Solution

```
// solution required
```

Responsive Web Design Principles

1. Create a Media Query

Description

Media Queries are a new technique introduced in CSS3 that change the presentation of content based on different viewport sizes. The viewport is a user's visible area of a web page, and is different depending on the device used to access the site. Media Queries consist of a media type, and if that media type matches the type of device the document is displayed on, the styles are applied. You can have as many selectors and styles inside your media query as you want. Here's an example of a media query that returns the content when the device's width is less than or equal to 100px: `@media (max-width: 100px) { /* CSS Rules */ }` and the following media query returns the content when the device's height is more than or equal to 350px: `@media (min-height: 350px) { /* CSS Rules */ }`

Remember, the CSS inside the media query is applied only if the media type matches that of the device being used.

Instructions

Add a media query, so that the `p` tag has a `font-size` of 10px when the device's height is less than or equal to 800px.

Challenge Seed

```

<style>
  p {
    font-size: 20px;
  }

  /* Add media query below */

```

```
</style>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus quis tempus massa. Aenean erat nisl, gravida vel vestibulum cursus, interdum sit amet lectus. Sed sit amet quam nibh. Suspendisse quis tincidunt nulla. In hac habitasse platea dictumst. Ut sit amet pretium nisl. Vivamus vel mi sem. Aenean sit amet consectetur sem. Suspendisse pretium, purus et gravida consequat, nunc ligula ultricies diam, at aliquet velit libero a dui.</p>
```

Solution

```
<style>
p {
  font-size: 20px;
}

@media (max-height: 800px) {
  p {
    font-size: 10px;
  }
}
</style>
```

```
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus quis tempus massa. Aenean erat nisl, gravida vel vestibulum cursus, interdum sit amet lectus. Sed sit amet quam nibh. Suspendisse quis tincidunt nulla. In hac habitasse platea dictumst. Ut sit amet pretium nisl. Vivamus vel mi sem. Aenean sit amet consectetur sem. Suspendisse pretium, purus et gravida consequat, nunc ligula ultricies diam, at aliquet velit libero a dui.</p>
```

2. Make an Image Responsive

Description

Making images responsive with CSS is actually very simple. Instead of applying an absolute width to an element: `img { width: 720px; }` You can use:

```
img {
  max-width: 100%;
  display: block;
  height: auto;
}
```

The `max-width` property of 100% scales the image to fit the width of its container, but the image won't stretch wider than its original width. Setting the `display` property to `block` changes the image from an inline element (its default), to a block element on its own line. The `height` property of `auto` keeps the original aspect ratio of the image.

Instructions

Add style rules for the `img` tag to make it responsive to the size of its container. It should display as a block-level element, it should fit the full width of its container without stretching, and it should keep its original aspect ratio.

Challenge Seed

```
<style>

</style>


```

Solution

```
// solution required
```

3. Use a Retina Image for Higher Resolution Displays

Description

With the increase of internet connected devices, their sizes and specifications vary, and the displays they use could be different externally and internally. Pixel density is an aspect that could be different on one device from others and this density is known as Pixel Per Inch(PPI) or Dots Per Inch(DPI). The most famous such display is the one known as a "Retina Display" on the latest Apple MacBook Pro notebooks, and recently iMac computers. Due to the difference in pixel density between a "Retina" and "Non-Retina" displays, some images who have not been made with a High-Resolution Display in mind could look "pixelated" when rendered on a High-Resolution display.

The simplest way to make your images properly appear on High-Resolution Displays, such as the MacBook Pros "retina display" is to define their `width` and `height` values as only half of what the original file is. Here is an example of an image that is only using half of the original height and width:

```
<style>
  img { height: 250px; width: 250px; }
</style>

```

Instructions

Set the `width` and `height` of the `img` tag to half of their original values. In this case, both the original `height` and the original `width` are 200px.

Challenge Seed

```
<style>
</style>

```

Solution

```
// solution required
```

4. Make Typography Responsive

Description

Instead of using `em` or `px` to size text, you can use viewport units for responsive typography. Viewport units, like percentages, are relative units, but they are based off different items. Viewport units are relative to the viewport dimensions (width or height) of a device, and percentages are relative to the size of the parent container element. The four different viewport units are:

- `vw`: `10vw` would be 10% of the viewport's width.
- `vh`: `3vh` would be 3% of the viewport's height.
- `vmin`: `70vmin` would be 70% of the viewport's smaller dimension (height vs. width).
- `vmax`: `100vmax` would be 100% of the viewport's bigger dimension (height vs. width).

Here is an example that sets a body tag to 30% of the viewport's width. `body { width: 30vw; }`

Instructions

Set the width of the h2 tag to 80% of the viewport's width and the width of the paragraph as 75% of the viewport's smaller dimension.

Challenge Seed

```
<style>
</style>

<h2>Importantus Ipsum</h2>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus quis tempus massa. Aenean erat nisl, gravida vel vestibulum cursus, interdum sit amet lectus. Sed sit amet quam nibh. Suspendisse quis tincidunt nulla. In hac habitasse platea dictumst. Ut sit amet pretium nisl. Vivamus vel mi sem. Aenean sit amet consectetur sem. Suspendisse pretium, purus et gravida consequat, nunc ligula ultricies diam, at aliquet velit libero a dui.</p>
```

Solution

```
var code = "h2 {width: 80vw;} p {width: 75vmin;}"
```

CSS Flexbox

1. Use display: flex to Position Two Boxes

Description

This section uses alternating challenge styles to show how to use CSS to position elements in a flexible way. First, a challenge will explain theory, then a practical challenge using a simple tweet component will apply the flexbox concept. Placing the CSS property `display: flex;` on an element allows you to use other flex properties to build a responsive page.

Instructions

Add the CSS property `display` to `#box-container` and set its value to `flex`.

Challenge Seed

```
<style>
#box-container {
  height: 500px;
}

#box-1 {
  background-color: dodgerblue;
  width: 50%;
  height: 50%;
}

#box-2 {
  background-color: orangered;
  width: 50%;
  height: 50%;
}
</style>
<div id="box-container">
  <div id="box-1"></div>
  <div id="box-2"></div>
</div>
```

Solution

```
// solution required
```

2. Add Flex Superpowers to the Tweet Embed

Description

To the right is the tweet embed that will be used as the practical example. Some of the elements would look better with a different layout. The last challenge demonstrated `display: flex`. Here you'll add it to several components in the tweet embed to start adjusting their positioning.

Instructions

Add the CSS property `display: flex` to all of the following items - note that the selectors are already set up in the CSS: `header`, the header's `.profile-name`, the header's `.follow-btn`, the header's `h3` and `h4`, the `footer`, and the footer's `.stats`.

Challenge Seed

```
<style>
  body {
    font-family: Arial, sans-serif;
  }
  header {
  }
  header .profile-thumbnail {
    width: 50px;
    height: 50px;
    border-radius: 4px;
  }
  header .profile-name {
    margin-left: 10px;
  }
  header .follow-btn {
    margin: 0 0 0 auto;
  }
  header .follow-btn button {
    border: 0;
    border-radius: 3px;
    padding: 5px;
  }
  header h3, header h4 {
    margin: 0;
  }
  #inner p {
    margin-bottom: 10px;
    font-size: 20px;
  }
  #inner hr {
    margin: 20px 0;
    border-style: solid;
    opacity: 0.1;
  }
  footer {
  }
  footer .stats {
    font-size: 15px;
  }
</style>
```

```

        }
        footer .stats strong {
            font-size: 18px;
        }
        footer .stats .likes {
            margin-left: 10px;
        }
        footer .cta {
            margin-left: auto;
        }
        footer .cta button {
            border: 0;
            background: transparent;
        }
    
```

</style>

```

<header>
    
    <div class="profile-name">
        <h3>Quincy Larson</h3>
        <h4>@ossia</h4>
    </div>
    <div class="follow-btn">
        <button>Follow</button>
    </div>
</header>
<div id="inner">
    <p>I meet so many people who are in search of that one trick that will help them work smart. Even if
    you work smart, you still have to work hard.</p>
    <span class="date">1:32 PM - 12 Jan 2018</span>
    <hr>
</div>
<footer>
    <div class="stats">
        <div class="Retweets">
            <strong>107</strong> Retweets
        </div>
        <div class="likes">
            <strong>431</strong> Likes
        </div>
    </div>
    <div class="cta">
        <button class="share-btn">Share</button>
        <button class="retweet-btn">Retweet</button>
        <button class="like-btn">Like</button>
    </div>
</footer>

```

Solution

```

<style>
    body {
        font-family: Arial, sans-serif;
    }
    header {
        display: flex;
    }
    header .profile-thumbnail {
        width: 50px;
        height: 50px;
        border-radius: 4px;
    }
    header .profile-name {
        display: flex;
        margin-left: 10px;
    }
    header .follow-btn {
        display: flex;
        margin: 0 0 0 auto;
    }
    header .follow-btn button {
        border: 0;
        border-radius: 3px;
    }

```

```
padding: 5px;
}
header h3, header h4 {
  display: flex;
  margin: 0;
}
#inner p {
  margin-bottom: 10px;
  font-size: 15px;
}
#inner hr {
  margin: 20px 0;
  border-style: solid;
  opacity: 0.1;
}
footer {
  display: flex;
}
footer .stats {
  display: flex;
  font-size: 15px;
}
footer .stats strong {
  font-size: 18px;
}
footer .stats .likes {
  margin-left: 10px;
}
footer .cta {
  margin-left: auto;
}
footer .cta button {
  border: 0;
  background: transparent;
}
</style>
<header>
  
  <div class="profile-name">
    <h3>Quincy Larson</h3>
    <h4>@ossia</h4>
  </div>
  <div class="follow-btn">
    <button>Follow</button>
  </div>
</header>
<div id="inner">
  <p>I meet so many people who are in search of that one trick that will help them work smart. Even if you work smart, you still have to work hard.</p>
  <span class="date">1:32 PM - 12 Jan 2018</span>
  <hr>
</div>
<footer>
  <div class="stats">
    <div class="Retweets">
      <strong>107</strong> Retweets
    </div>
    <div class="likes">
      <strong>431</strong> Likes
    </div>
  </div>
  <div class="cta">
    <button class="share-btn">Share</button>
    <button class="retweet-btn">Retweet</button>
    <button class="like-btn">Like</button>
  </div>
</footer>
```

3. Use the flex-direction Property to Make a Row

Description

Adding `display: flex` to an element turns it into a flex container. This makes it possible to align any children of that element into rows or columns. You do this by adding the `flex-direction` property to the parent item and setting it to row or column. Creating a row will align the children horizontally, and creating a column will align the children vertically. Other options for `flex-direction` are `row-reverse` and `column-reverse`. **Note**
The default value for the `flex-direction` property is `row`.

Instructions

Add the CSS property `flex-direction` to the `#box-container` element, and give it a value of `row-reverse`.

Challenge Seed

```
<style>
  #box-container {
    display: flex;
    height: 500px;
  }
  #box-1 {
    background-color: dodgerblue;
    width: 50%;
    height: 50%;
  }
  #box-2 {
    background-color: orangered;
    width: 50%;
    height: 50%;
  }
</style>

<div id="box-container">
  <div id="box-1"></div>
  <div id="box-2"></div>
</div>
```

Solution

```
// solution required
```

4. Apply the `flex-direction` Property to Create Rows in the Tweet Embed

Description

The `header` and `footer` in the tweet embed example have child items that could be arranged as rows using the `flex-direction` property. This tells CSS to align the children horizontally.

Instructions

Add the CSS property `flex-direction` to both the `header` and `footer` and set the value to `row`.

Challenge Seed

```
<style>
  body {
    font-family: Arial, sans-serif;
  }
```

```
}

header {
  display: flex;
}

header .profile-thumbnail {
  width: 50px;
  height: 50px;
  border-radius: 4px;
}

header .profile-name {
  display: flex;
  margin-left: 10px;
}

header .follow-btn {
  display: flex;
  margin: 0 0 0 auto;
}

header .follow-btn button {
  border: 0;
  border-radius: 3px;
  padding: 5px;
}

header h3, header h4 {
  display: flex;
  margin: 0;
}

#inner p {
  margin-bottom: 10px;
  font-size: 15px;
}

#inner hr {
  margin: 20px 0;
  border-style: solid;
  opacity: 0.1;
}

footer {
  display: flex;
}

footer .stats {
  display: flex;
  font-size: 15px;
}

footer .stats strong {
  font-size: 18px;
}

footer .stats .likes {
  margin-left: 10px;
}

footer .cta {
  margin-left: auto;
}

footer .cta button {
  border: 0;
  background: transparent;
}

</style>
<header>
  
  <div class="profile-name">
    <h3>Quincy Larson</h3>
    <h4>@ossia</h4>
  </div>
  <div class="follow-btn">
    <button>Follow</button>
  </div>
</header>
<div id="inner">
  <p>I meet so many people who are in search of that one trick that will help them work smart. Even if you work smart, you still have to work hard.</p>
  <span class="date">1:32 PM - 12 Jan 2018</span>
  <hr>
</div>
<footer>
```

```
<div class="stats">
  <div class="Retweets">
    <strong>107</strong> Retweets
  </div>
  <div class="likes">
    <strong>431</strong> Likes
  </div>
  <div class="cta">
    <button class="share-btn">Share</button>
    <button class="retweet-btn">Retweet</button>
    <button class="like-btn">Like</button>
  </div>
</div>
```

Solution

```
var code = "header {flex-direction: row;} footer {flex-direction: row;}"
```

5. Use the flex-direction Property to Make a Column

Description

The last two challenges used the `flex-direction` property set to `row`. This property can also create a column by vertically stacking the children of a flex container.

Instructions

Add the CSS property `flex-direction` to the `#box-container` element, and give it a value of `column`.

Challenge Seed

```
<style>
  #box-container {
    display: flex;
    height: 500px;
  }
  #box-1 {
    background-color: dodgerblue;
    width: 50%;
    height: 50%;
  }
  #box-2 {
    background-color: orangered;
    width: 50%;
    height: 50%;
  }
</style>

<div id="box-container">
  <div id="box-1"></div>
  <div id="box-2"></div>
</div>
```

Solution

```
// solution required
```

6. Apply the flex-direction Property to Create a Column in the Tweet Embed

Description

The tweet embed header and footer used the `flex-direction` property earlier with a `row` value. Similarly, the items inside the `.profile-name` element would work well stacked as a column.

Instructions

Add the CSS property `flex-direction` to the header's `.profile-name` element and set the value to `column`.

Challenge Seed

```
<style>
  body {
    font-family: Arial, sans-serif;
  }
  header, footer {
    display: flex;
    flex-direction: row;
  }
  header .profile-thumbnail {
    width: 50px;
    height: 50px;
    border-radius: 4px;
  }
  header .profile-name {
    display: flex;
    margin-left: 10px;
  }
  header .follow-btn {
    display: flex;
    margin: 0 0 0 auto;
  }
  header .follow-btn button {
    border: 0;
    border-radius: 3px;
    padding: 5px;
  }
  header h3, header h4 {
    display: flex;
    margin: 0;
  }
  #inner p {
    margin-bottom: 10px;
    font-size: 20px;
  }
  #inner hr {
    margin: 20px 0;
    border-style: solid;
    opacity: 0.1;
  }
  footer .stats {
    display: flex;
    font-size: 15px;
  }
  footer .stats strong {
    font-size: 18px;
  }
  footer .stats .likes {
    margin-left: 10px;
  }
  footer .cta {
    margin-left: auto;
  }
  footer .cta button {
    border: 0;
```

```

        background: transparent;
    }
</style>
<header>
    
    <div class="profile-name">
        <h3>Quincy Larson</h3>
        <h4>@ossia</h4>
    </div>
    <div class="follow-btn">
        <button>Follow</button>
    </div>
</header>
<div id="inner">
    <p>I meet so many people who are in search of that one trick that will help them work smart. Even if
you work smart, you still have to work hard.</p>
    <span class="date">1:32 PM - 12 Jan 2018</span>
    <hr>
</div>
<footer>
    <div class="stats">
        <div class="Retweets">
            <strong>107</strong> Retweets
        </div>
        <div class="likes">
            <strong>431</strong> Likes
        </div>
    </div>
    <div class="cta">
        <button class="share-btn">Share</button>
        <button class="retweet-btn">Retweet</button>
        <button class="like-btn">Like</button>
    </div>
</footer>

```

Solution

```
// solution required
```

7. Align Elements Using the justify-content Property

Description

Sometimes the flex items within a flex container do not fill all the space in the container. It is common to want to tell CSS how to align and space out the flex items a certain way. Fortunately, the `justify-content` property has several options to do this. But first, there is some important terminology to understand before reviewing those options. [Here is a useful image showing a row to illustrate the concepts below](#). Recall that setting a flex container as a row places the flex items side-by-side from left-to-right. A flex container set as a column places the flex items in a vertical stack from top-to-bottom. For each, the direction the flex items are arranged is called the **main axis**. For a row, this is a horizontal line that cuts through each item. And for a column, the main axis is a vertical line through the items. There are several options for how to space the flex items along the line that is the main axis. One of the most commonly used is `justify-content: center;`, which aligns all the flex items to the center inside the flex container. Others options include:

- `flex-start` : aligns items to the start of the flex container. For a row, this pushes the items to the left of the container. For a column, this pushes the items to the top of the container.
- `flex-end` : aligns items to the end of the flex container. For a row, this pushes the items to the right of the container. For a column, this pushes the items to the bottom of the container.
- `space-between` : aligns items to the center of the main axis, with extra space placed between the items. The first and last items are pushed to the very edge of the flex container. For example, in a row the first item is against the left side of the container, the last item is against the right side of the container, then the other items between them are spaced evenly.

- `space-around` : similar to `space-between` but the first and last items are not locked to the edges of the container, the space is distributed around all the items

Instructions

An example helps show this property in action. Add the CSS property `justify-content` to the `#box-container` element, and give it a value of `center`. **Bonus**

Try the other options for the `justify-content` property in the code editor to see their differences. But note that a value of `center` is the only one that will pass this challenge.

Challenge Seed

```
<style>
#box-container {
  background: gray;
  display: flex;
  height: 500px;
}

#box-1 {
  background-color: dodgerblue;
  width: 25%;
  height: 100%;
}

#box-2 {
  background-color: orangered;
  width: 25%;
  height: 100%;
}
</style>

<div id="box-container">
  <div id="box-1"></div>
  <div id="box-2"></div>
</div>
```

Solution

```
// solution required
```

8. Use the `justify-content` Property in the Tweet Embed

Description

The last challenge showed an example of the `justify-content` property. For the tweet embed, this property can be applied to align the items in the `.profile-name` element.

Instructions

Add the CSS property `justify-content` to the header's `.profile-name` element and set the value to any of the options from the last challenge.

Challenge Seed

```
<style>
body {
  font-family: Arial, sans-serif;
}
header, footer {
```

```
display: flex;
flex-direction: row;
}
header .profile-thumbnail {
  width: 50px;
  height: 50px;
  border-radius: 4px;
}
header .profile-name {
  display: flex;
  flex-direction: column;

  margin-left: 10px;
}
header .follow-btn {
  display: flex;
  margin: 0 0 0 auto;
}
header .follow-btn button {
  border: 0;
  border-radius: 3px;
  padding: 5px;
}
header h3, header h4 {
  display: flex;
  margin: 0;
}
#inner p {
  margin-bottom: 10px;
  font-size: 15px;
}
#inner hr {
  margin: 20px 0;
  border-style: solid;
  opacity: 0.1;
}
footer .stats {
  display: flex;
  font-size: 15px;
}
footer .stats strong {
  font-size: 18px;
}
footer .stats .likes {
  margin-left: 10px;
}
footer .cta {
  margin-left: auto;
}
footer .cta button {
  border: 0;
  background: transparent;
}
</style>
<header>
  
  <div class="profile-name">
    <h3>Quincy Larson</h3>
    <h4>@ossia</h4>
  </div>
  <div class="follow-btn">
    <button>Follow</button>
  </div>
</header>
<div id="inner">
  <p>I meet so many people who are in search of that one trick that will help them work smart. Even if you work smart, you still have to work hard.</p>
  <span class="date">1:32 PM - 12 Jan 2018</span>
  <hr>
</div>
<footer>
  <div class="stats">
    <div class="Retweets">
      <strong>107</strong> Retweets
    </div>
  </div>
</footer>
```

```
<div class="likes">
  <strong>431</strong> Likes
</div>
<div class="cta">
  <button class="share-btn">Share</button>
  <button class="retweet-btn">Retweet</button>
  <button class="like-btn">Like</button>
</div>
</footer>
```

Solution

```
var code = "header .profile-name {display: flex; flex-direction: column; justify-content: center; margin-left: 10px;}"
```

9. Align Elements Using the align-items Property

Description

The `align-items` property is similar to `justify-content`. Recall that the `justify-content` property aligned flex items along the main axis. For rows, the main axis is a horizontal line and for columns it is a vertical line. Flex containers also have a **cross axis** which is the opposite of the main axis. For rows, the cross axis is vertical and for columns, the cross axis is horizontal. CSS offers the `align-items` property to align flex items along the cross axis. For a row, it tells CSS how to push the items in the entire row up or down within the container. And for a column, how to push all the items left or right within the container. The different values available for `align-items` include:

- `flex-start` : aligns items to the start of the flex container. For rows, this aligns items to the top of the container.
For columns, this aligns items to the left of the container.
- `flex-end` : aligns items to the end of the flex container. For rows, this aligns items to the bottom of the container.
For columns, this aligns items to the right of the container.
- `center` : align items to the center. For rows, this vertically aligns items (equal space above and below the items).
For columns, this horizontally aligns them (equal space to the left and right of the items).
- `stretch` : stretch the items to fill the flex container. For example, rows items are stretched to fill the flex container top-to-bottom. This is the default value if no `align-items` value is specified.
- `baseline` : align items to their baselines. Baseline is a text concept, think of it as the line that the letters sit on.

Instructions

An example helps show this property in action. Add the CSS property `align-items` to the `#box-container` element, and give it a value of `center`. **Bonus**

Try the other options for the `align-items` property in the code editor to see their differences. But note that a value of `center` is the only one that will pass this challenge.

Challenge Seed

```
<style>
#box-container {
  background: gray;
  display: flex;
  height: 500px;
}

#box-1 {
  background-color: dodgerblue;
  width: 200px;
  font-size: 24px;
}

#box-2 {
```

```

background-color: orangered;
width: 200px;
font-size: 18px;
}
</style>

<div id="box-container">
<div id="box-1"><p>Hello</p></div>
<div id="box-2"><p>Goodbye</p></div>
</div>

```

Solution

```
// solution required
```

10. Use the align-items Property in the Tweet Embed

Description

The last challenge introduced the `align-items` property and gave an example. This property can be applied to a few tweet embed elements to align the flex items inside them.

Instructions

Add the CSS property `align-items` to the header's `.follow-btn` element. Set the value to `center`.

Challenge Seed

```

<style>
body {
  font-family: Arial, sans-serif;
}
header, footer {
  display: flex;
  flex-direction: row;
}
header .profile-thumbnail {
  width: 50px;
  height: 50px;
  border-radius: 4px;
}
header .profile-name {
  display: flex;
  flex-direction: column;
  justify-content: center;
  margin-left: 10px;
}
header .follow-btn {
  display: flex;

  margin: 0 0 0 auto;
}
header .follow-btn button {
  border: 0;
  border-radius: 3px;
  padding: 5px;
}
header h3, header h4 {
  display: flex;
  margin: 0;
}
#inner p {
  margin-bottom: 10px;
  font-size: 20px;
}

```

```

#inner hr {
  margin: 20px 0;
  border-style: solid;
  opacity: 0.1;
}
footer .stats {
  display: flex;
  font-size: 15px;
}
footer .stats strong {
  font-size: 18px;
}
footer .stats .likes {
  margin-left: 10px;
}
footer .cta {
  margin-left: auto;
}
footer .cta button {
  border: 0;
  background: transparent;
}
</style>
<header>
  
  <div class="profile-name">
    <h3>Quincy Larson</h3>
    <h4>@ossia</h4>
  </div>
  <div class="follow-btn">
    <button>Follow</button>
  </div>
</header>
<div id="inner">
  <p>I meet so many people who are in search of that one trick that will help them work smart. Even if you work smart, you still have to work hard.</p>
  <span class="date">1:32 PM - 12 Jan 2018</span>
  <hr>
</div>
<footer>
  <div class="stats">
    <div class="Retweets">
      <strong>107</strong> Retweets
    </div>
    <div class="likes">
      <strong>431</strong> Likes
    </div>
  </div>
  <div class="cta">
    <button class="share-btn">Share</button>
    <button class="retweet-btn">Retweet</button>
    <button class="like-btn">Like</button>
  </div>
</footer>

```

Solution

```
// solution required
```

11. Use the flex-wrap Property to Wrap a Row or Column

Description

CSS flexbox has a feature to split a flex item into multiple rows (or columns). By default, a flex container will fit all flex items together. For example, a row will all be on one line. However, using the `flex-wrap` property tells CSS to wrap

items. This means extra items move into a new row or column. The break point of where the wrapping happens depends on the size of the items and the size of the container. CSS also has options for the direction of the wrap:

- `nowrap` : this is the default setting, and does not wrap items.
- `wrap` : wraps items from left-to-right if they are in a row, or top-to-bottom if they are in a column.
- `wrap-reverse` : wraps items from right-to-left if they are in a row, or bottom-to-top if they are in a column.

Instructions

The current layout has too many boxes for one row. Add the CSS property `flex-wrap` to the `#box-container` element, and give it a value of `wrap`.

Challenge Seed

```
<style>
#box-container {
  background: gray;
  display: flex;
  height: 100%;

}
#box-1 {
  background-color: dodgerblue;
  width: 25%;
  height: 50%;
}

#box-2 {
  background-color: orangered;
  width: 25%;
  height: 50%;
}
#box-3 {
  background-color: violet;
  width: 25%;
  height: 50%;
}
#box-4 {
  background-color: yellow;
  width: 25%;
  height: 50%;
}
#box-5 {
  background-color: green;
  width: 25%;
  height: 50%;
}
#box-6 {
  background-color: black;
  width: 25%;
  height: 50%;
}
</style>

<div id="box-container">
<div id="box-1"></div>
<div id="box-2"></div>
<div id="box-3"></div>
<div id="box-4"></div>
<div id="box-5"></div>
<div id="box-6"></div>
</div>
```

Solution

```
// solution required
```

12. Use the flex-shrink Property to Shrink Items

Description

So far, all the properties in the challenges apply to the flex container (the parent of the flex items). However, there are several useful properties for the flex items. The first is the `flex-shrink` property. When it's used, it allows an item to shrink if the flex container is too small. Items shrink when the width of the parent container is smaller than the combined widths of all the flex items within it. The `flex-shrink` property takes numbers as values. The higher the number, the more it will shrink compared to the other items in the container. For example, if one item has a `flex-shrink` value of 1 and the other has a `flex-shrink` value of 3, the one with the value of 3 will shrink three times as much as the other.

Instructions

Add the CSS property `flex-shrink` to both `#box-1` and `#box-2`. Give `#box-1` a value of 1 and `#box-2` a value of 2.

Challenge Seed

```
<style>
  #box-container {
    display: flex;
    height: 500px;
  }
  #box-1 {
    background-color: dodgerblue;
    width: 100%;
    height: 200px;
  }
  #box-2 {
    background-color: orangered;
    width: 100%;
    height: 200px;
  }
</style>

<div id="box-container">
  <div id="box-1"></div>
  <div id="box-2"></div>
</div>
```

Solution

```
// solution required
```

13. Use the flex-grow Property to Expand Items

Description

The opposite of `flex-shrink` is the `flex-grow` property. Recall that `flex-shrink` controls the size of the items when the container shrinks. The `flex-grow` property controls the size of items when the parent container expands. Using a similar example from the last challenge, if one item has a `flex-grow` value of 1 and the other has a `flex-grow` value of 3, the one with the value of 3 will grow three times as much as the other.

Instructions

Add the CSS property `flex-grow` to both `#box-1` and `#box-2`. Give `#box-1` a value of 1 and `#box-2` a value of 2.

Challenge Seed

```

<style>
  #box-container {
    display: flex;
    height: 500px;
  }

  #box-1 {
    background-color: dodgerblue;
    height: 200px;
  }

  #box-2 {
    background-color: orangered;
    height: 200px;
  }
</style>

<div id="box-container">
  <div id="box-1"></div>
  <div id="box-2"></div>
</div>

```

Solution

```
// solution required
```

14. Use the flex-basis Property to Set the Initial Size of an Item

Description

The `flex-basis` property specifies the initial size of the item before CSS makes adjustments with `flex-shrink` or `flex-grow`. The units used by the `flex-basis` property are the same as other size properties (`px`, `em`, `%`, etc.). The value `auto` sizes items based on the content.

Instructions

Set the initial size of the boxes using `flex-basis`. Add the CSS property `flex-basis` to both `#box-1` and `#box-2`. Give `#box-1` a value of `10em` and `#box-2` a value of `20em`.

Challenge Seed

```

<style>
  #box-container {
    display: flex;
    height: 500px;
  }

  #box-1 {
    background-color: dodgerblue;
    height: 200px;
  }

  #box-2 {
    background-color: orangered;
    height: 200px;
  }
</style>

```

```

        }
    </style>

<div id="box-container">
    <div id="box-1"></div>
    <div id="box-2"></div>
</div>

```

Solution

```

<style>
#box-container {
    display: flex;
    height: 500px;
}

#box-1 {
    background-color: dodgerblue;
    height: 200px;
    flex-basis: 10em;
}

#box-2 {
    background-color: orangered;
    height: 200px;
    flex-basis: 20em;
}
</style>

<div id="box-container">
    <div id="box-1"></div>
    <div id="box-2"></div>
</div>

```

15. Use the flex Shorthand Property

Description

There is a shortcut available to set several flex properties at once. The `flex-grow`, `flex-shrink`, and `flex-basis` properties can all be set together by using the `flex` property. For example, `flex: 1 0 10px;` will set the item to `flex-grow: 1;`, `flex-shrink: 0;`, and `flex-basis: 10px;`. The default property settings are `flex: 0 1 auto;`.

Instructions

Add the CSS property `flex` to both `#box-1` and `#box-2`. Give `#box-1` the values so its `flex-grow` is 2, its `flex-shrink` is 2, and its `flex-basis` is 150px. Give `#box-2` the values so its `flex-grow` is 1, its `flex-shrink` is 1, and its `flex-basis` is 150px. These values will cause `#box-1` to grow to fill the extra space at twice the rate of `#box-2` when the container is greater than 300px and shrink at twice the rate of `#box-2` when the container is less than 300px. 300px is the combined size of the `flex-basis` values of the two boxes.

Challenge Seed

```

<style>
#box-container {
    display: flex;
    height: 500px;
}

#box-1 {
    background-color: dodgerblue;
    height: 200px;
}

#box-2 {

```

```
background-color: orangered;

height: 200px;
}

</style>

<div id="box-container">
  <div id="box-1"></div>
  <div id="box-2"></div>
</div>
```

Solution

```
// solution required
```

16. Use the order Property to Rearrange Items

Description

The `order` property is used to tell CSS the order of how flex items appear in the flex container. By default, items will appear in the same order they come in the source HTML. The property takes numbers as values, and negative numbers can be used.

Instructions

Add the CSS property `order` to both `#box-1` and `#box-2`. Give `#box-1` a value of 2 and give `#box-2` a value of 1.

Challenge Seed

```
<style>
  #box-container {
    display: flex;
    height: 500px;
  }
  #box-1 {
    background-color: dodgerblue;

    height: 200px;
    width: 200px;
  }

  #box-2 {
    background-color: orangered;

    height: 200px;
    width: 200px;
  }
</style>

<div id="box-container">
  <div id="box-1"></div>
  <div id="box-2"></div>
</div>
```

Solution

```
// solution required
```

17. Use the align-self Property

Description

The final property for flex items is `align-self`. This property allows you to adjust each item's alignment individually, instead of setting them all at once. This is useful since other common adjustment techniques using the CSS properties `float`, `clear`, and `vertical-align` do not work on flex items. `align-self` accepts the same values as `align-items` and will override any value set by the `align-items` property.

Instructions

Add the CSS property `align-self` to both `#box-1` and `#box-2`. Give `#box-1` a value of `center` and give `#box-2` a value of `flex-end`.

Challenge Seed

```
<style>
  #box-container {
    display: flex;
    height: 500px;
  }
  #box-1 {
    background-color: dodgerblue;
    height: 200px;
    width: 200px;
  }

  #box-2 {
    background-color: orangered;
    height: 200px;
    width: 200px;
  }
</style>

<div id="box-container">
  <div id="box-1"></div>
  <div id="box-2"></div>
</div>
```

Solution

```
// solution required
```

CSS Grid

1. Create Your First CSS Grid

Description

Turn any HTML element into a grid container by setting its `display` property to `grid`. This gives you the ability to use all the other properties associated with CSS Grid. **Note**

In CSS Grid, the parent element is referred to as the container and its children are called items.

Instructions

Change the display of the div with the `container` class to `grid`.

Challenge Seed

```

<style>
  .d1{background:LightSkyBlue;}
  .d2{background:LightSalmon;}
  .d3{background:PaleTurquoise;}
  .d4{background:LightPink;}
  .d5{background:PaleGreen;}

  .container {
    font-size: 40px;
    width: 100%;
    background: LightGray;
    /* add your code below this line */

    /* add your code above this line */
  }
</style>

<div class="container">
  <div class="d1">1</div>
  <div class="d2">2</div>
  <div class="d3">3</div>
  <div class="d4">4</div>
  <div class="d5">5</div>
</div>

```

Solution

```
var code = ".container {display: grid;}"
```

2. Add Columns with grid-template-columns

Description

Simply creating a grid element doesn't get you very far. You need to define the structure of the grid as well. To add some columns to the grid, use the `grid-template-columns` property on a grid container as demonstrated below:

```
.container {
  display: grid;
  grid-template-columns: 50px 50px;
}
```

This will give your grid two columns that are each 50px wide. The number of parameters given to the `grid-template-columns` property indicates the number of columns in the grid, and the value of each parameter indicates the width of each column.

Instructions

Give the grid container three columns that are each `100px` wide.

Challenge Seed

```

<style>
  .d1{background:LightSkyBlue;}
  .d2{background:LightSalmon;}
  .d3{background:PaleTurquoise;}
  .d4{background:LightPink;}
  .d5{background:PaleGreen;}

  .container {
    font-size: 40px;
    width: 100%;
    background: LightGray;
    /* add your code below this line */

    /* add your code above this line */
  }
</style>

<div class="container">
  <div class="d1">1</div>
  <div class="d2">2</div>
  <div class="d3">3</div>
  <div class="d4">4</div>
  <div class="d5">5</div>
</div>

```

```

width: 100%;
background: LightGray;
display: grid;
/* add your code below this line */

/* add your code above this line */
}
</style>

<div class="container">
<div class="d1">1</div>
<div class="d2">2</div>
<div class="d3">3</div>
<div class="d4">4</div>
<div class="d5">5</div>
</div>

```

Solution

```
var code = ".container {grid-template-columns: 100px 100px 100px;}"
```

3. Add Rows with grid-template-rows

Description

The grid you created in the last challenge will set the number of rows automatically. To adjust the rows manually, use the `grid-template-rows` property in the same way you used `grid-template-columns` in previous challenge.

Instructions

Add two rows to the grid that are `50px` tall each.

Challenge Seed

```

<style>
.d1{background:LightSkyBlue;}
.d2{background:LightSalmon;}
.d3{background:PaleTurquoise;}
.d4{background:LightPink;}
.d5{background:PaleGreen;}

.container {
  font-size: 40px;
  width: 100%;
  background: LightGray;
  display: grid;
  grid-template-columns: 100px 100px 100px;
  /* add your code below this line */

  /* add your code above this line */
}
</style>

<div class="container">
<div class="d1">1</div>
<div class="d2">2</div>
<div class="d3">3</div>
<div class="d4">4</div>
<div class="d5">5</div>
</div>

```

Solution

```
var code = ".container {grid-template-rows: 50px 50px;}"
```

4. Use CSS Grid units to Change the Size of Columns and Rows

Description

You can use absolute and relative units like `px` and `em` in CSS Grid to define the size of rows and columns. You can use these as well: `fr` : sets the column or row to a fraction of the available space, `auto` : sets the column or row to the width or height of its content automatically, `%` : adjusts the column or row to the percent width of its container. Here's the code that generates the output in the preview:

```
grid-template-columns: auto 50px 10% 2fr 1fr;
```

This snippet creates five columns. The first column is as wide as its content, the second column is 50px, the third column is 10% of its container, and for the last two columns; the remaining space is divided into three sections, two are allocated for the fourth column, and one for the fifth.

Instructions

Make a grid with three columns whose widths are as follows: 1fr, 100px, and 2fr.

Challenge Seed

```
<style>
.d1{background:LightSkyBlue;}
.d2{background:LightSalmon;}
.d3{background:PaleTurquoise;}
.d4{background:LightPink;}
.d5{background:PaleGreen;}

.container {
  font-size: 40px;
  width: 100%;
  background: LightGray;
  display: grid;
  /* modify the code below this line */

  grid-template-columns: auto 50px 10% 2fr 1fr;

  /* modify the code above this line */
  grid-template-rows: 50px 50px;
}
</style>

<div class="container">
  <div class="d1">1</div>
  <div class="d2">2</div>
  <div class="d3">3</div>
  <div class="d4">4</div>
  <div class="d5">5</div>
</div>
```

Solution

```
var code = ".container {grid-template-columns: 1fr 100px 2fr;}"
```

5. Create a Column Gap Using grid-column-gap

Description

So far in the grids you have created, the columns have all been tight up against each other. Sometimes you want a gap in between the columns. To add a gap between the columns, use the `grid-column-gap` property like this:

```
grid-column-gap: 10px;
```

This creates 10px of empty space between all of our columns.

Instructions

Give the columns in the grid a `20px` gap.

Challenge Seed

```
<style>
  .d1{background:LightSkyBlue;}
  .d2{background:LightSalmon;}
  .d3{background:PaleTurquoise;}
  .d4{background:LightPink;}
  .d5{background:PaleGreen;}

  .container {
    font-size: 40px;
    min-height: 300px;
    width: 100%;
    background: LightGray;
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;
    grid-template-rows: 1fr 1fr 1fr;
    /* add your code below this line */

    /* add your code above this line */
  }
</style>

<div class="container">
  <div class="d1">1</div>
  <div class="d2">2</div>
  <div class="d3">3</div>
  <div class="d4">4</div>
  <div class="d5">5</div>
</div>
```

Solution

```
var code = ".container {grid-column-gap: 20px;}"
```

6. Create a Row Gap using grid-row-gap

Description

You can add a gap in between the rows of a grid using `grid-row-gap` in the same way that you added a gap in between columns in the previous challenge.

Instructions

Create a gap for the rows that is `5px` tall.

Challenge Seed

```

<style>
  .d1{background:LightSkyBlue;}
  .d2{background:LightSalmon;}
  .d3{background:PaleTurquoise;}
  .d4{background:LightPink;}
  .d5{background:PaleGreen;}

  .container {
    font-size: 40px;
    min-height: 300px;
    width: 100%;
    background: LightGray;
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;
    grid-template-rows: 1fr 1fr 1fr;
    /* add your code below this line */

    /* add your code above this line */
  }
</style>

<div class="container">
  <div class="d1">1</div>
  <div class="d2">2</div>
  <div class="d3">3</div>
  <div class="d4">4</div>
  <div class="d5">5</div>
</div>

```

Solution

```
var code = ".container {grid-row-gap: 5px;}"
```

7. Add Gaps Faster with grid-gap

Description

`grid-gap` is a shorthand property for `grid-row-gap` and `grid-column-gap` from the previous two challenges that's more convenient to use. If `grid-gap` has one value, it will create a gap between all rows and columns. However, if there are two values, it will use the first one to set the gap between the rows and the second value for the columns.

Instructions

Use `grid-gap` to introduce a `10px` gap between the rows and `20px` gap between the columns.

Challenge Seed

```

<style>
  .d1{background:LightSkyBlue;}
  .d2{background:LightSalmon;}
  .d3{background:PaleTurquoise;}
  .d4{background:LightPink;}
  .d5{background:PaleGreen;}

  .container {
    font-size: 40px;
    min-height: 300px;
    width: 100%;
    background: LightGray;
    display: grid;

```

```
grid-template-columns: 1fr 1fr 1fr;  
grid-template-rows: 1fr 1fr 1fr;  
/* add your code below this line */  
  
/* add your code above this line */  
}  
</style>  
<div class="container">  
  <div class="d1">1</div>  
  <div class="d2">2</div>  
  <div class="d3">3</div>  
  <div class="d4">4</div>  
  <div class="d5">5</div>  
</div>
```

Solution

```
var code = ".container {grid-gap: 10px 20px;}"
```

8. Use grid-column to Control Spacing

Description

Up to this point, all the properties that have been discussed are for grid containers. The `grid-column` property is the first one for use on the grid items themselves. The hypothetical horizontal and vertical lines that create the grid are referred to as lines. These lines are numbered starting with 1 at the top left corner of the grid and move right for columns and down for rows, counting upward. This is what the lines look like for a 3x3 grid:

column lines

```
1  
2  
3  
4  
row lines
```

```
1  
2  
3  
4
```

To control the amount of columns an item will consume, you can use the `grid-column` property in conjunction with the line numbers you want the item to start and stop at. Here's an example:

```
grid-column: 1 / 3;
```

This will make the item start at the first vertical line of the grid on the left and span to the 3rd line of the grid, consuming two columns.

Instructions

Make the item with the class `item5` consume the last two columns of the grid.

Challenge Seed

```

<style>
.item1{background:LightSkyBlue;}
.item2{background:LightSalmon;}
.item3{background:PaleTurquoise;}
.item4{background:LightPink;}

.item5 {
  background: PaleGreen;
  /* add your code below this line */

  /* add your code above this line */
}

.container {
  font-size: 40px;
  min-height: 300px;
  width: 100%;
  background: LightGray;
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
}
</style>

<div class="container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
</div>

```

Solution

```
var code = ".item5 {grid-column: 2 / 4;}"
```

9. Use grid-row to Control Spacing

Description

Of course, you can make items consume multiple rows just like you can with columns. You define the horizontal lines you want an item to start and stop at using the `grid-row` property on a grid item.

Instructions

Make the element with the `item5` class consume the last two rows.

Challenge Seed

```

<style>
.item1{background:LightSkyBlue;}
.item2{background:LightSalmon;}
.item3{background:PaleTurquoise;}
.item4{background:LightPink;}

.item5 {
  background: PaleGreen;
  grid-column: 2 / 4;
  /* add your code below this line */

  /* add your code above this line */
}

```

```

}

.container {
  font-size: 40px;
  min-height: 300px;
  width: 100%;
  background: LightGray;
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
}
</style>

<div class="container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
</div>

```

Solution

```
var code = ".item5 {grid-row: 2 / 4;}"
```

10. Align an Item Horizontally using justify-self

Description

In CSS Grid, the content of each item is located in a box which is referred to as a cell. You can align the content's position within its cell horizontally using the `justify-self` property on a grid item. By default, this property has a value of `stretch`, which will make the content fill the whole width of the cell. This CSS Grid property accepts other values as well: `start` : aligns the content at the left of the cell, `center` : aligns the content in the center of the cell, `end` : aligns the content at the right of the cell.

Instructions

Use the `justify-self` property to center the item with the class `item2`.

Challenge Seed

```

<style>
  .item1{background: LightSkyBlue;}

  .item2 {
    background: LightSalmon;
    /* add your code below this line */

    /* add your code above this line */
  }

  .item3{background:PaleTurquoise;}
  .item4{background:LightPink;}
  .item5{background:PaleGreen;}

  .container {
    font-size: 40px;
    min-height: 300px;
    width: 100%;
    background: LightGray;
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;
    grid-template-rows: 1fr 1fr 1fr;
    grid-gap: 10px;
  }

```

```

        grid-template-rows: 1fr 1fr 1fr;
        grid-gap: 10px;
    }
</style>

<div class="container">
    <div class="item1">1</div>
    <div class="item2">2</div>
    <div class="item3">3</div>
    <div class="item4">4</div>
    <div class="item5">5</div>
</div>

```

Solution

```
var code = ".item2 {justify-self: center;}"
```

11. Align an Item Vertically using align-self

Description

Just as you can align an item horizontally, there's a way to align an item vertically as well. To do this, you use the `align-self` property on an item. This property accepts all of the same values as `justify-self` from the last challenge.

Instructions

Align the item with the class `item3` vertically at the `end`.

Challenge Seed

```

<style>
    .item1{background:LightSkyBlue;}
    .item2{background:LightSalmon;}

    .item3 {
        background: PaleTurquoise;
        /* add your code below this line */

        /* add your code above this line */
    }

    .item4{background:LightPink;}
    .item5{background:PaleGreen;}

    .container {
        font-size: 40px;
        min-height: 300px;
        width: 100%;
        background: LightGray;
        display: grid;
        grid-template-columns: 1fr 1fr 1fr;
        grid-template-rows: 1fr 1fr 1fr;
        grid-gap: 10px;
    }
</style>

<div class="container">
    <div class="item1">1</div>
    <div class="item2">2</div>
    <div class="item3">3</div>
    <div class="item4">4</div>
    <div class="item5">5</div>
</div>

```

Solution

```
var code = ".item3 {align-self: end;}"
```

12. Align All Items Horizontally using justify-items

Description

Sometimes you want all the items in your CSS Grid to share the same alignment. You can use the previously learned properties and align them individually, or you can align them all at once horizontally by using `justify-items` on your grid container. This property can accept all the same values you learned about in the previous two challenges, the difference being that it will move **all** the items in our grid to the desired alignment.

Instructions

Use this property to center all our items horizontally.

Challenge Seed

```
<style>
.item1{background:LightSkyBlue;}
.item2{background:LightSalmon;}
.item3{background:PaleTurquoise;}
.item4{background:LightPink;}
.item5{background:PaleGreen;}

.container {
  font-size: 40px;
  min-height: 300px;
  width: 100%;
  background: LightGray;
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
  /* add your code below this line */

  /* add your code above this line */
}
</style>

<div class="container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
</div>
```

Solution

```
var code = ".container {justify-items: center;}"
```

13. Align All Items Vertically using align-items

Description

Using the `align-items` property on a grid container will set the vertical alignment for all the items in our grid.

Instructions

Use it now to move all the items to the end of each cell.

Challenge Seed

```
<style>
.item1{background:LightSkyBlue;}
.item2{background:LightSalmon;}
.item3{background:PaleTurquoise;}
.item4{background:LightPink;}
.item5{background:PaleGreen; }

.container {
  font-size: 40px;
  min-height: 300px;
  width: 100%;
  background: LightGray;
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
  /* add your code below this line */

  /* add your code above this line */
}
</style>

<div class="container">
<div class="item1">1</div>
<div class="item2">2</div>
<div class="item3">3</div>
<div class="item4">4</div>
<div class="item5">5</div>
</div>
```

Solution

```
var code = ".container {align-items: end;}"
```

14. Divide the Grid Into an Area Template

Description

You can group cells of your grid together into an area and give the area a custom name. Do this by using `grid-template-areas` on the container like this:

```
grid-template-areas:
  "header header header"
  "advert content content"
  "footer footer footer";
```

The code above merges the top three cells together into an area named `header`, the bottom three cells into a `footer` area, and it makes two areas in the middle row; `advert` and `content`. **Note**

Every word in the code represents a cell and every pair of quotation marks represent a row. In addition to custom labels, you can use a period (.) to designate an empty cell in the grid.

Instructions

Place the area template so that the cell labeled `advert` becomes an empty cell.

Challenge Seed

```
<style>
.item1{background:LightSkyBlue;}
.item2{background:LightSalmon;}
.item3{background:PaleTurquoise;}
.item4{background:LightPink;}
.item5{background:PaleGreen;}

.container {
  font-size: 40px;
  min-height: 300px;
  width: 100%;
  background: LightGray;
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
  /* change code below this line */

  grid-template-areas:
    "header header header"
    "advert content content"
    "footer footer footer";
  /* change code above this line */
}
</style>

<div class="container">
<div class="item1">1</div>
<div class="item2">2</div>
<div class="item3">3</div>
<div class="item4">4</div>
<div class="item5">5</div>
</div>
```

Solution

```
<style>
.item1{background:LightSkyBlue;}
.item2{background:LightSalmon;}
.item3{background:PaleTurquoise;}
.item4{background:LightPink;}
.item5{background:PaleGreen;}

.container {
  font-size: 40px;
  min-height: 300px;
  width: 100%;
  background: LightGray;
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;

  grid-template-areas:
    "header header header"
    ". content content"
    "footer footer footer";
}
</style>

<div class="container">
<div class="item1">1</div>
<div class="item2">2</div>
<div class="item3">3</div>
<div class="item4">4</div>
```

```
<div class="item5">5</div>
</div>
```

15. Place Items in Grid Areas Using the grid-area Property

Description

After creating an areas template for your grid container, as shown in the previous challenge, you can place an item in your custom area by referencing the name you gave it. To do this, you use the `grid-area` property on an item like this:

```
.item1 { grid-area: header; }
```

This lets the grid know that you want the `item1` class to go in the area named `header`. In this case, the item will use the entire top row because that whole row is named as the header area.

Instructions

Place an element with the `item5` class in the `footer` area using the `grid-area` property.

Challenge Seed

```
<style>
.item1{background:LightSkyBlue;}
.item2{background:LightSalmon;}
.item3{background:PaleTurquoise;}
.item4{background:LightPink;}

.item5 {
  background: PaleGreen;
  /* add your code below this line */

  /* add your code above this line */
}

.container {
  font-size: 40px;
  min-height: 300px;
  width: 100%;
  background: LightGray;
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
  grid-template-areas:
    "header header header"
    "advert content content"
    "footer footer footer";
}
</style>

<div class="container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
</div>
```

Solution

```
var code = ".item5 {grid-area: footer;}"
```

16. Use grid-area Without Creating an Areas Template

Description

The `grid-area` property you learned in the last challenge can be used in another way. If your grid doesn't have an areas template to reference, you can create an area on the fly for an item to be placed like this:

```
| item1 { grid-area: 1/1/2/4; }
```

This is using the line numbers you learned about earlier to define where the area for this item will be. The numbers in the example above represent these values:

```
| grid-area: horizontal line to start at / vertical line to start at / horizontal line to end at / vertical line to end at;
```

So the item in the example will consume the rows between lines 1 and 2, and the columns between lines 1 and 4.

Instructions

Using the `grid-area` property, place the element with `item5` class between the third and fourth horizontal lines and between the first and fourth vertical lines.

Challenge Seed

```
<style>
  .item1{background:LightSkyBlue;}
  .item2{background:LightSalmon;}
  .item3{background:PaleTurquoise;}
  .item4{background:LightPink;}

  .item5 {
    background: PaleGreen;
    /* add your code below this line */

    /* add your code above this line */
  }

  .container {
    font-size: 40px;
    min-height: 300px;
    width: 100%;
    background: LightGray;
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;
    grid-template-rows: 1fr 1fr 1fr;
    grid-gap: 10px;
  }
</style>

<div class="container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
</div>
```

Solution

```
var code = ".item5 {grid-area: 3/1/4/4;}"
```

17. Reduce Repetition Using the repeat Function

Description

When you used `grid-template-columns` and `grid-template-rows` to define the structure of a grid, you entered a value for each row or column you created. Lets say you want a grid with 100 rows of the same height. It isn't very practical to insert 100 values individually. Fortunately, there's a better way - by using the `repeat` function to specify the number of times you want your column or row to be repeated, followed by a comma and the value you want to repeat. Here's an example that would create the 100 row grid, each row at 50px tall.

```
grid-template-rows: repeat(100, 50px);
```

You can also repeat multiple values with the `repeat` function and insert the function amongst other values when defining a grid structure. Here's what that looks like:

```
grid-template-columns: repeat(2, 1fr 50px) 20px;
```

This translates to:

```
grid-template-columns: 1fr 50px 1fr 50px 20px;
```

Note

`1fr 50px` is repeated twice followed by `20px`.

Instructions

Use `repeat` to remove repetition from the `grid-template-columns` property.

Challenge Seed

```
<style>
.item1{background:LightSkyBlue;}
.item2{background:LightSalmon;}
.item3{background:PaleTurquoise;}
.item4{background:LightPink;}
.item5{background:PaleGreen;}

.container {
  font-size: 40px;
  min-height: 300px;
  width: 100%;
  background: LightGray;
  display: grid;
  /* change the code below this line */

  grid-template-columns: 1fr 1fr 1fr;

  /* change the code above this line */
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
}
</style>

<div class="container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
</div>
```

Solution

```
var code = ".container {grid-template-columns: repeat(3, 1fr);}"
```

18. Limit Item Size Using the `minmax` Function

Description

There's another built-in function to use with `grid-template-columns` and `grid-template-rows` called `minmax`. It's used to limit the size of items when the grid container changes size. To do this you need to specify the acceptable size range for your item. Here is an example:

```
grid-template-columns: 100px minmax(50px, 200px);
```

In the code above, `grid-template-columns` is set to create two columns; the first is 100px wide, and the second has the minimum width of 50px and the maximum width of 200px.

Instructions

Using the `minmax` function, replace the `1fr` in the `repeat` function with a column size that has the minimum width of `90px` and the maximum width of `1fr`, and resize the preview panel to see the effect.

Challenge Seed

```
<style>
.item1{background:LightSkyBlue;}
.item2{background:LightSalmon;}
.item3{background:PaleTurquoise;}
.item4{background:LightPink;}
.item5{background:PaleGreen;}

.container {
  font-size: 40px;
  min-height: 300px;
  width: 100%;
  background: LightGray;
  display: grid;
  /* change the code below this line */

  grid-template-columns: repeat(3, 1fr);

  /* change the code above this line */
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
}
</style>

<div class="container">
<div class="item1">1</div>
<div class="item2">2</div>
<div class="item3">3</div>
<div class="item4">4</div>
<div class="item5">5</div>
</div>
```

Solution

```
var code = ".container {grid-template-columns: repeat(3, minmax(90px, 1fr));}"
```

19. Create Flexible Layouts Using auto-fill

Description

The `repeat` function comes with an option called `auto-fill`. This allows you to automatically insert as many rows or columns of your desired size as possible depending on the size of the container. You can create flexible layouts when combining `auto-fill` with `minmax`. In the preview, `grid-template-columns` is set to

```
repeat(auto-fill, minmax(60px, 1fr));
```

When the container changes size, this setup keeps inserting 60px columns and stretching them until it can insert another one. **Note**

If your container can't fit all your items on one row, it will move them down to a new one.

Instructions

In the first grid, use `auto-fill` with `repeat` to fill the grid with columns that have a minimum width of `60px` and maximum of `1fr`. Then resize the preview to see `auto-fill` in action.

Challenge Seed

```
<style>
.item1{background:LightSkyBlue;}
.item2{background:LightSalmon;}
.item3{background:PaleTurquoise;}
.item4{background:LightPink;}
.item5{background:PaleGreen;}

.container {
  font-size: 40px;
  min-height: 100px;
  width: 100%;
  background: LightGray;
  display: grid;
  /* change the code below this line */

  grid-template-columns: repeat(3, minmax(60px, 1fr));

  /* change the code above this line */
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
}

.container2 {
  font-size: 40px;
  min-height: 100px;
  width: 100%;
  background: Silver;
  display: grid;
  grid-template-columns: repeat(3, minmax(60px, 1fr));
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
}
</style>
<div class="container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
</div>
<div class="container2">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
</div>
```

Solution

```
var code = ".container {grid-template-columns: repeat(auto-fill, minmax(60px, 1fr));}"
```

20. Create Flexible Layouts Using `auto-fit`

Description

`auto-fit` works almost identically to `auto-fill`. The only difference is that when the container's size exceeds the size of all the items combined, `auto-fill` keeps inserting empty rows or columns and pushes your items to the side, while `auto-fit` collapses those empty rows or columns and stretches your items to fit the size of the container. **Note** If your container can't fit all your items on one row, it will move them down to a new one.

Instructions

In the second grid, use `auto-fit` with `repeat` to fill the grid with columns that have a minimum width of `60px` and maximum of `1fr`. Then resize the preview to see the difference.

Challenge Seed

```
<style>
.item1{background:LightSkyBlue;}
.item2{background:LightSalmon;}
.item3{background:PaleTurquoise;}
.item4{background:LightPink;}
.item5{background:PaleGreen;}

.container {
  font-size: 40px;
  min-height: 100px;
  width: 100%;
  background: LightGray;
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(60px, 1fr));
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
}

.container2 {
  font-size: 40px;
  min-height: 100px;
  width: 100%;
  background: Silver;
  display: grid;
  /* change the code below this line */

  grid-template-columns: repeat(3, minmax(60px, 1fr));

  /* change the code above this line */
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
}
</style>

<div class="container">
<div class="item1">1</div>
<div class="item2">2</div>
<div class="item3">3</div>
<div class="item4">4</div>
<div class="item5">5</div>
</div>
<div class="container2">
<div class="item1">1</div>
<div class="item2">2</div>
<div class="item3">3</div>
<div class="item4">4</div>
<div class="item5">5</div>
</div>
```

Solution

```
var code = ".container {grid-template-columns: repeat( auto-fill, minmax(60px, 1fr));} .container2
{grid-template-columns: repeat(auto-fit, minmax(60px, 1fr));}"
```

21. Use Media Queries to Create Responsive Layouts

Description

CSS Grid can be an easy way to make your site more responsive by using media queries to rearrange grid areas, change dimensions of a grid, and rearrange the placement of items. In the preview, when the viewport width is 300px or more, the number of columns changes from 1 to 2. The advertisement area then occupies the left column completely.

Instructions

When the viewport width is 400px or more, make the header area occupy the top row completely and the footer area occupy the bottom row completely.

Challenge Seed

```
<style>
.item1 {
  background: LightSkyBlue;
  grid-area: header;
}

.item2 {
  background: LightSalmon;
  grid-area: advert;
}

.item3 {
  background: PaleTurquoise;
  grid-area: content;
}

.item4 {
  background: lightpink;
  grid-area: footer;
}

.container {
  font-size: 1.5em;
  min-height: 300px;
  width: 100%;
  background: LightGray;
  display: grid;
  grid-template-columns: 1fr;
  grid-template-rows: 50px auto 1fr auto;
  grid-gap: 10px;
  grid-template-areas:
    "header"
    "advert"
    "content"
    "footer";
}

@media (min-width: 300px){
.container{
  grid-template-columns: auto 1fr;
  grid-template-rows: auto 1fr auto;
  grid-template-areas:
    "advert header"
    "advert content"
    "advert footer";
}
}

@media (min-width: 400px){
.container{
  /* change the code below this line */

  grid-template-areas:
    "advert header"
    "advert content"
    "advert footer";
}
}
```

```

    "advert content"
    "advert footer";

    /* change the code above this line */
}

</style>

<div class="container">
  <div class="item1">header</div>
  <div class="item2">advert</div>
  <div class="item3">content</div>
  <div class="item4">footer</div>
</div>

```

Solution

```

<style>
  .item1 {
    background: LightSkyBlue;
    grid-area: header;
  }

  .item2 {
    background: LightSalmon;
    grid-area: advert;
  }

  .item3 {
    background: PaleTurquoise;
    grid-area: content;
  }

  .item4 {
    background: lightpink;
    grid-area: footer;
  }

  .container {
    font-size: 1.5em;
    min-height: 300px;
    width: 100%;
    background: LightGray;
    display: grid;
    grid-template-columns: 1fr;
    grid-template-rows: 50px auto 1fr auto;
    grid-gap: 10px;
    grid-template-areas:
      "header"
      "advert"
      "content"
      "footer";
  }

  @media (min-width: 300px){
    .container{
      grid-template-columns: auto 1fr;
      grid-template-rows: auto 1fr auto;
      grid-template-areas:
        "advert header"
        "advert content"
        "advert footer";
    }
  }

  @media (min-width: 400px){
    .container{
      /* change the code below this line */

      grid-template-areas:
        "header header"
        "advert content"
        "footer footer";
    }
  }

```

```
/* change the code above this line */  
}  
}  
</style>  
  
<div class="container">  
  <div class="item1">header</div>  
  <div class="item2">advert</div>  
  <div class="item3">content</div>  
  <div class="item4">footer</div>  
</div>
```

22. Create Grids within Grids

Description

Turning an element into a grid only affects the behavior of its direct descendants. So by turning a direct descendant into a grid, you have a grid within a grid. For example, by setting the `display` and `grid-template-columns` properties of the element with the `item3` class, you create a grid within your grid.

Instructions

Turn the element with the `item3` class into a grid with two columns with a width of `auto` and `1fr` using `display` and `grid-template-columns`.

Challenge Seed

```
<style>  
.container {  
  font-size: 1.5em;  
  min-height: 300px;  
  width: 100%;  
  background: LightGray;  
  display: grid;  
  grid-template-columns: auto 1fr;  
  grid-template-rows: auto 1fr auto;  
  grid-gap: 10px;  
  grid-template-areas:  
    "advert header"  
    "advert content"  
    "advert footer";  
}  
.item1 {  
  background: LightSkyBlue;  
  grid-area: header;  
}  
  
.item2 {  
  background: LightSalmon;  
  grid-area: advert;  
}  
  
.item3 {  
  background: PaleTurquoise;  
  grid-area: content;  
  /* enter your code below this line */  
}  
  
  /* enter your code above this line */  
}  
  
.item4 {  
  background: lightpink;  
  grid-area: footer;  
}  
  
.itemOne {
```

```

        background: PaleGreen;
    }

.itemTwo {
    background: BlanchedAlmond;
}

</style>

<div class="container">
    <div class="item1">header</div>
    <div class="item2">advert</div>
    <div class="item3">
        <div class="itemOne">paragraph1</div>
        <div class="itemTwo">paragraph2</div>
    </div>
    <div class="item4">footer</div>
</div>

```

Solution

```
var code = ".item3 {grid-template-columns: auto 1fr; display: grid;}"
```

Responsive Web Design Projects

1. Build a Tribute Page

Description

Objective: Build a [CodePen.io](#) app that is functionally similar to this: <https://codepen.io/freeCodeCamp/full/zNqgVx>. Fulfill the below [user stories](#) and get all of the tests to pass. Give it your own personal style. You can use HTML, JavaScript, and CSS to complete this project. Plain CSS is recommended because that is what the lessons have covered so far and you should get some practice with plain CSS. You can use Bootstrap or SASS if you choose. Additional technologies (just for example jQuery, React, Angular, or Vue) are not recommended for this project, and using them is at your own risk. Other projects will give you a chance to work with different technology stacks like React. We will accept and try to fix all issue reports that use the suggested technology stack for this project. Happy coding!

User Story #1: My tribute page should have an element with a corresponding `id="main"`, which contains all other elements.

User Story #2: I should see an element with a corresponding `id="title"`, which contains a string (i.e. text) that describes the subject of the tribute page (e.g. "Dr. Norman Borlaug").

User Story #3: I should see a `div` element with a corresponding `id="img-div"`.

User Story #4: Within the `img-div` element, I should see an `img` element with a corresponding `id="image"`.

User Story #5: Within the `img-div` element, I should see an element with a corresponding `id="img-caption"` that contains textual content describing the image shown in `img-div`.

User Story #6: I should see an element with a corresponding `id="tribute-info"`, which contains textual content describing the subject of the tribute page.

User Story #7: I should see an `a` element with a corresponding `id="tribute-link"`, which links to an outside site that contains additional information about the subject of the tribute page.

HINT: You must give your element an attribute of `target` and set it to `_blank` in order for your link to open in a new tab (i.e. `target="_blank"`).

User Story #8: The `img` element should responsively resize, relative to the width of its parent element, without exceeding its original size.

User Story #9: The `img` element should be centered within its parent element.

You can build your project by forking [this CodePen pen](#). Or you can use this CDN link to run the tests in any environment you like: <https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js>. Once you're done, submit the URL to your working project with all its tests passing. Remember to use the [Read-Search-Ask](#) method if you get stuck.

Instructions

Challenge Seed

Solution

```
// solution required
```

2. Build a Survey Form

Description

Objective: Build a [CodePen.io](https://codepen.io/freeCodeCamp/full/VPaoNP) app that is functionally similar to this: <https://codepen.io/freeCodeCamp/full/VPaoNP>. Fulfill the below [user stories](#) and get all of the tests to pass. Give it your own personal style. You can use HTML, JavaScript, and CSS to complete this project. Plain CSS is recommended because that is what the lessons have covered so far and you should get some practice with plain CSS. You can use Bootstrap or SASS if you choose. Additional technologies (just for example jQuery, React, Angular, or Vue) are not recommended for this project, and using them is at your own risk. Other projects will give you a chance to work with different technology stacks like React. We will accept and try to fix all issue reports that use the suggested technology stack for this project. Happy coding!

User Story #1: I can see a title with `id="title"` in H1 sized text. **User Story #2:** I can see a short explanation with `id="description"` in P sized text. **User Story #3:** I can see a form with `id="survey-form"`. **User Story #4:** Inside the form element, I am required to enter my name in a field with `id="name"`. **User Story #5:** Inside the form element, I am required to enter an email in a field with `id="email"`. **User Story #6:** If I enter an email that is not formatted correctly, I will see an HTML5 validation error. **User Story #7:** Inside the form, I can enter a number in a field with `id="number"`. **User Story #8:** If I enter non-numbers in the number input, I will see an HTML5 validation error. **User Story #9:** If I enter numbers outside the range of the number input, which are defined by the `min` and `max` attributes, I will see an HTML5 validation error. **User Story #10:** For the name, email, and number input fields inside the form I can see corresponding labels that describe the purpose of each field with the following ids: `id="name-label"`, `id="email-label"`, and `id="number-label"`. **User Story #11:** For the name, email, and number input fields, I can see placeholder text that gives me a description or instructions for each field. **User Story #12:** Inside the form element, I can select an option from a dropdown that has a corresponding `id="dropdown"`. **User Story #13:** Inside the form element, I can select a field from one or more groups of radio buttons. Each group should be grouped using the `name` attribute. **User Story #14:** Inside the form element, I can select several fields from a series of checkboxes, each of which must have a `value` attribute. **User Story #15:** Inside the form element, I am presented with a `textarea` at the end for additional comments. **User Story #16:** Inside the form element, I am presented with a button with `id="submit"` to submit all my inputs. You can build your project by forking [this CodePen pen](#). Or you can use this CDN link to run the tests in any environment you like: <https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js> Once you're done, submit the URL to your working project with all its tests passing. Remember to use the [Read-Search-Ask](#) method if you get stuck.

Instructions

Challenge Seed

Solution

```
// solution required
```

3. Build a Product Landing Page

Description

Objective: Build a [CodePen.io](https://codepen.io/freeCodeCamp/full/RKRbwL) app that is functionally similar to this: <https://codepen.io/freeCodeCamp/full/RKRbwL>. Fulfill the below [user stories](#) and get all of the tests to pass. Give it your own personal style. You can use HTML, JavaScript, and CSS to complete this project. Plain CSS is recommended because that is what the lessons have covered so far and you should get some practice with plain CSS. You can use Bootstrap or SASS if you choose. Additional technologies (just for example jQuery, React, Angular, or Vue) are not recommended for this project, and using them

is at your own risk. Other projects will give you a chance to work with different technology stacks like React. We will accept and try to fix all issue reports that use the suggested technology stack for this project. Happy coding!

User Story #1: My product landing page should have a `header` element with a corresponding `id="header"` . **User Story #2:** I can see an image within the `header` element with a corresponding `id="header-img"` . A company logo would make a good image here. **User Story #3:** Within the `#header` element I can see a `nav` element with a corresponding `id="nav-bar"` . **User Story #4:** I can see at least three clickable elements inside the `nav` element, each with the class `nav-link` . **User Story #5:** When I click a `.nav-link` button in the `nav` element, I am taken to the corresponding section of the landing page. **User Story #6:** I can watch an embedded product video with `id="video"` . **User Story #7:** My landing page has a `form` element with a corresponding `id="form"` . **User Story #8:** Within the form, there is an `input` field with `id="email"` where I can enter an email address. **User Story #9:** The `#email` input field should have placeholder text to let the user know what the field is for. **User Story #10:** The `#email` input field uses HTML5 validation to confirm that the entered text is an email address. **User Story #11:** Within the form, there is a submit `input` with a corresponding `id="submit"` . **User Story #12:** When I click the `#submit` element, the email is submitted to a static page (use this mock URL: <https://www.freecodecamp.com/email-submit>) that confirms the email address was entered and that it posted successfully. **User Story #13:** The navbar should always be at the top of the viewport. **User Story #14:** My product landing page should have at least one media query. **User Story #15:** My product landing page should utilize CSS flexbox at least once. You can build your project by forking [this CodePen pen](#). Or you can use this CDN link to run the tests in any environment you like: <https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js> Once you're done, submit the URL to your working project with all its tests passing. Remember to use the [Read-Search-Ask](#) method if you get stuck.

Instructions

Challenge Seed

Solution

```
// solution required
```

4. Build a Technical Documentation Page

Description

Objective: Build a [CodePen.io](#) app that is functionally similar to this: <https://codepen.io/freeCodeCamp/full/NdrKKL>. Fulfill the below [user stories](#) and get all of the tests to pass. Give it your own personal style. You can use HTML, JavaScript, and CSS to complete this project. Plain CSS is recommended because that is what the lessons have covered so far and you should get some practice with plain CSS. You can use Bootstrap or SASS if you choose. Additional technologies (just for example jQuery, React, Angular, or Vue) are not recommended for this project, and using them is at your own risk. Other projects will give you a chance to work with different technology stacks like React. We will accept and try to fix all issue reports that use the suggested technology stack for this project. Happy coding!

User Story #1: I can see a `main` element with a corresponding `id="main-doc"` , which contains the page's main content (technical documentation). **User Story #2:** Within the `#main-doc` element, I can see several `section` elements, each with a class of `main-section` . There should be a minimum of 5. **User Story #3:** The first element within each `.main-section` should be a `header` element which contains text that describes the topic of that section. **User Story #4:** Each `section` element with the class of `main-section` should also have an id that corresponds with the text of each `header` contained within it. Any spaces should be replaced with underscores (e.g. The `section` that contains the header "Javascript and Java" should have a corresponding `id="Javascript_and_Java"`). **User Story #5:** The `.main-section` elements should contain at least 10 `p` elements total (not each). **User Story #6:** The `.main-section` elements should contain at least 5 `code` elements total (not each). **User Story #7:** The `.main-section` elements should contain at least 5 `li` items total (not each). **User Story #8:** I can see a `nav` element with a corresponding `id="navbar"` . **User Story #9:** The `navbar` element should contain one `header` element which contains text that describes the topic of the technical documentation. **User Story #10:** Additionally, the `navbar` should contain `link (a)` elements with the class of `nav-link` . There should be one for every element with the class `main-section` . **User Story #11:** The `header` element in the `navbar` must come before any `link (a)` elements in the `navbar`. **User Story #12:** Each element with the class of `nav-link` should contain text that corresponds to the `header` text within each `section` (e.g. if you have a "Hello world" section/header, your `navbar` should have an element which contains the text "Hello

world"). **User Story #13:** When I click on a navbar element, the page should navigate to the corresponding section of the `main-doc` element (e.g. If I click on a `nav-link` element that contains the text "Hello world", the page navigates to a `section` element that has that id and contains the corresponding `header`. **User Story #14:** On regular sized devices (laptops, desktops), the element with `id="navbar"` should be shown on the left side of the screen and should always be visible to the user. **User Story #15:** My Technical Documentation page should use at least one media query. You can build your project by forking [this CodePen pen](#). Or you can use this CDN link to run the tests in any environment you like: <https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js> Once you're done, submit the URL to your working project with all its tests passing. Remember to use the [Read-Search-Ask](#) method if you get stuck.

Instructions

Challenge Seed

Solution

```
// solution required
```

5. Build a Personal Portfolio Webpage

Description

Objective: Build a [CodePen.io](#) app that is functionally similar to this: <https://codepen.io/freeCodeCamp/full/zNBOYG>. Fulfill the below [user stories](#) and get all of the tests to pass. Give it your own personal style. You can use HTML, JavaScript, and CSS to complete this project. Plain CSS is recommended because that is what the lessons have covered so far and you should get some practice with plain CSS. You can use Bootstrap or SASS if you choose. Additional technologies (just for example jQuery, React, Angular, or Vue) are not recommended for this project, and using them is at your own risk. Other projects will give you a chance to work with different technology stacks like React. We will accept and try to fix all issue reports that use the suggested technology stack for this project. Happy coding! **User Story #1:** My portfolio should have a welcome section with an id of `welcome-section`. **User Story #2:** The welcome section should have an `h1` element that contains text. **User Story #3:** My portfolio should have a projects section with an id of `projects`. **User Story #4:** The projects section should contain at least one element with a class of `project-tile` to hold a project. **User Story #5:** The projects section should contain at least one link to a project. **User Story #6:** My portfolio should have a navbar with an id of `navbar`. **User Story #7:** The navbar should contain at least one link that I can click on to navigate to different sections of the page. **User Story #8:** My portfolio should have a link with an id of `profile-link`, which opens my GitHub or FCC profile in a new tab. **User Story #9:** My portfolio should have at least one media query. **User Story #10:** The height of the welcome section should be equal to the height of the viewport. **User Story #11:** The navbar should always be at the top of the viewport. You can build your project by forking [this CodePen pen](#). Or you can use this CDN link to run the tests in any environment you like: <https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js> Once you're done, submit the URL to your working project with all its tests passing. Remember to use the [Read-Search-Ask](#) method if you get stuck.

Instructions

Challenge Seed

Solution

```
// solution required
```

Javascript Algorithms And Data Structures Certification

Basic JavaScript

1. Comment Your JavaScript Code

Description

Comments are lines of code that JavaScript will intentionally ignore. Comments are a great way to leave notes to yourself and to other people who will later need to figure out what that code does. There are two ways to write comments in JavaScript: Using `//` will tell JavaScript to ignore the remainder of the text on the current line:

```
// This is an in-line comment.
```

You can make a multi-line comment beginning with `/*` and ending with `*/`:

```
/* This is a  
multi-line comment */
```

Best Practice

As you write code, you should regularly add comments to clarify the function of parts of your code. Good commenting can help communicate the intent of your code—both for others *and* for your future self.

Instructions

Try creating one of each type of comment.

Challenge Seed

Solution

```
// Fake Comment  
/* Another Comment */
```

2. Declare JavaScript Variables

Description

In computer science, data is anything that is meaningful to the computer. JavaScript provides seven different data types which are `undefined`, `null`, `boolean`, `string`, `symbol`, `number`, and `object`. For example, computers distinguish between numbers, such as the number `12`, and `strings`, such as `"12"`, `"dog"`, or `"123 cats"`, which are collections of characters. Computers can perform mathematical operations on a number, but not on a string. Variables allow computers to store and manipulate data in a dynamic fashion. They do this by using a "label" to point to the data rather than using the data itself. Any of the seven data types may be stored in a variable. Variables are similar to the `x` and `y` variables you use in mathematics, which means they're a simple name to represent the data we want to refer to. Computer variables differ from mathematical variables in that they can store different values at different times. We tell JavaScript to create or declare a variable by putting the keyword `var` in front of it, like so:

```
var ourName;
```

creates a variable called `ourName`. In JavaScript we end statements with semicolons. Variable names can be made up of numbers, letters, and `$` or `_`, but may not contain spaces or start with a number.

Instructions

Use the `var` keyword to create a variable called `myName`. **Hint**

Look at the `ourName` example if you get stuck.

Challenge Seed

```
// Example
var ourName;

// Declare myName below this line
```

After Test

```
if(typeof myName !== "undefined"){(function(v){return v;})(myName);}
```

Solution

```
var myName;
```

3. Storing Values with the Assignment Operator

Description

In JavaScript, you can store a value in a variable with the assignment operator. `myVariable = 5;` This assigns the `Number` value `5` to `myVariable`. Assignment always goes from right to left. Everything to the right of the `=` operator is resolved before the value is assigned to the variable to the left of the operator.

```
myVar = 5;
myNum = myVar;
```

This assigns `5` to `myVar` and then resolves `myVar` to `5` again and assigns it to `myNum`.

Instructions

Assign the value `7` to variable `a`. Assign the contents of `a` to variable `b`.

Challenge Seed

```
// Setup
var a;
var b = 2;

// Only change code below this line
```

Before Test

```
if (typeof a != 'undefined') {
  a = undefined;
}
if (typeof b != 'undefined') {
  b = undefined;
}
```

After Test

```
(function(a,b){return "a = " + a + ", b = " + b;})(a,b);
```

Solution

```
var a;
var b = 2;
a = 7;
b = a;
```

4. Initializing Variables with the Assignment Operator

Description

It is common to initialize a variable to an initial value in the same line as it is declared. `var myVar = 0;` Creates a new variable called `myVar` and assigns it an initial value of `0`.

Instructions

Define a variable `a` with `var` and initialize it to a value of `9`.

Challenge Seed

```
// Example
var ourVar = 19;

// Only change code below this line
```

After Test

```
if(typeof a !== 'undefined') { (function(a){return "a = " + a;})(a); } else { (function() {return 'a is undefined';})(); }
```

Solution

```
var a = 9;
```

5. Understanding Uninitialized Variables

Description

When JavaScript variables are declared, they have an initial value of `undefined`. If you do a mathematical operation on an `undefined` variable your result will be `Nan` which means "Not a Number". If you concatenate a string with an `undefined` variable, you will get a literal string of "`undefined`".

Instructions

Initialize the three variables `a`, `b`, and `c` with `5`, `10`, and `"I am a"` respectively so that they will not be `undefined`.

Challenge Seed

```
// Initialize these three variables
var a;
var b;
```

```
var c;

// Do not change code below this line

a = a + 1;
b = b + 5;
c = c + " String!";
```

After Test

```
(function(a,b,c){ return "a = " + a + ", b = " + b + ", c = '" + c + "'"; })(a,b,c);
```

Solution

```
var a = 5;
var b = 10;
var c = "I am a";
a = a + 1;
b = b + 5;
c = c + " String!";
```

6. Understanding Case Sensitivity in Variables

Description

In JavaScript all variables and function names are case sensitive. This means that capitalization matters. `MYVAR` is not the same as `MyVar` nor `myvar`. It is possible to have multiple distinct variables with the same name but different casing. It is strongly recommended that for the sake of clarity, you *do not* use this language feature.

Best Practice

Write variable names in JavaScript in camelCase. In camelCase, multi-word variable names have the first word in lowercase and the first letter of each subsequent word is capitalized. **Examples:**

```
var someVariable;
var anotherVariableName;
var thisVariableNameIsSoLong;
```

Instructions

Modify the existing declarations and assignments so their names use camelCase.
Do not create any new variables.

Challenge Seed

```
// Declarations
var StUdLyCapVaR;
var properCamelCase;
var TitleCaseOver;

// Assignments
STUDLYCAPVAR = 10;
ProperCamelCAse = "A String";
tITLEEcASEoVER = 9000;
```

Solution

```
var studyCapVar;
var properCamelCase;
```

```
var titleCaseOver;

studlyCapVar = 10;
properCamelCase = "A String";
titleCaseOver = 9000;
```

7. Add Two Numbers with JavaScript

Description

`Number` is a data type in JavaScript which represents numeric data. Now let's try to add two numbers using JavaScript. JavaScript uses the `+` symbol as an addition operation when placed between two numbers. **Example:**

```
myVar = 5 + 10; // assigned 15
```

Instructions

Change the `0` so that sum will equal `20`.

Challenge Seed

```
var sum = 10 + 0;
```

After Test

```
(function(z){return 'sum = '+z;})(sum);
```

Solution

```
var sum = 10 + 10;
```

8. Subtract One Number from Another with JavaScript

Description

We can also subtract one number from another. JavaScript uses the `-` symbol for subtraction.

Example

```
myVar = 12 - 6; // assigned 6
```

Instructions

Change the `0` so the difference is `12`.

Challenge Seed

```
var difference = 45 - 0;
```

After Test

```
(function(z){return 'difference = '+z;})(difference);
```

Solution

```
var difference = 45 - 33;
```

9. Multiply Two Numbers with JavaScript

Description

We can also multiply one number by another. JavaScript uses the `*` symbol for multiplication of two numbers.

Example

```
myVar = 13 * 13; // assigned 169
```

Instructions

Change the `0` so that product will equal `80`.

Challenge Seed

```
var product = 8 * 0;
```

After Test

```
(function(z){return 'product = '+z;})(product);
```

Solution

```
var product = 8 * 10;
```

10. Divide One Number by Another with JavaScript

Description

We can also divide one number by another. JavaScript uses the `/` symbol for division.

Example

```
myVar = 16 / 2; // assigned 8
```

Instructions

Change the `0` so that the quotient is equal to `2`.

Challenge Seed

```
var quotient = 66 / 0;
```

After Test

```
(function(z){return 'quotient = '+z;})(quotient);
```

Solution

```
var quotient = 66 / 33;
```

11. Increment a Number with JavaScript

Description

You can easily increment or add one to a variable with the `++` operator. `i++;` is the equivalent of `i = i + 1;` **Note**
The entire line becomes `i++;`, eliminating the need for the equal sign.

Instructions

Change the code to use the `++` operator on `myVar`. **Hint**

Learn more about [Arithmetic operators - Increment \(++\)](#).

Challenge Seed

```
var myVar = 87;

// Only change code below this line
myVar = myVar + 1;
```

After Test

```
(function(z){return 'myVar = ' + z;})(myVar);
```

Solution

```
var myVar = 87;
myVar++;
```

12. Decrement a Number with JavaScript

Description

You can easily decrement or decrease a variable by one with the `--` operator. `i--;` is the equivalent of `i = i - 1;` **Note**

The entire line becomes `i--;`, eliminating the need for the equal sign.

Instructions

Change the code to use the `--` operator on `myVar`.

Challenge Seed

```
var myVar = 11;
```

```
// Only change code below this line
myVar = myVar - 1;
```

After Test

```
(function(z){return 'myVar = ' + z;})(myVar);
```

Solution

```
var myVar = 11;
myVar--;
```

13. Create Decimal Numbers with JavaScript

Description

We can store decimal numbers in variables too. Decimal numbers are sometimes referred to as floating point numbers or floats. **Note**

Not all real numbers can accurately be represented in floating point. This can lead to rounding errors. [Details Here](#).

Instructions

Create a variable `myDecimal` and give it a decimal value with a fractional part (e.g. `5.7`).

Challenge Seed

```
var ourDecimal = 5.7;
// Only change code below this line
```

After Test

```
(function(){if(typeof myDecimal !== "undefined"){return myDecimal;}})();
```

Solution

```
var myDecimal = 9.9;
```

14. Multiply Two Decimals with JavaScript

Description

In JavaScript, you can also perform calculations with decimal numbers, just like whole numbers. Let's multiply two decimals together to get their product.

Instructions

Change the `0.0` so that product will equal `5.0`.

Challenge Seed

```
var product = 2.0 * 0.0;
```

After Test

```
(function(y){return 'product = '+y;})(product);
```

Solution

```
var product = 2.0 * 2.5;
```

15. Divide One Decimal by Another with JavaScript

Description

Now let's divide one decimal by another.

Instructions

Change the 0.0 so that quotient will equal to 2.2 .

Challenge Seed

```
var quotient = 0.0 / 2.0; // Fix this line
```

After Test

```
(function(y){return 'quotient = '+y;})(quotient);
```

Solution

```
var quotient = 4.4 / 2.0;
```

16. Finding a Remainder in JavaScript

Description

The remainder operator % gives the remainder of the division of two numbers. Example

5 % 2 = 1 because

Math.floor(5 / 2) = 2 (Quotient)

2 * 2 = 4

5 - 4 = 1 (Remainder)

Usage

In mathematics, a number can be checked to be even or odd by checking the remainder of the division of the number by 2 .

```
17 % 2 = 1 (17 is Odd)
48 % 2 = 0 (48 is Even)
```

Note

The remainder operator is sometimes incorrectly referred to as the "modulus" operator. It is very similar to modulus, but does not work properly with negative numbers.

Instructions

Set `remainder` equal to the remainder of `11` divided by `3` using the remainder (`%`) operator.

Challenge Seed

```
// Only change code below this line

var remainder;
```

After Test

```
(function(y){return 'remainder = '+y;})(remainder);
```

Solution

```
var remainder = 11 % 3;
```

17. Compound Assignment With Augmented Addition

Description

In programming, it is common to use assignments to modify the contents of a variable. Remember that everything to the right of the equals sign is evaluated first, so we can say: `myVar = myVar + 5;` to add `5` to `myVar`. Since this is such a common pattern, there are operators which do both a mathematical operation and assignment in one step. One such operator is the `+=` operator.

```
var myVar = 1;
myVar += 5;
console.log(myVar); // Returns 6
```

Instructions

Convert the assignments for `a`, `b`, and `c` to use the `+=` operator.

Challenge Seed

```
var a = 3;
var b = 17;
var c = 12;

// Only modify code below this line

a = a + 12;
b = 9 + b;
c = c + 7;
```

After Test

```
(function(a,b,c){ return "a = " + a + ", b = " + b + ", c = " + c; })(a,b,c);
```

Solution

```
var a = 3;
var b = 17;
var c = 12;

a += 12;
b += 9;
c += 7;
```

18. Compound Assignment With Augmented Subtraction

Description

Like the `+=` operator, `-=` subtracts a number from a variable. `myVar = myVar - 5;` will subtract 5 from `myVar`. This can be rewritten as: `myVar -= 5;`

Instructions

Convert the assignments for `a`, `b`, and `c` to use the `-=` operator.

Challenge Seed

```
var a = 11;
var b = 9;
var c = 3;

// Only modify code below this line

a = a - 6;
b = b - 15;
c = c - 1;
```

After Test

```
(function(a,b,c){ return "a = " + a + ", b = " + b + ", c = " + c; })(a,b,c);
```

Solution

```
var a = 11;
var b = 9;
var c = 3;

a -= 6;
b -= 15;
c -= 1;
```

19. Compound Assignment With Augmented Multiplication

Description

The `*=` operator multiplies a variable by a number. `myVar = myVar * 5;` will multiply `myVar` by 5 . This can be rewritten as: `myVar *= 5;`

Instructions

Convert the assignments for `a` , `b` , and `c` to use the `*=` operator.

Challenge Seed

```
var a = 5;
var b = 12;
var c = 4.6;

// Only modify code below this line

a = a * 5;
b = 3 * b;
c = c * 10;
```

After Test

```
(function(a,b,c){ return "a = " + a + ", b = " + b + ", c = " + c; })(a,b,c);
```

Solution

```
var a = 5;
var b = 12;
var c = 4.6;

a *= 5;
b *= 3;
c *= 10;
```

20. Compound Assignment With Augmented Division

Description

The `/=` operator divides a variable by another number. `myVar = myVar / 5;` Will divide `myVar` by 5 . This can be rewritten as: `myVar /= 5;`

Instructions

Convert the assignments for `a` , `b` , and `c` to use the `/=` operator.

Challenge Seed

```
var a = 48;
var b = 108;
var c = 33;

// Only modify code below this line

a = a / 12;
b = b / 4;
c = c / 11;
```

After Test

```
(function(a,b,c){ return "a = " + a + ", b = " + b + ", c = " + c; })(a,b,c);
```

Solution

```
var a = 48;
var b = 108;
var c = 33;

a /= 12;
b /= 4;
c /= 11;
```

21. Declare String Variables

Description

Previously we have used the code `var myName = "your name";` "your name" is called a string literal. It is a string because it is a series of zero or more characters enclosed in single or double quotes.

Instructions

Create two new string variables: `myFirstName` and `myLastName` and assign them the values of your first and last name, respectively.

Challenge Seed

```
// Example
var firstName = "Alan";
var lastName = "Turing";

// Only change code below this line
```

After Test

```
if(typeof myFirstName !== "undefined" && typeof myLastName !== "undefined"){{function(){return
myFirstName + ', ' + myLastName;}}()();}
```

Solution

```
var myFirstName = "Alan";
var myLastName = "Turing";
```

22. Escaping Literal Quotes in Strings

Description

When you are defining a string you must start and end with a single or double quote. What happens when you need a literal quote: " or ' inside of your string? In JavaScript, you can escape a quote from considering it as an end of string quote by placing a backslash (\) in front of the quote. `var sampleStr = "Alan said, \"Peter is learning JavaScript\".";` This signals to JavaScript that the following quote is not the end of the string, but should instead

appear inside the string. So if you were to print this to the console, you would get: Alan said, "Peter is learning JavaScript".

Instructions

Use backslashes to assign a string to the `myStr` variable so that if you were to print it to the console, you would see: I am a "double quoted" string inside "double quotes".

Challenge Seed

```
var myStr = ""; // Change this line
```

After Test

```
(function(){
  if(typeof myStr === 'string') {
    console.log("myStr = \"" + myStr + "\"");
  } else {
    console.log("myStr is undefined");
  }
})();
```

Solution

```
var myStr = "I am a \"double quoted\" string inside \"double quotes\".";
```

23. Quoting Strings with Single Quotes

Description

String values in JavaScript may be written with single or double quotes, as long as you start and end with the same type of quote. Unlike some other programming languages, single and double quotes work the same in JavaScript.

```
doubleQuoteStr = "This is a string";
singleQuoteStr = 'This is also a string';
```

The reason why you might want to use one type of quote over the other is if you want to use both in a string. This might happen if you want to save a conversation in a string and have the conversation in quotes. Another use for it would be saving an `<a>` tag with various attributes in quotes, all within a string.

```
conversation = 'Finn exclaims to Jake, "Algebraic!"';
```

However, this becomes a problem if you need to use the outermost quotes within it. Remember, a string has the same kind of quote at the beginning and end. But if you have that same quote somewhere in the middle, the string will stop early and throw an error.

```
goodStr = 'Jake asks Finn, "Hey, let\'s go on an adventure?"';
badStr = 'Finn responds, "Let's go!"'; // Throws an error
```

In the `goodStr` above, you can use both quotes safely by using the backslash `\` as an escape character. **Note** The backslash `\` should not be confused with the forward slash `/`. They do not do the same thing.

Instructions

Change the provided string to a string with single quotes at the beginning and end and no escape characters. Right now, the `<a>` tag in the string uses double quotes everywhere. You will need to change the outer quotes to single quotes so you can remove the escape characters.

Challenge Seed

```
var myStr = "<a href=\"http://www.example.com\" target=\"_blank\">Link</a>";
```

After Test

```
(function() { return "myStr = " + myStr; })();
```

Solution

```
var myStr = '<a href="http://www.example.com" target="_blank">Link</a>';
```

24. Escape Sequences in Strings

Description

Quotes are not the only characters that can be escaped inside a string. There are two reasons to use escaping characters:

1. To allow you to use characters you may not otherwise be able to type out, such as a carriage returns.
2. To allow you to represent multiple quotes in a string without JavaScript misinterpreting what you mean.

We learned this in the previous challenge.

Code	Output
\'	single quote
\"	double quote
\\"	backslash
\n	newline
\r	carriage return
\t	tab
\b	word boundary
\f	form feed

Note that the backslash itself must be escaped in order to display as a backslash.

Instructions

Assign the following three lines of text into the single variable `myStr` using escape sequences.

```
FirstLine
\SecondLine
ThirdLine
```

You will need to use escape sequences to insert special characters correctly. You will also need to follow the spacing as it looks above, with no spaces between escape sequences or words. Here is the text with the escape sequences written out. “FirstLine newline tab backslash SecondLine newline ThirdLine”

Challenge Seed

```
var myStr; // Change this line
```

After Test

```
(function(){
  if (myStr !== undefined){
    console.log('myStr:\n' + myStr);}})();
```

Solution

```
var myStr = "FirstLine\n\t\\SecondLine\nThirdLine";
```

25. Concatenating Strings with Plus Operator

Description

In JavaScript, when the `+` operator is used with a `String` value, it is called the concatenation operator. You can build a new string out of other strings by concatenating them together. **Example**

'My name is Alan,' + ' I concatenate.'

Note

Watch out for spaces. Concatenation does not add spaces between concatenated strings, so you'll need to add them yourself.

Instructions

Build `myStr` from the strings `"This is the start. "` and `"This is the end."` using the `+` operator.

Challenge Seed

```
// Example
var ourStr = "I come first. " + "I come second.';

// Only change code below this line

var myStr;
```

After Test

```
(function(){
  if(typeof myStr === 'string') {
    return 'myStr = "' + myStr + '"';
  } else {
    return 'myStr is not a string';
  }
})();
```

Solution

```
var ourStr = "I come first. " + "I come second.";
var myStr = "This is the start. " + "This is the end.";
```

26. Concatenating Strings with the Plus Equals Operator

Description

We can also use the `+=` operator to concatenate a string onto the end of an existing string variable. This can be very helpful to break a long string over several lines. **Note**

Watch out for spaces. Concatenation does not add spaces between concatenated strings, so you'll need to add them yourself.

Instructions

Build `myStr` over several lines by concatenating these two strings: "This is the first sentence. " and "This is the second sentence." using the `+=` operator. Use the `+=` operator similar to how it is shown in the editor. Start by assigning the first string to `myStr`, then add on the second string.

Challenge Seed

```
// Example
var ourStr = "I come first. ";
ourStr += "I come second.";

// Only change code below this line

var myStr;
```

After Test

```
(function(){
  if(typeof myStr === 'string') {
    return 'myStr = "' + myStr + '"';
  } else {
    return 'myStr is not a string';
  }
})();
```

Solution

```
var ourStr = "I come first. ";
ourStr += "I come second.";

var myStr = "This is the first sentence. ";
myStr += "This is the second sentence. ";
```

27. Constructing Strings with Variables

Description

Sometimes you will need to build a string, [Mad Libs](#) style. By using the concatenation operator (`+`), you can insert one or more variables into a string you're building.

Instructions

Set `myName` to a string equal to your name and build `myStr` with `myName` between the strings "My name is " and " and I am well!"

Challenge Seed

```
// Example
var ourName = "freeCodeCamp";
var ourStr = "Hello, our name is " + ourName + ", how are you?";

// Only change code below this line
var myName;
var myStr;
```

After Test

```
(function(){
  var output = [];
  if(typeof myName === 'string') {
    output.push('myName = "' + myName + '"');
  } else {
    output.push('myName is not a string');
  }
  if(typeof myStr === 'string') {
    output.push('myStr = "' + myStr + '"');
  } else {
    output.push('myStr is not a string');
  }
  return output.join('\n');
})();
```

Solution

```
var myName = "Bob";
var myStr = "My name is " + myName + " and I am well!";
```

28. Appending Variables to Strings

Description

Just as we can build a string over multiple lines out of string literals, we can also append variables to a string using the plus equals (`+=`) operator.

Instructions

Set `someAdjective` and append it to `myStr` using the `+=` operator.

Challenge Seed

```
// Example
var anAdjective = "awesome!";
var ourStr = "freeCodeCamp is ";
ourStr += anAdjective;

// Only change code below this line

var someAdjective;
var myStr = "Learning to code is ";
```

After Test

```
(function(){
  var output = [];
  if(typeof someAdjective === 'string') {
    output.push('someAdjective = "' + someAdjective + '"');
```

```

} else {
  output.push('someAdjective is not a string');
}
if(typeof myStr === 'string') {
  output.push('myStr = "' + myStr + '"');
} else {
  output.push('myStr is not a string');
}
return output.join('\n');
})());

```

Solution

```

var anAdjective = "awesome!";
var ourStr = "freeCodeCamp is ";
ourStr += anAdjective;

var someAdjective = "neat";
var myStr = "Learning to code is ";
myStr += someAdjective;

```

29. Find the Length of a String

Description

You can find the length of a `String` value by writing `.length` after the string variable or string literal. `"Alan Peter".length; // 10` For example, if we created a variable `var firstName = "Charles"`, we could find out how long the string `"Charles"` is by using the `firstName.length` property.

Instructions

Use the `.length` property to count the number of characters in the `lastName` variable and assign it to `lastNameLength`.

Challenge Seed

```

// Example
var firstNameLength = 0;
var firstName = "Ada";

firstNameLength = firstName.length;

// Setup
var lastNameLength = 0;
var lastName = "Lovelace";

// Only change code below this line.

lastNameLength = lastName;

```

After Test

```
if(typeof lastNameLength !== "undefined"){{function(){return lastNameLength;}}()();}
```

Solution

```

var firstNameLength = 0;
var firstName = "Ada";
firstNameLength = firstName.length;

```

```
var lastNameLength = 0;
var lastName = "Lovelace";
lastNameLength = lastName.length;
```

30. Use Bracket Notation to Find the First Character in a String

Description

Bracket notation is a way to get a character at a specific index within a string. Most modern programming languages, like JavaScript, don't start counting at 1 like humans do. They start at 0. This is referred to as Zero-based indexing. For example, the character at index 0 in the word "Charles" is "C". So if `var firstName = "Charles"`, you can get the value of the first letter of the string by using `firstName[0]`.

Instructions

Use bracket notation to find the first character in the `lastName` variable and assign it to `firstLetterOfLastName`. Hint Try looking at the `firstLetterOfFirstName` variable declaration if you get stuck.

Challenge Seed

```
// Example
var firstLetterOfFirstName = "";
var firstName = "Ada";

firstLetterOfFirstName = firstName[0];

// Setup
var firstLetterOfLastName = "";
var lastName = "Lovelace";

// Only change code below this line
firstLetterOfLastName = lastName;
```

After Test

```
(function(v){return v;})(firstLetterOfLastName);
```

Solution

```
var firstLetterOfLastName = "";
var lastName = "Lovelace";

// Only change code below this line
firstLetterOfLastName = lastName[0];
```

31. Understand String Immutability

Description

In JavaScript, `String` values are immutable, which means that they cannot be altered once created. For example, the following code:

```
var myStr = "Bob";
myStr[0] = "J";
```

cannot change the value of `myStr` to "Job", because the contents of `myStr` cannot be altered. Note that this does *not* mean that `myStr` cannot be changed, just that the individual characters of a string literal cannot be changed. The only way to change `myStr` would be to assign it with a new string, like this:

```
var myStr = "Bob";
myStr = "Job";
```

Instructions

Correct the assignment to `myStr` so it contains the string value of `Hello World` using the approach shown in the example above.

Challenge Seed

```
// Setup
var myStr = "Jello World";

// Only change code below this line

myStr[0] = "H"; // Fix Me
```

After Test

```
(function(v){return "myStr = " + v;})(myStr);
```

Solution

```
var myStr = "Jello World";
myStr = "Hello World";
```

32. Use Bracket Notation to Find the Nth Character in a String

Description

You can also use bracket notation to get the character at other positions within a string. Remember that computers start counting at `0`, so the first character is actually the zeroth character.

Instructions

Let's try to set `thirdLetterOfLastName` to equal the third letter of the `lastName` variable using bracket notation. Hint Try looking at the `secondLetterOfFirstName` variable declaration if you get stuck.

Challenge Seed

```
// Example
var firstName = "Ada";
var secondLetterOfFirstName = firstName[1];

// Setup
var lastName = "Lovelace";

// Only change code below this line.
```

```
var thirdLetterOfLastName = lastName;
```

After Test

```
(function(v){return v;})(thirdLetterOfLastName);
```

Solution

```
var lastName = "Lovelace";
var thirdLetterOfLastName = lastName[2];
```

33. Use Bracket Notation to Find the Last Character in a String

Description

In order to get the last letter of a string, you can subtract one from the string's length. For example, if `var firstName = "Charles"`, you can get the value of the last letter of the string by using `firstName(firstName.length - 1)`.

Instructions

Use bracket notation to find the last character in the `lastName` variable. **Hint**
Try looking at the `lastLetterOfFirstName` variable declaration if you get stuck.

Challenge Seed

```
// Example
var firstName = "Ada";
var lastLetterOfFirstName = firstName(firstName.length - 1);

// Setup
var lastName = "Lovelace";

// Only change code below this line.
var lastLetterOfLastName = lastName;
```

After Test

```
(function(v){return v;})(lastLetterOfLastName);
```

Solution

```
var firstName = "Ada";
var lastLetterOfFirstName = firstName(firstName.length - 1);

var lastName = "Lovelace";
var lastLetterOfLastName = lastName(lastName.length - 1);
```

34. Use Bracket Notation to Find the Nth-to-Last Character in a String

Description

You can use the same principle we just used to retrieve the last character in a string to retrieve the Nth-to-last character. For example, you can get the value of the third-to-last letter of the `var firstName = "Charles"` string by using `firstName[firstName.length - 3]`

Instructions

Use bracket notation to find the second-to-last character in the `lastName` string. **Hint**

Try looking at the `thirdToLastLetterOfFirstName` variable declaration if you get stuck.

Challenge Seed

```
// Example
var firstName = "Ada";
var thirdToLastLetterOfFirstName = firstName[firstName.length - 3];

// Setup
var lastName = "Lovelace";

// Only change code below this line
var secondToLastLetterOfLastName = lastName;
```

After Test

```
(function(v){return v;})(secondToLastLetterOfLastName);
```

Solution

```
var firstName = "Ada";
var thirdToLastLetterOfFirstName = firstName[firstName.length - 3];

var lastName = "Lovelace";
var secondToLastLetterOfLastName = lastName[lastName.length - 2];
```

35. Word Blanks

Description

We will now use our knowledge of strings to build a "Mad Libs" style word game we're calling "Word Blanks". You will create an (optionally humorous) "Fill in the Blanks" style sentence. In a "Mad Libs" game, you are provided sentences with some missing words, like nouns, verbs, adjectives and adverbs. You then fill in the missing pieces with words of your choice in a way that the completed sentence makes sense. Consider this sentence - "It was really ___, and we ___ ourselves ___. This sentence has three missing pieces- an adjective, a verb and an adverb, and we can add words of our choice to complete it. We can then assign the completed sentence to a variable as follows:

```
var sentence = "It was really" + "hot" + ", and we" + "laughed" + "ourselves" + "silly.;"
```

Instructions

In this challenge, we provide you with a noun, a verb, an adjective and an adverb. You need to form a complete sentence using words of your choice, along with the words we provide. You will need to use the string concatenation operator `+` to build a new string, using the provided variables: `myNoun` , `myAdjective` , `myVerb` , and `myAdverb` . You will then assign the formed string to the `result` variable. You will also need to account for spaces in your string, so that the final sentence has spaces between all the words. The result should be a complete sentence.

Challenge Seed

```
function wordBlanks(myNoun, myAdjective, myVerb, myAdverb) {
  // Your code below this line
  var result = "";

  // Your code above this line
  return result;
}

// Change the words here to test your function
wordBlanks("dog", "big", "ran", "quickly");
```

After Test

```
var test1 = wordBlanks("dog", "big", "ran", "quickly");
var test2 = wordBlanks("cat", "little", "hit", "slowly");
```

Solution

```
function wordBlanks(myNoun, myAdjective, myVerb, myAdverb) {
  var result = "";

  result = "Once there was a " + myNoun + " which was very " + myAdjective + ". ";
  result += "It " + myVerb + " " + myAdverb + " around the yard.";

  return result;
}
```

36. Store Multiple Values in one Variable using JavaScript Arrays

Description

With JavaScript array variables, we can store several pieces of data in one place. You start an array declaration with an opening square bracket, end it with a closing square bracket, and put a comma between each entry, like this: `var sandwich = ["peanut butter", "jelly", "bread"]`.

Instructions

Modify the new array `myArray` so that it contains both a `string` and a `number` (in that order). **Hint**
Refer to the example code in the text editor if you get stuck.

Challenge Seed

```
// Example
var ourArray = ["John", 23];

// Only change code below this line.
var myArray = [];
```

After Test

```
(function(z){return z;})(myArray);
```

Solution

```
var myArray = ["The Answer", 42];
```

37. Nest one Array within Another Array

Description

You can also nest arrays within other arrays, like this: `[["Bulls", 23], ["White Sox", 45]]`. This is also called a Multi-dimensional Array.

Instructions

Create a nested array called `myArray`.

Challenge Seed

```
// Example
var ourArray = [[ "the universe", 42], [ "everything", 101010]];

// Only change code below this line.
var myArray = [];
```

After Test

```
if(typeof myArray !== "undefined"){{function(){return myArray;}}()();}
```

Solution

```
var myArray = [[1,2,3]];
```

38. Access Array Data with Indexes

Description

We can access the data inside arrays using `indexes`. Array indexes are written in the same bracket notation that strings use, except that instead of specifying a character, they are specifying an entry in the array. Like strings, arrays use zero-based indexing, so the first element in an array is element `0`.

Example

```
var array = [50,60,70];
array[0]; // equals 50
var data = array[1]; // equals 60
```

Note

There shouldn't be any spaces between the array name and the square brackets, like `array [0]`. Although JavaScript is able to process this correctly, this may confuse other programmers reading your code.

Instructions

Create a variable called `myData` and set it to equal the first value of `myArray` using bracket notation.

Challenge Seed

```
// Example
var ourArray = [50,60,70];
var ourData = ourArray[0]; // equals 50

// Setup
var myArray = [50,60,70];

// Only change code below this line.
```

After Test

```
if(typeof myArray !== "undefined" && typeof myData !== "undefined"){{function(y,z){return 'myArray = ' +
JSON.stringify(y) + ', myData = ' + JSON.stringify(z);}})(myArray, myData);}
```

Solution

```
var myArray = [50,60,70];
var myData = myArray[0];
```

39. Modify Array Data With Indexes

Description

Unlike strings, the entries of arrays are mutable and can be changed freely. **Example**

```
var ourArray = [50,40,30];
ourArray[0] = 15; // equals [15,40,30]
```

Note

There shouldn't be any spaces between the array name and the square brackets, like `array [0]`. Although JavaScript is able to process this correctly, this may confuse other programmers reading your code.

Instructions

Modify the data stored at index `0` of `myArray` to a value of `45`.

Challenge Seed

```
// Example
var ourArray = [18,64,99];
ourArray[1] = 45; // ourArray now equals [18,45,99].

// Setup
var myArray = [18,64,99];

// Only change code below this line.
```

After Test

```
if(typeof myArray !== "undefined"){{function(){return myArray;}}()();}
```

Solution

```
var myArray = [18,64,99];
myArray[0] = 45;
```

40. Access Multi-Dimensional Arrays With Indexes

Description

One way to think of a multi-dimensional array, is as an *array of arrays*. When you use brackets to access your array, the first set of brackets refers to the entries in the outer-most (the first level) array, and each additional pair of brackets refers to the next level of entries inside. **Example**

```
var arr = [
  [1,2,3],
  [4,5,6],
  [7,8,9],
  [[10,11,12], 13, 14]
];
arr[3]; // equals [[10,11,12], 13, 14]
arr[3][0]; // equals [10,11,12]
arr[3][0][1]; // equals 11
```

Note

There shouldn't be any spaces between the array name and the square brackets, like `array [0][0]` and even this `array [0] [0]` is not allowed. Although JavaScript is able to process this correctly, this may confuse other programmers reading your code.

Instructions

Using bracket notation select an element from `myArray` such that `myData` is equal to `8`.

Challenge Seed

```
// Setup
var myArray = [[1,2,3], [4,5,6], [7,8,9], [[10,11,12], 13, 14]];

// Only change code below this line.
var myData = myArray[0][0];
```

After Test

```
if(typeof myArray !== "undefined"){(function(){return "myData: " + myData + " myArray: " +
JSON.stringify(myArray);})()}
```

Solution

```
var myArray = [[1,2,3],[4,5,6], [7,8,9], [[10,11,12], 13, 14]];
var myData = myArray[2][1];
```

41. Manipulate Arrays With `push()`

Description

An easy way to append data to the end of an array is via the `push()` function. `.push()` takes one or more parameters and "pushes" them onto the end of the array.

```
var arr = [1,2,3];
arr.push(4);
// arr is now [1,2,3,4]
```

Instructions

Push ["dog", 3] onto the end of the myArray variable.

Challenge Seed

```
// Example
var ourArray = ["Stimpson", "J", "cat"];
ourArray.push(["happy", "joy"]);
// ourArray now equals ["Stimpson", "J", "cat", ["happy", "joy"]]

// Setup
var myArray = [[{"John": 23}, {"cat": 2}]];

// Only change code below this line.
```

After Test

```
(function(z){return 'myArray = ' + JSON.stringify(z);})(myArray);
```

Solution

```
var myArray = [{"John": 23}, {"cat": 2}];
myArray.push(["dog", 3]);
```

42. Manipulate Arrays With pop()

Description

Another way to change the data in an array is with the `.pop()` function. `.pop()` is used to "pop" a value off of the end of an array. We can store this "popped off" value by assigning it to a variable. In other words, `.pop()` removes the last element from an array and returns that element. Any type of entry can be "popped" off of an array - numbers, strings, even nested arrays.

```
var threeArr = [1, 4, 6]; var oneDown = threeArr.pop(); console.log(oneDown); // Returns 6
console.log(threeArr); // Returns [1, 4]
```

Instructions

Use the `.pop()` function to remove the last item from `myArray`, assigning the "popped off" value to `removedFromMyArray`.

Challenge Seed

```
// Example
var ourArray = [1,2,3];
var removedFromOurArray = ourArray.pop();
// removedFromOurArray now equals 3, and ourArray now equals [1,2]

// Setup
var myArray = [{"John": 23}, {"cat": 2}];

// Only change code below this line.
var removedFromMyArray;
```

After Test

```
(function(y, z){return 'myArray = ' + JSON.stringify(y) + ' & removedFromMyArray = ' +
JSON.stringify(z);})(myArray, removedFromMyArray);
```

Solution

```
var myArray = [[{"John", 23}, ["cat", 2]];
var removedFromMyArray = myArray.pop();
```

43. Manipulate Arrays With shift()

Description

`pop()` always removes the last element of an array. What if you want to remove the first? That's where `.shift()` comes in. It works just like `.pop()`, except it removes the first element instead of the last.

Instructions

Use the `.shift()` function to remove the first item from `myArray`, assigning the "shifted off" value to `removedFromMyArray`.

Challenge Seed

```
// Example
var ourArray = ["Stimpson", "J", ["cat"]];
var removedFromOurArray = ourArray.shift();
// removedFromOurArray now equals "Stimpson" and ourArray now equals ["J", ["cat"]].

// Setup
var myArray = [[{"John", 23}, ["dog", 3]];

// Only change code below this line.
var removedFromMyArray;
```

After Test

```
(function(y, z){return 'myArray = ' + JSON.stringify(y) + ' & removedFromMyArray = ' +
JSON.stringify(z);})(myArray, removedFromMyArray);
```

Solution

```
var myArray = [[{"John", 23}, ["dog", 3]];

// Only change code below this line.
var removedFromMyArray = myArray.shift();
```

44. Manipulate Arrays With unshift()

Description

Not only can you `shift` elements off of the beginning of an array, you can also `unshift` elements to the beginning of an array i.e. add elements in front of the array. `.unshift()` works exactly like `.push()`, but instead of adding the element at the end of the array, `unshift()` adds the element at the beginning of the array.

Instructions

Add `["Paul", 35]` to the beginning of the `myArray` variable using `unshift()`.

Challenge Seed

```
// Example
var ourArray = ["Stimpson", "J", "cat"];
ourArray.shift(); // ourArray now equals ["J", "cat"]
ourArray.unshift("Happy");
// ourArray now equals ["Happy", "J", "cat"]

// Setup
var myArray = [[{"John": 23}, {"dog": 3}]];
myArray.shift();

// Only change code below this line.
```

After Test

```
(function(y, z){return 'myArray = ' + JSON.stringify(y);})(myArray);
```

Solution

```
var myArray = [{"John": 23}, {"dog": 3}];
myArray.shift();
myArray.unshift(["Paul", 35]);
```

45. Shopping List

Description

Create a shopping list in the variable `myList`. The list should be a multi-dimensional array containing several sub-arrays. The first element in each sub-array should contain a string with the name of the item. The second element should be a number representing the quantity i.e. `["Chocolate Bar", 15]`. There should be at least 5 sub-arrays in the list.

Instructions

Challenge Seed

```
var myList = [];
```

After Test

```
var count = 0;
var isArray = false;
var hasString = false;
var hasNumber = false;
(function(list){
  if(Array.isArray(myList)) {
    isArray = true;
    if(myList.length > 0) {
      hasString = true;
      hasNumber = true;
    }
  }
})({});
```

```

        for (var elem of myList) {
            if(!elem || !elem[0] || typeof elem[0] !== 'string') {
                hasString = false;
            }
            if(!elem || typeof elem[1] !== 'number') {
                hasNumber = false;
            }
        }
        count = myList.length;
        return JSON.stringify(myList);
    } else {
        return "myList is not an array";
    }
})(myList);

```

Solution

```

var myList = [
    ["Candy", 10],
    ["Potatoes", 12],
    ["Eggs", 12],
    ["Catfood", 1],
    ["Toads", 9]
];

```

46. Write Reusable JavaScript with Functions

Description

In JavaScript, we can divide up our code into reusable parts called functions. Here's an example of a function:

```

function functionName() {
    console.log("Hello World");
}

```

You can call or invoke this function by using its name followed by parentheses, like this: `functionName();` Each time the function is called it will print out the message "Hello World" on the dev console. All of the code between the curly braces will be executed every time the function is called.

Instructions

1. Create a function called `reusableFunction` which prints "Hi World" to the dev console.
2. Call the function.

Challenge Seed

```

// Example
function ourReusableFunction() {
    console.log("Heyya, World");
}

ourReusableFunction();

// Only change code below this line

```

Before Test

```

var logOutput = "";
var originalConsole = console
function capture() {

```

```

var nativeLog = console.log;
console.log = function (message) {
    if(message && message.trim) logOutput = message.trim();
    if(nativeLog.apply) {
        nativeLog.apply(originalConsole, arguments);
    } else {
        var nativeMsg = Array.prototype.slice.apply(arguments).join(' ');
        nativeLog(nativeMsg);
    }
};

function uncapture() {
    console.log = originalConsole.log;
}

capture();

```

After Test

```

uncapture();

if (typeof reusableFunction !== "function") {
    (function() { return "reusableFunction is not defined"; })();
} else {
    (function() { return logOutput || "console.log never called"; })();
}

```

Solution

```

function reusableFunction() {
    console.log("Hi World");
}
reusableFunction();

```

47. Passing Values to Functions with Arguments

Description

Parameters are variables that act as placeholders for the values that are to be input to a function when it is called. When a function is defined, it is typically defined along with one or more parameters. The actual values that are input (or "passed") into a function when it is called are known as arguments. Here is a function with two parameters, param1 and param2 :

```

function testFun(param1, param2) {
    console.log(param1, param2);
}

```

Then we can call `testFun` : `testFun("Hello", "World")`; We have passed two arguments, "Hello" and "World". Inside the function, param1 will equal "Hello" and param2 will equal "World". Note that you could call `testFun` again with different arguments and the parameters would take on the value of the new arguments.

Instructions

1. Create a function called `functionWithArgs` that accepts two arguments and outputs their sum to the dev console.
2. Call the function with two numbers as arguments.

Challenge Seed

```

// Example
function ourFunctionWithArgs(a, b) {
    console.log(a - b);
}

```

```

}
ourFunctionWithArgs(10, 5); // Outputs 5

// Only change code below this line.

```

Before Test

```

var logOutput = "";
var originalConsole = console
function capture() {
    var nativeLog = console.log;
    console.log = function (message) {
        if(message) logOutput = JSON.stringify(message).trim();
        if(nativeLog.apply) {
            nativeLog.apply(originalConsole, arguments);
        } else {
            var nativeMsg = Array.prototype.slice.apply(arguments).join(' ');
            nativeLog(nativeMsg);
        }
    };
}

function uncapture() {
    console.log = originalConsole.log;
}

capture();

```

After Test

```

uncapture();

if (typeof functionWithArgs !== "function") {
    (function() { return "functionWithArgs is not defined"; })();
} else {
    (function() { return logOutput || "console.log never called"; })();
}

```

Solution

```

function functionWithArgs(a, b) {
    console.log(a + b);
}
functionWithArgs(10, 5);

```

48. Global Scope and Functions

Description

In JavaScript, scope refers to the visibility of variables. Variables which are defined outside of a function block have Global scope. This means, they can be seen everywhere in your JavaScript code. Variables which are used without the `var` keyword are automatically created in the `global` scope. This can create unintended consequences elsewhere in your code or when running a function again. You should always declare your variables with `var`.

Instructions

Using `var`, declare a global variable `myGlobal` outside of any function. Initialize it with a value of `10`. Inside function `fun1`, assign `5` to `oopsGlobal` **without** using the `var` keyword.

Challenge Seed

```
// Declare your variable here

function fun1() {
  // Assign 5 to oopsGlobal Here
}

// Only change code above this line
function fun2() {
  var output = "";
  if (typeof myGlobal != "undefined") {
    output += "myGlobal: " + myGlobal;
  }
  if (typeof oopsGlobal != "undefined") {
    output += " oopsGlobal: " + oopsGlobal;
  }
  console.log(output);
}
```

Before Test

```
var logOutput = "";
var originalConsole = console
function capture() {
  var nativeLog = console.log;
  console.log = function (message) {
    logOutput = message;
    if(nativeLog.apply) {
      nativeLog.apply(originalConsole, arguments);
    } else {
      var nativeMsg = Array.prototype.slice.apply(arguments).join(' ');
      nativeLog(nativeMsg);
    }
  };
}

function uncapture() {
  console.log = originalConsole.log;
}
var oopsGlobal;
capture();
```

After Test

```
fun1();
fun2();
uncapture();
(function() { return logOutput || "console.log never called"; })();
```

Solution

```
// Declare your variable here
var myGlobal = 10;

function fun1() {
  // Assign 5 to oopsGlobal Here
  oopsGlobal = 5;
}

// Only change code above this line
function fun2() {
  var output = "";
  if(typeof myGlobal != "undefined") {
    output += "myGlobal: " + myGlobal;
  }
  if(typeof oopsGlobal != "undefined") {
    output += " oopsGlobal: " + oopsGlobal;
  }
}
```

```
        console.log(output);
    }
```

49. Local Scope and Functions

Description

Variables which are declared within a function, as well as the function parameters have local scope. That means, they are only visible within that function. Here is a function `myTest` with a local variable called `loc`.

```
function myTest() {
    var loc = "foo";
    console.log(loc);
}
myTest(); // logs "foo"
console.log(loc); // loc is not defined
```

`loc` is not defined outside of the function.

Instructions

Declare a local variable `myVar` inside `myLocalScope`. Run the tests and then follow the instructions commented out in the editor. Hint

Refreshing the page may help if you get stuck.

Challenge Seed

```
function myLocalScope() {
    'use strict'; // you shouldn't need to edit this line

    console.log(myVar);
}
myLocalScope();

// Run and check the console
// myVar is not defined outside of myLocalScope
console.log(myVar);

// Now remove the console log line to pass the test
```

Before Test

```
var logOutput = "";
var originalConsole = console
function capture() {
    var nativeLog = console.log;
    console.log = function (message) {
        logOutput += message;
        if(nativeLog.apply) {
            nativeLog.apply(originalConsole, arguments);
        } else {
            var nativeMsg = Array.prototype.slice.apply(arguments).join(' ');
            nativeLog(nativeMsg);
        }
    };
}

function uncapture() {
    console.log = originalConsole.log;
}
```

After Test

```
typeof myLocalScope === 'function' && (capture(), myLocalScope(), uncapture());
(function() { return logOutput || "console.log never called"; })();
```

Solution

```
function myLocalScope() {
  'use strict';

  var myVar;
  console.log(myVar);
}
myLocalScope();
```

50. Global vs. Local Scope in Functions

Description

It is possible to have both local and global variables with the same name. When you do this, the `local` variable takes precedence over the `global` variable. In this example:

```
var someVar = "Hat";
function myFun() {
  var someVar = "Head";
  return someVar;
}
```

The function `myFun` will return "Head" because the `local` version of the variable is present.

Instructions

Add a local variable to `myOutfit` function to override the value of `outerWear` with "sweater".

Challenge Seed

```
// Setup
var outerWear = "T-Shirt";

function myOutfit() {
  // Only change code below this line

  // Only change code above this line
  return outerWear;
}

myOutfit();
```

Solution

```
var outerWear = "T-Shirt";
function myOutfit() {
  var outerWear = "sweater";
  return outerWear;
}
```

51. Return a Value from a Function with Return

Description

We can pass values into a function with arguments. You can use a `return` statement to send a value back out of a function. **Example**

```
function plusThree(num) {
  return num + 3;
}

var answer = plusThree(5); // 8
```

`plusThree` takes an argument for `num` and returns a value equal to `num + 3`.

Instructions

Create a function `timesFive` that accepts one argument, multiplies it by `5`, and returns the new value. See the last line in the editor for an example of how you can test your `timesFive` function.

Challenge Seed

```
// Example
function minusSeven(num) {
  return num - 7;
}

// Only change code below this line

console.log(minusSeven(10));
```

Solution

```
function timesFive(num) {
  return num * 5;
}

timesFive(10);
```

52. Understanding Undefined Value returned from a Function

Description

A function can include the `return` statement but it does not have to. In the case that the function doesn't have a `return` statement, when you call it, the function processes the inner code but the returned value is `undefined`.

Example

```
var sum = 0;
function addSum(num) {
  sum = sum + num;
}
var returnedValue = addSum(3); // sum will be modified but returned value is undefined
```

`addSum` is a function without a `return` statement. The function will change the global `sum` variable but the returned value of the function is `undefined`

Instructions

Create a function `addFive` without any arguments. This function adds 5 to the `sum` variable, but its returned value is `undefined`.

Challenge Seed

```
// Example
var sum = 0;
function addThree() {
  sum = sum + 3;
}

// Only change code below this line
```

```
// Only change code above this line
var returnedValue = addFive();
```

After Test

```
var sum = 0;
function addThree() {sum = sum + 3;}
addThree();
addFive();
```

Solution

```
function addFive() {
  sum = sum + 5;
}
```

53. Assignment with a Returned Value

Description

If you'll recall from our discussion of [Storing Values with the Assignment Operator](#), everything to the right of the equal sign is resolved before the value is assigned. This means we can take the return value of a function and assign it to a variable. Assume we have pre-defined a function `sum` which adds two numbers together, then: `ourSum = sum(5, 12);` will call `sum` function, which returns a value of `17` and assigns it to `ourSum` variable.

Instructions

Call the `processArg` function with an argument of `7` and assign its return value to the variable `processed`.

Challenge Seed

```
// Example
var changed = 0;

function change(num) {
  return (num + 5) / 3;
}

changed = change(10);

// Setup
var processed = 0;

function processArg(num) {
  return (num + 3) / 5;
}

// Only change code below this line
```

After Test

```
(function(){return "processed = " + processed})();
```

Solution

```
var processed = 0;

function processArg(num) {
    return (num + 3) / 5;
}

processed = processArg(7);
```

54. Stand in Line

Description

In Computer Science a queue is an abstract Data Structure where items are kept in order. New items can be added at the back of the queue and old items are taken off from the front of the queue . Write a function `nextInLine` which takes an array (`arr`) and a number (`item`) as arguments. Add the number to the end of the array, then remove the first element of the array. The `nextInLine` function should then return the element that was removed.

Instructions

Challenge Seed

```
function nextInLine(arr, item) {
    // Your code here

    return item; // Change this line
}

// Test Setup
var testArr = [1,2,3,4,5];

// Display Code
console.log("Before: " + JSON.stringify(testArr));
console.log(nextInLine(testArr, 6)); // Modify this line to test
console.log("After: " + JSON.stringify(testArr));
```

Before Test

```
var logOutput = [];
var originalConsole = console
function capture() {
    var nativeLog = console.log;
    console.log = function (message) {
        logOutput.push(message);
        if(nativeLog.apply) {
            nativeLog.apply(originalConsole, arguments);
        } else {
            var nativeMsg = Array.prototype.slice.apply(arguments).join(' ');
            nativeLog(nativeMsg);
        }
    };
}

function uncapture() {
    console.log = originalConsole.log;
```

}

capture();

After Test

```
uncapture();
testArr = [1,2,3,4,5];
(function() { return logOutput.join("\n"); })();
```

Solution

```
var testArr = [1,2,3,4,5];

function nextInLine(arr, item) {
    arr.push(item);
    return arr.shift();
}
```

55. Understanding Boolean Values

Description

Another data type is the Boolean. Booleans may only be one of two values: `true` or `false`. They are basically little on-off switches, where `true` is "on" and `false` is "off." These two states are mutually exclusive. **Note** Boolean values are never written with quotes. The strings "true" and "false" are not Boolean and have no special meaning in JavaScript.

Instructions

Modify the `welcomeToBooleans` function so that it returns `true` instead of `false` when the run button is clicked.

Challenge Seed

```
function welcomeToBooleans() {

    // Only change code below this line.

    return false; // Change this line

    // Only change code above this line.
}
```

After Test

welcomeToBooleans();

Solution

```
function welcomeToBooleans() {
    return true; // Change this line
}
```

56. Use Conditional Logic with If Statements

Description

If statements are used to make decisions in code. The keyword `if` tells JavaScript to execute the code in the curly braces under certain conditions, defined in the parentheses. These conditions are known as Boolean conditions and they may only be `true` or `false`. When the condition evaluates to `true`, the program executes the statement inside the curly braces. When the Boolean condition evaluates to `false`, the statement inside the curly braces will not execute. **Pseudocode**

```
if (condition is true) {
    statement is executed
}
```

Example

```
function test (myCondition) {
    if (myCondition) {
        return "It was true";
    }
    return "It was false";
}
test(true); // returns "It was true"
test(false); // returns "It was false"
```

When `test` is called with a value of `true`, the `if` statement evaluates `myCondition` to see if it is `true` or not. Since it is `true`, the function returns `"It was true"`. When we call `test` with a value of `false`, `myCondition` is *not* `true` and the statement in the curly braces is not executed and the function returns `"It was false"`.

Instructions

Create an `if` statement inside the function to return `"Yes, that was true"` if the parameter `wasThatTrue` is `true` and return `"No, that was false"` otherwise.

Challenge Seed

```
// Example
function ourTrueOrFalse(isItTrue) {
    if (isItTrue) {
        return "Yes, it's true";
    }
    return "No, it's false";
}

// Setup
function trueOrFalse(wasThatTrue) {

    // Only change code below this line.

    // Only change code above this line.

}

// Change this value to test
trueOrFalse(true);
```

Solution

```
function trueOrFalse(wasThatTrue) {
    if (wasThatTrue) {
        return "Yes, that was true";
    }
    return "No, that was false";
}
```

57. Comparison with the Equality Operator

Description

There are many Comparison Operators in JavaScript. All of these operators return a boolean `true` or `false` value. The most basic operator is the equality operator `==`. The equality operator compares two values and returns `true` if they're equivalent or `false` if they are not. Note that equality is different from assignment (`=`), which assigns the value at the right of the operator to a variable in the left.

```
function equalityTest(myVal) {
  if (myVal == 10) {
    return "Equal";
  }
  return "Not Equal";
}
```

If `myVal` is equal to `10`, the equality operator returns `true`, so the code in the curly braces will execute, and the function will return `"Equal"`. Otherwise, the function will return `"Not Equal"`. In order for JavaScript to compare two different data types (for example, numbers and strings), it must convert one type to another. This is known as "Type Coercion". Once it does, however, it can compare terms as follows:

```
1 == 1 // true
1 == 2 // false
1 == '1' // true
"3" == 3 // true
```

Instructions

Add the equality operator to the indicated line so that the function will return `"Equal"` when `val` is equivalent to `12`

Challenge Seed

```
// Setup
function testEqual(val) {
  if (val) { // Change this line
    return "Equal";
  }
  return "Not Equal";
}

// Change this value to test
testEqual(10);
```

Solution

```
function testEqual(val) {
  if (val == 12) {
    return "Equal";
  }
  return "Not Equal";
}
```

58. Comparison with the Strict Equality Operator

Description

Strict equality (`==`) is the counterpart to the equality operator (`=`). However, unlike the equality operator, which attempts to convert both values being compared to a common type, the strict equality operator does not perform a type conversion. If the values being compared have different types, they are considered unequal, and the strict equality operator will return false. **Examples**

```
3 === 3 // true
3 === '3' // false
```

In the second example, `3` is a `Number` type and `'3'` is a `String` type.

Instructions

Use the strict equality operator in the `if` statement so the function will return "Equal" when `val` is strictly equal to `7`

Challenge Seed

```
// Setup
function testStrict(val) {
  if (val) { // Change this line
    return "Equal";
  }
  return "Not Equal";
}

// Change this value to test
testStrict(10);
```

Solution

```
function testStrict(val) {
  if (val === 7) {
    return "Equal";
  }
  return "Not Equal";
}
```

59. Practice comparing different values

Description

In the last two challenges, we learned about the equality operator (`=`) and the strict equality operator (`==`). Let's do a quick review and practice using these operators some more. If the values being compared are not of the same type, the equality operator will perform a type conversion, and then evaluate the values. However, the strict equality operator will compare both the data type and value as-is, without converting one type to the other. **Examples**

```
3 == '3' // returns true because JavaScript performs type conversion from string to number
3 === '3' // returns false because the types are different and type conversion is not performed
```

Note

In JavaScript, you can determine the type of a variable or a value with the `typeof` operator, as follows:

```
typeof 3 // returns 'number'
typeof '3' // returns 'string'
```

Instructions

The `compareEquality` function in the editor compares two values using the `equality operator`. Modify the function so that it returns "Equal" only when the values are strictly equal.

Challenge Seed

```
// Setup
function compareEquality(a, b) {
  if (a == b) { // Change this line
    return "Equal";
  }
  return "Not Equal";
}

// Change this value to test
compareEquality(10, "10");
```

Solution

```
function compareEquality(a,b) {
  if (a === b) {
    return "Equal";
  }
  return "Not Equal";
}
```

60. Comparison with the Inequality Operator

Description

The inequality operator (`!=`) is the opposite of the equality operator. It means "Not Equal" and returns `false` where equality would return `true` and *vice versa*. Like the equality operator, the inequality operator will convert data types of values while comparing. **Examples**

```
1 != 2 // true
1 != "1" // false
1 != '1' // false
1 != true // false
0 != false // false
```

Instructions

Add the inequality operator `!=` in the `if` statement so that the function will return "Not Equal" when `val` is not equivalent to `99`

Challenge Seed

```
// Setup
function testNotEqual(val) {
  if (val) { // Change this line
    return "Not Equal";
  }
  return "Equal";
}

// Change this value to test
testNotEqual(10);
```

Solution

```
function testNotEqual(val) {
  if (val != 99) {
    return "Not Equal";
  }
  return "Equal";
}
```

61. Comparison with the Strict Inequality Operator

Description

The strict inequality operator (`!==`) is the logical opposite of the strict equality operator. It means "Strictly Not Equal" and returns `false` where strict equality would return `true` and *vice versa*. Strict inequality will not convert data types.

Examples

```
3 !== 3 // false
3 !== '3' // true
4 !== 3 // true
```

Instructions

Add the strict inequality operator to the `if` statement so the function will return "Not Equal" when `val` is not strictly equal to 17

Challenge Seed

```
// Setup
function testStrictNotEqual(val) {
  // Only Change Code Below this Line

  if (val) {

    // Only Change Code Above this Line

    return "Not Equal";
  }
  return "Equal";
}

// Change this value to test
testStrictNotEqual(10);
```

Solution

```
function testStrictNotEqual(val) {
  if (val !== 17) {
    return "Not Equal";
  }
  return "Equal";
}
```

62. Comparison with the Greater Than Operator

Description

The greater than operator (`>`) compares the values of two numbers. If the number to the left is greater than the number to the right, it returns `true`. Otherwise, it returns `false`. Like the equality operator, greater than operator will convert data types of values while comparing. **Examples**

```
5 > 3 // true
7 > '3' // true
2 > 3 // false
'1' > 9 // false
```

Instructions

Add the greater than operator to the indicated lines so that the return statements make sense.

Challenge Seed

```
function testGreaterThan(val) {
  if (val) { // Change this line
    return "Over 100";
  }

  if (val) { // Change this line
    return "Over 10";
  }

  return "10 or Under";
}

// Change this value to test
testGreaterThan(10);
```

Solution

```
function testGreaterThan(val) {
  if (val > 100) { // Change this line
    return "Over 100";
  }
  if (val > 10) { // Change this line
    return "Over 10";
  }
  return "10 or Under";
}
```

63. Comparison with the Greater Than Or Equal To Operator

Description

The greater than or equal to operator (`>=`) compares the values of two numbers. If the number to the left is greater than or equal to the number to the right, it returns `true`. Otherwise, it returns `false`. Like the equality operator, greater than or equal to operator will convert data types while comparing. **Examples**

```
6 >= 6 // true
7 >= '3' // true
2 >= 3 // false
'7' >= 9 // false
```

Instructions

Add the greater than or equal to operator to the indicated lines so that the return statements make sense.

Challenge Seed

```
function testGreaterOrEqual(val) {
  if (val) { // Change this line
    return "20 or Over";
  }

  if (val) { // Change this line
    return "10 or Over";
  }

  return "Less than 10";
}
```

```
// Change this value to test
testGreaterOrEqual(10);
```

Solution

```
function testGreaterOrEqual(val) {
  if (val >= 20) { // Change this line
    return "20 or Over";
  }

  if (val >= 10) { // Change this line
    return "10 or Over";
  }

  return "Less than 10";
}
```

64. Comparison with the Less Than Operator

Description

The less than operator (`<`) compares the values of two numbers. If the number to the left is less than the number to the right, it returns `true`. Otherwise, it returns `false`. Like the equality operator, less than operator converts data types while comparing. **Examples**

```
2 < 5 // true
'3' < 7 // true
5 < 5 // false
3 < 2 // false
'8' < 4 // false
```

Instructions

Add the `less than` operator to the indicated lines so that the return statements make sense.

Challenge Seed

```
function testLessThan(val) {
  if (val) { // Change this line
    return "Under 25";
  }

  if (val) { // Change this line
    return "Under 55";
  }

  return "55 or Over";
}

// Change this value to test
testLessThan(10);
```

Solution

```
function testLessThan(val) {
  if (val < 25) { // Change this line
    return "Under 25";
  }

  if (val < 55) { // Change this line
    return "Under 55";
```

```

    }

    return "55 or Over";
}

```

65. Comparison with the Less Than Or Equal To Operator

Description

The less than or equal to operator (`<=`) compares the values of two numbers. If the number to the left is less than or equal to the number to the right, it returns `true`. If the number on the left is greater than the number on the right, it returns `false`. Like the equality operator, less than or equal to converts data types. **Examples**

```

4 <= 5 // true
'7' <= 7 // true
5 <= 5 // true
3 <= 2 // false
'8' <= 4 // false

```

Instructions

Add the less than or equal to operator to the indicated lines so that the return statements make sense.

Challenge Seed

```

function testLessOrEqual(val) {
  if (val) { // Change this line
    return "Smaller Than or Equal to 12";
  }

  if (val) { // Change this line
    return "Smaller Than or Equal to 24";
  }

  return "More Than 24";
}

// Change this value to test
testLessOrEqual(10);

```

Solution

```

function testLessOrEqual(val) {
  if (val <= 12) { // Change this line
    return "Smaller Than or Equal to 12";
  }

  if (val <= 24) { // Change this line
    return "Smaller Than or Equal to 24";
  }

  return "More Than 24";
}

```

66. Comparisons with the Logical And Operator

Description

Sometimes you will need to test more than one thing at a time. The logical and operator (`&&`) returns `true` if and only if the operands to the left and right of it are true. The same effect could be achieved by nesting an if statement inside another if:

```
if (num > 5) {
  if (num < 10) {
    return "Yes";
  }
}
return "No";
```

will only return "Yes" if `num` is greater than 5 and less than 10. The same logic can be written as:

```
if (num > 5 && num < 10) {
  return "Yes";
}
return "No";
```

Instructions

Combine the two if statements into one statement which will return "Yes" if `val` is less than or equal to 50 and greater than or equal to 25. Otherwise, will return "No".

Challenge Seed

```
function testLogicalAnd(val) {
  // Only change code below this line

  if (val) {
    if (val) {
      return "Yes";
    }
  }

  // Only change code above this line
  return "No";
}

// Change this value to test
testLogicalAnd(10);
```

Solution

```
function testLogicalAnd(val) {
  if (val >= 25 && val <= 50) {
    return "Yes";
  }
  return "No";
}
```

67. Comparisons with the Logical Or Operator

Description

The logical or operator (`||`) returns `true` if either of the operands is `true`. Otherwise, it returns `false`. The logical or operator is composed of two pipe symbols (`|`). This can typically be found between your Backspace and Enter keys. The pattern below should look familiar from prior waypoints:

```
if (num > 10) {
  return "No";
}
if (num < 5) {
  return "No";
```

```

    }
    return "Yes";
}

```

will return "Yes" only if `num` is between 5 and 10 (5 and 10 included). The same logic can be written as:

```

if(num > 10 || num < 5) {
    return "No";
}
return "Yes";
}

```

Instructions

Combine the two `if` statements into one statement which returns "Outside" if `val` is not between 10 and 20, inclusive. Otherwise, return "Inside".

Challenge Seed

```

function testLogicalOr(val) {
    // Only change code below this line

    if (val) {
        return "Outside";
    }

    if (val) {
        return "Outside";
    }

    // Only change code above this line
    return "Inside";
}

// Change this value to test
testLogicalOr(15);
}

```

Solution

```

function testLogicalOr(val) {
    if (val < 10 || val > 20) {
        return "Outside";
    }
    return "Inside";
}

```

68. Introducing Else Statements

Description

When a condition for an `if` statement is true, the block of code following it is executed. What about when that condition is false? Normally nothing would happen. With an `else` statement, an alternate block of code can be executed.

```

if (num > 10) {
    return "Bigger than 10";
} else {
    return "10 or Less";
}

```

Instructions

Combine the `if` statements into a single `if/else` statement.

Challenge Seed

```
function testElse(val) {
  var result = "";
  // Only change code below this line

  if (val > 5) {
    result = "Bigger than 5";
  }

  if (val <= 5) {
    result = "5 or Smaller";
  }

  // Only change code above this line
  return result;
}

// Change this value to test
testElse(4);
```

Solution

```
function testElse(val) {
  var result = "";
  if(val > 5) {
    result = "Bigger than 5";
  } else {
    result = "5 or Smaller";
  }
  return result;
}
```

69. Introducing Else If Statements

Description

If you have multiple conditions that need to be addressed, you can chain `if` statements together with `else if` statements.

```
if (num > 15) {
  return "Bigger than 15";
} else if (num < 5) {
  return "Smaller than 5";
} else {
  return "Between 5 and 15";
}
```

Instructions

Convert the logic to use `else if` statements.

Challenge Seed

```
function testElseIf(val) {
  if (val > 10) {
    return "Greater than 10";
  }

  if (val < 5) {
    return "Smaller than 5";
  }
```

```

    return "Between 5 and 10";
}

// Change this value to test
testElseIf(7);

```

Solution

```

function testElseIf(val) {
  if(val > 10) {
    return "Greater than 10";
  } else if(val < 5) {
    return "Smaller than 5";
  } else {
    return "Between 5 and 10";
  }
}

```

70. Logical Order in If Else Statements

Description

Order is important in `if`, `else if` statements. The function is executed from top to bottom so you will want to be careful of what statement comes first. Take these two functions as an example. Here's the first:

```

function foo(x) {
  if(x < 1) {
    return "Less than one";
  } else if (x < 2) {
    return "Less than two";
  } else {
    return "Greater than or equal to two";
  }
}

```

And the second just switches the order of the statements:

```

function bar(x) {
  if(x < 2) {
    return "Less than two";
  } else if (x < 1) {
    return "Less than one";
  } else {
    return "Greater than or equal to two";
  }
}

```

While these two functions look nearly identical if we pass a number to both we get different outputs.

```

foo(0) // "Less than one"
bar(0) // "Less than two"

```

Instructions

Change the order of logic in the function so that it will return the correct statements in all cases.

Challenge Seed

```

function orderMyLogic(val) {
  if (val < 10) {
    return "Less than 10";
  } else if (val < 5) {
    return "Less than 5";
  }
}

```

```

} else {
    return "Greater than or equal to 10";
}
}

// Change this value to test
orderMyLogic(7);

```

Solution

```

function orderMyLogic(val) {
    if(val < 5) {
        return "Less than 5";
    } else if (val < 10) {
        return "Less than 10";
    } else {
        return "Greater than or equal to 10";
    }
}

```

71. Chaining If Else Statements

Description

`if/else` statements can be chained together for complex logic. Here is pseudocode of multiple chained `if / else if` statements:

```

if (condition1) {
    statement1
} else if (condition2) {
    statement2
} else if (condition3) {
    statement3
    ...
} else {
    statementN
}

```

Instructions

Write chained `if / else if` statements to fulfill the following conditions:

- `num < 5` - return "Tiny"
- `num < 10` - return "Small"
- `num < 15` - return "Medium"
- `num < 20` - return "Large"
- `num >= 20` - return "Huge"

Challenge Seed

```

function testSize(num) {
    // Only change code below this line

    return "Change Me";
    // Only change code above this line
}

// Change this value to test
testSize(7);

```

Solution

```

function testSize(num) {
  if (num < 5) {
    return "Tiny";
  } else if (num < 10) {
    return "Small";
  } else if (num < 15) {
    return "Medium";
  } else if (num < 20) {
    return "Large";
  } else {
    return "Huge";
  }
}

```

72. Golf Code

Description

In the game of [golf](#) each hole has a `par` meaning the average number of `strokes` a golfer is expected to make in order to sink the ball in a hole to complete the play. Depending on how far above or below `par` your `strokes` are, there is a different nickname. Your function will be passed `par` and `strokes` arguments. Return the correct string according to this table which lists the strokes in order of priority; top (highest) to bottom (lowest):

Strokes	Return
1	"Hole-in-one!"
<= par - 2	"Eagle"
par - 1	"Birdie"
par	"Par"
par + 1	"Bogey"
par + 2	"Double Bogey"
>= par + 3	"Go Home!"

`par` and `strokes` will always be numeric and positive. We have added an array of all the names for your convenience.

Instructions

Challenge Seed

```

var names = ["Hole-in-one!", "Eagle", "Birdie", "Par", "Bogey", "Double Bogey", "Go Home!"];
function golfScore(par, strokes) {
  // Only change code below this line

  return "Change Me";
  // Only change code above this line
}

// Change these values to test
golfScore(5, 4);

```

Solution

```

function golfScore(par, strokes) {
  if (strokes === 1) {
    return "Hole-in-one!";
  }
}

```

```

if (strokes <= par - 2) {
    return "Eagle";
}

if (strokes === par - 1) {
    return "Birdie";
}

if (strokes === par) {
    return "Par";
}

if (strokes === par + 1) {
    return "Bogey";
}

if(strokes === par + 2) {
    return "Double Bogey";
}

return "Go Home!";
}

```

73. Selecting from Many Options with Switch Statements

Description

If you have many options to choose from, use a `switch` statement. A `switch` statement tests a value and can have many `case` statements which define various possible values. Statements are executed from the first matched `case` value until a `break` is encountered. Here is a pseudocode example:

```

switch(num) {
    case value1:
        statement1;
        break;
    case value2:
        statement2;
        break;
    ...
    case valueN:
        statementN;
        break;
}

```

`case` values are tested with strict equality (`==`). The `break` tells JavaScript to stop executing statements. If the `break` is omitted, the next statement will be executed.

Instructions

Write a `switch` statement which tests `val` and sets `answer` for the following conditions:

- 1 - "alpha"
- 2 - "beta"
- 3 - "gamma"
- 4 - "delta"

Challenge Seed

```

function caseInSwitch(val) {
    var answer = "";
    // Only change code below this line
}

```

```
// Only change code above this line
return answer;
}

// Change this value to test
caseInSwitch(1);
```

Solution

```
function caseInSwitch(val) {
  var answer = "";

  switch(val) {
    case 1:
      answer = "alpha";
      break;
    case 2:
      answer = "beta";
      break;
    case 3:
      answer = "gamma";
      break;
    case 4:
      answer = "delta";
  }
  return answer;
}
```

74. Adding a Default Option in Switch Statements

Description

In a `switch` statement you may not be able to specify all possible values as `case` statements. Instead, you can add the `default` statement which will be executed if no matching `case` statements are found. Think of it like the final `else` statement in an `if/else` chain. A `default` statement should be the last case.

```
switch (num) {
  case value1:
    statement1;
    break;
  case value2:
    statement2;
    break;
  ...
  default:
    defaultStatement;
    break;
}
```

Instructions

Write a `switch` statement to set `answer` for the following conditions:

```
"a" - "apple"
"b" - "bird"
"c" - "cat"
default - "stuff"
```

Challenge Seed

```
function switchOfStuff(val) {
  var answer = "";
  // Only change code below this line
```

```
// Only change code above this line
return answer;
}

// Change this value to test
switchOfStuff(1);
```

Solution

```
function switchOfStuff(val) {
  var answer = "";

  switch(val) {
    case "a":
      answer = "apple";
      break;
    case "b":
      answer = "bird";
      break;
    case "c":
      answer = "cat";
      break;
    default:
      answer = "stuff";
  }
  return answer;
}
```

75. Multiple Identical Options in Switch Statements

Description

If the `break` statement is omitted from a `switch` statement's `case`, the following `case` statement(s) are executed until a `break` is encountered. If you have multiple inputs with the same output, you can represent them in a `switch` statement like this:

```
switch(val) {
  case 1:
  case 2:
  case 3:
    result = "1, 2, or 3";
    break;
  case 4:
    result = "4 alone";
}
```

Cases for 1, 2, and 3 will all produce the same result.

Instructions

Write a `switch` statement to set `answer` for the following ranges:

1-3 - "Low"

4-6 - "Mid"

7-9 - "High" **Note**

You will need to have a `case` statement for each number in the range.

Challenge Seed

```

function sequentialSizes(val) {
  var answer = "";
  // Only change code below this line

  // Only change code above this line
  return answer;
}

// Change this value to test
sequentialSizes(1);

```

Solution

```

function sequentialSizes(val) {
  var answer = "";

  switch(val) {
    case 1:
    case 2:
    case 3:
      answer = "Low";
      break;
    case 4:
    case 5:
    case 6:
      answer = "Mid";
      break;
    case 7:
    case 8:
    case 9:
      answer = "High";
  }

  return answer;
}

```

76. Replacing If Else Chains with Switch

Description

If you have many options to choose from, a `switch` statement can be easier to write than many chained `if / else if` statements. The following:

```

if (val === 1) {
  answer = "a";
} else if (val === 2) {
  answer = "b";
} else {
  answer = "c";
}

```

can be replaced with:

```

switch(val) {
  case 1:
    answer = "a";
    break;
  case 2:
    answer = "b";
    break;
  default:
    answer = "c";
}

```

Instructions

Change the chained `if / else if` statements into a `switch` statement.

Challenge Seed

```
function chainToSwitch(val) {
  var answer = "";
  // Only change code below this line

  if (val === "bob") {
    answer = "Marley";
  } else if (val === 42) {
    answer = "The Answer";
  } else if (val === 1) {
    answer = "There is no #1";
  } else if (val === 99) {
    answer = "Missed me by this much!";
  } else if (val === 7) {
    answer = "Ate Nine";
  }

  // Only change code above this line
  return answer;
}

// Change this value to test
chainToSwitch(7);
```

Solution

```
function chainToSwitch(val) {
  var answer = "";

  switch(val) {
    case "bob":
      answer = "Marley";
      break;
    case 42:
      answer = "The Answer";
      break;
    case 1:
      answer = "There is no #1";
      break;
    case 99:
      answer = "Missed me by this much!";
      break;
    case 7:
      answer = "Ate Nine";
  }
  return answer;
}
```

77. Returning Boolean Values from Functions

Description

You may recall from [Comparison with the Equality Operator](#) that all comparison operators return a boolean `true` or `false` value. Sometimes people use an `if/else` statement to do a comparison, like this:

```
function isEqual(a,b) {
  if(a === b) {
    return true;
  } else {
    return false;
```

```

    }
}

```

But there's a better way to do this. Since `==` returns `true` or `false`, we can return the result of the comparison:

```

function isEqual(a,b) {
  return a === b;
}

```

Instructions

Fix the function `isLess` to remove the `if/else` statements.

Challenge Seed

```

function isLess(a, b) {
  // Fix this code
  if (a < b) {
    return true;
  } else {
    return false;
  }
}

// Change these values to test
isLess(10, 15);

```

Solution

```

function isLess(a, b) {
  return a < b;
}

```

78. Return Early Pattern for Functions

Description

When a `return` statement is reached, the execution of the current function stops and control returns to the calling location. **Example**

```

function myFun() {
  console.log("Hello");
  return "World";
  console.log("byebye")
}
myFun();

```

The above outputs "Hello" to the console, returns "World", but "byebye" is never output, because the function exits at the `return` statement.

Instructions

Modify the function `abTest` so that if `a` or `b` are less than `0` the function will immediately exit with a value of `undefined`. **Hint**

Remember that `undefined` is a keyword, not a string.

Challenge Seed

```

// Setup
function abTest(a, b) {

```

```
// Only change code below this line

// Only change code above this line

return Math.round(Math.pow(Math.sqrt(a) + Math.sqrt(b), 2));
}

// Change values below to test your code
abTest(2,2);
```

Solution

```
function abTest(a, b) {
  if(a < 0 || b < 0) {
    return undefined;
  }
  return Math.round(Math.pow(Math.sqrt(a) + Math.sqrt(b), 2));
}
```

79. Counting Cards

Description

In the casino game Blackjack, a player can gain an advantage over the house by keeping track of the relative number of high and low cards remaining in the deck. This is called [Card Counting](#). Having more high cards remaining in the deck favors the player. Each card is assigned a value according to the table below. When the count is positive, the player should bet high. When the count is zero or negative, the player should bet low.

Count Change	Cards
+1	2, 3, 4, 5, 6
0	7, 8, 9
-1	10, 'J', 'Q', 'K', 'A'

You will write a card counting function. It will receive a `card` parameter, which can be a number or a string, and increment or decrement the global `count` variable according to the card's value (see table). The function will then return a string with the current count and the string `Bet` if the count is positive, or `Hold` if the count is zero or negative. The current count and the player's decision (`Bet` or `Hold`) should be separated by a single space. **Example Output**

-3 Hold

5 Bet Hint

Do NOT reset `count` to 0 when value is 7, 8, or 9.

Do NOT return an array.

Do NOT include quotes (single or double) in the output.

Instructions

Challenge Seed

```
var count = 0;

function cc(card) {
  // Only change code below this line

  return "Change Me";
  // Only change code above this line
}
```

```
// Add/remove calls to test your function.
// Note: Only the last will display
cc(2); cc(3); cc(7); cc('K'); cc('A');
```

Solution

```
var count = 0;
function cc(card) {
  switch(card) {
    case 2:
    case 3:
    case 4:
    case 5:
    case 6:
      count++;
      break;
    case 10:
    case 'J':
    case 'Q':
    case 'K':
    case 'A':
      count--;
  }
  if(count > 0) {
    return count + " Bet";
  } else {
    return count + " Hold";
  }
}
```

80. Build JavaScript Objects

Description

You may have heard the term `object` before. Objects are similar to `arrays`, except that instead of using indexes to access and modify their data, you access the data in objects through what are called `properties`. Objects are useful for storing data in a structured way, and can represent real world objects, like a cat. Here's a sample cat object:

```
var cat = {
  "name": "Whiskers",
  "legs": 4,
  "tails": 1,
  "enemies": ["Water", "Dogs"]
};
```

In this example, all the properties are stored as strings, such as - `"name"` , `"legs"` , and `"tails"` . However, you can also use numbers as properties. You can even omit the quotes for single-word string properties, as follows:

```
var anotherObject = {
  make: "Ford",
  5: "five",
  "model": "focus"
};
```

However, if your object has any non-string properties, JavaScript will automatically typecast them as strings.

Instructions

Make an object that represents a dog called `myDog` which contains the properties `"name"` (a string), `"legs"` , `"tails"` and `"friends"` . You can set these object properties to whatever values you want, as long `"name"` is a string, `"legs"` and `"tails"` are numbers, and `"friends"` is an array.

Challenge Seed

```
// Example
var ourDog = {
  "name": "Camper",
  "legs": 4,
  "tails": 1,
  "friends": ["everything!"]
};

// Only change code below this line.

var myDog = {

};
```

After Test

```
(function(z){return z;})(myDog);
```

Solution

```
var myDog = {
  "name": "Camper",
  "legs": 4,
  "tails": 1,
  "friends": ["everything!"]
};
```

81. Accessing Object Properties with Dot Notation

Description

There are two ways to access the properties of an object: dot notation (.) and bracket notation ([]), similar to an array. Dot notation is what you use when you know the name of the property you're trying to access ahead of time. Here is a sample of using dot notation (.) to read an object's property:

```
var myObj = {
  prop1: "val1",
  prop2: "val2"
};
var prop1val = myObj.prop1; // val1
var prop2val = myObj.prop2; // val2
```

Instructions

Read in the property values of `testObj` using dot notation. Set the variable `hatValue` equal to the object's property `hat` and set the variable `shirtValue` equal to the object's property `shirt`.

Challenge Seed

```
// Setup
var testObj = {
  "hat": "ballcap",
  "shirt": "jersey",
  "shoes": "cleats"
};

// Only change code below this line
```

```
var hatValue = testObj;      // Change this line
var shirtValue = testObj;    // Change this line
```

After Test

```
(function(a,b) { return "hatValue = '" + a + "'", shirtValue = '" + b + "'"; })(hatValue,shirtValue);
```

Solution

```
var testObj = {
  "hat": "ballcap",
  "shirt": "jersey",
  "shoes": "cleats"
};

var hatValue = testObj.hat;
var shirtValue = testObj.shirt;
```

82. Accessing Object Properties with Bracket Notation

Description

The second way to access the properties of an object is bracket notation ([]). If the property of the object you are trying to access has a space in its name, you will need to use bracket notation. However, you can still use bracket notation on object properties without spaces. Here is a sample of using bracket notation to read an object's property:

```
var myObj = {
  "Space Name": "Kirk",
  "More Space": "Spock",
  "NoSpace": "USS Enterprise"
};
myObj["Space Name"]; // Kirk
myObj['More Space']; // Spock
myObj["NoSpace"]; // USS Enterprise
```

Note that property names with spaces in them must be in quotes (single or double).

Instructions

Read the values of the properties "an entree" and "the drink" of testObj using bracket notation and assign them to entreeValue and drinkValue respectively.

Challenge Seed

```
// Setup
var testObj = {
  "an entree": "hamburger",
  "my side": "veggies",
  "the drink": "water"
};

// Only change code below this line

var entreeValue = testObj;    // Change this line
var drinkValue = testObj;    // Change this line
```

After Test

```
(function(a,b) { return "entreeValue = '" + a + "'", drinkValue = '" + b + "'"; })
(entreeValue,drinkValue);
```

Solution

```
var testObj = {
  "an entree": "hamburger",
  "my side": "veggies",
  "the drink": "water"
};
var entreeValue = testObj["an entree"];
var drinkValue = testObj['the drink'];
```

83. Accessing Object Properties with Variables

Description

Another use of bracket notation on objects is to access a property which is stored as the value of a variable. This can be very useful for iterating through an object's properties or when accessing a lookup table. Here is an example of using a variable to access a property:

```
var dogs = {
  Fido: "Mutt", Hunter: "Doberman", Snoopie: "Beagle"
};
var myDog = "Hunter";
var myBreed = dogs[myDog];
console.log(myBreed); // "Doberman"
```

Another way you can use this concept is when the property's name is collected dynamically during the program execution, as follows:

```
var someObj = {
  propName: "John"
};
function propPrefix(str) {
  var s = "prop";
  return s + str;
}
var someProp = propPrefix("Name"); // someProp now holds the value 'propName'
console.log(someObj[someProp]); // "John"
```

Note that we do *not* use quotes around the variable name when using it to access the property because we are using the *value* of the variable, not the *name*.

Instructions

Use the `playerNumber` variable to look up player 16 in `testObj` using bracket notation. Then assign that name to the `player` variable.

Challenge Seed

```
// Setup
var testObj = {
  12: "Namath",
  16: "Montana",
  19: "Unitas"
};

// Only change code below this line;
```

```
var playerNumber;      // Change this Line
var player = testObj; // Change this Line
```

After Test

```
if(typeof player !== "undefined"){(function(v){return v;})(player);}
```

Solution

```
var testObj = {
  12: "Namath",
  16: "Montana",
  19: "Unitas"
};
var playerNumber = 16;
var player = testObj[playerNumber];
```

84. Updating Object Properties

Description

After you've created a JavaScript object, you can update its properties at any time just like you would update any other variable. You can use either dot or bracket notation to update. For example, let's look at `ourDog` :

```
var ourDog = {
  "name": "Camper",
  "legs": 4,
  "tails": 1,
  "friends": ["everything!"]
};
```

Since he's a particularly happy dog, let's change his name to "Happy Camper". Here's how we update his object's name property: `ourDog.name = "Happy Camper";` or `ourDog["name"] = "Happy Camper";` Now when we evaluate `ourDog.name` , instead of getting "Camper", we'll get his new name, "Happy Camper".

Instructions

Update the `myDog` object's name property. Let's change her name from "Coder" to "Happy Coder". You can use either dot or bracket notation.

Challenge Seed

```
// Example
var ourDog = {
  "name": "Camper",
  "legs": 4,
  "tails": 1,
  "friends": ["everything!"]
};

ourDog.name = "Happy Camper";

// Setup
var myDog = {
  "name": "Coder",
  "legs": 4,
  "tails": 1,
  "friends": ["freeCodeCamp Campers"]
};

// Only change code below this line.
```

After Test

```
(function(z){return z;})(myDog);
```

Solution

```
var myDog = {
  "name": "Coder",
  "legs": 4,
  "tails": 1,
  "friends": ["freeCodeCamp Campers"]
};
myDog.name = "Happy Coder";
```

85. Add New Properties to a JavaScript Object

Description

You can add new properties to existing JavaScript objects the same way you would modify them. Here's how we would add a "bark" property to ourDog : ourDog.bark = "bow-wow"; or ourDog["bark"] = "bow-wow"; Now when we evaluate ourDog.bark , we'll get his bark, "bow-wow".

Instructions

Add a "bark" property to myDog and set it to a dog sound, such as "woof". You may use either dot or bracket notation.

Challenge Seed

```
// Example
var ourDog = {
  "name": "Camper",
  "legs": 4,
  "tails": 1,
  "friends": ["everything!"]
};

ourDog.bark = "bow-wow";

// Setup
var myDog = {
  "name": "Happy Coder",
  "legs": 4,
  "tails": 1,
  "friends": ["freeCodeCamp Campers"]
};

// Only change code below this line.
```

After Test

```
(function(z){return z;})(myDog);
```

Solution

```
var myDog = {
  "name": "Happy Coder",
```

```

    "legs": 4,
    "tails": 1,
    "friends": ["freeCodeCamp Campers"]
};

myDog.bark = "Woof Woof";

```

86. Delete Properties from a JavaScript Object

Description

We can also delete properties from objects like this: `delete ourDog.bark;`

Instructions

Delete the "tails" property from `myDog`. You may use either dot or bracket notation.

Challenge Seed

```

// Example
var ourDog = {
  "name": "Camper",
  "legs": 4,
  "tails": 1,
  "friends": ["everything!"],
  "bark": "bow-wow"
};

delete ourDog.bark;

// Setup
var myDog = {
  "name": "Happy Coder",
  "legs": 4,
  "tails": 1,
  "friends": ["freeCodeCamp Campers"],
  "bark": "woof"
};

// Only change code below this line.

```

After Test

```
(function(z){return z;})(myDog);
```

Solution

```

var ourDog = {
  "name": "Camper",
  "legs": 4,
  "tails": 1,
  "friends": ["everything!"],
  "bark": "bow-wow"
};

var myDog = {
  "name": "Happy Coder",
  "legs": 4,
  "tails": 1,
  "friends": ["freeCodeCamp Campers"],
  "bark": "woof"
};

delete myDog.tails;

```

87. Using Objects for Lookups

Description

Objects can be thought of as a key/value storage, like a dictionary. If you have tabular data, you can use an object to "lookup" values rather than a `switch` statement or an `if/else` chain. This is most useful when you know that your input data is limited to a certain range. Here is an example of a simple reverse alphabet lookup:

```
var alpha = {
  1:"Z",
  2:"Y",
  3:"X",
  4:"W",
  ...
  24:"C",
  25:"B",
  26:"A"
};

alpha[2]; // "Y"
alpha[24]; // "C"

var value = 2;
alpha[value]; // "Y"
```

Instructions

Convert the switch statement into an object called `lookup`. Use it to look up `val` and assign the associated string to the `result` variable.

Challenge Seed

```
// Setup
function phoneticLookup(val) {
  var result = "";

  // Only change code below this line
  switch(val) {
    case "alpha":
      result = "Adams";
      break;
    case "bravo":
      result = "Boston";
      break;
    case "charlie":
      result = "Chicago";
      break;
    case "delta":
      result = "Denver";
      break;
    case "echo":
      result = "Easy";
      break;
    case "foxtrot":
      result = "Frank";
  }

  // Only change code above this line
  return result;
}

// Change this value to test
phoneticLookup("charlie");
```

Solution

```

function phoneticLookup(val) {
  var result = "";

  var lookup = {
    alpha: "Adams",
    bravo: "Boston",
    charlie: "Chicago",
    delta: "Denver",
    echo: "Easy",
    foxtrot: "Frank"
  };

  result = lookup[val];

  return result;
}

```

88. Testing Objects for Properties

Description

Sometimes it is useful to check if the property of a given object exists or not. We can use the `.hasOwnProperty(propname)` method of objects to determine if that object has the given property name. `.hasOwnProperty()` returns true or false if the property is found or not. **Example**

```

var myObj = {
  top: "hat",
  bottom: "pants"
};
myObj.hasOwnProperty("top"); // true
myObj.hasOwnProperty("middle"); // false

```

Instructions

Modify the function `checkObj` to test `myObj` for `checkProp`. If the property is found, return that property's value. If not, return "Not Found".

Challenge Seed

```

// Setup
var myObj = {
  gift: "pony",
  pet: "kitten",
  bed: "sleigh"
};

function checkObj(checkProp) {
  // Your Code Here

  return "Change Me!";
}

// Test your code by modifying these values
checkObj("gift");

```

Solution

```

var myObj = {
  gift: "pony",
  pet: "kitten",
  bed: "sleigh"
};

```

```
function checkObj(checkProp) {
  if(myObj.hasOwnProperty(checkProp)) {
    return myObj[checkProp];
  } else {
    return "Not Found";
  }
}
```

89. Manipulating Complex Objects

Description

Sometimes you may want to store data in a flexible Data Structure. A JavaScript object is one way to handle flexible data. They allow for arbitrary combinations of strings, numbers, booleans, arrays, functions, and objects. Here's an example of a complex data structure:

```
var ourMusic = [
  {
    "artist": "Daft Punk",
    "title": "Homework",
    "release_year": 1997,
    "formats": [
      "CD",
      "Cassette",
      "LP"
    ],
    "gold": true
  }
];
```

This is an array which contains one object inside. The object has various pieces of metadata about an album. It also has a nested "formats" array. If you want to add more album records, you can do this by adding records to the top level array. Objects hold data in a property, which has a key-value format. In the example above, "artist": "Daft Punk" is a property that has a key of "artist" and a value of "Daft Punk". [JavaScript Object Notation](#) or JSON is a related data interchange format used to store data.

```
{
  "artist": "Daft Punk",
  "title": "Homework",
  "release_year": 1997,
  "formats": [
    "CD",
    "Cassette",
    "LP"
  ],
  "gold": true
}
```

Note

You will need to place a comma after every object in the array, unless it is the last object in the array.

Instructions

Add a new album to the `myMusic` array. Add `artist` and `title` strings, `release_year` number, and a `formats` array of strings.

Challenge Seed

```
var myMusic = [
  {
    "artist": "Billy Joel",
    "title": "Piano Man",
```

```

    "release_year": 1973,
    "formats": [
      "CD",
      "8T",
      "LP"
    ],
    "gold": true
  }
  // Add record here
];

```

After Test

```
(function(x){ if (Array.isArray(x)) { return JSON.stringify(x); } return "myMusic is not an array"})
(myMusic);
```

Solution

```

var myMusic = [
{
  "artist": "Billy Joel",
  "title": "Piano Man",
  "release_year": 1973,
  "formats": [
    "CS",
    "8T",
    "LP" ],
  "gold": true
},
{
  "artist": "ABBA",
  "title": "Ring Ring",
  "release_year": 1973,
  "formats": [
    "CS",
    "8T",
    "LP",
    "CD",
  ]
}
];

```

90. Accessing Nested Objects

Description

The sub-properties of objects can be accessed by chaining together the dot or bracket notation. Here is a nested object:

```

var ourStorage = {
  "desk": {
    "drawer": "stapler"
  },
  "cabinet": {
    "top drawer": {
      "folder1": "a file",
      "folder2": "secrets"
    },
    "bottom drawer": "soda"
  }
};
ourStorage.cabinet["top drawer"].folder2; // "secrets"
ourStorage.desk.drawer; // "stapler"

```

Instructions

Access the `myStorage` object and assign the contents of the `glove box` property to the `gloveBoxContents` variable. Use bracket notation for properties with a space in their name.

Challenge Seed

```
// Setup
var myStorage = {
  "car": {
    "inside": {
      "glove box": "maps",
      "passenger seat": "crumbs"
    },
    "outside": {
      "trunk": "jack"
    }
  }
};

var gloveBoxContents = undefined; // Change this line
```

After Test

```
(function(x) {
  if(typeof x != 'undefined') {
    return "gloveBoxContents = " + x;
  }
  return "gloveBoxContents is undefined";
})(gloveBoxContents);
```

Solution

```
var myStorage = {
  "car": {
    "inside": {
      "glove box": "maps",
      "passenger seat": "crumbs"
    },
    "outside": {
      "trunk": "jack"
    }
  }
};
var gloveBoxContents = myStorage.car.inside["glove box"];
```

91. Accessing Nested Arrays

Description

As we have seen in earlier examples, objects can contain both nested objects and nested arrays. Similar to accessing nested objects, Array bracket notation can be chained to access nested arrays. Here is an example of how to access a nested array:

```
var ourPets = [
  {
    animalType: "cat",
    names: [
      "Meowzer",
      "Fluffy",
      "Kit-Cat"
    ]
}
```

```

},
{
  animalType: "dog",
  names: [
    "Spot",
    "Bowser",
    "Frankie"
  ]
}
];
ourPets[0].names[1]; // "Fluffy"
ourPets[1].names[0]; // "Spot"

```

Instructions

Retrieve the second tree from the variable `myPlants` using object dot and array bracket notation.

Challenge Seed

```

// Setup
var myPlants = [
  {
    type: "flowers",
    list: [
      "rose",
      "tulip",
      "dandelion"
    ]
  },
  {
    type: "trees",
    list: [
      "fir",
      "pine",
      "birch"
    ]
  }
];

// Only change code below this line

var secondTree = ""; // Change this line

```

After Test

```

(function(x) {
  if(typeof x != 'undefined') {
    return "secondTree = " + x;
  }
  return "secondTree is undefined";
})(secondTree);

```

Solution

```

var myPlants = [
  {
    type: "flowers",
    list: [
      "rose",
      "tulip",
      "dandelion"
    ]
  },
  {
    type: "trees",
    list: [

```

```

        "fir",
        "pine",
        "birch"
    ]
}
];

// Only change code below this line

var secondTree = myPlants[1].list[1];

```

92. Record Collection

Description

You are given a JSON object representing a part of your musical album collection. Each album has several properties and a unique id number as its key. Not all albums have complete information. Write a function which takes an album's id (like 2548), a property prop (like "artist" or "tracks"), and a value (like "Addicted to Love") to modify the data in this collection. If prop isn't "tracks" and value isn't empty (" "), update or set the value for that record album's property. Your function must always return the entire collection object. There are several rules for handling incomplete data: If prop is "tracks" but the album doesn't have a "tracks" property, create an empty array before adding the new value to the album's corresponding property. If prop is "tracks" and value isn't empty (" "), push the value onto the end of the album's existing tracks array. If value is empty (" "), delete the given prop property from the album. Hints

Use bracket notation when [accessing object properties with variables](#). Push is an array method you can read about on [Mozilla Developer Network](#). You may refer back to [Manipulating Complex Objects](#) Introducing JavaScript Object Notation (JSON) for a refresher.

Instructions

Challenge Seed

```

// Setup
var collection = {
  "2548": {
    "album": "Slippery When Wet",
    "artist": "Bon Jovi",
    "tracks": [
      "Let It Rock",
      "You Give Love a Bad Name"
    ]
  },
  "2468": {
    "album": "1999",
    "artist": "Prince",
    "tracks": [
      "1999",
      "Little Red Corvette"
    ]
  },
  "1245": {
    "artist": "Robert Palmer",
    "tracks": [ ]
  },
  "5439": {
    "album": "ABBA Gold"
  }
};

// Keep a copy of the collection for tests
var collectionCopy = JSON.parse(JSON.stringify(collection));

// Only change code below this line
function updateRecords(id, prop, value) {

```

```

    return collection;
}

// Alter values below to test your code
updateRecords(5439, "artist", "ABBA");

```

After Test

```
; (function(x) { return "collection = \n" + JSON.stringify(x, '\n', 2); })(collection);
```

Solution

```

var collection = {
  2548: {
    album: "Slippery When Wet",
    artist: "Bon Jovi",
    tracks: [
      "Let It Rock",
      "You Give Love a Bad Name"
    ],
  },
  2468: {
    album: "1999",
    artist: "Prince",
    tracks: [
      "1999",
      "Little Red Corvette"
    ],
  },
  1245: {
    artist: "Robert Palmer",
    tracks: [ ]
  },
  5439: {
    album: "ABBA Gold"
  }
};
// Keep a copy of the collection for tests
var collectionCopy = JSON.parse(JSON.stringify(collection));

// Only change code below this line
function updateRecords(id, prop, value) {
  if(value === "") delete collection[id][prop];
  else if(prop === "tracks") {
    collection[id][prop] = collection[id][prop] || [];
    collection[id][prop].push(value);
  } else {
    collection[id][prop] = value;
  }

  return collection;
}

```

93. Iterate with JavaScript While Loops

Description

You can run the same code multiple times by using a loop. The first type of loop we will learn is called a "while" loop because it runs "while" a specified condition is true and stops once that condition is no longer true.

```

var ourArray = [];
var i = 0;
while(i < 5) {
  ourArray.push(i);
  i++;
}

```

Let's try getting a while loop to work by pushing values to an array.

Instructions

Push the numbers 0 through 4 to `myArray` using a `while` loop.

Challenge Seed

```
// Setup
var myArray = [];

// Only change code below this line.
```

After Test

```
if(typeof myArray !== "undefined"){{function(){return myArray;}}()();}
```

Solution

```
var myArray = [];
var i = 0;
while(i < 5) {
  myArray.push(i);
  i++;
}
```

94. Iterate with JavaScript For Loops

Description

You can run the same code multiple times by using a loop. The most common type of JavaScript loop is called a "for loop" because it runs "for" a specific number of times. For loops are declared with three optional expressions separated by semicolons: `for ([initialization]; [condition]; [final-expression])` The `initialization` statement is executed one time only before the loop starts. It is typically used to define and setup your loop variable. The `condition` statement is evaluated at the beginning of every loop iteration and will continue as long as it evaluates to `true`. When `condition` is `false` at the start of the iteration, the loop will stop executing. This means if `condition` starts as `false`, your loop will never execute. The `final-expression` is executed at the end of each loop iteration, prior to the next `condition` check and is usually used to increment or decrement your loop counter. In the following example we initialize with `i = 0` and iterate while our condition `i < 5` is true. We'll increment `i` by 1 in each loop iteration with `i++` as our `final-expression`.

```
var ourArray = [];
for (var i = 0; i < 5; i++) {
  ourArray.push(i);
}
```

`ourArray` will now contain `[0,1,2,3,4]`.

Instructions

Use a `for` loop to work to push the values 1 through 5 onto `myArray`.

Challenge Seed

```
// Example
var ourArray = [];
```

```

for (var i = 0; i < 5; i++) {
  ourArray.push(i);
}

// Setup
var myArray = [];

// Only change code below this line.

```

After Test

```
if (typeof myArray !== "undefined") { (function() { return myArray; })(); }
```

Solution

```

var ourArray = [];
for (var i = 0; i < 5; i++) {
  ourArray.push(i);
}
var myArray = [];
for (var i = 1; i < 6; i++) {
  myArray.push(i);
}

```

95. Iterate Odd Numbers With a For Loop

Description

For loops don't have to iterate one at a time. By changing our `final-expression`, we can count by even numbers. We'll start at `i = 0` and loop while `i < 10`. We'll increment `i` by 2 each loop with `i += 2`.

```

var ourArray = [];
for (var i = 0; i < 10; i += 2) {
  ourArray.push(i);
}

```

`ourArray` will now contain `[0,2,4,6,8]`. Let's change our `initialization` so we can count by odd numbers.

Instructions

Push the odd numbers from 1 through 9 to `myArray` using a `for` loop.

Challenge Seed

```

// Example
var ourArray = [];

for (var i = 0; i < 10; i += 2) {
  ourArray.push(i);
}

// Setup
var myArray = [];

// Only change code below this line.

```

After Test

```
if(typeof myArray !== "undefined"){{function(){return myArray;}}()}();}
```

Solution

```
var ourArray = [];
for (var i = 0; i < 10; i += 2) {
  ourArray.push(i);
}
var myArray = [];
for (var i = 1; i < 10; i += 2) {
  myArray.push(i);
}
```

96. Count Backwards With a For Loop

Description

A for loop can also count backwards, so long as we can define the right conditions. In order to count backwards by twos, we'll need to change our initialization, condition, and final-expression. We'll start at `i = 10` and loop while `i > 0`. We'll decrement `i` by 2 each loop with `i -= 2`.

```
var ourArray = [];
for (var i=10; i>0; i-=2) {
  ourArray.push(i);
}
```

`ourArray` will now contain `[10,8,6,4,2]`. Let's change our initialization and final-expression so we can count backward by twos by odd numbers.

Instructions

Push the odd numbers from 9 through 1 to `myArray` using a `for` loop.

Challenge Seed

```
// Example
var ourArray = [];

for (var i = 10; i > 0; i -= 2) {
  ourArray.push(i);
}

// Setup
var myArray = [];

// Only change code below this line.
```

After Test

```
if(typeof myArray !== "undefined"){{function(){return myArray;}}()}();}
```

Solution

```
var ourArray = [];
for (var i = 10; i > 0; i -= 2) {
  ourArray.push(i);
}
var myArray = [];
```

```
for (var i = 9; i > 0; i -= 2) {
    myArray.push(i);
}
```

97. Iterate Through an Array with a For Loop

Description

A common task in JavaScript is to iterate through the contents of an array. One way to do that is with a `for` loop. This code will output each element of the array `arr` to the console:

```
var arr = [10,9,8,7,6];
for (var i = 0; i < arr.length; i++) {
    console.log(arr[i]);
}
```

Remember that Arrays have zero-based numbering, which means the last index of the array is `length - 1`. Our condition for this loop is `i < arr.length`, which stops when `i` is at `length - 1`.

Instructions

Declare and initialize a variable `total` to `0`. Use a `for` loop to add the value of each element of the `myArr` array to `total`.

Challenge Seed

```
// Example
var ourArr = [ 9, 10, 11, 12];
var ourTotal = 0;

for (var i = 0; i < ourArr.length; i++) {
    ourTotal += ourArr[i];
}

// Setup
var myArr = [ 2, 3, 4, 5, 6];

// Only change code below this line
```

After Test

```
(function(){if(typeof total !== 'undefined') { return "total = " + total; } else { return "total is undefined";}})()
```

Solution

```
var ourArr = [ 9, 10, 11, 12];
var ourTotal = 0;

for (var i = 0; i < ourArr.length; i++) {
    ourTotal += ourArr[i];
}

var myArr = [ 2, 3, 4, 5, 6];
var total = 0;

for (var i = 0; i < myArr.length; i++) {
    total += myArr[i];
}
```

98. Nesting For Loops

Description

If you have a multi-dimensional array, you can use the same logic as the prior waypoint to loop through both the array and any sub-arrays. Here is an example:

```
var arr = [
  [1,2], [3,4], [5,6]
];
for (var i=0; i < arr.length; i++) {
  for (var j=0; j < arr[i].length; j++) {
    console.log(arr[i][j]);
  }
}
```

This outputs each sub-element in `arr` one at a time. Note that for the inner loop, we are checking the `.length` of `arr[i]`, since `arr[i]` is itself an array.

Instructions

Modify function `multiplyAll` so that it multiplies the `product` variable by each number in the sub-arrays of `arr`

Challenge Seed

```
function multiplyAll(arr) {
  var product = 1;
  // Only change code below this line

  // Only change code above this line
  return product;
}

// Modify values below to test your code
multiplyAll([[1,2],[3,4],[5,6,7]]);
```

Solution

```
function multiplyAll(arr) {
  var product = 1;
  for (var i = 0; i < arr.length; i++) {
    for (var j = 0; j < arr[i].length; j++) {
      product *= arr[i][j];
    }
  }
  return product;
}

multiplyAll([[1,2],[3,4],[5,6,7]]);
```

99. Iterate with JavaScript Do...While Loops

Description

You can run the same code multiple times by using a loop. The next type of loop you will learn is called a "do...while" loop because it first will "do" one pass of the code inside the loop no matter what, and then it runs "while" a specified condition is true and stops once that condition is no longer true. Let's look at an example.

```
var ourArray = [];
var i = 0;
```

```
do {
  ourArray.push(i);
  i++;
} while (i < 5);
```

This behaves just as you would expect with any other type of loop, and the resulting array will look like [0, 1, 2, 3, 4] . However, what makes the do...while different from other loops is how it behaves when the condition fails on the first check. Let's see this in action. Here is a regular while loop that will run the code in the loop as long as i < 5 .

```
var ourArray = [];
var i = 5;
while (i < 5) {
  ourArray.push(i);
  i++;
}
```

Notice that we initialize the value of i to be 5. When we execute the next line, we notice that i is not less than 5. So we do not execute the code inside the loop. The result is that ourArray will end up with nothing added to it, so it will still look like this [] when all the code in the example above finishes running. Now, take a look at a do...while loop.

```
var ourArray = [];
var i = 5;
do {
  ourArray.push(i);
  i++;
} while (i < 5);
```

In this case, we initialize the value of i as 5, just like we did with the while loop. When we get to the next line, there is no check for the value of i , so we go to the code inside the curly braces and execute it. We will add one element to the array and increment i before we get to the condition check. Then, when we get to checking if i < 5 see that i is now 6, which fails the conditional check. So we exit the loop and are done. At the end of the above example, the value of ourArray is [5] . Essentially, a do...while loop ensures that the code inside the loop will run at least once. Let's try getting a do...while loop to work by pushing values to an array.

Instructions

Change the while loop in the code to a do...while loop so that the loop will only push the number 10 to myArray , and i will be equal to 11 when your code finishes running.

Challenge Seed

```
// Setup
var myArray = [];
var i = 10;

// Only change code below this line.

while (i < 5) {
  myArray.push(i);
  i++;
}
```

After Test

```
if(typeof myArray !== "undefined"){{function(){return myArray;}}()();}
```

Solution

```
var myArray = [];
var i = 10;
do {
  myArray.push(i);
  i++;
} while (i < 5)
```

100. Profile Lookup

Description

We have an array of objects representing different people in our contacts lists. A `lookUpProfile` function that takes `name` and a property (`prop`) as arguments has been pre-written for you. The function should check if `name` is an actual contact's `firstName` and the given property (`prop`) is a property of that contact. If both are true, then return the "value" of that property. If `name` does not correspond to any contacts then return "No such contact". If `prop` does not correspond to any valid properties of a contact found to match `name` then return "No such property".

Instructions

Challenge Seed

```
//Setup
var contacts = [
  {
    "firstName": "Akira",
    "lastName": "Laine",
    "number": "0543236543",
    "likes": ["Pizza", "Coding", "Brownie Points"]
  },
  {
    "firstName": "Harry",
    "lastName": "Potter",
    "number": "0994372684",
    "likes": ["Hogwarts", "Magic", "Hagrid"]
  },
  {
    "firstName": "Sherlock",
    "lastName": "Holmes",
    "number": "0487345643",
    "likes": ["Intriguing Cases", "Violin"]
  },
  {
    "firstName": "Kristian",
    "lastName": "Vos",
    "number": "unknown",
    "likes": ["JavaScript", "Gaming", "Foxes"]
  }
];

function lookUpProfile(name, prop){
  // Only change code below this line

  // Only change code above this line
}

// Change these values to test your function
lookUpProfile("Akira", "likes");
```

Solution

```
var contacts = [
  {
    "firstName": "Akira",
    "lastName": "Laine",
    "number": "0543236543",
    "likes": ["Pizza", "Coding", "Brownie Points"]
  },
  {
    "firstName": "Harry",
    "lastName": "Potter",
```

```

        "number": "0994372684",
        "likes": ["Hogwarts", "Magic", "Hagrid"]
    },
    {
        "firstName": "Sherlock",
        "lastName": "Holmes",
        "number": "0487345643",
        "likes": ["Intriguing Cases", "Violin"]
    },
    {
        "firstName": "Kristian",
        "lastName": "Vos",
        "number": "unknown",
        "likes": ["JavaScript", "Gaming", "Foxes"]
    },
];
//Write your function in between these comments
function lookUpProfile(name, prop){
    for(var i in contacts){
        if(contacts[i].firstName === name) {
            return contacts[i][prop] || "No such property";
        }
    }
    return "No such contact";
}
//Write your function in between these comments
lookUpProfile("Akira", "likes");

```

101. Generate Random Fractions with JavaScript

Description

Random numbers are useful for creating random behavior. JavaScript has a `Math.random()` function that generates a random decimal number between `0` (inclusive) and not quite up to `1` (exclusive). Thus `Math.random()` can return a `0` but never quite return a `1`. **Note**

Like [Storing Values with the Equal Operator](#), all function calls will be resolved before the `return` executes, so we can return the value of the `Math.random()` function.

Instructions

Change `randomFraction` to return a random number instead of returning `0`.

Challenge Seed

```

function randomFraction() {

    // Only change code below this line.

    return 0;

    // Only change code above this line.
}

```

After Test

```
(function(){return randomFraction();})();
```

Solution

```
function randomFraction() {
  return Math.random();
}
```

102. Generate Random Whole Numbers with JavaScript

Description

It's great that we can generate random decimal numbers, but it's even more useful if we use it to generate random whole numbers.

1. Use `Math.random()` to generate a random decimal.
2. Multiply that random decimal by `20`.
3. Use another function, `Math.floor()` to round the number down to its nearest whole number.

Remember that `Math.random()` can never quite return a `1` and, because we're rounding down, it's impossible to actually get `20`. This technique will give us a whole number between `0` and `19`. Putting everything together, this is what our code looks like: `Math.floor(Math.random() * 20)`; We are calling `Math.random()`, multiplying the result by `20`, then passing the value to `Math.floor()` function to round the value down to the nearest whole number.

Instructions

Use this technique to generate and return a random whole number between `0` and `9`.

Challenge Seed

```
var randomNumberBetween0and19 = Math.floor(Math.random() * 20);

function randomWholeNum() {

  // Only change code below this line.

  return Math.random();
}
```

After Test

```
(function(){return randomWholeNum();})();
```

Solution

```
var randomNumberBetween0and19 = Math.floor(Math.random() * 20);
function randomWholeNum() {
  return Math.floor(Math.random() * 10);
}
```

103. Generate Random Whole Numbers within a Range

Description

Instead of generating a random number between zero and a given number like we did before, we can generate a random number that falls within a range of two specific numbers. To do this, we'll define a minimum number `min` and a maximum number `max`. Here's the formula we'll use. Take a moment to read it and try to understand what this code is doing: `Math.floor(Math.random() * (max - min + 1)) + min`

Instructions

Create a function called `randomRange` that takes a range `myMin` and `myMax` and returns a random number that's greater than or equal to `myMin`, and is less than or equal to `myMax`, inclusive.

Challenge Seed

```
// Example
function ourRandomRange(ourMin, ourMax) {

    return Math.floor(Math.random() * (ourMax - ourMin + 1)) + ourMin;
}

ourRandomRange(1, 9);

// Only change code below this line.

function randomRange(myMin, myMax) {

    return 0; // Change this line

}

// Change these values to test your function
var myRandom = randomRange(5, 15);
```

After Test

```
var calcMin = 100;
var calcMax = -100;
for(var i = 0; i < 100; i++) {
    var result = randomRange(5,15);
    calcMin = Math.min(calcMin, result);
    calcMax = Math.max(calcMax, result);
}
(function(){
    if(typeof myRandom === 'number') {
        return "myRandom = " + myRandom;
    } else {
        return "myRandom undefined";
    }
})()
```

Solution

```
function randomRange(myMin, myMax) {
    return Math.floor(Math.random() * (myMax - myMin + 1)) + myMin;
}
```

104. Use the parseInt Function

Description

The `parseInt()` function parses a string and returns an integer. Here's an example: `var a = parseInt("007");` The above function converts the string "007" to an integer 7. If the first character in the string can't be converted into a number, then it returns `NaN`.

Instructions

Use `parseInt()` in the `convertToInteger` function so it converts the input string `str` into an integer, and returns it.

Challenge Seed

```
function convertToInteger(str) {
}

convertToInteger("56");
```

Solution

```
function convertToInteger(str) {
  return parseInt(str);
}
```

105. Use the parseInt Function with a Radix

Description

The `parseInt()` function parses a string and returns an integer. It takes a second argument for the radix, which specifies the base of the number in the string. The radix can be an integer between 2 and 36. The function call looks like: `parseInt(string, radix);` And here's an example: `var a = parseInt("11", 2);` The radix variable says that "11" is in the binary system, or base 2. This example converts the string "11" to an integer 3.

Instructions

Use `parseInt()` in the `convertToInteger` function so it converts a binary number to an integer and returns it.

Challenge Seed

```
function convertToInteger(str) {
}

convertToInteger("10011");
```

Solution

```
function convertToInteger(str) {
  return parseInt(str, 2);
}
```

106. Use the Conditional (Ternary) Operator

Description

The conditional operator, also called the ternary operator, can be used as a one line if-else expression. The syntax is: `condition ? statement-if-true : statement-if-false;` The following function uses an if-else statement to check a condition:

```
function findGreater(a, b) {
  if(a > b) {
    return "a is greater";
  }
  else {
```

```

    return "b is greater";
}
}

```

This can be re-written using the conditional operator :

```

function findGreater(a, b) {
  return a > b ? "a is greater" : "b is greater";
}

```

Instructions

Use the conditional operator in the `checkEqual` function to check if two numbers are equal or not. The function should return either "Equal" or "Not Equal".

Challenge Seed

```

function checkEqual(a, b) {
}

checkEqual(1, 2);

```

Solution

```

function checkEqual(a, b) {
  return a === b ? "Equal" : "Not Equal";
}

```

107. Use Multiple Conditional (Ternary) Operators

Description

In the previous challenge, you used a single conditional operator . You can also chain them together to check for multiple conditions. The following function uses if, else if, and else statements to check multiple conditions:

```

function findGreaterOrEqual(a, b) {
  if(a === b) {
    return "a and b are equal";
  }
  else if(a > b) {
    return "a is greater";
  }
  else {
    return "b is greater";
  }
}

```

The above function can be re-written using multiple conditional operators :

```

function findGreaterOrEqual(a, b) {
  return (a === b) ? "a and b are equal" : (a > b) ? "a is greater" : "b is greater";
}

```

Instructions

Use multiple conditional operators in the `checkSign` function to check if a number is positive, negative or zero.

Challenge Seed

```
function checkSign(num) {
}
checkSign(10);
```

Solution

```
function checkSign(num) {
  return (num > 0) ? 'positive' : (num < 0) ? 'negative' : 'zero';
}
```

ES6

1. Explore Differences Between the var and let Keywords

Description

One of the biggest problems with declaring variables with the `var` keyword is that you can overwrite variable declarations without an error.

```
var camper = 'James';
var camper = 'David';
console.log(camper);
// logs 'David'
```

As you can see in the code above, the `camper` variable is originally declared as `James` and then overridden to be `David`. In a small application, you might not run into this type of problem, but when your code becomes larger, you might accidentally overwrite a variable that you did not intend to overwrite. Because this behavior does not throw an error, searching and fixing bugs becomes more difficult.

A new keyword called `let` was introduced in ES6 to solve this potential issue with the `var` keyword. If you were to replace `var` with `let` in the variable declarations of the code above, the result would be an error.

```
let camper = 'James';
let camper = 'David'; // throws an error
```

This error can be seen in the console of your browser. So unlike `var`, when using `let`, a variable with the same name can only be declared once. Note the `"use strict"`. This enables Strict Mode, which catches common coding mistakes and "unsafe" actions. For instance:

```
"use strict";
x = 3.14; // throws an error because x is not declared
```

Instructions

Update the code so it only uses the `let` keyword.

Challenge Seed

```
var catName;
var quote;
function catTalk() {
  "use strict";

  catName = "Oliver";
  quote = catName + " says Meow!";

}
catTalk();
```

Solution

```
let catName;
let quote;
function catTalk() {
  'use strict';

  catName = 'Oliver';
  quote = catName + ' says Meow!';
}
catTalk();
```

2. Compare Scopes of the var and let Keywords

Description

When you declare a variable with the `var` keyword, it is declared globally, or locally if declared inside a function. The `let` keyword behaves similarly, but with some extra features. When you declare a variable with the `let` keyword inside a block, statement, or expression, its scope is limited to that block, statement, or expression. For example:

```
var numArray = [];
for (var i = 0; i < 3; i++) {
  numArray.push(i);
}
console.log(numArray);
// returns [0, 1, 2]
console.log(i);
// returns 3
```

With the `var` keyword, `i` is declared globally. So when `i++` is executed, it updates the global variable. This code is similar to the following:

```
var numArray = [];
var i;
for (i = 0; i < 3; i++) {
  numArray.push(i);
}
console.log(numArray);
// returns [0, 1, 2]
console.log(i);
// returns 3
```

This behavior will cause problems if you were to create a function and store it for later use inside a for loop that uses the `i` variable. This is because the stored function will always refer to the value of the updated global `i` variable.

```
var printNumTwo;
for (var i = 0; i < 3; i++) {
  if(i === 2){
    printNumTwo = function() {
      return i;
    };
  }
  console.log(printNumTwo());
}
// returns 3
```

As you can see, `printNumTwo()` prints 3 and not 2. This is because the value assigned to `i` was updated and the `printNumTwo()` returns the global `i` and not the value `i` had when the function was created in the for loop. The `let` keyword does not follow this behavior:

```
'use strict';
let printNumTwo;
for (let i = 0; i < 3; i++) {
  if (i === 2) {
```

```
printNumTwo = function() {
  return i;
}
}
console.log(printNumTwo());
// returns 2
console.log(i);
// returns "i is not defined"
```

`i` is not defined because it was not declared in the global scope. It is only declared within the for loop statement. `printNumTwo()` returned the correct value because three different `i` variables with unique values (0, 1, and 2) were created by the `let` keyword within the loop statement.

Instructions

Fix the code so that `i` declared in the if statement is a separate variable than `i` declared in the first line of the function. Be certain not to use the `var` keyword anywhere in your code. This exercise is designed to illustrate the difference between how `var` and `let` keywords assign scope to the declared variable. When programming a function similar to the one used in this exercise, it is often better to use different variable names to avoid confusion.

Challenge Seed

```
function checkScope() {
  'use strict';
  var i = 'function scope';
  if (true) {
    i = 'block scope';
    console.log('Block scope i is: ', i);
  }
  console.log('Function scope i is: ', i);
  return i;
}
```

Solution

```
function checkScope() {
  'use strict';
  let i = 'function scope';
  if (true) {
    let i = 'block scope';
    console.log('Block scope i is: ', i);
  }

  console.log('Function scope i is: ', i);
  return i;
}
```

3. Declare a Read-Only Variable with the `const` Keyword

Description

The keyword `let` is not the only new way to declare variables. In ES6, you can also declare variables using the `const` keyword. `const` has all the awesome features that `let` has, with the added bonus that variables declared using `const` are read-only. They are a constant value, which means that once a variable is assigned with `const`, it cannot be reassigned.

```
"use strict"
const FAV_PET = "Cats";
FAV_PET = "Dogs"; // returns error
```

As you can see, trying to reassign a variable declared with `const` will throw an error. You should always name variables you don't want to reassign using the `const` keyword. This helps when you accidentally attempt to reassign a variable that is meant to stay constant. A common practice when naming constants is to use all uppercase letters, with words separated by an underscore.

Note: It is common for developers to use uppercase variable identifiers for immutable values and lowercase or camelCase for mutable values (objects and arrays). In a later challenge you will see an example of a lowercase variable identifier being used for an array.

Instructions

Change the code so that all variables are declared using `let` or `const`. Use `let` when you want the variable to change, and `const` when you want the variable to remain constant. Also, rename variables declared with `const` to conform to common practices, meaning constants should be in all caps.

Challenge Seed

```
function printManyTimes(str) {
  "use strict";

  // change code below this line

  var sentence = str + " is cool!";
  for(var i = 0; i < str.length; i+=2) {
    console.log(sentence);
  }

  // change code above this line

}

printManyTimes("freeCodeCamp");
```

Solution

```
function printManyTimes(str) {
  "use strict";

  // change code below this line

  const SENTENCE = str + " is cool!";
  for(let i = 0; i < str.length; i+=2) {
    console.log(SENTENCE);
  }

  // change code above this line

}

printManyTimes("freeCodeCamp");
```

4. Mutate an Array Declared with `const`

Description

The `const` declaration has many use cases in modern JavaScript. Some developers prefer to assign all their variables using `const` by default, unless they know they will need to reassign the value. Only in that case, they use `let`. However, it is important to understand that objects (including arrays and functions) assigned to a variable using `const` are still mutable. Using the `const` declaration only prevents reassignment of the variable identifier.

```
"use strict";
const s = [5, 6, 7];
s = [1, 2, 3]; // throws error, trying to assign a const
s[2] = 45; // works just as it would with an array declared with var or let
console.log(s); // returns [5, 6, 45]
```

As you can see, you can mutate the object `[5, 6, 7]` itself and the variable `s` will still point to the altered array `[5, 6, 45]`. Like all arrays, the array elements in `s` are mutable, but because `const` was used, you cannot use the variable identifier `s` to point to a different array using the assignment operator.

Instructions

An array is declared as `const s = [5, 7, 2]`. Change the array to `[2, 5, 7]` using various element assignment.

Challenge Seed

```
const s = [5, 7, 2];
function editInPlace() {
  'use strict';
  // change code below this line

  // s = [2, 5, 7]; <- this is invalid

  // change code above this line
}
editInPlace();
```

Solution

```
const s = [5, 7, 2];
function editInPlace() {
  'use strict';
  // change code below this line

  // s = [2, 5, 7]; <- this is invalid
  s[0] = 2;
  s[1] = 5;
  s[2] = 7;
  // change code above this line
}
editInPlace();
```

5. Prevent Object Mutation

Description

As seen in the previous challenge, `const` declaration alone doesn't really protect your data from mutation. To ensure your data doesn't change, JavaScript provides a function `Object.freeze` to prevent data mutation. Once the object is frozen, you can no longer add, update, or delete properties from it. Any attempt at changing the object will be rejected without an error.

```
let obj = {
  name:"FreeCodeCamp",
  review:"Awesome"
};
Object.freeze(obj);
obj.review = "bad"; //will be ignored. Mutation not allowed
obj.newProp = "Test"; // will be ignored. Mutation not allowed
console.log(obj);
// { name: "FreeCodeCamp", review:"Awesome"}
```

Instructions

In this challenge you are going to use `Object.freeze` to prevent mathematical constants from changing. You need to freeze the `MATH_CONSTANTS` object so that no one is able to alter the value of `PI`, add, or delete properties.

Challenge Seed

```
function freezeObj() {
  'use strict';
  const MATH_CONSTANTS = {
    PI: 3.14
  };
  // change code below this line

  // change code above this line
  try {
    MATH_CONSTANTS.PI = 99;
  } catch( ex ) {
    console.log(ex);
  }
  return MATH_CONSTANTS.PI;
}
const PI = freezeObj();
```

Solution

```
function freezeObj() {
  'use strict';
  const MATH_CONSTANTS = {
    PI: 3.14
  };
  // change code below this line
  Object.freeze(MATH_CONSTANTS);

  // change code above this line
  try {
    MATH_CONSTANTS.PI = 99;
  } catch( ex ) {
    console.log(ex);
  }
  return MATH_CONSTANTS.PI;
}
const PI = freezeObj();
```

6. Use Arrow Functions to Write Concise Anonymous Functions

Description

In JavaScript, we often don't need to name our functions, especially when passing a function as an argument to another function. Instead, we create inline functions. We don't need to name these functions because we do not reuse them anywhere else. To achieve this, we often use the following syntax:

```
const myFunc = function() {
  const myVar = "value";
  return myVar;
}
```

ES6 provides us with the syntactic sugar to not have to write anonymous functions this way. Instead, you can use **arrow function syntax**:

```
const myFunc = () => {
  const myVar = "value";
  return myVar;
}
```

When there is no function body, and only a return value, arrow function syntax allows you to omit the keyword `return` as well as the brackets surrounding the code. This helps simplify smaller functions into one-line statements:

```
const myFunc = () => "value"
```

This code will still return `value` by default.

Instructions

Rewrite the function assigned to the variable `magic` which returns a new `Date()` to use arrow function syntax. Also make sure nothing is defined using the keyword `var`.

Challenge Seed

```
var magic = function() {
  "use strict";
  return new Date();
};
```

Solution

```
const magic = () => {
  "use strict";
  return new Date();
};
```

7. Write Arrow Functions with Parameters

Description

Just like a regular function, you can pass arguments into an arrow function.

```
// doubles input value and returns it
const doubler = (item) => item * 2;
```

If an arrow function has a single argument, the parentheses enclosing the argument may be omitted.

```
// the same function, without the argument parentheses
const doubler = item => item * 2;
```

It is possible to pass more than one argument into an arrow function.

```
// multiplies the first input value by the second and returns it
const multiplier = (item, multi) => item * multi;
```

Instructions

Rewrite the `myConcat` function which appends contents of `arr2` to `arr1` so that the function uses arrow function syntax.

Challenge Seed

```
var myConcat = function(arr1, arr2) {
  "use strict";
  return arr1.concat(arr2);
};
// test your code
console.log(myConcat([1, 2], [3, 4, 5]));
```

Solution

```
const myConcat = (arr1, arr2) => {
  "use strict";
```

```

    return arr1.concat(arr2);
};

// test your code
console.log(myConcat([1, 2], [3, 4, 5]));

```

8. Write Higher Order Arrow Functions

Description

It's time to look at higher-order functions and their common pair, arrow functions. Arrow functions work really well when combined with higher-order functions, such as `map()`, `filter()`, and `reduce()`.

But what are these functions? Lets look at the simplest example `forEach()`, and run it on the following array of sample Facebook posts.

```

let FBPosts = [
  {thumbnail: "someIcon", likes:432, shares: 600},
  {thumbnail: "Another icon", likes:300, shares: 501},
  {thumbnail: "Yet another", likes:40, shares: 550},
  {thumbnail: null, likes: 101, shares:0},
]

```

Of the two `forEach()` versions below, both perform the exact same log function, and each takes an anonymous callback with a parameter `post`. The difference is the syntax. One uses an arrow function and the other does not.

ES5

```

FBpost.forEach(function(post) {
  console.log(post) // log each post here
});

```

ES6

```

FBpost.forEach((post) => {
  console.log(post) // log each post here
});

```

`filter()` is very similar. Below it will iterate over the `FBPosts` array, perform the logic to filter out the items that do not meet the requirements, and return a new array, `results`.

```
let results = arr1.filter((post) => { return post.thumbnail !== null && post.likes > 100 && post.shares > 500 });
```

```
console.log(results); // [{thumbnail: "someIcon", likes: 432, shares: 600}, {thumbnail: "Another icon", likes: 300, shares: 501}]
```

Instructions

Use arrow function syntax to compute the square of *only* the positive integers (decimal numbers are not integers) in the array `realNumberArray` and store the new array in the variable `squaredIntegers`.

Challenge Seed

```

const realNumberArray = [4, 5.6, -9.8, 3.14, 42, 6, 8.34, -2];
const squareList = (arr) => {
  "use strict";
  const positiveIntegers = arr.filter((num) => {
    // add code here
  });
  const squaredIntegers = positiveIntegers.map((num) => {
    // add code here
  });

  return squaredIntegers;
};
// test your code
const squaredIntegers = squareList(realNumberArray);
console.log(squaredIntegers);

```

Solution

```
const realNumberArray = [4, 5.6, -9.8, 3.14, 42, 6, 8.34, -2];
const squareList = (arr) => {
  "use strict";
  const positiveIntegers = arr.filter((num) => {
    return num >= 0 && Number.isInteger(num);
    // add code here
  });
  const squaredIntegers = positiveIntegers.map((num) => {
    // add code here
    return num ** 2;
  });
  // add code here
  return squaredIntegers;
};
// test your code
const squaredIntegers = squareList(realNumberArray);
```

9. Set Default Parameters for Your Functions

Description

In order to help us create more flexible functions, ES6 introduces default parameters for functions. Check out this code:

```
function greeting(name = "Anonymous") {
  return "Hello " + name;
}
console.log(greeting("John")); // Hello John
console.log(greeting()); // Hello Anonymous
```

The default parameter kicks in when the argument is not specified (it is undefined). As you can see in the example above, the parameter `name` will receive its default value "Anonymous" when you do not provide a value for the parameter. You can add default values for as many parameters as you want.

Instructions

Modify the function `increment` by adding default parameters so that it will add 1 to `number` if `value` is not specified.

Challenge Seed

```
const increment = (function() {
  "use strict";
  return function increment(number, value) {
    return number + value;
  };
})();
console.log(increment(5, 2)); // returns 7
console.log(increment(5)); // returns 6
```

Solution

```
// solution required
```

10. Use the Rest Operator with Function Parameters

Description

In order to help us create more flexible functions, ES6 introduces the rest operator for function parameters. With the rest operator, you can create functions that take a variable number of arguments. These arguments are stored in an array that can be accessed later from inside the function. Check out this code:

```
function howMany(...args) {
  return "You have passed " + args.length + " arguments.";
}

console.log(howMany(0, 1, 2)); // You have passed 3 arguments
console.log(howMany("string", null, [1, 2, 3], {})); // You have passed 4 arguments.
```

The rest operator eliminates the need to check the `args` array and allows us to apply `map()`, `filter()` and `reduce()` on the parameters array.

Instructions

Modify the function `sum` using the rest parameter in such a way that the function `sum` is able to take any number of arguments and return their sum.

Challenge Seed

```
const sum = (function() {
  "use strict";
  return function sum(x, y, z) {
    const args = [x, y, z];
    return args.reduce((a, b) => a + b, 0);
  };
})();
console.log(sum(1, 2, 3)); // 6
```

Solution

```
// solution required
```

11. Use the Spread Operator to Evaluate Arrays In-Place

Description

ES6 introduces the spread operator, which allows us to expand arrays and other expressions in places where multiple parameters or elements are expected. The ES5 code below uses `apply()` to compute the maximum value in an array:

```
var arr = [6, 89, 3, 45];
var maximus = Math.max.apply(null, arr); // returns 89
```

We had to use `Math.max.apply(null, arr)` because `Math.max(arr)` returns `Nan`. `Math.max()` expects comma-separated arguments, but not an array. The spread operator makes this syntax much better to read and maintain.

```
const arr = [6, 89, 3, 45];
const maximus = Math.max(...arr); // returns 89
```

`...arr` returns an unpacked array. In other words, it *spreads* the array. However, the spread operator only works in-place, like in an argument to a function or in an array literal. The following code will not work:

```
const spreaded = ...arr; // will throw a syntax error
```

Instructions

Copy all contents of `arr1` into another array `arr2` using the spread operator.

Challenge Seed

```
const arr1 = ['JAN', 'FEB', 'MAR', 'APR', 'MAY'];
let arr2;
(function() {
  "use strict";
  arr2 = []; // change this line
})();
console.log(arr2);
```

Solution

```
const arr1 = ['JAN', 'FEB', 'MAR', 'APR', 'MAY'];
let arr2;
(function() {
  "use strict";
  arr2 = [...arr1]; // change this line
})();
console.log(arr2);
```

12. Use Destructuring Assignment to Assign Variables from Objects

Description

We saw earlier how spread operator can effectively spread, or unpack, the contents of the array. We can do something similar with objects as well. Destructuring assignment is special syntax for neatly assigning values taken directly from an object to variables. Consider the following ES5 code:

```
var voxel = {x: 3.6, y: 7.4, z: 6.54};
var x = voxel.x; // x = 3.6
var y = voxel.y; // y = 7.4
var z = voxel.z; // z = 6.54
```

Here's the same assignment statement with ES6 destructuring syntax:

```
const { x, y, z } = voxel; // x = 3.6, y = 7.4, z = 6.54
```

If instead you want to store the values of `voxel.x` into `a`, `voxel.y` into `b`, and `voxel.z` into `c`, you have that freedom as well.

```
const { x : a, y : b, z : c } = voxel; // a = 3.6, b = 7.4, c = 6.54
```

You may read it as "get the field `x` and copy the value into `a`," and so on.

Instructions

Use destructuring to obtain the average temperature for tomorrow from the input object `AVG_TEMPERATURES`, and assign value with key `tomorrow` to `tempOfTomorrow` in line.

Challenge Seed

```
const AVG_TEMPERATURES = {
  today: 77.5,
  tomorrow: 79
};

function getTempOfTmrw(avgTemperatures) {
  "use strict";
  // change code below this line
  const tempOfTomorrow = undefined; // change this line
  // change code above this line
  return tempOfTomorrow;
}

console.log(getTempOfTmrw(AVG_TEMPERATURES)); // should be 79
```

Solution

```
const AVG_TEMPERATURES = {
  today: 77.5,
  tomorrow: 79
};

function getTempOfTmrw(avgTemperatures) {
  "use strict";
  // change code below this line
  const {tomorrow:tempOfTomorrow} = avgTemperatures; // change this line
  // change code above this line
  return tempOfTomorrow;
}

console.log(getTempOfTmrw(AVG_TEMPERATURES)); // should be 79
```

13. Use Destructuring Assignment to Assign Variables from Nested Objects

Description

We can similarly destructure *nested* objects into variables. Consider the following code:

```
const a = {
  start: { x: 5, y: 6 },
  end: { x: 6, y: -9 }
};
const { start: { x: startX, y: startY } } = a;
console.log(startX, startY); // 5, 6
```

In the example above, the variable `startX` is assigned the value of `a.start.x`.

Instructions

Use destructuring assignment to obtain `max` of `forecast.tomorrow` and assign it to `maxOfTomorrow`.

Challenge Seed

```
const LOCAL_FORECAST = {
  today: { min: 72, max: 83 },
  tomorrow: { min: 73.3, max: 84.6 }
};

function getMaxOfTmrw(forecast) {
  "use strict";
  // change code below this line
  const maxOfTomorrow = undefined; // change this line
  // change code above this line
  return maxOfTomorrow;
}

console.log(getMaxOfTmrw(LOCAL_FORECAST)); // should be 84.6
```

Solution

```
const LOCAL_FORECAST = {
  today: { min: 72, max: 83 },
  tomorrow: { min: 73.3, max: 84.6 }
};
```

```

function getMaxOfTmrw(forecast) {
  "use strict";
  // change code below this line
  const {tomorrow : {max : maxOfTomorrow}} = forecast; // change this line
  // change code above this line
  return maxOfTomorrow;
}

console.log(getMaxOfTmrw(LOCAL_FORECAST)); // should be 84.6

```

14. Use Destructuring Assignment to Assign Variables from Arrays

Description

ES6 makes destructuring arrays as easy as destructuring objects. One key difference between the spread operator and array destructuring is that the spread operator unpacks all contents of an array into a comma-separated list. Consequently, you cannot pick or choose which elements you want to assign to variables. Destructuring an array lets us do exactly that:

```

const [a, b] = [1, 2, 3, 4, 5, 6];
console.log(a, b); // 1, 2

```

The variable `a` is assigned the first value of the array, and `b` is assigned the second value of the array. We can also access the value at any index in an array with destructuring by using commas to reach the desired index:

```

const [a, b,,, c] = [1, 2, 3, 4, 5, 6];
console.log(a, b, c); // 1, 2, 5

```

Instructions

Use destructuring assignment to swap the values of `a` and `b` so that `a` receives the value stored in `b`, and `b` receives the value stored in `a`.

Challenge Seed

```

let a = 8, b = 6;
(() => {
  "use strict";
  // change code below this line

  // change code above this line
})();
console.log(a); // should be 6
console.log(b); // should be 8

```

Solution

```
// solution required
```

15. Use Destructuring Assignment with the Rest Operator to Reassign Array Elements

Description

In some situations involving array destructuring, we might want to collect the rest of the elements into a separate array. The result is similar to `Array.prototype.slice()`, as shown below:

```
const [a, b, ...arr] = [1, 2, 3, 4, 5, 7];
console.log(a, b); // 1, 2
console.log(arr); // [3, 4, 5, 7]
```

Variables `a` and `b` take the first and second values from the array. After that, because of rest operator's presence, `arr` gets rest of the values in the form of an array. The rest element only works correctly as the last variable in the list. As in, you cannot use the rest operator to catch a subarray that leaves out last element of the original array.

Instructions

Use destructuring assignment with the rest operator to perform an effective `Array.prototype.slice()` so that `arr` is a sub-array of the original array `source` with the first two elements omitted.

Challenge Seed

```
const source = [1,2,3,4,5,6,7,8,9,10];
function removeFirstTwo(list) {
  "use strict";
  // change code below this line
  const arr = list; // change this
  // change code above this line
  return arr;
}
const arr = removeFirstTwo(source);
console.log(arr); // should be [3,4,5,6,7,8,9,10]
console.log(source); // should be [1,2,3,4,5,6,7,8,9,10];
```

Solution

```
const source = [1,2,3,4,5,6,7,8,9,10];
function removeFirstTwo(list) {
  "use strict";
  // change code below this line
  const [, , ...arr] = list;
  // change code above this line
  return arr;
}
const arr = removeFirstTwo(source);
```

16. Use Destructuring Assignment to Pass an Object as a Function's Parameters

Description

In some cases, you can destructure the object in a function argument itself. Consider the code below:

```
const profileUpdate = (profileData) => {
  const { name, age, nationality, location } = profileData;
  // do something with these variables
}
```

This effectively destructures the object sent into the function. This can also be done in-place:

```
const profileUpdate = ({ name, age, nationality, location }) => {
  /* do something with these fields */
}
```

This removes some extra lines and makes our code look neat. This has the added benefit of not having to manipulate an entire object in a function; only the fields that are needed are copied inside the function.

Instructions

Use destructuring assignment within the argument to the function `half` to send only `max` and `min` inside the function.

Challenge Seed

```
const stats = {
  max: 56.78,
  standard_deviation: 4.34,
  median: 34.54,
  mode: 23.87,
  min: -0.75,
  average: 35.85
};
const half = (function() {
  "use strict"; // do not change this line

  // change code below this line
  return function half(stats) {
    // use function argument destructuring
    return (stats.max + stats.min) / 2.0;
  };
  // change code above this line

})();
console.log(stats); // should be object
console.log(half(stats)); // should be 28.015
```

Solution

```
// solution required
```

17. Create Strings using Template Literals

Description

A new feature of ES6 is the template literal. This is a special type of string that makes creating complex strings easier. Template literals allow you to create multi-line strings and to use string interpolation features to create strings. Consider the code below:

```
const person = {
  name: "Zodiac Hasbro",
  age: 56
};

// Template literal with multi-line and string interpolation
const greeting = `Hello, my name is ${person.name}!
I am ${person.age} years old.`;

console.log(greeting); // prints
// Hello, my name is Zodiac Hasbro!
// I am 56 years old.
```

A lot of things happened there. Firstly, the example uses backticks (```), not quotes (`'` or `"`), to wrap the string. Secondly, notice that the string is multi-line, both in the code and the output. This saves inserting `\n` within strings. The `${variable}` syntax used above is a placeholder. Basically, you won't have to use concatenation with the `+` operator anymore. To add variables to strings, you just drop the variable in a template string and wrap it with `${}` and `}`. Similarly, you can include other expressions in your string literal, for example `${a + b}`. This new way of creating strings gives you more flexibility to create robust strings.

Instructions

Use template literal syntax with backticks to display each entry of the `result` object's `failure` array. Each entry should be wrapped inside an `li` element with the class attribute `text-warning`, and listed within the `resultDisplayArray`. Use an iterator method (any kind of loop) to get the desired output.

Challenge Seed

```
const result = {
  success: ["max-length", "no-amd", "prefer-arrow-functions"],
  failure: ["no-var", "var-on-top", "linebreak"],
  skipped: ["id-blacklist", "no-dup-keys"]
};

function makeList(arr) {
  "use strict";

  // change code below this line
  const resultDisplayArray = null;
  // change code above this line

  return resultDisplayArray;
}
/**/
/* makeList(result.failure) should return:
 * [ `<li class="text-warning">no-var</li>`,
 *   `<li class="text-warning">var-on-top</li>`,
 *   `<li class="text-warning">linebreak</li>` ]
 */
const resultDisplayArray = makeList(result.failure);
```

Solution

```
const result = {
  success: ["max-length", "no-amd", "prefer-arrow-functions"],
  failure: ["no-var", "var-on-top", "linebreak"],
  skipped: ["id-blacklist", "no-dup-keys"]
};
function makeList(arr) {
  "use strict";

  const resultDisplayArray = arr.map(val => `<li class="text-warning">${val}</li>`);

  return resultDisplayArray;
}
/**/
/* makeList(result.failure) should return:
 * [ `<li class="text-warning">no-var</li>`,
 *   `<li class="text-warning">var-on-top</li>`,
 *   `<li class="text-warning">linebreak</li>` ]
 */
const resultDisplayArray = makeList(result.failure);
```

18. Write Concise Object Literal Declarations Using Simple Fields

Description

ES6 adds some nice support for easily defining object literals. Consider the following code:

```
const get.mousePosition = (x, y) => ({
  x: x,
  y: y
});
```

`getMousePosition` is a simple function that returns an object containing two fields. ES6 provides the syntactic sugar to eliminate the redundancy of having to write `x: x`. You can simply write `x` once, and it will be converted to `x: x` (or something equivalent) under the hood. Here is the same function from above rewritten to use this new syntax:

```
const getMousePosition = (x, y) => ({ x, y });
```

Instructions

Use simple fields with object literals to create and return a `Person` object with `name`, `age` and `gender` properties.

Challenge Seed

```
const createPerson = (name, age, gender) => {
  "use strict";
  // change code below this line
  return {
    name: name,
    age: age,
    gender: gender
  };
  // change code above this line
};
console.log(createPerson("Zodiac Hasbro", 56, "male")); // returns a proper object
```

Solution

```
const createPerson = (name, age, gender) => {
  "use strict";
  return {
    name,
    age,
    gender
  };
};
```

19. Write Concise Declarative Functions with ES6

Description

When defining functions within objects in ES5, we have to use the keyword `function` as follows:

```
const person = {
  name: "Taylor",
  sayHello: function() {
    return `Hello! My name is ${this.name}.`;
  }
};
```

With ES6, You can remove the `function` keyword and colon altogether when defining functions in objects. Here's an example of this syntax:

```
const person = {
  name: "Taylor",
  sayHello() {
    return `Hello! My name is ${this.name}.`;
  }
};
```

Instructions

Refactor the function `setGear` inside the object `bicycle` to use the shorthand syntax described above.

Challenge Seed

```
// change code below this line
const bicycle = {
  gear: 2,
  setGear: function(newGear) {
    this.gear = newGear;
  }
};
// change code above this line
bicycle.setGear(3);
console.log(bicycle.gear);
```

Solution

```
const bicycle = {
  gear: 2,
  setGear(newGear) {
    this.gear = newGear;
  }
};
bicycle.setGear(3);
```

20. Use class Syntax to Define a Constructor Function

Description

ES6 provides a new syntax to help create objects, using the keyword `class`. This is to be noted, that the `class` syntax is just a syntax, and not a full-fledged class based implementation of object oriented paradigm, unlike in languages like Java, or Python, or Ruby etc. In ES5, we usually define a constructor function, and use the `new` keyword to instantiate an object.

```
var SpaceShuttle = function(targetPlanet){
  this.targetPlanet = targetPlanet;
}
var zeus = new SpaceShuttle('Jupiter');
```

The class syntax simply replaces the constructor function creation:

```
class SpaceShuttle {
  constructor(targetPlanet){
    this.targetPlanet = targetPlanet;
  }
}
const zeus = new SpaceShuttle('Jupiter');
```

Notice that the `class` keyword declares a new function, and a constructor was added, which would be invoked when `new` is called - to create a new object.

Note

`UpperCamelCase` should be used by convention for ES6 class names, as in `SpaceShuttle` used above.

Instructions

Use `class` keyword and write a proper constructor to create the `Vegetable` class. The `Vegetable` lets you create a vegetable object, with a property `name`, to be passed to constructor.

Challenge Seed

```
function makeClass() {
  "use strict";
  /* Alter code below this line */
```

```

/* Alter code above this line */
return Vegetable;
}
const Vegetable = makeClass();
const carrot = new Vegetable('carrot');
console.log(carrot.name); // => should be 'carrot'

```

Solution

```

function makeClass() {
  "use strict";
  /* Alter code below this line */
  class Vegetable {
    constructor(name){
      this.name = name;
    }
  }
  /* Alter code above this line */
  return Vegetable;
}
const Vegetable = makeClass();
const carrot = new Vegetable('carrot');
console.log(carrot.name); // => should be 'carrot'

```

21. Use getters and setters to Control Access to an Object

Description

You can obtain values from an object, and set a value of a property within an object. These are classically called getters and setters. Getter functions are meant to simply return (get) the value of an object's private variable to the user without the user directly accessing the private variable. Setter functions are meant to modify (set) the value of an object's private variable based on the value passed into the setter function. This change could involve calculations, or even overwriting the previous value completely.

```

class Book {
  constructor(author) {
    this._author = author;
  }
  // getter
  get writer(){
    return this._author;
  }
  // setter
  set writer(updatedAuthor){
    this._author = updatedAuthor;
  }
}
const lol = new Book('anonymous');
console.log(lol.writer); // anonymous
lol.writer = 'wut';
console.log(lol.writer); // wut

```

Notice the syntax we are using to invoke the getter and setter - as if they are not even functions. Getters and setters are important, because they hide internal implementation details.

Note:

It is a convention to precede the name of a private variable with an underscore (_). The practice itself does not make a variable private.

Instructions

Use `class` keyword to create a `Thermostat` class. The constructor accepts Fahrenheit temperature. Now create `getter` and `setter` in the class, to obtain the temperature in Celsius scale. Remember that $C = 5/9 * (F - 32)$ and $F = C * 9.0 / 5 + 32$, where F is the value of temperature in Fahrenheit scale, and C is the value of the same temperature in Celsius scale. Note When you implement this, you would be tracking the temperature inside the class in one scale - either Fahrenheit or Celsius. This is the power of getter or setter - you are creating an API for another user, who would get the correct result, no matter which one you track. In other words, you are abstracting implementation details from the consumer.

Challenge Seed

```
function makeClass() {
  "use strict";
  /* Alter code below this line */

  /* Alter code above this line */
  return Thermostat;
}
const Thermostat = makeClass();
const thermos = new Thermostat(76); // setting in Fahrenheit scale
let temp = thermos.temperature; // 24.44 in C
thermos.temperature = 26;
temp = thermos.temperature; // 26 in C
```

Solution

```
function makeClass() {
  "use strict";
  /* Alter code below this line */
  class Thermostat {
    constructor(fahrenheit) {
      this._tempInCelsius = 5/9 * (fahrenheit - 32);
    }
    get tempInCelsius(){
      return _tempInCelsius;
    }
    set tempInCelsius(newTemp){
      this._tempInCelsius = newTemp;
    }
  }
  /* Alter code above this line */
  return Thermostat;
}
const Thermostat = makeClass();
const thermos = new Thermostat(76); // setting in Fahrenheit scale
let temp = thermos.temperature; // 24.44 in C
thermos.temperature = 26;
temp = thermos.temperature; // 26 in C
```

22. Understand the Differences Between import and require

Description

In the past, the function `require()` would be used to import the functions and code in external files and modules. While handy, this presents a problem: some files and modules are rather large, and you may only need certain code from those external resources. ES6 gives us a very handy tool known as `import`. With it, we can choose which parts of a module or file to load into a given file, saving time and memory. Consider the following example. Imagine that `math_array_functions` has about 20 functions, but I only need one, `countItems`, in my current file. The old

`require()` approach would force me to bring in all 20 functions. With this new `import` syntax, I can bring in just the desired function, like so:

```
import { countItems } from "math_array_functions"
```

A description of the above code:

```
import { function } from "file_path_goes_here"
// We can also import variables the same way!
```

There are a few ways to write an `import` statement, but the above is a very common use-case. **Note**

The whitespace surrounding the function inside the curly braces is a best practice - it makes it easier to read the `import` statement. **Note**

The lessons in this section handle non-browser features. `import`, and the statements we introduce in the rest of these lessons, won't work on a browser directly. However, we can use various tools to create code out of this to make it work in browser. **Note**

In most cases, the file path requires a `./` before it; otherwise, node will look in the `node_modules` directory first trying to load it as a dependency.

Instructions

Add the appropriate `import` statement that will allow the current file to use the `capitalizeString` function. The file where this function lives is called `"string_functions"`, and it is in the same directory as the current file.

Challenge Seed

```
"use strict";
capitalizeString("hello!");
```

Before Test

```
self.require = function (str) {
  if (str === 'string_functions') {
    return {
      capitalizeString: str => str.toUpperCase()
    }
  }
};
```

Solution

```
import { capitalizeString } from 'string_functions';
capitalizeString("hello!");
```

23. Use export to Reuse a Code Block

Description

In the previous challenge, you learned about `import` and how it can be leveraged to import small amounts of code from large files. In order for this to work, though, we must utilize one of the statements that goes with `import`, known as `export`. When we want some code - a function, or a variable - to be usable in another file, we must export it in order to import it into another file. Like `import`, `export` is a non-browser feature. The following is what we refer to as a named export. With this, we can import any code we export into another file with the `import` syntax you learned in the last lesson. Here's an example:

```
const capitalizeString = (string) => {
  return string.charAt(0).toUpperCase() + string.slice(1);
}
export { capitalizeString } //How to export functions.
export const foo = "bar"; //How to export variables.
```

Alternatively, if you would like to compact all your `export` statements into one line, you can take this approach:

```
const capitalizeString = (string) => {
  return string.charAt(0).toUpperCase() + string.slice(1);
}
const foo = "bar";
export { capitalizeString, foo }
```

Either approach is perfectly acceptable.

Instructions

Below are two variables that I want to make available for other files to use. Utilizing the first way I demonstrated `export`, export the two variables.

Challenge Seed

```
"use strict";
const foo = "bar";
const bar = "foo";
```

Before Test

```
self.exports = function(){};
```

Solution

```
"use strict";
export const foo = "bar";
export const bar = "foo";
```

24. Use * to Import Everything from a File

Description

Suppose you have a file that you wish to import all of its contents into the current file. This can be done with the `import *` syntax. Here's an example where the contents of a file named `"math_functions"` are imported into a file in the same directory:

```
import * as myMathModule from "math_functions";
myMathModule.add(2,3);
myMathModule.subtract(5,3);
```

And breaking down that code:

```
import * as object_with_name_of_your_choice from "file_path_goes_here"
object_with_name_of_your_choice.imported_function
```

You may use any name following the `import * as` portion of the statement. In order to utilize this method, it requires an object that receives the imported values. From here, you will use the dot notation to call your imported values.

Instructions

The code below requires the contents of a file, `"capitalize_strings"`, found in the same directory as it, imported. Add the appropriate `import *` statement to the top of the file, using the object provided.

Challenge Seed

```
"use strict";
```

Before Test

```
self.require = function(str) {
  if (str === 'capitalize_strings') {
    return {
      capitalize: str => str.toUpperCase(),
      lowercase: str => str.toLowerCase()
    }
  }
};
```

Solution

```
import * as capitalize_strings from "capitalize_strings";
```

25. Create an Export Fallback with export default

Description

In the `export` lesson, you learned about the syntax referred to as a named export. This allowed you to make multiple functions and variables available for use in other files. There is another `export` syntax you need to know, known as `export default`. Usually you will use this syntax if only one value is being exported from a file. It is also used to create a fallback value for a file or module. Here is a quick example of `export default`:

```
export default function add(x,y) {
  return x + y;
}
```

Note: Since `export default` is used to declare a fallback value for a module or file, you can only have one value be a default export in each module or file. Additionally, you cannot use `export default` with `var`, `let`, or `const`

Instructions

The following function should be the fallback value for the module. Please add the necessary code to do so.

Challenge Seed

```
"use strict";
function subtract(x,y) {return x - y;}
```

Before Test

```
self.exports = function(){};
```

Solution

```
export default function subtract(x,y) {return x - y;}
```

26. Import a Default Export

Description

In the last challenge, you learned about `export default` and its uses. It is important to note that, to import a default export, you need to use a different `import` syntax. In the following example, we have a function, `add`, that is the default export of a file, "math_functions". Here is how to import it:

```
import add from "math_functions";
add(5,4); //Will return 9
```

The syntax differs in one key place - the imported value, `add`, is not surrounded by curly braces, `{}`. Unlike exported values, the primary method of importing a default export is to simply write the value's name after `import`.

Instructions

In the following code, please import the default export, `subtract`, from the file "math_functions", found in the same directory as this file.

Challenge Seed

```
"use strict";
subtract(7,4);
```

Before Test

```
self.require = function(str) {
  if (str === 'math_functions') {
    return function(a, b) {
      return a - b;
    }
  }
};
```

Solution

```
import subtract from "math_functions";
subtract(7,4);
```

Regular Expressions

1. Using the Test Method

Description

Regular expressions are used in programming languages to match parts of strings. You create patterns to help you do that matching. If you want to find the word "the" in the string "The dog chased the cat", you could use the following regular expression: `/the/`. Notice that quote marks are not required within the regular expression. JavaScript has multiple ways to use regexes. One way to test a regex is using the `.test()` method. The `.test()` method takes the regex, applies it to a string (which is placed inside the parentheses), and returns `true` or `false` if your pattern finds something or not.

```
let testStr = "freeCodeCamp";
let testRegex = /Code/;
testRegex.test(testStr);
// Returns true
```

Instructions

Apply the regex `myRegex` on the string `myString` using the `.test()` method.

Challenge Seed

```
let myString = "Hello, World!";
let myRegex = /Hello/;
let result = myRegex; // Change this line
```

Solution

```
// solution required
```

2. Match Literal Strings

Description

In the last challenge, you searched for the word "Hello" using the regular expression `/Hello/`. That regex searched for a literal match of the string "Hello". Here's another example searching for a literal match of the string "Kevin":

```
let testStr = "Hello, my name is Kevin.";
let testRegex = /Kevin/;
testRegex.test(testStr);
// Returns true
```

Any other forms of "Kevin" will not match. For example, the regex `/Kevin/` will not match "kevin" or "KEVIN".

```
let wrongRegex = /kevin/;
wrongRegex.test(testStr);
// Returns false
```

A future challenge will show how to match those other forms as well.

Instructions

Complete the regex `waldoRegex` to find "Waldo" in the string `waldoIsHiding` with a literal match.

Challenge Seed

```
let waldoIsHiding = "Somewhere Waldo is hiding in this text.";
let waldoRegex = /search/; // Change this line
let result = waldoRegex.test(waldoIsHiding);
```

Solution

```
// solution required
```

3. Match a Literal String with Different Possibilities

Description

Using regexes like `/coding/`, you can look for the pattern "coding" in another string. This is powerful to search single strings, but it's limited to only one pattern. You can search for multiple patterns using the alternation or OR operator: `|`. This operator matches patterns either before or after it. For example, if you wanted to match "yes" or

"no" , the regex you want is `/yes|no/` . You can also search for more than just two patterns. You can do this by adding more patterns with more OR operators separating them, like `/yes|no|maybe/` .

Instructions

Complete the regex `petRegex` to match the pets "dog" , "cat" , "bird" , or "fish" .

Challenge Seed

```
let petString = "James has a pet cat.";
let petRegex = /change/; // Change this line
let result = petRegex.test(petString);
```

Solution

```
// solution required
```

4. Ignore Case While Matching

Description

Up until now, you've looked at regexes to do literal matches of strings. But sometimes, you might want to also match case differences. Case (or sometimes letter case) is the difference between uppercase letters and lowercase letters. Examples of uppercase are "A" , "B" , and "C" . Examples of lowercase are "a" , "b" , and "c" . You can match both cases using what is called a flag. There are other flags but here you'll focus on the flag that ignores case - the `i` flag. You can use it by appending it to the regex. An example of using this flag is `/ignorecase/i` . This regex can match the strings "ignorecase" , "igNoreCase" , and "IgnoreCase" .

Instructions

Write a regex `fccRegex` to match "freeCodeCamp" , no matter its case. Your regex should not match any abbreviations or variations with spaces.

Challenge Seed

```
let myString = "freeCodeCamp";
let fccRegex = /change/; // Change this line
let result = fccRegex.test(myString);
```

Solution

```
// solution required
```

5. Extract Matches

Description

So far, you have only been checking if a pattern exists or not within a string. You can also extract the actual matches you found with the `.match()` method. To use the `.match()` method, apply the method on a string and pass in the regex inside the parentheses. Here's an example:

```
"Hello, World!".match(/Hello/);
// Returns ["Hello"]
let ourStr = "Regular expressions";
let ourRegex = /expressions/;
ourStr.match(ourRegex);
// Returns ["expressions"]
```

Instructions

Apply the `.match()` method to extract the word `coding`.

Challenge Seed

```
let extractStr = "Extract the word 'coding' from this string.";
let codingRegex = /change/; // Change this line
let result = extractStr; // Change this line
```

Solution

```
// solution required
```

6. Find More Than the First Match

Description

So far, you have only been able to extract or search a pattern once.

```
let testStr = "Repeat, Repeat, Repeat";
let ourRegex = /Repeat/;
testStr.match(ourRegex);
// Returns ["Repeat"]
```

To search or extract a pattern more than once, you can use the `g` flag.

```
let repeatRegex = /Repeat/g;
testStr.match(repeatRegex);
// Returns ["Repeat", "Repeat", "Repeat"]
```

Instructions

Using the regex `starRegex`, find and extract both `"Twinkle"` words from the string `twinkleStar`. **Note**
You can have multiple flags on your regex like `/search/gi`

Challenge Seed

```
let twinkleStar = "Twinkle, twinkle, little star";
let starRegex = /change/; // Change this line
let result = twinkleStar; // Change this line
```

Solution

```
let twinkleStar = "Twinkle, twinkle, little star";
let starRegex = /twinkle/gi;
let result = twinkleStar.match(starRegex);
```

7. Match Anything with Wildcard Period

Description

Sometimes you won't (or don't need to) know the exact characters in your patterns. Thinking of all words that match, say, a misspelling would take a long time. Luckily, you can save time using the wildcard character: `.`. The wildcard character `.` will match any one character. The wildcard is also called `dot` and `period`. You can use the wildcard character just like any other character in the regex. For example, if you wanted to match "hug" , "huh" , "hut" , and "hum" , you can use the regex `/hu./` to match all four words.

```
let humStr = "I'll hum a song";
let hugStr = "Bear hug";
let huRegex = /hu./;
humStr.match(huRegex); // Returns ["hum"]
hugStr.match(huRegex); // Returns ["hug"]
```

Instructions

Complete the regex `unRegex` so that it matches the strings "run" , "sun" , "fun" , "pun" , "nun" , and "bun" . Your regex should use the wildcard character.

Challenge Seed

```
let exampleStr = "Let's have fun with regular expressions!";
let unRegex = /change/; // Change this line
let result = unRegex.test(exampleStr);
```

Solution

```
// solution required
```

8. Match Single Character with Multiple Possibilities

Description

You learned how to match literal patterns (`/literal/`) and wildcard character (`/./`). Those are the extremes of regular expressions, where one finds exact matches and the other matches everything. There are options that are a balance between the two extremes. You can search for a literal pattern with some flexibility with `character classes` . Character classes allow you to define a group of characters you wish to match by placing them inside square (`[` and `]`) brackets. For example, you want to match "bag" , "big" , and "bug" but not "bog" . You can create the regex `/b[aiu]g/` to do this. The `[aiu]` is the character class that will only match the characters "a" , "i" , or "u" .

```
let bigStr = "big";
let bagStr = "bag";
let bugStr = "bug";
let bogStr = "bog";
let bgRegex = /b[aiu]g/;
bigStr.match(bgRegex); // Returns ["big"]
bagStr.match(bgRegex); // Returns ["bag"]
bugStr.match(bgRegex); // Returns ["bug"]
bogStr.match(bgRegex); // Returns null
```

Instructions

Use a character class with vowels (`a` , `e` , `i` , `o` , `u`) in your regex `vowelRegex` to find all the vowels in the string `quoteSample` . **Note**

Be sure to match both upper- and lowercase vowels.

Challenge Seed

```
let quoteSample = "Beware of bugs in the above code; I have only proved it correct, not tried it.";
let vowelRegex = /change/; // Change this line
let result = vowelRegex; // Change this line
```

Solution

```
// solution required
```

9. Match Letters of the Alphabet

Description

You saw how you can use `character sets` to specify a group of characters to match, but that's a lot of typing when you need to match a large range of characters (for example, every letter in the alphabet). Fortunately, there is a built-in feature that makes this short and simple. Inside a `character set`, you can define a range of characters to match using a hyphen character: `-`. For example, to match lowercase letters `a` through `e` you would use `[a-e]`.

```
let catStr = "cat";
let batStr = "bat";
let matStr = "mat";
let bgRegex = /[a-e]at/;
catStr.match(bgRegex); // Returns ["cat"]
batStr.match(bgRegex); // Returns ["bat"]
matStr.match(bgRegex); // Returns null
```

Instructions

Match all the letters in the string `quoteSample`. Note
Be sure to match both upper- and lowercase **letters**.

Challenge Seed

```
let quoteSample = "The quick brown fox jumps over the lazy dog.";
let alphabetRegex = /change/; // Change this line
let result = alphabetRegex; // Change this line
```

Solution

```
// solution required
```

10. Match Numbers and Letters of the Alphabet

Description

Using the hyphen (`-`) to match a range of characters is not limited to letters. It also works to match a range of numbers. For example, `/[0-5]/` matches any number between `0` and `5`, including the `0` and `5`. Also, it is possible to combine a range of letters and numbers in a single character set.

```
let jennyStr = "Jenny8675309";
let myRegex = /[a-z0-9]/ig;
// matches all letters and numbers in jennyStr
jennyStr.match(myRegex);
```

Instructions

Create a single regex that matches a range of letters between h and s , and a range of numbers between 2 and 6 . Remember to include the appropriate flags in the regex.

Challenge Seed

```
let quoteSample = "Blueberry 3.141592653s are delicious.";
let myRegex = /change/; // Change this line
let result = myRegex; // Change this line
```

Solution

```
// solution required
```

11. Match Single Characters Not Specified

Description

So far, you have created a set of characters that you want to match, but you could also create a set of characters that you do not want to match. These types of character sets are called negated character sets . To create a negated character set , you place a caret character (^) after the opening bracket and before the characters you do not want to match. For example, /[^aeiou]/gi matches all characters that are not a vowel. Note that characters like . , ! , [, @ , / and white space are matched - the negated vowel character set only excludes the vowel characters.

Instructions

Create a single regex that matches all characters that are not a number or a vowel. Remember to include the appropriate flags in the regex.

Challenge Seed

```
let quoteSample = "3 blind mice.";
let myRegex = /change/; // Change this line
let result = myRegex; // Change this line
```

Solution

```
// solution required
```

12. Match Characters that Occur One or More Times

Description

Sometimes, you need to match a character (or group of characters) that appears one or more times in a row. This means it occurs at least once, and may be repeated. You can use the `+` character to check if that is the case. Remember, the character or pattern has to be present consecutively. That is, the character has to repeat one after the other. For example, `/a+/g` would find one match in `"abc"` and return `["a"]`. Because of the `+`, it would also find a single match in `"aabc"` and return `["aa"]`. If it were instead checking the string `"abab"`, it would find two matches and return `["a", "a"]` because the `a` characters are not in a row - there is a `b` between them. Finally, since there is no `"a"` in the string `"bcd"`, it wouldn't find a match.

Instructions

You want to find matches when the letter `s` occurs one or more times in `"Mississippi"`. Write a regex that uses the `+` sign.

Challenge Seed

```
let difficultSpelling = "Mississippi";
let myRegex = /change/; // Change this line
let result = difficultSpelling.match(myRegex);
```

Solution

```
let difficultSpelling = "Mississippi";
let myRegex = /s+/g; // Change this line
let result = difficultSpelling.match(myRegex);
```

13. Match Characters that Occur Zero or More Times

Description

The last challenge used the plus `+` sign to look for characters that occur one or more times. There's also an option that matches characters that occur zero or more times. The character to do this is the asterisk `*` or star:

```
*.
let soccerWord = "ooooooooal!";
let gPhrase = "gut feeling";
let oPhrase = "over the moon";
let goRegex = /go*/;
soccerWord.match(goRegex); // Returns ["oooooooo"]
gPhrase.match(goRegex); // Returns ["g"]
oPhrase.match(goRegex); // Returns null
```

Instructions

Create a regex `chewieRegex` that uses the `*` character to match all the upper and lowercase `"a"` characters in `chewieQuote`. Your regex does not need flags, and it should not match any of the other quotes.

Challenge Seed

```
let chewieQuote = "AAAAAAAAAAAAARRGH!";
let chewieRegex = /change/; // Change this line
let result = chewieQuote.match(chewieRegex);
```

Solution

```
let chewieQuote = "AAAAAAAAAAAAARRGH!";
let chewieRegex = /Aa*/;
```

```
let result = chewieQuote.match(chewieRegex);
```

14. Find Characters with Lazy Matching

Description

In regular expressions, a `greedy` match finds the longest possible part of a string that fits the regex pattern and returns it as a match. The alternative is called a `lazy` match, which finds the smallest possible part of the string that satisfies the regex pattern. You can apply the regex `/t[a-z]*i/` to the string "titanic". This regex is basically a pattern that starts with `t`, ends with `i`, and has some letters in between. Regular expressions are by default `greedy`, so the match would return `["titani"]`. It finds the largest sub-string possible to fit the pattern. However, you can use the `? character` to change it to `lazy matching`. "titanic" matched against the adjusted regex of `/t[a-z]*?i/` returns `["ti"]`.

Instructions

Fix the regex `/<.*>/` to return the HTML tag `<h1>` and not the text "`<h1>Winter is coming</h1>`". Remember the wildcard `.` in a regular expression matches any character.

Challenge Seed

```
let text = "<h1>Winter is coming</h1>";
let myRegex = /<.*>/; // Change this line
let result = text.match(myRegex);
```

Solution

```
// solution required
```

15. Find One or More Criminals in a Hunt

Description

Time to pause and test your new regex writing skills. A group of criminals escaped from jail and ran away, but you don't know how many. However, you do know that they stay close together when they are around other people. You are responsible for finding all of the criminals at once. Here's an example to review how to do this: The regex `/z+/` matches the letter `z` when it appears one or more times in a row. It would find matches in all of the following strings:

```
"z"
"zzzzzz"
"ABCzzzz"
"zzzzABC"
"abczzzzzzzzzzzzzzzzzzabc"
```

But it does not find matches in the following strings since there are no letter `z` characters:

```
"""
"ABC"
"abcabc"
```

Instructions

Write a `greedy` regex that finds one or more criminals within a group of other people. A criminal is represented by the capital letter `C`.

Challenge Seed

```
// example crowd gathering
let crowd = 'P1P2P3P4P5P6CCCP7P8P9';

let reCriminals = ./; // Change this line

let matchedCriminals = crowd.match(reCriminals);
console.log(matchedCriminals);
```

Solution

```
// solution required
```

16. Match Beginning String Patterns

Description

Prior challenges showed that regular expressions can be used to look for a number of matches. They are also used to search for patterns in specific positions in strings. In an earlier challenge, you used the `caret` character (`^`) inside a character set to create a negated character set in the form `[^thingsThatWillNotBeMatched]`. Outside of a character set, the `caret` is used to search for patterns at the beginning of strings.

```
let firstString = "Ricky is first and can be found.";
let firstRegex = /^Ricky/;
firstRegex.test(firstString);
// Returns true
let notFirst = "You can't find Ricky now.";
firstRegex.test(notFirst);
// Returns false
```

Instructions

Use the `caret` character in a regex to find "Cal" only in the beginning of the string `rickyAndCal`.

Challenge Seed

```
let rickyAndCal = "Cal and Ricky both like racing.";
let calRegex = /change/; // Change this line
let result = calRegex.test(rickyAndCal);
```

Solution

```
// solution required
```

17. Match Ending String Patterns

Description

In the last challenge, you learned to use the `caret` character to search for patterns at the beginning of strings. There is also a way to search for patterns at the end of strings. You can search the end of strings using the dollar sign character `$` at the end of the regex.

```
let theEnding = "This is a never ending story";
let storyRegex = /story$/;
storyRegex.test(theEnding);
// Returns true
let noEnding = "Sometimes a story will have to end";
storyRegex.test(noEnding);
// Returns false
```

Instructions

Use the anchor character (\$) to match the string "caboose" at the end of the string caboose .

Challenge Seed

```
let caboose = "The last car on a train is the caboose";
let lastRegex = /change/; // Change this line
let result = lastRegex.test(caboose);
```

Solution

```
// solution required
```

18. Match All Letters and Numbers

Description

Using character classes, you were able to search for all letters of the alphabet with [a-z] . This kind of character class is common enough that there is a shortcut for it, although it includes a few extra characters as well. The closest character class in JavaScript to match the alphabet is \w . This shortcut is equal to [A-Za-z0-9_] . This character class matches upper and lowercase letters plus numbers. Note, this character class also includes the underscore character (_).

```
let longHand = /[A-Za-z0-9_]+/;
let shortHand = /\w+/";
let numbers = "42";
let varNames = "important_var";
longHand.test(numbers); // Returns true
shortHand.test(numbers); // Returns true
longHand.test(varNames); // Returns true
shortHand.test(varNames); // Returns true
```

These shortcut character classes are also known as shorthand character classes .

Instructions

Use the shorthand character class \w to count the number of alphanumeric characters in various quotes and strings.

Challenge Seed

```
let quoteSample = "The five boxing wizards jump quickly.";
let alphabetRegexV2 = /change/; // Change this line
let result = quoteSample.match(alphabetRegexV2).length;
```

Solution

```
// solution required
```

19. Match Everything But Letters and Numbers

Description

You've learned that you can use a shortcut to match alphanumerics `[A-Za-z0-9_]` using `\w`. A natural pattern you might want to search for is the opposite of alphanumerics. You can search for the opposite of the `\w` with `\W`. Note, the opposite pattern uses a capital letter. This shortcut is the same as `[^A-Za-z0-9_]`.

```
let shortHand = /\W/;
let numbers = "42%";
let sentence = "Coding!";
numbers.match(shortHand); // Returns ["%"]
sentence.match(shortHand); // Returns ["!"]
```

Instructions

Use the shorthand character class `\W` to count the number of non-alphanumeric characters in various quotes and strings.

Challenge Seed

```
let quoteSample = "The five boxing wizards jump quickly.";
let nonAlphabetRegex = /change/; // Change this line
let result = quoteSample.match(nonAlphabetRegex).length;
```

Solution

```
// solution required
```

20. Match All Numbers

Description

You've learned shortcuts for common string patterns like alphanumerics. Another common pattern is looking for just digits or numbers. The shortcut to look for digit characters is `\d`, with a lowercase `d`. This is equal to the character class `[0-9]`, which looks for a single character of any number between zero and nine.

Instructions

Use the shorthand character class `\d` to count how many digits are in movie titles. Written out numbers ("six" instead of 6) do not count.

Challenge Seed

```
let numString = "Your sandwich will be $5.00";
let numRegex = /change/; // Change this line
let result = numString.match(numRegex).length;
```

Solution

```
// solution required
```

21. Match All Non-Numbers

Description

The last challenge showed how to search for digits using the shortcut `\d` with a lowercase `d`. You can also search for non-digits using a similar shortcut that uses an uppercase `D` instead. The shortcut to look for non-digit characters is `\D`. This is equal to the character class `[^0-9]`, which looks for a single character that is not a number between zero and nine.

Instructions

Use the shorthand character class for non-digits `\D` to count how many non-digits are in movie titles.

Challenge Seed

```
let numString = "Your sandwich will be $5.00";
let noNumRegex = /change/; // Change this line
let result = numString.match(noNumRegex).length;
```

Solution

```
// solution required
```

22. Restrict Possible Usernames

Description

Usernames are used everywhere on the internet. They are what give users a unique identity on their favorite sites. You need to check all the usernames in a database. Here are some simple rules that users have to follow when creating their username. 1) The only numbers in the username have to be at the end. There can be zero or more of them at the end. 2) Username letters can be lowercase and uppercase. 3) Usernames have to be at least two characters long. A two-letter username can only use alphabet letter characters.

Instructions

Change the regex `userCheck` to fit the constraints listed above.

Challenge Seed

```
let username = "JackOfAllTrades";
let userCheck = /change/; // Change this line
let result = userCheck.test(username);
```

Solution

```
const userCheck = /^[A-Za-z]{2,}\d*$/;
```

23. Match Whitespace

Description

The challenges so far have covered matching letters of the alphabet and numbers. You can also match the whitespace or spaces between letters. You can search for whitespace using `\s`, which is a lowercase `s`. This pattern not only matches whitespace, but also carriage return, tab, form feed, and new line characters. You can think of it as similar to the character class `[\r\n\t\f\n\v]`.

```
let whiteSpace = "Whitespace. Whitespace everywhere!"
let spaceRegex = /\s/g;
whiteSpace.match(spaceRegex);
// Returns [" ", " "]
```

Instructions

Change the regex `countWhiteSpace` to look for multiple whitespace characters in a string.

Challenge Seed

```
let sample = "Whitespace is important in separating words";
let countWhiteSpace = /change/; // Change this line
let result = sample.match(countWhiteSpace);
```

Solution

```
let sample = "Whitespace is important in separating words";
let countWhiteSpace = /\s/g;
let result = sample.match(countWhiteSpace);
```

24. Match Non-Whitespace Characters

Description

You learned about searching for whitespace using `\s`, with a lowercase `s`. You can also search for everything except whitespace. Search for non-whitespace using `\S`, which is an uppercase `s`. This pattern will not match whitespace, carriage return, tab, form feed, and new line characters. You can think of it being similar to the character class `[^ \r\n\t\f\n\v]`.

```
let whiteSpace = "Whitespace. Whitespace everywhere!"
let nonSpaceRegex = /\S/g;
whiteSpace.match(nonSpaceRegex).length; // Returns 32
```

Instructions

Change the regex `countNonWhiteSpace` to look for multiple non-whitespace characters in a string.

Challenge Seed

```
let sample = "Whitespace is important in separating words";
let countNonWhiteSpace = /change/; // Change this line
let result = sample.match(countNonWhiteSpace);
```

Solution

```
// solution required
```

25. Specify Upper and Lower Number of Matches

Description

Recall that you use the plus sign `+` to look for one or more characters and the asterisk `*` to look for zero or more characters. These are convenient but sometimes you want to match a certain range of patterns. You can specify the lower and upper number of patterns with `quantity specifiers`. Quantity specifiers are used with curly brackets (`{` and `}`). You put two numbers between the curly brackets - for the lower and upper number of patterns. For example, to match only the letter `a` appearing between 3 and 5 times in the string `"ah"`, your regex would be `/a{3,5}h/`.

```
let A4 = "aaaah";
let A2 = "aah";
let multipleA = /a{3,5}h/;
multipleA.test(A4); // Returns true
multipleA.test(A2); // Returns false
```

Instructions

Change the regex `ohRegex` to match only 3 to 6 letter `h`'s in the word `"Oh no"`.

Challenge Seed

```
let ohStr = "Ohhh no";
let ohRegex = /change/; // Change this line
let result = ohRegex.test(ohStr);
```

Solution

```
// solution required
```

26. Specify Only the Lower Number of Matches

Description

You can specify the lower and upper number of patterns with `quantity specifiers` using curly brackets. Sometimes you only want to specify the lower number of patterns with no upper limit. To only specify the lower number of patterns, keep the first number followed by a comma. For example, to match only the string `"hah"` with the letter `a` appearing at least 3 times, your regex would be `/ha{3,}h/`.

```
let A4 = "aaaaah";
let A2 = "haah";
let A100 = "h" + "a".repeat(100) + "h";
let multipleA = /ha{3,}h/;
multipleA.test(A4); // Returns true
multipleA.test(A2); // Returns false
multipleA.test(A100); // Returns true
```

Instructions

Change the regex `haRegex` to match the word `"Hazzah"` only when it has four or more letter `z`'s.

Challenge Seed

```
let haStr = "Hazzzzah";
let haRegex = /change/; // Change this line
let result = haRegex.test(haStr);
```

Solution

```
// solution required
```

27. Specify Exact Number of Matches

Description

You can specify the lower and upper number of patterns with quantity specifiers using curly brackets. Sometimes you only want a specific number of matches. To specify a certain number of patterns, just have that one number between the curly brackets. For example, to match only the word "hah" with the letter a 3 times, your regex would be /ha{3}h/ .

```
let A4 = "haaaah";
let A3 = "haaah";
let A100 = "h" + "a".repeat(100) + "h";
let multipleHA = /ha{3}h/;
multipleHA.test(A4); // Returns false
multipleHA.test(A3); // Returns true
multipleHA.test(A100); // Returns false
```

Instructions

Change the regex timRegex to match the word "Timber" only when it has four letter m's.

Challenge Seed

```
let timStr = "Timmmmmber";
let timRegex = /change/; // Change this line
let result = timRegex.test(timStr);
```

Solution

```
// solution required
```

28. Check for All or None

Description

Sometimes the patterns you want to search for may have parts of it that may or may not exist. However, it may be important to check for them nonetheless. You can specify the possible existence of an element with a question mark, ?. This checks for zero or one of the preceding element. You can think of this symbol as saying the previous element is optional. For example, there are slight differences in American and British English and you can use the question mark to match both spellings.

```
let american = "color";
let british = "colour";
```

```
let rainbowRegex= /colou?r/;
rainbowRegex.test(american); // Returns true
rainbowRegex.test(british); // Returns true
```

Instructions

Change the regex `favRegex` to match both the American English (`favorite`) and the British English (`favourite`) version of the word.

Challenge Seed

```
let favWord = "favorite";
let favRegex = /change/; // Change this line
let result = favRegex.test(favWord);
```

Solution

```
let favWord = "favorite";
let favRegex = /favou?r/;
let result = favRegex.test(favWord);
```

29. Positive and Negative Lookahead

Description

Lookaheads are patterns that tell JavaScript to look-ahead in your string to check for patterns further along. This can be useful when you want to search for multiple patterns over the same string. There are two kinds of lookaheads : positive lookahead and negative lookahead . A positive lookahead will look to make sure the element in the search pattern is there, but won't actually match it. A positive lookahead is used as `(?=...)` where the `...` is the required part that is not matched. On the other hand, a negative lookahead will look to make sure the element in the search pattern is not there. A negative lookahead is used as `(?!...)` where the `...` is the pattern that you do not want to be there. The rest of the pattern is returned if the negative lookahead part is not present. Lookaheads are a bit confusing but some examples will help.

```
let quit = "qu";
let noquit = "qt";
let quRegex= /q(?=u)/;
let qRegex = /q(?!u)/;
quit.match(quRegex); // Returns ["q"]
noquit.match(qRegex); // Returns ["q"]
```

A more practical use of `lookaheads` is to check two or more patterns in one string. Here is a (naively) simple password checker that looks for between 3 and 6 characters and at least one number:

```
let password = "abc123";
let checkPass = /(?=\\w{3,6})(?=\\D*\\d)/;
checkPass.test(password); // Returns true
```

Instructions

Use lookaheads in the `pwRegex` to match passwords that are greater than 5 characters long, do not begin with numbers, and have two consecutive digits.

Challenge Seed

```
let sampleWord = "astronaut";
let pwRegex = /change/; // Change this line
let result = pwRegex.test(sampleWord);
```

Solution

```
var pwRegex = /(?=\\w{5})(?=\\D*\\d{2})/;
```

30. Reuse Patterns Using Capture Groups

Description

Some patterns you search for will occur multiple times in a string. It is wasteful to manually repeat that regex. There is a better way to specify when you have multiple repeat substrings in your string. You can search for repeat substrings using capture groups. Parentheses, (and), are used to find repeat substrings. You put the regex of the pattern that will repeat in between the parentheses. To specify where that repeat string will appear, you use a backslash (\) and then a number. This number starts at 1 and increases with each additional capture group you use. An example would be \1 to match the first group. The example below matches any word that occurs twice separated by a space:

```
let repeatStr = "regex regex";
let repeatRegex = /(\\w+)\\s\\1/;
repeatRegex.test(repeatStr); // Returns true
repeatStr.match(repeatRegex); // Returns ["regex regex", "regex"]
```

Using the `.match()` method on a string will return an array with the string it matches, along with its capture group.

Instructions

Use capture groups in `reRegex` to match numbers that are repeated only three times in a string, each separated by a space.

Challenge Seed

```
let repeatNum = "42 42 42";
let reRegex = /change/; // Change this line
let result = reRegex.test(repeatNum);
```

Solution

```
let repeatNum = "42 42 42";
let reRegex = /^(\d+)\\s\\1\\s\\1$/;
let result = reRegex.test(repeatNum);
```

31. Use Capture Groups to Search and Replace

Description

Searching is useful. However, you can make searching even more powerful when it also changes (or replaces) the text you match. You can search and replace text in a string using `.replace()` on a string. The inputs for `.replace()` is first the regex pattern you want to search for. The second parameter is the string to replace the match or a function to do something.

```
let wrongText = "The sky is silver.";
let silverRegex = /silver/;
wrongText.replace(silverRegex, "blue");
// Returns "The sky is blue."
```

You can also access capture groups in the replacement string with dollar signs (\$).

```
"Code Camp".replace(/(\w+)\s(\w+)/, '$2 $1');
// Returns "Camp Code"
```

Instructions

Write a regex so that it will search for the string "good" . Then update the `replaceText` variable to replace "good" with "okey-dokey" .

Challenge Seed

```
let huhText = "This sandwich is good.";
let fixRegex = /change/; // Change this line
let replaceText = ""; // Change this line
let result = huhText.replace(fixRegex, replaceText);
```

Solution

```
// solution required
```

32. Remove Whitespace from Start and End

Description

Sometimes whitespace characters around strings are not wanted but are there. Typical processing of strings is to remove the whitespace at the start and end of it.

Instructions

Write a regex and use the appropriate string methods to remove whitespace at the beginning and end of strings.

Note

The `.trim()` method would work here, but you'll need to complete this challenge using regular expressions.

Challenge Seed

```
let hello = "Hello, World! ";
let wsRegex = /change/; // Change this line
let result = hello; // Change this line
```

Solution

```
let hello = "Hello, World! ";
let wsRegex = /^(\s+)(.+[\^s])(\s+)$/;
let result = hello.replace(wsRegex, '$2');
```

Debugging

1. Use the JavaScript Console to Check the Value of a Variable

Description

Both Chrome and Firefox have excellent JavaScript consoles, also known as DevTools, for debugging your JavaScript. You can find Developer tools in your Chrome's menu or Web Console in Firefox's menu. If you're using a different browser, or a mobile phone, we strongly recommend switching to desktop Firefox or Chrome. The `console.log()` method, which "prints" the output of what's within its parentheses to the console, will likely be the most helpful debugging tool. Placing it at strategic points in your code can show you the intermediate values of variables. It's good practice to have an idea of what the output should be before looking at what it is. Having check points to see the status of your calculations throughout your code will help narrow down where the problem is. Here's an example to print 'Hello world!' to the console: `console.log('Hello world!');`

Instructions

Use the `console.log()` method to print the value of the variable `a` where noted in the code.

Challenge Seed

```
let a = 5;
let b = 1;
a++;
// Add your code below this line

let sumAB = a + b;
console.log(sumAB);
```

Solution

```
var a = 5; console.log(a);
```

2. Understanding the Differences between the freeCodeCamp and Browser Console

Description

You may have noticed that some freeCodeCamp JavaScript challenges include their own console. This console behaves a little differently than the browser console you used in the last challenge. The following challenge is meant to highlight some of the differences between the freeCodeCamp console and the browser console. First, the browser console. When you load and run an ordinary JavaScript file in your browser the `console.log()` statements will print exactly what you tell them to print to the browser console the exact number of times you requested. In your in-browser text editor the process is slightly different and can be confusing at first. Values passed to `console.log()` in the text editor block run each set of tests as well as one more time for any function calls that you have in your code. This lends itself to some interesting behavior and might trip you up in the beginning, because a logged value that you expect to see only once may print out many more times depending on the number of tests and the values being passed to those tests. If you would like to see only your single output and not have to worry about running through the test cycles, you can use `console.clear()`.

Instructions

Use `console.log()` to print the variables in the code where indicated.

Challenge Seed

```
// Open your browser console
let outputTwo = "This will print to the browser console 2 times";
```

```
// Use console.log() to print the outputTwo variable

let outputOne = "Try to get this to log only once to the browser console";
// Use console.clear() in the next line to print the outputOne only once

// Use console.log() to print the outputOne variable
```

Solution

```
let outputTwo = "This will print to the browser console 2 times"; console.log(outputTwo); let outputOne
= "Try to get this to log only once to the browser console";
console.clear();
console.log(outputOne);
```

3. Use typeof to Check the Type of a Variable

Description

You can use `typeof` to check the data structure, or type, of a variable. This is useful in debugging when working with multiple data types. If you think you're adding two numbers, but one is actually a string, the results can be unexpected. Type errors can lurk in calculations or function calls. Be careful especially when you're accessing and working with external data in the form of a JavaScript Object Notation (JSON) object. Here are some examples using `typeof`:

```
console.log(typeof ""); // outputs "string"
console.log(typeof 0); // outputs "number"
console.log(typeof []); // outputs "object"
console.log(typeof {}); // outputs "object"
```

JavaScript recognizes six primitive (immutable) data types: Boolean , Null , Undefined , Number , String , and Symbol (new with ES6) and one type for mutable items: Object . Note that in JavaScript, arrays are technically a type of object.

Instructions

Add two `console.log()` statements to check the `typeof` each of the two variables `seven` and `three` in the code.

Challenge Seed

```
let seven = 7;
let three = "3";
console.log(seven + three);
// Add your code below this line
```

Solution

```
let seven = 7;let three = "3";console.log(typeof seven);
console.log(typeof three);
```

4. Catch Misspelled Variable and Function Names

Description

The `console.log()` and `typeof` methods are the two primary ways to check intermediate values and types of program output. Now it's time to get into the common forms that bugs take. One syntax-level issue that fast typers can commiserate with is the humble spelling error. Transposed, missing, or mis-capitalized characters in a variable or function name will have the browser looking for an object that doesn't exist - and complain in the form of a reference error. JavaScript variable and function names are case-sensitive.

Instructions

Fix the two spelling errors in the code so the `networkingCapital` calculation works.

Challenge Seed

```
let receivables = 10;
let payables = 8;
let networkingCapital = recievables - payable;
console.log(`Net working capital is: ${networkingCapital}`);
```

Solution

```
// solution required
```

5. Catch Unclosed Parentheses, Brackets, Braces and Quotes

Description

Another syntax error to be aware of is that all opening parentheses, brackets, curly braces, and quotes have a closing pair. Forgetting a piece tends to happen when you're editing existing code and inserting items with one of the pair types. Also, take care when nesting code blocks into others, such as adding a callback function as an argument to a method. One way to avoid this mistake is as soon as the opening character is typed, immediately include the closing match, then move the cursor back between them and continue coding. Fortunately, most modern code editors generate the second half of the pair automatically.

Instructions

Fix the two pair errors in the code.

Challenge Seed

```
let myArray = [1, 2, 3;
let arraySum = myArray.reduce((previous, current => previous + current);
console.log(`Sum of array values is: ${arraySum}`);
```

Solution

```
// solution required
```

6. Catch Mixed Usage of Single and Double Quotes

Description

JavaScript allows the use of both single ("") and double (""" quotes to declare a string. Deciding which one to use generally comes down to personal preference, with some exceptions. Having two choices is great when a string has contractions or another piece of text that's in quotes. Just be careful that you don't close the string too early, which causes a syntax error. Here are some examples of mixing quotes:

```
// These are correct:  
const grouchoContraction = "I've had a perfectly wonderful evening, but this wasn't it.";  
const quoteInString = "Groucho Marx once said 'Quote me as saying I was mis-quoted.'";  
// This is incorrect:  
const uhOhGroucho = 'I've had a perfectly wonderful evening, but this wasn't it.';
```

Of course, it is okay to use only one style of quotes. You can escape the quotes inside the string by using the backslash (\) escape character:

```
// Correct use of same quotes:  
const allSameQuotes = 'I\'ve had a perfectly wonderful evening, but this wasn\'t it.';
```

Instructions

Fix the string so it either uses different quotes for the `href` value, or escape the existing ones. Keep the double quote marks around the entire string.

Challenge Seed

```
let innerHtml = "<p>Click here to <a href="#Home">return home</a></p>";  
console.log(innerHtml);
```

Solution

```
// solution required
```

7. Catch Use of Assignment Operator Instead of Equality Operator

Description

Branching programs, i.e. ones that do different things if certain conditions are met, rely on `if`, `else if`, and `else` statements in JavaScript. The condition sometimes takes the form of testing whether a result is equal to a value. This logic is spoken (in English, at least) as "if x equals y, then ..." which can literally translate into code using the `=`, or assignment operator. This leads to unexpected control flow in your program. As covered in previous challenges, the assignment operator (`=`) in JavaScript assigns a value to a variable name. And the `==` and `===` operators check for equality (the triple `==` tests for strict equality, meaning both value and type are the same). The code below assigns `x` to be 2, which evaluates as `true`. Almost every value on its own in JavaScript evaluates to `true`, except what are known as the "falsy" values: `false`, `0`, `""` (an empty string), `NaN`, `undefined`, and `null`.

```
let x = 1;  
let y = 2;  
if (x = y) {  
    // this code block will run for any value of y (unless y were originally set as a falsy)  
} else {  
    // this code block is what should run (but won't) in this example  
}
```

Instructions

Fix the condition so the program runs the right branch, and the appropriate value is assigned to `result`.

Challenge Seed

```
let x = 7;
let y = 9;
let result = "to come";

if(x == y) {
  result = "Equal!";
} else {
  result = "Not equal!";
}

console.log(result);
```

Solution

```
// solution required
```

8. Catch Missing Open and Closing Parenthesis After a Function Call

Description

When a function or method doesn't take any arguments, you may forget to include the (empty) opening and closing parentheses when calling it. Often times the result of a function call is saved in a variable for other use in your code. This error can be detected by logging variable values (or their types) to the console and seeing that one is set to a function reference, instead of the expected value the function returns. The variables in the following example are different:

```
function myFunction() {
  return "You rock!";
}
let varOne = myFunction; // set to equal a function
let varTwo = myFunction(); // set to equal the string "You rock!"
```

Instructions

Fix the code so the variable `result` is set to the value returned from calling the function `getNine`.

Challenge Seed

```
function getNine() {
  let x = 6;
  let y = 3;
  return x + y;
}

let result = getNine;
console.log(result);
```

Solution

```
// solution required
```

9. Catch Arguments Passed in the Wrong Order When Calling a Function

Description

Continuing the discussion on calling functions, the next bug to watch out for is when a function's arguments are supplied in the incorrect order. If the arguments are different types, such as a function expecting an array and an integer, this will likely throw a runtime error. If the arguments are the same type (all integers, for example), then the logic of the code won't make sense. Make sure to supply all required arguments, in the proper order to avoid these issues.

Instructions

The function `raiseToPower` raises a base to an exponent. Unfortunately, it's not called properly - fix the code so the value of `power` is the expected 8.

Challenge Seed

```
function raiseToPower(b, e) {
  return Math.pow(b, e);
}

let base = 2;
let exp = 3;
let power = raiseToPower(exp, base);
console.log(power);
```

Solution

```
// solution required
```

10. Catch Off By One Errors When Using Indexing

Description

Off by one errors (sometimes called OBOE) crop up when you're trying to target a specific index of a string or array (to slice or access a segment), or when looping over the indices of them. JavaScript indexing starts at zero, not one, which means the last index is always one less than the length of the item. If you try to access an index equal to the length, the program may throw an "index out of range" reference error or print `undefined`. When you use string or array methods that take index ranges as arguments, it helps to read the documentation and understand if they are inclusive (the item at the given index is part of what's returned) or not. Here are some examples of off by one errors:

```
let alphabet = "abcdefghijklmnopqrstuvwxyz";
let len = alphabet.length;
for (let i = 0; i <= len; i++) {
  // loops one too many times at the end
  console.log(alphabet[i]);
}
for (let j = 1; j < len; j++) {
  // loops one too few times and misses the first character at index 0
  console.log(alphabet[j]);
}
for (let k = 0; k < len; k++) {
  // Goldilocks approves - this is just right
```

```
    console.log(alphabet[k]);
}
```

Instructions

Fix the two indexing errors in the following function so all the numbers 1 through 5 are printed to the console.

Challenge Seed

```
function countToFive() {
  let firstFive = "12345";
  let len = firstFive.length;
  // Fix the line below
  for (let i = 1; i <= len; i++) {
    // Do not alter code below this line
    console.log(firstFive[i]);
  }
}

countToFive();
```

Solution

```
// solution required
```

11. Use Caution When Reinitializing Variables Inside a Loop

Description

Sometimes it's necessary to save information, increment counters, or re-set variables within a loop. A potential issue is when variables either should be reinitialized, and aren't, or vice versa. This is particularly dangerous if you accidentally reset the variable being used for the terminal condition, causing an infinite loop. Printing variable values with each cycle of your loop by using `console.log()` can uncover buggy behavior related to resetting, or failing to reset a variable.

Instructions

The following function is supposed to create a two-dimensional array with `m` rows and `n` columns of zeroes. Unfortunately, it's not producing the expected output because the `row` variable isn't being reinitialized (set back to an empty array) in the outer loop. Fix the code so it returns a correct 3x2 array of zeroes, which looks like `[[0, 0], [0, 0], [0, 0]]`.

Challenge Seed

```
function zeroArray(m, n) {
  // Creates a 2-D array with m rows and n columns of zeroes
  let newArray = [];
  let row = [];
  for (let i = 0; i < m; i++) {
    // Adds the m-th row into newArray

    for (let j = 0; j < n; j++) {
      // Pushes n zeroes into the current row to create the columns
      row.push(0);
    }
    // Pushes the current row, which now has n zeroes in it, to the array
    newArray.push(row);
  }
}
```

```

    return newArray;
}

let matrix = zeroArray(3, 2);
console.log(matrix);

```

Solution

```
// solution required
```

12. Prevent Infinite Loops with a Valid Terminal Condition

Description

The final topic is the dreaded infinite loop. Loops are great tools when you need your program to run a code block a certain number of times or until a condition is met, but they need a terminal condition that ends the looping. Infinite loops are likely to freeze or crash the browser, and cause general program execution mayhem, which no one wants. There was an example of an infinite loop in the introduction to this section - it had no terminal condition to break out of the `while` loop inside `loopy()`. Do NOT call this function!

```

function loopy() {
  while(true) {
    console.log("Hello, world!");
  }
}

```

It's the programmer's job to ensure that the terminal condition, which tells the program when to break out of the loop code, is eventually reached. One error is incrementing or decrementing a counter variable in the wrong direction from the terminal condition. Another one is accidentally resetting a counter or index variable within the loop code, instead of incrementing or decrementing it.

Instructions

The `myFunc()` function contains an infinite loop because the terminal condition `i != 4` will never evaluate to false (and break the looping) - `i` will increment by 2 each pass, and jump right over 4 since `i` is odd to start. Fix the comparison operator in the terminal condition so the loop only runs for `i` less than or equal to 4.

Challenge Seed

```

function myFunc() {
  for (let i = 1; i != 4; i += 2) {
    console.log("Still going!");
  }
}

```

Solution

```
// solution required
```

Basic Data Structures

1. Use an Array to Store a Collection of Data

Description

The below is an example of the simplest implementation of an array data structure. This is known as a one-dimensional array, meaning it only has one level, or that it does not have any other arrays nested within it. Notice it contains booleans, strings, and numbers, among other valid JavaScript data types:

```
let simpleArray = ['one', 2, 'three', true, false, undefined, null];
console.log(simpleArray.length);
// logs 7
```

All arrays have a `length` property, which as shown above, can be very easily accessed with the syntax `Array.length`. A more complex implementation of an array can be seen below. This is known as a multi-dimensional array, or an array that contains other arrays. Notice that this array also contains JavaScript objects, which we will examine very closely in our next section, but for now, all you need to know is that arrays are also capable of storing complex objects.

```
let complexArray = [
  [
    {
      one: 1,
      two: 2
    },
    {
      three: 3,
      four: 4
    }
  ],
  [
    {
      a: "a",
      b: "b"
    },
    {
      c: "c",
      d: "d"
    }
  ]
];
```

Instructions

We have defined a variable called `yourArray`. Complete the statement by assigning an array of at least 5 elements in length to the `yourArray` variable. Your array should contain at least one string, one number, and one boolean.

Challenge Seed

```
let yourArray; // change this line
```

Solution

```
// solution required
```

2. Access an Array's Contents Using Bracket Notation

Description

The fundamental feature of any data structure is, of course, the ability to not only store data, but to be able to retrieve that data on command. So, now that we've learned how to create an array, let's begin to think about how we can access that array's information. When we define a simple array as seen below, there are 3 items in it:

```
let ourArray = ["a", "b", "c"];
```

In an array, each array item has an index. This index doubles as the position of that item in the array, and how you reference it. However, it is important to note, that JavaScript arrays are zero-indexed, meaning that the first element of an array is actually at the *zeroth* position, not the first. In order to retrieve an element from an array we can enclose an index in brackets and append it to the end of an array, or more commonly, to a variable which references an array object. This is known as bracket notation. For example, if we want to retrieve the "a" from `ourArray` and assign it to a variable, we can do so with the following code:

```
let ourVariable = ourArray[0];
// ourVariable equals "a"
```

In addition to accessing the value associated with an index, you can also set an index to a value using the same notation:

```
ourArray[1] = "not b anymore";
// ourArray now equals ["a", "not b anymore", "c"];
```

Using bracket notation, we have now reset the item at index 1 from "b" , to "not b anymore" .

Instructions

In order to complete this challenge, set the 2nd position (index 1) of `myArray` to anything you want, besides "b" .

Challenge Seed

```
let myArray = ["a", "b", "c", "d"];
// change code below this line

//change code above this line
console.log(myArray);
```

Solution

```
// solution required
```

3. Add Items to an Array with `push()` and `unshift()`

Description

An array's length, like the data types it can contain, is not fixed. Arrays can be defined with a length of any number of elements, and elements can be added or removed over time; in other words, arrays are **mutable**. In this challenge, we will look at two methods with which we can programmatically modify an array: `Array.push()` and `Array.unshift()` . Both methods take one or more elements as parameters and add those elements to the array the method is being called on; the `push()` method adds elements to the end of an array, and `unshift()` adds elements to the beginning. Consider the following:

```
let twentyThree = 'XXIII';
let romanNumerals = ['XXI', 'XXII'];

romanNumerals.unshift('XIX', 'XX');
// now equals ['XIX', 'XX', 'XXI', 'XXII']

romanNumerals.push(twentyThree);
// now equals ['XIX', 'XX', 'XXI', 'XXII', 'XXIII'] Notice that we can also pass variables, which allows us even greater flexibility in dynamically modifying our array's data.
```

Instructions

We have defined a function, `mixedNumbers`, which we are passing an array as an argument. Modify the function by using `push()` and `unshift()` to add 'I', 2, 'three' to the beginning of the array and 7, 'VIII', 9 to the end so that the returned array contains representations of the numbers 1-9 in order.

Challenge Seed

```
function mixedNumbers(arr) {
  // change code below this line

  // change code above this line
  return arr;
}

// do not change code below this line
console.log(mixedNumbers(['IV', 5, 'six']));
```

Solution

```
// solution required
```

4. Remove Items from an Array with `pop()` and `shift()`

Description

Both `push()` and `unshift()` have corresponding methods that are nearly functional opposites: `pop()` and `shift()`. As you may have guessed by now, instead of adding, `pop()` removes an element from the end of an array, while `shift()` removes an element from the beginning. The key difference between `pop()` and `shift()` and their cousins `push()` and `unshift()`, is that neither method takes parameters, and each only allows an array to be modified by a single element at a time. Let's take a look:

```
let greetings = ['whats up?', 'hello', 'see ya!'];

greetings.pop();
// now equals ['whats up?', 'hello']

greetings.shift();
// now equals ['hello']
```

We can also return the value of the removed element with either method like this:

```
let popped = greetings.pop();
// returns 'hello'
// greetings now equals []
```

Instructions

We have defined a function, `popShift`, which takes an array as an argument and returns a new array. Modify the function, using `pop()` and `shift()`, to remove the first and last elements of the argument array, and assign the removed elements to their corresponding variables, so that the returned array contains their values.

Challenge Seed

```
function popShift(arr) {
  let popped; // change this line
  let shifted; // change this line
  return [shifted, popped];
}
```

```
// do not change code below this line
console.log(popShift(['challenge', 'is', 'not', 'complete']));
```

Solution

```
// solution required
```

5. Remove Items Using splice()

Description

Ok, so we've learned how to remove elements from the beginning and end of arrays using `shift()` and `pop()`, but what if we want to remove an element from somewhere in the middle? Or remove more than one element at once? Well, that's where `splice()` comes in. `splice()` allows us to do just that: remove any number of consecutive elements from anywhere in an array. `splice()` can take up to 3 parameters, but for now, we'll focus on just the first 2. The first two parameters of `splice()` are integers which represent indexes, or positions, of the array that `splice()` is being called upon. And remember, arrays are *zero-indexed*, so to indicate the first element of an array, we would use `0`. `splice()`'s first parameter represents the index on the array from which to begin removing elements, while the second parameter indicates the number of elements to delete. For example:

```
let array = ['today', 'was', 'not', 'so', 'great'];

array.splice(2, 2);
// remove 2 elements beginning with the 3rd element
// array now equals ['today', 'was', 'great']
```

`splice()` not only modifies the array it's being called on, but it also returns a new array containing the value of the removed elements:

```
let array = ['I', 'am', 'feeling', 'really', 'happy'];

let newArray = array.splice(3, 2);
// newArray equals ['really', 'happy']
```

Instructions

We've defined a function, `sumOfTen`, which takes an array as an argument and returns the sum of that array's elements. Modify the function, using `splice()`, so that it returns a value of `10`.

Challenge Seed

```
function sumOfTen(arr) {
  // change code below this line

  // change code above this line
  return arr.reduce((a, b) => a + b);
}

// do not change code below this line
console.log(sumOfTen([2, 5, 1, 5, 2, 1]));
```

Solution

```
// solution required
```

6. Add Items Using splice()

Description

Remember in the last challenge we mentioned that `splice()` can take up to three parameters? Well, we can go one step further with `splice()` — in addition to removing elements, we can use that third parameter, which represents one or more elements, to *add* them as well. This can be incredibly useful for quickly switching out an element, or a set of elements, for another. For instance, let's say you're storing a color scheme for a set of DOM elements in an array, and want to dynamically change a color based on some action:

```
function colorChange(arr, index, newColor) {
  arr.splice(index, 1, newColor);
  return arr;
}

let colorScheme = ['#878787', '#a08794', '#bb7e8c', '#c9b6be', '#d1becf'];

colorScheme = colorChange(colorScheme, 2, '#332327');
// we have removed '#bb7e8c' and added '#332327' in its place
// colorScheme now equals ['#878787', '#a08794', '#332327', '#c9b6be', '#d1becf']
```

This function takes an array of hex values, an index at which to remove an element, and the new color to replace the removed element with. The return value is an array containing a newly modified color scheme! While this example is a bit oversimplified, we can see the value that utilizing `splice()` to its maximum potential can have.

Instructions

We have defined a function, `htmlColorNames`, which takes an array of HTML colors as an argument. Modify the function using `splice()` to remove the first two elements of the array and add 'DarkSalmon' and 'BlanchedAlmond' in their respective places.

Challenge Seed

```
function htmlColorNames(arr) {
  // change code below this line

  // change code above this line
  return arr;
}

// do not change code below this line
console.log(htmlColorNames(['DarkGoldenRod', 'WhiteSmoke', 'LavenderBlush', 'PaleTurquoise',
  'FireBrick']));
```

Solution

```
// solution required
```

7. Copy Array Items Using `slice()`

Description

The next method we will cover is `slice()`. `slice()`, rather than modifying an array, copies, or *extracts*, a given number of elements to a new array, leaving the array it is called upon untouched. `slice()` takes only 2 parameters — the first is the index at which to begin extraction, and the second is the index at which to stop extraction (extraction will occur up to, but not including the element at this index). Consider this:

```
let weatherConditions = ['rain', 'snow', 'sleet', 'hail', 'clear'];

let todaysWeather = weatherConditions.slice(1, 3);
```

```
// todaysWeather equals ['snow', 'sleet'];
// weatherConditions still equals ['rain', 'snow', 'sleet', 'hail', 'clear']
```

In effect, we have created a new array by extracting elements from an existing array.

Instructions

We have defined a function, `forecast`, that takes an array as an argument. Modify the function using `slice()` to extract information from the argument array and return a new array that contains the elements 'warm' and 'sunny' .

Challenge Seed

```
function forecast(arr) {
  // change code below this line

  return arr;
}

// do not change code below this line
console.log(forecast(['cold', 'rainy', 'warm', 'sunny', 'cool', 'thunderstorms']));
```

Solution

```
// solution required
```

8. Copy an Array with the Spread Operator

Description

While `slice()` allows us to be selective about what elements of an array to copy, among several other useful tasks, ES6's new spread operator allows us to easily copy *all* of an array's elements, in order, with a simple and highly readable syntax. The spread syntax simply looks like this: ... In practice, we can use the spread operator to copy an array like so:

```
let thisArray = [true, true, undefined, false, null];
let thatArray = [...thisArray];
// thatArray equals [true, true, undefined, false, null]
// thisArray remains unchanged, and is identical to thatArray
```

Instructions

We have defined a function, `copyMachine` which takes `arr` (an array) and `num` (a number) as arguments. The function is supposed to return a new array made up of `num` copies of `arr`. We have done most of the work for you, but it doesn't work quite right yet. Modify the function using spread syntax so that it works correctly (hint: another method we have already covered might come in handy here!).

Challenge Seed

```
function copyMachine(arr, num) {
  let newArr = [];
  while (num >= 1) {
    // change code below this line

    // change code above this line
    num--;
  }
  return newArr;
}
```

```
// change code here to test different cases:  
console.log(copyMachine([true, false, true], 2));
```

Solution

```
// solution required
```

9. Combine Arrays with the Spread Operator

Description

Another huge advantage of the spread operator, is the ability to combine arrays, or to insert all the elements of one array into another, at any index. With more traditional syntaxes, we can concatenate arrays, but this only allows us to combine arrays at the end of one, and at the start of another. Spread syntax makes the following operation extremely simple:

```
let thisArray = ['sage', 'rosemary', 'parsley', 'thyme'];  
  
let thatArray = ['basil', 'cilantro', ...thisArray, 'coriander'];  
// thatArray now equals ['basil', 'cilantro', 'sage', 'rosemary', 'parsley', 'thyme', 'coriander']
```

Using spread syntax, we have just achieved an operation that would have been more complex and more verbose had we used traditional methods.

Instructions

We have defined a function `spreadOut` that returns the variable `sentence`. Modify the function using the spread operator so that it returns the array `['learning', 'to', 'code', 'is', 'fun']`.

Challenge Seed

```
function spreadOut() {  
  let fragment = ['to', 'code'];  
  let sentence; // change this line  
  return sentence;  
}  
  
// do not change code below this line  
console.log(spreadOut());
```

Solution

```
// solution required
```

10. Check For The Presence of an Element With `indexOf()`

Description

Since arrays can be changed, or *mutated*, at any time, there's no guarantee about where a particular piece of data will be on a given array, or if that element even still exists. Luckily, JavaScript provides us with another built-in method, `indexOf()`, that allows us to quickly and easily check for the presence of an element on an array. `indexOf()` takes an element as a parameter, and when called, it returns the position, or index, of that element, or `-1` if the element does not exist on the array. For example:

```
let fruits = ['apples', 'pears', 'oranges', 'peaches', 'pears'];

fruits.indexOf('dates') // returns -1
fruits.indexOf('oranges') // returns 2
fruits.indexOf('pears') // returns 1, the first index at which the element exists
```

Instructions

`indexOf()` can be incredibly useful for quickly checking for the presence of an element on an array. We have defined a function, `quickCheck`, that takes an array and an element as arguments. Modify the function using `indexOf()` so that it returns `true` if the passed element exists on the array, and `false` if it does not.

Challenge Seed

```
function quickCheck(arr, elem) {
  // change code below this line

  // change code above this line
}

// change code here to test different cases:
console.log(quickCheck(['squash', 'onions', 'shallots'], 'mushrooms'));
```

Solution

```
// solution required
```

11. Iterate Through All an Array's Items Using For Loops

Description

Sometimes when working with arrays, it is very handy to be able to iterate through each item to find one or more elements that we might need, or to manipulate an array based on which data items meet a certain set of criteria. JavaScript offers several built in methods that each iterate over arrays in slightly different ways to achieve different results (such as `every()`, `forEach()`, `map()`, etc.), however the technique which is most flexible and offers us the greatest amount of control is a simple `for` loop. Consider the following:

```
function greaterThanTen(arr) {
  let newArr = [];
  for (let i = 0; i < arr.length; i++) {
    if (arr[i] > 10) {
      newArr.push(arr[i]);
    }
  }
  return newArr;
}

greaterThanTen([2, 12, 8, 14, 80, 0, 1]);
// returns [12, 14, 80]
```

Using a `for` loop, this function iterates through and accesses each element of the array, and subjects it to a simple test that we have created. In this way, we have easily and programmatically determined which data items are greater than `10`, and returned a new array containing those items.

Instructions

We have defined a function, `filteredArray`, which takes `arr`, a nested array, and `elem` as arguments, and returns a new array. `elem` represents an element that may or may not be present on one or more of the arrays

nested within `arr`. Modify the function, using a `for` loop, to return a filtered version of the passed array such that any array nested within `arr` containing `elem` has been removed.

Challenge Seed

```
function filteredArray(arr, elem) {
  let newArr = [];
  // change code below this line

  // change code above this line
  return newArr;
}

// change code here to test different cases:
console.log(filteredArray([[3, 2, 3], [1, 6, 3], [3, 13, 26], [19, 3, 9]], 3));
```

Solution

```
// solution required
```

12. Create complex multi-dimensional arrays

Description

Awesome! You have just learned a ton about arrays! This has been a fairly high level overview, and there is plenty more to learn about working with arrays, much of which you will see in later sections. But before moving on to looking at Objects, lets take one more look, and see how arrays can become a bit more complex than what we have seen in previous challenges. One of the most powerful features when thinking of arrays as data structures, is that arrays can contain, or even be completely made up of other arrays. We have seen arrays that contain arrays in previous challenges, but fairly simple ones. However, arrays can contain an infinite depth of arrays that can contain other arrays, each with their own arbitrary levels of depth, and so on. In this way, an array can very quickly become very complex data structure, known as a multi-dimensional, or nested array. Consider the following example:

```
let nestedArray = [ // top, or first level - the outer most array
  ['deep'], // an array within an array, 2 levels of depth
  [
    ['deeper'], ['deeper'] // 2 arrays nested 3 levels deep
  ],
  [
    [
      ['deepest'], ['deepest'] // 2 arrays nested 4 levels deep
    ],
    [
      [
        ['deepest-est?'] // an array nested 5 levels deep
      ]
    ]
  ];
];
```

While this example may seem convoluted, this level of complexity is not unheard of, or even unusual, when dealing with large amounts of data. However, we can still very easily access the deepest levels of an array this complex with bracket notation:

```
console.log(nestedArray[2][1][0][0][0]);
// logs: deepest-est?
```

And now that we know where that piece of data is, we can reset it if we need to:

```

nestedArray[2][1][0][0][0] = 'deeper still';

console.log(nestedArray[2][1][0][0][0]);
// now logs: deeper still

```

Instructions

We have defined a variable, `myNestedArray`, set equal to an array. Modify `myNestedArray`, using any combination of strings, numbers, and booleans for data elements, so that it has exactly five levels of depth (remember, the outer-most array is level 1). Somewhere on the third level, include the string 'deep', on the fourth level, include the string 'deeper', and on the fifth level, include the string 'deepest' .

Challenge Seed

```

let myNestedArray = [
  // change code below this line
  ['unshift', false, 1, 2, 3, 'complex', 'nested'],
  ['loop', 'shift', 6, 7, 1000, 'method'],
  ['concat', false, true, 'spread', 'array'],
  ['mutate', 1327.98, 'splice', 'slice', 'push'],
  ['iterate', 1.3849, 7, '8.4876', 'arbitrary', 'depth']
  // change code above this line
];

```

Solution

```
// solution required
```

13. Add Key-Value Pairs to JavaScript Objects

Description

At their most basic, objects are just collections of key-value pairs, or in other words, pieces of data mapped to unique identifiers that we call properties or keys. Let's take a look at a very simple example:

```

let FCC_User = {
  username: 'awesome_coder',
  followers: 572,
  points: 1741,
  completedProjects: 15
};

```

The above code defines an object called `FCC_User` that has four properties, each of which map to a specific value. If we wanted to know the number of `followers` `FCC_User` has, we can access that property by writing:

```

let userData = FCC_User.followers;
// userData equals 572

```

This is called dot notation. Alternatively, we can also access the property with brackets, like so:

```

let userData = FCC_User['followers'];
// userData equals 572

```

Notice that with bracket notation, we enclosed `followers` in quotes. This is because the brackets actually allow us to pass a variable in to be evaluated as a property name (hint: keep this in mind for later!). Had we passed `followers` in without the quotes, the JavaScript engine would have attempted to evaluate it as a variable, and a `ReferenceError: followers is not defined` would have been thrown.

Instructions

Using the same syntax, we can also *add new key-value pairs to objects*. We've created a `foods` object with three entries. Add three more entries: bananas with a value of 13, grapes with a value of 35, and strawberries with a value of 27.

Challenge Seed

```
let foods = {
  apples: 25,
  oranges: 32,
  plums: 28
};

// change code below this line

// change code above this line

console.log(foods);
```

Solution

```
// solution required
```

14. Modify an Object Nested Within an Object

Description

Now let's take a look at a slightly more complex object. Object properties can be nested to an arbitrary depth, and their values can be any type of data supported by JavaScript, including arrays and even other objects.

Consider the following:

```
let nestedObject = {
  id: 28802695164,
  date: 'December 31, 2016',
  data: {
    totalUsers: 99,
    online: 80,
    onlineStatus: {
      active: 67,
      away: 13
    }
  }
};
```

`nestedObject` has three unique keys: `id`, whose value is a number, `date` whose value is a string, and `data`, whose value is an object which has yet another object nested within it. While structures can quickly become complex, we can still use the same notations to access the information we need.

Instructions

Here we've defined an object, `userActivity`, which includes another object nested within it. You can modify properties on this nested object in the same way you modified properties in the last challenge. Set the value of the `online` key to 45.

Challenge Seed

```
let userActivity = {
  id: 23894201352,
  date: 'January 1, 2017',
  data: {
```

```

        totalUsers: 51,
        online: 42
    }

    // change code below this line

    // change code above this line

    console.log(userActivity);

```

Solution

```
// solution required
```

15. Access Property Names with Bracket Notation

Description

In the first object challenge we mentioned the use of bracket notation as a way to access property values using the evaluation of a variable. For instance, imagine that our `foods` object is being used in a program for a supermarket cash register. We have some function that sets the `selectedFood` and we want to check our `foods` object for the presence of that food. This might look like:

```

let selectedFood = getCurrentFood(scannedItem);
let inventory = foods[selectedFood];

```

This code will evaluate the value stored in the `selectedFood` variable and return the value of that key in the `foods` object, or `undefined` if it is not present. Bracket notation is very useful because sometimes object properties are not known before runtime or we need to access them in a more dynamic way.

Instructions

We've defined a function, `checkInventory`, which receives a scanned item as an argument. Return the current value of the `scannedItem` key in the `foods` object. You can assume that only valid keys will be provided as an argument to `checkInventory`.

Challenge Seed

```

let foods = {
    apples: 25,
    oranges: 32,
    plums: 28,
    bananas: 13,
    grapes: 35,
    strawberries: 27
};
// do not change code above this line

function checkInventory(scannedItem) {
    // change code below this line

}

// change code below this line to test different cases:
console.log(checkInventory("apples"));

```

Solution

```
// solution required
```

16. Use the `delete` Keyword to Remove Object Properties

Description

Now you know what objects are and their basic features and advantages. In short, they are key-value stores which provide a flexible, intuitive way to structure data, *and*, they provide very fast lookup time. Throughout the rest of these challenges, we will describe several common operations you can perform on objects so you can become comfortable applying these useful data structures in your programs. In earlier challenges, we have both added to and modified an object's key-value pairs. Here we will see how we can *remove* a key-value pair from an object. Let's revisit our `foods` object example one last time. If we wanted to remove the `apples` key, we can remove it by using the `delete` keyword like this:

```
delete foods.apples;
```

Instructions

Use the `delete` keyword to remove the `oranges`, `plums`, and `strawberries` keys from the `foods` object.

Challenge Seed

```
let foods = {
  apples: 25,
  oranges: 32,
  plums: 28,
  bananas: 13,
  grapes: 35,
  strawberries: 27
};

// change code below this line

// change code above this line

console.log(foods);
```

Solution

```
// solution required
let foods = {
  apples: 25,
  oranges: 32,
  plums: 28,
  bananas: 13,
  grapes: 35,
  strawberries: 27
};

delete foods.oranges;
delete foods.plums;
delete foods.strawberries;

console.log(foods);
```

17. Check if an Object has a Property

Description

Now we can add, modify, and remove keys from objects. But what if we just wanted to know if an object has a specific property? JavaScript provides us with two different ways to do this. One uses the `hasOwnProperty()`

method and the other uses the `in` keyword. If we have an object `users` with a property of `Alan`, we could check for its presence in either of the following ways:

```
users.hasOwnProperty('Alan');
'Alan' in users;
// both return true
```

Instructions

We've created an object, `users`, with some users in it and a function `isEveryoneHere`, which we pass the `users` object to as an argument. Finish writing this function so that it returns `true` only if the `users` object contains all four names, `Alan`, `Jeff`, `Sarah`, and `Ryan`, as keys, and `false` otherwise.

Challenge Seed

```
let users = {
  Alan: {
    age: 27,
    online: true
  },
  Jeff: {
    age: 32,
    online: true
  },
  Sarah: {
    age: 48,
    online: true
  },
  Ryan: {
    age: 19,
    online: true
  }
};

function isEveryoneHere(obj) {
  // change code below this line

  // change code above this line
}

console.log(isEveryoneHere(users));
```

Solution

```
let users = {
  Alan: {
    age: 27,
    online: true
  },
  Jeff: {
    age: 32,
    online: true
  },
  Sarah: {
    age: 48,
    online: true
  },
  Ryan: {
    age: 19,
    online: true
  }
};

function isEveryoneHere(obj) {
  return [
    'Alan',
    'Jeff',
    'Sarah',
    'Ryan'
  ].every(i => obj.hasOwnProperty(i));
```

```
}
```

```
console.log(isEveryoneHere(users));
```

18. Iterate Through the Keys of an Object with a for...in Statement

Description

Sometimes you may need to iterate through all the keys within an object. This requires a specific syntax in JavaScript called a `for...in` statement. For our `users` object, this could look like:

```
for (let user in users) {
  console.log(user);
}

// logs:
Alan
Jeff
Sarah
Ryan
```

In this statement, we defined a variable `user`, and as you can see, this variable was reset during each iteration to each of the object's keys as the statement looped through the object, resulting in each user's name being printed to the console. NOTE:

Objects do not maintain an ordering to stored keys like arrays do; thus a key's position on an object, or the relative order in which it appears, is irrelevant when referencing or accessing that key.

Instructions

We've defined a function, `countOnline`; use a `for...in` statement within this function to loop through the `users` in the `users` object and return the number of users whose `online` property is set to `true`.

Challenge Seed

```
let users = {
  Alan: {
    age: 27,
    online: false
  },
  Jeff: {
    age: 32,
    online: true
  },
  Sarah: {
    age: 48,
    online: false
  },
  Ryan: {
    age: 19,
    online: true
  }
};

function countOnline(obj) {
  // change code below this line

  // change code above this line
}

console.log(countOnline(users));
```

Solution

```

let users = {
  Alan: {
    age: 27,
    online: false
  },
  Jeff: {
    age: 32,
    online: true
  },
  Sarah: {
    age: 48,
    online: false
  },
  Ryan: {
    age: 19,
    online: true
  }
};

function countOnline(obj) {
  let online = 0;
  for(let user in obj){
    if(obj[user].online == true) {
      online += 1;
    }
  }
  return online;
}

console.log(countOnline(users));

```

19. Generate an Array of All Object Keys with Object.keys()

Description

We can also generate an array which contains all the keys stored in an object using the `Object.keys()` method and passing in an object as the argument. This will return an array with strings representing each property in the object. Again, there will be no specific order to the entries in the array.

Instructions

Finish writing the `getArrayOfUsers` function so that it returns an array containing all the properties in the object it receives as an argument.

Challenge Seed

```

let users = {
  Alan: {
    age: 27,
    online: false
  },
  Jeff: {
    age: 32,
    online: true
  },
  Sarah: {
    age: 48,
    online: false
  },
  Ryan: {
    age: 19,
    online: true
  }
};

```

```

    }

}

function getArrayOfUsers(obj) {
  // change code below this line

  // change code above this line
}

console.log(getArrayOfUsers(users));

```

Solution

```

let users = {
  Alan: {
    age: 27,
    online: false
  },
  Jeff: {
    age: 32,
    online: true
  },
  Sarah: {
    age: 48,
    online: false
  },
  Ryan: {
    age: 19,
    online: true
  }
};

function getArrayOfUsers(obj) {
  return Object.keys(users);
}

console.log(getArrayOfUsers(users));

```

20. Modify an Array Stored in an Object

Description

Now you've seen all the basic operations for JavaScript objects. You can add, modify, and remove key-value pairs, check if keys exist, and iterate over all the keys in an object. As you continue learning JavaScript you will see even more versatile applications of objects. Additionally, the optional Advanced Data Structures lessons later in the curriculum also cover the ES6 Map and Set objects, both of which are similar to ordinary objects but provide some additional features. Now that you've learned the basics of arrays and objects, you're fully prepared to begin tackling more complex problems using JavaScript!

Instructions

Take a look at the object we've provided in the code editor. The `user` object contains three keys. The `data` key contains five keys, one of which contains an array of `friends`. From this, you can see how flexible objects are as data structures. We've started writing a function `addFriend`. Finish writing it so that it takes a `user` object and adds the name of the `friend` argument to the array stored in `user.data.friends` and returns that array.

Challenge Seed

```

let user = {
  name: 'Kenneth',
  age: 28,
  data: {
    username: 'kennethCodesAllDay',

```

```

joinDate: 'March 26, 2016',
organization: 'freeCodeCamp',
friends: [
  'Sam',
  'Kira',
  'Tomo'
],
location: {
  city: 'San Francisco',
  state: 'CA',
  country: 'USA'
}
};

function addFriend(userObj, friend) {
  // change code below this line

  // change code above this line
}

console.log(addFriend(user, 'Pete'));

```

Solution

```
// solution required
```

Basic Algorithm Scripting

1. Convert Celsius to Fahrenheit

Description

The algorithm to convert from Celsius to Fahrenheit is the temperature in Celsius times $\frac{9}{5}$, plus 32. You are given a variable `celsius` representing a temperature in Celsius. Use the variable `fahrenheit` already defined and assign it the Fahrenheit temperature equivalent to the given Celsius temperature. Use the algorithm mentioned above to help convert the Celsius temperature to Fahrenheit. Don't worry too much about the function and return statements as they will be covered in future challenges. For now, only use operators that you have already learned.

Instructions

Challenge Seed

```

function convertToF(celsius) {
  let fahrenheit;
  return fahrenheit;
}

convertToF(30);

```

Solution

```

function convertToF(celsius) {
  let fahrenheit = celsius * 9/5 + 32;

  return fahrenheit;
}

```

```
convertToF(30);
```

2. Reverse a String

Description

Reverse the provided string. You may need to turn the string into an array before you can reverse it. Your result must be a string. Remember to use [Read-Search-Ask](#) if you get stuck. Write your own code.

Instructions

Challenge Seed

```
function reverseString(str) {  
    return str;  
}  
  
reverseString("hello");
```

Solution

```
function reverseString(str) {  
    return str.split('').reverse().join('');  
}  
  
reverseString("hello");
```

3. Factorialize a Number

Description

Return the factorial of the provided integer. If the integer is represented with the letter n, a factorial is the product of all positive integers less than or equal to n. Factorials are often represented with the shorthand notation $n!$. For example: $5! = 1 * 2 * 3 * 4 * 5 = 120$. Only integers greater than or equal to zero will be supplied to the function. Remember to use [Read-Search-Ask](#) if you get stuck. Write your own code.

Instructions

Challenge Seed

```
function factorialize(num) {  
    return num;  
}  
  
factorialize(5);
```

Solution

```
function factorialize(num) {  
    return num < 1 ? 1 : num * factorialize(num - 1);  
}
```

```
factorialize(5);
```

4. Find the Longest Word in a String

Description

Return the length of the longest word in the provided sentence. Your response should be a number. Remember to use [Read-Search-Ask](#) if you get stuck. Write your own code.

Instructions

Challenge Seed

```
function findLongestWordLength(str) {
  return str.length;
}

findLongestWordLength("The quick brown fox jumped over the lazy dog");
```

Solution

```
function findLongestWordLength(str) {
  return str.split(' ').sort((a, b) => b.length - a.length)[0].length;
}

findLongestWordLength("The quick brown fox jumped over the lazy dog");
```

5. Return Largest Numbers in Arrays

Description

Return an array consisting of the largest number from each provided sub-array. For simplicity, the provided array will contain exactly 4 sub-arrays. Remember, you can iterate through an array with a simple for loop, and access each member with array syntax `arr[i]`. Remember to use [Read-Search-Ask](#) if you get stuck. Write your own code.

Instructions

Challenge Seed

```
function largestOfFour(arr) {
  // You can do this!
  return arr;
}

largestOfFour([[4, 5, 1, 3], [13, 27, 18, 26], [32, 35, 37, 39], [1000, 1001, 857, 1]]);
```

Solution

```
function largestOfFour(arr) {
  return arr.map(subArr => Math.max(...subArr));
}
```

```
largestOfFour([[4, 5, 1, 3], [13, 27, 18, 26], [32, 35, 37, 39], [1000, 1001, 857, 1]]);
```

6. Confirm the Ending

Description

Check if a string (first argument, `str`) ends with the given target string (second argument, `target`). This challenge *can* be solved with the `.endsWith()` method, which was introduced in ES2015. But for the purpose of this challenge, we would like you to use one of the JavaScript substring methods instead. Remember to use [Read-Search-Ask](#) if you get stuck. Write your own code.

Instructions

Challenge Seed

```
function confirmEnding(str, target) {
  // "Never give up and good luck will find you."
  // -- Falcor
  return str;
}

confirmEnding("Bastian", "n");
```

Solution

```
function confirmEnding(str, target) {
  return str.substring(str.length - target.length) === target;
}

confirmEnding("Bastian", "n");
```

7. Repeat a String Repeat a String

Description

Repeat a given string `str` (first argument) for `num` times (second argument). Return an empty string if `num` is not a positive number. Remember to use [Read-Search-Ask](#) if you get stuck. Write your own code.

Instructions

Challenge Seed

```
function repeatStringNumTimes(str, num) {
  // repeat after me
  return str;
}

repeatStringNumTimes("abc", 3);
```

Solution

```

function repeatStringNumTimes(str, num) {
  if (num < 0) return '';
  return num === 1 ? str : str + repeatStringNumTimes(str, num-1);
}

repeatStringNumTimes("abc", 3);

```

8. Truncate a String

Description

Truncate a string (first argument) if it is longer than the given maximum string length (second argument). Return the truncated string with a ... ending. Remember to use [Read-Search-Ask](#) if you get stuck. Write your own code.

Instructions

Challenge Seed

```

function truncateString(str, num) {
  // Clear out that junk in your trunk
  return str;
}

truncateString("A-tisket a-tasket A green and yellow basket", 8);

```

Solution

```

function truncateString(str, num) {
  if (num >= str.length) {
    return str;
  }

  return str.slice(0, num) + "...";
}

truncateString("A-tisket a-tasket A green and yellow basket", 8);

```

9. Finders Keepers

Description

Create a function that looks through an array (first argument) and returns the first element in the array that passes a truth test (second argument). If no element passes the test, return undefined. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```

function findElement(arr, func) {
  let num = 0;
  return num;
}

findElement([1, 2, 3, 4], num => num % 2 === 0);

```

Solution

```
function findElement(arr, func) {
  return arr.filter(func)[0];
}

findElement([1, 2, 3, 4], num => num % 2 === 0);
```

10. Boo who

Description

Check if a value is classified as a boolean primitive. Return true or false. Boolean primitives are true and false. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```
function booWho(bool) {
  // What is the new fad diet for ghost developers? The Boolean.
  return bool;
}

booWho(null);
```

Solution

```
function booWho(bool) {
  return typeof bool === "boolean";
}

booWho(null);
```

11. Title Case a Sentence

Description

Return the provided string with the first letter of each word capitalized. Make sure the rest of the word is in lower case. For the purpose of this exercise, you should also capitalize connecting words like "the" and "of". Remember to use [Read-Search-Ask](#) if you get stuck. Write your own code.

Instructions

Challenge Seed

```
function titleCase(str) {
  return str;
}

titleCase("I'm a little tea pot");
```

Solution

```
function titleCase(str) {
  return str.split(' ').map(word => word.charAt(0).toUpperCase() +
  word.substring(1).toLowerCase()).join(' ');
}

titleCase("I'm a little tea pot");
```

12. Slice and Splice

Description

You are given two arrays and an index. Use the array methods `slice` and `splice` to copy each element of the first array into the second array, in order. Begin inserting elements at index `n` of the second array. Return the resulting array. The input arrays should remain the same after the function runs. Remember to use [Read-Search-Ask](#) if you get stuck. Write your own code.

Instructions

Challenge Seed

```
function frankenSplice(arr1, arr2, n) {
  // It's alive. It's alive!
  return arr2;
}

frankenSplice([1, 2, 3], [4, 5, 6], 1);
```

After Test

```
let testArr1 = [1, 2];
let testArr2 = ["a", "b"];
```

Solution

```
function frankenSplice(arr1, arr2, n) {
  // It's alive. It's alive!
  let result = arr2.slice();
  for (let i = 0; i < arr1.length; i++) {
    result.splice(n+i, 0, arr1[i]);
  }
  return result;
}

frankenSplice([1, 2, 3], [4, 5], 1);
```

13. Falsy Bouncer

Description

Remove all falsy values from an array. Falsy values in JavaScript are `false`, `null`, `0`, `""`, `undefined`, and `NaN`. Hint: Try converting each value to a Boolean. Remember to use [Read-Search-Ask](#) if you get stuck. Write your own code.

Instructions

Challenge Seed

```
function bouncer(arr) {
  // Don't show a false ID to this bouncer.
  return arr;
}

bouncer([7, "ate", "", false, 9]);
```

Solution

```
function bouncer(arr) {
  return arr.filter(e => e);
}

bouncer([7, "ate", "", false, 9]);
```

14. Where do I Belong

Description

Return the lowest index at which a value (second argument) should be inserted into an array (first argument) once it has been sorted. The returned value should be a number. For example, `getIndexToIns([1,2,3,4], 1.5)` should return `1` because it is greater than `1` (index 0), but less than `2` (index 1). Likewise, `getIndexToIns([20,3,5], 19)` should return `2` because once the array has been sorted it will look like `[3,5,20]` and `19` is less than `20` (index 2) and greater than `5` (index 1). Remember to use [Read-Search-Ask](#) if you get stuck. Write your own code.

Instructions

Challenge Seed

```
function getIndexToIns(arr, num) {
  // Find my place in this sorted array.
  return num;
}

getIndexToIns([40, 60], 50);
```

Solution

```
function getIndexToIns(arr, num) {
  arr = arr.sort((a, b) => a - b);

  for (let i = 0; i < arr.length; i++) {
    if (arr[i] >= num) {
      return i;
    }
  }

  return arr.length;
}

getIndexToIns([40, 60], 50);
```

15. Mutations

Description

Return true if the string in the first element of the array contains all of the letters of the string in the second element of the array. For example, `["hello", "Hello"]`, should return true because all of the letters in the second string are present in the first, ignoring case. The arguments `["hello", "hey"]` should return false because the string "hello" does not contain a "y". Lastly, `["Alien", "line"]`, should return true because all of the letters in "line" are present in "Alien". Remember to use [Read-Search-Ask](#) if you get stuck. Write your own code.

Instructions

Challenge Seed

```
function mutation(arr) {
  return arr;
}

mutation(["hello", "hey"]);
```

Solution

```
function mutation(arr) {
  let hash = Object.create(null);

  arr[0].toLowerCase().split('').forEach(c => hash[c] = true);

  return !arr[1].toLowerCase().split('').filter(c => !hash[c]).length;
}

mutation(["hello", "hey"]);
```

16. Chunky Monkey

Description

Write a function that splits an array (first argument) into groups the length of `size` (second argument) and returns them as a two-dimensional array. Remember to use [Read-Search-Ask](#) if you get stuck. Write your own code.

Instructions

Challenge Seed

```
function chunkArrayInGroups(arr, size) {
  // Break it up.
  return arr;
}

chunkArrayInGroups(["a", "b", "c", "d"], 2);
```

Solution

```
function chunkArrayInGroups(arr, size) {
  let out = [];

  for (let i = 0; i < arr.length; i += size) {
    out.push(arr.slice(i, i + size));
  }

  return out;
}
```

```

for (let i = 0; i < arr.length; i += size) {
  out.push(arr.slice(i, i + size));
}

return out;
}

chunkArrayInGroups(["a", "b", "c", "d"], 2);

```

Object Oriented Programming

1. Create a Basic JavaScript Object

Description

Think about things people see everyday, like cars, shops, and birds. These are all objects : tangible things people can observe and interact with. What are some qualities of these objects ? A car has wheels. Shops sell items. Birds have wings. These qualities, or properties , define what makes up an object . Note that similar objects share the same properties , but may have different values for those properties . For example, all cars have wheels, but not all cars have the same number of wheels. Objects in JavaScript are used to model real-world objects, giving them properties and behavior just like their real-world counterparts. Here's an example using these concepts to create a duck object :

```

let duck = {
  name: "Aflac",
  numLegs: 2
};

```

This duck object has two property/value pairs: a name of "Aflac" and a numLegs of 2.

Instructions

Create a dog object with name and numLegs properties, and set them to a string and a number, respectively.

Challenge Seed

```

let dog = {
};

```

Solution

```

let dog = {
  name: '',
  numLegs: 4
};

```

2. Use Dot Notation to Access the Properties of an Object

Description

The last challenge created an object with various properties , now you'll see how to access the values of those properties . Here's an example:

```
let duck = {
  name: "Aflac",
  numLegs: 2
};
console.log(duck.name);
// This prints "Aflac" to the console
```

Dot notation is used on the object `name`, `duck`, followed by the name of the property, `name`, to access the value of "Aflac".

Instructions

Print both properties of the dog object below to your console.

Challenge Seed

```
let dog = {
  name: "Spot",
  numLegs: 4
};
// Add your code below this line
```

Solution

```
let dog = {
  name: "Spot",
  numLegs: 4
};
console.log(dog.name);
console.log(dog.numLegs);
```

3. Create a Method on an Object

Description

Objects can have a special type of property, called a method. Methods are properties that are functions. This adds different behavior to an object. Here is the duck example with a method:

```
let duck = {
  name: "Aflac",
  numLegs: 2,
  sayName: function() {return "The name of this duck is " + duck.name + ".`}
};
duck.sayName();
// Returns "The name of this duck is Aflac."
```

The example adds the `sayName` method, which is a function that returns a sentence giving the name of the duck. Notice that the method accessed the `name` property in the return statement using `duck.name`. The next challenge will cover another way to do this.

Instructions

Using the `dog` object, give it a method called `sayLegs`. The method should return the sentence "This dog has 4 legs."

Challenge Seed

```
let dog = {
  name: "Spot",
  numLegs: 4,
};

dog.sayLegs();
```

Solution

```
let dog = {
  name: "Spot",
  numLegs: 4,
  sayLegs () {
    return 'This dog has ' + this.numLegs + ' legs.';
  }
};

dog.sayLegs();
```

4. Make Code More Reusable with the this Keyword

Description

The last challenge introduced a method `to the duck object`. It used `duck.name` dot notation to access the value for the `name` property within the return statement: `sayName: function() {return "The name of this duck is " + duck.name + ".;}"` While this is a valid way to access the object's property, there is a pitfall here. If the variable `name` changes, any code referencing the original name would need to be updated as well. In a short object definition, it isn't a problem, but if an object has many references to its properties there is a greater chance for error. A way to avoid these issues is with the `this` keyword:

```
let duck = {
  name: "Aflac",
  numLegs: 2,
  sayName: function() {return "The name of this duck is " + this.name + ".;"}
};
```

`this` is a deep topic, and the above example is only one way to use it. In the current context, `this` refers to the object that the method is associated with: `duck`. If the object's name is changed to `mallard`, it is not necessary to find all the references to `duck` in the code. It makes the code reusable and easier to read.

Instructions

Modify the `dog.sayLegs` method to remove any references to `dog`. Use the `duck` example for guidance.

Challenge Seed

```
let dog = {
  name: "Spot",
  numLegs: 4,
  sayLegs: function() {return "This dog has " + dog.numLegs + " legs.";}
};

dog.sayLegs();
```

Solution

```
let dog = {
  name: "Spot",
  numLegs: 4,
```

```

sayLegs () {
    return 'This dog has ' + this.numLegs + ' legs.';
}

dog.sayLegs();

```

5. Define a Constructor Function

Description

Constructors are functions that create new objects. They define properties and behaviors that will belong to the new object. Think of them as a blueprint for the creation of new objects. Here is an example of a constructor :

```

function Bird() {
    this.name = "Albert";
    this.color = "blue";
    this.numLegs = 2;
}

```

This constructor defines a `Bird` object with properties `name`, `color`, and `numLegs` set to `Albert`, `blue`, and `2`, respectively. Constructors follow a few conventions:

- Constructors are defined with a capitalized name to distinguish them from other functions that are not constructors .
- Constructors use the keyword `this` to set properties of the object they will create. Inside the constructor , `this` refers to the new object it will create.
- Constructors define properties and behaviors instead of returning a value as other functions might.

Instructions

Create a constructor , `Dog` , with properties `name` , `color` , and `numLegs` that are set to a string, a string, and a number, respectively.

Challenge Seed

Solution

```

function Dog (name, color, numLegs) {
    this.name = 'name';
    this.color = 'color';
    this.numLegs = 4;
}

```

6. Use a Constructor to Create Objects

Description

Here's the `Bird` constructor from the previous challenge:

```

function Bird() {
    this.name = "Albert";
    this.color = "blue";
    this.numLegs = 2;
    // "this" inside the constructor always refers to the object being created
}

```

```
let blueBird = new Bird();
```

Notice that the `new` operator is used when calling a constructor. This tells JavaScript to create a new instance of `Bird` called `blueBird`. Without the `new` operator, this inside the constructor would not point to the newly created object, giving unexpected results. Now `blueBird` has all the properties defined inside the `Bird` constructor:

```
blueBird.name; // => Albert
blueBird.color; // => blue
blueBird.numLegs; // => 2
```

Just like any other object, its properties can be accessed and modified:

```
blueBird.name = 'Elvira';
blueBird.name; // => Elvira
```

Instructions

Use the `Dog` constructor from the last lesson to create a new instance of `Dog`, assigning it to a variable `hound`.

Challenge Seed

```
function Dog() {
  this.name = "Rupert";
  this.color = "brown";
  this.numLegs = 4;
}
// Add your code below this line
```

Solution

```
function Dog() {
  this.name = "Rupert";
  this.color = "brown";
  this.numLegs = 4;
}
const hound = new Dog();
```

7. Extend Constructors to Receive Arguments

Description

The `Bird` and `Dog` constructors from last challenge worked well. However, notice that all `Birds` that are created with the `Bird` constructor are automatically named `Albert`, are blue in color, and have two legs. What if you want birds with different values for name and color? It's possible to change the properties of each bird manually but that would be a lot of work:

```
let swan = new Bird();
swan.name = "Carlos";
swan.color = "white";
```

Suppose you were writing a program to keep track of hundreds or even thousands of different birds in an aviary. It would take a lot of time to create all the birds, then change the properties to different values for every one. To more easily create different `Bird` objects, you can design your `Bird` constructor to accept parameters:

```
function Bird(name, color) {
  this.name = name;
  this.color = color;
  this.numLegs = 2;
}
```

Then pass in the values as arguments to define each unique bird into the `Bird` constructor: `let cardinal = new Bird("Bruce", "red");` This gives a new instance of `Bird` with name and color properties set to Bruce and red, respectively. The `numLegs` property is still set to 2. The `cardinal` has these properties:

```
cardinal.name // => Bruce
cardinal.color // => red
cardinal.numLegs // => 2
```

The constructor is more flexible. It's now possible to define the properties for each `Bird` at the time it is created, which is one way that JavaScript constructors are so useful. They group objects together based on shared characteristics and behavior and define a blueprint that automates their creation.

Instructions

Create another `Dog` constructor. This time, set it up to take the parameters `name` and `color`, and have the property `numLegs` fixed at 4. Then create a new `Dog` saved in a variable `terrier`. Pass it two strings as arguments for the `name` and `color` properties.

Challenge Seed

```
function Dog() {  
}
```

Solution

```
function Dog (name, color) {  
    this.numLegs = 4;  
    this.name = name;  
    this.color = color;  
}  
  
const terrier = new Dog();
```

8. Verify an Object's Constructor with `instanceof`

Description

Anytime a constructor function creates a new object, that object is said to be an `instance` of its constructor. JavaScript gives a convenient way to verify this with the `instanceof` operator. `instanceof` allows you to compare an object to a constructor, returning `true` or `false` based on whether or not that object was created with the constructor. Here's an example:

```
let Bird = function(name, color) {  
    this.name = name;  
    this.color = color;  
    this.numLegs = 2;  
}  
  
let crow = new Bird("Alexis", "black");  
  
crow instanceof Bird; // => true
```

If an object is created without using a constructor, `instanceof` will verify that it is not an instance of that constructor:

```
let canary = {  
    name: "Mildred",  
    color: "Yellow",  
    numLegs: 2
```

```
};

canary instanceof Bird; // => false
```

Instructions

Create a new instance of the `House` constructor, calling it `myHouse` and passing a number of bedrooms. Then, use `instanceof` to verify that it is an instance of `House`.

Challenge Seed

```
/* jshint expr: true */

function House(numBedrooms) {
  this.numBedrooms = numBedrooms;
}

// Add your code below this line
```

Solution

```
function House(numBedrooms) {
  this.numBedrooms = numBedrooms;
}
const myHouse = new House(4);
console.log(myHouse instanceof House);
```

9. Understand Own Properties

Description

In the following example, the `Bird` constructor defines two properties: `name` and `numLegs`:

```
function Bird(name) {
  this.name = name;
  this.numLegs = 2;
}

let duck = new Bird("Donald");
let canary = new Bird("Tweety");
```

`name` and `numLegs` are called `own properties`, because they are defined directly on the instance object. That means that `duck` and `canary` each has its own separate copy of these properties. In fact every instance of `Bird` will have its own copy of these properties. The following code adds all of the `own properties` of `duck` to the array `ownProps`:

```
let ownProps = [];

for (let property in duck) {
  if(duck.hasOwnProperty(property)) {
    ownProps.push(property);
  }
}

console.log(ownProps); // prints [ "name", "numLegs" ]
```

Instructions

Add the own properties of canary to the array ownProps .

Challenge Seed

```
function Bird(name) {
  this.name = name;
  this.numLegs = 2;
}

let canary = new Bird("Tweety");
let ownProps = [];
// Add your code below this line
```

Solution

```
function Bird(name) {
  this.name = name;
  this.numLegs = 2;
}

let canary = new Bird("Tweety");
function getOwnProps (obj) {
  const props = [];

  for (let prop in obj) {
    if (obj.hasOwnProperty(prop)) {
      props.push(prop);
    }
  }

  return props;
}

const ownProps = getOwnProps(canary);
```

10. Use Prototype Properties to Reduce Duplicate Code

Description

Since numLegs will probably have the same value for all instances of Bird , you essentially have a duplicated variable numLegs inside each Bird instance. This may not be an issue when there are only two instances, but imagine if there are millions of instances. That would be a lot of duplicated variables. A better way is to use Bird's prototype . The prototype is an object that is shared among ALL instances of Bird . Here's how to add numLegs to the Bird prototype :

```
Bird.prototype.numLegs = 2;
```

Now all instances of Bird have the numLegs property.

```
console.log(duck.numLegs); // prints 2
console.log(canary.numLegs); // prints 2
```

Since all instances automatically have the properties on the prototype , think of a prototype as a "recipe" for creating objects. Note that the prototype for duck and canary is part of the Bird constructor as Bird.prototype . Nearly every object in JavaScript has a prototype property which is part of the constructor function that created it.

Instructions

Add a numLegs property to the prototype of Dog

Challenge Seed

```
function Dog(name) {
  this.name = name;
}

// Add your code above this line
let beagle = new Dog("Snoopy");
```

Solution

```
function Dog (name) {
  this.name = name;
}
Dog.prototype.numLegs = 4;
let beagle = new Dog("Snoopy");
```

11. Iterate Over All Properties

Description

You have now seen two kinds of properties: `own properties` and `prototype properties`. `Own properties` are defined directly on the object instance itself. `And prototype properties` are defined on the `prototype`.

```
function Bird(name) {
  this.name = name; //own property
}

Bird.prototype.numLegs = 2; // prototype property

let duck = new Bird("Donald");
```

Here is how you add `duck`'s own properties to the array `ownProps` and prototype properties to the array `prototypeProps`:

```
let ownProps = [];
let prototypeProps = [];

for (let property in duck) {
  if(duck.hasOwnProperty(property)) {
    ownProps.push(property);
  } else {
    prototypeProps.push(property);
  }
}

console.log(ownProps); // prints ["name"]
console.log(prototypeProps); // prints ["numLegs"]
```

Instructions

Add all of the `own properties` of `beagle` to the array `ownProps`. Add all of the `prototype properties` of `Dog` to the array `prototypeProps`.

Challenge Seed

```
function Dog(name) {
  this.name = name;
```

```

}

Dog.prototype.numLegs = 4;

let beagle = new Dog("Snoopy");

let ownProps = [];
let prototypeProps = [];

// Add your code below this line

```

Solution

```

function Dog(name) {
  this.name = name;
}

Dog.prototype.numLegs = 4;

let beagle = new Dog("Snoopy");

let ownProps = [];
let prototypeProps = [];
for (let prop in beagle) {
  if (beagle.hasOwnProperty(prop)) {
    ownProps.push(prop);
  } else {
    prototypeProps.push(prop);
  }
}

```

12. Understand the Constructor Property

Description

There is a special `constructor` property located on the object instances `duck` and `beagle` that were created in the previous challenges:

```

let duck = new Bird();
let beagle = new Dog();

console.log(duck.constructor === Bird); //prints true
console.log(beagle.constructor === Dog); //prints true

```

Note that the `constructor` property is a reference to the constructor function that created the instance. The advantage of the `constructor` property is that it's possible to check for this property to find out what kind of object it is. Here's an example of how this could be used:

```

function joinBirdFraternity(candidate) {
  if (candidate.constructor === Bird) {
    return true;
  } else {
    return false;
  }
}

```

Note

Since the `constructor` property can be overwritten (which will be covered in the next two challenges) it's generally better to use the `instanceof` method to check the type of an object.

Instructions

Write a `joinDogFraternity` function that takes a `candidate` parameter and, using the `constructor` property, return `true` if the candidate is a `Dog`, otherwise return `false`.

Challenge Seed

```
function Dog(name) {
  this.name = name;
}

// Add your code below this line
function joinDogFraternity(candidate) {

}
```

Solution

```
function Dog(name) {
  this.name = name;
}
function joinDogFraternity(candidate) {
  return candidate.constructor === Dog;
}
```

13. Change the Prototype to a New Object

Description

Up until now you have been adding properties to the `prototype` individually:

```
Bird.prototype.numLegs = 2;
```

This becomes tedious after more than a few properties.

```
Bird.prototype.eat = function() {
  console.log("nom nom nom");
}
```

```
Bird.prototype.describe = function() {
  console.log("My name is " + this.name);
}
```

A more efficient way is to set the `prototype` to a new object that already contains the properties. This way, the properties are added all at once:

```
Bird.prototype = {
  numLegs: 2,
  eat: function() {
    console.log("nom nom nom");
  },
  describe: function() {
    console.log("My name is " + this.name);
  }
};
```

Instructions

Add the property `numLegs` and the two methods `eat()` and `describe()` to the prototype of `Dog` by setting the `prototype` to a new object.

Challenge Seed

```

function Dog(name) {
  this.name = name;
}

Dog.prototype = {
  // Add your code below this line

};

```

Solution

```

function Dog(name) {
  this.name = name;
}
Dog.prototype = {
  numLegs: 4,
  eat () {
    console.log('nom nom nom');
  },
  describe () {
    console.log('My name is ' + this.name);
  }
};

```

14. Remember to Set the Constructor Property when Changing the Prototype

Description

There is one crucial side effect of manually setting the prototype to a new object. It erases the `constructor` property! This property can be used to check which constructor function created the instance, but since the property has been overwritten, it now gives false results:

```

duck.constructor === Bird; // false -- Oops
duck.constructor === Object; // true, all objects inherit from Object.prototype
duck instanceof Bird; // true, still works

```

To fix this, whenever a prototype is manually set to a new object, remember to define the `constructor` property:

```

Bird.prototype = {
  constructor: Bird, // define the constructor property
  numLegs: 2,
  eat: function() {
    console.log("nom nom nom");
  },
  describe: function() {
    console.log("My name is " + this.name);
  }
};

```

Instructions

Define the `constructor` property on the `Dog` prototype.

Challenge Seed

```

function Dog(name) {
  this.name = name;
}

// Modify the code below this line

```

```
Dog.prototype = {

  numLegs: 4,
  eat: function() {
    console.log("nom nom nom");
  },
  describe: function() {
    console.log("My name is " + this.name);
  }
};
```

Solution

```
function Dog(name) {
  this.name = name;
}
Dog.prototype = {
  constructor: Dog,
  numLegs: 4,
  eat: function() {
    console.log("nom nom nom");
  },
  describe: function() {
    console.log("My name is " + this.name);
  }
};
```

15. Understand Where an Object's Prototype Comes From

Description

Just like people inherit genes from their parents, an object inherits its prototype directly from the constructor function that created it. For example, here the `Bird` constructor creates the `duck` object:

```
function Bird(name) {
  this.name = name;
}
```

```
let duck = new Bird("Donald");
```

duck inherits its prototype from the `Bird` constructor function. You can show this relationship with the `isPrototypeOf` method:

```
Bird.prototype.isPrototypeOf(duck);
// returns true
```

Instructions

Use `isPrototypeOf` to check the prototype of `beagle`.

Challenge Seed

```
function Dog(name) {
  this.name = name;
}

let beagle = new Dog("Snoopy");

// Add your code below this line
```

Solution

```
function Dog(name) {
  this.name = name;
}
let beagle = new Dog("Snoopy");
Dog.prototype.isPrototypeOf(beagle);
```

16. Understand the Prototype Chain

Description

All objects in JavaScript (with a few exceptions) have a `prototype`. Also, an object's `prototype` itself is an object.

```
function Bird(name) {
  this.name = name;
}

typeof Bird.prototype; // => object
```

Because a `prototype` is an object, a `prototype` can have its own `prototype`! In this case, the `prototype` of `Bird.prototype` is `Object.prototype`:

```
Object.prototype.isPrototypeOf(Bird.prototype);
// returns true
```

How is this useful? You may recall the `hasOwnProperty` method from a previous challenge:

```
let duck = new Bird("Donald");
duck.hasOwnProperty("name"); // => true
```

The `hasOwnProperty` method is defined in `Object.prototype`, which can be accessed by `Bird.prototype`, which can then be accessed by `duck`. This is an example of the prototype chain. In this prototype chain, `Bird` is the supertype for `duck`, while `duck` is the subtype. `Object` is a supertype for both `Bird` and `duck`. `Object` is a supertype for all objects in JavaScript. Therefore, any object can use the `hasOwnProperty` method.

Instructions

Modify the code to show the correct prototype chain.

Challenge Seed

```
function Dog(name) {
  this.name = name;
}

let beagle = new Dog("Snoopy");

Dog.prototype.isPrototypeOf(beagle); // => true

// Fix the code below so that it evaluates to true
???.isPrototypeOf(Dog.prototype);
```

Solution

```
function Dog(name) {
  this.name = name;
}
let beagle = new Dog("Snoopy");
Dog.prototype.isPrototypeOf(beagle);
Object.prototype.isPrototypeOf(Dog.prototype);
```

17. Use Inheritance So You Don't Repeat Yourself

Description

There's a principle in programming called `Don't Repeat Yourself (DRY)`. The reason repeated code is a problem is because any change requires fixing code in multiple places. This usually means more work for programmers and more room for errors. Notice in the example below that the `describe` method is shared by `Bird` and `Dog`:

```
Bird.prototype = {
  constructor: Bird,
  describe: function() {
    console.log("My name is " + this.name);
  }
};

Dog.prototype = {
  constructor: Dog,
  describe: function() {
    console.log("My name is " + this.name);
  }
};
```

The `describe` method is repeated in two places. The code can be edited to follow the `DRY` principle by creating a supertype (or parent) called `Animal`:

```
function Animal() { }

Animal.prototype = {
  constructor: Animal,
  describe: function() {
    console.log("My name is " + this.name);
  }
};
```

Since `Animal` includes the `describe` method, you can remove it from `Bird` and `Dog`:

```
Bird.prototype = {
  constructor: Bird
};

Dog.prototype = {
  constructor: Dog
};
```

Instructions

The `eat` method is repeated in both `Cat` and `Bear`. Edit the code in the spirit of `DRY` by moving the `eat` method to the `Animal` supertype.

Challenge Seed

```
function Cat(name) {
  this.name = name;
}

Cat.prototype = {
  constructor: Cat,
  eat: function() {
    console.log("nom nom nom");
  }
};

function Bear(name) {
  this.name = name;
}
```

```

Bear.prototype = {
  constructor: Bear,
  eat: function() {
    console.log("nom nom nom");
  }
};

function Animal() { }

Animal.prototype = {
  constructor: Animal,
};

}

```

Solution

```

function Cat(name) {
  this.name = name;
}

Cat.prototype = {
  constructor: Cat
};

function Bear(name) {
  this.name = name;
}

Bear.prototype = {
  constructor: Bear
};

function Animal() { }

Animal.prototype = {
  constructor: Animal,
  eat: function() {
    console.log("nom nom nom");
  }
};

```

18. Inherit Behaviors from a Supertype

Description

In the previous challenge, you created a supertype called `Animal` that defined behaviors shared by all animals:

```

function Animal() { }

Animal.prototype.eat = function() {
  console.log("nom nom nom");
};

```

This and the next challenge will cover how to reuse `Animal`'s methods inside `Bird` and `Dog` without defining them again. It uses a technique called `inheritance`. This challenge covers the first step: make an instance of the supertype (or parent). You already know one way to create an instance of `Animal` using the `new` operator:

```
let animal = new Animal();
```

There are some disadvantages when using this syntax for `inheritance`, which are too complex for the scope of this challenge. Instead, here's an alternative approach without those disadvantages:

```
let animal = Object.create(Animal.prototype);
```

`Object.create(obj)` creates a new object, and sets `obj` as the new object's prototype. Recall that the prototype is like the "recipe" for creating an object. By setting the prototype of `animal` to be `Animal`'s prototype, you are effectively giving the `animal` instance the same "recipe" as any other instance of `Animal`.

```
animal.eat(); // prints "nom nom nom"
animal instanceof Animal; // => true
```

Instructions

Use `Object.create` to make two instances of `Animal` named `duck` and `beagle`.

Challenge Seed

```
function Animal() { }

Animal.prototype = {
  constructor: Animal,
  eat: function() {
    console.log("nom nom nom");
  }
};

// Add your code below this line

let duck; // Change this line
let beagle; // Change this line

duck.eat(); // Should print "nom nom nom"
beagle.eat(); // Should print "nom nom nom"
```

Solution

```
function Animal() { }

Animal.prototype = {
  constructor: Animal,
  eat: function() {
    console.log("nom nom nom");
  }
};
let duck = Object.create(Animal.prototype);
let beagle = Object.create(Animal.prototype);

duck.eat();
beagle.eat();
```

19. Set the Child's Prototype to an Instance of the Parent

Description

In the previous challenge you saw the first step for inheriting behavior from the supertype (or parent) `Animal`: making a new instance of `Animal`. This challenge covers the next step: set the prototype of the subtype (or child)—in this case, `Bird`—to be an instance of `Animal`.

```
Bird.prototype = Object.create(Animal.prototype);
```

Remember that the `prototype` is like the "recipe" for creating an object. In a way, the recipe for `Bird` now includes all the key "ingredients" from `Animal`.

```
let duck = new Bird("Donald");
duck.eat(); // prints "nom nom nom"
```

`duck` inherits all of `Animal`'s properties, including the `eat` method.

Instructions

Modify the code so that instances of `Dog` inherit from `Animal`.

Challenge Seed

```
function Animal() { }

Animal.prototype = {
  constructor: Animal,
  eat: function() {
    console.log("nom nom nom");
  }
};

function Dog() { }

// Add your code below this line

let beagle = new Dog();
beagle.eat(); // Should print "nom nom nom"
```

Solution

```
function Animal() { }

Animal.prototype = {
  constructor: Animal,
  eat: function() {
    console.log("nom nom nom");
  }
};

function Dog() { }
Dog.prototype = Object.create(Animal.prototype);

let beagle = new Dog();
beagle.eat();
```

20. Reset an Inherited Constructor Property

Description

When an object inherits its `prototype` from another object, it also inherits the supertype's `constructor` property. Here's an example:

```
function Bird() {}

Bird.prototype = Object.create(Animal.prototype);
let duck = new Bird();
duck.constructor // function Animal(){...}
```

But `duck` and all instances of `Bird` should show that they were constructed by `Bird` and not `Animal`. To do so, you can manually set `Bird`'s `constructor` property to the `Bird` object:

```
Bird.prototype.constructor = Bird;
duck.constructor // function Bird(){...}
```

Instructions

Fix the code so `duck.constructor` and `beagle.constructor` return their respective constructors.

Challenge Seed

```
function Animal() { }

function Bird() { }

function Dog() { }
```

```

Bird.prototype = Object.create(Animal.prototype);
Dog.prototype = Object.create(Animal.prototype);

// Add your code below this line

let duck = new Bird();
let beagle = new Dog();

```

Solution

```

function Animal() { }
function Bird() { }
function Dog() { }
Bird.prototype = Object.create(Animal.prototype);
Dog.prototype = Object.create(Animal.prototype);
Dog.prototype.constructor = Dog;
Bird.prototype.constructor = Bird;
let duck = new Bird();
let beagle = new Dog();

```

21. Add Methods After Inheritance

Description

A constructor function that inherits its `prototype` object from a supertype constructor function can still have its own methods in addition to inherited methods. For example, `Bird` is a constructor that inherits its `prototype` from `Animal`:

```

function Animal() { }
Animal.prototype.eat = function() {
  console.log("nom nom nom");
};

function Bird() { }
Bird.prototype = Object.create(Animal.prototype);
Bird.prototype.constructor = Bird;

```

In addition to what is inherited from `Animal`, you want to add behavior that is unique to `Bird` objects. Here, `Bird` will get a `fly()` function. Functions are added to `Bird`'s `prototype` the same way as any constructor function:

```

Bird.prototype.fly = function() {
  console.log("I'm flying!");
};

```

Now instances of `Bird` will have both `eat()` and `fly()` methods:

```

let duck = new Bird();
duck.eat(); // prints "nom nom nom"
duck.fly(); // prints "I'm flying!"

```

Instructions

Add all necessary code so the `Dog` object inherits from `Animal` and the `Dog`'s `prototype` `constructor` is set to `Dog`. Then add a `bark()` method to the `Dog` object so that `beagle` can both `eat()` and `bark()`. The `bark()` method should print "Woof!" to the console.

Challenge Seed

```

function Animal() { }
Animal.prototype.eat = function() { console.log("nom nom nom"); };

```

```
function Dog() { }

// Add your code below this line

// Add your code above this line

let beagle = new Dog();

beagle.eat(); // Should print "nom nom nom"
beagle.bark(); // Should print "Woof!"
```

Solution

```
function Animal() { }
Animal.prototype.eat = function() { console.log("nom nom nom"); };

function Dog() { }
Dog.prototype = Object.create(Animal.prototype);
Dog.prototype.constructor = Dog;
Dog.prototype.bark = function () {
  console.log('Woof!');
};
let beagle = new Dog();

beagle.eat();
beagle.bark();
```

22. Override Inherited Methods

Description

In previous lessons, you learned that an object can inherit its behavior (methods) from another object by cloning its prototype object:

```
ChildObject.prototype = Object.create(ParentObject.prototype);
```

Then the ChildObject received its own methods by chaining them onto its prototype :

```
ChildObject.prototype.methodName = function() {...};
```

It's possible to override an inherited method. It's done the same way - by adding a method to

ChildObject.prototype using the same method name as the one to override. Here's an example of Bird overriding the eat() method inherited from Animal :

```
function Animal() {}
Animal.prototype.eat = function() {
  return "nom nom nom";
};

function Bird() {}

// Inherit all methods from Animal
Bird.prototype = Object.create(Animal.prototype);

// Bird.eat() overrides Animal.eat()
Bird.prototype.eat = function() {
  return "peck peck peck";
};
```

If you have an instance let duck = new Bird(); and you call duck.eat() , this is how JavaScript looks for the method on duck's prototype chain: 1. duck => Is eat() defined here? No. 2. Bird => Is eat() defined here? => Yes. Execute it and stop searching. 3. Animal => eat() is also defined, but JavaScript stopped searching before reaching this level. 4. Object => JavaScript stopped searching before reaching this level.

Instructions

Override the `fly()` method for `Penguin` so that it returns "Alas, this is a flightless bird."

Challenge Seed

```
function Bird() { }

Bird.prototype.fly = function() { return "I am flying!"; };

function Penguin() { }
Penguin.prototype = Object.create(Bird.prototype);
Penguin.prototype.constructor = Penguin;

// Add your code below this line

// Add your code above this line

let penguin = new Penguin();
console.log(penguin.fly());
```

Solution

```
function Bird() { }

Bird.prototype.fly = function() { return "I am flying!"; };

function Penguin() { }
Penguin.prototype = Object.create(Bird.prototype);
Penguin.prototype.constructor = Penguin;
Penguin.prototype.fly = () => 'Alas, this is a flightless bird.';
let penguin = new Penguin();
console.log(penguin.fly());
```

23. Use a Mixin to Add Common Behavior Between Unrelated Objects

Description

As you have seen, behavior is shared through inheritance. However, there are cases when inheritance is not the best solution. Inheritance does not work well for unrelated objects like `Bird` and `Airplane`. They can both fly, but a `Bird` is not a type of `Airplane` and vice versa. For unrelated objects, it's better to use `mixins`. A mixin allows other objects to use a collection of functions.

```
let flyMixin = function(obj) {
  obj.fly = function() {
    console.log("Flying, wooosh!");
  }
};
```

The `flyMixin` takes any object and gives it the `fly` method.

```
let bird = {
  name: "Donald",
  numLegs: 2
};

let plane = {
  model: "777",
  numPassengers: 524
```

```
};

flyMixin(bird);
flyMixin(plane);
```

Here `bird` and `plane` are passed into `flyMixin`, which then assigns the `fly` function to each object. Now `bird` and `plane` can both fly:

```
bird.fly(); // prints "Flying, wooosh!"
plane.fly(); // prints "Flying, wooosh!"
```

Note how the `mixin` allows for the same `fly` method to be reused by unrelated objects `bird` and `plane`.

Instructions

Create a `mixin` named `glideMixin` that defines a method named `glide`. Then use the `glideMixin` to give both `bird` and `boat` the ability to glide.

Challenge Seed

```
let bird = {
  name: "Donald",
  numLegs: 2
};

let boat = {
  name: "Warrior",
  type: "race-boat"
};

// Add your code below this line
```

Solution

```
let bird = {
  name: "Donald",
  numLegs: 2
};

let boat = {
  name: "Warrior",
  type: "race-boat"
};

function glideMixin (obj) {
  obj.glide = () => 'Gliding!';
}

glideMixin(bird);
glideMixin(boat);
```

24. Use Closure to Protect Properties Within an Object from Being Modified Externally

Description

In the previous challenge, `bird` had a public property `name`. It is considered public because it can be accessed and changed outside of `bird`'s definition.

```
bird.name = "Duffy";
```

Therefore, any part of your code can easily change the name of `bird` to any value. Think about things like passwords and bank accounts being easily changeable by any part of your codebase. That could cause a lot of issues.

The simplest way to make this public property private is by creating a variable within the constructor function. This changes the scope of that variable to be within the constructor function versus available globally. This way, the variable can only be accessed and changed by methods also within the constructor function.

```
function Bird() {
  let hatchedEgg = 10; // private variable

  /* publicly available method that a bird object can use */
  this.getHatchedEggCount = function() {
    return hatchedEgg;
  };
}

let ducky = new Bird();
ducky.getHatchedEggCount(); // returns 10
```

Here `getHatchedEggCount` is a privileged method, because it has access to the private variable `hatchedEgg`. This is possible because `hatchedEgg` is declared in the same context as `getHatchedEggCount`. In JavaScript, a function always has access to the context in which it was created. This is called `closure`.

Instructions

Change how `weight` is declared in the `Bird` function so it is a private variable. Then, create a method `getWeight` that returns the value of `weight` 15.

Challenge Seed

```
function Bird() {
  this.weight = 15;

}
```

Solution

```
function Bird() {
  let weight = 15;

  this.getWeight = () => weight;
}
```

25. Understand the Immediately Invoked Function Expression (IIFE)

Description

A common pattern in JavaScript is to execute a function as soon as it is declared:

```
(function () {
  console.log("Chirp, chirp!");
})(); // this is an anonymous function expression that executes right away
// Outputs "Chirp, chirp!" immediately
```

Note that the function has no name and is not stored in a variable. The two parentheses () at the end of the function expression cause it to be immediately executed or invoked. This pattern is known as an **immediately invoked function expression or IIFE**.

Instructions

Rewrite the function `makeNest` and remove its call so instead it's an **anonymous** immediately invoked function expression (IIFE).

Challenge Seed

```
function makeNest() {
  console.log("A cozy nest is ready");
}

makeNest();
```

Solution

```
(function () {
  console.log("A cozy nest is ready");
})();
```

26. Use an IIFE to Create a Module

Description

An **immediately invoked function expression (IIFE)** is often used to group related functionality into a single object or module. For example, an earlier challenge defined two mixins:

```
function glideMixin(obj) {
  obj.glide = function() {
    console.log("Gliding on the water");
  };
}

function flyMixin(obj) {
  obj.fly = function() {
    console.log("Flying, wooosh!");
  };
}
```

We can group these mixins into a module as follows:

```
let motionModule = (function () {
  return {
    glideMixin: function (obj) {
      obj.glide = function() {
        console.log("Gliding on the water");
      };
    },
    flyMixin: function(obj) {
      obj.fly = function() {
        console.log("Flying, wooosh!");
      };
    }
  }
})() // The two parentheses cause the function to be immediately invoked
```

Note that you have an immediately invoked function expression (IIFE) that returns an object motionModule . This returned object contains all of the mixin behaviors as properties of the object. The advantage of the module pattern is that all of the motion behaviors can be packaged into a single object that can then be used by other parts of your code. Here is an example using it:

```
motionModule.glideMixin(duck);
duck.glide();
```

Instructions

Create a module named funModule to wrap the two mixins isCuteMixin and singMixin . funModule should return an object.

Challenge Seed

```
let isCuteMixin = function(obj) {
  obj.isCute = function() {
    return true;
  };
}
let singMixin = function(obj) {
  obj.sing = function() {
    console.log("Singing to an awesome tune");
  };
};
```

Solution

```
const funModule = (function () {
  return {
    isCuteMixin: obj => {
      obj.isCute = () => true;
    },
    singMixin: obj => {
      obj.sing = () => console.log("Singing to an awesome tune");
    }
  };
})();
```

Functional Programming

1. Learn About Functional Programming

Description

Functional programming is a style of programming where solutions are simple, isolated functions, without any side effects outside of the function scope. INPUT -> PROCESS -> OUTPUT Functional programming is about: 1) Isolated functions - there is no dependence on the state of the program, which includes global variables that are subject to change 2) Pure functions - the same input always gives the same output 3) Functions with limited side effects - any changes, or mutations, to the state of the program outside the function are carefully controlled

Instructions

The members of freeCodeCamp happen to love tea. In the code editor, the prepareTea and getTea functions are already defined for you. Call the getTea function to get 40 cups of tea for the team, and store them in the tea4TeamFCC variable.

Challenge Seed

```

/**
 * A long process to prepare tea.
 * @return {string} A cup of tea.
 */
const prepareTea = () => 'greenTea';

/**
 * Get given number of cups of tea.
 * @param {number} numOfCups Number of required cups of tea.
 * @return {Array<string>} Given amount of tea cups.
 */
const getTea = (numOfCups) => {
  const teaCups = [];

  for(let cups = 1; cups <= numOfCups; cups += 1) {
    const teaCup = prepareTea();
    teaCups.push(teaCup);
  }

  return teaCups;
};

// Add your code below this line

const tea4TeamFCC = null; // :(

// Add your code above this line

console.log(tea4TeamFCC);

```

Solution

```
// solution required
```

2. Understand Functional Programming Terminology

Description

The FCC Team had a mood swing and now wants two types of tea: green tea and black tea. General Fact: Client mood swings are pretty common. With that information, we'll need to revisit the `getTea` function from last challenge to handle various tea requests. We can modify `getTea` to accept a function as a parameter to be able to change the type of tea it prepares. This makes `getTea` more flexible, and gives the programmer more control when client requests change. But first, let's cover some functional terminology: `Callbacks` are the functions that are slipped or passed into another function to decide the invocation of that function. You may have seen them passed to other methods, for example in `filter`, the callback function tells JavaScript the criteria for how to filter an array. Functions that can be assigned to a variable, passed into another function, or returned from another function just like any other normal value, are called `first class functions`. In JavaScript, all functions are `first class functions`. The functions that take a function as an argument, or return a function as a return value are called `higher order functions`. When the functions are passed in to another function or returned from another function, then those functions which gets passed in or returned can be called a `lambda`.

Instructions

Prepare 27 cups of green tea and 13 cups of black tea and store them in `tea4GreenTeamFCC` and `tea4BlackTeamFCC` variables, respectively. Note that the `getTea` function has been modified so it now takes a function as the first argument. Note: The data (the number of cups of tea) is supplied as the last argument. We'll discuss this more in later lessons.

Challenge Seed

```

/**
 * A long process to prepare green tea.
 * @return {string} A cup of green tea.
 */
const prepareGreenTea = () => 'greenTea';

/**
 * A long process to prepare black tea.
 * @return {string} A cup of black tea.
 */
const prepareBlackTea = () => 'blackTea';

/**
 * Get given number of cups of tea.
 * @param {function():string} prepareTea The type of tea preparing function.
 * @param {number} numOfCups Number of required cups of tea.
 * @return {Array<string>} Given amount of tea cups.
 */
const getTea = (prepareTea, numOfCups) => {
  const teaCups = [];

  for(let cups = 1; cups <= numOfCups; cups += 1) {
    const teaCup = prepareTea();
    teaCups.push(teaCup);
  }

  return teaCups;
};

// Add your code below this line

const tea4GreenTeamFCC = null; // :(
const tea4BlackTeamFCC = null; // :(

// Add your code above this line

console.log(
  tea4GreenTeamFCC,
  tea4BlackTeamFCC
);

```

Solution

```
// solution required
```

3. Understand the Hazards of Using Imperative Code

Description

Functional programming is a good habit. It keeps your code easy to manage, and saves you from sneaky bugs. But before we get there, let's look at an imperative approach to programming to highlight where you may have issues. In English (and many other languages), the imperative tense is used to give commands. Similarly, an imperative style in programming is one that gives the computer a set of statements to perform a task. Often the statements change the state of the program, like updating global variables. A classic example is writing a `for` loop that gives exact directions to iterate over the indices of an array. In contrast, functional programming is a form of declarative programming. You tell the computer what you want done by calling a method or function. JavaScript offers many predefined methods that handle common tasks so you don't need to write out how the computer should perform them. For example, instead of using the `for` loop mentioned above, you could call the `map` method which handles the details of iterating over an array. This helps to avoid semantic errors, like the "Off By One Errors" that were covered in the Debugging section. Consider the scenario: you are browsing the web in your browser, and want to track the tabs you have opened. Let's try to model this using some simple object-oriented code. A `Window` object is made up of tabs, and you usually have more than one `Window` open. The titles of each open site in each `Window` object is held in an array. After working in the browser (opening new tabs, merging windows, and closing tabs), you want to print the tabs that are still open. Closed tabs are removed from

the array and new tabs (for simplicity) get added to the end of it. The code editor shows an implementation of this functionality with functions for `tabOpen()`, `tabClose()`, and `join()`. The array `tabs` is part of the Window object that stores the name of the open pages.

Instructions

Run the code in the editor. It's using a method that has side effects in the program, causing incorrect output. The final list of open tabs should be `['FB', 'Gitter', 'Reddit', 'Twitter', 'Medium', 'new tab', 'Netflix', 'YouTube', 'Vine', 'GMail', 'Work mail', 'Docs', 'freeCodeCamp', 'new tab']` but the output will be slightly different. Work through the code and see if you can figure out the problem, then advance to the next challenge to learn more.

Instructions

Challenge Seed

```
// tabs is an array of titles of each site open within the window
var Window = function(tabs) {
    this.tabs = tabs; // we keep a record of the array inside the object
};

// When you join two windows into one window
Window.prototype.join = function (otherWindow) {
    this.tabs = this.tabs.concat(otherWindow.tabs);
    return this;
};

// When you open a new tab at the end
Window.prototype.tabOpen = function (tab) {
    this.tabs.push('new tab'); // let's open a new tab for now
    return this;
};

// When you close a tab
Window.prototype.tabClose = function (index) {
    var tabsBeforeIndex = this.tabs.splice(0, index); // get the tabs before the tab
    var tabsAfterIndex = this.tabs.splice(index); // get the tabs after the tab

    this.tabs = tabsBeforeIndex.concat(tabsAfterIndex); // join them together
    return this;
};

// Let's create three browser windows
var workWindow = new Window(['GMail', 'Inbox', 'Work mail', 'Docs', 'freeCodeCamp']); // Your mailbox, drive, and other work sites
var socialWindow = new Window(['FB', 'Gitter', 'Reddit', 'Twitter', 'Medium']); // Social sites
var videoWindow = new Window(['Netflix', 'YouTube', 'Vimeo', 'Vine']); // Entertainment sites

// Now perform the tab opening, closing, and other operations
var finalTabs = socialWindow
    .tabOpen() // Open a new tab for cat memes
    .join(videoWindow.tabClose(2)) // Close third tab in video window, and join
    .join(workWindow.tabClose(1).tabOpen());

alert(finalTabs.tabs);
```

Solution

```
// solution required
```

4. Avoid Mutations and Side Effects Using Functional Programming

Description

If you haven't already figured it out, the issue in the previous challenge was with the `splice` call in the `tabClose()` function. Unfortunately, `splice` changes the original array it is called on, so the second call to it used a modified array, and gave unexpected results. This is a small example of a much larger pattern - you call a function on a variable, array, or an object, and the function changes the variable or something in the object. One of the core principles of functional programming is to not change things. Changes lead to bugs. It's easier to prevent bugs knowing that your functions don't change anything, including the function arguments or any global variable. The previous example didn't have any complicated operations but the `splice` method changed the original array, and resulted in a bug. Recall that in functional programming, changing or altering things is called `mutation`, and the outcome is called a `side effect`. A function, ideally, should be a `pure function`, meaning that it does not cause any side effects. Let's try to master this discipline and not alter any variable or object in our code.

Instructions

Fill in the code for the function `incrementer` so it returns the value of the global variable `fixedValue` increased by one.

Challenge Seed

```
// the global variable
var fixedValue = 4;

function incrementer () {
  // Add your code below this line

  // Add your code above this line
}

var newValue = incrementer(); // Should equal 5
console.log(fixedValue); // Should print 4
```

Solution

```
var fixedValue = 4

function incrementer() {
  return fixedValue + 1
}

var newValue = incrementer(); // Should equal 5
```

5. Pass Arguments to Avoid External Dependence in a Function

Description

The last challenge was a step closer to functional programming principles, but there is still something missing. We didn't alter the global variable value, but the function `incrementer` would not work without the global variable `fixedValue` being there. Another principle of functional programming is to always declare your dependencies explicitly. This means if a function depends on a variable or object being present, then pass that variable or object directly into the function as an argument. There are several good consequences from this principle. The function is easier to test, you know exactly what input it takes, and it won't depend on anything else in your program. This can give you more confidence when you alter, remove, or add new code. You would know what you can or cannot change and you can see where the potential traps are. Finally, the function would always produce the same output for the same set of inputs, no matter what part of the code executes it.

Instructions

Let's update the `incrementer` function to clearly declare its dependencies. Write the `incrementer` function so it takes an argument, and then increases the value by one.

Challenge Seed

```
// the global variable
var fixedValue = 4;

// Add your code below this line
function incrementer () {

    // Add your code above this line
}

var newValue = incrementer(fixedValue); // Should equal 5
console.log(fixedValue); // Should print 4
```

Solution

```
// the global variable
var fixedValue = 4;

const incrementer = val => val + 1;

var newValue = incrementer(fixedValue); // Should equal 5
console.log(fixedValue); // Should print 4
```

6. Refactor Global Variables Out of Functions

Description

So far, we have seen two distinct principles for functional programming: 1) Don't alter a variable or object - create new variables and objects and return them if need be from a function. 2) Declare function arguments - any computation inside a function depends only on the arguments, and not on any global object or variable. Adding one to a number is not very exciting, but we can apply these principles when working with arrays or more complex objects.

Instructions

Rewrite the code so the global array `bookList` is not changed inside either function. The `add` function should add the given `bookName` to the end of an array. The `remove` function should remove the given `bookName` from an array. Both functions should return an array, and any new parameters should be added before the `bookName` parameter.

Challenge Seed

```
// the global variable
var bookList = ["The Hound of the Baskervilles", "On The Electrodynamics of Moving Bodies", "Philosophiae Naturalis Principia Mathematica", "Disquisitiones Arithmeticae"];

/* This function should add a book to the list and return the list */
// New parameters should come before bookName

// Add your code below this line
function add (bookName) {

    bookList.push(bookName);
```

```

return bookList;

// Add your code above this line
}

/* This function should remove a book from the list and return the list */
// New parameters should come before the bookName one

// Add your code below this line
function remove (bookName) {
  var book_index = bookList.indexOf(bookName);
  if (book_index >= 0) {

    bookList.splice(book_index, 1);
    return bookList;

    // Add your code above this line
  }
}

var newBookList = add(bookList, 'A Brief History of Time');
var newerBookList = remove(bookList, 'On The Electrodynamics of Moving Bodies');
var newestBookList = remove(add(bookList, 'A Brief History of Time'), 'On The Electrodynamics of Moving
Bodies');

console.log(bookList);

```

Solution

```
// solution required
```

7. Use the map Method to Extract Data from an Array

Description

So far we have learned to use pure functions to avoid side effects in a program. Also, we have seen the value in having a function only depend on its input arguments. This is only the beginning. As its name suggests, functional programming is centered around a theory of functions. It would make sense to be able to pass them as arguments to other functions, and return a function from another function. Functions are considered First Class Objects in JavaScript, which means they can be used like any other object. They can be saved in variables, stored in an object, or passed as function arguments. Let's start with some simple array functions, which are methods on the array object prototype. In this exercise we are looking at `Array.prototype.map()`, or more simply `map`. Remember that the `map` method is a way to iterate over each item in an array. It creates a new array (without changing the original one) after applying a callback function to every element.

Instructions

The `watchList` array holds objects with information on several movies. Use `map` to pull the title and rating from `watchList` and save the new array in the `rating` variable. The code in the editor currently uses a `for` loop to do this, replace the loop functionality with your `map` expression.

Challenge Seed

```

// the global variable
var watchList = [
  {
    "Title": "Inception",
    "Year": "2010",
    "Rated": "PG-13",
    "Released": "16 Jul 2010",
    "Runtime": "148 min",
    "Genre": "Action, Adventure, Crime",
    "Director": "Christopher Nolan",

```

```

        "Writer": "Christopher Nolan",
        "Actors": "Leonardo DiCaprio, Joseph Gordon-Levitt, Ellen Page, Tom Hardy",
        "Plot": "A thief, who steals corporate secrets through use of dream-sharing
technology, is given the inverse task of planting an idea into the mind of a CEO.",
        "Language": "English, Japanese, French",
        "Country": "USA, UK",
        "Awards": "Won 4 Oscars. Another 143 wins & 198 nominations.",
        "Poster": "http://ia.media-
imdb.com/images/M/MV5BMjAxMzY3NjcxNF5BMl5BanBnXkFtZTcwNTI5OTM0Mw@._V1_SX300.jpg",
        "Metascore": "74",
        "imdbRating": "8.8",
        "imdbVotes": "1,446,708",
        "imdbID": "tt1375666",
        "Type": "movie",
        "Response": "True"
    },
    {
        "Title": "Interstellar",
        "Year": "2014",
        "Rated": "PG-13",
        "Released": "07 Nov 2014",
        "Runtime": "169 min",
        "Genre": "Adventure, Drama, Sci-Fi",
        "Director": "Christopher Nolan",
        "Writer": "Jonathan Nolan, Christopher Nolan",
        "Actors": "Ellen Burstyn, Matthew McConaughey, Mackenzie Foy, John Lithgow",
        "Plot": "A team of explorers travel through a wormhole in space in an attempt to
ensure humanity's survival.",
        "Language": "English",
        "Country": "USA, UK",
        "Awards": "Won 1 Oscar. Another 39 wins & 132 nominations.",
        "Poster": "http://ia.media-
imdb.com/images/M/MV5BMjIxNTU4MzY4MF5BMl5BanBnXkFtZTgwMzM4ODI3MjE@._V1_SX300.jpg",
        "Metascore": "74",
        "imdbRating": "8.6",
        "imdbVotes": "910,366",
        "imdbID": "tt0816692",
        "Type": "movie",
        "Response": "True"
    },
    {
        "Title": "The Dark Knight",
        "Year": "2008",
        "Rated": "PG-13",
        "Released": "18 Jul 2008",
        "Runtime": "152 min",
        "Genre": "Action, Adventure, Crime",
        "Director": "Christopher Nolan",
        "Writer": "Jonathan Nolan (screenplay), Christopher Nolan (screenplay), Christopher
Nolan (story), David S. Goyer (story), Bob Kane (characters)",
        "Actors": "Christian Bale, Heath Ledger, Aaron Eckhart, Michael Caine",
        "Plot": "When the menace known as the Joker wreaks havoc and chaos on the people of
Gotham, the caped crusader must come to terms with one of the greatest psychological tests of his
ability to fight injustice.",
        "Language": "English, Mandarin",
        "Country": "USA, UK",
        "Awards": "Won 2 Oscars. Another 146 wins & 142 nominations.",
        "Poster": "http://ia.media-
imdb.com/images/M/MV5BMTMxNTMwODM0NF5BMl5BanBnXkFtZTcwODAyMTk2Mw@._V1_SX300.jpg",
        "Metascore": "82",
        "imdbRating": "9.0",
        "imdbVotes": "1,652,832",
        "imdbID": "tt0468569",
        "Type": "movie",
        "Response": "True"
    },
    {
        "Title": "Batman Begins",
        "Year": "2005",
        "Rated": "PG-13",
        "Released": "15 Jun 2005",
        "Runtime": "140 min",
        "Genre": "Action, Adventure",
        "Director": "Christopher Nolan",
        "Writer": "Bob Kane (characters), David S. Goyer (story), Christopher Nolan
(screenplay), David S. Goyer (screenplay)",
        "Actors": "Christian Bale, Michael Caine, Liam Neeson, Katie Holmes",
    }

```

```

        "Plot": "After training with his mentor, Batman begins his fight to free crime-ridden
Gotham City from the corruption that Scarecrow and the League of Shadows have cast upon it.",
        "Language": "English, Urdu, Mandarin",
        "Country": "USA, UK",
        "Awards": "Nominated for 1 Oscar. Another 15 wins & 66 nominations.",
        "Poster": "http://ia.media-
imdb.com/images/M/MV5BNTM3OTc0MzM20V5BMl5BanBnXkFtZTywNzUwMTI3._V1_SX300.jpg",
        "Metascore": "70",
        "imdbRating": "8.3",
        "imdbVotes": "972,584",
        "imdbID": "tt0372784",
        "Type": "movie",
        "Response": "True"
    },
    {
        "Title": "Avatar",
        "Year": "2009",
        "Rated": "PG-13",
        "Released": "18 Dec 2009",
        "Runtime": "162 min",
        "Genre": "Action, Adventure, Fantasy",
        "Director": "James Cameron",
        "Writer": "James Cameron",
        "Actors": "Sam Worthington, Zoe Saldana, Sigourney Weaver, Stephen Lang",
        "Plot": "A paraplegic marine dispatched to the moon Pandora on a unique mission
becomes torn between following his orders and protecting the world he feels is his home.",
        "Language": "English, Spanish",
        "Country": "USA, UK",
        "Awards": "Won 3 Oscars. Another 80 wins & 121 nominations.",
        "Poster": "http://ia.media-
imdb.com/images/M/MV5BMTYwOTEwNjAzMl5BMl5BanBnXkFtZTcwODc5MTUwMw@@._V1_SX300.jpg",
        "Metascore": "83",
        "imdbRating": "7.9",
        "imdbVotes": "876,575",
        "imdbID": "tt0499549",
        "Type": "movie",
        "Response": "True"
    }
];
// Add your code below this line

var rating = [];
for(var i=0; i < watchList.length; i++){
    rating.push({title: watchList[i]["Title"], rating: watchList[i]["imdbRating"]});
}

// Add your code above this line

console.log(JSON.stringify(rating));

```

Solution

```

// the global variable
var watchList = [
    {
        "Title": "Inception",
        "Year": "2010",
        "Rated": "PG-13",
        "Released": "16 Jul 2010",
        "Runtime": "148 min",
        "Genre": "Action, Adventure, Crime",
        "Director": "Christopher Nolan",
        "Writer": "Christopher Nolan",
        "Actors": "Leonardo DiCaprio, Joseph Gordon-Levitt, Ellen Page, Tom Hardy",
        "Plot": "A thief, who steals corporate secrets through use of dream-sharing
technology, is given the inverse task of planting an idea into the mind of a CEO.",
        "Language": "English, Japanese, French",
        "Country": "USA, UK",
        "Awards": "Won 4 Oscars. Another 143 wins & 198 nominations.",
        "Poster": "http://ia.media-
imdb.com/images/M/MV5BMjAxMzY3NjcxNF5BMl5BanBnXkFtZTcwNTI5OTM0Mw@@._V1_SX300.jpg",
        "Metascore": "74",
        "imdbRating": "8.8",

```

```

        "imdbVotes": "1,446,708",
        "imdbID": "tt1375666",
        "Type": "movie",
        "Response": "True"
    },
    {
        "Title": "Interstellar",
        "Year": "2014",
        "Rated": "PG-13",
        "Released": "07 Nov 2014",
        "Runtime": "169 min",
        "Genre": "Adventure, Drama, Sci-Fi",
        "Director": "Christopher Nolan",
        "Writer": "Jonathan Nolan, Christopher Nolan",
        "Actors": "Ellen Burstyn, Matthew McConaughey, Mackenzie Foy, John Lithgow",
        "Plot": "A team of explorers travel through a wormhole in space in an attempt to ensure humanity's survival.",
        "Language": "English",
        "Country": "USA, UK",
        "Awards": "Won 1 Oscar. Another 39 wins & 132 nominations.",
        "Poster": "http://ia.media-imdb.com/images/M/MV5BmjIxNTU4MzY4MF5BMl5BanBnXkFtZTgwMzM4ODI3MjE@._V1_SX300.jpg",
        "Metascore": "74",
        "imdbRating": "8.6",
        "imdbVotes": "910,366",
        "imdbID": "tt0816692",
        "Type": "movie",
        "Response": "True"
    },
    {
        "Title": "The Dark Knight",
        "Year": "2008",
        "Rated": "PG-13",
        "Released": "18 Jul 2008",
        "Runtime": "152 min",
        "Genre": "Action, Adventure, Crime",
        "Director": "Christopher Nolan",
        "Writer": "Jonathan Nolan (screenplay), Christopher Nolan (screenplay), Christopher Nolan (story), David S. Goyer (story), Bob Kane (characters)",
        "Actors": "Christian Bale, Heath Ledger, Aaron Eckhart, Michael Caine",
        "Plot": "When the menace known as the Joker wreaks havoc and chaos on the people of Gotham, the caped crusader must come to terms with one of the greatest psychological tests of his ability to fight injustice.",
        "Language": "English, Mandarin",
        "Country": "USA, UK",
        "Awards": "Won 2 Oscars. Another 146 wins & 142 nominations.",
        "Poster": "http://ia.media-imdb.com/images/M/MV5BMTMxNTMwODM0NDI5BMl5BanBnXkFtZTcwODAyMTk2Mw@._V1_SX300.jpg",
        "Metascore": "82",
        "imdbRating": "9.0",
        "imdbVotes": "1,652,832",
        "imdbID": "tt0468569",
        "Type": "movie",
        "Response": "True"
    },
    {
        "Title": "Batman Begins",
        "Year": "2005",
        "Rated": "PG-13",
        "Released": "15 Jun 2005",
        "Runtime": "140 min",
        "Genre": "Action, Adventure",
        "Director": "Christopher Nolan",
        "Writer": "Bob Kane (characters), David S. Goyer (story), Christopher Nolan (screenplay), David S. Goyer (screenplay)",
        "Actors": "Christian Bale, Michael Caine, Liam Neeson, Katie Holmes",
        "Plot": "After training with his mentor, Batman begins his fight to free crime-ridden Gotham City from the corruption that Scarecrow and the League of Shadows have cast upon it.",
        "Language": "English, Urdu, Mandarin",
        "Country": "USA, UK",
        "Awards": "Nominated for 1 Oscar. Another 15 wins & 66 nominations.",
        "Poster": "http://ia.media-imdb.com/images/M/MV5BNTM3OTc0MzM2OV5BMl5BanBnXkFtZTYwNzUwMTI3._V1_SX300.jpg",
        "Metascore": "70",
        "imdbRating": "8.3",
        "imdbVotes": "972,584",
        "imdbID": "tt0372784",
    }

```

```

        "Type": "movie",
        "Response": "True"
    },
    {
        "Title": "Avatar",
        "Year": "2009",
        "Rated": "PG-13",
        "Released": "18 Dec 2009",
        "Runtime": "162 min",
        "Genre": "Action, Adventure, Fantasy",
        "Director": "James Cameron",
        "Writer": "James Cameron",
        "Actors": "Sam Worthington, Zoe Saldana, Sigourney Weaver, Stephen Lang",
        "Plot": "A paraplegic marine dispatched to the moon Pandora on a unique mission becomes torn between following his orders and protecting the world he feels is his home.",
        "Language": "English, Spanish",
        "Country": "USA, UK",
        "Awards": "Won 3 Oscars. Another 80 wins & 121 nominations.",
        "Poster": "http://ia.media-imdb.com/images/MV5BMTYwOTEwNjAzMl5BMl5BanBnXkFtZTcwODc5MTUwMw@@._V1_SX300.jpg",
        "Metascore": "83",
        "imdbRating": "7.9",
        "imdbVotes": "876,575",
        "imdbID": "tt0499549",
        "Type": "movie",
        "Response": "True"
    }
];

var rating = watchList.map(function(movie) {
    return {
        title: movie["Title"],
        rating: movie["imdbRating"]
    }
});

```

8. Implement map on a Prototype

Description

As you have seen from applying `Array.prototype.map()`, or simply `map()` earlier, the `map` method returns an array of the same length as the one it was called on. It also doesn't alter the original array, as long as its callback function doesn't. In other words, `map` is a pure function, and its output depends solely on its inputs. Plus, it takes another function as its argument. It would teach us a lot about `map` to try to implement a version of it that behaves exactly like the `Array.prototype.map()` with a `for` loop or `Array.prototype.forEach()`. Note: A pure function is allowed to alter local variables defined within its scope, although, it's preferable to avoid that as well.

Instructions

Write your own `Array.prototype.myMap()`, which should behave exactly like `Array.prototype.map()`. You may use a `for` loop or the `forEach` method.

Challenge Seed

```

// the global Array
var s = [23, 65, 98, 5];

Array.prototype.myMap = function(callback){
    var newArray = [];
    // Add your code below this line

    // Add your code above this line
    return newArray;
};

var new_s = s.myMap(function(item){

```

```
    return item * 2;
});
```

Solution

```
// solution required
```

9. Use the filter Method to Extract Data from an Array

Description

Another useful array function is `Array.prototype.filter()`, or simply `filter()`. The `filter` method returns a new array which is at most as long as the original array, but usually has fewer items. `Filter` doesn't alter the original array, just like `map`. It takes a callback function that applies the logic inside the callback on each element of the array. If an element returns true based on the criteria in the callback function, then it is included in the new array.

Instructions

The variable `watchList` holds an array of objects with information on several movies. Use a combination of `filter` and `map` to return a new array of objects with only `title` and `rating` keys, but where `imdbRating` is greater than or equal to 8.0. Note that the rating values are saved as strings in the object and you may want to convert them into numbers to perform mathematical operations on them.

Challenge Seed

```
// the global variable
var watchList = [
  {
    "Title": "Inception",
    "Year": "2010",
    "Rated": "PG-13",
    "Released": "16 Jul 2010",
    "Runtime": "148 min",
    "Genre": "Action, Adventure, Crime",
    "Director": "Christopher Nolan",
    "Writer": "Christopher Nolan",
    "Actors": "Leonardo DiCaprio, Joseph Gordon-Levitt, Ellen Page, Tom Hardy",
    "Plot": "A thief, who steals corporate secrets through use of dream-sharing technology, is given the inverse task of planting an idea into the mind of a CEO.",
    "Language": "English, Japanese, French",
    "Country": "USA, UK",
    "Awards": "Won 4 Oscars. Another 143 wins & 198 nominations.",
    "Poster": "http://ia.media-imdb.com/images/M/MV5BMjAxMzY3NjcxNF5BMl5BanBnXkFtZTcwNTI5OTM0Mw@@._V1_SX300.jpg",
    "Metascore": "74",
    "imdbRating": "8.8",
    "imdbVotes": "1,446,708",
    "imdbID": "tt1375666",
    "Type": "movie",
    "Response": "True"
  },
  {
    "Title": "Interstellar",
    "Year": "2014",
    "Rated": "PG-13",
    "Released": "07 Nov 2014",
    "Runtime": "169 min",
    "Genre": "Adventure, Drama, Sci-Fi",
    "Director": "Christopher Nolan",
    "Writer": "Jonathan Nolan, Christopher Nolan",
    "Actors": "Ellen Burstyn, Matthew McConaughey, Mackenzie Foy, John Lithgow",
    "Plot": "A team of explorers travel through a wormhole in space in an attempt to ensure humanity's survival."
  }
]
```

```

        "Language": "English",
        "Country": "USA, UK",
        "Awards": "Won 1 Oscar. Another 39 wins & 132 nominations.",
        "Poster": "http://ia.media-imdb.com/images/M/MV5BMjIxNTU4MzY4MF5BMl5BanBnXkFtZTgwMzM4ODI3MjE@._V1_SX300.jpg",
        "Metascore": "74",
        "imdbRating": "8.6",
        "imdbVotes": "910,366",
        "imdbID": "tt0816692",
        "Type": "movie",
        "Response": "True"
    },
    {
        "Title": "The Dark Knight",
        "Year": "2008",
        "Rated": "PG-13",
        "Released": "18 Jul 2008",
        "Runtime": "152 min",
        "Genre": "Action, Adventure, Crime",
        "Director": "Christopher Nolan",
        "Writer": "Jonathan Nolan (screenplay), Christopher Nolan (screenplay), Christopher Nolan (story), David S. Goyer (story), Bob Kane (characters)",
        "Actors": "Christian Bale, Heath Ledger, Aaron Eckhart, Michael Caine",
        "Plot": "When the menace known as the Joker wreaks havoc and chaos on the people of Gotham, the caped crusader must come to terms with one of the greatest psychological tests of his ability to fight injustice.",
        "Language": "English, Mandarin",
        "Country": "USA, UK",
        "Awards": "Won 2 Oscars. Another 146 wins & 142 nominations.",
        "Poster": "http://ia.media-imdb.com/images/M/MV5BMTMxNTMwODM0NF5BMl5BanBnXkFtZTcwODAyMTk2Mw@@._V1_SX300.jpg",
        "Metascore": "82",
        "imdbRating": "9.0",
        "imdbVotes": "1,652,832",
        "imdbID": "tt0468569",
        "Type": "movie",
        "Response": "True"
    },
    {
        "Title": "Batman Begins",
        "Year": "2005",
        "Rated": "PG-13",
        "Released": "15 Jun 2005",
        "Runtime": "140 min",
        "Genre": "Action, Adventure",
        "Director": "Christopher Nolan",
        "Writer": "Bob Kane (characters), David S. Goyer (story), Christopher Nolan (screenplay), David S. Goyer (screenplay)",
        "Actors": "Christian Bale, Michael Caine, Liam Neeson, Katie Holmes",
        "Plot": "After training with his mentor, Batman begins his fight to free crime-ridden Gotham City from the corruption that Scarecrow and the League of Shadows have cast upon it.",
        "Language": "English, Urdu, Mandarin",
        "Country": "USA, UK",
        "Awards": "Nominated for 1 Oscar. Another 15 wins & 66 nominations.",
        "Poster": "http://ia.media-imdb.com/images/M/MV5BNTM3OTc0MzM2OV5BMl5BanBnXkFtZTYwNzUwMTI3._V1_SX300.jpg",
        "Metascore": "70",
        "imdbRating": "8.3",
        "imdbVotes": "972,584",
        "imdbID": "tt0372784",
        "Type": "movie",
        "Response": "True"
    },
    {
        "Title": "Avatar",
        "Year": "2009",
        "Rated": "PG-13",
        "Released": "18 Dec 2009",
        "Runtime": "162 min",
        "Genre": "Action, Adventure, Fantasy",
        "Director": "James Cameron",
        "Writer": "James Cameron",
        "Actors": "Sam Worthington, Zoe Saldana, Sigourney Weaver, Stephen Lang",
        "Plot": "A paraplegic marine dispatched to the moon Pandora on a unique mission becomes torn between following his orders and protecting the world he feels is his home.",
        "Language": "English, Spanish",
        "Country": "USA, UK",
    }
}

```

```

        "Awards": "Won 3 Oscars. Another 80 wins & 121 nominations.",
        "Poster": "http://ia.media-
imdb.com/images/M/MV5BMTYwOTEwNjAzM15BMl5BanBnXkFtZTcwODc5MTUwMw@@._V1_SX300.jpg",
        "Metascore": "83",
        "imdbRating": "7.9",
        "imdbVotes": "876,575",
        "imdbID": "tt0499549",
        "Type": "movie",
        "Response": "True"
    }
];

// Add your code below this line

var filteredList;

// Add your code above this line

console.log(filteredList);

```

Solution

```
// solution required
```

10. Implement the filter Method on a Prototype

Description

It would teach us a lot about the `filter` method if we try to implement a version of it that behaves exactly like `Array.prototype.filter()`. It can use either a `for` loop or `Array.prototype.forEach()`. Note: A pure function is allowed to alter local variables defined within its scope, although, it's preferable to avoid that as well.

Instructions

Write your own `Array.prototype.myFilter()`, which should behave exactly like `Array.prototype.filter()`. You may use a `for` loop or the `Array.prototype.forEach()` method.

Challenge Seed

```

// the global Array
var s = [23, 65, 98, 5];

Array.prototype.myFilter = function(callback){
    var newArray = [];
    // Add your code below this line

    // Add your code above this line
    return newArray;
};

var new_s = s.myFilter(function(item){
    return item % 2 === 1;
});

```

Solution

```
// solution required
```

11. Return Part of an Array Using the slice Method

Description

The `slice` method returns a copy of certain elements of an array. It can take two arguments, the first gives the index of where to begin the slice, the second is the index for where to end the slice (and it's non-inclusive). If the arguments are not provided, the default is to start at the beginning of the array through the end, which is an easy way to make a copy of the entire array. The `slice` method does not mutate the original array, but returns a new one. Here's an example:

```
var arr = ["Cat", "Dog", "Tiger", "Zebra"];
var newArray = arr.slice(1, 3);
// Sets newArray to ["Dog", "Tiger"]
```

Instructions

Use the `slice` method in the `sliceArray` function to return part of the `anim` array given the provided `beginSlice` and `endSlice` indices. The function should return an array.

Challenge Seed

```
function sliceArray(anim, beginSlice, endSlice) {
  // Add your code below this line

  // Add your code above this line
}

var inputAnim = ["Cat", "Dog", "Tiger", "Zebra", "Ant"];
sliceArray(inputAnim, 1, 3);
```

Solution

```
// solution required
```

12. Remove Elements from an Array Using slice Instead of splice

Description

A common pattern while working with arrays is when you want to remove items and keep the rest of the array. JavaScript offers the `splice` method for this, which takes arguments for the index of where to start removing items, then the number of items to remove. If the second argument is not provided, the default is to remove items through the end. However, the `splice` method mutates the original array it is called on. Here's an example:

```
var cities = ["Chicago", "Delhi", "Islamabad", "London", "Berlin"];
cities.splice(3, 1); // Returns "London" and deletes it from the cities array
// cities is now ["Chicago", "Delhi", "Islamabad", "Berlin"]
```

As we saw in the last challenge, the `slice` method does not mutate the original array, but returns a new one which can be saved into a variable. Recall that the `slice` method takes two arguments for the indices to begin and end the slice (the end is non-inclusive), and returns those items in a new array. Using the `slice` method instead of `splice` helps to avoid any array-mutating side effects.

Instructions

Rewrite the function `nonMutatingSplice` by using `slice` instead of `splice`. It should limit the provided `cities` array to a length of 3, and return a new array with only the first three items. Do not mutate the original array provided to the function.

Challenge Seed

```
function nonMutatingSplice(cities) {
  // Add your code below this line
  return cities.splice(3);

  // Add your code above this line
}
var inputCities = ["Chicago", "Delhi", "Islamabad", "London", "Berlin"];
nonMutatingSplice(inputCities);
```

Solution

```
// solution required
```

13. Combine Two Arrays Using the `concat` Method

Description

Concatenation means to join items end to end. JavaScript offers the `concat` method for both strings and arrays that work in the same way. For arrays, the method is called on one, then another array is provided as the argument to `concat`, which is added to the end of the first array. It returns a new array and does not mutate either of the original arrays. Here's an example:

```
[1, 2, 3].concat([4, 5, 6]);
// Returns a new array [1, 2, 3, 4, 5, 6]
```

Instructions

Use the `concat` method in the `nonMutatingConcat` function to concatenate `attach` to the end of `original`. The function should return the concatenated array.

Challenge Seed

```
function nonMutatingConcat(original, attach) {
  // Add your code below this line

  // Add your code above this line
}
var first = [1, 2, 3];
var second = [4, 5];
nonMutatingConcat(first, second);
```

Solution

```
// solution required
```

14. Add Elements to the End of an Array Using `concat` Instead of `push`

Description

Functional programming is all about creating and using non-mutating functions. The last challenge introduced the `concat` method as a way to combine arrays into a new one without mutating the original arrays. Compare `concat` to the `push` method. `Push` adds an item to the end of the same array it is called on, which mutates that array. Here's an example:

```
var arr = [1, 2, 3];
arr.push([4, 5, 6]);
// arr is changed to [1, 2, 3, [4, 5, 6]]
// Not the functional programming way
```

`Concat` offers a way to add new items to the end of an array without any mutating side effects.

Instructions

Change the `nonMutatingPush` function so it uses `concat` to add `newItem` to the end of `original` instead of `push`. The function should return an array.

Challenge Seed

```
function nonMutatingPush(original, newItem) {
  // Add your code below this line
  return original.push(newItem);

  // Add your code above this line
}
var first = [1, 2, 3];
var second = [4, 5];
nonMutatingPush(first, second);
```

Solution

```
// solution required
```

15. Use the reduce Method to Analyze Data

Description

`Array.prototype.reduce()`, or simply `reduce()`, is the most general of all array operations in JavaScript. You can solve almost any array processing problem using the `reduce` method. This is not the case with the `filter` and `map` methods since they do not allow interaction between two different elements of the array. For example, if you want to compare elements of the array, or add them together, `filter` or `map` could not process that. The `reduce` method allows for more general forms of array processing, and it's possible to show that both `filter` and `map` can be derived as a special application of `reduce`. However, before we get there, let's practice using `reduce` first.

Instructions

The variable `watchList` holds an array of objects with information on several movies. Use `reduce` to find the average IMDB rating of the movies directed by Christopher Nolan. Recall from prior challenges how to `filter` data and `map` over it to pull what you need. You may need to create other variables, and return the average rating from `getRating` function. Note that the rating values are saved as strings in the object and need to be converted into numbers before they are used in any mathematical operations.

Challenge Seed

```
// the global variable
var watchList = [
    {
        "Title": "Inception",
        "Year": "2010",
        "Rated": "PG-13",
        "Released": "16 Jul 2010",
        "Runtime": "148 min",
        "Genre": "Action, Adventure, Crime",
        "Director": "Christopher Nolan",
        "Writer": "Christopher Nolan",
        "Actors": "Leonardo DiCaprio, Joseph Gordon-Levitt, Ellen Page, Tom Hardy",
        "Plot": "A thief, who steals corporate secrets through use of dream-sharing technology, is given the inverse task of planting an idea into the mind of a CEO.",
        "Language": "English, Japanese, French",
        "Country": "USA, UK",
        "Awards": "Won 4 Oscars. Another 143 wins & 198 nominations.",
        "Poster": "http://ia.media-imdb.com/images/M/MV5BMjAxMzY3NjcxNF5BMl5BanBnXkFtZTcwNTI50TM0Mw@._V1_SX300.jpg",
        "Metascore": "74",
        "imdbRating": "8.8",
        "imdbVotes": "1,446,708",
        "imdbID": "tt1375666",
        "Type": "movie",
        "Response": "True"
    },
    {
        "Title": "Interstellar",
        "Year": "2014",
        "Rated": "PG-13",
        "Released": "07 Nov 2014",
        "Runtime": "169 min",
        "Genre": "Adventure, Drama, Sci-Fi",
        "Director": "Christopher Nolan",
        "Writer": "Jonathan Nolan, Christopher Nolan",
        "Actors": "Ellen Burstyn, Matthew McConaughey, Mackenzie Foy, John Lithgow",
        "Plot": "A team of explorers travel through a wormhole in space in an attempt to ensure humanity's survival.",
        "Language": "English",
        "Country": "USA, UK",
        "Awards": "Won 1 Oscar. Another 39 wins & 132 nominations.",
        "Poster": "http://ia.media-imdb.com/images/M/MV5BMjIxNTU4MzY4MF5BMl5BanBnXkFtZTgwMzM4ODI3MjE@._V1_SX300.jpg",
        "Metascore": "74",
        "imdbRating": "8.6",
        "imdbVotes": "910,366",
        "imdbID": "tt0816692",
        "Type": "movie",
        "Response": "True"
    },
    {
        "Title": "The Dark Knight",
        "Year": "2008",
        "Rated": "PG-13",
        "Released": "18 Jul 2008",
        "Runtime": "152 min",
        "Genre": "Action, Adventure, Crime",
        "Director": "Christopher Nolan",
        "Writer": "Jonathan Nolan (screenplay), Christopher Nolan (screenplay), Christopher Nolan (story), David S. Goyer (story), Bob Kane (characters)",
        "Actors": "Christian Bale, Heath Ledger, Aaron Eckhart, Michael Caine",
        "Plot": "When the menace known as the Joker wreaks havoc and chaos on the people of Gotham, the caped crusader must come to terms with one of the greatest psychological tests of his ability to fight injustice.",
        "Language": "English, Mandarin",
        "Country": "USA, UK",
        "Awards": "Won 2 Oscars. Another 146 wins & 142 nominations.",
        "Poster": "http://ia.media-imdb.com/images/M/MV5BMTMxNTUwODM0Nj5BMl5BanBnXkFtZTcwODAyMTk2Mw@._V1_SX300.jpg",
        "Metascore": "82",
        "imdbRating": "9.0",
        "imdbVotes": "1,652,832",
        "imdbID": "tt0468569",
        "Type": "movie",
        "Response": "True"
    }
]
```

```

        {
            "Title": "Batman Begins",
            "Year": "2005",
            "Rated": "PG-13",
            "Released": "15 Jun 2005",
            "Runtime": "140 min",
            "Genre": "Action, Adventure",
            "Director": "Christopher Nolan",
            "Writer": "Bob Kane (characters), David S. Goyer (story), Christopher Nolan (screenplay), David S. Goyer (screenplay)",
            "Actors": "Christian Bale, Michael Caine, Liam Neeson, Katie Holmes",
            "Plot": "After training with his mentor, Batman begins his fight to free crime-ridden Gotham City from the corruption that Scarecrow and the League of Shadows have cast upon it.",
            "Language": "English, Urdu, Mandarin",
            "Country": "USA, UK",
            "Awards": "Nominated for 1 Oscar. Another 15 wins & 66 nominations.",
            "Poster": "http://ia.media-imdb.com/images/M/MV5BNTM3OTc0MzM2OV5BMl5BanBnXkFtZTYwNzUwMTI3._V1_SX300.jpg",
            "Metascore": "70",
            "imdbRating": "8.3",
            "imdbVotes": "972,584",
            "imdbID": "tt0372784",
            "Type": "movie",
            "Response": "True"
        },
        {
            "Title": "Avatar",
            "Year": "2009",
            "Rated": "PG-13",
            "Released": "18 Dec 2009",
            "Runtime": "162 min",
            "Genre": "Action, Adventure, Fantasy",
            "Director": "James Cameron",
            "Writer": "James Cameron",
            "Actors": "Sam Worthington, Zoe Saldana, Sigourney Weaver, Stephen Lang",
            "Plot": "A paraplegic marine dispatched to the moon Pandora on a unique mission becomes torn between following his orders and protecting the world he feels is his home.",
            "Language": "English, Spanish",
            "Country": "USA, UK",
            "Awards": "Won 3 Oscars. Another 80 wins & 121 nominations.",
            "Poster": "http://ia.media-imdb.com/images/M/MV5BMTYwOTEwNjAzMl5BMl5BanBnXkFtZTcwODc5MTUwMw@@._V1_SX300.jpg",
            "Metascore": "83",
            "imdbRating": "7.9",
            "imdbVotes": "876,575",
            "imdbID": "tt0499549",
            "Type": "movie",
            "Response": "True"
        }
    ];
}

function getRating(watchList){
    // Add your code below this line
    var averageRating;

    // Add your code above this line
    return averageRating;
}
console.log(getRating(watchList));

```

Solution

```

// the global variable
var watchList = [
    {
        "Title": "Inception",
        "Year": "2010",
        "Rated": "PG-13",
        "Released": "16 Jul 2010",
        "Runtime": "148 min",
        "Genre": "Action, Adventure, Crime",
        "Director": "Christopher Nolan",
        "Writer": "Christopher Nolan",

```

```

        "Actors": "Leonardo DiCaprio, Joseph Gordon-Levitt, Ellen Page, Tom Hardy",
        "Plot": "A thief, who steals corporate secrets through use of dream-sharing
technology, is given the inverse task of planting an idea into the mind of a CEO.",
        "Language": "English, Japanese, French",
        "Country": "USA, UK",
        "Awards": "Won 4 Oscars. Another 143 wins & 198 nominations.",
        "Poster": "http://ia.media-
imdb.com/images/M/MV5BMjAxMzY3NjcxNF5BMl5BanBnXkFtZTcwNTI50TM0Mw@._V1_SX300.jpg",
        "Metascore": "74",
        "imdbRating": "8.8",
        "imdbVotes": "1,446,708",
        "imdbID": "tt1375666",
        "Type": "movie",
        "Response": "True"
    },
    {
        "Title": "Interstellar",
        "Year": "2014",
        "Rated": "PG-13",
        "Released": "07 Nov 2014",
        "Runtime": "169 min",
        "Genre": "Adventure, Drama, Sci-Fi",
        "Director": "Christopher Nolan",
        "Writer": "Jonathan Nolan, Christopher Nolan",
        "Actors": "Ellen Burstyn, Matthew McConaughey, Mackenzie Foy, John Lithgow",
        "Plot": "A team of explorers travel through a wormhole in space in an attempt to
ensure humanity's survival.",
        "Language": "English",
        "Country": "USA, UK",
        "Awards": "Won 1 Oscar. Another 39 wins & 132 nominations.",
        "Poster": "http://ia.media-
imdb.com/images/M/MV5BMjIxNTU4MzY4MF5BMl5BanBnXkFtZTgwMzM4ODI3MjE@._V1_SX300.jpg",
        "Metascore": "74",
        "imdbRating": "8.6",
        "imdbVotes": "910,366",
        "imdbID": "tt0816692",
        "Type": "movie",
        "Response": "True"
    },
    {
        "Title": "The Dark Knight",
        "Year": "2008",
        "Rated": "PG-13",
        "Released": "18 Jul 2008",
        "Runtime": "152 min",
        "Genre": "Action, Adventure, Crime",
        "Director": "Christopher Nolan",
        "Writer": "Jonathan Nolan (screenplay), Christopher Nolan (screenplay), Christopher
Nolan (story), David S. Goyer (story), Bob Kane (characters)",
        "Actors": "Christian Bale, Heath Ledger, Aaron Eckhart, Michael Caine",
        "Plot": "When the menace known as the Joker wreaks havoc and chaos on the people of
Gotham, the caped crusader must come to terms with one of the greatest psychological tests of his
ability to fight injustice.",
        "Language": "English, Mandarin",
        "Country": "USA, UK",
        "Awards": "Won 2 Oscars. Another 146 wins & 142 nominations.",
        "Poster": "http://ia.media-
imdb.com/images/M/MV5BMTMxNTMwODM0NF5BMl5BanBnXkFtZTcwODAyMTk2Mw@._V1_SX300.jpg",
        "Metascore": "82",
        "imdbRating": "9.0",
        "imdbVotes": "1,652,832",
        "imdbID": "tt0468569",
        "Type": "movie",
        "Response": "True"
    },
    {
        "Title": "Batman Begins",
        "Year": "2005",
        "Rated": "PG-13",
        "Released": "15 Jun 2005",
        "Runtime": "140 min",
        "Genre": "Action, Adventure",
        "Director": "Christopher Nolan",
        "Writer": "Bob Kane (characters), David S. Goyer (story), Christopher Nolan
(screenplay), David S. Goyer (screenplay)",
        "Actors": "Christian Bale, Michael Caine, Liam Neeson, Katie Holmes",
        "Plot": "After training with his mentor, Batman begins his fight to free crime-ridden

```

```

Gotham City from the corruption that Scarecrow and the League of Shadows have cast upon it.",
    "Language": "English, Urdu, Mandarin",
    "Country": "USA, UK",
    "Awards": "Nominated for 1 Oscar. Another 15 wins & 66 nominations.",
    "Poster": "http://ia.media-imdb.com/images/M/MV5BNTM3OTc0MzM2OV5BMl5BanBnXkFtZTYwNzUwMTI3._V1_SX300.jpg",
        "Metascore": "70",
        "imdbRating": "8.3",
        "imdbVotes": "972,584",
        "imdbID": "tt0372784",
        "Type": "movie",
        "Response": "True"
    },
    {
        "Title": "Avatar",
        "Year": "2009",
        "Rated": "PG-13",
        "Released": "18 Dec 2009",
        "Runtime": "162 min",
        "Genre": "Action, Adventure, Fantasy",
        "Director": "James Cameron",
        "Writer": "James Cameron",
        "Actors": "Sam Worthington, Zoe Saldana, Sigourney Weaver, Stephen Lang",
        "Plot": "A paraplegic marine dispatched to the moon Pandora on a unique mission becomes torn between following his orders and protecting the world he feels is his home.",
        "Language": "English, Spanish",
        "Country": "USA, UK",
        "Awards": "Won 3 Oscars. Another 80 wins & 121 nominations.",
        "Poster": "http://ia.media-imdb.com/images/M/MV5BMTYwOTEwNjAzMl5BMl5BanBnXkFtZTcwODc5MTUwMw@@._V1_SX300.jpg",
            "Metascore": "83",
            "imdbRating": "7.9",
            "imdbVotes": "876,575",
            "imdbID": "tt0499549",
            "Type": "movie",
            "Response": "True"
    }
];
}

function getRating(watchList){
    var averageRating;
    const rating = watchList
        .filter(obj => obj.Director === "Christopher Nolan")
        .map(obj => Number(obj.imdbRating));
    averageRating = rating.reduce((accum, curr) => accum + curr)/rating.length;
    return averageRating;
}

```

16. Sort an Array Alphabetically using the sort Method

Description

The `sort` method sorts the elements of an array according to the callback function. For example:

```

function ascendingOrder(arr) {
    return arr.sort(function(a, b) {
        return a - b;
    });
}
ascendingOrder([1, 5, 2, 3, 4]);
// Returns [1, 2, 3, 4, 5]

function reverseAlpha(arr) {
    return arr.sort(function(a, b) {
        return a === b ? 0 : a < b ? 1 : -1;
    });
}
reverseAlpha(['l', 'h', 'z', 'b', 's']);
// Returns [z, s, l, h, b]

```

Note: It's encouraged to provide a callback function to specify how to sort the array items. JavaScript's default sorting method is by string Unicode point value, which may return unexpected results.

Instructions

Use the `sort` method in the `alphabeticalOrder` function to sort the elements of `arr` in alphabetical order.

Challenge Seed

```
function alphabeticalOrder(arr) {
  // Add your code below this line

  // Add your code above this line
}
alphabeticalOrder(["a", "d", "c", "a", "z", "g"]);
```

Solution

```
// solution required
```

17. Return a Sorted Array Without Changing the Original Array

Description

A side effect of the `sort` method is that it changes the order of the elements in the original array. In other words, it mutates the array in place. One way to avoid this is to first concatenate an empty array to the one being sorted (remember that `concat` returns a new array), then run the `sort` method.

Instructions

Use the `sort` method in the `nonMutatingSort` function to sort the elements of an array in ascending order. The function should return a new array, and not mutate the `globalArray` variable.

Challenge Seed

```
var globalArray = [5, 6, 3, 2, 9];
function nonMutatingSort(arr) {
  // Add your code below this line

  // Add your code above this line
}
nonMutatingSort(globalArray);
```

Solution

```
// solution required
```

18. Split a String into an Array Using the `split` Method

Description

The `split` method splits a string into an array of strings. It takes an argument for the delimiter, which can be a character to use to break up the string or a regular expression. For example, if the delimiter is a space, you get an array of words, and if the delimiter is an empty string, you get an array of each character in the string. Here are two examples that split one string by spaces, then another by digits using a regular expression:

```
var str = "Hello World";
var bySpace = str.split(" ");
// Sets bySpace to ["Hello", "World"]

var otherString = "How9are7you2today";
var byDigits = otherString.split(/\d/);
// Sets byDigits to ["How", "are", "you", "today"]
```

Since strings are immutable, the `split` method makes it easier to work with them.

Instructions

Use the `split` method inside the `splitify` function to split `str` into an array of words. The function should return the array. Note that the words are not always separated by spaces, and the array should not contain punctuation.

Challenge Seed

```
function splitify(str) {
  // Add your code below this line

  // Add your code above this line
}
splitify("Hello World,I-am code");
```

Solution

```
// solution required
```

19. Combine an Array into a String Using the `join` Method

Description

The `join` method is used to join the elements of an array together to create a string. It takes an argument for the delimiter that is used to separate the array elements in the string. Here's an example:

```
var arr = ["Hello", "World"];
var str = arr.join(" ");
// Sets str to "Hello World"
```

Instructions

Use the `join` method (among others) inside the `sentensify` function to make a sentence from the words in the string `str`. The function should return a string. For example, "I-like-Star-Wars" would be converted to "I like Star Wars". For this challenge, do not use the `replace` method.

Challenge Seed

```
function sentensify(str) {
  // Add your code below this line

  // Add your code above this line
}

sentensify("May-the-force-be-with-you");
```

Solution

```
// solution required
```

20. Apply Functional Programming to Convert Strings to URL Slugs

Description

The last several challenges covered a number of useful array and string methods that follow functional programming principles. We've also learned about `reduce`, which is a powerful method used to reduce problems to simpler forms. From computing averages to sorting, any array operation can be achieved by applying it. Recall that `map` and `filter` are special cases of `reduce`. Let's combine what we've learned to solve a practical problem. Many content management sites (CMS) have the titles of a post added to part of the URL for simple bookmarking purposes. For example, if you write a Medium post titled "Stop Using Reduce", it's likely the URL would have some form of the title string in it (".../stop-using-reduce"). You may have already noticed this on the freeCodeCamp site.

Instructions

Fill in the `urlSlug` function so it converts a string `title` and returns the hyphenated version for the URL. You can use any of the methods covered in this section, and don't use `replace`. Here are the requirements: The input is a string with spaces and title-cased words The output is a string with the spaces between words replaced by a hyphen (-) The output should be all lower-cased letters The output should not have any spaces

Challenge Seed

```
// the global variable
var globalTitle = "Winter Is Coming";

// Add your code below this line
function urlSlug(title) {

}

// Add your code above this line

var winterComing = urlSlug(globalTitle); // Should be "winter-is-coming"
```

Solution

```
// solution required
```

21. Use the `every` Method to Check that Every Element in an Array Meets a Criteria

Description

The `every` method works with arrays to check if *every* element passes a particular test. It returns a Boolean value - `true` if all values meet the criteria, `false` if not. For example, the following code would check if every element in the `numbers` array is less than 10:

```
var numbers = [1, 5, 8, 0, 10, 11];
numbers.every(function(currentValue) {
  return currentValue < 10;
});
// Returns false
```

Instructions

Use the `every` method inside the `checkPositive` function to check if every element in `arr` is positive. The function should return a Boolean value.

Challenge Seed

```
function checkPositive(arr) {
  // Add your code below this line

  // Add your code above this line
}
checkPositive([1, 2, 3, -4, 5]);
```

Solution

```
// solution required
```

22. Use the `some` Method to Check that Any Elements in an Array Meet a Criteria

Description

The `some` method works with arrays to check if *any* element passes a particular test. It returns a Boolean value - `true` if any of the values meet the criteria, `false` if not. For example, the following code would check if any element in the `numbers` array is less than 10:

```
var numbers = [10, 50, 8, 220, 110, 11];
numbers.some(function(currentValue) {
  return currentValue < 10;
});
// Returns true
```

Instructions

Use the `some` method inside the `checkPositive` function to check if any element in `arr` is positive. The function should return a Boolean value.

Challenge Seed

```
function checkPositive(arr) {
  // Add your code below this line

  // Add your code above this line
```

```

    }
checkPositive([1, 2, 3, -4, 5]);

```

Solution

```
// solution required
```

23. Introduction to Currying and Partial Application

Description

The **arity** of a function is the number of arguments it requires. Currying a function means to convert a function of N arity into N functions of arity 1. In other words, it restructures a function so it takes one argument, then returns another function that takes the next argument, and so on. Here's an example:

```

//Un-curried function
function unCurried(x, y) {
  return x + y;
}

//Curried function
function curried(x) {
  return function(y) {
    return x + y;
  }
}

//Alternative using ES6
const curried = x => y => x + y

curried(1)(2) // Returns 3

```

This is useful in your program if you can't supply all the arguments to a function at one time. You can save each function call into a variable, which will hold the returned function reference that takes the next argument when it's available. Here's an example using the `curried` function in the example above:

```

// Call a curried function in parts:
var funcForY = curried(1);
console.log(funcForY(2)); // Prints 3

```

Similarly, partial application can be described as applying a few arguments to a function at a time and returning another function that is applied to more arguments. Here's an example:

```

//Impartial function
function impartial(x, y, z) {
  return x + y + z;
}

var partialFn = impartial.bind(this, 1, 2);
partialFn(10); // Returns 13

```

Instructions

Fill in the body of the `add` function so it uses currying to add parameters `x`, `y`, and `z`.

Challenge Seed

```

function add(x) {
  // Add your code below this line

  // Add your code above this line

```

```
}

add(10)(20)(30);
```

Solution

```
const add = x => y => z => x + y + z
```

Intermediate Algorithm Scripting

1. Sum All Numbers in a Range

Description

We'll pass you an array of two numbers. Return the sum of those two numbers plus the sum of all the numbers between them. The lowest number will not always come first. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```
function sumAll(arr) {
  return 1;
}

sumAll([1, 4]);
```

Solution

```
function sumAll(arr) {
  var sum = 0;
  arr.sort(function(a,b) {return a-b;});
  for (var i = arr[0]; i <= arr[1]; i++) {
    sum += i;
  }
  return sum;
}
```

2. Diff Two Arrays

Description

Compare two arrays and return a new array with any items only found in one of the two given arrays, but not both. In other words, return the symmetric difference of the two arrays. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code. Note You can return the array with its elements in any order.

Instructions

Challenge Seed

```

function diffArray(arr1, arr2) {
  var newArr = [];
  // Same, same; but different.
  return newArr;
}

diffArray([1, 2, 3, 5], [1, 2, 3, 4, 5]);

```

Solution

```

function diffArray(arr1, arr2) {
  var newArr = [];
  var h1 = Object.create(null);
  arr1.forEach(function(e) {
    h1[e] = e;
  });

  var h2 = Object.create(null);
  arr2.forEach(function(e) {
    h2[e] = e;
  });

  Object.keys(h1).forEach(function(e) {
    if (!(e in h2)) newArr.push(h1[e]);
  });
  Object.keys(h2).forEach(function(e) {
    if (!(e in h1)) newArr.push(h2[e]);
  });
  // Same, same; but different.
  return newArr;
}

```

3. Seek and Destroy

Description

You will be provided with an initial array (the first argument in the destroyer function), followed by one or more arguments. Remove all elements from the initial array that are of the same value as these arguments. Note You have to use the `arguments` object. Remember to use [Read-Search-Ask](#) if you get stuck. Write your own code.

Instructions

Challenge Seed

```

function destroyer(arr) {
  // Remove all the values
  return arr;
}

destroyer([1, 2, 3, 1, 2, 3], 2, 3);

```

Solution

```

function destroyer(arr) {
  var hash = Object.create(null);
  [].slice.call(arguments, 1).forEach(function(e) {
    hash[e] = true;
  });
  // Remove all the values
  return arr.filter(function(e) { return !(e in hash); });
}

```

```
destroyer([1, 2, 3, 1, 2, 3], 2, 3);
```

4. Wherefore art thou

Description

Make a function that looks through an array of objects (first argument) and returns an array of all objects that have matching name and value pairs (second argument). Each name and value pair of the source object has to be present in the object from the collection if it is to be included in the returned array. For example, if the first argument is `[{ first: "Romeo", last: "Montague" }, { first: "Mercutio", last: null }, { first: "Tybalt", last: "Capulet" }]`, and the second argument is `{ last: "Capulet" }`, then you must return the third object from the array (the first argument), because it contains the name and its value, that was passed on as the second argument. Remember to use [Read-Search-Ask](#) if you get stuck. Write your own code.

Instructions

Challenge Seed

```
function whatIsInAName(collection, source) {
  // What's in a name?
  var arr = [];
  // Only change code below this line

  // Only change code above this line
  return arr;
}

whatIsInAName([{ first: "Romeo", last: "Montague" }, { first: "Mercutio", last: null }, { first: "Tybalt", last: "Capulet" }], { last: "Capulet" });
```

Solution

```
function whatIsInAName(collection, source) {
  var arr = [];
  var keys = Object.keys(source);
  collection.forEach(function(e) {
    if(keys.every(function(key) {return e[key] === source[key];})) {
      arr.push(e);
    }
  });
  return arr;
}
```

5. Spinal Tap Case

Description

Convert a string to spinal case. Spinal case is all-lowercase-words-joined-by-dashes. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```

function spinalCase(str) {
  // "It's such a fine line between stupid, and clever."
  // --David St. Hubbins
  return str;
}

spinalCase('This Is Spinal Tap');

```

Solution

```

function spinalCase(str) {
  // "It's such a fine line between stupid, and clever."
  // --David St. Hubbins
  str = str.replace(/([a-z](?=[A-Z]))/g, '$1 ');
  return str.toLowerCase().replace(/\ |\_/.g, '-');
}

```

6. Pig Latin

Description

Translate the provided string to pig latin. [Pig Latin](#) takes the first consonant (or consonant cluster) of an English word, moves it to the end of the word and suffixes an "ay". If a word begins with a vowel you just add "way" to the end. If a word does not contain a vowel, just add "ay" to the end. Input strings are guaranteed to be English words in all lowercase. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```

function translatePigLatin(str) {
  return str;
}

translatePigLatin("consonant");

```

Solution

```

function translatePigLatin(str) {
  if (isVowel(str.charAt(0))) return str + "way";
  var front = [];
  str = str.split('');
  while (str.length && !isVowel(str[0])) {
    front.push(str.shift());
  }
  return [].concat(str, front).join('') + 'ay';

  function isVowel(c) {
    return ['a', 'e', 'i', 'o', 'u'].indexOf(c.toLowerCase()) !== -1;
  }
}

```

7. Search and Replace

Description

Perform a search and replace on the sentence using the arguments provided and return the new sentence. First argument is the sentence to perform the search and replace on. Second argument is the word that you will be replacing (before). Third argument is what you will be replacing the second argument with (after). Note Preserve the case of the first character in the original word when you are replacing it. For example if you mean to replace the word "Book" with the word "dog", it should be replaced as "Dog" Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```
function myReplace(str, before, after) {
  return str;
}

myReplace("A quick brown fox jumped over the lazy dog", "jumped", "leaped");
```

Solution

```
function myReplace(str, before, after) {
  if (before.charAt(0) === before.charAt(0).toUpperCase()) {
    after = after.charAt(0).toUpperCase() + after.substring(1);
  } else {
    after = after.charAt(0).toLowerCase() + after.substring(1);
  }
  return str.replace(before, after);
}
```

8. DNA Pairing

Description

The DNA strand is missing the pairing element. Take each character, get its pair, and return the results as a 2d array. [Base pairs](#) are a pair of AT and CG. Match the missing element to the provided character. Return the provided character as the first element in each array. For example, for the input GCG, return [[["G", "C"], ["C", "G"], ["G", "C"]]] The character and its pair are paired up in an array, and all the arrays are grouped into one encapsulating array. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```
function pairElement(str) {
  return str;
}

pairElement("GCG");
```

Solution

```
var lookup = Object.create(null);
lookup.A = 'T';
lookup.T = 'A';
lookup.C = 'G';
lookup.G = 'C';
```

```
function pairElement(str) {
  return str.split('').map(function(p) {return [p, lookup[p]];});
}
```

9. Missing letters

Description

Find the missing letter in the passed letter range and return it. If all letters are present in the range, return undefined. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```
function fearNotLetter(str) {
  return str;
}

fearNotLetter("abce");
```

Solution

```
function fearNotLetter (str) {
  for (var i = str.charCodeAt(0); i <= str.charCodeAt(str.length - 1); i++) {
    var letter = String.fromCharCode(i);
    if (str.indexOf(letter) === -1) {
      return letter;
    }
  }
  return undefined;
}
```

10. Sorted Union

Description

Write a function that takes two or more arrays and returns a new array of unique values in the order of the original provided arrays. In other words, all values present from all arrays should be included in their original order, but with no duplicates in the final array. The unique numbers should be sorted by their original order, but the final array should not be sorted in numerical order. Check the assertion tests for examples. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```
function uniteUnique(arr) {
  return arr;
}

uniteUnique([1, 3, 2], [5, 2, 1, 4], [2, 1]);
```

Solution

```
function uniteUnique(arr) {
  return [].slice.call(arguments).reduce(function(a, b) {
    return [].concat(a, b.filter(function(e) {return a.indexOf(e) === -1;}));
  }, []);
}
```

11. Convert HTML Entities

Description

Convert the characters & , < , > , " (double quote), and ' (apostrophe), in a string to their corresponding HTML entities. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```
function convertHTML(str) {
  // &colon;&rpar;
  return str;
}

convertHTML("Dolce & Gabbana");
```

Solution

```
var MAP = { '&': '&amp;',
  '<': '&lt;',
  '>': '&gt;',
  '\"': '&quot;',
  '\''': '&apos;'};

function convertHTML(str) {
  return str.replace(/([&<>"'])/g, function(c) {
    return MAP[c];
  });
}
```

12. Sum All Odd Fibonacci Numbers

Description

Given a positive integer `num` , return the sum of all odd Fibonacci numbers that are less than or equal to `num` . The first two numbers in the Fibonacci sequence are 1 and 1. Every additional number in the sequence is the sum of the two previous numbers. The first six numbers of the Fibonacci sequence are 1, 1, 2, 3, 5 and 8. For example, `sumFibs(10)` should return `10` because all odd Fibonacci numbers less than or equal to `10` are 1, 1, 3, and 5. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```
function sumFibs(num) {
  return num;
}

sumFibs(4);
```

Solution

```
function sumFibs(num) {
  var a = 1;
  var b = 1;
  var s = 0;
  while (a <= num) {
    if (a % 2 !== 0) {
      s += a;
    }
    a = [b, b+b+a][0];
  }
  return s;
}
```

13. Sum All Primes

Description

Sum all the prime numbers up to and including the provided number. A prime number is defined as a number greater than one and having only two divisors, one and itself. For example, 2 is a prime number because it's only divisible by one and two. The provided number may not be a prime. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```
function sumPrimes(num) {
  return num;
}

sumPrimes(10);
```

Solution

```
function eratosthenesArray(n) {
  var primes = [];
  if (n > 2) {
    var half = n>>1;
    var sieve = Array(half);
    for (var i = 1, limit = Math.sqrt(n)>>1; i <= limit; i++) {
      if (!sieve[i]) {
        for (var step = 2*i+1, j = (step*step)>>1; j < half; j+=step) {
          sieve[j] = true;
        }
      }
    }
    primes.push(2);
    for (var p = 1; p < half; p++) {
      if (!sieve[p]) primes.push(2*p+1);
    }
  }
  return primes;
}
```

```
function sumPrimes(num) {
  return eratosthenesArray(num+1).reduce(function(a,b) {return a+b;}, 0);
}

sumPrimes(10);
```

14. Smallest Common Multiple

Description

Find the smallest common multiple of the provided parameters that can be evenly divided by both, as well as by all sequential numbers in the range between these parameters. The range will be an array of two numbers that will not necessarily be in numerical order. For example, if given 1 and 3, find the smallest common multiple of both 1 and 3 that is also evenly divisible by all numbers *between* 1 and 3. The answer here would be 6. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```
function smallestCommons(arr) {
  return arr;
}

smallestCommons([1,5]);
```

Solution

```
function gcd(a, b) {
  while (b !== 0) {
    a = [b, b = a % b][0];
  }
  return a;
}

function lcm(a, b) {
  return (a * b) / gcd(a, b);
}

function smallestCommons(arr) {
  arr.sort(function(a,b) {return a-b;});
  var rng = [];
  for (var i = arr[0]; i <= arr[1]; i++) {
    rng.push(i);
  }
  return rng.reduce(lcm);
}
```

15. Drop it

Description

Given the array `arr`, iterate through and remove each element starting from the first element (the 0 index) until the function `func` returns `true` when the iterated element is passed through it. Then return the rest of the array once the condition is satisfied, otherwise, `arr` should be returned as an empty array. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```
function dropElements(arr, func) {
  // Drop them elements.
  return arr;
}

dropElements([1, 2, 3], function(n) {return n < 3;});
```

Solution

```
function dropElements(arr, func) {
  // Drop them elements.
  while (arr.length && !func(arr[0])) {
    arr.shift();
  }
  return arr;
}
```

16. Steamroller

Description

Flatten a nested array. You must account for varying levels of nesting. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```
function steamrollArray(arr) {
  // I'm a steamroller, baby
  return arr;
}

steamrollArray([1, [2], [3, [[4]]]]);
```

Solution

```
function steamrollArray(arr) {
  if (!Array.isArray(arr)) {
    return [arr];
  }
  var out = [];
  arr.forEach(function(e) {
    steamrollArray(e).forEach(function(v) {
      out.push(v);
    });
  });
  return out;
}
```

17. Binary Agents

Description

Return an English translated sentence of the passed binary string. The binary string will be space separated. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```
function binaryAgent(str) {
  return str;
}

binaryAgent("01000001 0110010 01100101 01101110 00100111 01110100 00100000 01100010 01101111 01101110
01100110 01101001 01110010 01100101 01110011 00100000 01100110 01110101 01101110 00100001 00111111");
```

Solution

```
function binaryAgent(str) {
  return str.split(' ').map(function(s) { return parseInt(s, 2); }).map(function(b) { return
String.fromCharCode(b); }).join('');
}
```

18. Everything Be True

Description

Check if the predicate (second argument) is truthy on all elements of a collection (first argument). In other words, you are given an array collection of objects. The predicate `pre` will be an object property and you need to return `true` if its value is `truthy`. Otherwise, return `false`. In JavaScript, `truthy` values are values that translate to `true` when evaluated in a Boolean context. Remember, you can access object properties through either dot notation or `[]` notation. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```
function truthCheck(collection, pre) {
  // Is everyone being true?
  return pre;
}

truthCheck([{ "user": "Tinky-Winky", "sex": "male"}, {"user": "Dipsy", "sex": "male"}, {"user": "Laa-Laa", "sex": "female"}, {"user": "Po", "sex": "female"}], "sex");
```

Solution

```
function truthCheck(collection, pre) {
  // Does everyone have one of these?
  return collection.every(function(e) { return e[pre]; });
}
```

19. Arguments Optional

Description

Create a function that sums two arguments together. If only one argument is provided, then return a function that expects one argument and returns the sum. For example, `addTogether(2, 3)` should return `5`, and `addTogether(2)` should return a function. Calling this returned function with a single argument will then return the sum: `var sumTwoAnd = addTogether(2); sumTwoAnd(3)` returns `5`. If either argument isn't a valid number, return `undefined`. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```
function addTogether() {
  return false;
}

addTogether(2,3);
```

Solution

```
function addTogether() {
  var a = arguments[0];
  if (toString.call(a) !== '[object Number]') return;
  if (arguments.length === 1) {
    return function(b) {
      if (toString.call(b) !== '[object Number]') return;
      return a + b;
    };
  }
  var b = arguments[1];
  if (toString.call(b) !== '[object Number]') return;
  return a + arguments[1];
}
```

20. Make a Person

Description

Fill in the object constructor with the following methods below:

```
getFirstName() getLastName() getFullName() setFirstName(first) setLastName(last)
setFullName(firstAndLast)
```

Run the tests to see the expected output for each method. The methods that take an argument must accept only one argument and it has to be a string. These methods must be the only available means of interacting with the object. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```
var Person = function(firstAndLast) {
  // Complete the method below and implement the others similarly
  this.getFullName = function() {
    return "";
  };
}
```

```

    return firstAndLast;
};

var bob = new Person('Bob Ross');
bob.getFullName();

```

Solution

```

var Person = function(firstAndLast) {

    var firstName, lastName;

    function updateName(str) {
        firstName = str.split(" ")[0];
        lastName = str.split(" ")[1];
    }

    updateName(firstAndLast);

    this.getFirstName = function(){
        return firstName;
    };

    this.getLastName = function(){
        return lastName;
    };

    this.getFullName = function(){
        return firstName + " " + lastName;
    };

    this.setFirstName = function(str){
        firstName = str;
    };

    this.setLastName = function(str){
        lastName = str;
    };

    this.setFullName = function(str){
        updateName(str);
    };
};

var bob = new Person('Bob Ross');
bob.getFullName();

```

21. Map the Debris

Description

Return a new array that transforms the elements' average altitude into their orbital periods (in seconds). The array will contain objects in the format `{name: 'name', avgAlt: avgAlt}`. You can read about orbital periods [on Wikipedia](#). The values should be rounded to the nearest whole number. The body being orbited is Earth. The radius of the earth is 6367.4447 kilometers, and the GM value of earth is $398600.4418 \text{ km}^3\text{s}^{-2}$. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```

function orbitalPeriod(arr) {
    var GM = 398600.4418;

```

```

var earthRadius = 6367.4447;
return arr;
}

orbitalPeriod([{name : "sputnik", avgAlt : 35873.5553}]);

```

Solution

```

function orbitalPeriod(arr) {
  var GM = 398600.4418;
  var earthRadius = 6367.4447;
  var TAU = 2 * Math.PI;
  return arr.map(function(obj) {
    return {
      name: obj.name,
      orbitalPeriod: Math.round(TAU * Math.sqrt(Math.pow(obj.avgAlt+earthRadius, 3)/GM))
    };
  });
}

orbitalPeriod([{name : "sputkin", avgAlt : 35873.5553}]);

```

JavaScript Algorithms and Data Structures Projects

1. Palindrome Checker

Description

Return `true` if the given string is a palindrome. Otherwise, return `false`. A palindrome is a word or sentence that's spelled the same way both forward and backward, ignoring punctuation, case, and spacing. Note You'll need to remove all non-alphanumeric characters (punctuation, spaces and symbols) and turn everything into the same case (lower or upper case) in order to check for palindromes. We'll pass strings with varying formats, such as `"racecar"`, `"RaceCar"`, and `"race CAR"` among others. We'll also pass strings with special symbols, such as `"2A3*3a2"`, `"2A3 3a2"`, and `"2_A3*3#A2"`. Remember to use [Read-Search-Ask](#) if you get stuck. Write your own code.

Instructions

Challenge Seed

```

function palindrome(str) {
  // Good luck!
  return true;
}

```

```
palindrome("eye");
```

Solution

```

function palindrome(str) {
  var string = str.toLowerCase().split(/[^A-Za-z0-9]/gi).join('');
  var aux = string.split('');
  if (aux.join('') === aux.reverse().join('')){
    return true;
  }
}

```

```

    return false;
}

```

2. Roman Numeral Converter

Description

Convert the given number into a roman numeral. All [roman numerals](#) answers should be provided in upper-case. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```

function convertToRoman(num) {
  return num;
}

convertToRoman(36);

```

Solution

```

function convertToRoman(num) {
  var ref = [['M', 1000], ['CM', 900], ['D', 500], ['CD', 400], ['C', 100], ['XC', 90], ['L', 50],
  ['XL', 40], ['X', 10], ['IX', 9], ['V', 5], ['IV', 4], ['I', 1]];
  var res = [];
  ref.forEach(function(p) {
    while (num >= p[1]) {
      res.push(p[0]);
      num -= p[1];
    }
  });
  return res.join('');
}

```

3. Caesars Cipher

Description

One of the simplest and most widely known ciphers is a Caesar cipher , also known as a shift cipher . In a shift cipher the meanings of the letters are shifted by some set amount. A common modern use is the ROT13 cipher, where the values of the letters are shifted by 13 places. Thus 'A' ↔ 'N', 'B' ↔ 'O' and so on. Write a function which takes a ROT13 encoded string as input and returns a decoded string. All letters will be uppercase. Do not transform any non-alphabetic character (i.e. spaces, punctuation), but do pass them on. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```

function rot13(str) { // LBH QVQ VG!

  return str;
}

```

```
// Change the inputs below to test
rot13("SERR PBQR PNZC");
```

Solution

```
var lookup = {
  'A': 'N', 'B': 'O', 'C': 'P', 'D': 'Q',
  'E': 'R', 'F': 'S', 'G': 'T', 'H': 'U',
  'I': 'V', 'J': 'W', 'K': 'X', 'L': 'Y',
  'M': 'Z', 'N': 'A', 'O': 'B', 'P': 'C',
  'Q': 'D', 'R': 'E', 'S': 'F', 'T': 'G',
  'U': 'H', 'V': 'I', 'W': 'J', 'X': 'K',
  'Y': 'L', 'Z': 'M'
};

function rot13(encodedStr) {
  var codeArr = encodedStr.split(""); // String to Array
  var decodedArr = []; // Your Result goes here
  // Only change code below this line

  decodedArr = codeArr.map(function(letter) {
    if(lookup.hasOwnProperty(letter)) {
      letter = lookup[letter];
    }
    return letter;
  });

  // Only change code above this line
  return decodedArr.join("") // Array to String
}
```

4. Telephone Number Validator

Description

Return `true` if the passed string looks like a valid US phone number. The user may fill out the form field any way they choose as long as it has the format of a valid US number. The following are examples of valid formats for US numbers (refer to the tests below for other variants):

```
555-555-5555
(555)555-5555
(555) 555-5555
555 555 5555
5555555555
1 555 555 5555
```

For this challenge you will be presented with a string such as `800-692-7753` or `800-six427676;laskdjf`. Your job is to validate or reject the US phone number based on any combination of the formats provided above. The area code is required. If the country code is provided, you must confirm that the country code is `1`. Return `true` if the string is a valid US phone number; otherwise return `false`. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Instructions

Challenge Seed

```
function telephoneCheck(str) {
  // Good luck!
  return true;
}

telephoneCheck("555-555-5555");
```

Solution

```
var re = /^( [+]?1[\s]?)(?:[([(:(?:[2-9]1[02-9]|[2-9][02-8][0-9])])[ \s?])|(?:(?:[2-9]1[02-9]|[2-9][02-8][0-9])[\s.-?])}{1}([2-9]1[02-9]|[2-9][02-9]1|[2-9][02-9]{2}[\s.-?])}{1}([0-9]{4}){1}$/;

function telephoneCheck(str) {
  return re.test(str);
}

telephoneCheck("555-555-5555");
```

5. Cash Register

Description

Design a cash register drawer function `checkCashRegister()` that accepts purchase price as the first argument (`price`), payment as the second argument (`cash`), and cash-in-drawer (`cid`) as the third argument. `cid` is a 2D array listing available currency. The `checkCashRegister()` function should always return an object with a `status` key and a `change` key. Return `{status: "INSUFFICIENT_FUNDS", change: []}` if cash-in-drawer is less than the change due, or if you cannot return the exact change. Return `{status: "CLOSED", change: [...]}` with cash-in-drawer as the value for the key `change` if it is equal to the change due. Otherwise, return `{status: "OPEN", change: [...]}`, with the change due in coins and bills, sorted in highest to lowest order, as the value of the `change` key. Remember to use [Read-Search-Ask](#) if you get stuck. Try to pair program. Write your own code.

Currency Unit	Amount
Penny	\$0.01 (PENNY)
Nickel	\$0.05 (NICKEL)
Dime	\$0.1 (DIME)
Quarter	\$0.25 (QUARTER)
Dollar	\$1 (DOLLAR)
Five Dollars	\$5 (FIVE)
Ten Dollars	\$10 (TEN)
Twenty Dollars	\$20 (TWENTY)
One-hundred Dollars	\$100 (ONE HUNDRED)

Instructions

Challenge Seed

```
function checkCashRegister(price, cash, cid) {
  var change;
  // Here is your change, ma'am.
  return change;
}

// Example cash-in-drawer array:
// [["PENNY", 1.01],
//  ["NICKEL", 2.05],
//  ["DIME", 3.1],
//  ["QUARTER", 4.25],
//  ["ONE", 90],
//  ["FIVE", 55],
//  ["TEN", 20],
//  ["TWENTY", 60],
//  ["ONE HUNDRED", 100]]
```

```
checkCashRegister(19.5, 20, [["PENNY", 1.01], ["NICKEL", 2.05], ["DIME", 3.1], ["QUARTER", 4.25], ["ONE", 90], ["FIVE", 55], ["TEN", 20], ["TWENTY", 60], ["ONE HUNDRED", 100]]);
```

Solution

```
var denom = [
  { name: 'ONE HUNDRED', val: 100},
  { name: 'TWENTY', val: 20},
  { name: 'TEN', val: 10},
  { name: 'FIVE', val: 5},
  { name: 'ONE', val: 1},
  { name: 'QUARTER', val: 0.25},
  { name: 'DIME', val: 0.1},
  { name: 'NICKEL', val: 0.05},
  { name: 'PENNY', val: 0.01}
];

function checkCashRegister(price, cash, cid) {
  var output = {status: null, change: []};
  var change = cash - price;
  var register = cid.reduce(function(acc, curr) {
    acc.total += curr[1];
    acc[curr[0]] = curr[1];
  }, {total: 0});
  if(register.total === change) {
    output.status = 'CLOSED';
    output.change = cid;
    return output;
  }
  if(register.total < change) {
    output.status = 'INSUFFICIENT_FUNDS';
    return output;
  }
  var change_arr = denom.reduce(function(acc, curr) {
    var value = 0;
    while(register[curr.name] > 0 && change >= curr.val) {
      change -= curr.val;
      register[curr.name] -= curr.val;
      value += curr.val;
      change = Math.round(change * 100) / 100;
    }
    if(value > 0) {
      acc.push([ curr.name, value ]);
    }
  }, []);
  if(change_arr.length < 1 || change > 0) {
    output.status = 'INSUFFICIENT_FUNDS';
    return output;
  }
  output.status = 'OPEN';
  output.change = change_arr;
  return output;
}
```

Front End Libraries Certification

Bootstrap

1. Use Responsive Design with Bootstrap Fluid Containers

Description

In the HTML5 and CSS section of freeCodeCamp we built a Cat Photo App. Now let's go back to it. This time, we'll style it using the popular Bootstrap responsive CSS framework. Bootstrap will figure out how wide your screen is and respond by resizing your HTML elements - hence the name `Responsive Design`. With responsive design, there is no need to design a mobile version of your website. It will look good on devices with screens of any width. You can add Bootstrap to any app by adding the following code to the top of your HTML: `<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg320mUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous"/>` In this case, we've already added it for you to this page behind the scenes. Note that using either `>` or `/>` to close the `link` tag is acceptable. To get started, we should nest all of our HTML (except the `link` tag and the `style` element) in a `div` element with the class `container-fluid`.

Instructions

Challenge Seed

```
<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, Monospace;
  }

  p {
    font-size: 16px;
    font-family: Monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }

  .smaller-image {
    width: 100px;
  }
</style>

<h2 class="red-text">CatPhotoApp</h2>

<p>Click here for <a href="#">cat photos</a>.</p>

<a href="#"></a>

<p>Things cats love:</p>
<ul>
  <li>cat nip</li>
  <li>laser pointers</li>
  <li>lasagna</li>
</ul>
```

```

<p>Top 3 things cats hate:</p>
<ol>
  <li>flea treatment</li>
  <li>thunder</li>
  <li>other cats</li>
</ol>
<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor"> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
  <label><input type="checkbox" name="personality"> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Crazy</label>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>

```

Solution

```
// solution required
```

2. Make Images Mobile Responsive

Description

First, add a new image below the existing one. Set its `src` attribute to `https://bit.ly/fcc-running-cats`. It would be great if this image could be exactly the width of our phone's screen. Fortunately, with Bootstrap, all we need to do is add the `img-responsive` class to your image. Do this, and the image should perfectly fit the width of your page.

Instructions

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, Monospace;
  }

  p {
    font-size: 16px;
    font-family: Monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }

  .smaller-image {
    width: 100px;
  }
</style>

<div class="container-fluid">
  <h2 class="red-text">CatPhotoApp</h2>
  <p>Click here for <a href="#">cat photos</a>.</p>
  <a href="#"></a>

<p>Things cats love:</p>
<ul>
  <li>cat nip</li>
  <li>laser pointers</li>
  <li>lasagna</li>
</ul>
<p>Top 3 things cats hate:</p>
<ol>
  <li>flea treatment</li>
  <li>thunder</li>
  <li>other cats</li>
</ol>
<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor"> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
  <label><input type="checkbox" name="personality"> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Crazy</label>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</div>
```

Solution

```
// solution required
```

3. Center Text with Bootstrap

Description

Now that we're using Bootstrap, we can center our heading element to make it look better. All we need to do is add the class `text-center` to our `h2` element. Remember that you can add several classes to the same element by separating each of them with a space, like this: `<h2 class="red-text text-center">your text</h2>`

Instructions

Challenge Seed

```
<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, Monospace;
  }

  p {
    font-size: 16px;
    font-family: Monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }

  .smaller-image {
    width: 100px;
  }
</style>
```

```

        }
    </style>

<div class="container-fluid">
    <h2 class="red-text">CatPhotoApp</h2>

    <p>Click here for <a href="#">cat photos</a>.</p>

    <a href="#"></a>

    
    <p>Things cats love:</p>
    <ul>
        <li>cat nip</li>
        <li>laser pointers</li>
        <li>lasagna</li>
    </ul>
    <p>Top 3 things cats hate:</p>
    <ol>
        <li>flea treatment</li>
        <li>thunder</li>
        <li>other cats</li>
    </ol>
    <form action="/submit-cat-photo">
        <label><input type="radio" name="indoor-outdoor"> Indoor</label>
        <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
        <label><input type="checkbox" name="personality"> Loving</label>
        <label><input type="checkbox" name="personality"> Lazy</label>
        <label><input type="checkbox" name="personality"> Crazy</label>
        <input type="text" placeholder="cat photo URL" required>
        <button type="submit">Submit</button>
    </form>
</div>

```

Solution

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
    .red-text {
        color: red;
    }

    h2 {
        font-family: Lobster, Monospace;
    }

    p {
        font-size: 16px;
        font-family: Monospace;
    }

    .thick-green-border {
        border-color: green;
        border-width: 10px;
        border-style: solid;
        border-radius: 50%;
    }

    .smaller-image {
        width: 100px;
    }
</style>

<div class="container-fluid">
    <h2 class="red-text text-center">CatPhotoApp</h2>

    <p>Click here for <a href="#">cat photos</a>.</p>

    <a href="#"></a>

```

```

the camera.">
<p>Things cats love:</p>
<ul>
  <li>cat nip</li>
  <li>laser pointers</li>
  <li>lasagna</li>
</ul>
<p>Top 3 things cats hate:</p>
<ol>
  <li>flea treatment</li>
  <li>thunder</li>
  <li>other cats</li>
</ol>
<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor"> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
  <label><input type="checkbox" name="personality"> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Crazy</label>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</div>

```

4. Create a Bootstrap Button

Description

Bootstrap has its own styles for `button` elements, which look much better than the plain HTML ones. Create a new `button` element below your large kitten photo. Give it the `btn` and `btn-default` classes, as well as the text of "Like".

Instructions

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, Monospace;
  }

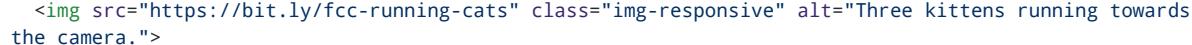
  p {
    font-size: 16px;
    font-family: Monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }

  .smaller-image {
    width: 100px;
  }
</style>

<div class="container-fluid">
  <h2 class="red-text text-center">CatPhotoApp</h2>

  <p>Click here for <a href="#">cat photos</a>.</p>
  <a href="#"></a>

Three kittens running towards the camera.



Things cats love:



- cat nip
- laser pointers
- lasagna



Top 3 things cats hate:



- flea treatment
- thunder
- other cats


   Indoor
   Outdoor
   Loving
   Lazy
   Crazy
  
  Submit


```

Solution

```

<html>
<head>
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
.red-text {
  color: red;
}

h2 {
  font-family: Lobster, Monospace;
}

p {
  font-size: 16px;
  font-family: Monospace;
}

.thick-green-border {
  border-color: green;
  border-width: 10px;
  border-style: solid;
  border-radius: 50%;
}

.smaller-image {
  width: 100px;
}
</style>
</head>
<body>
<div class="container-fluid">
  <h2 class="red-text text-center">CatPhotoApp</h2>
  <p>Click here for <a href="#">cat photos</a>.</p>
  <a href="#"></a>
  
  <!-- ADD Bootstrap Styled Button -->
  <button class="btn btn-default">Like</button>
</div>

```

```

<p>Things cats love:</p>
<ul>
  <li>cat nip</li>
  <li>laser pointers</li>
  <li>lasagna</li>
</ul>
<p>Top 3 things cats hate:</p>
<ol>
  <li>flea treatment</li>
  <li>thunder</li>
  <li>other cats</li>
</ol>
<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor"> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
  <label><input type="checkbox" name="personality"> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Crazy</label>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</div>
</html>

```

5. Create a Block Element Bootstrap Button

Description

Normally, your button elements with the `btn` and `btn-default` classes are only as wide as the text that they contain. For example: `<button class="btn btn-default">Submit</button>` This button would only be as wide as the word "Submit". Submit By making them block elements with the additional class of `btn-block`, your button will stretch to fill your page's entire horizontal space and any elements following it will flow onto a "new line" below the block. `<button class="btn btn-default btn-block">Submit</button>` This button would take up 100% of the available width. Note that these buttons still need the `btn` class. Add Bootstrap's `btn-block` class to your Bootstrap button.

Instructions

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, Monospace;
  }

  p {
    font-size: 16px;
    font-family: Monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }

  .smaller-image {
    width: 100px;
  }
</style>

```

```

<div class="container-fluid">
  <h2 class="red-text text-center">CatPhotoApp</h2>

  <p>Click here for <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  
  <button class="btn btn-default">Like</button>
  <p>Things cats love:</p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
  <form action="/submit-cat-photo">
    <label><input type="radio" name="indoor-outdoor"> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
    <label><input type="checkbox" name="personality"> Loving</label>
    <label><input type="checkbox" name="personality"> Lazy</label>
    <label><input type="checkbox" name="personality"> Crazy</label>
    <input type="text" placeholder="cat photo URL" required>
    <button type="submit">Submit</button>
  </form>
</div>

```

Solution

```
// solution required
```

6. Taste the Bootstrap Button Color Rainbow

Description

The `btn-primary` class is the main color you'll use in your app. It is useful for highlighting actions you want your user to take. Replace Bootstrap's `btn-default` class by `btn-primary` in your button. Note that this button will still need the `btn` and `btn-block` classes.

Instructions

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, Monospace;
  }

  p {
    font-size: 16px;
    font-family: Monospace;
  }

```

```

.thick-green-border {
  border-color: green;
  border-width: 10px;
  border-style: solid;
  border-radius: 50%;
}

.smaller-image {
  width: 100px;
}

```

</style>

```

<div class="container-fluid">
  <h2 class="red-text text-center">CatPhotoApp</h2>

  <p>Click here for <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  
  <button class="btn btn-default btn-block">Like</button>
  <p>Things cats love:</p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
  <form action="/submit-cat-photo">
    <label><input type="radio" name="indoor-outdoor"> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
    <label><input type="checkbox" name="personality"> Loving</label>
    <label><input type="checkbox" name="personality"> Lazy</label>
    <label><input type="checkbox" name="personality"> Crazy</label>
    <input type="text" placeholder="cat photo URL" required>
    <button type="submit">Submit</button>
  </form>
</div>

```

Solution

```
// solution required
```

7. Call out Optional Actions with btn-info

Description

Bootstrap comes with several pre-defined colors for buttons. The `btn-info` class is used to call attention to optional actions that the user can take. Create a new block-level Bootstrap button below your "Like" button with the text "Info", and add Bootstrap's `btn-info` and `btn-block` classes to it. Note that these buttons still need the `btn` and `btn-block` classes.

Instructions

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, Monospace;
  }

  p {
    font-size: 16px;
    font-family: Monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }

  .smaller-image {
    width: 100px;
  }
</style>

<div class="container-fluid">
  <h2 class="red-text text-center">CatPhotoApp</h2>

  <p>Click here for <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  
  <button class="btn btn-block btn-primary">Like</button>
  <p>Things cats love:</p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
  <form action="/submit-cat-photo">
    <label><input type="radio" name="indoor-outdoor"> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
    <label><input type="checkbox" name="personality"> Loving</label>
    <label><input type="checkbox" name="personality"> Lazy</label>
    <label><input type="checkbox" name="personality"> Crazy</label>
    <input type="text" placeholder="cat photo URL" required>
    <button type="submit">Submit</button>
  </form>
</div>

```

Solution

```
// solution required
```

8. Warn Your Users of a Dangerous Action with `btn-danger`

Description

Bootstrap comes with several pre-defined colors for buttons. The `btn-danger` class is the button color you'll use to notify users that the button performs a destructive action, such as deleting a cat photo. Create a button with the text "Delete" and give it the class `btn-danger`. Note that these buttons still need the `btn` and `btn-block` classes.

Instructions

Challenge Seed

```
<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, Monospace;
  }

  p {
    font-size: 16px;
    font-family: Monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }

  .smaller-image {
    width: 100px;
  }
</style>

<div class="container-fluid">
  <h2 class="red-text text-center">CatPhotoApp</h2>

  <p>Click here for <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  
  <button class="btn btn-block btn-primary">Like</button>
  <button class="btn btn-block btn-info">Info</button>
  <p>Things cats love:</p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
  <form action="/submit-cat-photo">
    <label><input type="radio" name="indoor-outdoor"> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
    <label><input type="checkbox" name="personality"> Loving</label>
    <label><input type="checkbox" name="personality"> Lazy</label>
    <label><input type="checkbox" name="personality"> Crazy</label>
    <input type="text" placeholder="cat photo URL" required>
    <button type="submit">Submit</button>
  </form>
</div>
```

Solution

```
// solution required
```

9. Use the Bootstrap Grid to Put Elements Side By Side

Description

Bootstrap uses a responsive 12-column grid system, which makes it easy to put elements into rows and specify each element's relative width. Most of Bootstrap's classes can be applied to a `div` element. Bootstrap has different column width attributes that it uses depending on how wide the user's screen is. For example, phones have narrow screens, and laptops have wider screens. Take for example Bootstrap's `col-md-*` class. Here, `md` means medium, and `*` is a number specifying how many columns wide the element should be. In this case, the column width of an element on a medium-sized screen, such as a laptop, is being specified. In the Cat Photo App that we're building, we'll use `col-xs-*`, where `xs` means extra small (like an extra-small mobile phone screen), and `*` is the number of columns specifying how many columns wide the element should be. Put the `Like`, `Info` and `Delete` buttons side-by-side by nesting all three of them within one `<div class="row">` element, then each of them within a `<div class="col-xs-4">` element. The `row` class is applied to a `div`, and the buttons themselves can be nested within it.

Instructions

Challenge Seed

```
<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, Monospace;
  }

  p {
    font-size: 16px;
    font-family: Monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }

  .smaller-image {
    width: 100px;
  }
</style>

<div class="container-fluid">
  <h2 class="red-text text-center">CatPhotoApp</h2>

  <p>Click here for <a href="#">cat photos</a>.</p>

  <a href="#"></a>

  
  <button class="btn btn-block btn-primary">Like</button>
  <button class="btn btn-block btn-info">Info</button>
  <button class="btn btn-block btn-danger">Delete</button>
  <p>Things cats love:</p>
```

```

<ul>
  <li>cat nip</li>
  <li>laser pointers</li>
  <li>lasagna</li>
</ul>
<p>Top 3 things cats hate:</p>
<ol>
  <li>flea treatment</li>
  <li>thunder</li>
  <li>other cats</li>
</ol>
<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor"> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
  <label><input type="checkbox" name="personality"> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Crazy</label>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</div>

```

Solution

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, Monospace;
  }

  p {
    font-size: 16px;
    font-family: Monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }

  .smaller-image {
    width: 100px;
  }
</style>

<div class="container-fluid">
  <h2 class="red-text text-center">CatPhotoApp</h2>

  <p>Click here for <a href="#">cat photos</a>.</p>
  <a href="#"></a>

  
  <div class="row">
    <div class="col-xs-4">
      <button class="btn btn-block btn-primary">Like</button>
    </div>
    <div class="col-xs-4">
      <button class="btn btn-block btn-info">Info</button>
    </div>
    <div class="col-xs-4">
      <button class="btn btn-block btn-danger">Delete</button>
    </div>
  </div>
</div>

```

<p>Things cats love:</p>

```

<ul>
  <li>cat nip</li>
  <li>laser pointers</li>
  <li>lasagna</li>
</ul>
<p>Top 3 things cats hate:</p>
<ol>
  <li>flea treatment</li>
  <li>thunder</li>
  <li>other cats</li>
</ol>
<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor"> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
  <label><input type="checkbox" name="personality"> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Crazy</label>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</div>

```

10. Ditch Custom CSS for Bootstrap

Description

We can clean up our code and make our Cat Photo App look more conventional by using Bootstrap's built-in styles instead of the custom styles we created earlier. Don't worry - there will be plenty of time to customize our CSS later. Delete the `.red-text`, `p`, and `.smaller-image` CSS declarations from your `style` element so that the only declarations left in your `style` element are `h2` and `thick-green-border`. Then delete the `p` element that contains a dead link. Then remove the `red-text` class from your `h2` element and replace it with the `text-primary` Bootstrap class. Finally, remove the "smaller-image" class from your first `img` element and replace it with the `img-responsive` class.

Instructions

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  .red-text {
    color: red;
  }

  h2 {
    font-family: Lobster, Monospace;
  }

  p {
    font-size: 16px;
    font-family: Monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }

  .smaller-image {
    width: 100px;
  }
</style>

<div class="container-fluid">
  <h2 class="red-text text-center">CatPhotoApp</h2>

```

```

<p>Click here for <a href="#">cat photos</a>.</p>

<a href="#"></a>


<div class="row">
  <div class="col-xs-4">
    <button class="btn btn-block btn-primary">Like</button>
  </div>
  <div class="col-xs-4">
    <button class="btn btn-block btn-info">Info</button>
  </div>
  <div class="col-xs-4">
    <button class="btn btn-block btn-danger">Delete</button>
  </div>
</div>
<p>Things cats love:</p>
<ul>
  <li>cat nip</li>
  <li>laser pointers</li>
  <li>lasagna</li>
</ul>
<p>Top 3 things cats hate:</p>
<ol>
  <li>flea treatment</li>
  <li>thunder</li>
  <li>other cats</li>
</ol>
<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor"> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
  <label><input type="checkbox" name="personality"> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Crazy</label>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</div>

```

Solution

```
// solution required
```

11. Use a span to Target Inline Elements

Description

You can use spans to create inline elements. Remember when we used the `btn-block` class to make the button fill the entire row? normal button `btn-block` button That illustrates the difference between an "inline" element and a "block" element. By using the inline `span` element, you can put several elements on the same line, and even style different parts of the same line differently. Nest the word "love" in your "Things cats love" element below within a `span` element. Then give that `span` the class `text-danger` to make the text red. Here's how you would do this with the "Top 3 things cats hate" element: `<p>Top 3 things cats hate:</p>`

Instructions

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
```

```

h2 {
  font-family: Lobster, Monospace;
}

.thick-green-border {
  border-color: green;
  border-width: 10px;
  border-style: solid;
  border-radius: 50%;
}


```

</style>

```

<div class="container-fluid">
  <h2 class="text-primary text-center">CatPhotoApp</h2>

  <a href="#"></a>

  
  <div class="row">
    <div class="col-xs-4">
      <button class="btn btn-block btn-primary">Like</button>
    </div>
    <div class="col-xs-4">
      <button class="btn btn-block btn-info">Info</button>
    </div>
    <div class="col-xs-4">
      <button class="btn btn-block btn-danger">Delete</button>
    </div>
  </div>
  <p>Things cats love:</p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
  <form action="/submit-cat-photo">
    <label><input type="radio" name="indoor-outdoor"> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
    <label><input type="checkbox" name="personality"> Loving</label>
    <label><input type="checkbox" name="personality"> Lazy</label>
    <label><input type="checkbox" name="personality"> Crazy</label>
    <input type="text" placeholder="cat photo URL" required>
    <button type="submit">Submit</button>
  </form>
</div>

```

Solution

```
// solution required
```

12. Create a Custom Heading

Description

We will make a simple heading for our Cat Photo App by putting the title and relaxing cat image in the same row. Remember, Bootstrap uses a responsive grid system, which makes it easy to put elements into rows and specify each element's relative width. Most of Bootstrap's classes can be applied to a `div` element. Nest your first image and your `h2` element within a single `<div class="row">` element. Nest your `h2` element within a `<div class="col-xs-8">`

and your image in a `<div class="col-xs-4">` so that they are on the same line. Notice how the image is now just the right size to fit along the text?

Instructions

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">

<style>
  h2 {
    font-family: Lobster, Monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }
</style>

<div class="container-fluid">
  <h2 class="text-primary text-center">CatPhotoApp</h2>

  <a href="#"></a>

  
  <div class="row">
    <div class="col-xs-4">
      <button class="btn btn-block btn-primary">Like</button>
    </div>
    <div class="col-xs-4">
      <button class="btn btn-block btn-info">Info</button>
    </div>
    <div class="col-xs-4">
      <button class="btn btn-block btn-danger">Delete</button>
    </div>
  </div>
  <p>Things cats <span class="text-danger">love:</span></p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
  <form action="/submit-cat-photo">
    <label><input type="radio" name="indoor-outdoor"> Indoor</label>
    <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
    <label><input type="checkbox" name="personality"> Loving</label>
    <label><input type="checkbox" name="personality"> Lazy</label>
    <label><input type="checkbox" name="personality"> Crazy</label>
    <input type="text" placeholder="cat photo URL" required>
    <button type="submit">Submit</button>
  </form>
</div>

```

Solution

```
// solution required
```

13. Add Font Awesome Icons to our Buttons

Description

Font Awesome is a convenient library of icons. These icons are vector graphics, stored in the `.svg` file format. These icons are treated just like fonts. You can specify their size using pixels, and they will assume the font size of their parent HTML elements. You can include Font Awesome in any app by adding the following code to the top of your HTML: `<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-awesome.min.css" integrity="sha384-XdYbMnZ/QjLh6iI4ogqCTaIjrFk87ip+ekIjeZch0Y+PvJ8CDYtEs1ipDmPorQ+" crossorigin="anonymous">` In this case, we've already added it for you to this page behind the scenes. The `i` element was originally used to make other elements italic, but is now commonly used for icons. You can add the Font Awesome classes to the `i` element to turn it into an icon, for example: `<i class="fa fa-info-circle"></i>` Note that the `span` element is also acceptable for use with icons. Use Font Awesome to add a thumbs-up icon to your like button by giving it an `i` element with the classes `fa` and `fa-thumbs-up`; make sure to keep the text "Like" next to the icon.

Instructions

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  h2 {
    font-family: Lobster, Monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }
</style>

<div class="container-fluid">
  <div class="row">
    <div class="col-xs-8">
      <h2 class="text-primary text-center">CatPhotoApp</h2>
    </div>
    <div class="col-xs-4">
      <a href="#"></a>
    </div>
    
  <div class="row">
    <div class="col-xs-4">
      <button class="btn btn-block btn-primary">Like</button>
    </div>
    <div class="col-xs-4">
      <button class="btn btn-block btn-info">Info</button>
    </div>
    <div class="col-xs-4">
      <button class="btn btn-block btn-danger">Delete</button>
    </div>
  </div>
  <p>Things cats <span class="text-danger">love:</span></p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
</div>

```

```

</ol>
<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor"> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
  <label><input type="checkbox" name="personality"> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Crazy</label>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</div>

```

Solution

```
// solution required
```

14. Add Font Awesome Icons to all of our Buttons

Description

Font Awesome is a convenient library of icons. These icons are vector graphics, stored in the `.svg` file format. These icons are treated just like fonts. You can specify their size using pixels, and they will assume the font size of their parent HTML elements. Use Font Awesome to add an `info-circle` icon to your info button and a `trash` icon to your delete button. Note: The `span` element is an acceptable alternative to the `i` element for the directions below.

Instructions

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  h2 {
    font-family: Lobster, Monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }
</style>

<div class="container-fluid">
  <div class="row">
    <div class="col-xs-8">
      <h2 class="text-primary text-center">CatPhotoApp</h2>
    </div>
    <div class="col-xs-4">
      <a href="#"></a>
    </div>
    <div src="https://bit.ly/fcc-running-cats" class="img-responsive" alt="Three kittens running towards the camera.">
      <div class="row">
        <div class="col-xs-4">
          <button class="btn btn-block btn-primary"><i class="fa fa-thumbs-up"></i> Like</button>
        </div>
        <div class="col-xs-4">
          <button class="btn btn-block btn-info">Info</button>
        </div>
        <div class="col-xs-4">
          <button class="btn btn-block btn-danger">Delete</button>
        </div>
      </div>
    </div>
  </div>
</div>

```

```

</div>
<p>Things cats <span class="text-danger">love:</span></p>
<ul>
  <li>cat nip</li>
  <li>laser pointers</li>
  <li>lasagna</li>
</ul>
<p>Top 3 things cats hate:</p>
<ol>
  <li>flea treatment</li>
  <li>thunder</li>
  <li>other cats</li>
</ol>
<form action="/submit-cat-photo">
  <label><input type="radio" name="indoor-outdoor"> Indoor</label>
  <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
  <label><input type="checkbox" name="personality"> Loving</label>
  <label><input type="checkbox" name="personality"> Lazy</label>
  <label><input type="checkbox" name="personality"> Crazy</label>
  <input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</div>

```

Solution

```
// solution required
```

15. Responsively Style Radio Buttons

Description

You can use Bootstrap's `col-xs-*` classes on `form` elements, too! This way, our radio buttons will be evenly spread out across the page, regardless of how wide the screen resolution is. Nest both your radio buttons within a `<div class="row">` element. Then nest each of them within a `<div class="col-xs-6">` element. **Note:** As a reminder, radio buttons are `input` elements of type `radio`.

Instructions

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  h2 {
    font-family: Lobster, Monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }
</style>

<div class="container-fluid">
  <div class="row">
    <div class="col-xs-8">
      <h2 class="text-primary text-center">CatPhotoApp</h2>
    </div>
    <div class="col-xs-4">
      <a href="#"></a>
    </div>
  </div>

```

```


<button class="btn btn-block btn-primary"><i class="fa fa-thumbs-up"></i> Like</button>



<button class="btn btn-block btn-info"><i class="fa fa-info-circle"></i> Info</button>



<button class="btn btn-block btn-danger"><i class="fa fa-trash"></i> Delete</button>



Things cats <span class="text-danger">love:</span></p>



- cat nip
- laser pointers
- lasagna



Top 3 things cats hate:



- flea treatment
- thunder
- other cats


<form action="/submit-cat-photo">
<label><input type="radio" name="indoor-outdoor"> Indoor</label>
<label><input type="radio" name="indoor-outdoor"> Outdoor</label>
<label><input type="checkbox" name="personality"> Loving</label>
<label><input type="checkbox" name="personality"> Lazy</label>
<label><input type="checkbox" name="personality"> Crazy</label>
<input type="text" placeholder="cat photo URL" required>
<button type="submit">Submit</button>
</form>

```

Solution

```
// solution required
```

16. Responsively Style Checkboxes

Description

Since Bootstrap's `col-xs-*` classes are applicable to all `form` elements, you can use them on your checkboxes too! This way, the checkboxes will be evenly spread out across the page, regardless of how wide the screen resolution is.

Instructions

Nest all three of your checkboxes in a `<div class="row">` element. Then nest each of them in a `<div class="col-xs-4">` element.

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  h2 {
    font-family: Lobster, Monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }

```

```

        }

    </style>

<div class="container-fluid">
  <div class="row">
    <div class="col-xs-8">
      <h2 class="text-primary text-center">CatPhotoApp</h2>
    </div>
    <div class="col-xs-4">
      <a href="#"></a>
    </div>
  </div>
  
  <div class="row">
    <div class="col-xs-4">
      <button class="btn btn-block btn-primary"><i class="fa fa-thumbs-up"></i> Like</button>
    </div>
    <div class="col-xs-4">
      <button class="btn btn-block btn-info"><i class="fa fa-info-circle"></i> Info</button>
    </div>
    <div class="col-xs-4">
      <button class="btn btn-block btn-danger"><i class="fa fa-trash"></i> Delete</button>
    </div>
  </div>
  <p>Things cats love:<span class="text-danger">love:</span></p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
  <form action="/submit-cat-photo">
    <div class="row">
      <div class="col-xs-6">
        <label><input type="radio" name="indoor-outdoor"> Indoor</label>
      </div>
      <div class="col-xs-6">
        <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
      </div>
    </div>
    <label><input type="checkbox" name="personality"> Loving</label>
    <label><input type="checkbox" name="personality"> Lazy</label>
    <label><input type="checkbox" name="personality"> Crazy</label>
    <input type="text" placeholder="cat photo URL" required>
    <button type="submit">Submit</button>
  </form>
</div>

```

Solution

```
// solution required
```

17. Style Text Inputs as Form Controls

Description

You can add the `fa-paper-plane` Font Awesome icon by adding `<i class="fa fa-paper-plane"></i>` within your `submit` button element. Give your form's text input field a class of `form-control`. Give your form's submit button the classes `btn` `btn-primary`. Also give this button the Font Awesome icon of `fa-paper-plane`. All textual `<input>`, `<textarea>`, and `<select>` elements with the class `.form-control` have a width of 100%.

Instructions

Challenge Seed

```
<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  h2 {
    font-family: Lobster, Monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }

</style>

<div class="container-fluid">
  <div class="row">
    <div class="col-xs-8">
      <h2 class="text-primary text-center">CatPhotoApp</h2>
    </div>
    <div class="col-xs-4">
      <a href="#"></a>
    </div>
    
  <div class="row">
    <div class="col-xs-4">
      <button class="btn btn-block btn-primary"><i class="fa fa-thumbs-up"></i> Like</button>
    </div>
    <div class="col-xs-4">
      <button class="btn btn-block btn-info"><i class="fa fa-info-circle"></i> Info</button>
    </div>
    <div class="col-xs-4">
      <button class="btn btn-block btn-danger"><i class="fa fa-trash"></i> Delete</button>
    </div>
  </div>
  <p>Things cats love:<span class="text-danger">love:</span></p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>
  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>
  <form action="/submit-cat-photo">
    <div class="row">
      <div class="col-xs-6">
        <label><input type="radio" name="indoor-outdoor"> Indoor</label>
      </div>
      <div class="col-xs-6">
        <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
      </div>
    </div>
    <div class="row">
      <div class="col-xs-4">
        <label><input type="checkbox" name="personality"> Loving</label>
      </div>
      <div class="col-xs-4">
        <label><input type="checkbox" name="personality"> Lazy</label>
      </div>
      <div class="col-xs-4">
        <label><input type="checkbox" name="personality"> Crazy</label>
      </div>
    </div>
  </form>
</div>
```

```

<input type="text" placeholder="cat photo URL" required>
  <button type="submit">Submit</button>
</form>
</div>

```

Solution

```
// solution required
```

18. Line up Form Elements Responsively with Bootstrap

Description

Now let's get your form `input` and your submission `button` on the same line. We'll do this the same way we have previously: by using a `div` element with the class `row`, and other `div` elements within it using the `col-xs-*` class. Nest both your form's text `input` and submit `button` within a `div` with the class `row`. Nest your form's text `input` within a `div` with the class of `col-xs-7`. Nest your form's submit `button` in a `div` with the class `col-xs-5`. This is the last challenge we'll do for our Cat Photo App for now. We hope you've enjoyed learning Font Awesome, Bootstrap, and responsive design!

Instructions

Challenge Seed

```

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">
<style>
  h2 {
    font-family: Lobster, Monospace;
  }

  .thick-green-border {
    border-color: green;
    border-width: 10px;
    border-style: solid;
    border-radius: 50%;
  }

</style>

<div class="container-fluid">
  <div class="row">
    <div class="col-xs-8">
      <h2 class="text-primary text-center">CatPhotoApp</h2>
    </div>
    <div class="col-xs-4">
      <a href="#"></a>
    </div>
  </div>
  
<div class="row">
  <div class="col-xs-4">
    <button class="btn btn-block btn-primary"><i class="fa fa-thumbs-up"></i> Like</button>
  </div>
  <div class="col-xs-4">
    <button class="btn btn-block btn-info"><i class="fa fa-info-circle"></i> Info</button>
  </div>
  <div class="col-xs-4">
    <button class="btn btn-block btn-danger"><i class="fa fa-trash"></i> Delete</button>
  </div>
</div>
<p>Things cats <span class="text-danger">love:</span></p>
<ul>

```

```

<li>cat nip</li>
<li>laser pointers</li>
<li>lasagna</li>
</ul>
<p>Top 3 things cats hate:</p>
<ol>
  <li>flea treatment</li>
  <li>thunder</li>
  <li>other cats</li>
</ol>
<form action="/submit-cat-photo">
  <div class="row">
    <div class="col-xs-6">
      <label><input type="radio" name="indoor-outdoor"> Indoor</label>
    </div>
    <div class="col-xs-6">
      <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
    </div>
  </div>
  <div class="row">
    <div class="col-xs-4">
      <label><input type="checkbox" name="personality"> Loving</label>
    </div>
    <div class="col-xs-4">
      <label><input type="checkbox" name="personality"> Lazy</label>
    </div>
    <div class="col-xs-4">
      <label><input type="checkbox" name="personality"> Crazy</label>
    </div>
  </div>
  <input type="text" class="form-control" placeholder="cat photo URL" required>
  <button type="submit" class="btn btn-primary"><i class="fa fa-paper-plane"></i> Submit</button>
</form>
</div>

```

Solution

```
// solution required
```

19. Create a Bootstrap Headline

Description

Now let's build something from scratch to practice our HTML, CSS and Bootstrap skills. We'll build a jQuery playground, which we'll soon put to use in our jQuery challenges. To start with, create an `h3` element, with the text `jQuery Playground`. Color your `h3` element with the `text-primary` Bootstrap class, and center it with the `text-center` Bootstrap class.

Instructions

Challenge Seed

Solution

```
<h3 class="text-primary text-center">jQuery Playground</h3>
```

20. House our page within a Bootstrap container-fluid div

Description

Now let's make sure all the content on your page is mobile-responsive. Let's nest your `h3` element within a `div` element with the class `container-fluid`.

Instructions

Challenge Seed

```
<h3 class="text-primary text-center">jQuery Playground</h3>
```

Solution

```
// solution required
```

21. Create a Bootstrap Row

Description

Now we'll create a Bootstrap row for our inline elements. Create a `div` element below the `h3` tag, with a class of `row`.

Instructions

Challenge Seed

```
<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
</div>
```

Solution

```
// solution required
```

22. Split Your Bootstrap Row

Description

Now that we have a Bootstrap Row, let's split it into two columns to house our elements. Create two `div` elements within your row, both with the class `col-xs-6`.

Instructions

Challenge Seed

```
<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">

    </div>
  </div>
```

Solution

```
// solution required
```

23. Create Bootstrap Wells

Description

Bootstrap has a class called `well` that can create a visual sense of depth for your columns. Nest one `div` element with the class `well` within each of your `col-xs-6` `div` elements.

Instructions

Challenge Seed

```
<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">

      </div>
      <div class="col-xs-6">

        </div>
      </div>
    </div>
  </div>
```

Solution

```
// solution required
```

24. Add Elements within Your Bootstrap Wells

Description

Now we're several `div` elements deep on each column of our row. This is as deep as we'll need to go. Now we can add our button elements. Nest three button elements within each of your `well` `div` elements.

Instructions

Challenge Seed

```
<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
      <div class="well">

        </div>
      </div>
    <div class="col-xs-6">
      <div class="well">

        </div>
      </div>
    </div>
  </div>
```

Solution

```
// solution required
```

25. Apply the Default Bootstrap Button Style

Description

Bootstrap has another button class called `btn-default`. Apply both the `btn` and `btn-default` classes to each of your button elements.

Instructions

Challenge Seed

```
<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
      <div class="well">
        <button></button>
        <button></button>
        <button></button>
      </div>
    </div>
    <div class="col-xs-6">
      <div class="well">
        <button></button>
        <button></button>
        <button></button>
      </div>
    </div>
  </div>
</div>
```

Solution

```
// solution required
```

26. Create a Class to Target with jQuery Selectors

Description

Not every class needs to have corresponding CSS. Sometimes we create classes just for the purpose of selecting these elements more easily using jQuery. Give each of your `button` elements the class `target`.

Instructions

Challenge Seed

```
<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
      <div class="well">
        <button class="btn btn-default"></button>
        <button class="btn btn-default"></button>
        <button class="btn btn-default"></button>
      </div>
    </div>
    <div class="col-xs-6">
      <div class="well">
        <button class="btn btn-default"></button>
        <button class="btn btn-default"></button>
        <button class="btn btn-default"></button>
      </div>
    </div>
  </div>
</div>
```

Solution

```
// solution required
```

27. Add id Attributes to Bootstrap Elements

Description

Recall that in addition to class attributes, you can give each of your elements an `id` attribute. Each `id` must be unique to a specific element and used only once per page. Let's give a unique `id` to each of our `div` elements of class `well`. Remember that you can give an element an `id` like this: `<div class="well" id="center-well">` Give the well on the left the `id` of `left-well`. Give the well on the right the `id` of `right-well`.

Instructions

Challenge Seed

```
<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
      <div class="well">
        <button class="btn btn-default target"></button>
        <button class="btn btn-default target"></button>
        <button class="btn btn-default target"></button>
      </div>
    </div>
  </div>
</div>
```

```
<div class="col-xs-6">
  <div class="well">
    <button class="btn btn-default target"></button>
    <button class="btn btn-default target"></button>
    <button class="btn btn-default target"></button>
  </div>
</div>
</div>
</div>
```

Solution

```
// solution required
```

28. Label Bootstrap Wells

Description

For the sake of clarity, let's label both of our wells with their ids. Above your left-well, inside its `col-xs-6` `div` element, add a `h4` element with the text `#left-well`. Above your right-well, inside its `col-xs-6` `div` element, add a `h4` element with the text `#right-well`.

Instructions

Challenge Seed

```
<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">

      <div class="well" id="left-well">
        <button class="btn btn-default target"></button>
        <button class="btn btn-default target"></button>
        <button class="btn btn-default target"></button>
      </div>
    </div>
    <div class="col-xs-6">

      <div class="well" id="right-well">
        <button class="btn btn-default target"></button>
        <button class="btn btn-default target"></button>
        <button class="btn btn-default target"></button>
      </div>
    </div>
  </div>
</div>
```

Solution

```
// solution required
```

29. Give Each Element a Unique id

Description

We will also want to be able to use jQuery to target each button by its unique id. Give each of your buttons a unique id, starting with `target1` and ending with `target6`. Make sure that `target1` to `target3` are in `#left-well`, and `target4` to `target6` are in `#right-well`.

Instructions

Challenge Seed

```
<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
      <h4>#left-well</h4>
      <div class="well" id="left-well">
        <button class="btn btn-default target"></button>
        <button class="btn btn-default target"></button>
        <button class="btn btn-default target"></button>
      </div>
    </div>
    <div class="col-xs-6">
      <h4>#right-well</h4>
      <div class="well" id="right-well">
        <button class="btn btn-default target" id="target1"></button>
        <button class="btn btn-default target" id="target2"></button>
        <button class="btn btn-default target" id="target3"></button>
      </div>
    </div>
  </div>
</div>
```

Solution

```
// solution required
```

30. Label Bootstrap Buttons

Description

Just like we labeled our wells, we want to label our buttons. Give each of your `button` elements text that corresponds to its `id`'s selector.

Instructions

Challenge Seed

```
<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
      <h4>#left-well</h4>
      <div class="well" id="left-well">
        <button class="btn btn-default target" id="target1">target1</button>
        <button class="btn btn-default target" id="target2">target2</button>
        <button class="btn btn-default target" id="target3">target3</button>
      </div>
    </div>
    <div class="col-xs-6">
      <h4>#right-well</h4>
      <div class="well" id="right-well">
        <button class="btn btn-default target" id="target4">target4</button>
        <button class="btn btn-default target" id="target5">target5</button>
      </div>
    </div>
  </div>
</div>
```

```

        <button class="btn btn-default target" id="target6">/</button>
    </div>
</div>
</div>
</div>

```

Solution

```
// solution required
```

31. Use Comments to Clarify Code

Description

When we start using jQuery, we will modify HTML elements without needing to actually change them in HTML. Let's make sure that everyone knows they shouldn't actually modify any of this code directly. Remember that you can start a comment with `<!--` and end a comment with `-->`. Add a comment at the top of your HTML that says `Only change code above this line.`

Instructions

Challenge Seed

```


<h3 class="text-primary text-center">jQuery Playground</h3>
    <div class="row">
        <div class="col-xs-6">
            <h4>#left-well</h4>
            <div class="well" id="left-well">
                <button class="btn btn-default target" id="target1">#target1</button>
                <button class="btn btn-default target" id="target2">#target2</button>
                <button class="btn btn-default target" id="target3">#target3</button>
            </div>
        </div>
        <div class="col-xs-6">
            <h4>#right-well</h4>
            <div class="well" id="right-well">
                <button class="btn btn-default target" id="target4">#target4</button>
                <button class="btn btn-default target" id="target5">#target5</button>
                <button class="btn btn-default target" id="target6">#target6</button>
            </div>
        </div>
    </div>


```

Solution

```
// solution required
```

jQuery

1. Learn How Script Tags and Document Ready Work

Description

Now we're ready to learn jQuery, the most popular JavaScript tool of all time. Before we can start using jQuery, we need to add some things to our HTML. First, add a `script` element at the top of your page. Be sure to close it on the following line. Your browser will run any JavaScript inside a `script` element, including jQuery. Inside your `script` element, add this code: `$(document).ready(function() {` to your `script`. Then close it on the following line (still inside your `script` element) with: `});` We'll learn more about `functions` later. The important thing to know is that code you put inside this `function` will run as soon as your browser has loaded your page. This is important because without your `document ready` function, your code may run before your HTML is rendered, which would cause bugs.

Instructions

Challenge Seed

```
<!-- Only change code above this line. -->

<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
      <h4>#left-well</h4>
      <div class="well" id="left-well">
        <button class="btn btn-default target" id="target1">#target1</button>
        <button class="btn btn-default target" id="target2">#target2</button>
        <button class="btn btn-default target" id="target3">#target3</button>
      </div>
    </div>
    <div class="col-xs-6">
      <h4>#right-well</h4>
      <div class="well" id="right-well">
        <button class="btn btn-default target" id="target4">#target4</button>
        <button class="btn btn-default target" id="target5">#target5</button>
        <button class="btn btn-default target" id="target6">#target6</button>
      </div>
    </div>
  </div>
</div>
```

Solution

```
// solution required
```

2. Target HTML Elements with Selectors Using jQuery

Description

Now we have a `document ready` function. Now let's write our first jQuery statement. All jQuery functions start with a `$`, usually referred to as a dollar sign operator, or as `bling`. jQuery often selects an HTML element with a `selector`, then does something to that element. For example, let's make all of your `button` elements bounce. Just add this code inside your `document ready` function: `$("button").addClass("animated bounce");` Note that we've already included both the `jQuery` library and the `Animate.css` library in the background so that you can use them in the editor. So you are using `jQuery` to apply the `Animate.css` `bounce` class to your `button` elements.

Instructions

Challenge Seed

```
<script>
  $(document).ready(function() {
    });
  
```

```
</script>

<!-- Only change code above this line. -->

<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
      <h4>#left-well</h4>
      <div class="well" id="left-well">
        <button class="btn btn-default target" id="target1">#target1</button>
        <button class="btn btn-default target" id="target2">#target2</button>
        <button class="btn btn-default target" id="target3">#target3</button>
      </div>
    </div>
    <div class="col-xs-6">
      <h4>#right-well</h4>
      <div class="well" id="right-well">
        <button class="btn btn-default target" id="target4">#target4</button>
        <button class="btn btn-default target" id="target5">#target5</button>
        <button class="btn btn-default target" id="target6">#target6</button>
      </div>
    </div>
  </div>
</div>
```

Solution

```
// solution required
```

3. Target Elements by Class Using jQuery

Description

You see how we made all of your `button` elements bounce? We selected them with `$("button")`, then we added some CSS classes to them with `.addClass("animated bounce");`. You just used jQuery's `.addClass()` function, which allows you to add classes to elements. First, let's target your `div` elements with the class `well` by using the `$(".well")` selector. Note that, just like with CSS declarations, you type a `.` before the class's name. Then use jQuery's `.addClass()` function to add the classes `animated` and `shake`. For example, you could make all the elements with the class `text-primary` shake by adding the following to your document ready function: `$(".text-primary").addClass("animated shake");`

Instructions

Challenge Seed

```
<script>
$(document).ready(function() {
  $("button").addClass("animated bounce");
});
</script>

<!-- Only change code above this line. -->

<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
      <h4>#left-well</h4>
      <div class="well" id="left-well">
        <button class="btn btn-default target" id="target1">#target1</button>
        <button class="btn btn-default target" id="target2">#target2</button>
        <button class="btn btn-default target" id="target3">#target3</button>
      </div>
    </div>
  </div>
</div>
```

```

</div>
<div class="col-xs-6">
  <h4>#right-well</h4>
  <div class="well" id="right-well">
    <button class="btn btn-default target" id="target4">#target4</button>
    <button class="btn btn-default target" id="target5">#target5</button>
    <button class="btn btn-default target" id="target6">#target6</button>
  </div>
</div>
</div>
</div>

```

Solution

```
// solution required
```

4. Target Elements by id Using jQuery

Description

You can also target elements by their id attributes. First target your `button` element with the id `target3` by using the `$("#target3")` selector. Note that, just like with CSS declarations, you type a `#` before the id's name. Then use jQuery's `.addClass()` function to add the classes `animated` and `fadeOut`. Here's how you'd make the `button` element with the id `target6` fade out: `($("#target6").addClass("animated fadeOut"))`.

Instructions

Challenge Seed

```

<script>
$(document).ready(function() {
  $("button").addClass("animated bounce");
  $(".well").addClass("animated shake");

});
</script>

<!-- Only change code above this line. -->

<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
      <h4>#left-well</h4>
      <div class="well" id="left-well">
        <button class="btn btn-default target" id="target1">#target1</button>
        <button class="btn btn-default target" id="target2">#target2</button>
        <button class="btn btn-default target" id="target3">#target3</button>
      </div>
    </div>
    <div class="col-xs-6">
      <h4>#right-well</h4>
      <div class="well" id="right-well">
        <button class="btn btn-default target" id="target4">#target4</button>
        <button class="btn btn-default target" id="target5">#target5</button>
        <button class="btn btn-default target" id="target6">#target6</button>
      </div>
    </div>
  </div>
</div>

```

Solution

```
// solution required
```

5. Delete Your jQuery Functions

Description

These animations were cool at first, but now they're getting kind of distracting. Delete all three of these jQuery functions from your document ready function , but leave your document ready function itself intact.

Instructions

Challenge Seed

```
<script>
$(document).ready(function() {
    $("button").addClass("animated bounce");
    $(".well").addClass("animated shake");
    $("#target3").addClass("animated fadeIn");

});
</script>

<!-- Only change code above this line. -->

<div class="container-fluid">
    <h3 class="text-primary text-center">jQuery Playground</h3>
    <div class="row">
        <div class="col-xs-6">
            <h4>#left-well</h4>
            <div class="well" id="left-well">
                <button class="btn btn-default target" id="target1">#target1</button>
                <button class="btn btn-default target" id="target2">#target2</button>
                <button class="btn btn-default target" id="target3">#target3</button>
            </div>
        </div>
        <div class="col-xs-6">
            <h4>#right-well</h4>
            <div class="well" id="right-well">
                <button class="btn btn-default target" id="target4">#target4</button>
                <button class="btn btn-default target" id="target5">#target5</button>
                <button class="btn btn-default target" id="target6">#target6</button>
            </div>
        </div>
    </div>
</div>
```

Solution

```
// solution required
```

6. Target the Same Element with Multiple jQuery Selectors

Description

Now you know three ways of targeting elements: by type: `$(".button")` , by class: `$(".btn")` , and by id `$("#target1")` . Although it is possible to add multiple classes in a single `.addClass()` call, let's add them to the

same element in *three separate ways*. Using `.addClass()`, add only one class at a time to the same element, three different ways: Add the `animated` class to all elements with type `button`. Add the `shake` class to all the buttons with class `.btn`. Add the `btn-primary` class to the button with id `#target1`. **Note**
You should only be targeting one element and adding only one class at a time. Altogether, your three individual selectors will end up adding the three classes `shake`, `animated`, and `btn-primary` to `#target1`.

Instructions

Challenge Seed

```
<script>
$(document).ready(function() {

});

</script>

<!-- Only change code above this line. --&gt;

&lt;div class="container-fluid"&gt;
  &lt;h3 class="text-primary text-center"&gt;jQuery Playground&lt;/h3&gt;
  &lt;div class="row"&gt;
    &lt;div class="col-xs-6"&gt;
      &lt;h4&gt;#left-well&lt;/h4&gt;
      &lt;div class="well" id="left-well"&gt;
        &lt;button class="btn btn-default target" id="target1"&gt;#target1&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target2"&gt;#target2&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target3"&gt;#target3&lt;/button&gt;
      &lt;/div&gt;
    &lt;/div&gt;
    &lt;div class="col-xs-6"&gt;
      &lt;h4&gt;#right-well&lt;/h4&gt;
      &lt;div class="well" id="right-well"&gt;
        &lt;button class="btn btn-default target" id="target4"&gt;#target4&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target5"&gt;#target5&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target6"&gt;#target6&lt;/button&gt;
      &lt;/div&gt;
    &lt;/div&gt;
  &lt;/div&gt;
&lt;/div&gt;</pre>

```

Solution

```
// solution required
```

7. Remove Classes from an Element with jQuery

Description

In the same way you can add classes to an element with jQuery's `addClass()` function, you can remove them with jQuery's `removeClass()` function. Here's how you would do this for a specific button:

```
$("#target2").removeClass("btn-default");
```

Let's remove the `btn-default` class from all of our `button` elements.

Instructions

Challenge Seed

```
<script>
$(document).ready(function() {
  $("button").addClass("animated bounce");
  $(".well").addClass("animated shake");
```

```

$( "#target3" ).addClass("animated fadeIn");

});

</script>

<!-- Only change code above this line. --&gt;

&lt;div class="container-fluid"&gt;
  &lt;h3 class="text-primary text-center"&gt;jQuery Playground&lt;/h3&gt;
  &lt;div class="row"&gt;
    &lt;div class="col-xs-6"&gt;
      &lt;h4&gt;#left-well&lt;/h4&gt;
      &lt;div class="well" id="left-well"&gt;
        &lt;button class="btn btn-default target" id="target1"&gt;#target1&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target2"&gt;#target2&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target3"&gt;#target3&lt;/button&gt;
      &lt;/div&gt;
    &lt;/div&gt;
    &lt;div class="col-xs-6"&gt;
      &lt;h4&gt;#right-well&lt;/h4&gt;
      &lt;div class="well" id="right-well"&gt;
        &lt;button class="btn btn-default target" id="target4"&gt;#target4&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target5"&gt;#target5&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target6"&gt;#target6&lt;/button&gt;
      &lt;/div&gt;
    &lt;/div&gt;
  &lt;/div&gt;
&lt;/div&gt;
</pre>

```

Solution

```
// solution required
```

8. Change the CSS of an Element Using jQuery

Description

We can also change the CSS of an HTML element directly with jQuery. jQuery has a function called `.css()` that allows you to change the CSS of an element. Here's how we would change its color to blue: `$("#target1").css("color", "blue");` This is slightly different from a normal CSS declaration, because the CSS property and its value are in quotes, and separated with a comma instead of a colon. Delete your jQuery selectors, leaving an empty `document ready` function. Select `target1` and change its color to red.

Instructions

Challenge Seed

```

<script>
$(document).ready(function() {
  $("button").addClass("animated bounce");
  $(".well").addClass("animated shake");
  $("#target3").addClass("animated fadeIn");
  $("button").removeClass("btn-default");

});

</script>

<!-- Only change code above this line. --&gt;

&lt;div class="container-fluid"&gt;
  &lt;h3 class="text-primary text-center"&gt;jQuery Playground&lt;/h3&gt;
  &lt;div class="row"&gt;
    &lt;div class="col-xs-6"&gt;
      &lt;h4&gt;#left-well&lt;/h4&gt;
</pre>

```

```

<div class="well" id="left-well">
  <button class="btn btn-default target" id="target1">#target1</button>
  <button class="btn btn-default target" id="target2">#target2</button>
  <button class="btn btn-default target" id="target3">#target3</button>
</div>
</div>
<div class="col-xs-6">
  <h4>#right-well</h4>
  <div class="well" id="right-well">
    <button class="btn btn-default target" id="target4">#target4</button>
    <button class="btn btn-default target" id="target5">#target5</button>
    <button class="btn btn-default target" id="target6">#target6</button>
  </div>
</div>
</div>
</div>

```

Solution

```
// solution required
```

9. Disable an Element Using jQuery

Description

You can also change the non-CSS properties of HTML elements with jQuery. For example, you can disable buttons. When you disable a button, it will become grayed-out and can no longer be clicked. jQuery has a function called `.prop()` that allows you to adjust the properties of elements. Here's how you would disable all buttons:
`$(“button”).prop(“disabled”, true);` Disable only the `target1` button.

Instructions

Challenge Seed

```

<script>
$(document).ready(function() {
  $("#target1").css("color", "red");

});
</script>

<!-- Only change code above this line. --&gt;

&lt;div class="container-fluid"&gt;
  &lt;h3 class="text-primary text-center"&gt;jQuery Playground&lt;/h3&gt;
  &lt;div class="row"&gt;
    &lt;div class="col-xs-6"&gt;
      &lt;h4&gt;#left-well&lt;/h4&gt;
      &lt;div class="well" id="left-well"&gt;
        &lt;button class="btn btn-default target" id="target1"&gt;#target1&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target2"&gt;#target2&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target3"&gt;#target3&lt;/button&gt;
      &lt;/div&gt;
    &lt;/div&gt;
    &lt;div class="col-xs-6"&gt;
      &lt;h4&gt;#right-well&lt;/h4&gt;
      &lt;div class="well" id="right-well"&gt;
        &lt;button class="btn btn-default target" id="target4"&gt;#target4&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target5"&gt;#target5&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target6"&gt;#target6&lt;/button&gt;
      &lt;/div&gt;
    &lt;/div&gt;
  &lt;/div&gt;
&lt;/div&gt;
</pre>

```

Solution

```
// solution required
```

10. Change Text Inside an Element Using jQuery

Description

Using jQuery, you can change the text between the start and end tags of an element. You can even change HTML markup. jQuery has a function called `.html()` that lets you add HTML tags and text within an element. Any content previously within the element will be completely replaced with the content you provide using this function. Here's how you would rewrite and emphasize the text of our heading: `$("#h3").html("jQuery Playground");` jQuery also has a similar function called `.text()` that only alters text without adding tags. In other words, this function will not evaluate any HTML tags passed to it, but will instead treat it as the text you want to replace the existing content with. Change the button with id `target4` by emphasizing its text. Check this [link](#) to know more on the difference between `<i>` and `` and their uses. Note that while the `<i>` tag has traditionally been used to emphasize text, it has since been coopted for use as a tag for icons. The `` tag is now widely accepted as the tag for emphasis. Either will work for this challenge.

Instructions

Challenge Seed

```
<script>
$(document).ready(function() {
  $("#target1").css("color", "red");

});
</script>

<!-- Only change code above this line. --&gt;

<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
      <h4>#left-well</h4>
      <div class="well" id="left-well">
        <button class="btn btn-default target" id="target1">#target1</button>
        <button class="btn btn-default target" id="target2">#target2</button>
        <button class="btn btn-default target" id="target3">#target3</button>
      </div>
    </div>
    <div class="col-xs-6">
      <h4>#right-well</h4>
      <div class="well" id="right-well">
        <button class="btn btn-default target" id="target4">#target4</button>
        <button class="btn btn-default target" id="target5">#target5</button>
        <button class="btn btn-default target" id="target6">#target6</button>
      </div>
    </div>
  </div>

```

Solution

```
<script>
$(document).ready(function() {
  $("#target1").css("color", "red");
  $("#target4").html('<em>#target4</em>');
});
</script>
```

```

</script>

<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
      <h4>#left-well</h4>
      <div class="well" id="left-well">
        <button class="btn btn-default target" id="target1">#target1</button>
        <button class="btn btn-default target" id="target2">#target2</button>
        <button class="btn btn-default target" id="target3">#target3</button>
      </div>
    </div>
    <div class="col-xs-6">
      <h4>#right-well</h4>
      <div class="well" id="right-well">
        <button class="btn btn-default target" id="target4">#target4</button>
        <button class="btn btn-default target" id="target5">#target5</button>
        <button class="btn btn-default target" id="target6">#target6</button>
      </div>
    </div>
  </div>
</div>

```

11. Remove an Element Using jQuery

Description

Now let's remove an HTML element from your page using jQuery. jQuery has a function called `.remove()` that will remove an HTML element entirely. Remove element `target4` from the page by using the `.remove()` function.

Instructions

Challenge Seed

```

<script>
$(document).ready(function() {
  $("#target1").css("color", "red");
  $("#target1").prop("disabled", true);

});
</script>

<!-- Only change code above this line. --&gt;

&lt;div class="container-fluid"&gt;
  &lt;h3 class="text-primary text-center"&gt;jQuery Playground&lt;/h3&gt;
  &lt;div class="row"&gt;
    &lt;div class="col-xs-6"&gt;
      &lt;h4&gt;#left-well&lt;/h4&gt;
      &lt;div class="well" id="left-well"&gt;
        &lt;button class="btn btn-default target" id="target1"&gt;#target1&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target2"&gt;#target2&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target3"&gt;#target3&lt;/button&gt;
      &lt;/div&gt;
    &lt;/div&gt;
    &lt;div class="col-xs-6"&gt;
      &lt;h4&gt;#right-well&lt;/h4&gt;
      &lt;div class="well" id="right-well"&gt;
        &lt;button class="btn btn-default target" id="target4"&gt;#target4&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target5"&gt;#target5&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target6"&gt;#target6&lt;/button&gt;
      &lt;/div&gt;
    &lt;/div&gt;
  &lt;/div&gt;
&lt;/div&gt;
</pre>

```

Solution

```
// solution required
```

12. Use appendTo to Move Elements with jQuery

Description

Now let's try moving elements from one `div` to another. jQuery has a function called `appendTo()` that allows you to select HTML elements and append them to another element. For example, if we wanted to move `target4` from our right well to our left well, we would use: `$("#target4").appendTo("#left-well");` Move your `target2` element from your `left-well` to your `right-well`.

Instructions

Challenge Seed

```
<script>
$(document).ready(function() {
  $("#target1").css("color", "red");
  $("#target1").prop("disabled", true);
  $("#target4").remove();

});
</script>

<!-- Only change code above this line. --&gt;

&lt;div class="container-fluid"&gt;
  &lt;h3 class="text-primary text-center"&gt;jQuery Playground&lt;/h3&gt;
  &lt;div class="row"&gt;
    &lt;div class="col-xs-6"&gt;
      &lt;h4&gt;#left-well&lt;/h4&gt;
      &lt;div class="well" id="left-well"&gt;
        &lt;button class="btn btn-default target" id="target1"&gt;#target1&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target2"&gt;#target2&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target3"&gt;#target3&lt;/button&gt;
      &lt;/div&gt;
    &lt;/div&gt;
    &lt;div class="col-xs-6"&gt;
      &lt;h4&gt;#right-well&lt;/h4&gt;
      &lt;div class="well" id="right-well"&gt;
        &lt;button class="btn btn-default target" id="target4"&gt;#target4&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target5"&gt;#target5&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target6"&gt;#target6&lt;/button&gt;
      &lt;/div&gt;
    &lt;/div&gt;
  &lt;/div&gt;
&lt;/div&gt;</pre>

```

Solution

```
// solution required
```

13. Clone an Element Using jQuery

Description

In addition to moving elements, you can also copy them from one place to another. jQuery has a function called `clone()` that makes a copy of an element. For example, if we wanted to copy `target2` from our `left-well` to our `right-well`, we would use: `$("#target2").clone().appendTo("#right-well");` Did you notice this involves sticking two jQuery functions together? This is called `function chaining` and it's a convenient way to get things done with jQuery. Clone your `target5` element and append it to your `left-well`.

Instructions

Challenge Seed

```
<script>
$(document).ready(function() {
  $("#target1").css("color", "red");
  $("#target1").prop("disabled", true);
  $("#target4").remove();
  $("#target2").appendTo("#right-well");

});
</script>

<!-- Only change code above this line. -->

<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
      <h4>#left-well</h4>
      <div class="well" id="left-well">
        <button class="btn btn-default target" id="target1">#target1</button>
        <button class="btn btn-default target" id="target2">#target2</button>
        <button class="btn btn-default target" id="target3">#target3</button>
      </div>
    </div>
    <div class="col-xs-6">
      <h4>#right-well</h4>
      <div class="well" id="right-well">
        <button class="btn btn-default target" id="target4">#target4</button>
        <button class="btn btn-default target" id="target5">#target5</button>
        <button class="btn btn-default target" id="target6">#target6</button>
      </div>
    </div>
  </div>
</div>
```

Solution

```
// solution required
```

14. Target the Parent of an Element Using jQuery

Description

Every HTML element has a `parent` element from which it inherits properties. For example, your `jQuery` `Playground` `h3` element has the parent element of `<div class="container-fluid">`, which itself has the parent `body`. jQuery has a function called `parent()` that allows you to access the parent of whichever element you've selected. Here's an example of how you would use the `parent()` function if you wanted to give the parent element of the `left-well` element a background color of blue: `$("#left-well").parent().css("background-color", "blue")` Give the parent of the `#target1` element a `background-color` of red.

Instructions

Challenge Seed

```

<script>
$(document).ready(function() {
  $("#target1").css("color", "red");
  $("#target1").prop("disabled", true);
  $("#target4").remove();
  $("#target2").appendTo("#right-well");
  $("#target5").clone().appendTo("#left-well");

});
</script>

<!-- Only change code above this line. --&gt;

&lt;body&gt;
&lt;div class="container-fluid"&gt;
  &lt;h3 class="text-primary text-center"&gt;jQuery Playground&lt;/h3&gt;
  &lt;div class="row"&gt;
    &lt;div class="col-xs-6"&gt;
      &lt;h4&gt;#left-well&lt;/h4&gt;
      &lt;div class="well" id="left-well"&gt;
        &lt;button class="btn btn-default target" id="target1"&gt;#target1&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target2"&gt;#target2&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target3"&gt;#target3&lt;/button&gt;
      &lt;/div&gt;
    &lt;/div&gt;
    &lt;div class="col-xs-6"&gt;
      &lt;h4&gt;#right-well&lt;/h4&gt;
      &lt;div class="well" id="right-well"&gt;
        &lt;button class="btn btn-default target" id="target4"&gt;#target4&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target5"&gt;#target5&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target6"&gt;#target6&lt;/button&gt;
      &lt;/div&gt;
    &lt;/div&gt;
  &lt;/div&gt;
&lt;/body&gt;
</pre>

```

Solution

```
// solution required
```

15. Target the Children of an Element Using jQuery

Description

When HTML elements are placed one level below another they are called `children` of that element. For example, the button elements in this challenge with the text "#target1", "#target2", and "#target3" are all `children` of the `<div class="well" id="left-well">` element. jQuery has a function called `children()` that allows you to access the children of whichever element you've selected. Here's an example of how you would use the `children()` function to give the children of your `left-well` element the color blue : `$("#left-well").children().css("color", "blue")`

Instructions

Give all the children of your `right-well` element the color orange.

Challenge Seed

```

<script>
$(document).ready(function() {
  $("#target1").css("color", "red");
  $("#target1").prop("disabled", true);

```

```

$( "#target4" ).remove();
$( "#target2" ).appendTo( "#right-well" );
$( "#target5" ).clone().appendTo( "#left-well" );
$( "#target1" ).parent().css("background-color", "red");

});

</script>

<!-- Only change code above this line. --&gt;

&lt;div class="container-fluid"&gt;
  &lt;h3 class="text-primary text-center"&gt;jQuery Playground&lt;/h3&gt;
  &lt;div class="row"&gt;
    &lt;div class="col-xs-6"&gt;
      &lt;h4&gt;#left-well&lt;/h4&gt;
      &lt;div class="well" id="left-well"&gt;
        &lt;button class="btn btn-default target" id="target1"&gt;#target1&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target2"&gt;#target2&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target3"&gt;#target3&lt;/button&gt;
      &lt;/div&gt;
    &lt;/div&gt;
    &lt;div class="col-xs-6"&gt;
      &lt;h4&gt;#right-well&lt;/h4&gt;
      &lt;div class="well" id="right-well"&gt;
        &lt;button class="btn btn-default target" id="target4"&gt;#target4&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target5"&gt;#target5&lt;/button&gt;
        &lt;button class="btn btn-default target" id="target6"&gt;#target6&lt;/button&gt;
      &lt;/div&gt;
    &lt;/div&gt;
  &lt;/div&gt;
&lt;/div&gt;
</pre>

```

Solution

```
// solution required
```

16. Target a Specific Child of an Element Using jQuery

Description

You've seen why id attributes are so convenient for targeting with jQuery selectors. But you won't always have such neat ids to work with. Fortunately, jQuery has some other tricks for targeting the right elements. jQuery uses CSS Selectors to target elements. The `target:nth-child(n)` CSS selector allows you to select all the nth elements with the target class or element type. Here's how you would give the third element in each well the bounce class:

`$(".target:nth-child(3)").addClass("animated bounce");` Make the second child in each of your well elements bounce. You must select the elements' children with the `target` class.

Instructions

Challenge Seed

```

<script>
$(document).ready(function() {
  $("#target1").css("color", "red");
  $("#target1").prop("disabled", true);
  $("#target4").remove();
  $("#target2").appendTo("#right-well");
  $("#target5").clone().appendTo("#left-well");
  $("#target1").parent().css("background-color", "red");
  $("#right-well").children().css("color", "orange");

});
</script>

```

```
<!-- Only change code above this line. -->

<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
      <h4>#left-well</h4>
      <div class="well" id="left-well">
        <button class="btn btn-default target" id="target1">#target1</button>
        <button class="btn btn-default target" id="target2">#target2</button>
        <button class="btn btn-default target" id="target3">#target3</button>
      </div>
    </div>
    <div class="col-xs-6">
      <h4>#right-well</h4>
      <div class="well" id="right-well">
        <button class="btn btn-default target" id="target4">#target4</button>
        <button class="btn btn-default target" id="target5">#target5</button>
        <button class="btn btn-default target" id="target6">#target6</button>
      </div>
    </div>
  </div>
</div>
```

Solution

```
// solution required
```

17. Target Even Elements Using jQuery

Description

You can also target elements based on their positions using `:odd` or `:even` selectors. Note that jQuery is zero-indexed which means the first element in a selection has a position of 0. This can be a little confusing as, counter-intuitively, `:odd` selects the second element (position 1), fourth element (position 3), and so on. Here's how you would target all the odd elements with class `target` and give them classes: `$(".target:odd").addClass("animated shake");` Try selecting all the even `target` elements and giving them the classes of `animated` and `shake`. Remember that `even` refers to the position of elements with a zero-based system in mind.

Instructions

Challenge Seed

```
<script>
$(document).ready(function() {
  $("#target1").css("color", "red");
  $("#target1").prop("disabled", true);
  $("#target4").remove();
  $("#target2").appendTo("#right-well");
  $("#target5").clone().appendTo("#left-well");
  $("#target1").parent().css("background-color", "red");
  $("#right-well").children().css("color", "orange");
  $("#left-well").children().css("color", "green");
  $(".target:nth-child(2)").addClass("animated bounce");

});
</script>
```

```
<!-- Only change code above this line. -->

<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
```

```

<h4>#left-well</h4>
<div class="well" id="left-well">
  <button class="btn btn-default target" id="target1">#target1</button>
  <button class="btn btn-default target" id="target2">#target2</button>
  <button class="btn btn-default target" id="target3">#target3</button>
</div>
</div>
<div class="col-xs-6">
  <h4>#right-well</h4>
  <div class="well" id="right-well">
    <button class="btn btn-default target" id="target4">#target4</button>
    <button class="btn btn-default target" id="target5">#target5</button>
    <button class="btn btn-default target" id="target6">#target6</button>
  </div>
</div>
</div>

```

Solution

```
// solution required
```

18. Use jQuery to Modify the Entire Page

Description

We're done playing with our jQuery playground. Let's tear it down! jQuery can target the `body` element as well. Here's how we would make the entire body fade out: `$("body").addClass("animated fadeIn")`; But let's do something more dramatic. Add the classes `animated` and `hinge` to your `body` element.

Instructions

Challenge Seed

```

<script>
$(document).ready(function() {
  $("#target1").css("color", "red");
  $("#target1").prop("disabled", true);
  $("#target4").remove();
  $("#target2").appendTo("#right-well");
  $("#target5").clone().appendTo("#left-well");
  $("#target1").parent().css("background-color", "red");
  $("#right-well").children().css("color", "orange");
  $("#left-well").children().css("color", "green");
  $(".target:nth-child(2)").addClass("animated bounce");
  $(".target:even").addClass("animated shake");

});
</script>

<!-- Only change code above this line. -->

<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
      <h4>#left-well</h4>
      <div class="well" id="left-well">
        <button class="btn btn-default target" id="target1">#target1</button>
        <button class="btn btn-default target" id="target2">#target2</button>
        <button class="btn btn-default target" id="target3">#target3</button>
      </div>
    </div>
    <div class="col-xs-6">
      <h4>#right-well</h4>
    </div>
  </div>
</div>

```

```
<div class="well" id="right-well">
  <button class="btn btn-default target" id="target4">#target4</button>
  <button class="btn btn-default target" id="target5">#target5</button>
  <button class="btn btn-default target" id="target6">#target6</button>
</div>
</div>
</div>
</div>
```

Solution

```
// solution required
```

Sass

1. Store Data with Sass Variables

Description

One feature of Sass that's different than CSS is it uses variables. They are declared and set to store data, similar to JavaScript. In JavaScript, variables are defined using the `let` and `const` keywords. In Sass, variables start with a `$` followed by the variable name. Here are a couple examples:

```
$main-fonts: Arial, sans-serif;
$headings-color: green;

//To use variables:
h1 {
  font-family: $main-fonts;
  color: $headings-color;
}
```

One example where variables are useful is when a number of elements need to be the same color. If that color is changed, the only place to edit the code is the variable value.

Instructions

Create a variable `$text-color` and set it to red. Then change the value of the `color` property for the `.blog-post` and `h2` to the `$text-color` variable.

Challenge Seed

```
<style type='text/sass'>

  .header{
    text-align: center;
  }
  .blog-post, h2 {
    color: red;
  }
</style>

<h1 class="header">Learn Sass</h1>
<div class="blog-post">
  <h2>Some random title</h2>
  <p>This is a paragraph with some random text in it</p>
</div>
<div class="blog-post">
  <h2>Header #2</h2>
  <p>Here is some more random text.</p>
```

```
</div>
<div class="blog-post">
  <h2>Here is another header</h2>
  <p>Even more random text within a paragraph</p>
</div>
```

Solution

```
// solution required
```

2. Nest CSS with Sass

Description

Sass allows nesting of CSS rules, which is a useful way of organizing a style sheet. Normally, each element is targeted on a different line to style it, like so:

```
nav {
  background-color: red;
}

nav ul {
  list-style: none;
}

nav ul li {
  display: inline-block;
}
```

For a large project, the CSS file will have many lines and rules. This is where nesting can help organize your code by placing child style rules within the respective parent elements:

```
nav {
  background-color: red;

  ul {
    list-style: none;

    li {
      display: inline-block;
    }
  }
}
```

Instructions

Use the nesting technique shown above to re-organize the CSS rules for both children of `.blog-post` element. For testing purposes, the `h1` should come before the `p` element.

Challenge Seed

```
<style type='text/sass'>
.blog-post {

}

h1 {
  text-align: center;
  color: blue;
}

p {
  font-size: 20px;
```

```

        }
    </style>

<div class="blog-post">
    <h1>Blog Title</h1>
    <p>This is a paragraph</p>
</div>

```

Solution

```
// solution required
```

3. Create Reusable CSS with Mixins

Description

In Sass, a `mixin` is a group of CSS declarations that can be reused throughout the style sheet. Newer CSS features take time before they are fully adopted and ready to use in all browsers. As features are added to browsers, CSS rules using them may need vendor prefixes. Consider "box-shadow":

```

div {
    -webkit-box-shadow: 0px 0px 4px #fff;
    -moz-box-shadow: 0px 0px 4px #fff;
    -ms-box-shadow: 0px 0px 4px #fff;
    box-shadow: 0px 0px 4px #fff;
}

```

It's a lot of typing to re-write this rule for all the elements that have a `box-shadow`, or to change each value to test different effects. `Mixins` are like functions for CSS. Here is how to write one:

```

@mixin box-shadow($x, $y, $blur, $c){
    -webkit-box-shadow: $x, $y, $blur, $c;
    -moz-box-shadow: $x, $y, $blur, $c;
    -ms-box-shadow: $x, $y, $blur, $c;
    box-shadow: $x, $y, $blur, $c;
}

```

The definition starts with `@mixin` followed by a custom name. The parameters (the `$x`, `$y`, `$blur`, and `$c` in the example above) are optional. Now any time a `box-shadow` rule is needed, only a single line calling the `mixin` replaces having to type all the vendor prefixes. A `mixin` is called with the `@include` directive:

```

div {
    @include box-shadow(0px, 0px, 4px, #fff);
}

```

Instructions

Write a `mixin` for `border-radius` and give it a `$radius` parameter. It should use all the vendor prefixes from the example. Then use the `border-radius` `mixin` to give the `#awesome` element a border radius of 15px.

Challenge Seed

```

<style type='text/sass'>

#awesome {
    width: 150px;
    height: 150px;
    background-color: green;

}
</style>

```

```
<div id="awesome"></div>
```

Solution

```
// solution required
```

4. Use @if and @else to Add Logic To Your Styles

Description

The `@if` directive in Sass is useful to test for a specific case - it works just like the `if` statement in JavaScript.

```
@ mixin make-bold($bool) {
  @if $bool == true {
    font-weight: bold;
  }
}
```

And just like in JavaScript, `@else if` and `@else` test for more conditions:

```
@ mixin text-effect($val) {
  @if $val == danger {
    color: red;
  }
  @else if $val == alert {
    color: yellow;
  }
  @else if $val == success {
    color: green;
  }
  @else {
    color: black;
  }
}
```

Instructions

Create a mixin called `border-stroke` that takes a parameter `$val`. The mixin should check for the following conditions using `@if`, `@else if`, and `@else`:

- light - 1px solid black
- medium - 3px solid black
- heavy - 6px solid black

If `$val` is not `light`, `medium`, or `heavy`, the border should be set to `none`.

Challenge Seed

```
<style type='text/sass'>
```

```
#box {
  width: 150px;
  height: 150px;
  background-color: red;
  @include border-stroke(medium);
}
</style>

<div id="box"></div>
```

Solution

```
// solution required
```

5. Use @for to Create a Sass Loop

Description

The `@for` directive adds styles in a loop, very similar to a `for` loop in JavaScript. `@for` is used in two ways: "start through end" or "start to end". The main difference is that the "start to end" *excludes* the end number as part of the count, and "start **through** end" *includes* the end number as part of the count. Here's a start **through** end example:

```
@for $i from 1 through 12 {  
  .col-#${$i} { width: 100%/12 * $i; }  
}
```

The `#$i` part is the syntax to combine a variable (`i`) with text to make a string. When the Sass file is converted to CSS, it looks like this:

```
.col-1 {  
  width: 8.33333%;  
}  
  
.col-2 {  
  width: 16.66667%;  
}  
  
...  
  
.col-12 {  
  width: 100%;  
}
```

This is a powerful way to create a grid layout. Now you have twelve options for column widths available as CSS classes.

Instructions

Write a `@for` directive that takes a variable `$j` that goes from 1 **to** 6. It should create 5 classes called `.text-1` to `.text-5` where each has a `font-size` set to 10px multiplied by the index.

Challenge Seed

```
<style type='text/sass'>  
  
  </style>  
  
  <p class="text-1">Hello</p>  
  <p class="text-2">Hello</p>  
  <p class="text-3">Hello</p>  
  <p class="text-4">Hello</p>  
  <p class="text-5">Hello</p>
```

Solution

```
<style type='text/sass'>  
  
  @for $i from 1 through 5 {
```

```

.text-#${i} { font-size: 10px * $i; }

}

</style>

<p class="text-1">Hello</p>
<p class="text-2">Hello</p>
<p class="text-3">Hello</p>
<p class="text-4">Hello</p>
<p class="text-5">Hello</p>

<style type='text/sass'>

@for $i from 1 to 6 {
  .text-#${i} { font-size: 10px * $i; }
}

</style>

<p class="text-1">Hello</p>
<p class="text-2">Hello</p>
<p class="text-3">Hello</p>
<p class="text-4">Hello</p>
<p class="text-5">Hello</p>

```

6. Use @each to Map Over Items in a List

Description

The last challenge showed how the `@for` directive uses a starting and ending value to loop a certain number of times. Sass also offers the `@each` directive which loops over each item in a list or map. On each iteration, the variable gets assigned to the current value from the list or map.

```

@each $color in blue, red, green {
  .#${color}-text {color: $color;}
}

```

A map has slightly different syntax. Here's an example:

```

$colors: (color1: blue, color2: red, color3: green);

@each $key, $color in $colors {
  .#${color}-text {color: $color;}
}

```

Note that the `$key` variable is needed to reference the keys in the map. Otherwise, the compiled CSS would have `color1`, `color2` ... in it. Both of the above code examples are converted into the following CSS:

```

.blue-text {
  color: blue;
}

.red-text {
  color: red;
}

.green-text {
  color: green;
}

```

Instructions

Write an `@each` directive that goes through a list: `blue, black, red` and assigns each variable to a `.color-bg` class, where the "color" part changes for each item. Each class should set the `background-color` the respective color.

Challenge Seed

```
<style type='text/sass'>

div {
  height: 200px;
  width: 200px;
}
</style>

<div class="blue-bg"></div>
<div class="black-bg"></div>
<div class="red-bg"></div>
```

Solution

The solution requires using the \$color variable twice: once for the class name and once for setting the background color. You can use either the list or map data type.

List Data type

```
<style type='text/sass'>

@each $color in blue, black, red {
  .#{$color}-bg {background-color: $color;}
}

div {
  height: 200px;
  width: 200px;
}
</style>

<div class="blue-bg"></div>
<div class="black-bg"></div>
<div class="red-bg"></div>
```

Map Data type

```
<style type='text/sass'>

$colors: (color1: blue, color2: black, color3: red);

@each $key, $color in $colors {
  .#{$color}-bg {background-color: $color;}
}

div {
  height: 200px;
  width: 200px;
}
</style>

<div class="blue-bg"></div>
<div class="black-bg"></div>
<div class="red-bg"></div>
```

7. Apply a Style Until a Condition is Met with @while

Description

The `@while` directive is an option with similar functionality to the JavaScript `while` loop. It creates CSS rules until a condition is met. The `@for` challenge gave an example to create a simple grid system. This can also work with `@while`.

```
$x: 1;
@while $x < 13 {
  .col-#{$x} { width: 100%/12 * $x; }
  $x: $x + 1;
}
```

First, define a variable `$x` and set it to 1. Next, use the `@while` directive to create the grid system while `$x` is less than 13. After setting the CSS rule for `width`, `$x` is incremented by 1 to avoid an infinite loop.

Instructions

Use `@while` to create a series of classes with different `font-sizes`. There should be 10 different classes from `text-1` to `text-10`. Then set `font-size` to 5px multiplied by the current index number. Make sure to avoid an infinite loop!

Challenge Seed

```
<style type='text/sass'>

</style>

<p class="text-1">Hello</p>
<p class="text-2">Hello</p>
<p class="text-3">Hello</p>
<p class="text-4">Hello</p>
<p class="text-5">Hello</p>
<p class="text-6">Hello</p>
<p class="text-7">Hello</p>
<p class="text-8">Hello</p>
<p class="text-9">Hello</p>
<p class="text-10">Hello</p>
```

Solution

```
// solution required
```

8. Split Your Styles into Smaller Chunks with Partials

Description

Partials in Sass are separate files that hold segments of CSS code. These are imported and used in other Sass files. This is a great way to group similar code into a module to keep it organized. Names for partials start with the underscore (`_`) character, which tells Sass it is a small segment of CSS and not to convert it into a CSS file. Also, Sass files end with the `.scss` file extension. To bring the code in the partial into another Sass file, use the `@import` directive. For example, if all your `mixins` are saved in a partial named `"_mixins.scss"`, and they are needed in the `"main.scss"` file, this is how to use them in the main file:

```
// In the main.scss file
```

```
@import 'mixins'
```

Note that the underscore is not needed in the `import` statement - Sass understands it is a `partial`. Once a `partial` is imported into a file, all variables, `mixins`, and other code are available to use.

Instructions

Write an `@import` statement to import a partial named `_variables.scss` into the `main.scss` file.

Challenge Seed

```
// The main.scss file
```

Solution

```
// solution required
```

9. Extend One Set of CSS Styles to Another Element

Description

Sass has a feature called `extend` that makes it easy to borrow the CSS rules from one element and build upon them in another. For example, the below block of CSS rules style a `.panel` class. It has a `background-color`, `height` and `border`.

```
.panel{
  background-color: red;
  height: 70px;
  border: 2px solid green;
}
```

Now you want another panel called `.big-panel`. It has the same base properties as `.panel`, but also needs a `width` and `font-size`. It's possible to copy and paste the initial CSS rules from `.panel`, but the code becomes repetitive as you add more types of panels. The `extend` directive is a simple way to reuse the rules written for one element, then add more for another:

```
.big-panel{
  @extend .panel;
  width: 150px;
  font-size: 2em;
}
```

The `.big-panel` will have the same properties as `.panel` in addition to the new styles.

Instructions

Make a class `.info-important` that extends `.info` and also has a `background-color` set to magenta.

Challenge Seed

```
<style type='text/sass'>
  h3{
    text-align: center;
  }
  .info{
    width: 200px;
    border: 1px solid black;
    margin: 0 auto;
  }

</style>
<h3>Posts</h3>
```

```
<div class="info-important">
  <p>This is an important post. It should extend the class ".info" and have its own CSS styles.</p>
</div>

<div class="info">
  <p>This is a simple post. It has basic styling and can be extended for other uses.</p>
</div>
```

Solution

```
// solution required
```

React

1. Create a Simple JSX Element

Description

Intro: React is an Open Source view library created and maintained by Facebook. It's a great tool to render the User Interface (UI) of modern web applications. React uses a syntax extension of JavaScript called JSX that allows you to write HTML directly within JavaScript. This has several benefits. It lets you use the full programmatic power of JavaScript within HTML, and helps to keep your code readable. For the most part, JSX is similar to the HTML that you have already learned, however there are a few key differences that will be covered throughout these challenges. For instance, because JSX is a syntactic extension of JavaScript, you can actually write JavaScript directly within JSX. To do this, you simply include the code you want to be treated as JavaScript within curly braces: { 'this is treated as JavaScript code' } . Keep this in mind, since it's used in several future challenges. However, because JSX is not valid JavaScript, JSX code must be compiled into JavaScript. The transpiler Babel is a popular tool for this process. For your convenience, it's already added behind the scenes for these challenges. If you happen to write syntactically invalid JSX, you will see the first test in these challenges fail. It's worth noting that under the hood the challenges are calling `ReactDOM.render(JSX, document.getElementById('root'))` . This function call is what places your JSX into React's own lightweight representation of the DOM. React then uses snapshots of its own DOM to optimize updating only specific parts of the actual DOM.

Instructions

Instructions: The current code uses JSX to assign a `div` element to the constant `JSX` . Replace the `div` with an `h1` element and add the text `Hello JSX!` inside it.

Challenge Seed

```
const JSX = <div></div>;
```

After Test

```
ReactDOM.render(JSX, document.getElementById('root'))
```

Solution

```
const JSX = <h1>Hello JSX!</h1>;
```

2. Create a Complex JSX Element

Description

The last challenge was a simple example of JSX, but JSX can represent more complex HTML as well. One important thing to know about nested JSX is that it must return a single element. This one parent element would wrap all of the other levels of nested elements. For instance, several JSX elements written as siblings with no parent wrapper element will not transpile. Here's an example: **Valid JSX:**

```
<div>
  <p>Paragraph One</p>
  <p>Paragraph Two</p>
  <p>Paragraph Three</p>
</div>
```

Invalid JSX:

```
<p>Paragraph One</p>
<p>Paragraph Two</p>
<p>Paragraph Three</p>
```

Instructions

Define a new constant `JSX` that renders a `div` which contains the following elements in order: An `h1`, a `p`, and an unordered list that contains three `li` items. You can include any text you want within each element. **Note:** When rendering multiple elements like this, you can wrap them all in parentheses, but it's not strictly required. Also notice this challenge uses a `div` tag to wrap all the child elements within a single parent element. If you remove the `div`, the JSX will no longer transpile. Keep this in mind, since it will also apply when you return JSX elements in React components.

Challenge Seed

```
// write your code here
```

After Test

```
ReactDOM.render(JSX, document.getElementById('root'))
```

Solution

```
const JSX = (
  <div>
    <h1>Hello JSX!</h1>
    <p>Some info</p>
    <ul>
      <li>An item</li>
      <li>Another item</li>
      <li>A third item</li>
    </ul>
  </div>);
```

3. Add Comments in JSX

Description

JSX is a syntax that gets compiled into valid JavaScript. Sometimes, for readability, you might need to add comments to your code. Like most programming languages, JSX has its own way to do this. To put comments inside JSX, you use the syntax `{/* */}` to wrap around the comment text.

Instructions

The code editor has a JSX element similar to what you created in the last challenge. Add a comment somewhere within the provided `div` element, without modifying the existing `h1` or `p` elements.

Challenge Seed

```
const JSX = (
  <div>
    <h1>This is a block of JSX</h1>
    <p>Here's a subtitle</p>
  </div>
);
```

After Test

```
ReactDOM.render(JSX, document.getElementById('root'))
```

Solution

```
const JSX = (
  <div>
    <h1>This is a block of JSX</h1>
    { /* this is a JSX comment */
      <p>Here's a subtitle</p>
    }
  </div>);
```

4. Render HTML Elements to the DOM

Description

So far, you've learned that JSX is a convenient tool to write readable HTML within JavaScript. With React, we can render this JSX directly to the HTML DOM using React's rendering API known as ReactDOM. ReactDOM offers a simple method to render React elements to the DOM which looks like this: `ReactDOM.render(componentToRender, targetNode)`, where the first argument is the React element or component that you want to render, and the second argument is the DOM node that you want to render the component to. As you would expect, `ReactDOM.render()` must be called after the JSX element declarations, just like how you must declare variables before using them.

Instructions

The code editor has a simple JSX component. Use the `ReactDOM.render()` method to render this component to the page. You can pass defined JSX elements directly in as the first argument and use `document.getElementById()` to select the DOM node to render them to. There is a `div` with `id='challenge-node'` available for you to use. Make sure you don't change the `JSX` constant.

Challenge Seed

```
const JSX = (
  <div>
    <h1>Hello World</h1>
    <p>Lets render this to the DOM</p>
  </div>
);
// change code below this line
```

Solution

```
const JSX = (
  <div>
    <h1>Hello World</h1>
    <p>Let's render this to the DOM</p>
  </div>
);
// change code below this line
ReactDOM.render(JSX, document.getElementById('challenge-node'));
```

5. Define an HTML Class in JSX

Description

Now that you're getting comfortable writing JSX, you may be wondering how it differs from HTML. So far, it may seem that HTML and JSX are exactly the same. One key difference in JSX is that you can no longer use the word `class` to define HTML classes. This is because `class` is a reserved word in JavaScript. Instead, JSX uses `className`. In fact, the naming convention for all HTML attributes and event references in JSX become camelCase. For example, a click event in JSX is `onClick`, instead of `onclick`. Likewise, `onchange` becomes `onChange`. While this is a subtle difference, it is an important one to keep in mind moving forward.

Instructions

Apply a class of `myDiv` to the `div` provided in the JSX code.

Challenge Seed

```
const JSX = (
  <div>
    <h1>Add a class to this div</h1>
  </div>
);
```

After Test

```
ReactDOM.render(JSX, document.getElementById('root'))
```

Solution

```
const JSX = (
  <div className = 'myDiv'>
    <h1>Add a class to this div</h1>
  </div>);
```

6. Learn About Self-Closing JSX Tags

Description

So far, you've seen how JSX differs from HTML in a key way with the use of `className` vs. `class` for defining HTML classes. Another important way in which JSX differs from HTML is in the idea of the self-closing tag. In HTML, almost all tags have both an opening and closing tag: `<div></div>`; the closing tag always has a forward slash before the tag name that you are closing. However, there are special instances in HTML called "self-closing tags", or tags that don't require both an opening and closing tag before another tag can start. For example the line-break tag can be written as `
` or as `
`, but should never be written as `
</br>`, since it doesn't contain any content. In JSX, the rules are a little different. Any JSX element can be written with a self-closing tag, and every element must be closed.

The line-break tag, for example, must always be written as `
` in order to be valid JSX that can be transpiled. A `<div>`, on the other hand, can be written as `<div />` or `<div></div>`. The difference is that in the first syntax version there is no way to include anything in the `<div />`. You will see in later challenges that this syntax is useful when rendering React components.

Instructions

Fix the errors in the code editor so that it is valid JSX and successfully transpiles. Make sure you don't change any of the content - you only need to close tags where they are needed.

Challenge Seed

```
const JSX = (
  <div>
    /* remove comment and change code below this line
    <h2>Welcome to React!</h2> <br />
    <p>Be sure to close all tags!</p>
    <hr />
    remove comment and change code above this line */
  </div>
);
```

After Test

```
ReactDOM.render(JSX, document.getElementById('root'))
```

Solution

```
const JSX = (
  <div>
    /* change code below this line */
    <h2>Welcome to React!</h2> <br />
    <p>Be sure to close all tags!</p>
    <hr />
    /* change code above this line */
  </div>
);
```

7. Create a Stateless Functional Component

Description

Components are the core of React. Everything in React is a component and here you will learn how to create one. There are two ways to create a React component. The first way is to use a JavaScript function. Defining a component in this way creates a *stateless functional component*. The concept of state in an application will be covered in later challenges. For now, think of a stateless component as one that can receive data and render it, but does not manage or track changes to that data. (We'll cover the second way to create a React component in the next challenge.) To create a component with a function, you simply write a JavaScript function that returns either JSX or `null`. One important thing to note is that React requires your function name to begin with a capital letter. Here's an example of a stateless functional component that assigns an HTML class in JSX:

```
// After being transpiled, the <div> will have a CSS class of 'customClass'
const DemoComponent = function() {
  return (
    <div className='customClass' />
  );
};
```

Because a JSX component represents HTML, you could put several components together to create a more complex HTML page. This is one of the key advantages of the component architecture React provides. It allows you to compose your UI from many separate, isolated components. This makes it easier to build and maintain complex user interfaces.

Instructions

The code editor has a function called `MyComponent`. Complete this function so it returns a single `div` element which contains some string of text. **Note:** The text is considered a child of the `div` element, so you will not be able to use a self-closing tag.

Challenge Seed

```
const MyComponent = function() {
  // change code below this line

  // change code above this line
}
```

After Test

```
ReactDOM.render(<MyComponent />, document.getElementById('root'))
```

Solution

```
const MyComponent = function() {
  // change code below this line
  return (
    <div>
      Demo Solution
    </div>
  );
  // change code above this line
}
```

8. Create a React Component

Description

The other way to define a React component is with the ES6 `class` syntax. In the following example, `Kitten` extends `React.Component`:

```
class Kitten extends React.Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <h1>Hi</h1>
    );
  }
}
```

This creates an ES6 class `Kitten` which extends the `React.Component` class. So the `Kitten` class now has access to many useful React features, such as local state and lifecycle hooks. Don't worry if you aren't familiar with these terms yet, they will be covered in greater detail in later challenges. Also notice the `Kitten` class has a `constructor` defined within it that calls `super()`. It uses `super()` to call the constructor of the parent class, in this case `React.Component`.

The constructor is a special method used during the initialization of objects that are created with the `class` keyword. It is best practice to call a component's `constructor` with `super`, and pass `props` to both. This makes sure the component is initialized properly. For now, know that it is standard for this code to be included. Soon you will see other uses for the `constructor` as well as `props`.

Instructions

`MyComponent` is defined in the code editor using class syntax. Finish writing the `render` method so it returns a `div` element that contains an `h1` with the text `Hello React!`.

Challenge Seed

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    // change code below this line

    // change code above this line
  }
};
```

After Test

```
ReactDOM.render(<MyComponent />, document.getElementById('root'))
```

Solution

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    // change code below this line
    return (
      <div>
        <h1>Hello React!</h1>
      </div>
    );
    // change code above this line
  }
};
```

9. Create a Component with Composition

Description

Now we will look at how we can compose multiple React components together. Imagine you are building an App and have created three components, a `Navbar`, `Dashboard`, and `Footer`. To compose these components together, you could create an `App` *parent* component which renders each of these three components as *children*. To render a component as a child in a React component, you include the component name written as a custom HTML tag in the JSX. For example, in the `render` method you could write:

```
return (
  <App>
    <Navbar />
    <Dashboard />
    <Footer />
```

```
</App>
)
```

When React encounters a custom HTML tag that references another component (a component name wrapped in `</>` like in this example), it renders the markup for that component in the location of the tag. This should illustrate the parent/child relationship between the `App` component and the `Navbar`, `Dashboard`, and `Footer`.

Instructions

In the code editor, there is a simple functional component called `ChildComponent` and a React component called `ParentComponent`. Compose the two together by rendering the `ChildComponent` within the `ParentComponent`. Make sure to close the `ChildComponent` tag with a forward slash. **Note:** `ChildComponent` is defined with an ES6 arrow function because this is a very common practice when using React. However, know that this is just a function. If you aren't familiar with the arrow function syntax, please refer to the JavaScript section.

Challenge Seed

```
const ChildComponent = () => {
  return (
    <div>
      <p>I am the child</p>
    </div>
  );
};

class ParentComponent extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h1>I am the parent</h1>
        { /* change code below this line */ }

        { /* change code above this line */ }
      </div>
    );
  }
};
```

After Test

```
ReactDOM.render(<ParentComponent />, document.getElementById('root'))
```

Solution

```
const ChildComponent = () => {
  return (
    <div>
      <p>I am the child</p>
    </div>
  );
};

class ParentComponent extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h1>I am the parent</h1>
        { /* change code below this line */ }
        <ChildComponent />
        { /* change code above this line */ }
      </div>
    );
  }
};
```

```

        </div>
    );
}
};
```

10. Use React to Render Nested Components

Description

The last challenge showed a simple way to compose two components, but there are many different ways you can compose components with React. Component composition is one of React's powerful features. When you work with React, it is important to start thinking about your user interface in terms of components like the App example in the last challenge. You break down your UI into its basic building blocks, and those pieces become the components. This helps to separate the code responsible for the UI from the code responsible for handling your application logic. It can greatly simplify the development and maintenance of complex projects.

Instructions

There are two functional components defined in the code editor, called `TypesOfFruit` and `Fruits`. Take the `TypesOfFruit` component and compose it, or *nest* it, within the `Fruits` component. Then take the `Fruits` component and nest it within the `TypesOfFood` component. The result should be a child component, nested within a parent component, which is nested within a parent component of its own!

Challenge Seed

```

const TypesOfFruit = () => {
  return (
    <div>
      <h2>Fruits:</h2>
      <ul>
        <li>Apples</li>
        <li>Blueberries</li>
        <li>Strawberries</li>
        <li>Bananas</li>
      </ul>
    </div>
  );
};

const Fruits = () => {
  return (
    <div>
      { /* change code below this line */ }

      { /* change code above this line */ }
    </div>
  );
};

class TypesOfFood extends React.Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <div>
        <h1>Types of Food:</h1>
        { /* change code below this line */ }

        { /* change code above this line */ }
      </div>
    );
  }
};
```

After Test

```
ReactDOM.render(<TypesOfFood />, document.getElementById('root'))
```

Solution

```
const TypesOfFruit = () => {
  return (
    <div>
      <h2>Fruits:</h2>
      <ul>
        <li>Apples</li>
        <li>Blueberries</li>
        <li>Strawberries</li>
        <li>Bananas</li>
      </ul>
    </div>
  );
};

const Fruits = () => {
  return (
    <div>
      { /* change code below this line */ }
      <TypesOfFruit />
      { /* change code above this line */ }
    </div>
  );
};

class TypesOfFood extends React.Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <div>
        <h1>Types of Food:</h1>
        { /* change code below this line */ }
        <Fruits />
        { /* change code above this line */ }
      </div>
    );
  }
};
```

11. Compose React Components

Description

As the challenges continue to use more complex compositions with React components and JSX, there is one important point to note. Rendering ES6 style class components within other components is no different than rendering the simple components you used in the last few challenges. You can render JSX elements, stateless functional components, and ES6 class components within other components.

Instructions

In the code editor, the `TypesOfFood` component is already rendering a component called `Vegetables`. Also, there is the `Fruits` component from the last challenge. Nest two components inside of `Fruits` — first `NonCitrus`, and then `Citrus`. Both of these components are provided for you in the background. Next, nest the `Fruits` class component into the `TypesOfFood` component, below the `h1` header and above `Vegetables`. The result should be a series of nested components, which uses two different component types.

Challenge Seed

```

class Fruits extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h2>Fruits:</h2>
        { /* change code below this line */ }

        { /* change code above this line */ }
      </div>
    );
  }
};

class TypesOfFood extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h1>Types of Food:</h1>
        { /* change code below this line */ }

        { /* change code above this line */ }
        <Vegetables />
      </div>
    );
  }
};

```

Before Test

```

class NonCitrus extends React.Component {
  render() {
    return (
      <div>
        <h4>Non-Citrus:</h4>
        <ul>
          <li>Apples</li>
          <li>Blueberries</li>
          <li>Strawberries</li>
          <li>Bananas</li>
        </ul>
      </div>
    );
  }
};

class Citrus extends React.Component {
  render() {
    return (
      <div>
        <h4>Citrus:</h4>
        <ul>
          <li>Lemon</li>
          <li>Lime</li>
          <li>Orange</li>
          <li>Grapefruit</li>
        </ul>
      </div>
    );
  }
};

class Vegetables extends React.Component {
  render() {
    return (
      <div>
        <h2>Vegetables:</h2>
        <ul>

```

```

        <li>Brussel Sprouts</li>
        <li>Broccoli</li>
        <li>Squash</li>
    </ul>
</div>
);
}
}

```

After Test

```
ReactDOM.render(<TypesOfFood />, document.getElementById('root'))
```

Solution

```

class Fruits extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h2>Fruits:</h2>
        { /* change code below this line */ }
        <NonCitrus />
        <Citrus />
        { /* change code above this line */ }
      </div>
    )
  }
}

class TypesOfFood extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h1>Types of Food:</h1>
        { /* change code below this line */ }
        <Fruits />
        { /* change code above this line */ }
        <Vegetables />
      </div>
    );
  }
}

```

12. Render a Class Component to the DOM

Description

You may remember using the ReactDOM API in an earlier challenge to render JSX elements to the DOM. The process for rendering React components will look very similar. The past few challenges focused on components and composition, so the rendering was done for you behind the scenes. However, none of the React code you write will render to the DOM without making a call to the ReactDOM API. Here's a refresher on the syntax:

`ReactDOM.render(componentToRender, targetNode)`. The first argument is the React component that you want to render. The second argument is the DOM node that you want to render that component within. React components are passed into `ReactDOM.render()` a little differently than JSX elements. For JSX elements, you pass in the name of the element that you want to render. However, for React components, you need to use the same syntax as if you were rendering a nested component, for example `ReactDOM.render(<ComponentToRender />, targetNode)`. You use this syntax for both ES6 class components and functional components.

Instructions

Both the `Fruits` and `Vegetables` components are defined for you behind the scenes. Render both components as children of the `TypesOfFood` component, then render `TypesOfFood` to the DOM. There is a `div` with `id='challenge-node'` available for you to use.

Challenge Seed

```
class TypesOfFood extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h1>Types of Food:</h1>
        {/* change code below this line */}

        {/* change code above this line */}
      </div>
    );
  }
}

// change code below this line
```

Before Test

```
const Fruits = () => {
  return (
    <div>
      <h2>Fruits:</h2>
      <h4>Non-Citrus:</h4>
      <ul>
        <li>Apples</li>
        <li>Blueberries</li>
        <li>Strawberries</li>
        <li>Bananas</li>
      </ul>
      <h4>Citrus:</h4>
      <ul>
        <li>Lemon</li>
        <li>Lime</li>
        <li>Orange</li>
        <li>Grapefruit</li>
      </ul>
    </div>
  );
};

const Vegetables = () => {
  return (
    <div>
      <h2>Vegetables:</h2>
      <ul>
        <li>Brussel Sprouts</li>
        <li>Broccoli</li>
        <li>Squash</li>
      </ul>
    </div>
  );
};
```

Solution

```
class TypesOfFood extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
```

```

        return (
      <div>
        <h1>Types of Food:</h1>
        {/* change code below this line */}
        <Fruits />
        <Vegetables />
        {/* change code above this line */}
      </div>
    );
}

// change code below this line
ReactDOM.render(<TypesOfFood />, document.getElementById('challenge-node'));

```

13. Write a React Component from Scratch

Description

Now that you've learned the basics of JSX and React components, it's time to write a component on your own. React components are the core building blocks of React applications so it's important to become very familiar with writing them. Remember, a typical React component is an ES6 class which extends `React.Component`. It has a `render` method that returns HTML (from JSX) or `null`. This is the basic form of a React component. Once you understand this well, you will be prepared to start building more complex React projects.

Instructions

Define a class `MyComponent` that extends `React.Component`. Its `render` method should return a `div` that contains an `h1` tag with the text: `My First React Component!` in it. Use this text exactly, the case and punctuation matter. Make sure to call the constructor for your component, too. Render this component to the DOM using `ReactDOM.render()`. There is a `div` with `id='challenge-node'` available for you to use.

Challenge Seed

```
// change code below this line
```

Solution

```

// change code below this line
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h1>My First React Component!</h1>
      </div>
    );
  }
}

ReactDOM.render(<MyComponent />, document.getElementById('challenge-node'));

```

14. Pass Props to a Stateless Functional Component

Description

The previous challenges covered a lot about creating and composing JSX elements, functional components, and ES6 style class components in React. With this foundation, it's time to look at another feature very common in React: **props**. In React, you can pass props, or properties, to child components. Say you have an `App` component which renders a child component called `Welcome` which is a stateless functional component. You can pass `Welcome` a `user` property by writing:

```
<App>
  <Welcome user='Mark' />
</App>
```

You use **custom HTML attributes** created by you and supported by React to be passed to the component. In this case, the created property `user` is passed to the component `Welcome`. Since `Welcome` is a stateless functional component, it has access to this value like so:

```
const Welcome = (props) => <h1>Hello, {props.user}!</h1>
```

It is standard to call this value `props` and when dealing with stateless functional components, you basically consider it as an argument to a function which returns JSX. You can access the value of the argument in the function body. With class components, you will see this is a little different.

Instructions

There are `Calendar` and `CurrentDate` components in the code editor. When rendering `CurrentDate` from the `Calendar` component, pass in a property of `date` assigned to the current date from JavaScript's `Date` object. Then access this prop in the `CurrentDate` component, showing its value within the `p` tags. Note that for prop values to be evaluated as JavaScript, they must be enclosed in curly brackets, for instance `date={Date()}`.

Challenge Seed

```
const CurrentDate = (props) => {
  return (
    <div>
      { /* change code below this line */ }
      <p>The current date is: </p>
      { /* change code above this line */ }
    </div>
  );
};

class Calendar extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h3>What date is it?</h3>
        { /* change code below this line */ }
        <CurrentDate />
        { /* change code above this line */ }
      </div>
    );
  }
};
```

After Test

```
ReactDOM.render(<Calendar />, document.getElementById('root'))
```

Solution

```
const CurrentDate = (props) => {
  return (
    <div>
      { /* change code below this line */ }
      <p>The current date is: {props.date}</p>
      { /* change code above this line */ }
    </div>
  );
};
```

```

        </div>
    );
}

class Calendar extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h3>What date is it?</h3>
        { /* change code below this line */ }
        <CurrentDate date={Date()} />
        { /* change code above this line */ }
      </div>
    );
}
}

```

15. Pass an Array as Props

Description

The last challenge demonstrated how to pass information from a parent component to a child component as `props` or properties. This challenge looks at how arrays can be passed as `props`. To pass an array to a JSX element, it must be treated as JavaScript and wrapped in curly braces.

```

<ParentComponent>
  <ChildComponent colors={"green", "blue", "red"} />
</ParentComponent>

```

The child component then has access to the array property `colors`. Array methods such as `join()` can be used when accessing the property. `const ChildComponent = (props) => <p>{props.colors.join(', ')</p>}` This will join all `colors` array items into a comma separated string and produce: `<p>green, blue, red</p>` Later, we will learn about other common methods to render arrays of data in React.

Instructions

There are `List` and `ToDo` components in the code editor. When rendering each `List` from the `ToDo` component, pass in a `tasks` property assigned to an array of to-do tasks, for example `["walk dog", "workout"]`. Then access this `tasks` array in the `List` component, showing its value within the `p` element. Use `join(", ")` to display the `props.tasks` array in the `p` element as a comma separated list. Today's list should have at least 2 tasks and tomorrow's should have at least 3 tasks.

Challenge Seed

```

const List= (props) => {
  { /* change code below this line */ }
  return <p>{}</p>
  { /* change code above this line */ }
};

class ToDo extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h1>To Do Lists</h1>
        <h2>Today</h2>
        { /* change code below this line */ }
        <List/>
        <h2>Tomorrow</h2>
        <List/>
      </div>
    );
}

```

```

    { /* change code above this line */ }
  </div>
);
}
};

```

After Test

```
ReactDOM.render(<ToDo />, document.getElementById('root'))
```

Solution

```

const List= (props) => {
  return <p>{props.tasks.join(', ')</p>
};

class ToDo extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h1>To Do Lists</h1>
        <h2>Today</h2>
        <List tasks={['study', 'exercise']} />
        <h2>Tomorrow</h2>
        <List tasks={['call Sam', 'grocery shopping', 'order tickets']} />
      </div>
    );
  }
};

```

16. Use Default Props

Description

React also has an option to set default props. You can assign default props to a component as a property on the component itself and React assigns the default prop if necessary. This allows you to specify what a prop value should be if no value is explicitly provided. For example, if you declare `MyComponent.defaultProps = { location: 'San Francisco' }`, you have defined a `location` prop that's set to the string `San Francisco`, unless you specify otherwise. React assigns default props if props are undefined, but if you pass `null` as the value for a prop, it will remain `null`.

Instructions

The code editor shows a `ShoppingCart` component. Define default props on this component which specify a prop `items` with a value of `0`.

Challenge Seed

```

const ShoppingCart = (props) => {
  return (
    <div>
      <h1>Shopping Cart Component</h1>
    </div>
  )
};
// change code below this line

```

After Test

```
ReactDOM.render(<ShoppingCart />, document.getElementById('root'))
```

Solution

```
const ShoppingCart = (props) => {
  return (
    <div>
      <h1>Shopping Cart Component</h1>
    </div>
  )
};

// change code below this line
ShoppingCart.defaultProps = {
  items: 0
}
```

17. Override Default Props

Description

The ability to set default props is a useful feature in React. The way to override the default props is to explicitly set the prop values for a component.

Instructions

The `ShoppingCart` component now renders a child component `Items`. This `Items` component has a default prop `quantity` set to the integer `0`. Override the default prop by passing in a value of `10` for `quantity`.

Note: Remember that the syntax to add a prop to a component looks similar to how you add HTML attributes.

However, since the value for `quantity` is an integer, it won't go in quotes but it should be wrapped in curly braces. For example, `{100}`. This syntax tells JSX to interpret the value within the braces directly as JavaScript.

Challenge Seed

```
const Items = (props) => {
  return <h1>Current Quantity of Items in Cart: {props.quantity}</h1>
}

Items.defaultProps = {
  quantity: 0
}

class ShoppingCart extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    { /* change code below this line */ }
    return <Items />
    { /* change code above this line */ }
  }
};
```

After Test

```
ReactDOM.render(<ShoppingCart />, document.getElementById('root'))
```

Solution

```

const Items = (props) => {
  return <h1>Current Quantity of Items in Cart: {props.quantity}</h1>
};

Items.defaultProps = {
  quantity: 0
};

class ShoppingCart extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    { /* change code below this line */ }
    return <Items quantity = {10} />
    { /* change code above this line */ }
  }
};

```

18. Use PropTypes to Define the Props You Expect

Description

React provides useful type-checking features to verify that components receive props of the correct type. For example, your application makes an API call to retrieve data that you expect to be in an array, which is then passed to a component as a prop. You can set `propTypes` on your component to require the data to be of type `array`. This will throw a useful warning when the data is of any other type. It's considered a best practice to set `propTypes` when you know the type of a prop ahead of time. You can define a `propTypes` property for a component in the same way you defined `defaultProps`. Doing this will check that props of a given key are present with a given type. Here's an example to require the type `function` for a prop called `handleClick`:

```
MyComponent.propTypes = { handleClick: PropTypes.func.isRequired }
```

In the example above, the `PropTypes.func` part checks that `handleClick` is a function. Adding `isRequired` tells React that `handleClick` is a required property for that component. You will see a warning if that prop isn't provided. Also notice that `func` represents `function`. Among the seven JavaScript primitive types, `function` and `boolean` (written as `bool`) are the only two that use unusual spelling. In addition to the primitive types, there are other types available. For example, you can check that a prop is a React element. Please refer to the [documentation](<https://reactjs.org/docs/jsx-in-depth.html#specifying-the-react-element-type>) for all of the options.

Note: As of React v15.5.0, `PropTypes` is imported independently from React, like this: `import React, { PropTypes } from 'react'`;

Instructions

Define `propTypes` for the `Items` component to require `quantity` as a prop and verify that it is of type `number`.

Challenge Seed

```

const Items = (props) => {
  return <h1>Current Quantity of Items in Cart: {props.quantity}</h1>
};

// change code below this line

// change code above this line

Items.defaultProps = {
  quantity: 0
};

class ShoppingCart extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return <Items />
  }
};

```

```

    }
};
```

Before Test

```
var PropTypes = {
  number: {isRequired: true}
};
```

After Test

```
ReactDOM.render(<ShoppingCart />, document.getElementById('root'))
```

Solution

```
const Items = (props) => {
  return <h1>Current Quantity of Items in Cart: {props.quantity}</h1>
};

// change code below this line
Items.propTypes = {
  quantity: PropTypes.number.isRequired
};
// change code above this line

Items.defaultProps = {
  quantity: 0
};

class ShoppingCart extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return <Items />
  }
};
```

19. Access Props Using this.props

Description

The last several challenges covered the basic ways to pass props to child components. But what if the child component that you're passing a prop to is an ES6 class component, rather than a stateless functional component? The ES6 class component uses a slightly different convention to access props. Anytime you refer to a class component within itself, you use the `this` keyword. To access props within a class component, you preface the code that you use to access it with `this.` For example, if an ES6 class component has a prop called `data`, you write `{this.props.data}` in JSX.

Instructions

Render an instance of the `ReturnTempPassword` component in the parent component `ResetPassword`. Here, give `ReturnTempPassword` a prop of `tempPassword` and assign it a value of a string that is at least 8 characters long. Within the child, `ReturnTempPassword`, access the `tempPassword` prop within the `strong` tags to make sure the user sees the temporary password.

Challenge Seed

```

class ReturnTempPassword extends React.Component {
  constructor(props) {
    super(props);

  }
  render() {
    return (
      <div>
        { /* change code below this line */ }
        <p>Your temporary password is: <strong></strong></p>
        { /* change code above this line */ }
      </div>
    );
  }
};

class ResetPassword extends React.Component {
  constructor(props) {
    super(props);

  }
  render() {
    return (
      <div>
        <h2>Reset Password</h2>
        <h3>We've generated a new temporary password for you.</h3>
        <h3>Please reset this password from your account settings ASAP.</h3>
        { /* change code below this line */ }

        { /* change code above this line */ }
      </div>
    );
  }
};

```

After Test

```
ReactDOM.render(<ResetPassword />, document.getElementById('root'))
```

Solution

```

class ReturnTempPassword extends React.Component {
  constructor(props) {
    super(props);

  }
  render() {
    return (
      <div>
        <p>Your temporary password is: <strong>{this.props.tempPassword}</strong></p>
      </div>
    );
  }
};

class ResetPassword extends React.Component {
  constructor(props) {
    super(props);

  }
  render() {
    return (
      <div>
        <h2>Reset Password</h2>
        <h3>We've generated a new temporary password for you.</h3>
        <h3>Please reset this password from your account settings ASAP.</h3>
        { /* change code below this line */ }
        <ReturnTempPassword tempPassword="serrPbqrPnzc" />
        { /* change code above this line */ }
      </div>
    );
  }
};

```

```

    }
};
```

20. Review Using Props with Stateless Functional Components

Description

Except for the last challenge, you've been passing props to stateless functional components. These components act like pure functions. They accept props as input and return the same view every time they are passed the same props. You may be wondering what state is, and the next challenge will cover it in more detail. Before that, here's a review of the terminology for components. A *stateless functional component* is any function you write which accepts props and returns JSX. A *stateless component*, on the other hand, is a class that extends `React.Component`, but does not use internal state (covered in the next challenge). Finally, a *stateful component* is any component that does maintain its own internal state. You may see stateful components referred to simply as components or React components. A common pattern is to try to minimize statefulness and to create stateless functional components wherever possible. This helps contain your state management to a specific area of your application. In turn, this improves development and maintenance of your app by making it easier to follow how changes to state affect its behavior.

Instructions

The code editor has a `CampSite` component that renders a `Camper` component as a child. Define the `Camper` component and assign it default props of `{ name: 'CamperBot' }`. Inside the `Camper` component, render any code that you want, but make sure to have one `p` element that includes only the `name` value that is passed in as a prop. Finally, define `propTypes` on the `Camper` component to require `name` to be provided as a prop and verify that it is of type `string`.

Challenge Seed

```

class CampSite extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <Camper/>
      </div>
    );
  }
}
// change code below this line
```

Before Test

```

var PropTypes = {
  string: {isRequired: true }
};
```

After Test

```
ReactDOM.render(<CampSite />, document.getElementById('root'))
```

Solution

```

class CampSite extends React.Component {
  constructor(props) {
```

```

    super(props);
}
render() {
  return (
    <div>
      <Camper/>
    </div>
  );
}
// change code below this line

const Camper = (props) => {
  return (
    <div>
      <p>{props.name}</p>
    </div>
  );
};

Camper.propTypes = {
  name: PropTypes.string.isRequired
};

Camper.defaultProps = {
  name: 'CamperBot'
};

```

21. Create a Stateful Component

Description

One of the most important topics in React is `state`. State consists of any data your application needs to know about, that can change over time. You want your apps to respond to state changes and present an updated UI when necessary. React offers a nice solution for the state management of modern web applications. You create state in a React component by declaring a `state` property on the component class in its `constructor`. This initializes the component with `state` when it is created. The `state` property must be set to a JavaScript object. Declaring it looks like this:

```

this.state = {
  // describe your state here
} You have access to the state object throughout the life of your component. You can update it, render it in your UI, and pass it as props to child components. The state object can be as complex or as simple as you need it to be. Note that you must create a class component by extending React.Component in order to create state like this.

```

Instructions

There is a component in the code editor that is trying to render a `name` property from its `state`. However, there is no `state` defined. Initialize the component with `state` in the `constructor` and assign your name to a property of `name`.

Challenge Seed

```

class StatefulComponent extends React.Component {
  constructor(props) {
    super(props);
    // initialize state here
  }
  render() {
    return (
      <div>
        <h1>{this.state.name}</h1>
      </div>
    );
  }
}

```

```

    }
};
```

After Test

```
ReactDOM.render(<StatefulComponent />, document.getElementById('root'))
```

Solution

```

class StatefulComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'freeCodeCamp!'
    }
  }
  render() {
    return (
      <div>
        <h1>{this.state.name}</h1>
      </div>
    );
  }
}
```

22. Render State in the User Interface

Description

Once you define a component's initial state, you can display any part of it in the UI that is rendered. If a component is stateful, it will always have access to the data in `state` in its `render()` method. You can access the data with `this.state`. If you want to access a state value within the `return` of the `render` method, you have to enclose the value in curly braces. `State` is one of the most powerful features of components in React. It allows you to track important data in your app and render a UI in response to changes in this data. If your data changes, your UI will change. React uses what is called a virtual DOM, to keep track of changes behind the scenes. When state data updates, it triggers a re-render of the components using that data - including child components that received the data as a prop. React updates the actual DOM, but only where necessary. This means you don't have to worry about changing the DOM. You simply declare what the UI should look like. Note that if you make a component stateful, no other components are aware of its `state`. Its `state` is completely encapsulated, or local to that component, unless you pass state data to a child component as `props`. This notion of encapsulated `state` is very important because it allows you to write certain logic, then have that logic contained and isolated in one place in your code.

Instructions

In the code editor, `MyComponent` is already stateful. Define an `h1` tag in the component's `render` method which renders the value of `name` from the component's `state`. **Note:** The `h1` should only render the value from `state` and nothing else. In JSX, any code you write with curly braces `{ }` will be treated as JavaScript. So to access the value from `state` just enclose the reference in curly braces.

Challenge Seed

```

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'freeCodeCamp'
    }
  }
  render() {
    return (

```

```

<div>
  { /* change code below this line */ }

  { /* change code above this line */ }
</div>
);
}
};

```

After Test

```
ReactDOM.render(<MyComponent />, document.getElementById('root'))
```

Solution

```

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'freeCodeCamp'
    }
  }
  render() {
    return (
      <div>
        { /* change code below this line */ }
        <h1>{this.state.name}</h1>
        { /* change code above this line */ }
      </div>
    );
  }
};

```

23. Render State in the User Interface Another Way

Description

There is another way to access `state` in a component. In the `render()` method, before the `return` statement, you can write JavaScript directly. For example, you could declare functions, access data from `state` or `props`, perform computations on this data, and so on. Then, you can assign any data to variables, which you have access to in the `return` statement.

Instructions

In the `MyComponent` render method, define a `const` called `name` and set it equal to the `name` value in the component's `state`. Because you can write JavaScript directly in this part of the code, you don't have to enclose this reference in curly braces. Next, in the `return` statement, render this value in an `h1` tag using the variable `name`. Remember, you need to use the JSX syntax (curly braces for JavaScript) in the `return` statement.

Challenge Seed

```

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'freeCodeCamp'
    }
  }
  render() {
    // change code below this line

    // change code above this line
  }
};

```

```

    return (
      <div>
        { /* change code below this line */ }

        { /* change code above this line */ }
      </div>
    );
}
);

```

After Test

```
ReactDOM.render(<MyComponent />, document.getElementById('root'))
```

Solution

```

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'freeCodeCamp'
    }
  }
  render() {
    // change code below this line
    const name = this.state.name;
    // change code above this line
    return (
      <div>
        { /* change code below this line */ }
        <h1>{name}</h1>
        { /* change code above this line */ }
      </div>
    );
  }
}

```

24. Set State with this.setState

Description

The previous challenges covered component state and how to initialize state in the `constructor`. There is also a way to change the component's state. React provides a method for updating component state called `setState`. You call the `setState` method within your component class like so: `this.setState()`, passing in an object with key-value pairs. The keys are your state properties and the values are the updated state data. For instance, if we were storing a `username` in state and wanted to update it, it would look like this:

```

this.setState({
  username: 'Lewis'
});

```

React expects you to never modify state directly, instead always use `this.setState()` when state changes occur. Also, you should note that React may batch multiple state updates in order to improve performance. What this means is that state updates through the `setState` method can be asynchronous. There is an alternative syntax for the `setState` method which provides a way around this problem. This is rarely needed but it's good to keep it in mind! Please consult the [React documentation](#) for further details.

Instructions

There is a `button` element in the code editor which has an `onClick()` handler. This handler is triggered when the `button` receives a click event in the browser, and runs the `handleClick` method defined on `MyComponent`. Within the `handleClick` method, update the component state using `this.setState()`. Set the `name` property in state to

equal the string `React Rocks!`. Click the button and watch the rendered state update. Don't worry if you don't fully understand how the click handler code works at this point. It's covered in upcoming challenges.

Challenge Seed

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'Initial State'
    };
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    // change code below this line

    // change code above this line
  }
  render() {
    return (
      <div>
        <button onClick={this.handleClick}>Click Me</button>
        <h1>{this.state.name}</h1>
      </div>
    );
  }
};
```

After Test

```
ReactDOM.render(<MyComponent />, document.getElementById('root'))
```

Solution

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'Initial State'
    };
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    // change code below this line
    this.setState({
      name: 'React Rocks!'
    });
    // change code above this line
  }
  render() {
    return (
      <div>
        <button onClick = {this.handleClick}>Click Me</button>
        <h1>{this.state.name}</h1>
      </div>
    );
  }
};
```

25. Bind 'this' to a Class Method

Description

In addition to setting and updating `state`, you can also define methods for your component class. A class method typically needs to use the `this` keyword so it can access properties on the class (such as `state` and `props`) inside the scope of the method. There are a few ways to allow your class methods to access `this`. One common way is to explicitly bind `this` in the constructor so `this` becomes bound to the class methods when the component is initialized. You may have noticed the last challenge used `this.handleClick = this.handleClick.bind(this)` for its `handleClick` method in the constructor. Then, when you call a function like `this.setState()` within your class method, `this` refers to the class and will not be `undefined`. **Note:** The `this` keyword is one of the most confusing aspects of JavaScript but it plays an important role in React. Although its behavior here is totally normal, these lessons aren't the place for an in-depth review of `this` so please refer to other lessons if the above is confusing!

Instructions

The code editor has a component with a `state` that keeps track of an item count. It also has a method which allows you to increment this item count. However, the method doesn't work because it's using the `this` keyword that is `undefined`. Fix it by explicitly binding `this` to the `addItem()` method in the component's constructor. Next, add a click handler to the `button` element in the `render` method. It should trigger the `addItem()` method when the button receives a click event. Remember that the method you pass to the `onClick` handler needs curly braces because it should be interpreted directly as JavaScript. Once you complete the above steps you should be able to click the button and see the item count increment in the HTML.

Challenge Seed

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      itemCount: 0
    };
    // change code below this line

    // change code above this line
  }
  addItem() {
    this.setState({
      itemCount: this.state.itemCount + 1
    });
  }
  render() {
    return (
      <div>
        { /* change code below this line */ }
        <button>Click Me</button>
        { /* change code above this line */ }
        <h1>Current Item Count: {this.state.itemCount}</h1>
      </div>
    );
  }
}
```

After Test

```
ReactDOM.render(<MyComponent />, document.getElementById('root'))
```

Solution

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      itemCount: 0
    };
    this.addItem = this.addItem.bind(this);
  }
  addItem() {
    this.setState({
```

```

        itemCount: this.state.itemCount + 1
    });
}

render() {
    return (
        <div>
            <button onClick = {this.addItem}>Click Me</button>
            <h1>Current Item Count: {this.state.itemCount}</h1>
        </div>
    );
}
};

```

26. Use State to Toggle an Element

Description

You can use `state` in React applications in more complex ways than what you've seen so far. One example is to monitor the status of a value, then render the UI conditionally based on this value. There are several different ways to accomplish this, and the code editor shows one method.

Instructions

`MyComponent` has a `visibility` property which is initialized to `false`. The `render` method returns one view if the value of `visibility` is true, and a different view if it is false. Currently, there is no way of updating the `visibility` property in the component's `state`. The value should toggle back and forth between true and false. There is a click handler on the button which triggers a class method called `toggleVisibility()`. Define this method so the `state` of `visibility` toggles to the opposite value when the method is called. If `visibility` is `false`, the method sets it to `true`, and vice versa. Finally, click the button to see the conditional rendering of the component based on its `state`.

Hint: Don't forget to bind the `this` keyword to the method in the constructor!

Challenge Seed

```

class MyComponent extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            visibility: false
        };
        // change code below this line

        // change code above this line
    }
    // change code below this line

    // change code above this line
    render() {
        if (this.state.visibility) {
            return (
                <div>
                    <button onClick={this.toggleVisibility}>Click Me</button>
                    <h1>Now you see me!</h1>
                </div>
            );
        } else {
            return (
                <div>
                    <button onClick={this.toggleVisibility}>Click Me</button>
                </div>
            );
        }
    }
};

```

After Test

```
ReactDOM.render(<MyComponent />, document.getElementById('root'))
```

Solution

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      visibility: false
    };
    this.toggleVisibility = this.toggleVisibility.bind(this);
  }
  toggleVisibility() {
    this.setState({
      visibility: !this.state.visibility
    });
  }
  render() {
    if (this.state.visibility) {
      return (
        <div>
          <button onClick = {this.toggleVisibility}>Click Me</button>
          <h1>Now you see me!</h1>
        </div>
      );
    } else {
      return (
        <div>
          <button onClick = {this.toggleVisibility}>Click Me</button>
        </div>
      );
    }
  }
};
```

27. Write a Simple Counter

Description

You can design a more complex stateful component by combining the concepts covered so far. These include initializing `state`, writing methods that set `state`, and assigning click handlers to trigger these methods.

Instructions

The `Counter` component keeps track of a `count` value in `state`. There are two buttons which call methods `increment()` and `decrement()`. Write these methods so the counter value is incremented or decremented by 1 when the appropriate button is clicked. Also, create a `reset()` method so when the reset button is clicked, the count is set to 0. **Note:** Make sure you don't modify the `classNames` of the buttons. Also, remember to add the necessary bindings for the newly-created methods in the constructor.

Challenge Seed

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
    // change code below this line

    // change code above this line
  }
  // change code below this line
```

```
// change code above this line
render() {
  return (
    <div>
      <button className='inc' onClick={this.increment}>Increment!</button>
      <button className='dec' onClick={this.decrement}>Decrement!</button>
      <button className='reset' onClick={this.reset}>Reset</button>
      <h1>Current Count: {this.state.count}</h1>
    </div>
  );
}
};
```

After Test

```
ReactDOM.render(<Counter />, document.getElementById('root'))
```

Solution

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
    this.increment = this.increment.bind(this);
    this.decrement = this.decrement.bind(this);
    this.reset = this.reset.bind(this);
  }
  reset() {
    this.setState({
      count: 0
    });
  }
  increment() {
    this.setState({
      count: this.state.count + 1
    });
  }
  decrement() {
    this.setState({
      count: this.state.count - 1
    });
  }
  render() {
    return (
      <div>
        <button className='inc' onClick={this.increment}>Increment!</button>
        <button className='dec' onClick={this.decrement}>Decrement!</button>
        <button className='reset' onClick={this.reset}>Reset</button>
        <h1>Current Count: {this.state.count}</h1>
      </div>
    );
  }
};
```

28. Create a Controlled Input

Description

Your application may have more complex interactions between `state` and the rendered UI. For example, form control elements for text input, such as `input` and `textarea`, maintain their own state in the DOM as the user types. With React, you can move this mutable state into a React component's `state`. The user's input becomes part of the application `state`, so React controls the value of that input field. Typically, if you have React components with input fields the user can type into, it will be a controlled input form.

Instructions

The code editor has the skeleton of a component called `ControlledInput` to create a controlled `input` element. The component's state is already initialized with an `input` property that holds an empty string. This value represents the text a user types into the `input` field. First, create a method called `handleChange()` that has a parameter called `event`. When the method is called, it receives an `event` object that contains a string of text from the `input` element. You can access this string with `event.target.value` inside the method. Update the `input` property of the component's state with this new string. In the render method, create the `input` element above the `h4` tag. Add a `value` attribute which is equal to the `input` property of the component's state. Then add an `onChange()` event handler set to the `handleChange()` method. When you type in the input box, that text is processed by the `handleChange()` method, set as the `input` property in the local state, and rendered as the value in the `input` box on the page. The component state is the single source of truth regarding the input data. Last but not least, don't forget to add the necessary bindings in the constructor.

Challenge Seed

```
class ControlledInput extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      input: ''
    };
    // change code below this line

    // change code above this line
  }
  // change code below this line

  // change code above this line
  render() {
    return (
      <div>
        { /* change code below this line */

          { /* change code above this line */}
          <h4>Controlled Input:</h4>
          <p>{this.state.input}</p>
        </div>
      );
    }
  };
}
```

After Test

```
ReactDOM.render(<ControlledInput />, document.getElementById('root'))
```

Solution

```
class ControlledInput extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      input: ''
    };
    this.handleChange = this.handleChange.bind(this);
  }
  handleChange(event) {
    this.setState({
      input: event.target.value
    });
  }
  render() {
    return (
      <div>
        <input
          value={this.state.input}
          onChange={this.handleChange} />
      </div>
    );
  }
}
```

```

<h4>Controlled Input:</h4>

<p>{this.state.input}</p>
</div>
);
}
);

```

29. Create a Controlled Form

Description

The last challenge showed that React can control the internal state for certain elements like `input` and `textarea`, which makes them controlled components. This applies to other form elements as well, including the regular HTML `form` element.

Instructions

The `MyForm` component is set up with an empty `form` with a submit handler. The submit handler will be called when the form is submitted. We've added a button which submits the form. You can see it has the `type` set to `submit` indicating it is the button controlling the form. Add the `input` element in the `form` and set its `value` and `onChange()` attributes like the last challenge. You should then complete the `handleSubmit` method so that it sets the component state property `submit` to the current input value in the local `state`. **Note:** You also must call `event.preventDefault()` in the submit handler, to prevent the default form submit behavior which will refresh the web page. Finally, create an `h1` tag after the `form` which renders the `submit` value from the component's `state`. You can then type in the form and click the button (or press enter), and you should see your input rendered to the page.

Challenge Seed

```

class MyForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      input: '',
      submit: ''
    };
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }
  handleChange(event) {
    this.setState({
      input: event.target.value
    });
  }
  handleSubmit(event) {
    // change code below this line

    // change code above this line
  }
  render() {
    return (
      <div>
        <form onSubmit={this.handleSubmit}>
          { /* change code below this line */ }

          { /* change code above this line */ }
          <button type='submit'>Submit!</button>
        </form>
        { /* change code below this line */ }

        { /* change code above this line */ }
      </div>
    );
  }
};

```

After Test

```
ReactDOM.render(<MyForm />, document.getElementById('root'))
```

Solution

```
class MyForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      input: '',
      submit: ''
    };
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }
  handleChange(event) {
    this.setState({
      input: event.target.value
    });
  }
  handleSubmit(event) {
    event.preventDefault()
    this.setState({
      submit: this.state.input
    });
  }
  render() {
    return (
      <div>
        <form onSubmit={this.handleSubmit}>
          <input
            value={this.state.input}
            onChange={this.handleChange} />
          <button type='submit'>Submit!</button>
        </form>
        <h1>{this.state.submit}</h1>
      </div>
    );
  }
}
```

30. Pass State as Props to Child Components

Description

You saw a lot of examples that passed props to child JSX elements and child React components in previous challenges. You may be wondering where those props come from. A common pattern is to have a stateful component containing the `state` important to your app, that then renders child components. You want these components to have access to some pieces of that `state`, which are passed in as props. For example, maybe you have an `App` component that renders a `Navbar`, among other components. In your `App`, you have `state` that contains a lot of user information, but the `Navbar` only needs access to the user's username so it can display it. You pass that piece of `state` to the `Navbar` component as a prop. This pattern illustrates some important paradigms in React. The first is *unidirectional data flow*. State flows in one direction down the tree of your application's components, from the stateful parent component to child components. The child components only receive the state data they need. The second is that complex stateful apps can be broken down into just a few, or maybe a single, stateful component. The rest of your components simply receive state from the parent as props, and render a UI from that state. It begins to create a separation where state management is handled in one part of code and UI rendering in another. This principle of separating state logic from UI logic is one of React's key principles. When it's used correctly, it makes the design of complex, stateful applications much easier to manage.

Instructions

The `MyApp` component is stateful and renders a `Navbar` component as a child. Pass the `name` property in its `state` down to the child component, then show the `name` in the `h1` tag that's part of the `Navbar` `render` method.

Challenge Seed

```
class MyApp extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'CamperBot'
    }
  }
  render() {
    return (
      <div>
        <Navbar /* your code here */ />
      </div>
    );
  }
}

class Navbar extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h1>Hello, my name is: {/* your code here */} </h1>
      </div>
    );
  }
}
```

After Test

```
ReactDOM.render(<MyApp />, document.getElementById('root'))
```

Solution

```
class MyApp extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'CamperBot'
    }
  }
  render() {
    return (
      <div>
        <Navbar name={this.state.name}/>
      </div>
    );
  }
}

class Navbar extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h1>Hello, my name is: {this.props.name}</h1>
      </div>
    );
  }
}
```

31. Pass a Callback as Props

Description

You can pass `state` as props to child components, but you're not limited to passing data. You can also pass handler functions or any method that's defined on a React component to a child component. This is how you allow child components to interact with their parent components. You pass methods to a child just like a regular prop. It's assigned a name and you have access to that method name under `this.props` in the child component.

Instructions

There are three components outlined in the code editor. The `MyApp` component is the parent that will render the `GetInput` and `RenderInput` child components. Add the `GetInput` component to the render method in `MyApp`, then pass it a prop called `input` assigned to `inputValue` from `MyApp`'s `state`. Also create a prop called `handleChange` and pass the input handler `handleChange` to it. Next, add `RenderInput` to the render method in `MyApp`, then create a prop called `input` and pass the `inputValue` from `state` to it. Once you are finished you will be able to type in the `input` field in the `GetInput` component, which then calls the handler method in its parent via props. This updates the `input` in the `state` of the parent, which is passed as props to both children. Observe how the data flows between the components and how the single source of truth remains the `state` of the parent component. Admittedly, this example is a bit contrived, but should serve to illustrate how data and callbacks can be passed between React components.

Challenge Seed

```
class MyApp extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      inputValue: ''
    }
    this.handleChange = this.handleChange.bind(this);
  }
  handleChange(event) {
    this.setState({
      inputValue: event.target.value
    });
  }
  render() {
    return (
      <div>
        { /* change code below this line */ }

        { /* change code above this line */ }
      </div>
    );
  }
}

class GetInput extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h3>Get Input:</h3>
        <input
          value={this.props.input}
          onChange={this.props.handleChange}/>
      </div>
    );
  }
}

class RenderInput extends React.Component {
```

```

constructor(props) {
  super(props);
}
render() {
  return (
    <div>
      <h3>Input Render:</h3>
      <p>{this.props.input}</p>
    </div>
  );
}

```

After Test

```
ReactDOM.render(<MyApp />, document.getElementById('root'))
```

Solution

```

class MyApp extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      inputValue: ''
    }
    this.handleChange = this.handleChange.bind(this);
  }
  handleChange(event) {
    this.setState({
      inputValue: event.target.value
    });
  }
  render() {
    return (
      <div>
        <GetInput
          input={this.state.inputValue}
          handleChange={this.handleChange}>
        <RenderInput
          input={this.state.inputValue}>
      </div>
    );
  }
}

class GetInput extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h3>Get Input:</h3>
        <input
          value={this.props.input}
          onChange={this.props.handleChange}>
      </div>
    );
  }
}

class RenderInput extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h3>Input Render:</h3>
        <p>{this.props.input}</p>
      </div>
    );
  }
}

```

```

    }
};
```

32. Use the Lifecycle Method componentWillMount

Description

React components have several special methods that provide opportunities to perform actions at specific points in the lifecycle of a component. These are called lifecycle methods, or lifecycle hooks, and allow you to catch components at certain points in time. This can be before they are rendered, before they update, before they receive props, before they unmount, and so on. Here is a list of some of the main lifecycle methods: `componentWillMount()`
`componentDidMount()` `componentWillReceiveProps()` `shouldComponentUpdate()` `componentWillUpdate()`
`componentDidUpdate()` `componentWillUnmount()` The next several lessons will cover some of the basic use cases for these lifecycle methods.

Note: The `componentWillMount` Lifecycle method will be deprecated in a future version of 16.X and removed in version 17. ([Source](#))

Instructions

The `componentWillMount()` method is called before the `render()` method when a component is being mounted to the DOM. Log something to the console within `componentWillMount()` - you may want to have your browser console open to see the output.

Challenge Seed

```

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
  }
  componentWillMount() {
    // change code below this line

    // change code above this line
  }
  render() {
    return <div />
  }
};
```

After Test

```
ReactDOM.render(<MyComponent />, document.getElementById('root'))
```

Solution

```

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
  }
  componentWillMount() {
    // change code below this line
    console.log('Component is mounting...');
    // change code above this line
  }
  render() {
    return <div />
  }
};
```

33. Use the Lifecycle Method `componentDidMount`

Description

Most web developers, at some point, need to call an API endpoint to retrieve data. If you're working with React, it's important to know where to perform this action. The best practice with React is to place API calls or any calls to your server in the lifecycle method `componentDidMount()`. This method is called after a component is mounted to the DOM. Any calls to `setState()` here will trigger a re-rendering of your component. When you call an API in this method, and set your state with the data that the API returns, it will automatically trigger an update once you receive the data.

Instructions

There is a mock API call in `componentDidMount()`. It sets state after 2.5 seconds to simulate calling a server to retrieve data. This example requests the current total active users for a site. In the render method, render the value of `activeUsers` in the `h1`. Watch what happens in the preview, and feel free to change the timeout to see the different effects.

Challenge Seed

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      activeUsers: null
    };
  }
  componentDidMount() {
    setTimeout( () => {
      this.setState({
        activeUsers: 1273
      });
    }, 2500);
  }
  render() {
    return (
      <div>
        <h1>Active Users: { /* change code here */ }</h1>
      </div>
    );
  }
}
```

After Test

```
ReactDOM.render(<MyComponent />, document.getElementById('root'))
```

Solution

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      activeUsers: null
    };
  }
  componentDidMount() {
    setTimeout( () => {
      this.setState({
        activeUsers: 1273
      });
    }, 2500);
  }
  render() {
    return (

```

```

        <div>
          <h1>Active Users: {this.state.activeUsers}</h1>
        </div>
      );
    }
  };

```

34. Add Event Listeners

Description

The `componentDidMount()` method is also the best place to attach any event listeners you need to add for specific functionality. React provides a synthetic event system which wraps the native event system present in browsers. This means that the synthetic event system behaves exactly the same regardless of the user's browser - even if the native events may behave differently between different browsers. You've already been using some of these synthetic event handlers such as `onClick()`. React's synthetic event system is great to use for most interactions you'll manage on DOM elements. However, if you want to attach an event handler to the document or window objects, you have to do this directly.

Instructions

Attach an event listener in the `componentDidMount()` method for `keydown` events and have these events trigger the callback `handleKeyPress()`. You can use `document.addEventListener()` which takes the event (in quotes) as the first argument and the callback as the second argument. Then, in `componentWillUnmount()`, remove this same event listener. You can pass the same arguments to `document.removeEventListener()`. It's good practice to use this lifecycle method to do any clean up on React components before they are unmounted and destroyed. Removing event listeners is an example of one such clean up action.

Challenge Seed

```

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      message: ''
    };
    this.handleEnter = this.handleEnter.bind(this);
    this.handleKeyPress = this.handleKeyPress.bind(this);
  }
  // change code below this line
  componentDidMount() {

  }
  componentWillUnmount() {

  }
  // change code above this line
  handleEnter() {
    this.setState({
      message: this.state.message + 'You pressed the enter key! '
    });
  }
  handleKeyPress(event) {
    if (event.keyCode === 13) {
      this.handleEnter();
    }
  }
  render() {
    return (
      <div>
        <h1>{this.state.message}</h1>
      </div>
    );
  }
}

```

After Test

```
ReactDOM.render(<MyComponent />, document.getElementById('root'))
```

Solution

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      message: ''
    };
    this.handleKeyPress = this.handleKeyPress.bind(this);
    this.handleEnter = this.handleEnter.bind(this);
  }
  componentDidMount() {
    // change code below this line
    document.addEventListener('keydown', this.handleKeyPress);
    // change code above this line
  }
  componentWillUnmount() {
    // change code below this line
    document.removeEventListener('keydown', this.handleKeyPress);
    // change code above this line
  }
  handleEnter() {
    this.setState({
      message: this.state.message + 'You pressed the enter key! '
    });
  }
  handleKeyPress(event) {
    if (event.keyCode === 13) {
      this.handleEnter();
    }
  }
  render() {
    return (
      <div>
        <h1>{this.state.message}</h1>
      </div>
    );
  }
};
```

35. Manage Updates with Lifecycle Methods

Description

Another lifecycle method is `componentWillReceiveProps()` which is called whenever a component is receiving new props. This method receives the new props as an argument, which is usually written as `nextProps`. You can use this argument and compare with `this.props` and perform actions before the component updates. For example, you may call `setState()` locally before the update is processed. Another method is `componentDidUpdate()`, and is called immediately after a component re-renders. Note that rendering and mounting are considered different things in the component lifecycle. When a page first loads, all components are mounted and this is where methods like `componentWillMount()` and `componentDidMount()` are called. After this, as state changes, components re-render themselves. The next challenge covers this in more detail.

Instructions

The child component `Dialog` receives `message` props from its parent, the `Controller` component. Write the `componentWillReceiveProps()` method in the `Dialog` component and have it log `this.props` and `nextProps` to the console. You'll need to pass `nextProps` as an argument to this method and although it's possible to name it anything, name it `nextProps` here. Next, add `componentDidUpdate()` in the `Dialog` component, and log a statement that says

the component has updated. This method works similar to `componentWillUpdate()`, which is provided for you. Now click the button to change the message and watch your browser console. The order of the console statements show the order the methods are called. **Note:** You'll need to write the lifecycle methods as normal functions and not as arrow functions to pass the tests (there is also no advantage to writing lifecycle methods as arrow functions).

Challenge Seed

```
class Dialog extends React.Component {
  constructor(props) {
    super(props);
  }
  componentWillMount() {
    console.log('Component is about to update...');
  }
  // change code below this line

  // change code above this line
  render() {
    return <h1>{this.props.message}</h1>
  }
};

class Controller extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      message: 'First Message'
    };
    this.changeMessage = this.changeMessage.bind(this);
  }
  changeMessage() {
    this.setState({
      message: 'Second Message'
    });
  }
  render() {
    return (
      <div>
        <button onClick={this.changeMessage}>Update</button>
        <Dialog message={this.state.message}/>
      </div>
    );
  }
};
```

After Test

```
ReactDOM.render(<Controller />, document.getElementById('root'))
```

Solution

```
class Dialog extends React.Component {
  constructor(props) {
    super(props);
  }
  componentWillMount() {
    console.log('Component is about to update...');
  }
  // change code below this line
  componentWillReceiveProps(nextProps) {
    console.log(this.props, nextProps);
  }
  componentDidUpdate() {
    console.log('Component re-rendered');
  }
  // change code above this line
  render() {
    return <h1>{this.props.message}</h1>
  }
};
```

```

};

class Controller extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      message: 'First Message'
    };
  this.changeMessage = this.changeMessage.bind(this);
}
changeMessage() {
  this.setState({
    message: 'Second Message'
  });
}
render() {
  return (
    <div>
      <button onClick={this.changeMessage}>Update</button>
      <Dialog message={this.state.message}/>
    </div>
  );
}
}

```

36. Optimize Re-Renders with shouldComponentUpdate

Description

So far, if any component receives new `state` or new `props`, it re-renders itself and all its children. This is usually okay. But React provides a lifecycle method you can call when child components receive new `state` or `props`, and declare specifically if the components should update or not. The method is `shouldComponentUpdate()`, and it takes `nextProps` and `nextState` as parameters. This method is a useful way to optimize performance. For example, the default behavior is that your component re-renders when it receives new `props`, even if the `props` haven't changed. You can use `shouldComponentUpdate()` to prevent this by comparing the `props`. The method must return a boolean value that tells React whether or not to update the component. You can compare the current `props` (`this.props`) to the next `props` (`nextProps`) to determine if you need to update or not, and return `true` or `false` accordingly.

Instructions

The `shouldComponentUpdate()` method is added in a component called `OnlyEvens`. Currently, this method returns `true` so `OnlyEvens` re-renders every time it receives new `props`. Modify the method so `OnlyEvens` updates only if the value of its new `props` is even. Click the `Add` button and watch the order of events in your browser's console as the other lifecycle hooks are triggered.

Challenge Seed

```

class OnlyEvens extends React.Component {
  constructor(props) {
    super(props);
  }
  shouldComponentUpdate(nextProps, nextState) {
    console.log('Should I update?');
    // change code below this line
    return true;
    // change code above this line
  }
  componentWillReceiveProps(nextProps) {
    console.log('Receiving new props...');

  }
  componentDidUpdate() {
    console.log('Component re-rendered.');
  }
  render() {
    return <h1>{this.props.value}</h1>
  }
}

```

```

};

class Controller extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: 0
    };
    this.addValue = this.addValue.bind(this);
  }
  addValue() {
    this.setState({
      value: this.state.value + 1
    });
  }
  render() {
    return (
      <div>
        <button onClick={this.addValue}>Add</button>
        <OnlyEvens value={this.state.value}>/>
      </div>
    );
  }
}

```

After Test

```
ReactDOM.render(<Controller />, document.getElementById('root'))
```

Solution

```

class OnlyEvens extends React.Component {
  constructor(props) {
    super(props);
  }
  shouldComponentUpdate(nextProps, nextState) {
    console.log('Should I update?');
    // change code below this line
    return nextProps.value % 2 === 0;
    // change code above this line
  }
  componentWillReceiveProps(nextProps) {
    console.log('Receiving new props...');

  }
  componentDidUpdate() {
    console.log('Component re-rendered.');
  }
  render() {
    return <h1>{this.props.value}</h1>
  }
}

class Controller extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: 0
    };
    this.addValue = this.addValue.bind(this);
  }
  addValue() {
    this.setState({
      value: this.state.value + 1
    });
  }
  render() {
    return (
      <div>
        <button onClick={this.addValue}>Add</button>
        <OnlyEvens value={this.state.value}>/>
      </div>
    );
  }
}

```

```

    }
};
```

37. Introducing Inline Styles

Description

There are other complex concepts that add powerful capabilities to your React code. But you may be wondering about the more simple problem of how to style those JSX elements you create in React. You likely know that it won't be exactly the same as working with HTML because of [the way you apply classes to JSX elements](#). If you import styles from a stylesheet, it isn't much different at all. You apply a class to your JSX element using the `className` attribute, and apply styles to the class in your stylesheet. Another option is to apply *inline* styles, which are very common in ReactJS development. You apply inline styles to JSX elements similar to how you do it in HTML, but with a few JSX differences. Here's an example of an inline style in HTML: `<div style="color: yellow; font-size: 16px">Mellow Yellow</div>` JSX elements use the `style` attribute, but because of the way JSX is transpiled, you can't set the value to a `string`. Instead, you set it equal to a JavaScript `object`. Here's an example: `<div style={{color: "yellow", fontSize: 16}}>Mellow Yellow</div>` Notice how we camelCase the "fontSize" property? This is because React will not accept kebab-case keys in the style object. React will apply the correct property name for us in the HTML.

Instructions

Add a `style` attribute to the `div` in the code editor to give the text a color of red and font size of 72px. Note that you can optionally set the font size to be a number, omitting the units "px", or write it as "72px".

Challenge Seed

```

class Colorful extends React.Component {
  render() {
    return (
      <div>Big Red</div>
    );
  }
};
```

After Test

```
ReactDOM.render(<Colorful />, document.getElementById('root'))
```

Solution

```

class Colorful extends React.Component {
  render() {
    return (
      <div style={{color: "red", fontSize: 72}}>Big Red</div>
    );
  }
};
```

38. Add Inline Styles in React

Description

You may have noticed in the last challenge that there were several other syntax differences from HTML inline styles in addition to the `style` attribute set to a JavaScript object. First, the names of certain CSS style properties use camel

case. For example, the last challenge set the size of the font with `fontSize` instead of `font-size`. Hyphenated words like `font-size` are invalid syntax for JavaScript object properties, so React uses camel case. As a rule, any hyphenated style properties are written using camel case in JSX. All property value length units (like `height`, `width`, and `fontSize`) are assumed to be in `px` unless otherwise specified. If you want to use `em`, for example, you wrap the value and the units in quotes, like `{fontSize: "4em"}`. Other than the length values that default to `px`, all other property values should be wrapped in quotes.

Instructions

If you have a large set of styles, you can assign a `style` object to a constant to keep your code organized. Uncomment the `styles` constant and declare an `object` with three style properties and their values. Give the `div` a color of "purple", a font-size of `40`, and a border of "2px solid purple". Then set the `style` attribute equal to the `styles` constant.

Challenge Seed

```
// const styles =
// change code above this line
class Colorful extends React.Component {
  render() {
    // change code below this line
    return (
      <div style={{color: "yellow", fontSize: 24}}>Style Me!</div>
    );
    // change code above this line
  }
};
```

After Test

```
ReactDOM.render(<Colorful />, document.getElementById('root'))
```

Solution

```
const styles = {
  color: "purple",
  fontSize: 40,
  border: "2px solid purple"
};
// change code above this line
class Colorful extends React.Component {
  render() {
    // change code below this line
    return (
      <div style={styles}>Style Me!</div>
    );
    // change code above this line
  }
};
```

39. Use Advanced JavaScript in React Render Method

Description

In previous challenges, you learned how to inject JavaScript code into JSX code using curly braces, `{ }`, for tasks like accessing props, passing props, accessing state, inserting comments into your code, and most recently, styling your components. These are all common use cases to put JavaScript in JSX, but they aren't the only way that you can utilize JavaScript code in your React components. You can also write JavaScript directly in your `render` methods, before the `return` statement, **without** inserting it inside of curly braces. This is because it is not yet within the JSX code. When

you want to use a variable later in the JSX code *inside* the `return` statement, you place the variable name inside curly braces.

Instructions

In the code provided, the `render` method has an array that contains 20 phrases to represent the answers found in the classic 1980's Magic Eight Ball toy. The button click event is bound to the `ask` method, so each time the button is clicked a random number will be generated and stored as the `randomIndex` in state. On line 52, delete the string "change me!" and reassign the `answer` const so your code randomly accesses a different index of the `possibleAnswers` array each time the component updates. Finally, insert the `answer` const inside the `p` tags.

Challenge Seed

```
const inputStyle = {
  width: 235,
  margin: 5
}

class MagicEightBall extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      userInput: '',
      randomIndex: ''
    }
    this.ask = this.ask.bind(this);
    this.handleChange = this.handleChange.bind(this);
  }
  ask() {
    if (this.state.userInput) {
      this.setState({
        randomIndex: Math.floor(Math.random() * 20),
        userInput: ''
      });
    }
  }
  handleChange(event) {
    this.setState({
      userInput: event.target.value
    });
  }
  render() {
    const possibleAnswers = [
      'It is certain',
      'It is decidedly so',
      'Without a doubt',
      'Yes, definitely',
      'You may rely on it',
      'As I see it, yes',
      'Outlook good',
      'Yes',
      'Signs point to yes',
      'Reply hazy try again',
      'Ask again later',
      'Better not tell you now',
      'Cannot predict now',
      'Concentrate and ask again',
      'Don\'t count on it',
      'My reply is no',
      'My sources say no',
      'Most likely',
      'Outlook not so good',
      'Very doubtful'
    ];
    const answer = 'change me!' // << change code here
    return (
      <div>
        <input
          type="text"
          value={this.state.userInput}
          onChange={this.handleChange}
          style={inputStyle} /><br />
```

```

<button onClick={this.ask}>
  Ask the Magic Eight Ball!
</button><br />
<h3>Answer:</h3>
<p>
  { /* change code below this line */ }

  { /* change code above this line */ }
</p>
</div>
);
}
};

```

After Test

```

var possibleAnswers = [ 'It is certain', 'It is decidedly so', 'Without a doubt', 'Yes, definitely',
  'You may rely on it', 'As I see it, yes', 'Outlook good', 'Yes', 'Signs point to yes', 'Reply hazy try
  again', 'Ask again later', 'Better not tell you now', 'Cannot predict now', 'Concentrate and ask again',
  'Don\'t count on it', 'My reply is no', 'My sources say no', 'Outlook not so good', 'Very doubtful',
  'Most likely' ];
ReactDOM.render(<MagicEightBall />, document.getElementById('root'))

```

Solution

```

const inputStyle = {
  width: 235,
  margin: 5
}

class MagicEightBall extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      userInput: '',
      randomIndex: ''
    }
    this.ask = this.ask.bind(this);
    this.handleChange = this.handleChange.bind(this);
  }
  ask() {
    if (this.state.userInput) {
      this.setState({
        randomIndex: Math.floor(Math.random() * 20),
        userInput: ''
      });
    }
  }
  handleChange(event) {
    this.setState({
      userInput: event.target.value
    });
  }
  render() {
    const possibleAnswers = [
      "It is certain", "It is decidedly so", "Without a doubt",
      "Yes, definitely", "You may rely on it", "As I see it, yes",
      "Outlook good", "Yes", "Signs point to yes", "Reply hazy try again",
      "Ask again later", "Better not tell you now", "Cannot predict now",
      "Concentrate and ask again", "Don't count on it", "My reply is no",
      "My sources say no", "Outlook not so good", "Very doubtful", "Most likely"
    ];
    const answer = possibleAnswers[this.state.randomIndex];
    return (
      <div>
        <input
          type="text"
          value={this.state.userInput}
          onChange={this.handleChange}
          style={inputStyle} /><br />
        <button onClick={this.ask}>Ask the Magic Eight Ball!</button><br />
        <h3>Answer:</h3>
      </div>
    );
  }
}

```

```

        <p>
          {answer}
        </p>
      </div>
    );
}
};

```

40. Render with an If/Else Condition

Description

Another application of using JavaScript to control your rendered view is to tie the elements that are rendered to a condition. When the condition is true, one view renders. When it's false, it's a different view. You can do this with a standard `if/else` statement in the `render()` method of a React component.

Instructions

MyComponent contains a `boolean` in its state which tracks whether you want to display some element in the UI or not. The `button` toggles the state of this value. Currently, it renders the same UI every time. Rewrite the `render()` method with an `if/else` statement so that if `display` is `true`, you return the current markup. Otherwise, return the markup without the `h1` element. **Note:** You must write an `if/else` to pass the tests. Use of the ternary operator will not pass here.

Challenge Seed

```

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      display: true
    }
    this.toggleDisplay = this.toggleDisplay.bind(this);
  }
  toggleDisplay() {
    this.setState({
      display: !this.state.display
    });
  }
  render() {
    // change code below this line

    return (
      <div>
        <button onClick={this.toggleDisplay}>Toggle Display</button>
        <h1>Displayed!</h1>
      </div>
    );
  }
};

```

After Test

```
ReactDOM.render(<MyComponent />, document.getElementById('root'))
```

Solution

```

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      display: true
    }
  }
  toggleDisplay() {
    this.setState({
      display: !this.state.display
    });
  }
  render() {
    if (this.state.display) {
      return (
        <div>
          <button onClick={this.toggleDisplay}>Toggle Display</button>
          <h1>Displayed!</h1>
        </div>
      );
    } else {
      return (
        <div>
          <button onClick={this.toggleDisplay}>Toggle Display</button>
        </div>
      );
    }
  }
};

```

```

        }
    this.toggleDisplay = this.toggleDisplay.bind(this);
}
toggleDisplay() {
    this.setState({
        display: !this.state.display
    });
}
render() {
    // change code below this line
    if (this.state.display) {
        return (
            <div>
                <button onClick={this.toggleDisplay}>Toggle Display</button>
                <h1>Displayed!</h1>
            </div>
        );
    } else {
        return (
            <div>
                <button onClick={this.toggleDisplay}>Toggle Display</button>
            </div>
        );
    }
}
};

```

41. Use `&&` for a More Concise Conditional

Description

The `if/else` statements worked in the last challenge, but there's a more concise way to achieve the same result. Imagine that you are tracking several conditions in a component and you want different elements to render depending on each of these conditions. If you write a lot of `else if` statements to return slightly different UIs, you may repeat code which leaves room for error. Instead, you can use the `&&` logical operator to perform conditional logic in a more concise way. This is possible because you want to check if a condition is `true`, and if it is, return some markup. Here's an example: `{condition && <p>markup</p>}` If the `condition` is `true`, the markup will be returned. If the condition is `false`, the operation will immediately return `false` after evaluating the condition and return nothing. You can include these statements directly in your JSX and string multiple conditions together by writing `&&` after each one. This allows you to handle more complex conditional logic in your `render()` method without repeating a lot of code.

Instructions

Solve the previous example again, so the `h1` only renders if `display` is `true`, but use the `&&` logical operator instead of an `if/else` statement.

Challenge Seed

```

class MyComponent extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            display: true
        }
        this.toggleDisplay = this.toggleDisplay.bind(this);
    }
    toggleDisplay() {
        this.setState({
            display: !this.state.display
        });
    }
    render() {
        // change code below this line
        return (
            <div>

```

```

        <button onClick={this.toggleDisplay}>Toggle Display</button>
        <h1>Displayed!</h1>
    </div>
);
}
};

```

After Test

```
ReactDOM.render(<MyComponent />, document.getElementById('root'))
```

Solution

```

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      display: true
    }
  this.toggleDisplay = this.toggleDisplay.bind(this);
}
toggleDisplay() {
  this.setState({
    display: !this.state.display
  });
}
render() {
  // change code below this line
  return (
    <div>
      <button onClick={this.toggleDisplay}>Toggle Display</button>
      {this.state.display && <h1>Displayed!</h1>}
    </div>
  );
}
};

```

42. Use a Ternary Expression for Conditional Rendering

Description

Before moving on to dynamic rendering techniques, there's one last way to use built-in JavaScript conditionals to render what you want: the **ternary operator**. The ternary operator is often utilized as a shortcut for `if/else` statements in JavaScript. They're not quite as robust as traditional `if/else` statements, but they are very popular among React developers. One reason for this is because of how JSX is compiled, `if/else` statements can't be inserted directly into JSX code. You might have noticed this a couple challenges ago — when an `if/else` statement was required, it was always *outside* the `return` statement. Ternary expressions can be an excellent alternative if you want to implement conditional logic within your JSX. Recall that a ternary operator has three parts, but you can combine several ternary expressions together. Here's the basic syntax:

`condition ? expressionIfTrue : expressionIfFalse`

Instructions

The code editor has three constants defined within the `CheckUserAge` component's `render()` method. They are called `buttonOne`, `buttonTwo`, and `buttonThree`. Each of these is assigned a simple JSX expression representing a button element. First, initialize the state of `CheckUserAge` with `input` and `userAge` both set to values of an empty string. Once the component is rendering information to the page, users should have a way to interact with it. Within the component's `return` statement, set up a ternary expression that implements the following logic: when the page first loads, render the submit button, `buttonOne`, to the page. Then, when a user enters their age and clicks the button, render a different button based on the age. If a user enters a number less than `18`, render `buttonThree`. If a user enters a number greater than or equal to `18`, render `buttonTwo`.

Challenge Seed

```

const inputStyle = {
  width: 235,
  margin: 5
}

class CheckUserAge extends React.Component {
  constructor(props) {
    super(props);
    // change code below this line

    // change code above this line
    this.submit = this.submit.bind(this);
    this.handleChange = this.handleChange.bind(this);
  }
  handleChange(e) {
    this.setState({
      input: e.target.value,
      userAge: ''
    });
  }
  submit() {
    this.setState({
      userAge: this.state.input
    });
  }
  render() {
    const buttonOne = <button onClick={this.submit}>Submit</button>;
    const buttonTwo = <button>You May Enter</button>;
    const buttonThree = <button>You Shall Not Pass</button>;
    return (
      <div>
        <h3>Enter Your Age to Continue</h3>
        <input
          style={inputStyle}
          type="number"
          value={this.state.input}
          onChange={this.handleChange} /><br />
        {
          /* change code here */
        }
      </div>
    );
  }
}

```

After Test

```
ReactDOM.render(<CheckUserAge />, document.getElementById('root'))
```

Solution

```

const inputStyle = {
  width: 235,
  margin: 5
}

class CheckUserAge extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      userAge: '',
      input: ''
    }
    this.submit = this.submit.bind(this);
    this.handleChange = this.handleChange.bind(this);
  }
  handleChange(e) {
    this.setState({
      input: e.target.value,

```

```

        userAge: ''
    });
}

submit() {
    this.setState({
        userAge: this.state.input
    });
}

render() {
    const buttonOne = <button onClick={this.submit}>Submit</button>;
    const buttonTwo = <button>You May Enter</button>;
    const buttonThree = <button>You Shall Not Pass</button>;
    return (
        <div>
            <h3>Enter Your Age to Continue</h3>
            <input
                style={inputStyle}
                type="number"
                value={this.state.input}
                onChange={this.handleChange} /><br />
            {
                this.state.userAge === '' ?
                buttonOne :
                this.state.userAge >= 18 ?
                buttonTwo :
                buttonThree
            }
        </div>
    );
}
};

```

43. Render Conditionally from Props

Description

So far, you've seen how to use `if/else`, `&&`, `null` and the ternary operator (`condition ? expressionIfTrue : expressionIfFalse`) to make conditional decisions about what to render and when. However, there's one important topic left to discuss that lets you combine any or all of these concepts with another powerful React feature: props. Using props to conditionally render code is very common with React developers — that is, they use the value of a given prop to automatically make decisions about what to render. In this challenge, you'll set up a child component to make rendering decisions based on props. You'll also use the ternary operator, but you can see how several of the other concepts that were covered in the last few challenges might be just as useful in this context.

Instructions

The code editor has two components that are partially defined for you: a parent called `GameOfChance`, and a child called `Results`. They are used to create a simple game where the user presses a button to see if they win or lose. First, you'll need a simple expression that randomly returns a different value every time it is run. You can use `Math.random()`. This method returns a value between `0` (inclusive) and `1` (exclusive) each time it is called. So for 50/50 odds, use `Math.random() > .5` in your expression. Statistically speaking, this expression will return `true` 50% of the time, and `false` the other 50%. On line 30, replace the comment with this expression to complete the variable declaration. Now you have an expression that you can use to make a randomized decision in the code. Next you need to implement this. Render the `Results` component as a child of `GameOfChance`, and pass in `expression` as a prop called `fiftyFifty`. In the `Results` component, write a ternary expression to render the text "You Win!" or "You Lose!" based on the `fiftyFifty` prop that's being passed in from `GameOfChance`. Finally, make sure the `handleClick()` method is correctly counting each turn so the user knows how many times they've played. This also serves to let the user know the component has actually updated in case they win or lose twice in a row.

Challenge Seed

```

class Results extends React.Component {
    constructor(props) {
        super(props);
    }
}

```

```

    }
    render() {
      return (
        <h1>
        {
          /* change code here */
        }
        </h1>
      )
    };
  };

  class GameOfChance extends React.Component {
    constructor(props) {
      super(props);
      this.state = {
        counter: 1
      }
      this.handleClick = this.handleClick.bind(this);
    }
    handleClick() {
      this.setState({
        counter: 0 // change code here
      });
    }
    render() {
      let expression = null; // change code here
      return (
        <div>
          <button onClick={this.handleClick}>Play Again</button>
          { /* change code below this line */ }

          { /* change code above this line */ }
          <p>'Turn: ' + this.state.counter</p>
        </div>
      );
    }
  };
}

```

After Test

```
ReactDOM.render(<GameOfChance />, document.getElementById('root'))
```

Solution

```

class Results extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <h1>
      {
        this.props.fiftyFifty ?
        'You Win!' :
        'You Lose!'
      }
      </h1>
    );
  };
}

class GameOfChance extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      counter: 1
    }
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    this.setState({

```

```

        counter: this.state.counter + 1
    });
}

render() {
    const expression = Math.random() > .5;
    return (
        <div>
            <button onClick={this.handleClick}>Play Again</button>
            <Results fiftyFifty={expression} />
            <p>'Turn: ' + this.state.counter</p>
        </div>
    );
}
};

```

44. Change Inline CSS Conditionally Based on Component State

Description

At this point, you've seen several applications of conditional rendering and the use of inline styles. Here's one more example that combines both of these topics. You can also render CSS conditionally based on the state of a React component. To do this, you check for a condition, and if that condition is met, you modify the styles object that's assigned to the JSX elements in the render method. This paradigm is important to understand because it is a dramatic shift from the more traditional approach of applying styles by modifying DOM elements directly (which is very common with jQuery, for example). In that approach, you must keep track of when elements change and also handle the actual manipulation directly. It can become difficult to keep track of changes, potentially making your UI unpredictable. When you set a style object based on a condition, you describe how the UI should look as a function of the application's state. There is a clear flow of information that only moves in one direction. This is the preferred method when writing applications with React.

Instructions

The code editor has a simple controlled input component with a styled border. You want to style this border red if the user types more than 15 characters of text in the input box. Add a condition to check for this and, if the condition is valid, set the input border style to `3px solid red`. You can try it out by entering text in the input.

Challenge Seed

```

class GateKeeper extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            input: ''
        };
        this.handleChange = this.handleChange.bind(this);
    }
    handleChange(event) {
        this.setState({ input: event.target.value })
    }
    render() {
        let inputStyle = {
            border: '1px solid black'
        };
        // change code below this line

        // change code above this line
        return (
            <div>
                <h3>Don't Type Too Much:</h3>
                <input
                    type="text"
                    style={inputStyle}
                    value={this.state.input}

```

```

        onChange={this.handleChange} />
      </div>
    );
}
);

```

After Test

```
ReactDOM.render(<GateKeeper />, document.getElementById('root'))
```

Solution

```

class GateKeeper extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      input: ''
    };
    this.handleChange = this.handleChange.bind(this);
  }
  handleChange(event) {
    this.setState({ input: event.target.value })
  }
  render() {
    let inputStyle = {
      border: '1px solid black'
    };
    if (this.state.input.length > 15) {
      inputStyle.border = '3px solid red';
    };
    return (
      <div>
        <h3>Don't Type Too Much:</h3>
        <input
          type="text"
          style={inputStyle}
          value={this.state.input}
          onChange={this.handleChange} />
      </div>
    );
  }
};

```

45. Use Array.map() to Dynamically Render Elements

Description

Conditional rendering is useful, but you may need your components to render an unknown number of elements. Often in reactive programming, a programmer has no way to know what the state of an application is until runtime, because so much depends on a user's interaction with that program. Programmers need to write their code to correctly handle that unknown state ahead of time. Using `Array.map()` in React illustrates this concept. For example, you create a simple "To Do List" app. As the programmer, you have no way of knowing how many items a user might have on their list. You need to set up your component to ***dynamically render*** the correct number of list elements long before someone using the program decides that today is laundry day.

Instructions

The code editor has most of the `MyToDoList` component set up. Some of this code should look familiar if you completed the controlled form challenge. You'll notice a `textarea` and a `button`, along with a couple of methods that track their states, but nothing is rendered to the page yet. Inside the `constructor`, create a `this.state` object and define two states: `userInput` should be initialized as an empty string, and `toDoList` should be initialized as an empty array. Next, delete the comment in the `render()` method next to the `items` variable. In its place, map over the

`toDoList` array stored in the component's internal state and dynamically render a `li` for each item. Try entering the string `eat, code, sleep, repeat` into the `textarea`, then click the button and see what happens. **Note:** You may know that all sibling child elements created by a mapping operation like this do need to be supplied with a unique `key` attribute. Don't worry, this is the topic of the next challenge.

Challenge Seed

```
const textAreaStyles = {
  width: 235,
  margin: 5
};

class MyToDoList extends React.Component {
  constructor(props) {
    super(props);
    // change code below this line

    // change code above this line
    this.handleSubmit = this.handleSubmit.bind(this);
    this.handleChange = this.handleChange.bind(this);
  }
  handleSubmit() {
    const itemsArray = this.state.userInput.split(',');
    this.setState({
      toDoList: itemsArray
    });
  }
  handleChange(e) {
    this.setState({
      userInput: e.target.value
    });
  }
  render() {
    const items = null; // change code here
    return (
      <div>
        <textarea
          onChange={this.handleChange}
          value={this.state.userInput}
          style={textAreaStyles}
          placeholder="Separate Items With Commas" /><br />
        <button onClick={this.handleSubmit}>Create List</button>
        <h1>My "To Do" List:</h1>
        <ul>
          {items}
        </ul>
      </div>
    );
  }
}
```

After Test

```
ReactDOM.render(<MyToDoList />, document.getElementById('root'))
```

Solution

```
const textAreaStyles = {
  width: 235,
  margin: 5
};

class MyToDoList extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      toDoList: [],
      userInput: ''
    }
  }
}
```

```

    this.handleSubmit = this.handleSubmit.bind(this);
    this.handleChange = this.handleChange.bind(this);
}
handleSubmit() {
  const itemsArray = this.state.userInput.split(',');
  this.setState({
    toDoList: itemsArray
  });
}
handleChange(e) {
  this.setState({
    userInput: e.target.value
  });
}
render() {
  const items = this.state.toDoList.map( (item, i) => {
    return <li key={i}>{item}</li>
  });
  return (
    <div>
      <textarea
        onChange={this.handleChange}
        value={this.state.userInput}
        style={textAreaStyles}
        placeholder="Separate Items With Commas" /><br />
      <button onClick={this.handleSubmit}>Create List</button>
      <h1>My "To Do" List:</h1>
      <ul>
        {items}
      </ul>
    </div>
  );
}
};

```

46. Give Sibling Elements a Unique Key Attribute

Description

The last challenge showed how the `map` method is used to dynamically render a number of elements based on user input. However, there was an important piece missing from that example. When you create an array of elements, each one needs a `key` attribute set to a unique value. React uses these keys to keep track of which items are added, changed, or removed. This helps make the re-rendering process more efficient when the list is modified in any way.

Note: Keys only need to be unique between sibling elements, they don't need to be globally unique in your application.

Instructions

The code editor has an array with some front end frameworks and a stateless functional component named `Frameworks()`. `Frameworks()` needs to map the array to an unordered list, much like in the last challenge. Finish writing the `map` callback to return an `li` element for each framework in the `frontEndFrameworks` array. This time, make sure to give each `li` a `key` attribute, set to a unique value. The `li` elements should also contain text from `frontEndFrameworks`. Normally, you want to make the key something that uniquely identifies the element being rendered. As a last resort the array index may be used, but typically you should try to use a unique identification.

Challenge Seed

```

const frontEndFrameworks = [
  'React',
  'Angular',
  'Ember',
  'Knockout',
  'Backbone',
  'Vue'
]

```

```

];
}

function Frameworks() {
  const renderFrameworks = null; // change code here
  return (
    <div>
      <h1>Popular Front End JavaScript Frameworks</h1>
      <ul>
        {renderFrameworks}
      </ul>
    </div>
  );
}

```

After Test

```
ReactDOM.render(<Frameworks />, document.getElementById('root'))
```

Solution

```

const frontEndFrameworks = [
  'React',
  'Angular',
  'Ember',
  'Knockout',
  'Backbone',
  'Vue'
];

function Frameworks() {
  const renderFrameworks = frontEndFrameworks.map((fw, i) => <li key={i}>{fw}</li>);
  return (
    <div>
      <h1>Popular Front End JavaScript Frameworks</h1>
      <ul>
        {renderFrameworks}
      </ul>
    </div>
  );
}

```

47. Use Array.filter() to Dynamically Filter an Array

Description

The `map` array method is a powerful tool that you will use often when working with React. Another method related to `map` is `filter`, which filters the contents of an array based on a condition, then returns a new array. For example, if you have an array of users that all have a property `online` which can be set to `true` or `false`, you can filter only those users that are online by writing: `let onlineUsers = users.filter(user => user.online);`

Instructions

In the code editor, `MyComponent`'s `state` is initialized with an array of users. Some users are online and some aren't. Filter the array so you see only the users who are online. To do this, first use `filter` to return a new array containing only the users whose `online` property is `true`. Then, in the `renderOnline` variable, map over the filtered array, and return a `li` element for each user that contains the text of their `username`. Be sure to include a unique `key` as well, like in the last challenges.

Challenge Seed

```

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      users: [
        {
          username: 'Jeff',
          online: true
        },
        {
          username: 'Alan',
          online: false
        },
        {
          username: 'Mary',
          online: true
        },
        {
          username: 'Jim',
          online: false
        },
        {
          username: 'Sara',
          online: true
        },
        {
          username: 'Laura',
          online: true
        }
      ]
    }
  }
  render() {
    const usersOnline = null; // change code here
    const renderOnline = null; // change code here
    return (
      <div>
        <h1>Current Online Users:</h1>
        <ul>
          {renderOnline}
        </ul>
      </div>
    );
  }
};

```

After Test

```
ReactDOM.render(<MyComponent />, document.getElementById('root'))
```

Solution

```

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      users: [
        {
          username: 'Jeff',
          online: true
        },
        {
          username: 'Alan',
          online: false
        },
        {
          username: 'Mary',
          online: true
        },
        {
          username: 'Jim',
          online: false
        }
      ]
    }
  }
  render() {
    const usersOnline = null; // change code here
    const renderOnline = null; // change code here
    return (
      <div>
        <h1>Current Online Users:</h1>
        <ul>
          {renderOnline}
        </ul>
      </div>
    );
  }
};

```

```

        online: false
    },
    {
        username: 'Sara',
        online: true
    },
    {
        username: 'Laura',
        online: true
    }
]
}
}

render() {
    const usersOnline = this.state.users.filter(user => {
        return user.online;
    });
    const renderOnlineUsers = usersOnline.map(user => {
        return (
            <li key={user.username}>{user.username}</li>
        );
    });
    return (
        <div>
            <h1>Current Online Users:</h1>
            <ul>
                {renderOnlineUsers}
            </ul>
        </div>
    );
}
}

```

48. Render React on the Server with `renderToString`

Description

So far, you have been rendering React components on the client. Normally, this is what you will always do. However, there are some use cases where it makes sense to render a React component on the server. Since React is a JavaScript view library and you can run JavaScript on the server with Node, this is possible. In fact, React provides a `renderToString()` method you can use for this purpose. There are two key reasons why rendering on the server may be used in a real world app. First, without doing this, your React apps would consist of a relatively empty HTML file and a large bundle of JavaScript when it's initially loaded to the browser. This may not be ideal for search engines that are trying to index the content of your pages so people can find you. If you render the initial HTML markup on the server and send this to the client, the initial page load contains all of the page's markup which can be crawled by search engines. Second, this creates a faster initial page load experience because the rendered HTML is smaller than the JavaScript code of the entire app. React will still be able to recognize your app and manage it after the initial load.

Instructions

The `renderToString()` method is provided on `ReactDOMServer`, which is available here as a global object. The method takes one argument which is a React element. Use this to render `App` to a string.

Challenge Seed

```

class App extends React.Component {
    constructor(props) {
        super(props);
    }
    render() {
        return <div/>
    }
}

// change code below this line

```

Before Test

```
var ReactDOMServer = { renderToString(x) { return null; } };
```

After Test

```
ReactDOM.render(<App />, document.getElementById('root'))
```

Solution

```
class App extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return <div>
  }
};

// change code below this line
ReactDOMServer.renderToString(<App/>);
```

Redux

1. Create a Redux Store

Description

Redux is a state management framework that can be used with a number of different web technologies, including React. In Redux, there is a single state object that's responsible for the entire state of your application. This means if you had a React app with ten components, and each component had its own local state, the entire state of your app would be defined by a single state object housed in the Redux store. This is the first important principle to understand when learning Redux: the Redux store is the single source of truth when it comes to application state. This also means that any time any piece of your app wants to update state, it **must** do so through the Redux store. The unidirectional data flow makes it easier to track state management in your app.

Instructions

The Redux store is an object which holds and manages application state. There is a method called `createStore()` on the Redux object, which you use to create the Redux store. This method takes a `reducer` function as a required argument. The `reducer` function is covered in a later challenge, and is already defined for you in the code editor. It simply takes `state` as an argument and returns `state`. Declare a `store` variable and assign it to the `createStore()` method, passing in the `reducer` as an argument. **Note:** The code in the editor uses ES6 default argument syntax to initialize this state to hold a value of `5`. If you're not familiar with default arguments, you can refer to the [ES6 section in the Curriculum](#) which covers this topic.

Challenge Seed

```
const reducer = (state = 5) => {
  return state;
}

// Redux methods are available from a Redux object
// For example: Redux.createStore()
// Define the store here:
```

Solution

```
const reducer = (state = 5) => {
  return state;
}

// Redux methods are available from a Redux object
// For example: Redux.createStore()
// Define the store here:

const store = Redux.createStore(reducer);
```

2. Get State from the Redux Store

Description

The Redux store object provides several methods that allow you to interact with it. For example, you can retrieve the current state held in the Redux store object with the `getState()` method.

Instructions

The code from the previous challenge is re-written more concisely in the code editor. Use `store.getState()` to retrieve the state from the `store`, and assign this to a new variable `currentState`.

Challenge Seed

```
const store = Redux.createStore(
  (state = 5) => state
);

// change code below this line
```

Solution

```
const store = Redux.createStore(
  (state = 5) => state
);

// change code below this line
const currentState = store.getState();
```

3. Define a Redux Action

Description

Since Redux is a state management framework, updating state is one of its core tasks. In Redux, all state updates are triggered by dispatching actions. An action is simply a JavaScript object that contains information about an action event that has occurred. The Redux store receives these action objects, then updates its state accordingly. Sometimes a Redux action also carries some data. For example, the action carries a username after a user logs in. While the data is optional, actions must carry a `type` property that specifies the 'type' of action that occurred. Think of Redux actions as messengers that deliver information about events happening in your app to the Redux store. The store then conducts the business of updating state based on the action that occurred.

Instructions

Writing a Redux action is as simple as declaring an object with a type property. Declare an object `action` and give it a property `type` set to the string '`LOGIN`'.

Challenge Seed

```
// Define an action here:
```

Solution

```
const action = {
  type: 'LOGIN'
}
```

4. Define an Action Creator

Description

After creating an action, the next step is sending the action to the Redux store so it can update its state. In Redux, you define action creators to accomplish this. An action creator is simply a JavaScript function that returns an action. In other words, action creators create objects that represent action events.

Instructions

Define a function named `actionCreator()` that returns the `action` object when called.

Challenge Seed

```
const action = {
  type: 'LOGIN'
}
// Define an action creator here:
```

Solution

```
const action = {
  type: 'LOGIN'
}
// Define an action creator here:
const actionCreator = () => {
  return action;
};
```

5. Dispatch an Action Event

Description

`dispatch` method is what you use to dispatch actions to the Redux store. Calling `store.dispatch()` and passing the value returned from an action creator sends an action back to the store. Recall that action creators return an object with a type property that specifies the action that has occurred. Then the method dispatches an action object to the Redux store. Based on the previous challenge's example, the following lines are equivalent, and both dispatch the action of type `LOGIN`:

```
store.dispatch(actionCreator());
store.dispatch({ type: 'LOGIN' });
```

Instructions

The Redux store in the code editor has an initialized state that's an object containing a `login` property currently set to `false`. There's also an action creator called `loginAction()` which returns an action of type `LOGIN`. Dispatch the `LOGIN` action to the Redux store by calling the `dispatch` method, and pass in the action created by `loginAction()`.

Challenge Seed

```
const store = Redux.createStore(
  (state = {login: false}) => state
);

const loginAction = () => {
  return {
    type: 'LOGIN'
  }
};

// Dispatch the action here:
```

Solution

```
const store = Redux.createStore(
  (state = {login: false}) => state
);

const loginAction = () => {
  return {
    type: 'LOGIN'
  }
};

// Dispatch the action here:
store.dispatch(loginAction());
```

6. Handle an Action in the Store

Description

After an action is created and dispatched, the Redux store needs to know how to respond to that action. This is the job of a `reducer` function. Reducers in Redux are responsible for the state modifications that take place in response to actions. A reducer takes `state` and `action` as arguments, and it always returns a new `state`. It is important to see that this is the **only** role of the reducer. It has no side effects — it never calls an API endpoint and it never has any hidden surprises. The reducer is simply a pure function that takes state and action, then returns new state. Another key principle in Redux is that `state` is read-only. In other words, the `reducer` function must **always** return a new copy of `state` and never modify state directly. Redux does not enforce state immutability, however, you are responsible for enforcing it in the code of your reducer functions. You'll practice this in later challenges.

Instructions

The code editor has the previous example as well as the start of a `reducer` function for you. Fill in the body of the `reducer` function so that if it receives an action of type `'LOGIN'` it returns a state object with `login` set to `true`. Otherwise, it returns the current `state`. Note that the current `state` and the dispatched `action` are passed to the reducer, so you can access the action's type directly with `action.type`.

Challenge Seed

```

const defaultState = {
  login: false
};

const reducer = (state = defaultState, action) => {
  // change code below this line

  // change code above this line
};

const store = Redux.createStore(reducer);

const loginAction = () => {
  return {
    type: 'LOGIN'
  }
};

```

Solution

```

const defaultState = {
  login: false
};

const reducer = (state = defaultState, action) => {

  if (action.type === 'LOGIN') {
    return {login: true}
  }

  else {
    return state
  }
};

const store = Redux.createStore(reducer);

const loginAction = () => {
  return {
    type: 'LOGIN'
  }
};

```

7. Use a Switch Statement to Handle Multiple Actions

Description

You can tell the Redux store how to handle multiple action types. Say you are managing user authentication in your Redux store. You want to have a state representation for when users are logged in and when they are logged out. You represent this with a single state object with the property `authenticated`. You also need action creators that create actions corresponding to user login and user logout, along with the action objects themselves.

Instructions

The code editor has a store, actions, and action creators set up for you. Fill in the `reducer` function to handle multiple authentication actions. Use a JavaScript `switch` statement in the `reducer` to respond to different action events. This is a standard pattern in writing Redux reducers. The switch statement should switch over `action.type` and return the appropriate authentication state. **Note:** At this point, don't worry about state immutability, since it is small and simple in this example. For each action, you can return a new object — for example, `{authenticated: true}`. Also, don't forget to write a `default` case in your switch statement that returns the current `state`. This is important because once your app has multiple reducers, they are all run any time an action dispatch is made, even when the action isn't related to that reducer. In such a case, you want to make sure that you return the current `state`.

Challenge Seed

```
const defaultState = {
  authenticated: false
};

const authReducer = (state = defaultState, action) => {
  // change code below this line

  // change code above this line
};

const store = Redux.createStore(authReducer);

const loginUser = () => {
  return {
    type: 'LOGIN'
  };
};

const logoutUser = () => {
  return {
    type: 'LOGOUT'
  };
};
```

Solution

```
const defaultState = {
  authenticated: false
};

const authReducer = (state = defaultState, action) => {

  switch (action.type) {

    case 'LOGIN':
      return {
        authenticated: true
      }

    case 'LOGOUT':
      return {
        authenticated: false
      }

    default:
      return state;
  }
};

const store = Redux.createStore(authReducer);

const loginUser = () => {
  return {
    type: 'LOGIN'
  };
};

const logoutUser = () => {
  return {
    type: 'LOGOUT'
  };
};
```

8. Use const for Action Types

Description

A common practice when working with Redux is to assign action types as read-only constants, then reference these constants wherever they are used. You can refactor the code you're working with to write the action types as `const` declarations.

Instructions

Declare `LOGIN` and `LOGOUT` as `const` values and assign them to the strings '`LOGIN`' and '`LOGOUT`', respectively. Then, edit the `authReducer()` and the action creators to reference these constants instead of string values. **Note:** It's generally a convention to write constants in all uppercase, and this is standard practice in Redux as well.

Challenge Seed

```
// change code below this line

// change code above this line

const defaultState = {
  authenticated: false
};

const authReducer = (state = defaultState, action) => {

  switch (action.type) {

    case 'LOGIN':
      return {
        authenticated: true
      }

    case 'LOGOUT':
      return {
        authenticated: false
      }

    default:
      return state;
  }
};

const store = Redux.createStore(authReducer);

const loginUser = () => {
  return {
    type: 'LOGIN'
}
};

const logoutUser = () => {
  return {
    type: 'LOGOUT'
}
};
```

Solution

```
const LOGIN = 'LOGIN';
const LOGOUT = 'LOGOUT';

const defaultState = {
  authenticated: false
};

const authReducer = (state = defaultState, action) => {

  switch (action.type) {
```

```

    case LOGIN:
      return {
        authenticated: true
      }

    case LOGOUT:
      return {
        authenticated: false
      }

    default:
      return state;
  }
};

const store = Redux.createStore(authReducer);

const loginUser = () => {
  return {
    type: LOGIN
  }
};

const logoutUser = () => {
  return {
    type: LOGOUT
  }
};

```

9. Register a Store Listener

Description

Another method you have access to on the `Redux store` object is `store.subscribe()`. This allows you to subscribe listener functions to the store, which are called whenever an action is dispatched against the store. One simple use for this method is to subscribe a function to your store that simply logs a message every time an action is received and the store is updated.

Instructions

Write a callback function that increments the global variable `count` every time the store receives an action, and pass this function in to the `store.subscribe()` method. You'll see that `store.dispatch()` is called three times in a row, each time directly passing in an action object. Watch the console output between the action dispatches to see the updates take place.

Challenge Seed

```

const ADD = 'ADD';

const reducer = (state = 0, action) => {
  switch(action.type) {
    case ADD:
      return state + 1;
    default:
      return state;
  }
};

const store = Redux.createStore(reducer);

// global count variable:
let count = 0;

// change code below this line

```

```
// change code above this line

store.dispatch({type: ADD});
console.log(count);
store.dispatch({type: ADD});
console.log(count);
store.dispatch({type: ADD});
console.log(count);
```

Before Test

```
count = 0;
```

Solution

```
const ADD = 'ADD';

const reducer = (state = 0, action) => {
  switch(action.type) {
    case ADD:
      return state + 1;
    default:
      return state;
  }
};

const store = Redux.createStore(reducer);
let count = 0;
// change code below this line

store.subscribe( () =>
{
  count++;
});

// change code above this line

store.dispatch({type: ADD});
store.dispatch({type: ADD});
store.dispatch({type: ADD});
```

10. Combine Multiple Reducers

Description

When the state of your app begins to grow more complex, it may be tempting to divide state into multiple pieces. Instead, remember the first principle of Redux: all app state is held in a single state object in the store. Therefore, Redux provides reducer composition as a solution for a complex state model. You define multiple reducers to handle different pieces of your application's state, then compose these reducers together into one root reducer. The root reducer is then passed into the Redux `createStore()` method. In order to let us combine multiple reducers together, Redux provides the `combineReducers()` method. This method accepts an object as an argument in which you define properties which associate keys to specific reducer functions. The name you give to the keys will be used by Redux as the name for the associated piece of state. Typically, it is a good practice to create a reducer for each piece of application state when they are distinct or unique in some way. For example, in a note-taking app with user authentication, one reducer could handle authentication while another handles the text and notes that the user is submitting. For such an application, we might write the `combineReducers()` method like this:

```
const rootReducer = Redux.combineReducers({
  auth: authenticationReducer,
  notes: notesReducer
});
```

Now, the key `notes` will contain all of the state associated with our notes and handled by our `notesReducer`. This is how multiple reducers can be composed to manage more complex application state. In this example, the state held in the Redux store would then be a single object containing `auth` and `notes` properties.

Instructions

There are `counterReducer()` and `authReducer()` functions provided in the code editor, along with a Redux store. Finish writing the `rootReducer()` function using the `Redux.combineReducers()` method. Assign `counterReducer` to a key called `count` and `authReducer` to a key called `auth`.

Challenge Seed

```
const INCREMENT = 'INCREMENT';
const DECREMENT = 'DECREMENT';

const counterReducer = (state = 0, action) => {
  switch(action.type) {
    case INCREMENT:
      return state + 1;
    case DECREMENT:
      return state - 1;
    default:
      return state;
  }
};

const LOGIN = 'LOGIN';
const LOGOUT = 'LOGOUT';

const authReducer = (state = {authenticated: false}, action) => {
  switch(action.type) {
    case LOGIN:
      return {
        authenticated: true
      }
    case LOGOUT:
      return {
        authenticated: false
      }
    default:
      return state;
  }
};

const rootReducer = // define the root reducer here

const store = Redux.createStore(rootReducer);
```

Solution

```
const INCREMENT = 'INCREMENT';
const DECREMENT = 'DECREMENT';

const counterReducer = (state = 0, action) => {
  switch(action.type) {
    case INCREMENT:
      return state + 1;
    case DECREMENT:
      return state - 1;
    default:
      return state;
  }
};

const LOGIN = 'LOGIN';
const LOGOUT = 'LOGOUT';

const authReducer = (state = {authenticated: false}, action) => {
  switch(action.type) {
    case LOGIN:
```

```

        return {
          authenticated: true
        }
      case LOGOUT:
        return {
          authenticated: false
        }
      default:
        return state;
    }
};

const rootReducer = Redux.combineReducers({
  count: counterReducer,
  auth: authReducer
});

const store = Redux.createStore(rootReducer);

```

11. Send Action Data to the Store

Description

By now you've learned how to dispatch actions to the Redux store, but so far these actions have not contained any information other than a `type`. You can also send specific data along with your actions. In fact, this is very common because actions usually originate from some user interaction and tend to carry some data with them. The Redux store often needs to know about this data.

Instructions

There's a basic `notesReducer()` and an `addNoteText()` action creator defined in the code editor. Finish the body of the `addNoteText()` function so that it returns an `action` object. The object should include a `type` property with a value of `ADD_NOTE`, and also a `text` property set to the `note` data that's passed into the action creator. When you call the action creator, you'll pass in specific note information that you can access for the object. Next, finish writing the `switch` statement in the `notesReducer()`. You need to add a case that handles the `addNoteText()` actions. This case should be triggered whenever there is an action of type `ADD_NOTE` and it should return the `text` property on the incoming `action` as the new `state`. The action is dispatched at the bottom of the code. Once you're finished, run the code and watch the console. That's all it takes to send action-specific data to the store and use it when you update `store state`.

Challenge Seed

```

const ADD_NOTE = 'ADD_NOTE';

const notesReducer = (state = 'Initial State', action) => {
  switch(action.type) {
    // change code below this line

    // change code above this line
    default:
      return state;
  }
};

const addNoteText = (note) => {
  // change code below this line

  // change code above this line
};

const store = Redux.createStore(notesReducer);

console.log(store.getState());
store.dispatch(addNoteText('Hello!'));
console.log(store.getState());

```

Solution

```

const ADD_NOTE = 'ADD_NOTE';

const notesReducer = (state = 'Initial State', action) => {
  switch(action.type) {
    // change code below this line
    case ADD_NOTE:
      return action.text;
    // change code above this line
    default:
      return state;
  }
};

const addNoteText = (note) => {
  // change code below this line
  return {
    type: ADD_NOTE,
    text: note
  }
  // change code above this line
};

const store = Redux.createStore(notesReducer);

console.log(store.getState());
store.dispatch(addNoteText('Hello Redux!'));
console.log(store.getState());

```

12. Use Middleware to Handle Asynchronous Actions

Description

So far these challenges have avoided discussing asynchronous actions, but they are an unavoidable part of web development. At some point you'll need to call asynchronous endpoints in your Redux app, so how do you handle these types of requests? Redux provides middleware designed specifically for this purpose, called Redux Thunk middleware. Here's a brief description how to use this with Redux. To include Redux Thunk middleware, you pass it as an argument to `Redux.applyMiddleware()`. This statement is then provided as a second optional parameter to the `createStore()` function. Take a look at the code at the bottom of the editor to see this. Then, to create an asynchronous action, you return a function in the action creator that takes `dispatch` as an argument. Within this function, you can dispatch actions and perform asynchronous requests. In this example, an asynchronous request is simulated with a `setTimeout()` call. It's common to dispatch an action before initiating any asynchronous behavior so that your application state knows that some data is being requested (this state could display a loading icon, for instance). Then, once you receive the data, you dispatch another action which carries the data as a payload along with information that the action is completed. Remember that you're passing `dispatch` as a parameter to this special action creator. This is what you'll use to dispatch your actions, you simply pass the action directly to `dispatch` and the middleware takes care of the rest.

Instructions

Write both dispatches in the `handleAsync()` action creator. Dispatch `requestingData()` before the `setTimeout()` (the simulated API call). Then, after you receive the (pretend) data, dispatch the `receivedData()` action, passing in this data. Now you know how to handle asynchronous actions in Redux. Everything else continues to behave as before.

Challenge Seed

```

const REQUESTING_DATA = 'REQUESTING_DATA'
const RECEIVED_DATA = 'RECEIVED_DATA'

```

```

const requestingData = () => { return {type: REQUESTING_DATA} }
const receivedData = (data) => { return {type: RECEIVED_DATA, users: data.users} }

const handleAsync = () => {
  return function(dispatch) {
    // dispatch request action here

    setTimeout(function() {
      let data = {
        users: ['Jeff', 'William', 'Alice']
      }
      // dispatch received data action here

      }, 2500);
    }
};

const defaultState = {
  fetching: false,
  users: []
};

const asyncDataReducer = (state = defaultState, action) => {
  switch(action.type) {
    case REQUESTING_DATA:
      return {
        fetching: true,
        users: []
      }
    case RECEIVED_DATA:
      return {
        fetching: false,
        users: action.users
      }
    default:
      return state;
  }
};

const store = Redux.createStore(
  asyncDataReducer,
  Redux.applyMiddleware(ReduxThunk.default)
);

```

Solution

```

const REQUESTING_DATA = 'REQUESTING_DATA'
const RECEIVED_DATA = 'RECEIVED_DATA'

const requestingData = () => { return {type: REQUESTING_DATA} }
const receivedData = (data) => { return {type: RECEIVED_DATA, users: data.users} }

const handleAsync = () => {
  return function(dispatch) {
    dispatch(requestingData());
    setTimeout(function() {
      let data = {
        users: ['Jeff', 'William', 'Alice']
      }
      dispatch(receivedData(data));
      }, 2500);
    }
};

const defaultState = {
  fetching: false,
  users: []
};

const asyncDataReducer = (state = defaultState, action) => {
  switch(action.type) {
    case REQUESTING_DATA:
      return {
        fetching: true,

```

```

        users: []
    }
    case RECEIVED_DATA:
    return {
        fetching: false,
        users: action.users
    }
default:
return state;
}
};

const store = Redux.createStore(
    asyncDataReducer,
    Redux.applyMiddleware(ReduxThunk.default)
);

```

13. Write a Counter with Redux

Description

Now you've learned all the core principles of Redux! You've seen how to create actions and action creators, create a Redux store, dispatch your actions against the store, and design state updates with pure reducers. You've even seen how to manage complex state with reducer composition and handle asynchronous actions. These examples are simplistic, but these concepts are the core principles of Redux. If you understand them well, you're ready to start building your own Redux app. The next challenges cover some of the details regarding state immutability, but first, here's a review of everything you've learned so far.

Instructions

In this lesson, you'll implement a simple counter with Redux from scratch. The basics are provided in the code editor, but you'll have to fill in the details! Use the names that are provided and define `incAction` and `decAction` action creators, the `counterReducer()`, `INCREMENT` and `DECREMENT` action types, and finally the Redux `store`. Once you're finished you should be able to dispatch `INCREMENT` or `DECREMENT` actions to increment or decrement the state held in the `store`. Good luck building your first Redux app!

Challenge Seed

```

const INCREMENT = null; // define a constant for increment action types
const DECREMENT = null; // define a constant for decrement action types

const counterReducer = null; // define the counter reducer which will increment or decrement the state
                           // based on the action it receives

const incAction = null; // define an action creator for incrementing

const decAction = null; // define an action creator for decrementing

const store = null; // define the Redux store here, passing in your reducers

```

Solution

```

const INCREMENT = 'INCREMENT';
const DECREMENT = 'DECREMENT';

const counterReducer = (state = 0, action) => {
    switch(action.type) {
        case INCREMENT:
            return state + 1;
        case DECREMENT:
            return state - 1;
        default:
            return state;
    }
};

```

```

};

const incAction = () => {
  return {
    type: INCREMENT
  }
};

const decAction = () => {
  return {
    type: DECREMENT
  }
};

const store = Redux.createStore(counterReducer);

```

14. Never Mutate State

Description

These final challenges describe several methods of enforcing the key principle of state immutability in Redux. Immutable state means that you never modify state directly, instead, you return a new copy of state. If you took a snapshot of the state of a Redux app over time, you would see something like `state 1`, `state 2`, `state 3`, `state 4`, ... and so on where each state may be similar to the last, but each is a distinct piece of data. This immutability, in fact, is what provides such features as time-travel debugging that you may have heard about. Redux does not actively enforce state immutability in its store or reducers, that responsibility falls on the programmer. Fortunately, JavaScript (especially ES6) provides several useful tools you can use to enforce the immutability of your state, whether it is a `string`, `number`, `array`, or `object`. Note that strings and numbers are primitive values and are immutable by nature. In other words, `3` is always `3`. You cannot change the value of the number `3`. An `array` or `object`, however, is mutable. In practice, your state will probably consist of an `array` or `object`, as these are useful data structures for representing many types of information.

Instructions

There is a `store` and `reducer` in the code editor for managing to-do items. Finish writing the `ADD_TO_DO` case in the reducer to append a new to-do to the state. There are a few ways to accomplish this with standard JavaScript or ES6. See if you can find a way to return a new array with the item from `action.todo` appended to the end.

Challenge Seed

```

const ADD_TO_DO = 'ADD_TO_DO';

// A list of strings representing tasks to do:
const todos = [
  'Go to the store',
  'Clean the house',
  'Cook dinner',
  'Learn to code',
];

const immutableReducer = (state = todos, action) => {
  switch(action.type) {
    case ADD_TO_DO:
      // don't mutate state here or the tests will fail
      return
    default:
      return state;
  }
};

// an example todo argument would be 'Learn React',
const addToDo = (todo) => {
  return {
    type: ADD_TO_DO,

```

```

        todo
    }

const store = Redux.createStore(immutableReducer);

```

Solution

```

const ADD_TO_DO = 'ADD_TO_DO';

// A list of strings representing tasks to do:
const todos = [
  'Go to the store',
  'Clean the house',
  'Cook dinner',
  'Learn to code',
];

const immutableReducer = (state = todos, action) => {
  switch(action.type) {
    case ADD_TO_DO:
      return state.concat(action.todo);
    default:
      return state;
  }
};

// an example todo argument would be 'Learn React',
const addToDo = (todo) => {
  return {
    type: ADD_TO_DO,
    todo
  }
}

const store = Redux.createStore(imutableReducer);

```

15. Use the Spread Operator on Arrays

Description

One solution from ES6 to help enforce state immutability in Redux is the spread operator: `...array`. The spread operator has a variety of applications, one of which is well-suited to the previous challenge of producing a new array from an existing array. This is relatively new, but commonly used syntax. For example, if you have an array `myArray` and write:

```
let newArray = [...myArray];
```

`newArray` is now a clone of `myArray`. Both arrays still exist separately in memory. If you perform a mutation like `newArray.push(5)`, `myArray` doesn't change. The `...` effectively *spreads* out the values in `myArray` into a new array. To clone an array but add additional values in the new array, you could write

```
[...myArray, 'new value']
```

This would return a new array composed of the values in `myArray` and the string 'new value' as the last value. The spread syntax can be used multiple times in array composition like this, but it's important to note that it only makes a shallow copy of the array. That is to say, it only provides immutable array operations for one-dimensional arrays.

Instructions

Use the spread operator to return a new copy of state when a to-do is added.

Challenge Seed

```

const immutableReducer = (state = ['Do not mutate state!'], action) => {
  switch(action.type) {
    case 'ADD_TO_DO':
      // don't mutate state here or the tests will fail
      return

```

```

    default:
      return state;
  };

const addToDo = (todo) => {
  return {
    type: 'ADD_TO_DO',
    todo
  }
}

const store = Redux.createStore(immutableReducer);

```

Solution

```

const immutableReducer = (state = ['Do not mutate state!'], action) => {
  switch(action.type) {
    case 'ADD_TO_DO':
      return [
        ...state,
        action.todo
      ];
    default:
      return state;
  }
};

const addToDo = (todo) => {
  return {
    type: 'ADD_TO_DO',
    todo
  }
}

const store = Redux.createStore(imutableReducer);

```

16. Remove an Item from an Array

Description

Time to practice removing items from an array. The spread operator can be used here as well. Other useful JavaScript methods include `slice()` and `concat()`.

Instructions

The reducer and action creator were modified to remove an item from an array based on the index of the item. Finish writing the reducer so a new state array is returned with the item at the specific index removed.

Challenge Seed

```

const immutableReducer = (state = [0,1,2,3,4,5], action) => {
  switch(action.type) {
    case 'REMOVE_ITEM':
      // don't mutate state here or the tests will fail
      return
    default:
      return state;
  }
};

const removeItem = (index) => {
  return {
    type: 'REMOVE_ITEM',
    index
  }
}

```

```

    }

const store = Redux.createStore(immutableReducer);

```

Solution

```

const immutableReducer = (state = [0,1,2,3,4,5], action) => {
  switch(action.type) {
    case 'REMOVE_ITEM':
      return [
        ...state.slice(0, action.index),
        ...state.slice(action.index + 1)
      ];
    default:
      return state;
  }
};

const removeItem = (index) => {
  return {
    type: 'REMOVE_ITEM',
    index
  }
}

const store = Redux.createStore(immutableReducer);

```

17. Copy an Object with Object.assign

Description

The last several challenges worked with arrays, but there are ways to help enforce state immutability when state is an object, too. A useful tool for handling objects is the `Object.assign()` utility. `Object.assign()` takes a target object and source objects and maps properties from the source objects to the target object. Any matching properties are overwritten by properties in the source objects. This behavior is commonly used to make shallow copies of objects by passing an empty object as the first argument followed by the object(s) you want to copy. Here's an example: `const newObject = Object.assign({}, obj1, obj2);` This creates `newObject` as a new object, which contains the properties that currently exist in `obj1` and `obj2`.

Instructions

The Redux state and actions were modified to handle an `object` for the `state`. Edit the code to return a new `state` object for actions with type `ONLINE`, which set the `status` property to the string `online`. Try to use `Object.assign()` to complete the challenge.

Challenge Seed

```

const defaultState = {
  user: 'CamperBot',
  status: 'offline',
  friends: '732,982',
  community: 'freeCodeCamp'
};

const immutableReducer = (state = defaultState, action) => {
  switch(action.type) {
    case 'ONLINE':
      // don't mutate state here or the tests will fail
      return
    default:
      return state;
  }
};

```

```

};

const wakeUp = () => {
  return {
    type: 'ONLINE'
  }
};

const store = Redux.createStore(immutableReducer);

```

Solution

```

const defaultState = {
  user: 'CamperBot',
  status: 'offline',
  friends: '732,982',
  community: 'freeCodeCamp'
};

const immutableReducer = (state = defaultState, action) => {
  switch(action.type) {
    case 'ONLINE':
      return Object.assign({}, state, {
        status: 'online'
      });
    default:
      return state;
  }
};

const wakeUp = () => {
  return {
    type: 'ONLINE'
  }
};

const store = Redux.createStore(imutableReducer);

```

React and Redux

1. Getting Started with React Redux

Description

This series of challenges introduces how to use Redux with React. First, here's a review of some of the key principles of each technology. React is a view library that you provide with data, then it renders the view in an efficient, predictable way. Redux is a state management framework that you can use to simplify the management of your application's state. Typically, in a React Redux app, you create a single Redux store that manages the state of your entire app. Your React components subscribe to only the pieces of data in the store that are relevant to their role. Then, you dispatch actions directly from React components, which then trigger store updates. Although React components can manage their own state locally, when you have a complex app, it's generally better to keep the app state in a single location with Redux. There are exceptions when individual components may have local state specific only to them. Finally, because Redux is not designed to work with React out of the box, you need to use the `react-redux` package. It provides a way for you to pass Redux state and `dispatch` to your React components as `props`. Over the next few challenges, first, you'll create a simple React component which allows you to input new text messages. These are added to an array that's displayed in the view. This should be a nice review of what you learned in the React lessons. Next, you'll create a Redux store and actions that manage the state of the messages array. Finally, you'll use `react-redux` to connect the Redux store with your component, thereby extracting the local state into the Redux store.

Instructions

Start with a `DisplayMessages` component. Add a constructor to this component and initialize it with a state that has two properties: `input`, that's set to an empty string, and `messages`, that's set to an empty array.

Challenge Seed

```
class DisplayMessages extends React.Component {
  // change code below this line

  // change code above this line
  render() {
    return <div>
  }
};
```

After Test

```
ReactDOM.render(<DisplayMessages />, document.getElementById('root'))
```

Solution

```
class DisplayMessages extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      input: '',
      messages: []
    }
  }
  render() {
    return <div>/>
  }
};
```

2. Manage State Locally First

Description

Here you'll finish creating the `DisplayMessages` component.

Instructions

First, in the `render()` method, have the component render an `input` element, `button` element, and `ul` element. When the `input` element changes, it should trigger a `handleChange()` method. Also, the `input` element should render the value of `input` that's in the component's state. The `button` element should trigger a `submitMessage()` method when it's clicked. Second, write these two methods. The `handleChange()` method should update the `input` with what the user is typing. The `submitMessage()` method should concatenate the current message (stored in `input`) to the `messages` array in local state, and clear the value of the `input`. Finally, use the `ul` to map over the array of `messages` and render it to the screen as a list of `li` elements.

Challenge Seed

```
class DisplayMessages extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      input: '',
      messages: []
    }
  }
};
```

```
// add handleChange() and submitMessage() methods here

render() {
  return (
    <div>
      <h2>Type in a new Message:</h2>
      { /* render an input, button, and ul here */ }

      { /* change code above this line */ }
    </div>
  );
}

};
```

After Test

```
ReactDOM.render(<DisplayMessages />, document.getElementById('root'))
```

Solution

```
class DisplayMessages extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      input: '',
      messages: []
    }
  this.handleChange = this.handleChange.bind(this);
  this.submitMessage = this.submitMessage.bind(this);
}
handleChange(event) {
  this.setState({
    input: event.target.value
  });
}
submitMessage() {
  const currentMessage = this.state.input;
  this.setState({
    input: '',
    messages: this.state.messages.concat(currentMessage)
  });
}
render() {
  return (
    <div>
      <h2>Type in a new Message:</h2>
      <input
        value={this.state.input}
        onChange={this.handleChange}/><br />
      <button onClick={this.submitMessage}>Submit</button>
      <ul>
        {this.state.messages.map( (message, idx) => {
          return (
            <li key={idx}>{message}</li>
          )
        })
      }
      </ul>
    </div>
  );
}
};
```

3. Extract State Logic to Redux

Description

Now that you finished the React component, you need to move the logic it's performing locally in its state into Redux. This is the first step to connect the simple React app to Redux. The only functionality your app has is to add new messages from the user to an unordered list. The example is simple in order to demonstrate how React and Redux work together.

Instructions

First, define an action type 'ADD' and set it to a const ADD . Next, define an action creator addMessage() which creates the action to add a message. You'll need to pass a message to this action creator and include the message in the returned action . Then create a reducer called messageReducer() that handles the state for the messages. The initial state should equal an empty array. This reducer should add a message to the array of messages held in state, or return the current state. Finally, create your Redux store and pass it the reducer.

Challenge Seed

```
// define ADD, addMessage(), messageReducer(), and store here:
```

Solution

```
const ADD = 'ADD';

const addMessage = (message) => {
  return {
    type: ADD,
    message
  };
};

const messageReducer = (state = [], action) => {
  switch (action.type) {
    case ADD:
      return [
        ...state,
        action.message
      ];
    default:
      return state;
  }
};

const store = Redux.createStore(messageReducer);
```

4. Use Provider to Connect Redux to React

Description

In the last challenge, you created a Redux store to handle the messages array and created an action for adding new messages. The next step is to provide React access to the Redux store and the actions it needs to dispatch updates. React Redux provides its react-redux package to help accomplish these tasks. React Redux provides a small API with two key features: Provider and connect . Another challenge covers connect . The Provider is a wrapper component from React Redux that wraps your React app. This wrapper then allows you to access the Redux store and dispatch functions throughout your component tree. Provider takes two props, the Redux store and the child components of your app. Defining the Provider for an App component might look like this:

```
<Provider store={store}>
  <App/>
</Provider>
```

Instructions

The code editor now shows all your Redux and React code from the past several challenges. It includes the Redux store, actions, and the `DisplayMessages` component. The only new piece is the `AppWrapper` component at the bottom. Use this top level component to render the `Provider` from `ReactRedux`, and pass the Redux store as a prop. Then render the `DisplayMessages` component as a child. Once you are finished, you should see your React component rendered to the page. **Note:** React Redux is available as a global variable here, so you can access the `Provider` with dot notation. The code in the editor takes advantage of this and sets it to a constant `Provider` for you to use in the `AppWrapper` render method.

Challenge Seed

```
// Redux Code:
const ADD = 'ADD';

const addMessage = (message) => {
  return {
    type: ADD,
    message
  }
};

const messageReducer = (state = [], action) => {
  switch (action.type) {
    case ADD:
      return [
        ...state,
        action.message
      ];
    default:
      return state;
  }
};

const store = Redux.createStore(messageReducer);

// React Code:

class DisplayMessages extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      input: '',
      messages: []
    }
    this.handleChange = this.handleChange.bind(this);
    this.submitMessage = this.submitMessage.bind(this);
  }
  handleChange(event) {
    this.setState({
      input: event.target.value
    });
  }
  submitMessage() {
    const currentMessage = this.state.input;
    this.setState({
      input: '',
      messages: this.state.messages.concat(currentMessage)
    });
  }
  render() {
    return (
      <div>
        <h2>Type in a new Message:</h2>
        <input
          value={this.state.input}
          onChange={this.handleChange}><br/>
        <button onClick={this.submitMessage}>Submit</button>
        <ul>
          {this.state.messages.map( (message, idx) => {
            return (
              <li key={idx}>{message}</li>
            )
          })
        
```

```

        })
    }
  </ul>
</div>
);
}

const Provider = Redux.Provider;

class AppWrapper extends React.Component {
// render the Provider here

// change code above this line
};

```

After Test

```
ReactDOM.render(<AppWrapper />, document.getElementById('root'))
```

Solution

```

// Redux Code:
const ADD = 'ADD';

const addMessage = (message) => {
  return {
    type: ADD,
    message
  };
};

const messageReducer = (state = [], action) => {
  switch (action.type) {
    case ADD:
      return [
        ...state,
        action.message
      ];
    default:
      return state;
  }
};

const store = Redux.createStore(messageReducer);

// React Code:

class DisplayMessages extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      input: '',
      messages: []
    }
    this.handleChange = this.handleChange.bind(this);
    this.submitMessage = this.submitMessage.bind(this);
  }
  handleChange(event) {
    this.setState({
      input: event.target.value
    });
  }
  submitMessage() {
    const currentMessage = this.state.input;
    this.setState({
      input: '',
      messages: this.state.messages.concat(currentMessage)
    });
  }
  render() {
    return (

```

```

<div>
  <h2>Type in a new Message:</h2>
  <input
    value={this.state.input}
    onChange={this.handleChange}><br/>
  <button onClick={this.submitMessage}>Submit</button>
  <ul>
    {this.state.messages.map( (message, idx) => {
      return (
        <li key={idx}>{message}</li>
      )
    })
  </ul>
</div>
);
}

const Provider = Redux.Provider;

class AppWrapper extends React.Component {
// change code below this line
render() {
  return (
    <Provider store = {store}>
      <DisplayMessages/>
    </Provider>
  );
}
// change code above this line
};

```

5. Map State to Props

Description

The Provider component allows you to provide state and dispatch to your React components, but you must specify exactly what state and actions you want. This way, you make sure that each component only has access to the state it needs. You accomplish this by creating two functions: `mapStateToProps()` and `mapDispatchToProps()`. In these functions, you declare what pieces of state you want to have access to and which action creators you need to be able to dispatch. Once these functions are in place, you'll see how to use the `React Redux connect` method to connect them to your components in another challenge. **Note:** Behind the scenes, React Redux uses the `store.subscribe()` method to implement `mapStateToProps()`.

Instructions

Create a function `mapStateToProps()`. This function should take `state` as an argument, then return an object which maps that state to specific property names. These properties will become accessible to your component via `props`. Since this example keeps the entire state of the app in a single array, you can pass that entire state to your component. Create a property `messages` in the object that's being returned, and set it to `state`.

Challenge Seed

```

const state = [];

// change code below this line

```

Solution

```

const state = [];

// change code below this line

```

```
const mapStateToProps = (state) => {
  return {
    messages: state
  };
};
```

6. Map Dispatch to Props

Description

The `mapDispatchToProps()` function is used to provide specific action creators to your React components so they can dispatch actions against the Redux store. It's similar in structure to the `mapStateToProps()` function you wrote in the last challenge. It returns an object that maps dispatch actions to property names, which become component props. However, instead of returning a piece of `state`, each property returns a function that calls `dispatch` with an action creator and any relevant action data. You have access to this `dispatch` because it's passed in to `mapDispatchToProps()` as a parameter when you define the function, just like you passed `state` to `mapStateToProps()`. Behind the scenes, React Redux is using Redux's `store.dispatch()` to conduct these dispatches with `mapDispatchToProps()`. This is similar to how it uses `store.subscribe()` for components that are mapped to `state`. For example, you have a `loginUser()` action creator that takes a `username` as an action payload. The object returned from `mapDispatchToProps()` for this action creator would look something like:

```
{
  submitLoginUser: function(username) {
    dispatch(loginUser(username));
  }
}
```

Instructions

The code editor provides an action creator called `addMessage()`. Write the function `mapDispatchToProps()` that takes `dispatch` as an argument, then returns an object. The object should have a property `submitNewMessage` set to the `dispatch` function, which takes a parameter for the new message to add when it dispatches `addMessage()`.

Challenge Seed

```
const addMessage = (message) => {
  return {
    type: 'ADD',
    message: message
  };
};

// change code below this line
```

Solution

```
const addMessage = (message) => {
  return {
    type: 'ADD',
    message: message
  };
};

// change code below this line

const mapDispatchToProps = (dispatch) => {
  return {
    submitNewMessage: function(message) {
      dispatch(addMessage(message));
    }
  };
}
```

```

    }
};
```

7. Connect Redux to React

Description

Now that you've written both the `mapStateToProps()` and the `mapDispatchToProps()` functions, you can use them to map state and dispatch to the props of one of your React components. The `connect` method from React Redux can handle this task. This method takes two optional arguments, `mapStateToProps()` and `mapDispatchToProps()`. They are optional because you may have a component that only needs access to state but doesn't need to dispatch any actions, or vice versa. To use this method, pass in the functions as arguments, and immediately call the result with your component. This syntax is a little unusual and looks like: `connect(mapStateToProps, mapDispatchToProps)(MyComponent)` **Note:** If you want to omit one of the arguments to the `connect` method, you pass `null` in its place.

Instructions

The code editor has the `mapStateToProps()` and `mapDispatchToProps()` functions and a new React component called `Presentational`. Connect this component to Redux with the `connect` method from the `ReactRedux` global object, and call it immediately on the `Presentational` component. Assign the result to a new `const` called `ConnectedComponent` that represents the connected component. That's it, now you're connected to Redux! Try changing either of `connect`'s arguments to `null` and observe the test results.

Challenge Seed

```

const addMessage = (message) => {
  return {
    type: 'ADD',
    message: message
  };
}

const mapStateToProps = (state) => {
  return {
    messages: state
  };
};

const mapDispatchToProps = (dispatch) => {
  return {
    submitNewMessage: (message) => {
      dispatch(addMessage(message));
    }
  };
};

class Presentational extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return <h3>This is a Presentational Component</h3>
  }
};

const connect = ReactRedux.connect;
// change code below this line
```

After Test

```

const store = Redux.createStore(
  (state = '__INITIAL__STATE__', action) => state
);
```

```

class AppWrapper extends React.Component {
  render() {
    return (
      <ReactRedux.Provider store = {store}>
        <ConnectedComponent/>
      </ReactRedux.Provider>
    );
  }
}
ReactDOM.render(<AppWrapper />, document.getElementById('root'))

```

Solution

```

const addMessage = (message) => {
  return {
    type: 'ADD',
    message: message
  };
};

const mapStateToProps = (state) => {
  return {
    messages: state
  };
};

const mapDispatchToProps = (dispatch) => {
  return {
    submitNewMessage: (message) => {
      dispatch(addMessage(message));
    }
  };
};

class Presentational extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return <h3>This is a Presentational Component</h3>
  }
};

const connect = ReactRedux.connect;
// change code below this line

const ConnectedComponent = connect(mapStateToProps, mapDispatchToProps)(Presentational);

```

8. Connect Redux to the Messages App

Description

Now that you understand how to use `connect` to connect React to Redux, you can apply what you've learned to your React component that handles messages. In the last lesson, the component you connected to Redux was named `Presentational`, and this wasn't arbitrary. This term *generally* refers to React components that are not directly connected to Redux. They are simply responsible for the presentation of UI and do this as a function of the props they receive. By contrast, container components are connected to Redux. These are typically responsible for dispatching actions to the store and often pass store state to child components as props.

Instructions

The code editor has all the code you've written in this section so far. The only change is that the React component is renamed to `Presentational`. Create a new component held in a constant called `Container` that uses `connect` to connect the `Presentational` component to Redux. Then, in the `AppWrapper`, render the React Redux Provider

component. Pass `Provider` the Redux store as a prop and render `Container` as a child. Once everything is setup, you will see the messages app rendered to the page again.

Challenge Seed

```
// Redux:
const ADD = 'ADD';

const addMessage = (message) => {
  return {
    type: ADD,
    message: message
  }
};

const messageReducer = (state = [], action) => {
  switch (action.type) {
    case ADD:
      return [
        ...state,
        action.message
      ];
    default:
      return state;
  }
};

const store = Redux.createStore(messageReducer);

// React:
class Presentational extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      input: '',
      messages: []
    }
    this.handleChange = this.handleChange.bind(this);
    this.submitMessage = this.submitMessage.bind(this);
  }
  handleChange(event) {
    this.setState({
      input: event.target.value
    });
  }
  submitMessage() {
    const currentMessage = this.state.input;
    this.setState({
      input: '',
      messages: this.state.messages.concat(currentMessage)
    });
  }
  render() {
    return (
      <div>
        <h2>Type in a new Message:</h2>
        <input
          value={this.state.input}
          onChange={this.handleChange}><br/>
        <button onClick={this.submitMessage}>Submit</button>
        <ul>
          {this.state.messages.map( (message, idx) => {
            return (
              <li key={idx}>{message}</li>
            )
          })
        }
        </ul>
      </div>
    );
  }
}

// React-Redux:
const mapStateToProps = (state) => {
```

```

    return { messages: state }
};

const mapDispatchToProps = (dispatch) => {
  return {
    submitNewMessage: (newMessage) => {
      dispatch(addMessage(newMessage))
    }
  };
};

const Provider = Redux.Provider;
const connect = Redux.connect;

// define the Container component here:

class AppWrapper extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    // complete the return statement:
    return (null);
  }
};

```

After Test

```
ReactDOM.render(<AppWrapper />, document.getElementById('root'))
```

Solution

```

// Redux:
const ADD = 'ADD';

const addMessage = (message) => {
  return {
    type: ADD,
    message: message
  };
};

const messageReducer = (state = [], action) => {
  switch (action.type) {
    case ADD:
      return [
        ...state,
        action.message
      ];
    default:
      return state;
  }
};

const store = Redux.createStore(messageReducer);

// React:
class Presentational extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      input: '',
      messages: []
    };
  }
  this.handleChange = this.handleChange.bind(this);
  this.submitMessage = this.submitMessage.bind(this);
}
handleChange(event) {
  this.setState({
    input: event.target.value
  });
}

```

```

    }
    submitMessage() {
      const currentMessage = this.state.input;
      this.setState({
        input: '',
        messages: this.state.messages.concat(currentMessage)
      });
    }
    render() {
      return (
        <div>
          <h2>Type in a new Message:</h2>
          <input
            value={this.state.input}
            onChange={this.handleChange}/><br/>
          <button onClick={this.submitMessage}>Submit</button>
          <ul>
            {this.state.messages.map( (message, idx) => {
              return (
                <li key={idx}>{message}</li>
              )
            })
          </ul>
        </div>
      );
    }
  };

// React-Redux:
const mapStateToProps = (state) => {
  return { messages: state }
};

const mapDispatchToProps = (dispatch) => {
  return {
    submitNewMessage: (newMessage) => {
      dispatch(addMessage(newMessage))
    }
  }
};

const Provider = Redux.Provider;
const connect = Redux.connect;

// define the Container component here:
const Container = connect(mapStateToProps, mapDispatchToProps)(Presentational);

class AppWrapper extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    // complete the return statement:
    return (
      <Provider store={store}>
        <Container/>
      </Provider>
    );
  }
};

```

9. Extract Local State into Redux

Description

You're almost done! Recall that you wrote all the Redux code so that Redux could control the state management of your React messages app. Now that Redux is connected, you need to extract the state management out of the `Presentational` component and into Redux. Currently, you have Redux connected, but you are handling the state locally within the `Presentational` component.

Instructions

In the Presentational component, first, remove the `messages` property in the local `state`. These messages will be managed by Redux. Next, modify the `submitMessage()` method so that it dispatches `submitNewMessage()` from `this.props`, and pass in the current message input from local `state` as an argument. Because you removed `messages` from local state, remove the `messages` property from the call to `this.setState()` here as well. Finally, modify the `render()` method so that it maps over the messages received from `props` rather than `state`. Once these changes are made, the app will continue to function the same, except Redux manages the state. This example also illustrates how a component may have local `state`: your component still tracks user input locally in its own `state`. You can see how Redux provides a useful state management framework on top of React. You achieved the same result using only React's local state at first, and this is usually possible with simple apps. However, as your apps become larger and more complex, so does your state management, and this is the problem Redux solves.

Challenge Seed

```
// Redux:
const ADD = 'ADD';

const addMessage = (message) => {
  return {
    type: ADD,
    message: message
  }
};

const messageReducer = (state = [], action) => {
  switch (action.type) {
    case ADD:
      return [
        ...state,
        action.message
      ];
    default:
      return state;
  }
};

const store = Redux.createStore(messageReducer);

// React:
const Provider = ReactRedux.Provider;
const connect = ReactRedux.connect;

// Change code below this line
class Presentational extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      input: '',
      messages: []
    }
    this.handleChange = this.handleChange.bind(this);
    this.submitMessage = this.submitMessage.bind(this);
  }
  handleChange(event) {
    this.setState({
      input: event.target.value
    });
  }
  submitMessage() {
    this.setState({
      input: '',
      messages: this.state.messages.concat(this.state.input)
    });
  }
  render() {
    return (
      <div>
        <h2>Type in a new Message:</h2>
        <input
          value={this.state.input}
          onChange={this.handleChange}/><br/>
    
```

```

        <button onClick={this.submitMessage}>Submit</button>
        <ul>
          {this.state.messages.map( (message, idx) => {
            return (
              <li key={idx}>{message}</li>
            )
          })
        </ul>
      </div>
    );
  }
};

// Change code above this line

const mapStateToProps = (state) => {
  return {messages: state}
};

const mapDispatchToProps = (dispatch) => {
  return {
    submitNewMessage: (message) => {
      dispatch(addMessage(message))
    }
  }
};

const Container = connect(mapStateToProps, mapDispatchToProps)(Presentational);

class AppWrapper extends React.Component {
  render() {
    return (
      <Provider store={store}>
        <Container/>
      </Provider>
    );
  }
};

```

After Test

```
ReactDOM.render(<AppWrapper />, document.getElementById('root'))
```

Solution

```

// Redux:
const ADD = 'ADD';

const addMessage = (message) => {
  return {
    type: ADD,
    message: message
  }
};

const messageReducer = (state = [], action) => {
  switch (action.type) {
    case ADD:
      return [
        ...state,
        action.message
      ];
    default:
      return state;
  }
};

const store = Redux.createStore(messageReducer);

// React:
const Provider = ReactRedux.Provider;
const connect = ReactRedux.connect;

```

```
// Change code below this line
class Presentational extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      input: ''
    }
  this.handleChange = this.handleChange.bind(this);
  this.submitMessage = this.submitMessage.bind(this);
}
handleChange(event) {
  this.setState({
    input: event.target.value
  });
}
submitMessage() {
  this.props.submitNewMessage(this.state.input);
  this.setState({
    input: ''
  });
}
render() {
  return (
    <div>
      <h2>Type in a new Message:</h2>
      <input
        value={this.state.input}
        onChange={this.handleChange}/><br/>
      <button onClick={this.submitMessage}>Submit</button>
      <ul>
        {this.props.messages.map( (message, idx) => {
          return (
            <li key={idx}>{message}</li>
          )
        })
      </ul>
    </div>
  );
}
// Change code above this line

const mapStateToProps = (state) => {
  return {messages: state}
};

const mapDispatchToProps = (dispatch) => {
  return {
    submitNewMessage: (message) => {
      dispatch(addMessage(message))
    }
  }
};

const Container = connect(mapStateToProps, mapDispatchToProps)(Presentational);

class AppWrapper extends React.Component {
  render() {
    return (
      <Provider store={store}>
        <Container/>
      </Provider>
    );
  }
};
```

10. Moving Forward From Here

Description

Congratulations! You finished the lessons on React and Redux. There's one last item worth pointing out before you move on. Typically, you won't write React apps in a code editor like this. This challenge gives you a glimpse of what the syntax looks like if you're working with npm and a file system on your own machine. The code should look similar, except for the use of `import` statements (these pull in all of the dependencies that have been provided for you in the challenges). The "Managing Packages with npm" section covers npm in more detail. Finally, writing React and Redux code generally requires some configuration. This can get complicated quickly. If you are interested in experimenting on your own machine, the [Create React App](#) comes configured and ready to go. Alternatively, you can enable Babel as a JavaScript Preprocessor in CodePen, add React and ReactDOM as external JavaScript resources, and work there as well.

Instructions

Log the message 'Now I know React and Redux!' to the console.

Challenge Seed

```
// import React from 'react'
// import ReactDOM from 'react-dom'
// import { Provider, connect } from 'react-redux'
// import { createStore, combineReducers, applyMiddleware } from 'redux'
// import thunk from 'redux-thunk'

// import rootReducer from './redux/reducers'
// import App from './components/App'

// const store = createStore(
//   rootReducer,
//   applyMiddleware(thunk)
// );

// ReactDOM.render(
//   <Provider store={store}>
//     <App/>
//   </Provider>,
//   document.getElementById('root')
// );

// change code below this line
```

Solution

```
console.log('Now I know React and Redux!');
```

Front End Libraries Projects

1. Build a Random Quote Machine

Description

Objective: Build a [CodePen.io](#) app that is functionally similar to this: <https://codepen.io/freeCodeCamp/full/qRZeGZ>. Fulfill the below [user stories](#) and get all of the tests to pass. Give it your own personal style. You can use any mix of HTML, JavaScript, CSS, Bootstrap, SASS, React, Redux, and jQuery to complete this project. You should use a frontend framework (like React for example) because this section is about learning frontend frameworks. Additional technologies not listed above are not recommended and using them is at your own risk. We are looking at supporting other frontend frameworks like Angular and Vue, but they are not currently supported. We will accept and try to fix all issue reports that use the suggested technology stack for this project. Happy coding!

User Story #1: I can see a wrapper element with a corresponding `id="quote-box"`.

User Story #2: Within `#quote-box`, I can see an element with a corresponding `id="text"`.

User Story #3: Within `#quote-box`, I can see an element with a corresponding `id="author"`.

User Story #4: Within `#quote-box`, I can see a clickable element with a corresponding `id="new-quote"`.

User Story #5: Within `#quote-box`, I can see a clickable element with a corresponding `id="tweet-quote"`. **User Story #6:** On first load, my quote machine displays a random quote in the element with `id="text"`. **User Story #7:** On first load, my quote machine displays the random quote's author in the element with `id="author"`. **User Story #8:** When the `#new-quote` button is clicked, my quote machine should fetch a new quote and display it in the `#text` element. **User Story #9:** My quote machine should fetch the new quote's author when the `#new-quote` button is clicked and display it in the `#author` element. **User Story #10:** I can tweet the current quote by clicking on the `#tweet-quote` a element. This a element should include the "twitter.com/intent/tweet" path in its `href` attribute to tweet the current quote. **User Story #11:** The `#quote-box` wrapper element should be horizontally centered. Please run tests with browser's zoom level at 100% and page maximized. You can build your project by forking [this CodePen pen](#). Or you can use this CDN link to run the tests in any environment you like: <https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js> Once you're done, submit the URL to your working project with all its tests passing. Remember to use the [Read-Search-Ask](#) method if you get stuck.

Instructions

Challenge Seed

Solution

```
// solution required
```

2. Build a Markdown Previewer

Description

Objective: Build a [CodePen.io](#) app that is functionally similar to this: <https://codepen.io/freeCodeCamp/full/GrZVW0>. Fulfill the below [user stories](#) and get all of the tests to pass. Give it your own personal style. You can use any mix of HTML, JavaScript, CSS, Bootstrap, SASS, React, Redux, and jQuery to complete this project. You should use a frontend framework (like React for example) because this section is about learning frontend frameworks. Additional technologies not listed above are not recommended and using them is at your own risk. We are looking at supporting other frontend frameworks like Angular and Vue, but they are not currently supported. We will accept and try to fix all issue reports that use the suggested technology stack for this project. Happy coding! **User Story #1:** I can see a `textarea` element with a corresponding `id="editor"`. **User Story #2:** I can see an element with a corresponding `id="preview"`. **User Story #3:** When I enter text into the `#editor` element, the `#preview` element is updated as I type to display the content of the `textarea`. **User Story #4:** When I enter GitHub flavored markdown into the `#editor` element, the text is rendered as HTML in the `#preview` element as I type (HINT: You don't need to parse Markdown yourself - you can import the Marked library for this: <https://cdnjs.com/libraries/marked>). **User Story #5:** When my markdown previewer first loads, the default text in the `#editor` field should contain valid markdown that represents at least one of each of the following elements: a header (H1 size), a sub header (H2 size), a link, inline code, a code block, a list item, a blockquote, an image, and bolded text. **User Story #6:** When my markdown previewer first loads, the default markdown in the `#editor` field should be rendered as HTML in the `#preview` element. **Optional Bonus (you do not need to make this test pass):** My markdown previewer interprets carriage returns and renders them as `br` (line break) elements. You can build your project by forking [this CodePen pen](#). Or you can use this CDN link to run the tests in any environment you like: <https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js> Once you're done, submit the URL to your working project with all its tests passing. Remember to use the [Read-Search-Ask](#) method if you get stuck.

Instructions

Challenge Seed

Solution

```
// solution required
```

3. Build a Drum Machine

Description

Objective: Build a [CodePen.io](https://codepen.io/freeCodeCamp/full/MJyNMd) app that is functionally similar to this: <https://codepen.io/freeCodeCamp/full/MJyNMd>. Fulfill the below [user stories](#) and get all of the tests to pass. Give it your own personal style. You can use any mix of HTML, JavaScript, CSS, Bootstrap, SASS, React, Redux, and jQuery to complete this project. You should use a frontend framework (like React for example) because this section is about learning frontend frameworks. Additional technologies not listed above are not recommended and using them is at your own risk. We are looking at supporting other frontend frameworks like Angular and Vue, but they are not currently supported. We will accept and try to fix all issue reports that use the suggested technology stack for this project. Happy coding! **User Story #1:** I should be able to see an outer container with a corresponding `id="drum-machine"` that contains all other elements. **User Story #2:** Within `#drum-machine` I can see an element with a corresponding `id="display"`. **User Story #3:** Within `#drum-machine` I can see 9 clickable drum pad elements, each with a class name of `drum-pad`, a unique id that describes the audio clip the drum pad will be set up to trigger, and an inner text that corresponds to one of the following keys on the keyboard: Q, W, E, A, S, D, Z, X, C. The drum pads MUST be in this order. **User Story #4:** Within each `.drum-pad`, there should be an HTML5 `audio` element which has a `src` attribute pointing to an audio clip, a class name of `clip`, and an id corresponding to the inner text of its parent `.drum-pad` (e.g. `id="Q"`, `id="W"`, `id="E"` etc.). **User Story #5:** When I click on a `.drum-pad` element, the audio clip contained in its child `audio` element should be triggered. **User Story #6:** When I press the trigger key associated with each `.drum-pad`, the audio clip contained in its child `audio` element should be triggered (e.g. pressing the Q key should trigger the drum pad which contains the string "Q", pressing the W key should trigger the drum pad which contains the string "W", etc.). **User Story #7:** When a `.drum-pad` is triggered, a string describing the associated audio clip is displayed as the inner text of the `#display` element (each string must be unique). You can build your project by forking [this CodePen pen](#). Or you can use this CDN link to run the tests in any environment you like: <https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js>. Once you're done, submit the URL to your working project with all its tests passing. Remember to use the [Read-Search-Ask](#) method if you get stuck.

Instructions

Challenge Seed

Solution

```
// solution required
```

4. Build a JavaScript Calculator

Description

Objective: Build a [CodePen.io](https://codepen.io/freeCodeCamp/full/wgGVVX) app that is functionally similar to this: <https://codepen.io/freeCodeCamp/full/wgGVVX>. Fulfill the below [user stories](#) and get all of the tests to pass. Give it your own personal style. You can use any mix of HTML, JavaScript, CSS, Bootstrap, SASS, React, Redux, and jQuery to complete this project. You should use a frontend framework (like React for example) because this section is about learning frontend frameworks. Additional technologies not listed above are not recommended and using them is at your own risk. We are looking at supporting other frontend frameworks like Angular and Vue, but they are not currently supported. We will accept and try to fix all issue reports that use the suggested technology stack for this project. Happy coding! **User Story #1:** My calculator should contain a clickable element containing an = (equal sign) with a corresponding `id="equals"`. **User Story #2:** My calculator should contain 10 clickable elements containing one number each from 0-9, with the following corresponding IDs: `id="zero"`, `id="one"`, `id="two"`, `id="three"`, `id="four"`, `id="five"`, `id="six"`, `id="seven"`, `id="eight"`, and `id="nine"`. **User Story #3:** My calculator should contain 4 clickable elements each containing one of the 4 primary mathematical operators with the following corresponding IDs: `id="add"`, `id="subtract"`,

`id="multiply"` , `id="divide"` . **User Story #4:** My calculator should contain a clickable element containing a `.` (decimal point) symbol with a corresponding `id="decimal"` . **User Story #5:** My calculator should contain a clickable element with an `id="clear"` . **User Story #6:** My calculator should contain an element to display values with a corresponding `id="display"` . **User Story #7:** At any time, pressing the clear button clears the input and output values, and returns the calculator to its initialized state; 0 should be shown in the element with the id of `display` . **User Story #8:** As I input numbers, I should be able to see my input in the element with the id of `display` . **User Story #9:** In any order, I should be able to add, subtract, multiply and divide a chain of numbers of any length, and when I hit `=` , the correct result should be shown in the element with the id of `display` . **User Story #10:** When inputting numbers, my calculator should not allow a number to begin with multiple zeros. **User Story #11:** When the decimal element is clicked, a `.` should append to the currently displayed value; two `.` in one number should not be accepted. **User Story #12:** I should be able to perform any operation `(+, -, *, /)` on numbers containing decimal points. **User Story #13:** If 2 or more operators are entered consecutively, the operation performed should be the last operator entered. **User Story #14:** Pressing an operator immediately following `=` should start a new calculation that operates on the result of the previous evaluation. **User Story #15:** My calculator should have several decimal places of precision when it comes to rounding (note that there is no exact standard, but you should be able to handle calculations like `2 / 7` with reasonable precision to at least 4 decimal places). **Note On Calculator Logic:** It should be noted that there are two main schools of thought on calculator input logic: immediate execution logic and formula logic. Our example utilizes formula logic and observes order of operation precedence, immediate execution does not. Either is acceptable, but please note that depending on which you choose, your calculator may yield different results than ours for certain equations (see below example). As long as your math can be verified by another production calculator, please do not consider this a bug. **EXAMPLE:** `3 + 5 x 6 - 2 / 4 =`

- **Immediate Execution Logic:** 11.5
- **Formula/Expression Logic:** 32.5

You can build your project by forking [this CodePen pen](#). Or you can use this CDN link to run the tests in any environment you like: <https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js> Once you're done, submit the URL to your working project with all its tests passing. Remember to use the [Read-Search-Ask](#) method if you get stuck.

Instructions

Challenge Seed

Solution

```
// solution required
```

5. Build a Pomodoro Clock

Description

Objective: Build a [CodePen.io](#) app that is functionally similar to this: <https://codepen.io/freeCodeCamp/full/XpKrrW>. Fulfill the below [user stories](#) and get all of the tests to pass. Give it your own personal style. You can use any mix of HTML, JavaScript, CSS, Bootstrap, SASS, React, Redux, and jQuery to complete this project. You should use a frontend framework (like React for example) because this section is about learning frontend frameworks. Additional technologies not listed above are not recommended and using them is at your own risk. We are looking at supporting other frontend frameworks like Angular and Vue, but they are not currently supported. We will accept and try to fix all issue reports that use the suggested technology stack for this project. Happy coding! **User Story #1:** I can see an element with `id="break-label"` that contains a string (e.g. "Break Length"). **User Story #2:** I can see an element with `id="session-label"` that contains a string (e.g. "Session Length"). **User Story #3:** I can see two clickable elements with corresponding IDs: `id="break-decrement"` and `id="session-decrement"` . **User Story #4:** I can see two clickable elements with corresponding IDs: `id="break-increment"` and `id="session-increment"` . **User Story #5:** I can see an element with a corresponding `id="break-length"` , which by default (on load) displays a value of 5. **User Story #6:** I can see an element with a corresponding `id="session-length"` , which by default displays a value of 25. **User Story #7:** I can see an element with a corresponding `id="timer-label"` , that contains a string indicating a session is initialized (e.g. "Session"). **User Story #8:** I can see an element with corresponding `id="time-left"` . **NOTE:** Paused or

running, the value in this field should always be displayed in `mm:ss` format (i.e. 25:00). **User Story #9:** I can see a clickable element with a corresponding `id="start_stop"`. **User Story #10:** I can see a clickable element with a corresponding `id="reset"`. **User Story #11:** When I click the element with the id of `reset`, any running timer should be stopped, the value within `id="break-length"` should return to 5, the value within `id="session-length"` should return to 25, and the element with `id="time-left"` should reset to its default state. **User Story #12:** When I click the element with the id of `break-decrement`, the value within `id="break-length"` decrements by a value of 1, and I can see the updated value. **User Story #13:** When I click the element with the id of `break-increment`, the value within `id="break-length"` increments by a value of 1, and I can see the updated value. **User Story #14:** When I click the element with the id of `session-decrement`, the value within `id="session-length"` decrements by a value of 1, and I can see the updated value. **User Story #15:** When I click the element with the id of `session-increment`, the value within `id="session-length"` increments by a value of 1, and I can see the updated value. **User Story #16:** I should not be able to set a session or break length to ≤ 0 . **User Story #17:** I should not be able to set a session or break length to > 60 . **User Story #18:** When I first click the element with `id="start_stop"`, the timer should begin running from the value currently displayed in `id="session-length"`, even if the value has been incremented or decremented from the original value of 25. **User Story #19:** If the timer is running, the element with the id of `time-left` should display the remaining time in `mm:ss` format (decrementing by a value of 1 and updating the display every 1000ms). **User Story #20:** If the timer is running and I click the element with `id="start_stop"`, the countdown should pause. **User Story #21:** If the timer is paused and I click the element with `id="start_stop"`, the countdown should resume running from the point at which it was paused. **User Story #22:** When a session countdown reaches zero (NOTE: timer MUST reach 00:00), and a new countdown begins, the element with the id of `timer-label` should display a string indicating a break has begun. **User Story #23:** When a session countdown reaches zero (NOTE: timer MUST reach 00:00), a new break countdown should begin, counting down from the value currently displayed in the `id="break-length"` element. **User Story #24:** When a break countdown reaches zero (NOTE: timer MUST reach 00:00), and a new countdown begins, the element with the id of `timer-label` should display a string indicating a session has begun. **User Story #25:** When a break countdown reaches zero (NOTE: timer MUST reach 00:00), a new session countdown should begin, counting down from the value currently displayed in the `id="session-length"` element. **User Story #26:** When a countdown reaches zero (NOTE: timer MUST reach 00:00), a sound indicating that time is up should play. This should utilize an HTML5 `audio` tag and have a corresponding `id="beep"`. **User Story #27:** The audio element with `id="beep"` must be 1 second or longer. **User Story #28:** The audio element with id of `beep` must stop playing and be rewound to the beginning when the element with the id of `reset` is clicked. You can build your project by forking [this CodePen pen](#). Or you can use this CDN link to run the tests in any environment you like:
<https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js> Once you're done, submit the URL to your working project with all its tests passing. Remember to use the [Read-Search-Ask](#) method if you get stuck.

Instructions

Challenge Seed

Solution

```
// solution required
```

Data Visualization Certification

Data Visualization with D3

1. Add Document Elements with D3

Description

D3 has several methods that let you add and change elements in your document. The `select()` method selects one element from the document. It takes an argument for the name of the element you want and returns an HTML node for the first element in the document that matches the name. Here's an example: `const anchor = d3.select("a");` The above example finds the first anchor tag on the page and saves an HTML node for it in the variable `anchor`. You can use the selection with other methods. The "d3" part of the example is a reference to the D3 object, which is how you access D3 methods. Two other useful methods are `append()` and `text()`. The `append()` method takes an argument for the element you want to add to the document. It appends an HTML node to a selected item, and returns a handle to that node. The `text()` method either sets the text of the selected node, or gets the current text. To set the value, you pass a string as an argument inside the parentheses of the method. Here's an example that selects an unordered list, appends a list item, and adds text:

```
d3.select("ul")
.append("li")
.text("Very important item");
```

D3 allows you to chain several methods together with periods to perform a number of actions in a row.

Instructions

Use the `select` method to select the `body` tag in the document. Then append an `h1` tag to it, and add the text "Learning D3" into the `h1` element.

Challenge Seed

```
<body>
<script>
// Add your code below this line

// Add your code above this line
</script>
</body>
```

Solution

```
// solution required
```

2. Select a Group of Elements with D3

Description

D3 also has the `selectAll()` method to select a group of elements. It returns an array of HTML nodes for all the items in the document that match the input string. Here's an example to select all the anchor tags in a document: `const anchors = d3.selectAll("a");` Like the `select()` method, `selectAll()` supports method chaining, and you can use it with other methods.

Instructions

Select all of the `li` tags in the document, and change their text to "list item" by chaining the `.text()` method.

Challenge Seed

```
<body>
  <ul>
    <li>Example</li>
    <li>Example</li>
    <li>Example</li>
  </ul>
  <script>
    // Add your code below this line

    // Add your code above this line
  </script>
</body>
```

Solution

```
// solution required
```

3. Work with Data in D3

Description

The D3 library focuses on a data-driven approach. When you have a set of data, you can apply D3 methods to display it on the page. Data comes in many formats, but this challenge uses a simple array of numbers. The first step is to make D3 aware of the data. The `data()` method is used on a selection of DOM elements to attach the data to those elements. The data set is passed as an argument to the method. A common workflow pattern is to create a new element in the document for each piece of data in the set. D3 has the `enter()` method for this purpose. When `enter()` is combined with the `data()` method, it looks at the selected elements from the page and compares them to the number of data items in the set. If there are fewer elements than data items, it creates the missing elements. Here is an example that selects a `ul` element and creates a new list item based on the number of entries in the array:

```
<body>
  <ul></ul>
  <script>
    const dataset = ["a", "b", "c"];
    d3.select("ul").selectAll("li")
      .data(dataset)
      .enter()
      .append("li")
      .text("New item");
  </script>
</body>
```

It may seem confusing to select elements that don't exist yet. This code is telling D3 to first select the `ul` on the page. Next, select all list items, which returns an empty selection. Then the `data()` method reviews the dataset and runs the following code three times, once for each item in the array. The `enter()` method sees there are no `li` elements on the page, but it needs 3 (one for each piece of data in `dataset`). New `li` elements are appended to the `ul` and have the text "New item".

Instructions

Select the `body` node, then select all `h2` elements. Have D3 create and append an `h2` tag for each item in the `dataset` array. The text in the `h2` should say "New Title". Your code should use the `data()` and `enter()` methods.

Challenge Seed

```
<body>
  <script>
    const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

    // Add your code below this line

    // Add your code above this line
  </script>
</body>
```

Solution

```
// solution required
```

4. Work with Dynamic Data in D3

Description

The last two challenges cover the basics of displaying data dynamically with D3 using the `data()` and `enter()` methods. These methods take a data set and, together with the `append()` method, create a new DOM element for each entry in the data set. In the previous challenge, you created a new `h2` element for each item in the `dataset` array, but they all contained the same text, "New Title". This is because you have not made use of the data that is bound to each of the `h2` elements. The D3 `text()` method can take a string or a callback function as an argument: `selection.text((d) => d)` In the example above, the parameter `d` refers to a single entry in the dataset that a selection is bound to. Using the current example as context, the first `h2` element is bound to 12, the second `h2` element is bound to 31, the third `h2` element is bound to 22, and so on.

Instructions

Change the `text()` method so that each `h2` element displays the corresponding value from the `dataset` array with a single space and "USD". For example, the first heading should be "12 USD".

Challenge Seed

```
<body>
  <script>
    const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

    d3.select("body").selectAll("h2")
      .data(dataset)
      .enter()
      .append("h2")
      // Add your code below this line

      .text("New Title");

    // Add your code above this line
  </script>
</body>
```

Solution

```
// solution required
```

5. Add Inline Styling to Elements

Description

D3 lets you add inline CSS styles on dynamic elements with the `style()` method. The `style()` method takes a comma-separated key-value pair as an argument. Here's an example to set the selection's text color to blue:

```
selection.style("color", "blue");
```

Instructions

Add the `style()` method to the code in the editor to make all the displayed text have a `font-family` of `verdana`.

Challenge Seed

```
<body>
  <script>
    const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

    d3.select("body").selectAll("h2")
      .data(dataset)
      .enter()
      .append("h2")
      .text((d) => (d + " USD"))
    // Add your code below this line

    // Add your code above this line
  </script>
</body>
```

Solution

```
// solution required
```

6. Change Styles Based on Data

Description

D3 is about visualization and presentation of data. It's likely you'll want to change the styling of elements based on the data. You can use a callback function in the `style()` method to change the styling for different elements. For example, you may want to color a data point blue if has a value less than 20, and red otherwise. You can use a callback function in the `style()` method and include the conditional logic. The callback function uses the `d` parameter to represent the data point:

```
selection.style("color", (d) => {
  /* Logic that returns the color based on a condition */
});
```

The `style()` method is not limited to setting the `color` - it can be used with other CSS properties as well.

Instructions

Add the `style()` method to the code in the editor to set the `color` of the `h2` elements conditionally. Write the callback function so if the data value is less than 20, it returns "red", otherwise it returns "green". **Note**
You can use if-else logic, or the ternary operator.

Challenge Seed

```
<body>
<script>
  const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

  d3.select("body").selectAll("h2")
    .data(dataset)
    .enter()
    .append("h2")
    .text((d) => (d + " USD"))
  // Add your code below this line

  // Add your code above this line
</script>
</body>
```

Solution

```
// solution required
```

7. Add Classes with D3

Description

Using a lot of inline styles on HTML elements gets hard to manage, even for smaller apps. It's easier to add a class to elements and style that class one time using CSS rules. D3 has the `attr()` method to add any HTML attribute to an element, including a class name. The `attr()` method works the same way that `style()` does. It takes comma-separated values, and can use a callback function. Here's an example to add a class of "container" to a selection:

```
selection.attr("class", "container");
```

Note that the "class" parameter will remain the same whenever you need to add a class and only the "container" parameter will change.

Instructions

Add the `attr()` method to the code in the editor and put a class of `bar` on the `div` elements.

Challenge Seed

```
<style>
  .bar {
    width: 25px;
    height: 100px;
    display: inline-block;
    background-color: blue;
  }
</style>
<body>
<script>
  const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

  d3.select("body").selectAll("div")
    .data(dataset)
    .enter()
    .append("div")
```

```
// Add your code below this line
```

```
// Add your code above this line
</script>
</body>
```

Solution

```
<style>
  .bar {
    width: 25px;
    height: 100px;
    display: inline-block;
    background-color: blue;
  }
</style>
<body>
  <script>
    const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

    d3.select("body").selectAll("div")
      .data(dataset)
      .enter()
      .append("div")
    // Add your code below this line
      .attr("class", "bar");
    // Add your code above this line
  </script>
</body>
```

8. Update the Height of an Element Dynamically

Description

The previous challenges covered how to display data from an array and how to add CSS classes. You can combine these lessons to create a simple bar chart. There are two steps to this: 1) Create a `div` for each data point in the array 2) Give each `div` a dynamic height, using a callback function in the `style()` method that sets `height` equal to the data value Recall the format to set a style using a callback function: `selection.style("cssProperty", (d) => d)`

Instructions

Add the `style()` method to the code in the editor to set the `height` property for each element. Use a callback function to return the value of the data point with the string "px" added to it.

Challenge Seed

```
<style>
  .bar {
    width: 25px;
    height: 100px;
    display: inline-block;
    background-color: blue;
  }
</style>
<body>
  <script>
    const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

    d3.select("body").selectAll("div")
      .data(dataset)
      .enter()
      .append("div")
```

```
.attr("class", "bar")
// Add your code below this line

// Add your code above this line
</script>
</body>
```

Solution

```
// solution required
```

9. Change the Presentation of a Bar Chart

Description

The last challenge created a bar chart, but there are a couple of formatting changes that could improve it: 1) Add space between each bar to visually separate them, which is done by adding a margin to the CSS for the `bar` class 2) Increase the height of the bars to better show the difference in values, which is done by multiplying the value by a number to scale the height

Instructions

First, add a `margin` of `2px` to the `bar` class in the `style` tag. Next, change the callback function in the `style()` method so it returns a value 10 times the original data value (plus the "px"). **Note**
Multiplying each data point by the *same* constant only alters the scale. It's like zooming in, and it doesn't change the meaning of the underlying data.

Challenge Seed

```
<style>
  .bar {
    width: 25px;
    height: 100px;
    /* Add your code below this line */

    /* Add your code above this line */
    display: inline-block;
    background-color: blue;
  }
</style>
<body>
  <script>
    const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

    d3.select("body").selectAll("div")
      .data(dataset)
      .enter()
      .append("div")
      .attr("class", "bar")
      // Add your code below this line
      .style("height", (d) => (d + "px"))

    // Add your code above this line
  </script>
</body>
```

Solution

```
// solution required
```

10. Learn About SVG in D3

Description

SVG stands for Scalable Vector Graphics . Here "scalable" means that, if you zoom in or out on an object, it would not appear pixelated. It scales with the display system, whether it's on a small mobile screen or a large TV monitor. SVG is used to make common geometric shapes. Since D3 maps data into a visual representation, it uses SVG to create the shapes for the visualization. SVG shapes for a web page must go within an HTML `svg` tag. CSS can be scalable when styles use relative units (such as `vh` , `vw` , or percentages), but using SVG is more flexible to build data visualizations.

Instructions

Add an `svg` node to the `body` using `append()` . Give it a `width` attribute set to the provided `w` constant and a `height` attribute set to the provided `h` constant using the `attr()` or `style()` methods for each. You'll see it in the output because there's a `background-color` of pink applied to it in the `style` tag. **Note**
When using `attr()` width and height attributes do not have units. This is the building block of scaling - the element will always have a 5:1 width to height ratio, no matter what the zoom level is.

Challenge Seed

```

<style>
  svg {
    background-color: pink;
  }
</style>
<body>
  <script>
    const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

    const w = 500;
    const h = 100;

    const svg = d3.select("body")
      // Add your code below this line

      // Add your code above this line
  </script>
</body>

```

Solution

```
// solution required
```

11. Display Shapes with SVG

Description

The last challenge created an `svg` element with a given width and height, which was visible because it had a `background-color` applied to it in the `style` tag. The code made space for the given width and height. The next step is to create a shape to put in the `svg` area. There are a number of supported shapes in SVG, such as rectangles and circles. They are used to display data. For example, a rectangle (`<rect>`) SVG shape could create a bar in a bar chart. When you place a shape into the `svg` area, you can specify where it goes with `x` and `y` coordinates. The origin point of (0, 0) is in the upper-left corner. Positive values for `x` push the shape to the right, and positive values for `y` push

the shape down from the origin point. To place a shape in the middle of the 500 (width) x 100 (height) `svg` from last challenge, the `x` coordinate would be 250 and the `y` coordinate would be 50. An `SVG rect` has four attributes. There are the `x` and `y` coordinates for where it is placed in the `svg` area. It also has a `height` and `width` to specify the size.

Instructions

Add a `rect` shape to the `svg` using `append()`, and give it a `width` attribute of 25 and `height` attribute of 100. Also, give the `rect` `x` and `y` attributes each set to 0.

Challenge Seed

```
<body>
  <script>
    const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

    const w = 500;
    const h = 100;

    const svg = d3.select("body")
      .append("svg")
      .attr("width", w)
      .attr("height", h)
      // Add your code below this line

      // Add your code above this line
  </script>
</body>
```

Solution

```
// solution required
```

12. Create a Bar for Each Data Point in the Set

Description

The last challenge added only one rectangle to the `svg` element to represent a bar. Here, you'll combine what you've learned so far about `data()`, `enter()`, and `SVG shapes` to create and append a rectangle for each data point in `dataset`. A previous challenge showed the format for how to create and append a `div` for each item in `dataset`:

```
d3.select("body").selectAll("div")
  .data(dataset)
  .enter()
  .append("div")
```

There are a few differences working with `rect` elements instead of `divs`. The `rects` must be appended to an `svg` element, not directly to the `body`. Also, you need to tell D3 where to place each `rect` within the `svg` area. The bar placement will be covered in the next challenge.

Instructions

Use the `data()`, `enter()`, and `append()` methods to create and append a `rect` for each item in `dataset`. The bars should display all on top of each other, this will be fixed in the next challenge.

Challenge Seed

```

<body>
  <script>
    const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

    const w = 500;
    const h = 100;

    const svg = d3.select("body")
      .append("svg")
      .attr("width", w)
      .attr("height", h);

    svg.selectAll("rect")
      // Add your code below this line

      // Add your code above this line
      .attr("x", 0)
      .attr("y", 0)
      .attr("width", 25)
      .attr("height", 100);
  </script>
</body>

```

Solution

```
// solution required
```

13. Dynamically Set the Coordinates for Each Bar

Description

The last challenge created and appended a rectangle to the `svg` element for each point in `dataset` to represent a bar. Unfortunately, they were all stacked on top of each other. The placement of a rectangle is handled by the `x` and `y` attributes. They tell D3 where to start drawing the shape in the `svg` area. The last challenge set them each to 0, so every bar was placed in the upper-left corner. For a bar chart, all of the bars should sit on the same vertical level, which means the `y` value stays the same (at 0) for all bars. The `x` value, however, needs to change as you add new bars. Remember that larger `x` values push items farther to the right. As you go through the array elements in `dataset`, the `x` value should increase. The `attr()` method in D3 accepts a callback function to dynamically set that attribute. The callback function takes two arguments, one for the data point itself (usually `d`) and one for the index of the data point in the array. The second argument for the index is optional. Here's the format:

```

selection.attr("property", (d, i) => {
  /*
  * d is the data point value
  * i is the index of the data point in the array
  */
})

```

It's important to note that you do NOT need to write a `for` loop or use `forEach()` to iterate over the items in the data set. Recall that the `data()` method parses the data set, and any method that's chained after `data()` is run once for each item in the data set.

Instructions

Change the `x` attribute callback function so it returns the index times 30. **Note**

Each bar has a width of 25, so increasing each `x` value by 30 adds some space between the bars. Any value greater than 25 would work in this example.

Challenge Seed

```

<body>
  <script>
    const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

    const w = 500;
    const h = 100;

    const svg = d3.select("body")
      .append("svg")
      .attr("width", w)
      .attr("height", h);

    svg.selectAll("rect")
      .data(dataset)
      .enter()
      .append("rect")
      .attr("x", (d, i) => {
        // Add your code below this line

        // Add your code above this line
      })
      .attr("y", 0)
      .attr("width", 25)
      .attr("height", 100);
  </script>
</body>

```

Solution

```
// solution required
```

14. Dynamically Change the Height of Each Bar

Description

The height of each bar can be set to the value of the data point in the array, similar to how the `x` value was set dynamically.

```

selection.attr("property", (d, i) => {
  /*
  * d is the data point value
  * i is the index of the data point in the array
  */
})
```

Instructions

Change the callback function for the `height` attribute to return the data value times 3. **Note**

Remember that multiplying all data points by the same constant scales the data (like zooming in). It helps to see the differences between bar values in this example.

Challenge Seed

```

<body>
  <script>
    const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

    const w = 500;
    const h = 100;

    const svg = d3.select("body")
```

```

.append("svg")
.attr("width", w)
.attr("height", h);

svg.selectAll("rect")
.data(dataset)
.enter()
.append("rect")
.attr("x", (d, i) => i * 30)
.attr("y", 0)
.attr("width", 25)
.attr("height", (d, i) => {
  // Add your code below this line

  // Add your code above this line
});
</script>
</body>

```

Solution

```
// solution required
```

15. Invert SVG Elements

Description

You may have noticed the bar chart looked like it's upside-down, or inverted. This is because of how SVG uses (x, y) coordinates. In SVG, the origin point for the coordinates is in the upper-left corner. An `x` coordinate of 0 places a shape on the left edge of the SVG area. A `y` coordinate of 0 places a shape on the top edge of the SVG area. Higher `x` values push the rectangle to the right. Higher `y` values push the rectangle down. To make the bars right-side-up, you need to change the way the `y` coordinate is calculated. It needs to account for both the height of the bar and the total height of the SVG area. The height of the SVG area is 100. If you have a data point of 0 in the set, you would want the bar to start at the bottom of the SVG area (not the top). To do this, the `y` coordinate needs a value of 100. If the data point value were 1, you would start with a `y` coordinate of 100 to set the bar at the bottom. Then you need to account for the height of the bar of 1, so the final `y` coordinate would be 99. The `y` coordinate that is `y = heightOfSVG - heightOfBar` would place the bars right-side-up.

Instructions

Change the callback function for the `y` attribute to set the bars right-side-up. Remember that the `height` of the bar is 3 times the data value `d`. **Note**

In general, the relationship is `y = h - m * d`, where `m` is the constant that scales the data points.

Challenge Seed

```

<body>
<script>
const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

const w = 500;
const h = 100;

const svg = d3.select("body")
.append("svg")
.attr("width", w)
.attr("height", h);

svg.selectAll("rect")
.data(dataset)
.enter()

```

```

.append("rect")
.attr("x", (d, i) => i * 30)
.attr("y", (d, i) => {
  // Add your code below this line

  // Add your code above this line
})
.attr("width", 25)
.attr("height", (d, i) => 3 * d);
</script>
</body>

```

Solution

```
// solution required
```

16. Change the Color of an SVG Element

Description

The bars are in the right position, but they are all the same black color. SVG has a way to change the color of the bars. In SVG, a `rect` shape is colored with the `fill` attribute. It supports hex codes, color names, and `rgb` values, as well as more complex options like gradients and transparency.

Instructions

Add an `attr()` method to set the "fill" of all the bars to the color "navy".

Challenge Seed

```

<body>
<script>
const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

const w = 500;
const h = 100;

const svg = d3.select("body")
  .append("svg")
  .attr("width", w)
  .attr("height", h);

svg.selectAll("rect")
  .data(dataset)
  .enter()
  .append("rect")
  .attr("x", (d, i) => i * 30)
  .attr("y", (d, i) => h - 3 * d)
  .attr("width", 25)
  .attr("height", (d, i) => 3 * d)
  // Add your code below this line

  // Add your code above this line
</script>
</body>

```

Solution

```
// solution required
```

17. Add Labels to D3 Elements

Description

D3 lets you label a graph element, such as a bar, using the SVG `text` element. Like the `rect` element, a `text` element needs to have `x` and `y` attributes, to place it on the SVG canvas. It also needs to access the data to display those values. D3 gives you a high level of control over how you label your bars.

Instructions

The code in the editor already binds the data to each new `text` element. First, append `text` nodes to the `svg`. Next, add attributes for the `x` and `y` coordinates. They should be calculated the same way as the `rect` ones, except the `y` value for the `text` should make the label sit 3 units higher than the bar. Finally, use the D3 `text()` method to set the label equal to the data point value. **Note**

For the label to sit higher than the bar, decide if the `y` value for the `text` should be 3 greater or 3 less than the `y` value for the bar.

Challenge Seed

```
<body>
<script>
const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

const w = 500;
const h = 100;

const svg = d3.select("body")
    .append("svg")
    .attr("width", w)
    .attr("height", h);

svg.selectAll("rect")
    .data(dataset)
    .enter()
    .append("rect")
    .attr("x", (d, i) => i * 30)
    .attr("y", (d, i) => h - 3 * d)
    .attr("width", 25)
    .attr("height", (d, i) => 3 * d)
    .attr("fill", "navy");

svg.selectAll("text")
    .data(dataset)
    .enter()
    // Add your code below this line

        // Add your code above this line
</script>
<body>
```

Solution

```
// solution required
```

18. Style D3 Labels

Description

D3 methods can add styles to the bar labels. The `fill` attribute sets the color of the text for a `text` node. The `style()` method sets CSS rules for other styles, such as "font-family" or "font-size".

Instructions

Set the `font-size` of the `text` elements to 25px, and the color of the text to red.

Challenge Seed

```
<body>
<script>
const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

const w = 500;
const h = 100;

const svg = d3.select("body")
    .append("svg")
    .attr("width", w)
    .attr("height", h);

svg.selectAll("rect")
    .data(dataset)
    .enter()
    .append("rect")
    .attr("x", (d, i) => i * 30)
    .attr("y", (d, i) => h - 3 * d)
    .attr("width", 25)
    .attr("height", (d, i) => d * 3)
    .attr("fill", "navy");

svg.selectAll("text")
    .data(dataset)
    .enter()
    .append("text")
    .text((d) => d)
    .attr("x", (d, i) => i * 30)
    .attr("y", (d, i) => h - (3 * d) - 3)
    // Add your code below this line

    // Add your code above this line
</script>
</body>
```

Solution

```
// solution required
```

19. Add a Hover Effect to a D3 Element

Description

It's possible to add effects that highlight a bar when the user hovers over it with the mouse. So far, the styling for the rectangles is applied with the built-in D3 and SVG methods, but you can use CSS as well. You set the CSS class on the

SVG elements with the `attr()` method. Then the `:hover` pseudo-class for your new class holds the style rules for any hover effects.

Instructions

Use the `attr()` method to add a class of `bar` to all the `rect` elements. This changes the `fill` color of the bar to brown when you mouse over it.

Challenge Seed

```
<style>
  .bar:hover {
    fill: brown;
  }
</style>
<body>
  <script>
    const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

    const w = 500;
    const h = 100;

    const svg = d3.select("body")
      .append("svg")
      .attr("width", w)
      .attr("height", h);

    svg.selectAll("rect")
      .data(dataset)
      .enter()
      .append("rect")
      .attr("x", (d, i) => i * 30)
      .attr("y", (d, i) => h - 3 * d)
      .attr("width", 25)
      .attr("height", (d, i) => 3 * d)
      .attr("fill", "navy")
    // Add your code below this line

    // Add your code above this line

    svg.selectAll("text")
      .data(dataset)
      .enter()
      .append("text")
      .text((d) => d)
      .attr("x", (d, i) => i * 30)
      .attr("y", (d, i) => h - (3 * d) - 3);

  </script>
</body>
```

Solution

```
// solution required
```

20. Add a Tooltip to a D3 Element

Description

A tooltip shows more information about an item on a page when the user hovers over that item. There are several ways to add a tooltip to a visualization, this challenge uses the SVG `title` element. `title` pairs with the `text()` method to dynamically add data to the bars.

Instructions

Append a `title` element under each `rect` node. Then call the `text()` method with a callback function so the text displays the data value.

Challenge Seed

```

<style>
  .bar:hover {
    fill: brown;
  }
</style>
<body>
  <script>
    const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];

    const w = 500;
    const h = 100;

    const svg = d3.select("body")
      .append("svg")
      .attr("width", w)
      .attr("height", h);

    svg.selectAll("rect")
      .data(dataset)
      .enter()
      .append("rect")
      .attr("x", (d, i) => i * 30)
      .attr("y", (d, i) => h - 3 * d)
      .attr("width", 25)
      .attr("height", (d, i) => d * 3)
      .attr("fill", "navy")
      .attr("class", "bar")
    // Add your code below this line

    // Add your code above this line

    svg.selectAll("text")
      .data(dataset)
      .enter()
      .append("text")
      .text((d) => d)
      .attr("x", (d, i) => i * 30)
      .attr("y", (d, i) => h - (d * 3 + 3))

  </script>
</body>

```

Solution

```
// solution required
```

21. Create a Scatterplot with SVG Circles

Description

A scatter plot is another type of visualization. It usually uses circles to map data points, which have two values each. These values tie to the `x` and `y` axes, and are used to position the circle in the visualization. SVG has a `circle` tag to create the circle shape. It works a lot like the `rect` elements you used for the bar chart.

Instructions

Use the `data()`, `enter()`, and `append()` methods to bind dataset to new `circle` elements that are appended to the SVG canvas. **Note**

The circles won't be visible because we haven't set their attributes yet. We'll do that in the next challenge.

Challenge Seed

```
<body>
<script>
  const dataset = [
    [ 34,    78 ],
    [ 109,   280 ],
    [ 310,   120 ],
    [ 79,    411 ],
    [ 420,   220 ],
    [ 233,   145 ],
    [ 333,   96 ],
    [ 222,   333 ],
    [ 78,    320 ],
    [ 21,    123 ]
  ];

  const w = 500;
  const h = 500;

  const svg = d3.select("body")
    .append("svg")
    .attr("width", w)
    .attr("height", h);

  svg.selectAll("circle")
    // Add your code below this line

    // Add your code above this line

  </script>
</body>
```

Solution

```
// solution required
```

22. Add Attributes to the Circle Elements

Description

The last challenge created the `circle` elements for each point in the `dataset`, and appended them to the SVG canvas. But D3 needs more information about the position and size of each `circle` to display them correctly. A `circle` in SVG has three main attributes. The `cx` and `cy` attributes are the coordinates. They tell D3 where to position the *center* of the shape on the SVG canvas. The `radius` (`r` attribute) gives the size of the `circle`. Just like the `rect` `y` coordinate, the `cy` attribute for a `circle` is measured from the top of the SVG canvas, not from the bottom. All three attributes can use a callback function to set their values dynamically. Remember that all methods chained after `data(dataset)` run once per item in `dataset`. The `d` parameter in the callback function refers to the current item in `dataset`, which is an array for each point. You use bracket notation, like `d[0]`, to access the values in that array.

Instructions

Add `cx`, `cy`, and `r` attributes to the `circle` elements. The `cx` value should be the first number in the array for each item in `dataset`. The `cy` value should be based off the second number in the array, but make sure to show the

chart right-side-up and not inverted. The `r` value should be 5 for all circles.

Challenge Seed

```
<body>
<script>
  const dataset = [
    [ 34,    78 ],
    [ 109,   280 ],
    [ 310,   120 ],
    [ 79,    411 ],
    [ 420,   220 ],
    [ 233,   145 ],
    [ 333,   96 ],
    [ 222,   333 ],
    [ 78,    320 ],
    [ 21,    123 ]
  ];

  const w = 500;
  const h = 500;

  const svg = d3.select("body")
    .append("svg")
    .attr("width", w)
    .attr("height", h);

  svg.selectAll("circle")
    .data(dataset)
    .enter()
    .append("circle")
    // Add your code below this line

    // Add your code above this line

  </script>
</body>
```

Solution

```
// solution required
```

23. Add Labels to Scatter Plot Circles

Description

You can add text to create labels for the points in a scatter plot. The goal is to display the comma-separated values for the first (`x`) and second (`y`) fields of each item in `dataset`. The `text` nodes need `x` and `y` attributes to position it on the SVG canvas. In this challenge, the `y` value (which determines height) can use the same value that the `circle` uses for its `cy` attribute. The `x` value can be slightly larger than the `cx` value of the `circle`, so the label is visible. This will push the label to the right of the plotted point.

Instructions

Label each point on the scatter plot using the `text` elements. The text of the label should be the two values separated by a comma and a space. For example, the label for the first point is "34, 78". Set the `x` attribute so it's 5 units more than the value you used for the `cx` attribute on the `circle`. Set the `y` attribute the same way that's used for the `cy` value on the `circle`.

Challenge Seed

```

<body>
  <script>
    const dataset = [
      [ 34,    78 ],
      [ 109,   280 ],
      [ 310,   120 ],
      [ 79,    411 ],
      [ 420,   220 ],
      [ 233,   145 ],
      [ 333,   96 ],
      [ 222,   333 ],
      [ 78,    320 ],
      [ 21,    123 ]
    ];

    const w = 500;
    const h = 500;

    const svg = d3.select("body")
      .append("svg")
      .attr("width", w)
      .attr("height", h);

    svg.selectAll("circle")
      .data(dataset)
      .enter()
      .append("circle")
      .attr("cx", (d, i) => d[0])
      .attr("cy", (d, i) => h - d[1])
      .attr("r", 5);

    svg.selectAll("text")
      .data(dataset)
      .enter()
      .append("text")
    // Add your code below this line

    // Add your code above this line
  </script>
</body>

```

Solution

```
// solution required
```

24. Create a Linear Scale with D3

Description

The bar and scatter plot charts both plotted data directly onto the SVG canvas. However, if the height of a bar or one of the data points were larger than the SVG height or width values, it would go outside the SVG area. In D3, there are scales to help plot data. Scales are functions that tell the program how to map a set of raw data points onto the pixels of the SVG canvas. For example, say you have a 100x500-sized SVG canvas and you want to plot Gross Domestic Product (GDP) for a number of countries. The set of numbers would be in the billion or trillion-dollar range. You provide D3 a type of scale to tell it how to place the large GDP values into that 100x500-sized area. It's unlikely you would plot raw data as-is. Before plotting it, you set the scale for your entire data set, so that the `x` and `y` values fit your canvas width and height. D3 has several scale types. For a linear scale (usually used with quantitative data), there is the D3 method `scaleLinear()`: `const scale = d3.scaleLinear()` By default, a scale uses the identity relationship. The value of the input is the same as the value of the output. A separate challenge covers how to change this.

Instructions

Change the `scale` variable to create a linear scale. Then set the `output` variable to the scale called with an input argument of 50.

Challenge Seed

```
<body>
<script>
// Add your code below this line

const scale = undefined; // Create the scale here
const output = scale(); // Call the scale with an argument here

// Add your code above this line

d3.select("body")
.append("h2")
.text(output);

</script>
</body>
```

Solution

```
// solution required
```

25. Set a Domain and a Range on a Scale

Description

By default, scales use the identity relationship - the input value maps to the output value. But scales can be much more flexible and interesting. Say a data set has values ranging from 50 to 480. This is the input information for a scale, and is also known as the domain. You want to map those points along the `x` axis on the SVG canvas, between 10 units and 500 units. This is the output information, which is also known as the range. The `domain()` and `range()` methods set these values for the scale. Both methods take an array of at least two elements as an argument. Here's an example:

```
// Set a domain
// The domain covers the set of input values
scale.domain([50, 480]);
// Set a range
// The range covers the set of output values
scale.range([10, 500]);
scale(50) // Returns 10
scale(480) // Returns 500
scale(325) // Returns 323.37
scale(750) // Returns 807.67
d3.scaleLinear()
```

Notice that the scale uses the linear relationship between the domain and range values to figure out what the output should be for a given number. The minimum value in the domain (50) maps to the minimum value (10) in the range.

Instructions

Create a scale and set its domain to [250, 500] and range to [10, 150]. **Note**
You can chain the `domain()` and `range()` methods onto the `scale` variable.

Challenge Seed

```

<body>
  <script>
    // Add your code below this line
    const scale = d3.scaleLinear();

    // Add your code above this line
    const output = scale(50);
    d3.select("body")
      .append("h2")
      .text(output);
  </script>
</body>

```

Solution

```
// solution required
```

26. Use the d3.max and d3.min Functions to Find Minimum and Maximum Values in a Dataset

Description

The D3 methods `domain()` and `range()` set that information for your scale based on the data. There are a couple methods to make that easier. Often when you set the domain, you'll want to use the minimum and maximum values within the data set. Trying to find these values manually, especially in a large data set, may cause errors. D3 has two methods - `min()` and `max()` to return this information. Here's an example:

```

const exampleData = [34, 234, 73, 90, 6, 52];
d3.min(exampleData) // Returns 6
d3.max(exampleData) // Returns 234

```

A dataset may have nested arrays, like the `[x, y]` coordinate pairs that were in the scatter plot example. In that case, you need to tell D3 how to calculate the maximum and minimum. Fortunately, both the `min()` and `max()` methods take a callback function. In this example, the callback function's argument `d` is for the current inner array. The callback needs to return the element from the inner array (the `x` or `y` value) over which you want to compute the maximum or minimum. Here's an example for how to find the min and max values with an array of arrays:

```

const locationData = [[1, 7],[6, 3],[8, 3]];
// Returns the smallest number out of the first elements
const minX = d3.min(locationData, (d) => d[0]);
// minX compared 1, 6, and 8 and is set to 1

```

Instructions

The `positionData` variable holds a 3-dimensional (3D) array. Use a D3 method to find the maximum value of the `z` coordinate (the third value) from the arrays and save it in the `output` variable. **Note**

Fun fact - D3 can plot 3D arrays.

Challenge Seed

```

<body>
  <script>
    const positionData = [[1, 7, -4],[6, 3, 8],[2, 9, 3]]
    // Add your code below this line

    const output = undefined; // Change this line

    // Add your code above this line
  </script>
</body>

```

```
d3.select("body")
  .append("h2")
  .text(output)
</script>
</body>
```

Solution

```
// solution required
```

27. Use Dynamic Scales

Description

The D3 `min()` and `max()` methods are useful to help set the scale. Given a complex data set, one priority is to set the scale so the visualization fits the SVG container's width and height. You want all the data plotted inside the SVG canvas so it's visible on the web page. The example below sets the x-axis scale for scatter plot data. The `domain()` method passes information to the scale about the raw data values for the plot. The `range()` method gives it information about the actual space on the web page for the visualization. In the example, the domain goes from 0 to the maximum in the set. It uses the `max()` method with a callback function based on the x values in the arrays. The range uses the SVG canvas' width (`w`), but it includes some padding, too. This puts space between the scatter plot dots and the edge of the SVG canvas.

```
const dataset = [
  [ 34, 78 ],
  [ 109, 280 ],
  [ 310, 120 ],
  [ 79, 411 ],
  [ 420, 220 ],
  [ 233, 145 ],
  [ 333, 96 ],
  [ 222, 333 ],
  [ 78, 320 ],
  [ 21, 123 ]
];
const w = 500;
const h = 500;

// Padding between the SVG canvas boundary and the plot
const padding = 30;
const xScale = d3.scaleLinear()
  .domain([0, d3.max(dataset, (d) => d[0])])
  .range([padding, w - padding]);
```

The padding may be confusing at first. Picture the x-axis as a horizontal line from 0 to 500 (the width value for the SVG canvas). Including the padding in the `range()` method forces the plot to start at 30 along that line (instead of 0), and end at 470 (instead of 500).

Instructions

Use the `yScale` variable to create a linear y-axis scale. The domain should start at zero and go to the maximum y value in the set. The range should use the SVG height (`h`) and include padding. **Note**
Remember to keep the plot right-side-up. When you set the range for the y coordinates, the higher value (height minus padding) is the first argument, and the lower value is the second argument.

Challenge Seed

```

<body>
<script>
  const dataset = [
    [ 34,    78 ],
    [ 109,   280 ],
    [ 310,   120 ],
    [ 79,    411 ],
    [ 420,   220 ],
    [ 233,   145 ],
    [ 333,   96 ],
    [ 222,   333 ],
    [ 78,    320 ],
    [ 21,    123 ]
  ];

  const w = 500;
  const h = 500;

  // Padding between the SVG canvas boundary and the plot
  const padding = 30;

  // Create an x and y scale

  const xScale = d3.scaleLinear()
    .domain([0, d3.max(dataset, (d) => d[0])])
    .range([padding, w - padding]);

  // Add your code below this line

  const yScale = undefined;

  // Add your code above this line

  const output = yScale(411); // Returns 30
  d3.select("body")
    .append("h2")
    .text(output)
</script>
</body>

```

Solution

```
// solution required
```

28. Use a Pre-Defined Scale to Place Elements

Description

With the scales set up, it's time to map the scatter plot again. The scales are like processing functions that turn the x and y raw data into values that fit and render correctly on the SVG canvas. They keep the data within the screen's plotting area. You set the coordinate attribute values for an SVG shape with the scaling function. This includes x and y attributes for rect or text elements, or cx and cy for circles. Here's an example:

```
shape
  .attr("x", (d) => xScale(d[0]))
```

Scales set shape coordinate attributes to place the data points onto the SVG canvas. You don't need to apply scales when you display the actual data value, for example, in the `text()` method for a tooltip or label.

Instructions

Use `xScale` and `yScale` to position both the `circle` and `text` shapes onto the SVG canvas. For the `circles`, apply the scales to set the `cx` and `cy` attributes. Give them a radius of 5 units, too. For the `text` elements, apply the

scales to set the `x` and `y` attributes. The labels should be offset to the right of the dots. To do this, add 10 units to the `x` data value before passing it to the `xScale`.

Challenge Seed

```

<body>
  <script>
    const dataset = [
      [ 34,      78 ],
      [ 109,     280 ],
      [ 310,     120 ],
      [ 79,      411 ],
      [ 420,     220 ],
      [ 233,     145 ],
      [ 333,     96 ],
      [ 222,     333 ],
      [ 78,      320 ],
      [ 21,      123 ]
    ];

    const w = 500;
    const h = 500;
    const padding = 60;

    const xScale = d3.scaleLinear()
      .domain([0, d3.max(dataset, (d) => d[0])])
      .range([padding, w - padding]);

    const yScale = d3.scaleLinear()
      .domain([0, d3.max(dataset, (d) => d[1])])
      .range([h - padding, padding]);

    const svg = d3.select("body")
      .append("svg")
      .attr("width", w)
      .attr("height", h);

    svg.selectAll("circle")
      .data(dataset)
      .enter()
      .append("circle")
    // Add your code below this line

    // Add your code above this line

    svg.selectAll("text")
      .data(dataset)
      .enter()
      .append("text")
      .text((d) => (d[0] + ", "
      + d[1]))
    // Add your code below this line

    // Add your code above this line
  </script>
</body>

```

Solution

```
// solution required
```

29. Add Axes to a Visualization

Description

Another way to improve the scatter plot is to add an x-axis and a y-axis. D3 has two methods `axisLeft()` and `axisBottom()` to render the y and x axes, respectively. (Axes is the plural form of axis). Here's an example to create the x-axis based on the `xScale` in the previous challenges: `const xAxis = d3.axisBottom(xScale);` The next step is to render the axis on the SVG canvas. To do so, you can use a general SVG component, the `g` element. The `g` stands for group. Unlike `rect`, `circle`, and `text`, an axis is just a straight line when it's rendered. Because it is a simple shape, using `g` works. The last step is to apply a `transform` attribute to position the axis on the SVG canvas in the right place. Otherwise, the line would render along the border of SVG canvas and wouldn't be visible. SVG supports different types of `transforms`, but positioning an axis needs `translate`. When it's applied to the `g` element, it moves the whole group over and down by the given amounts. Here's an example:

```
const xAxis = d3.axisBottom(xScale);

svg.append("g")
  .attr("transform", "translate(0, " + (h - padding) + ")")
  .call(xAxis);
```

The above code places the x-axis at the bottom of the SVG canvas. Then it's passed as an argument to the `call()` method. The y-axis works the same way, except the `translate` argument is in the form `(x, 0)`. Because `translate` is a string in the `attr()` method above, you can use concatenation to include variable values for its arguments.

Instructions

The scatter plot now has an x-axis. Create a y-axis in a variable named `yAxis` using the `axisLeft()` method. Then render the axis using a `g` element. Make sure to use a `transform` attribute to translate the axis by the amount of padding units right, and 0 units down. Remember to `call()` the axis.

Challenge Seed

```
<body>
<script>
  const dataset = [
    [ 34,      78 ],
    [ 109,     280 ],
    [ 310,     120 ],
    [ 79,      411 ],
    [ 420,     220 ],
    [ 233,     145 ],
    [ 333,     96 ],
    [ 222,     333 ],
    [ 78,      320 ],
    [ 21,      123 ]
  ];

  const w = 500;
  const h = 500;
  const padding = 60;

  const xScale = d3.scaleLinear()
    .domain([0, d3.max(dataset, (d) => d[0])])
    .range([padding, w - padding]);

  const yScale = d3.scaleLinear()
    .domain([0, d3.max(dataset, (d) => d[1])])
    .range([h - padding, padding]);

  const svg = d3.select("body")
    .append("svg")
    .attr("width", w)
    .attr("height", h);

  svg.selectAll("circle")
    .data(dataset)
    .enter()
    .append("circle")
    .attr("cx", (d) => xScale(d[0]))
    .attr("cy", (d) => yScale(d[1]))
    .attr("r", (d) => 5);
```

```

svg.selectAll("text")
  .data(dataset)
  .enter()
  .append("text")
  .text((d) => (d[0] + "," + d[1]))
  .attr("x", (d) => xScale(d[0] + 10))
  .attr("y", (d) => yScale(d[1]))

const xAxis = d3.axisBottom(xScale);
// Add your code below this line
const yAxis = undefined;
// Add your code above this line

svg.append("g")
  .attr("transform", "translate(0," + (h - padding) + ")")
  .call(xAxis);

// Add your code below this line

// Add your code above this line

</script>
</body>

```

Solution

```
// solution required
```

JSON APIs and Ajax

1. Handle Click Events with JavaScript using the onclick property

Description

You want your code to execute only once your page has finished loading. For that purpose, you can attach a JavaScript event to the document called `DOMContentLoaded`. Here's the code that does this:

```

document.addEventListener('DOMContentLoaded',function() {
  });

```

You can implement event handlers that go inside of the `DOMContentLoaded` function. You can implement an `onclick` event handler which triggers when the user clicks on the element with id `getMessage`, by adding the following code:

```

document.getElementById('getMessage').onclick=function(){}

```

Instructions

Add a click event handler inside of the `DOMContentLoaded` function for the element with id of `getMessage`.

Challenge Seed

```

<script>
  document.addEventListener('DOMContentLoaded',function(){
    // Add your code below this line

    // Add your code above this line
  });

```

```

</script>
<style>
  body {
    text-align: center;
    font-family: "Helvetica", sans-serif;
  }
  h1 {
    font-size: 2em;
    font-weight: bold;
  }
  .box {
    border-radius: 5px;
    background-color: #eee;
    padding: 20px 5px;
  }
  button {
    color: white;
    background-color: #4791d0;
    border-radius: 5px;
    border: 1px solid #4791d0;
    padding: 5px 10px 8px 10px;
  }
  button:hover {
    background-color: #0f5897;
    border: 1px solid #0f5897;
  }
</style>
<h1>Cat Photo Finder</h1>
<p class="message">
  The message will go here
</p>
<p>
  <button id="getMessage">
    Get Message
  </button>
</p>

```

Solution

```
// solution required
```

2. Change Text with click Events

Description

When the click event happens, you can use JavaScript to update an HTML element. For example, when a user clicks the "Get Message" button, it changes the text of the element with the class `message` to say "Here is the message". This works by adding the following code within the click event: `document.getElementsByClassName('message')[0].textContent="Here is the message";`

Instructions

Add code inside the `onclick` event handler to change the text inside the `message` element to say "Here is the message".

Challenge Seed

```

<script>
  document.addEventListener('DOMContentLoaded',function(){
    document.getElementById('getMessage').onclick=function(){
      // Add your code below this line

      // Add your code above this line
    }
  })

```

```

        }
    });
</script>
<style>
  body {
    text-align: center;
    font-family: "Helvetica", sans-serif;
  }
  h1 {
    font-size: 2em;
    font-weight: bold;
  }
  .box {
    border-radius: 5px;
    background-color: #eee;
    padding: 20px 5px;
  }
  button {
    color: white;
    background-color: #4791d0;
    border-radius: 5px;
    border: 1px solid #4791d0;
    padding: 5px 10px 8px 10px;
  }
  button:hover {
    background-color: #0f5897;
    border: 1px solid #0f5897;
  }
</style>
<h1>Cat Photo Finder</h1>
<p class="message">
  The message will go here
</p>
<p>
  <button id="getMessage">
    Get Message
  </button>
</p>

```

Solution

```
// solution required
```

3. Get JSON with the JavaScript XMLHttpRequest Method

Description

You can also request data from an external source. This is where APIs come into play. Remember that APIs - or Application Programming Interfaces - are tools that computers use to communicate with one another. You'll learn how to update HTML with the data we get from APIs using a technology called AJAX. Most web APIs transfer data in a format called JSON. JSON stands for JavaScript Object Notation. JSON syntax looks very similar to JavaScript object literal notation. JSON has object properties and their current values, sandwiched between a `{` and a `}`. These properties and their values are often referred to as "key-value pairs". However, JSON transmitted by APIs are sent as bytes, and your application receives it as a `string`. These can be converted into JavaScript objects, but they are not JavaScript objects by default. The `JSON.parse` method parses the string and constructs the JavaScript object described by it. You can request the JSON from freeCodeCamp's Cat Photo API. Here's the code you can put in your click event to do this:

```

req=new XMLHttpRequest();
req.open("GET",'./json/cats.json',true);
req.send();
req.onload=function(){
  json=JSON.parse(req.responseText);
  document.getElementsByClassName('message')[0].innerHTML=JSON.stringify(json);
};

```

Here's a review of what each piece is doing. The JavaScript `XMLHttpRequest` object has a number of properties and methods that are used to transfer data. First, an instance of the `XMLHttpRequest` object is created and saved in the `req` variable. Next, the `open` method initializes a request - this example is requesting data from an API, therefore is a "GET" request. The second argument for `open` is the URL of the API you are requesting data from. The third argument is a Boolean value where `true` makes it an asynchronous request. The `send` method sends the request. Finally, the `onload` event handler parses the returned data and applies the `JSON.stringify` method to convert the JavaScript object into a string. This string is then inserted as the message text.

Instructions

Update the code to create and send a "GET" request to the freeCodeCamp Cat Photo API. Then click the "Get Message" button. Your AJAX function will replace the "The message will go here" text with the raw JSON output from the API.

Challenge Seed

```
<script>
  document.addEventListener('DOMContentLoaded',function(){
    document.getElementById('getMessage').onclick=function(){
      // Add your code below this line

      // Add your code above this line
    };
  });
</script>
<style>
  body {
    text-align: center;
    font-family: "Helvetica", sans-serif;
  }
  h1 {
    font-size: 2em;
    font-weight: bold;
  }
  .box {
    border-radius: 5px;
    background-color: #eee;
    padding: 20px 5px;
  }
  button {
    color: white;
    background-color: #4791d0;
    border-radius: 5px;
    border: 1px solid #4791d0;
    padding: 5px 10px 8px 10px;
  }
  button:hover {
    background-color: #0F5897;
    border: 1px solid #0F5897;
  }
</style>
<h1>Cat Photo Finder</h1>
<p class="message">
  The message will go here
</p>
<p>
  <button id="getMessage">
    Get Message
  </button>
</p>
```

Solution

```
// solution required
<script>
  document.addEventListener('DOMContentLoaded',function(){
    document.getElementById('getMessage').onclick=function(){
      const req = new XMLHttpRequest();
      req.open('GET', '/json/cats.json', true);
      req.onload = function() {
        const data = JSON.parse(this.responseText);
        document.querySelector('.message').innerHTML = data.message;
      }
    }
  });
</script>
```

```

req.send();
req.onload = () => {
  const json = JSON.parse(req.responseText);
  document.getElementsByClassName('message')[0].innerHTML = JSON.stringify(json);
};

});
</script>
<style>
body {
  text-align: center;
  font-family: "Helvetica", sans-serif;
}
h1 {
  font-size: 2em;
  font-weight: bold;
}
.box {
  border-radius: 5px;
  background-color: #eee;
  padding: 20px 5px;
}
button {
  color: white;
  background-color: #4791d0;
  border-radius: 5px;
  border: 1px solid #4791d0;
  padding: 5px 10px 8px 10px;
}
button:hover {
  background-color: #0f5897;
  border: 1px solid #0f5897;
}
</style>
<h1>Cat Photo Finder</h1>
<p class="message">
  The message will go here
</p>
<p>
  <button id="getMessage">
    Get Message
  </button>
</p>
</p>

```

4. Access the JSON Data from an API

Description

In the previous challenge, you saw how to get JSON data from the freeCodeCamp Cat Photo API. Now you'll take a closer look at the returned data to better understand the JSON format. Recall some notation in JavaScript:

- [] -> Square brackets represent an array
- { } -> Curly brackets represent an object
- " " -> Double quotes represent a string. They are also used for key names in JSON

Understanding the structure of the data that an API returns is important because it influences how you retrieve the values you need. On the right, click the "Get Message" button to load the freeCodeCamp Cat Photo API JSON into the HTML. The first and last character you see in the JSON data are square brackets []. This means that the returned data is an array. The second character in the JSON data is a curly { bracket, which starts an object. Looking closely, you can see that there are three separate objects. The JSON data is an array of three objects, where each object contains information about a cat photo. You learned earlier that objects contain "key-value pairs" that are separated by commas. In the Cat Photo example, the first object has "id":0 where "id" is a key and 0 is its corresponding value. Similarly, there are keys for "imageLink", "altText", and "codeNames". Each cat photo object has these same keys, but with different values. Another interesting "key-value pair" in the first object is "codeNames": ["Juggernaut", "Mrs. Wallace", "ButterCup"] . Here "codeNames" is the key and its value is an array of three strings. It's possible to have arrays of objects as well as a key with an array as a value. Remember how to access data in arrays and objects. Arrays use bracket notation to access a specific index of an item. Objects use either bracket or dot notation to access the

value of a given property. Here's an example that prints the "altText" of the first cat photo - note that the parsed JSON data in the editor is saved in a variable called `json` :

```
console.log(json[0].altText);
// Prints "A white cat wearing a green helmet shaped melon on its head."
```

Instructions

For the cat with the "id" of 2, print to the console the second value in the `codeNames` array. You should use bracket and dot notation on the object (which is saved in the variable `json`) to access the value.

Challenge Seed

```
<script>
  document.addEventListener('DOMContentLoaded', function(){
    document.getElementById('getMessage').onclick=function(){
      req=new XMLHttpRequest();
      req.open("GET",'./json/cats.json',true);
      req.send();
      req.onload=function(){
        json=JSON.parse(req.responseText);
        document.getElementsByClassName('message')[0].innerHTML=JSON.stringify(json);
        // Add your code below this line

        // Add your code above this line
      };
    });
  });
</script>
<style>
  body {
    text-align: center;
    font-family: "Helvetica", sans-serif;
  }
  h1 {
    font-size: 2em;
    font-weight: bold;
  }
  .box {
    border-radius: 5px;
    background-color: #eee;
    padding: 20px 5px;
  }
  button {
    color: white;
    background-color: #4791d0;
    border-radius: 5px;
    border: 1px solid #4791d0;
    padding: 5px 10px 8px 10px;
  }
  button:hover {
    background-color: #0F5897;
    border: 1px solid #0F5897;
  }
</style>
<h1>Cat Photo Finder</h1>
<p class="message">
  The message will go here
</p>
<p>
  <button id="getMessage">
    Get Message
  </button>
</p>
```

Solution

```
// solution required
```

5. Convert JSON Data to HTML

Description

Now that you're getting data from a JSON API, you can display it in the HTML. You can use a `forEach` method to loop through the data since the cat photo objects are held in an array. As you get to each item, you can modify the HTML elements. First, declare an `html` variable with `var html = "";`. Then, loop through the JSON, adding HTML to the variable that wraps the key names in `strong` tags, followed by the value. When the loop is finished, you render it. Here's the code that does this:

```
json.forEach(function(val) {
  var keys = Object.keys(val);
  html += "<div class = 'cat'>";
  keys.forEach(function(key) {
    html += "<strong>" + key + "</strong>: " + val[key] + "<br>";
  });
  html += "</div><br>";
});
```

Instructions

Add a `forEach` method to loop over the JSON data and create the HTML elements to display it. Here is some example JSON

```
[
  {
    "id":0,
    "imageLink":"https://s3.amazonaws.com/freecodecamp/funny-cat.jpg",
    "altText":"A white cat wearing a green helmet shaped melon on its head. ",
    "codeNames":["Juggernaut", "Mrs. Wallace", "Buttercup"]
  }
]
```

Challenge Seed

```
<script>
  document.addEventListener('DOMContentLoaded',function(){
    document.getElementById('getMessage').onclick=function(){
      req=new XMLHttpRequest();
      req.open("GET",'./json/cats.json',true);
      req.send();
      req.onload=function(){
        json=JSON.parse(req.responseText);
        var html = "";
        // Add your code below this line

        // Add your code above this line
        document.getElementsByClassName('message')[0].innerHTML=html;
      };
    };
  });
</script>
<style>
  body {
    text-align: center;
    font-family: "Helvetica", sans-serif;
  }
  h1 {
    font-size: 2em;
    font-weight: bold;
  }
  .box {
    border-radius: 5px;
  }
</style>
```

```

background-color: #eee;
padding: 20px 5px;
}
button {
  color: white;
  background-color: #4791d0;
  border-radius: 5px;
  border: 1px solid #4791d0;
  padding: 5px 10px 8px 10px;
}
button:hover {
  background-color: #0f5897;
  border: 1px solid #0f5897;
}

```

</style>

<h1>Cat Photo Finder</h1>

<p class="message">

 The message will go here

</p>

<p>

 <button id="getMessage">

 Get Message

 </button>

</p>

Solution

```
// solution required
```

6. Render Images from Data Sources

Description

The last few challenges showed that each object in the JSON array contains an `imageLink` key with a value that is the URL of a cat's image. When you're looping through these objects, you can use this `imageLink` property to display this image in an `img` element. Here's the code that does this:

```
html += "<img src = '" + val.imageLink + "' " + "alt='"
+ val.altText + "'>";
```

Instructions

Add code to use the `imageLink` and `altText` properties in an `img` tag.

Challenge Seed

```

<script>
  document.addEventListener('DOMContentLoaded',function(){
    document.getElementById('getMessage').onclick=function(){
      req=new XMLHttpRequest();
      req.open("GET",'./json/cats.json',true);
      req.send();
      req.onload=function(){
        json=JSON.parse(req.responseText);
        var html = "";
        json.forEach(function(val) {
          html += "<div class = 'cat'>";
          // Add your code below this line

          // Add your code above this line
          html += "</div><br>";
        });
        document.getElementsByClassName('message')[0].innerHTML=html;
      };
    };
  });

```

```

</script>
<style>
  body {
    text-align: center;
    font-family: "Helvetica", sans-serif;
  }
  h1 {
    font-size: 2em;
    font-weight: bold;
  }
  .box {
    border-radius: 5px;
    background-color: #eee;
    padding: 20px 5px;
  }
  button {
    color: white;
    background-color: #4791d0;
    border-radius: 5px;
    border: 1px solid #4791d0;
    padding: 5px 10px 8px 10px;
  }
  button:hover {
    background-color: #0f5897;
    border: 1px solid #0f5897;
  }
</style>
<h1>Cat Photo Finder</h1>
<p class="message">
  The message will go here
</p>
<p>
  <button id="getMessage">
    Get Message
  </button>
</p>

```

Solution

```
// solution required
```

7. Pre-filter JSON to Get the Data You Need

Description

If you don't want to render every cat photo you get from the freeCodeCamp Cat Photo API, you can pre-filter the JSON before looping through it. Given that the JSON data is stored in an array, you can use the `filter` method to filter out the cat whose "id" key has a value of 1. Here's the code to do this:

```

json = json.filter(function(val) {
  return (val.id !== 1);
});
```

Instructions

Add code to `filter` the json data to remove the cat with the "id" value of 1.

Challenge Seed

```

<script>
  document.addEventListener('DOMContentLoaded',function(){
    document.getElementById('getMessage').onclick=function(){
      req=new XMLHttpRequest();
      req.open("GET",'./json/cats.json',true);
      req.send();
```

```

req.onload=function(){
    json=JSON.parse(req.responseText);
    var html = "";
    // Add your code below this line

    // Add your code above this line
    json.forEach(function(val) {
        html += "<div class = 'cat'>

        html += "<img src = '" + val.imageLink + "' " + "alt='" + val.altText + "'>

        html += "</div>
    });
    document.getElementsByClassName('message')[0].innerHTML=html;
};

});
});

</script>
<style>
body {
    text-align: center;
    font-family: "Helvetica", sans-serif;
}
h1 {
    font-size: 2em;
    font-weight: bold;
}
.box {
    border-radius: 5px;
    background-color: #eee;
    padding: 20px 5px;
}
button {
    color: white;
    background-color: #4791d0;
    border-radius: 5px;
    border: 1px solid #4791d0;
    padding: 5px 10px 8px 10px;
}
button:hover {
    background-color: #0F5897;
    border: 1px solid #0F5897;
}
</style>
<h1>Cat Photo Finder</h1>
<p class="message">
    The message will go here
</p>
<p>
    <button id="getMessage">
        Get Message
    </button>
</p>

```

Solution

```
// solution required
```

8. Get Geolocation Data to Find A User's GPS Coordinates

Description

Another cool thing you can do is access your user's current location. Every browser has a built in navigator that can give you this information. The navigator will get the user's current longitude and latitude. You will see a prompt to allow or block this site from knowing your current location. The challenge can be completed either way, as long as the code is correct. By selecting allow, you will see the text on the output phone change to your latitude and longitude. Here's code that does this:

```

if (navigator.geolocation){
  navigator.geolocation.getCurrentPosition(function(position) {
    document.getElementById('data').innerHTML="latitude: " + position.coords.latitude + "<br>longitude: " +
    position.coords.longitude;
  });
}

```

First, it checks if the `navigator.geolocation` object exists. If it does, the `getCurrentPosition` method on that object is called, which initiates an asynchronous request for the user's position. If the request is successful, the callback function in the method runs. This function accesses the `position` object's values for latitude and longitude using dot notation and updates the HTML.

Instructions

Add the example code inside the `script` tags to check a user's current location and insert it into the HTML.

Challenge Seed

```

<script>
  // Add your code below this line

  // Add your code above this line
</script>
<h4>You are here:</h4>
<div id="data">

</div>

```

Solution

```
// solution required
```

9. Post Data with the JavaScript XMLHttpRequest Method

Description

In the previous examples, you received data from an external resource. You can also send data to an external resource, as long as that resource supports AJAX requests and you know the URL. JavaScript's `XMLHttpRequest` method is also used to post data to a server. Here's an example:

```

req=new XMLHttpRequest();
req.open("POST",url,true);
req.setRequestHeader('Content-Type','text/plain');
req.onreadystatechange=function(){
  if(req.readyState==4 && req.status==200){
    document.getElementsByClassName('message')[0].innerHTML=req.responseText;
  }
};
req.send(userName);

```

You've seen several of these methods before. Here the `open` method initializes the request as a "POST" to the given URL of the external resource, and uses the `true` Boolean to make it asynchronous. The `setRequestHeader` method sets the value of an HTTP request header, which contains information about the sender and the request. It must be called after the `open` method, but before the `send` method. The two parameters are the name of the header and the value to set as the body of that header. Next, the `onreadystatechange` event listener handles a change in the state of the request. A `readyState` of 4 means the operation is complete, and a `status` of 200 means it was a successful

request. The document's HTML can be updated. Finally, the `send` method sends the request with the `userName` value, which was given by the user in the `input` field.

Instructions

Update the code to create and send a "POST" request. Then enter your name in input box and click "Send Message". Your AJAX function will replace "Reply from Server will be here." with the reply of the server. In this case, it is your name appended with " loves cats".

Challenge Seed

```
<script>
  document.addEventListener('DOMContentLoaded',function(){
    document.getElementById('sendMessage').onclick=function(){

      var userName=document.getElementById('name').value;
      // Add your code below this line

      // Add your code above this line
    };
  });
</script>
<style>
  body {
    text-align: center;
    font-family: "Helvetica", sans-serif;
  }
  h1 {
    font-size: 2em;
    font-weight: bold;
  }
  .box {
    border-radius: 5px;
    background-color: #eee;
    padding: 20px 5px;
  }
  button {
    color: white;
    background-color: #4791d0;
    border-radius: 5px;
    border: 1px solid #4791d0;
    padding: 5px 10px 8px 10px;
  }
  button:hover {
    background-color: #0F5897;
    border: 1px solid #0F5897;
  }
</style>
<h1>Cat Friends</h1>
<p class="message">
  Reply from Server will be here
</p>
<p>
  <label for="name">Your name:<br/>
    <input type="text" id="name"/>
  </label>
  <button id="sendMessage">
    Send Message
  </button>
</p>
```

Solution

```
// solution required
```

Data Visualization Projects

1. Visualize Data with a Bar Chart

Description

Objective: Build a [CodePen.io](#) app that is functionally similar to this: <https://codepen.io/freeCodeCamp/full/GrZVaM>. Fulfill the below [user stories](#) and get all of the tests to pass. Give it your own personal style. You can use HTML, JavaScript, CSS, and the D3 svg-based visualization library. The tests require axes to be generated using the D3 axis property, which automatically generates ticks along the axis. These ticks are required for passing the D3 tests because their positions are used to determine alignment of graphed elements. You will find information about generating axes at <https://github.com/d3/d3/blob/master/API.md#axes-d3-axis>. Required (non-virtual) DOM elements are queried on the moment of each test. If you use a frontend framework (like Vue for example), the test results may be inaccurate for dynamic content. We hope to accommodate them eventually, but these frameworks are not currently supported for D3 projects.

User Story #1: My chart should have a title with a corresponding `id="title"`.

User Story #2: My chart should have a `g` element x-axis with a corresponding `id="x-axis"`.

User Story #3: My chart should have a `g` element y-axis with a corresponding `id="y-axis"`.

User Story #4: Both axes should contain multiple tick labels, each with the corresponding `class="tick"`.

User Story #5: My chart should have a `rect` element for each data point with a corresponding `class="bar"` displaying the data.

User Story #6: Each bar should have the properties `data-date` and `data-gdp` containing date and GDP values.

User Story #7: The bar elements' `data-date` properties should match the order of the provided data.

User Story #8: The bar elements' `data-gdp` properties should match the order of the provided data.

User Story #9: Each bar element's height should accurately represent the data's corresponding GDP.

User Story #10: The `data-date` attribute and its corresponding bar element should align with the corresponding value on the x-axis.

User Story #11: The `data-gdp` attribute and its corresponding bar element should align with the corresponding value on the y-axis.

User Story #12: I can mouse over an area and see a tooltip with a corresponding `id="tooltip"` which displays more information about the area.

User Story #13: My tooltip should have a `data-date` property that corresponds to the `data-date` of the active area.

Here is the dataset you will need to complete this project: <https://raw.githubusercontent.com/freeCodeCamp/ProjectReferenceData/master/GDP-data.json> You can build your project by forking [this CodePen pen](#). Or you can use this CDN link to run the tests in any environment you like: <https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js>. Once you're done, submit the URL to your working project with all its tests passing. Remember to use the [Read-Search-Ask](#) method if you get stuck.

Instructions

Challenge Seed

Solution

```
// solution required
```

2. Visualize Data with a Scatterplot Graph

Description

Objective: Build a [CodePen.io](#) app that is functionally similar to this: <https://codepen.io/freeCodeCamp/full/bgpXyK>. Fulfill the below [user stories](#) and get all of the tests to pass. Give it your own personal style. You can use HTML, JavaScript, CSS, and the D3 svg-based visualization library. The tests require axes to be generated using the D3 axis property, which automatically generates ticks along the axis. These ticks are required for passing the D3 tests because their positions are used to determine alignment of graphed elements. You will find information about generating axes at <https://github.com/d3/d3/blob/master/API.md#axes-d3-axis>. Required (non-virtual) DOM elements are queried on the moment of each test. If you use a frontend framework (like Vue for example), the test results may be inaccurate for dynamic content. We hope to accommodate them eventually, but these frameworks are not currently supported for D3 projects.

User Story #1: I can see a title element that has a corresponding `id="title"`.

User Story #2: I can

see an x-axis that has a corresponding `id="x-axis"`. **User Story #3:** I can see a y-axis that has a corresponding `id="y-axis"`. **User Story #4:** I can see dots, that each have a class of `dot`, which represent the data being plotted. **User Story #5:** Each dot should have the properties `data-xvalue` and `data-yvalue` containing their corresponding x and y values. **User Story #6:** The `data-xvalue` and `data-yvalue` of each dot should be within the range of the actual data and in the correct data format. For `data-xvalue`, integers (full years) or Date objects are acceptable for test evaluation. For `data-yvalue` (minutes), use Date objects. **User Story #7:** The `data-xvalue` and its corresponding dot should align with the corresponding point/value on the x-axis. **User Story #8:** The `data-yvalue` and its corresponding dot should align with the corresponding point/value on the y-axis. **User Story #9:** I can see multiple tick labels on the y-axis with `%M:%S` time format. **User Story #10:** I can see multiple tick labels on the x-axis that show the year. **User Story #11:** I can see that the range of the x-axis labels are within the range of the actual x-axis data. **User Story #12:** I can see that the range of the y-axis labels are within the range of the actual y-axis data. **User Story #13:** I can see a legend containing descriptive text that has `id="legend"`. **User Story #14:** I can mouse over an area and see a tooltip with a corresponding `id="tooltip"` which displays more information about the area. **User Story #15:** My tooltip should have a `data-year` property that corresponds to the `data-xvalue` of the active area. Here is the dataset you will need to complete this project:

<https://raw.githubusercontent.com/freeCodeCamp/ProjectReferenceData/master/cyclist-data.json> You can build your project by forking [this CodePen pen](#). Or you can use this CDN link to run the tests in any environment you like: <https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js> Once you're done, submit the URL to your working project with all its tests passing. Remember to use the [Read-Search-Ask](#) method if you get stuck.

Instructions

Challenge Seed

Solution

```
// solution required
```

3. Visualize Data with a Heat Map

Description

Objective: Build a [CodePen.io](#) app that is functionally similar to this: <https://codepen.io/freeCodeCamp/full/JExgeY>. Fulfill the below [user stories](#) and get all of the tests to pass. Give it your own personal style. You can use HTML, JavaScript, CSS, and the D3 svg-based visualization library. Required (non-virtual) DOM elements are queried on the moment of each test. If you use a frontend framework (like Vue for example), the test results may be inaccurate for dynamic content. We hope to accommodate them eventually, but these frameworks are not currently supported for D3 projects. **User Story #1:** My heat map should have a title with a corresponding `id="title"`. **User Story #2:** My heat map should have a description with a corresponding `id="description"`. **User Story #3:** My heat map should have an x-axis with a corresponding `id="x-axis"`. **User Story #4:** My heat map should have a y-axis with a corresponding `id="y-axis"`. **User Story #5:** My heat map should have `rect` elements with a `class="cell"` that represent the data. **User Story #6:** There should be at least 4 different fill colors used for the cells. **User Story #7:** Each cell will have the properties `data-month`, `data-year`, `data-temp` containing their corresponding month, year, and temperature values. **User Story #8:** The `data-month`, `data-year` of each cell should be within the range of the data. **User Story #9:** My heat map should have cells that align with the corresponding month on the y-axis. **User Story #10:** My heat map should have cells that align with the corresponding year on the x-axis. **User Story #11:** My heat map should have multiple tick labels on the y-axis with the full month name. **User Story #12:** My heat map should have multiple tick labels on the x-axis with the years between 1754 and 2015. **User Story #13:** My heat map should have a legend with a corresponding `id="legend"`. **User Story #14:** My legend should contain `rect` elements. **User Story #15:** The `rect` elements in the legend should use at least 4 different fill colors. **User Story #16:** I can mouse over an area and see a tooltip with a corresponding `id="tooltip"` which displays more information about the area. **User Story #17:** My tooltip should have a `data-year` property that corresponds to the `data-year` of the active area. Here is the dataset you will need to complete this project:

<https://raw.githubusercontent.com/freeCodeCamp/ProjectReferenceData/master/global-temperature.json> You can build your project by forking [this CodePen pen](#). Or you can use this CDN link to run the tests in any environment you

like: <https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js> Once you're done, submit the URL to your working project with all its tests passing. Remember to use the [Read-Search-Ask](#) method if you get stuck.

Instructions

Challenge Seed

Solution

```
// solution required
```

4. Visualize Data with a Choropleth Map

Description

Objective: Build a [CodePen.io](#) app that is functionally similar to this: <https://codepen.io/freeCodeCamp/full/EZKqza>. Fulfill the below [user stories](#) and get all of the tests to pass. Give it your own personal style. You can use HTML, JavaScript, CSS, and the D3 svg-based visualization library. Required (non-virtual) DOM elements are queried on the moment of each test. If you use a frontend framework (like Vue for example), the test results may be inaccurate for dynamic content. We hope to accommodate them eventually, but these frameworks are not currently supported for D3 projects. **User Story #1:** My choropleth should have a title with a corresponding `id="title"`. **User Story #2:** My choropleth should have a description element with a corresponding `id="description"`. **User Story #3:** My choropleth should have counties with a corresponding `class="county"` that represent the data. **User Story #4:** There should be at least 4 different fill colors used for the counties. **User Story #5:** My counties should each have `data-fips` and `data-education` properties containing their corresponding fips and education values. **User Story #6:** My choropleth should have a county for each provided data point. **User Story #7:** The counties should have `data-fips` and `data-education` values that match the sample data. **User Story #8:** My choropleth should have a legend with a corresponding `id="legend"`. **User Story #9:** There should be at least 4 different fill colors used for the legend. **User Story #10:** I can mouse over an area and see a tooltip with a corresponding `id="tooltip"` which displays more information about the area. **User Story #11:** My tooltip should have a `data-education` property that corresponds to the `data-education` of the active area. Here are the datasets you will need to complete this project:

- **US Education Data:** https://cdn.freecodecamp.org/testable-projects-fcc/data/choropleth_map/for_user_education.json
- **US County Data:** https://cdn.freecodecamp.org/testable-projects-fcc/data/choropleth_map/counties.json

You can build your project by forking [this CodePen pen](#). Or you can use this CDN link to run the tests in any environment you like: <https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js> Once you're done, submit the URL to your working project with all its tests passing. Remember to use the [Read-Search-Ask](#) method if you get stuck.

Instructions

Challenge Seed

Solution

```
// solution required
```

5. Visualize Data with a Treemap Diagram

Description

Objective: Build a [CodePen.io](https://codepen.io/freeCodeCamp/full/KaNGNR) app that is functionally similar to this: <https://codepen.io/freeCodeCamp/full/KaNGNR>. Fulfill the below [user stories](#) and get all of the tests to pass. Give it your own personal style. You can use HTML, JavaScript, CSS, and the D3 svg-based visualization library. The tests require axes to be generated using the D3 axis property, which automatically generates ticks along the axis. These ticks are required for passing the D3 tests because their positions are used to determine alignment of graphed elements. You will find information about generating axes at <https://github.com/d3/d3/blob/master/API.md#axes-d3-axis>. Required (non-virtual) DOM elements are queried on the moment of each test. If you use a frontend framework (like Vue for example), the test results may be inaccurate for dynamic content. We hope to accommodate them eventually, but these frameworks are not currently supported for D3 projects.

User Story #1: My tree map should have a title with a corresponding `id="title"`.

User Story #2: My tree map should have a description with a corresponding `id="description"`.

User Story #3: My tree map should have `rect` elements with a corresponding `class="tile"` that represent the data.

User Story #4: There should be at least 2 different fill colors used for the tiles.

User Story #5: Each tile should have the properties `data-name`, `data-category`, and `data-value` containing their corresponding name, category, and value.

User Story #6: The area of each tile should correspond to the `data-value` amount: tiles with a larger `data-value` should have a bigger area.

User Story #7: My tree map should have a legend with corresponding `id="legend"`.

User Story #8: My legend should have `rect` elements with a corresponding `class="legend-item"`.

User Story #9: The `rect` elements in the legend should use at least 2 different fill colors.

User Story #10: I can mouse over an area and see a tooltip with a corresponding `id="tooltip"` which displays more information about the area.

User Story #11: My tooltip should have a `data-value` property that corresponds to the `data-value` of the active area. For this project you can use any of the following datasets:

- **Kickstarter Pledges:** https://cdn.freecodecamp.org/testable-projects-fcc/data/tree_map/kickstarter-funding-data.json
- **Movie Sales:** https://cdn.freecodecamp.org/testable-projects-fcc/data/tree_map/movie-data.json
- **Video Game Sales:** https://cdn.freecodecamp.org/testable-projects-fcc/data/tree_map/video-game-sales-data.json

You can build your project by forking [this CodePen pen](#). Or you can use this CDN link to run the tests in any environment you like: <https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js>. Once you're done, submit the URL to your working project with all its tests passing. Remember to use the [Read-Search-Ask](#) method if you get stuck.

Instructions

Challenge Seed

Solution

```
// solution required
```

Apis And Microservices Certification

Managing Packages with Npm

1. How to Use package.json, the Core of Any Node.js Project or npm Package

Description

The file package.json is the center of any Node.js project or npm package. It stores information about your project just like the <head>-section in a HTML document describes the content of a webpage. The package.json consists of a single JSON-object where information is stored in "key": value-pairs. There are only two required fields in a minimal package.json - name and version - but it's a good practice to provide additional information about your project that could be useful to future users or maintainers. The author-field If you go to the Glitch project that you set up previously and look at on the left side of your screen, you'll find the file tree where you can see an overview of the various files in your project. Under the file tree's back-end section, you'll find package.json - the file that we'll be improving in the next couple of challenges. One of the most common pieces of information in this file is the author-field that specifies who's the creator of a project. It can either be a string or an object with contact details. The object is recommended for bigger projects but in our case, a simple string like the following example will do. "author": "Jane Doe", Instructions Add your name to the author-field in the package.json of your Glitch project. Remember that you're writing JSON. All field-names must use double-quotes ("), e.g. "author" All fields must be separated with a comma (,)

Instructions

Challenge Seed

Solution

```
// solution required
```

2. Add a Description to Your package.json

Description

The next part of a good package.json is the description-field, where a short but informative description about your project belongs. If you some day plan to publish a package to npm, remember that this is the string that should sell your idea to the user when they decide whether to install your package or not. However, that's not the only use case for the description: It's a great way to summarize what a project does, it's just as important for your normal Node.js-projects to help other developers, future maintainers or even your future self understand the project quickly. Regardless of what you plan for your project, a description is definitely recommended. Let's add something similar to this: "description": "A project that does something awesome", Instructions Add a description to the package.json in your Glitch project. Remember to use double-quotes for field-names ("") and commas (,) to separate fields.

Instructions

Challenge Seed

Solution

```
// solution required
```

3. Add Keywords to Your package.json

Description

The keywords-field is where you can describe your project using related keywords. Example "keywords": ["descriptive", "related", "words"], As you can see, this field is structured as an array of double-quoted strings. Instructions Add an array of suitable strings to the keywords-field in the package.json of your Glitch project. One of the keywords should be freecodecamp.

Instructions

Challenge Seed

Solution

```
// solution required
```

4. Add a License to Your package.json

Description

The license-field is where you inform users of your project what they are allowed to do with it. Some common licenses for open source projects include MIT and BSD. <http://choosealicense.com> is a great resource if you want to learn more about what license could fit your project. License information is not required. Copyright laws in most countries will give you ownership of what you create by default. However, it's always a good practice to explicitly state what users can and can't do. Example "license": "MIT", Instructions Fill the license-field in the package.json of your Glitch project as you find suitable.

Instructions

Challenge Seed

Solution

```
// solution required
```

5. Add a Version to Your package.json

Description

The version is together with name one of the required fields in a package.json. This field describes the current version of your project. Example "version": "1.2", Instructions Add a version to the package.json in your Glitch project.

Instructions

Challenge Seed

Solution

```
// solution required
```

6. Expand Your Project with External Packages from npm

Description

One of the biggest reasons to use a package manager is their powerful dependency management. Instead of manually having to make sure that you get all dependencies whenever you set up a project on a new computer, npm automatically installs everything for you. But how can npm know exactly what your project needs? Meet the dependencies-section of your package.json. In the dependencies-section, packages your project require are stored using the following format: "dependencies": { "package-name": "version", "express": "4.14.0" } Instructions Add version 2.14.0 of the package moment to the dependencies-field of your package.json Moment is a handy library for working with time and dates.

Instructions

Challenge Seed

Solution

```
// solution required
```

7. Manage npm Dependencies By Understanding Semantic Versioning

Description

Versions of the npm packages in the dependencies-section of your package.json follow what's called Semantic Versioning (SemVer), an industry standard for software versioning aiming to make it easier to manage dependencies. Libraries, frameworks or other tools published on npm should use SemVer in order to clearly communicate what kind of changes that projects who depend on the package can expect if they update. SemVer doesn't make sense in projects without public APIs - so unless your project is similar to the examples above, use another versioning format. So why do you need to understand SemVer? Knowing SemVer can be useful when you develop software that use external dependencies (which you almost always do). One day, your understanding of these numbers will save you from accidentally introducing breaking changes to your project without understanding why things "that worked yesterday" suddenly don't. This is how Semantic Versioning works according to the official website: Given a version number MAJOR.MINOR.PATCH, increment the: MAJOR version when you make incompatible API changes, MINOR version when you add functionality in a backwards-compatible manner, and PATCH version when you make backwards-compatible bug fixes. This means that PATCHes are bug fixes and MINORS add new features but neither of them break what worked before. Finally, MAJORS add changes that won't work with earlier versions. Example A semantic version number: 1.3.8 Instructions In the dependencies-section of your package.json, change the version of moment to match MAJOR version 2, MINOR version 10 and PATCH version 2

Instructions

Challenge Seed

Solution

```
// solution required
```

8. Use the Tilde-Character to Always Use the Latest Patch Version of a Dependency

Description

In the last challenge, we told npm to only include a specific version of a package. That's a useful way to freeze your dependencies if you need to make sure that different parts of your project stay compatible with each other. But in most use cases you don't want to miss bug fixes, since they often include important security patches and (hopefully) don't break things in doing so. To allow a npm dependency to get updated to the latest PATCH-version, you can prefix the dependency's version with the tilde-character (~). In package.json, our current rule for how npm may upgrade moment is to use a specific version only (2.10.2), but we want to allow the latest 2.10.x-version. Example "some-package-name": "~1.3.8" allows updates to any 1.3.x version. Instructions Use the tilde-character (~) to prefix the version of moment in your dependencies and allow npm to update it to any new PATCH release. Note that the version numbers themselves should not be changed.

Instructions

Challenge Seed

Solution

```
// solution required
```

9. Use the Caret-Character to Use the Latest Minor Version of a Dependency

Description

Similar to how the tilde (~) we learned about in the last challenge allow npm to install the latest PATCH for a dependency, the caret (^) allows npm to install future updates as well. The difference is that the caret will allow both MINOR updates and PATCHes. At the moment, your current version of moment should be ~2.10.2 which allows npm to install to the latest 2.10.x-version. If we instead were to use the caret (^) as our version prefix, npm would instead be allowed to update to any 2.x.x-version. Example "some-package-name": "^1.3.8" allows updates to any 1.x.x version. Instructions Use the caret-character (^) to prefix the version of moment in your dependencies and allow npm to update it to any new MINOR release. Note that the version numbers themselves not should be changed.

Instructions

Challenge Seed

Solution

```
// solution required
```

10. Remove a Package from Your Dependencies

Description

Now you've tested a few ways you can manage dependencies of your project by using the package.json's dependencies-section. You've included external packages by adding them to the file and even told npm what types of versions you want by using special characters as the tilde (~) or the caret (^). But what if you want to remove an external package that you no longer need? You might already have guessed it - Just remove the corresponding "key": value-pair for that from your dependencies. This same method applies to removing other fields in your package.json as well. Instructions Remove the package moment from your dependencies. Make sure you have the right amount of commas after removing it.

Instructions

Challenge Seed

Solution

```
// solution required
```

Basic Node and Express

1. Meet the Node console

Description

During the development process, it is important to be able to check what's going on in your code. Node is just a JavaScript environment. Like client side JavaScript, you can use the console to display useful debug information. On your local machine, you would see the console output in a terminal. On Glitch you can open the logs in the lower part of the screen. You can toggle the log panel with the button 'Logs' (top-left, under the app name). To get started, just print the classic "Hello World" in the console. We recommend to keep the log panel open while working at these challenges. Reading the logs you can be aware of the nature of the errors that may occur.

Instructions

Modify the `myApp.js` file to log "Hello World" to the console.

Challenge Seed

Solution

```
// solution required
```

2. Start a Working Express Server

Description

In the first two lines of the file `myApp.js` you can see how it's easy to create an Express app object. This object has several methods, and we will learn many of them in these challenges. One fundamental method is `app.listen(port)`.

It tells your server to listen on a given port, putting it in running state. You can see it at the bottom of the file. It is inside comments because for testing reasons we need the app to be running in background. All the code that you may want to add goes between these two fundamental parts. Glitch stores the port number in the environment variable `process.env.PORT`. Its value is `3000`. Let's serve our first string! In Express, routes takes the following structure: `app.METHOD(PATH, HANDLER)`. `METHOD` is an http method in lowercase. `PATH` is a relative path on the server (it can be a string, or even a regular expression). `HANDLER` is a function that Express calls when the route is matched. Handlers take the form `function(req, res) {...}`, where `req` is the request object, and `res` is the response object. For example, the handler

```
function(req, res) {  
  res.send('Response String');  
}
```

will serve the string 'Response String'. Use the `app.get()` method to serve the string Hello Express, to GET requests matching the / root path. Be sure that your code works by looking at the logs, then see the results in your browser, clicking the button 'Show Live' in the Glitch UI.

Instructions

Challenge Seed

Solution

```
// solution required
```

3. Serve an HTML File

Description

We can respond with a file using the method `res.sendFile(path)`. You can put it inside the `app.get('/', ...)` route handler. Behind the scenes this method will set the appropriate headers to instruct your browser on how to handle the file you want to send, according to its type. Then it will read and send the file. This method needs an absolute file path. We recommend you to use the Node global variable `__dirname` to calculate the path. e.g. `absolutePath = __dirname + relativePath/file.ext`. The file to send is `/views/index.html`. Try to 'Show Live' your app, you should see a big HTML heading (and a form that we will use later...), with no style applied. Note: You can edit the solution of the previous challenge, or create a new one. If you create a new solution, keep in mind that Express evaluates the routes from top to bottom. It executes the handler for the first match. You have to comment out the preceding solution, or the server will keep responding with a string.

Instructions

Challenge Seed

Solution

```
// solution required
```

4. Serve Static Assets

Description

An HTML server usually has one or more directories that are accessible by the user. You can place there the static assets needed by your application (stylesheets, scripts, images). In Express you can put in place this functionality using the middleware `express.static(path)`, where the parameter is the absolute path of the folder containing the assets. If you don't know what a middleware is, don't worry. We'll discuss about it later in details. Basically middlewares are functions that intercept route handlers, adding some kind of information. A middleware needs to be mounted using the method `app.use(path, middlewareFunction)`. The first path argument is optional. If you don't pass it, the middleware will be executed for all the requests. Mount the `express.static()` middleware for all the requests with `app.use()`. The absolute path to the assets folder is `__dirname + /public`. Now your app should be able to serve a CSS stylesheet. From outside the public folder will appear mounted to the root directory. Your front-page should look a little better now!

Instructions

Challenge Seed

Solution

```
// solution required
```

5. Serve JSON on a Specific Route

Description

While an HTML server serves (you guessed it!) HTML, an API serves data. A REST (REpresentational State Transfer) API allows data exchange in a simple way, without the need for clients to know any detail about the server. The client only needs to know where the resource is (the URL), and the action it wants to perform on it (the verb). The GET verb is used when you are fetching some information, without modifying anything. These days, the preferred data format for moving information around the web is JSON. Simply put, JSON is a convenient way to represent a JavaScript object as a string, so it can be easily transmitted. Let's create a simple API by creating a route that responds with JSON at the path `/json`. You can do it as usual, with the `app.get()` method. Inside the route handler use the method `res.json()`, passing in an object as an argument. This method closes the request-response loop, returning the data. Behind the scenes it converts a valid JavaScript object into a string, then sets the appropriate headers to tell your browser that you are serving JSON, and sends the data back. A valid object has the usual structure `{key: data}`. Data can be a number, a string, a nested object or an array. Data can also be a variable or the result of a function call, in which case it will be evaluated before being converted into a string. Serve the object `{"message": "Hello json"}` as a response in JSON format, to the GET requests to the route `/json`. Then point your browser to `your-app-url/json`, you should see the message on the screen.

Instructions

Challenge Seed

Solution

```
// solution required
```

6. Use the .env File

Description

The `.env` file is a hidden file that is used to pass environment variables to your application. This file is secret, no one but you can access it, and it can be used to store data that you want to keep private or hidden. For example, you can store API keys from external services or your database URI. You can also use it to store configuration options. By setting configuration options, you can change the behavior of your application, without the need to rewrite some code. The environment variables are accessible from the app as `process.env.VAR_NAME`. The `process.env` object is a global Node object, and variables are passed as strings. By convention, the variable names are all uppercase, with words separated by an underscore. The `.env` is a shell file, so you don't need to wrap names or values in quotes. It is also important to note that there cannot be space around the equals sign when you are assigning values to your variables, e.g. `VAR_NAME=value`. Usually, you will put each variable definition on a separate line. Let's add an environment variable as a configuration option. Store the variable `MESSAGE_STYLE=uppercase` in the `.env` file. Then tell the `GET /json` route handler that you created in the last challenge to transform the response object's message to uppercase if `process.env.MESSAGE_STYLE` equals `uppercase`. The response object should become `{"message": "HELLO JSON"}`.

Instructions

Challenge Seed

Solution

```
// solution required
```

7. Implement a Root-Level Request Logger Middleware

Description

Before we introduced the `express.static()` middleware function. Now it's time to see what middleware is, in more detail. Middleware functions are functions that take 3 arguments: the request object, the response object, and the `next` function in the application's request-response cycle. These functions execute some code that can have side effects on the app, and usually add informations to the request or response objects. They can also end the cycle sending the response, when some condition is met. If they don't send the response, when they are done they start the execution of the next function in the stack. This is triggered calling the 3rd argument `next()`. More information in the [express documentation](#). Look at the following example :

```
function(req, res, next) {
  console.log("I'm a middleware...");
  next();
}
```

Let's suppose we mounted this function on a route. When a request matches the route, it displays the string "I'm a middleware...". Then it executes the `next` function in the stack. In this exercise we are going to build a root-level middleware. As we have seen in challenge 4, to mount a middleware function at root level we can use the method `app.use(<mware-function>)`. In this case the function will be executed for all the requests, but you can also set more specific conditions. For example, if you want a function to be executed only for POST requests, you could use `app.post(<mware-function>)`. Analogous methods exist for all the http verbs (GET, DELETE, PUT, ...). Build a simple logger. For every request, it should log in the console a string taking the following format: `method path - ip`. An example would look like: `GET /json - ::ffff:127.0.0.1`. Note that there is a space between `method` and `path` and that the dash separating `path` and `ip` is surrounded by a space on either side. You can get the request method (http verb), the relative route path, and the caller's ip from the request object, using `req.method`, `req.path` and `req.ip`. Remember to call `next()` when you are done, or your server will be stuck forever. Be sure to have the 'Logs' opened, and see what happens when some request arrives... Hint: Express evaluates functions in the order they appear in the code. This is true for middleware too. If you want it to work for all the routes, it should be mounted before them.

Instructions

Challenge Seed

Solution

```
// solution required
```

8. Chain Middleware to Create a Time Server

Description

Middleware can be mounted at a specific route using `app.METHOD(path, middlewareFunction)`. Middleware can also be chained inside route definition. Look at the following example:

```
app.get('/user', function(req, res, next) {
  req.user = getTheUserSync(); // Hypothetical synchronous operation
  next();
}, function(req, res) {
  res.send(req.user);
})
```

This approach is useful to split the server operations into smaller units. That leads to a better app structure, and the possibility to reuse code in different places. This approach can also be used to perform some validation on the data. At each point of the middleware stack you can block the execution of the current chain and pass control to functions specifically designed to handle errors. Or you can pass control to the next matching route, to handle special cases. We will see how in the advanced Express section. In the route `app.get('/now', ...)` chain a middleware function and the final handler. In the middleware function you should add the current time to the request object in the `req.time` key. You can use `new Date().toString()`. In the handler, respond with a JSON object, taking the structure `{time: req.time}`. Hint: The test will not pass if you don't chain the middleware. If you mount the function somewhere else, the test will fail, even if the output result is correct.

Instructions

Challenge Seed

Solution

```
// solution required
```

9. Get Route Parameter Input from the Client

Description

When building an API, we have to allow users to communicate to us what they want to get from our service. For example, if the client is requesting information about a user stored in the database, they need a way to let us know which user they're interested in. One possible way to achieve this result is by using route parameters. Route parameters are named segments of the URL, delimited by slashes (/). Each segment captures the value of the part of the URL which matches its position. The captured values can be found in the `req.params` object.

```
route_path: '/user/:userId/book/:bookId'
actual_request_URL: '/user/546/book/6754'
req.params: {userId: '546', bookId: '6754'}
```

Build an echo server, mounted at the route `GET /:word/echo`. Respond with a JSON object, taking the structure `{echo: word}`. You can find the word to be repeated at `req.params.word`. You can test your route from your browser's address bar, visiting some matching routes, e.g. `your-app-rootpath/freecodecamp/echo`

Instructions

Challenge Seed

Solution

```
// solution required
```

10. Get Query Parameter Input from the Client

Description

Another common way to get input from the client is by encoding the data after the route path, using a query string. The query string is delimited by a question mark (?), and includes field=value couples. Each couple is separated by an ampersand (&). Express can parse the data from the query string, and populate the object `req.query`. Some characters cannot be in URLs, they have to be encoded in a [different format](#) before you can send them. If you use the API from JavaScript, you can use specific methods to encode/decode these characters.

```
route_path: '/library'  
actual_request_URL: '/library?userId=546&bookId=6754'  
req.query: {userId: '546', bookId: '6754'}
```

Build an API endpoint, mounted at `GET /name`. Respond with a JSON document, taking the structure `{ name: 'firstname lastname' }`. The first and last name parameters should be encoded in a query string e.g. `?first=firstname&last=lastname`. TIP: In the following exercise we are going to receive data from a POST request, at the same `/name` route path. If you want you can use the method `app.route(path).get(handler).post(handler)`. This syntax allows you to chain different verb handlers on the same path route. You can save a bit of typing, and have cleaner code.

Instructions

Challenge Seed

Solution

```
// solution required
```

11. Use body-parser to Parse POST Requests

Description

Besides GET there is another common http verb, it is POST. POST is the default method used to send client data with HTML forms. In the REST convention POST is used to send data to create new items in the database (a new user, or a new blog post). We don't have a database in this project, but we are going to learn how to handle POST requests anyway. In these kind of requests the data doesn't appear in the URL, it is hidden in the request body. This is a part of the HTML request, also called payload. Since HTML is text based, even if you don't see the data, it doesn't mean that they are secret. The raw content of an HTTP POST request is shown below:

```
POST /path/subpath HTTP/1.0  
From: john@example.com  
User-Agent: someBrowser/1.0  
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 20
name=John+Doe&age=25
```

As you can see the body is encoded like the query string. This is the default format used by HTML forms. With Ajax we can also use JSON to be able to handle data having a more complex structure. There is also another type of encoding: multipart/form-data. This one is used to upload binary files. In this exercise we will use an urlencoded body. To parse the data coming from POST requests, you have to install a package: the body-parser. This package allows you to use a series of middleware, which can decode data in different formats. See the docs [here](#). Install the body-parser module in your package.json. Then require it at the top of the file. Store it in a variable named bodyParser. The middleware to handle url encoded data is returned by `bodyParser.urlencoded({extended: false})`. `extended=false` is a configuration option that tells the parser to use the classic encoding. When using it, values can be only strings or arrays. The extended version allows more data flexibility, but it is outmatched by JSON. Pass to `app.use()` the function returned by the previous method call. As usual, the middleware must be mounted before all the routes which need it.

Instructions

Challenge Seed

Solution

```
// solution required
```

12. Get Data from POST Requests

Description

Mount a POST handler at the path `/name`. It's the same path as before. We have prepared a form in the html frontpage. It will submit the same data of exercise 10 (Query string). If the body-parser is configured correctly, you should find the parameters in the object `req.body`. Have a look at the usual library example:

```
route: POST '/library'
urlencoded_body: userId=546&bookId=6754
req.body: {userId: '546', bookId: '6754'}
```

Respond with the same JSON object as before: `{name: 'firstname lastname'}`. Test if your endpoint works using the html form we provided in the app frontpage. Tip: There are several other http methods other than GET and POST. And by convention there is a correspondence between the http verb, and the operation you are going to execute on the server. The conventional mapping is: POST (sometimes PUT) - Create a new resource using the information sent with the request, GET - Read an existing resource without modifying it, PUT or PATCH (sometimes POST) - Update a resource using the data sent, DELETE => Delete a resource. There are also a couple of other methods which are used to negotiate a connection with the server. Except from GET, all the other methods listed above can have a payload (i.e. the data into the request body). The body-parser middleware works with these methods as well.

Instructions

Challenge Seed

Solution

```
// solution required
```

MongoDB and Mongoose

1. Install and Set Up Mongoose

Description

Add mongodb and mongoose to the project's package.json. Then require mongoose. Store your mLab database URI in the private .env file as MONGO_URI. Connect to the database using mongoose.connect()

Instructions

Challenge Seed

Solution

```
// solution required
```

2. Create a Model

Description

First of all we need a Schema. Each schema maps to a MongoDB collection. It defines the shape of the documents within that collection. Schemas are building block for Models. They can be nested to create complex models, but in this case we'll keep things simple. A model allows you to create instances of your objects, called documents. Create a person having this prototype : - Person Prototype - -- name : string [required] age : number favoriteFoods : array of strings (*) Use the mongoose basic schema types. If you want you can also add more fields, use simple validators like required or unique, and set default values. See the [mongoose docs](#). [C]RUD Part I - CREATE Note: Glitch is a real server, and in real servers the interactions with the db happen in handler functions. These function are executed when some event happens (e.g. someone hits an endpoint on your API). We'll follow the same approach in these exercises. The done() function is a callback that tells us that we can proceed after completing an asynchronous operation such as inserting, searching, updating or deleting. It's following the Node convention and should be called as done(null, data) on success, or done(err) on error. Warning - When interacting with remote services, errors may occur! /* Example */ var someFunc = function(done) { //... do something (risky) ... if(error) return done(error); done(null, result); };

Instructions

Challenge Seed

Solution

```
// solution required
```

3. Create and Save a Record of a Model

Description

Create a document instance using the Person constructor you built before. Pass to the constructor an object having the fields name, age, and favoriteFoods. Their types must be conformant to the ones in the Person Schema. Then call the method document.save() on the returned document instance. Pass to it a callback using the Node convention. This

```
is a common pattern, all the following CRUD methods take a callback function like this as the last argument. /*  
Example */ // ... person.save(function(err, data) { // ...do your stuff here... });
```

Instructions

Challenge Seed

Solution

```
// solution required
```

4. Create Many Records with model.create()

Description

Sometimes you need to create many instances of your models, e.g. when seeding a database with initial data. Model.create() takes an array of objects like [{name: 'John', ...}, {...}, ...] as the first argument, and saves them all in the db. Create many people with Model.create(), using the function argument arrayOfPeople.

Instructions

Challenge Seed

Solution

```
// solution required
```

5. Use model.find() to Search Your Database

Description

Find all the people having a given name, using Model.find() -> [Person] In its simplest usage, Model.find() accepts a query document (a JSON object) as the first argument, then a callback. It returns an array of matches. It supports an extremely wide range of search options. Check it in the docs. Use the function argument personName as search key.

Instructions

Challenge Seed

Solution

```
// solution required
```

6. Use model.findOne() to Return a Single Matching Document from Your Database

Description

Model.findOne() behaves like .find(), but it returns only one document (not an array), even if there are multiple items. It is especially useful when searching by properties that you have declared as unique. Find just one person which has a certain food in her favorites, using Model.findOne() -> Person. Use the function argument food as search key.

Instructions

Challenge Seed

Solution

```
// solution required
```

7. Use model.findById() to Search Your Database By _id

Description

When saving a document, mongodb automatically adds the field _id, and set it to a unique alphanumeric key. Searching by _id is an extremely frequent operation, so mongoose provides a dedicated method for it. Find the (only!!) person having a given _id, using Model.findById() -> Person. Use the function argument personId as search key.

Instructions

Challenge Seed

Solution

```
// solution required
```

8. Perform Classic Updates by Running Find, Edit, then Save

Description

In the good old days this was what you needed to do if you wanted to edit a document and be able to use it somehow e.g. sending it back in a server response. Mongoose has a dedicated updating method : Model.update(). It is binded to the low-level mongo driver. It can bulk edit many documents matching certain criteria, but it doesn't send back the updated document, only a 'status' message. Furthermore it makes model validations difficult, because it just directly calls the mongo driver. Find a person by _id (use any of the above methods) with the parameter personId as search key. Add "hamburger" to the list of her favoriteFoods (you can use Array.push()). Then - inside the find callback - save() the updated Person. [*] Hint: This may be tricky if in your Schema you declared favoriteFoods as an Array, without specifying the type (i.e. [String]). In that case favoriteFoods defaults to Mixed type, and you have to manually mark it as edited using document.markModified('edited-field'). ([#Mixed](http://mongoosejs.com/docs/schematypes.html))

Instructions

Challenge Seed

Solution

```
// solution required
```

9. Perform New Updates on a Document Using model.findOneAndUpdate()

Description

Recent versions of mongoose have methods to simplify documents updating. Some more advanced features (i.e. pre/post hooks, validation) behave differently with this approach, so the Classic method is still useful in many situations. `findByIdAndUpdate()` can be used when searching by Id. Find a person by Name and set her age to 20. Use the function parameter `personName` as search key. Hint: We want you to return the updated document. To do that you need to pass the options document `{ new: true }` as the 3rd argument to `findOneAndUpdate()`. By default these methods return the unmodified object.

Instructions

Challenge Seed

Solution

```
// solution required
```

10. Delete One Document Using model.findByIdAndRemove

Description

Delete one person by her `_id`. You should use one of the methods `findByIdAndRemove()` or `findOneAndRemove()`. They are like the previous update methods. They pass the removed document to the cb. As usual, use the function argument `personId` as search key.

Instructions

Challenge Seed

Solution

```
// solution required
```

11. Delete Many Documents with model.remove()

Description

Model.remove() is useful to delete all the documents matching given criteria. Delete all the people whose name is "Mary", using Model.remove(). Pass it to a query document with the "name" field set, and of course a callback. Note: Model.remove() doesn't return the deleted document, but a JSON object containing the outcome of the operation, and the number of items affected. Don't forget to pass it to the done() callback, since we use it in tests.

Instructions

Challenge Seed

Solution

```
// solution required
```

12. Chain Search Query Helpers to Narrow Search Results

Description

If you don't pass the callback as the last argument to Model.find() (or to the other search methods), the query is not executed. You can store the query in a variable for later use. This kind of object enables you to build up a query using chaining syntax. The actual db search is executed when you finally chain the method .exec(). Pass your callback to this last method. There are many query helpers, here we'll use the most 'famous' ones. Find people who like "burrito". Sort them by name, limit the results to two documents, and hide their age. Chain .find(), .sort(), .limit(), .select(), and then .exec(). Pass the done(err, data) callback to exec().

Instructions

Challenge Seed

Solution

```
// solution required
```

Apis and Microservices Projects

1. Timestamp Microservice

Description

Build a full stack JavaScript app that is functionally similar to this: <https://curse-arrow.glitch.me/>. Working on this project will involve you writing your code on Glitch on our starter project. After completing this project you can copy your public glitch url (to the homepage of your app) into this screen to test it! Optionally you may choose to write your project on another platform but it must be publicly visible for our testing. Start this project on Glitch using [this link](#) or clone [this repository](#) on GitHub! If you use Glitch, remember to save the link to your project somewhere safe!

Instructions

Challenge Seed

Solution

```
// solution required
```

2. Request Header Parser Microservice

Description

Build a full stack JavaScript app that is functionally similar to this: <https://dandelion-roar.glitch.me/>. Working on this project will involve you writing your code on Glitch on our starter project. After completing this project you can copy your public glitch url (to the homepage of your app) into this screen to test it! Optionally you may choose to write your project on another platform but it must be publicly visible for our testing. Start this project on Glitch using [this link](#) or clone [this repository](#) on GitHub! If you use Glitch, remember to save the link to your project somewhere safe!

Instructions

Challenge Seed

Solution

```
// solution required
```

3. URL Shortener Microservice

Description

Build a full stack JavaScript app that is functionally similar to this: <https://thread-paper.glitch.me/>. Working on this project will involve you writing your code on Glitch on our starter project. After completing this project you can copy your public glitch url (to the homepage of your app) into this screen to test it! Optionally you may choose to write your project on another platform but it must be publicly visible for our testing. Start this project on Glitch using [this link](#) or clone [this repository](#) on GitHub! If you use Glitch, remember to save the link to your project somewhere safe!

Instructions

Challenge Seed

Solution

```
// solution required
```

4. Exercise Tracker

Description

Build a full stack JavaScript app that is functionally similar to this: <https://fuschia-custard.glitch.me/>. Working on this project will involve you writing your code on Glitch on our starter project. After completing this project you can copy your public glitch url (to the homepage of your app) into this screen to test it! Optionally you may choose to write your

project on another platform but it must be publicly visible for our testing. Start this project on Glitch using [this link](#) or clone [this repository](#) on GitHub! If you use Glitch, remember to save the link to your project somewhere safe!

Instructions

Challenge Seed

Solution

```
// solution required
```

5. File Metadata Microservice

Description

Build a full stack JavaScript app that is functionally similar to this: <https://purple-paladin.glitch.me/>. Working on this project will involve you writing your code on Glitch on our starter project. After completing this project you can copy your public glitch url (to the homepage of your app) into this screen to test it! Optionally you may choose to write your project on another platform but it must be publicly visible for our testing. Start this project on Glitch using [this link](#) or clone [this repository](#) on GitHub! If you use Glitch, remember to save the link to your project somewhere safe!

Instructions

Challenge Seed

Solution

```
// solution required
```

Information Security And Quality Assurance Certification

Information Security with HelmetJS

1. Install and Require Helmet

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Helmet helps you secure your Express apps by setting various HTTP headers. Install the package, then require it.

Instructions

Challenge Seed

Solution

```
// solution required
```

2. Hide Potentially Dangerous Information Using helmet.hidePoweredBy()

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Hackers can exploit known vulnerabilities in Express/Node if they see that your site is powered by Express. X-Powered-By: Express is sent in every request coming from Express by default. The helmet.hidePoweredBy() middleware will remove the X-Powered-By header. You can also explicitly set the header to something else, to throw people off. e.g.
app.use(helmet.hidePoweredBy({ setTo: 'PHP 4.2.0' }))

Instructions

Challenge Seed

Solution

```
// solution required
```

3. Mitigate the Risk of Clickjacking with helmet.frameguard()

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Your page could be put in a `<frame>` or `<iframe>` without your consent. This can result in clickjacking attacks, among other things. Clickjacking is a technique of tricking a user into interacting with a page different from what the user thinks it

is. This can be obtained executing your page in a malicious context, by mean of iframing. In that context a hacker can put a hidden layer over your page. Hidden buttons can be used to run bad scripts. This middleware sets the X-Frame-Options header. It restricts who can put your site in a frame. It has three modes: DENY, SAMEORIGIN, and ALLOW-FROM. We don't need our app to be framed. You should use `helmet.frameguard()` passing with the configuration object `{action: 'deny'}`.

Instructions

Challenge Seed

Solution

```
// solution required
```

4. Mitigate the Risk of Cross Site Scripting (XSS) Attacks with `helmet.xssFilter()`

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Cross-site scripting (XSS) is a frequent type of attack where malicious scripts are injected into vulnerable pages, with the purpose of stealing sensitive data like session cookies, or passwords. The basic rule to lower the risk of an XSS attack is simple: "Never trust user's input". As a developer you should always sanitize all the input coming from the outside. This includes data coming from forms, GET query urls, and even from POST bodies. Sanitizing means that you should find and encode the characters that may be dangerous e.g. <, >. Modern browsers can help mitigating the risk by adopting better software strategies. Often these are configurable via http headers. The X-XSS-Protection HTTP header is a basic protection. The browser detects a potential injected script using a heuristic filter. If the header is enabled, the browser changes the script code, neutralizing it. It still has limited support.

Instructions

Challenge Seed

Solution

```
// solution required
```

5. Avoid Inferring the Response MIME Type with `helmet.noSniff()`

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Browsers can use content or MIME sniffing to adapt to different datatypes coming from a response. They override the Content-Type headers to guess and process the data. While this can be convenient in some scenarios, it can also lead to some dangerous attacks. This middleware sets the X-Content-Type-Options header to nosniff. This instructs the browser to not bypass the provided Content-Type.

Instructions

Challenge Seed

Solution

```
// solution required
```

6. Prevent IE from Opening Untrusted HTML with helmet.ieNoOpen()

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Some web applications will serve untrusted HTML for download. Some versions of Internet Explorer by default open those HTML files in the context of your site. This means that an untrusted HTML page could start doing bad things in the context of your pages. This middleware sets the X-Download-Options header to noopener. This will prevent IE users from executing downloads in the trusted site's context.

Instructions

Challenge Seed

Solution

```
// solution required
```

7. Ask Browsers to Access Your Site via HTTPS Only with helmet.hsts()

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). HTTP Strict Transport Security (HSTS) is a web security policy which helps to protect websites against protocol downgrade attacks and cookie hijacking. If your website can be accessed via HTTPS you can ask user's browsers to avoid using insecure HTTP. By setting the header Strict-Transport-Security, you tell the browsers to use HTTPS for the future requests in a specified amount of time. This will work for the requests coming after the initial request. Configure helmet.hsts() to use HTTPS for the next 90 days. Pass the config object {maxAge: timeInMilliseconds, force: true}. Glitch already has hsts enabled. To override its settings you need to set the field "force" to true in the config object. We will intercept and restore the Glitch header, after inspecting it for testing. Note: Configuring HTTPS on a custom website requires the acquisition of a domain, and a SSL/TSL Certificate.

Instructions

Challenge Seed

Solution

```
// solution required
```

8. Disable DNS Prefetching with helmet.dnsPrefetchControl()

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). To improve performance, most browsers prefetch DNS records for the links in a page. In that way the destination ip is already known when the user clicks on a link. This may lead to over-use of the DNS service (if you own a big website, visited by millions people...), privacy issues (one eavesdropper could infer that you are on a certain page), or page statistics alteration (some links may appear visited even if they are not). If you have high security needs you can disable DNS prefetching, at the cost of a performance penalty.

Instructions

Challenge Seed

Solution

```
// solution required
```

9. Disable Client-Side Caching with helmet.noCache()

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). If you are releasing an update for your website, and you want the users to always download the newer version, you can (try to) disable caching on client's browser. It can be useful in development too. Caching has performance benefits, which you will lose, so only use this option when there is a real need.

Instructions

Challenge Seed

Solution

```
// solution required
```

10. Set a Content Security Policy with helmet.contentSecurityPolicy()

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). This challenge highlights one promising new defense that can significantly reduce the risk and impact of many type of attacks in modern browsers. By setting and configuring a Content Security Policy you can prevent the injection of anything unintended into your page. This will protect your app from XSS vulnerabilities, undesired tracking, malicious frames, and much more. CSP works by defining a whitelist of content sources which are trusted. You can configure

them for each kind of resource a web page may need (scripts, stylesheets, fonts, frames, media, and so on...). There are multiple directives available, so a website owner can have a granular control. See [HTML 5 Rocks](#), [KeyCDN](#) for more details. Unfortunately CSP is unsupported by older browser. By default, directives are wide open, so it's important to set the defaultSrc directive as a fallback. Helmet supports both defaultSrc and default-src naming styles. The fallback applies for most of the unspecified directives. In this exercise, use helmet.contentSecurityPolicy(), and configure it setting the defaultSrc directive to ["self"] (the list of allowed sources must be in an array), in order to trust only your website address by default. Set also the scriptSrc directive so that you will allow scripts to be downloaded from your website, and from the domain 'trusted-cdn.com'. Hint: in the "self" keyword, the single quotes are part of the keyword itself, so it needs to be enclosed in double quotes to be working.

Instructions

Challenge Seed

Solution

```
// solution required
```

11. Configure Helmet Using the 'parent' helmet() Middleware

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). app.use(helmet()) will automatically include all the middleware introduced above, except noCache(), and contentSecurityPolicy(), but these can be enabled if necessary. You can also disable or configure any other middleware individually, using a configuration object. // Example app.use(helmet({ frameguard: { // configure action: 'deny' }, contentSecurityPolicy: { // enable and configure directives: { defaultSrc: ["self"], styleSrc: ['style.com'] }, dnsPrefetchControl: false // disable } })) We introduced each middleware separately for teaching purpose, and for ease of testing. Using the 'parent' helmet() middleware is easiest, and cleaner, for a real project.

Instructions

Challenge Seed

Solution

```
// solution required
```

12. Understand BCrypt Hashes

Description

For the following challenges, you will be working with a new starter project that is different from earlier challenges. This project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). BCrypt hashes are very secure. A hash is basically a fingerprint of the original data- always unique. This is accomplished by feeding the original data into a algorithm and having returned a fixed length result. To further complicate this process and make it more secure, you can also *salt* your hash. Salting your hash involves adding random data to the original data before the hashing process which makes it even harder to crack the hash. BCrypt hashes will always looks like

`$2a$13$ZyprE5MRw2Q3WpN0GZWGbeG7ADUre1Q8Q0.uUUtcbq1oU0yvzav0m` which does have a structure. The first small bit of data `$2a` is defining what kind of hash algorithm was used. The next portion `$13` defines the *cost*. Cost is about how much power it takes to compute the hash. It is on a logarithmic scale of 2^{cost} and determines how many times the data is put through the hashing algorithm. For example, at a cost of 10 you are able to hash 10 passwords a second on an average computer, however at a cost of 15 it takes 3 seconds per hash... and to take it further, at a cost of 31 it would take multiple days to complete a hash. A cost of 12 is considered very secure at this time. The last portion of your hash `$ZyprE5MRw2Q3WpN0GZWGbeG7ADUre1Q8Q0.uUUtcbq1oU0yvzav0m`, looks like 1 large string of numbers, periods, and letters but it is actually 2 separate pieces of information. The first 22 characters is the salt in plain text, and the rest is the hashed password!

To begin using BCrypt, add it as a dependency in your project and require it as 'bcrypt' in your server. Submit your page when you think you've got it right.

Instructions

Challenge Seed

Solution

```
// solution required
```

13. Hash and Compare Passwords Asynchronously

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). As hashing is designed to be computationally intensive, it is recommended to do so asynchronously on your server as to avoid blocking incoming connections while you hash. All you have to do to hash a password asynchronous is call `bcrypt.hash(myPlaintextPassword, saltRounds, (err, hash) => { /*Store hash in your db*/ });`

Add this hashing function to your server (we've already defined the variables used in the function for you to use) and log it to the console for you to see! At this point you would normally save the hash to your database. Now when you need to figure out if a new input is the same data as the hash you would just use the compare function `bcrypt.compare(myPlaintextPassword, hash, (err, res) => { /*res == true or false*/ });`. Add this into your existing hash function (since you need to wait for the hash to complete before calling the compare function) after you log the completed hash and log 'res' to the console within the compare. You should see in the console a hash then 'true' is printed! If you change 'myPlaintextPassword' in the compare function to 'someOtherPlaintextPassword' then it should say false.

```
bcrypt.hash('passw0rd!', 13, (err, hash) => {
  console.log(hash); // $2a$12$Y.PHPE15wR25qrرتgGkiYe2sXo98cjuMCG1YwSI5rJW1DSJp0gEYS
  bcrypt.compare('passw0rd!', hash, (err, res) => {
    console.log(res); // true
  });
});
```

Submit your page when you think you've got it right.

Instructions

Challenge Seed

Solution

```
// solution required
```

14. Hash and Compare Passwords Synchronously

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Hashing synchronously is just as easy to do but can cause lag if using it server side with a high cost or with hashing done very often. Hashing with this method is as easy as calling `var hash = bcrypt.hashSync(myPlaintextPassword, saltRounds);`

Add this method of hashing to your code and then log the result to the console. Again, the variables used are already defined in the server so you won't need to adjust them. You may notice even though you are hashing the same password as in the `async` function, the result in the console is different- this is due to the salt being randomly generated each time as seen by the first 22 characters in the third string of the hash. Now to compare a password input with the new sync hash, you would use the `compareSync` method: `var result = bcrypt.compareSync(myPlaintextPassword, hash);` with the result being a boolean true or false. Add this function in and log to the console the result to see it working. Submit your page when you think you've got it right. If you ran into errors during these challenges you can take a look at the example completed code [here](#).

Instructions

Challenge Seed

Solution

```
// solution required
```

Quality Assurance and Testing with Chai

1. Learn How JavaScript Assertions Work

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Use `assert.isNull()` or `assert.isNotNull()` to make the tests pass.

Instructions

Challenge Seed

Solution

```
// solution required
```

2. Test if a Variable or Function is Defined

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Use assert.isDefined() or assert.isUndefined() to make the tests pass

Instructions

Challenge Seed

Solution

```
// solution required
```

3. Use Assert.isOK and Assert.isNotOK

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Use assertisOk() or assert.isNotOk() to make the tests pass. .isOk(truthy) and .isNotOk(falsey) will pass.

Instructions

Challenge Seed

Solution

```
// solution required
```

4. Test for Truthiness

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Use assert.isTrue() or assert.isNotTrue() to make the tests pass. .isTrue(true) and .isNotTrue(everything else) will pass. .isFalse() and .isNotFalse() also exist.

Instructions

Challenge Seed

Solution

```
// solution required
```

5. Use the Double Equals to Assert Equality

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). `.equal()`, `.notEqual()` `.equal()` compares objects using `'=='`

Instructions

Challenge Seed

Solution

```
// solution required
```

6. Use the Triple Equals to Assert Strict Equality

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). `.strictEqual()`, `.notStrictEqual()` `.strictEqual()` compares objects using `'==='`

Instructions

Challenge Seed

Solution

```
// solution required
```

7. Assert Deep Equality with `.deepEqual` and `.notDeepEqual`

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). `.deepEqual()`, `.notDeepEqual()` `.deepEqual()` asserts that two object are deep equal

Instructions

Challenge Seed

Solution

```
// solution required
```

8. Compare the Properties of Two Elements

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#).
.isAbove()
=> a > b , .isAtMost() => a <= b

Instructions

Challenge Seed

Solution

```
// solution required
```

9. Test if One Value is Below or At Least as Large as Another

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#).
.isBelow()
=> a < b , .isAtLeast => a >= b

Instructions

Challenge Seed

Solution

```
// solution required
```

10. Test if a Value Falls within a Specific Range

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#).
.approximately .approximately(actual, expected, range, [message]) actual = expected +/- range Choose the minimum range (3rd parameter) to make the test always pass it should be less than 1

Instructions

Challenge Seed

Solution

```
// solution required
```

11. Test if a Value is an Array

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#).

Instructions

Challenge Seed

Solution

```
// solution required
```

12. Test if an Array Contains an Item

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#).

Instructions

Challenge Seed

Solution

```
// solution required
```

13. Test if a Value is a String

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). #isString asserts that the actual value is a string.

Instructions

Challenge Seed

Solution

```
// solution required
```

14. Test if a String Contains a Substring

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). #include (on #notInclude) works for strings too !! It asserts that the actual string contains the expected substring

Instructions

Challenge Seed

Solution

```
// solution required
```

15. Use Regular Expressions to Test a String

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). #match Asserts that the actual value matches the second argument regular expression.

Instructions

Challenge Seed

Solution

```
// solution required
```

16. Test if an Object has a Property

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). #property asserts that the actual object has a given property. Use #property or #notProperty where appropriate

Instructions

Challenge Seed

Solution

```
// solution required
```

17. Test if a Value is of a Specific Data Structure Type

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). #typeOf asserts that value's type is the given string, as determined by Object.prototype.toString. Use #typeOf or #notTypeOf where appropriate

Instructions

Challenge Seed

Solution

```
// solution required
```

18. Test if an Object is an Instance of a Constructor

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). #instanceOf asserts that an object is an instance of a constructor. Use #instanceOf or #notInstanceOf where appropriate

Instructions

Challenge Seed

Solution

```
// solution required
```

19. Run Functional Tests on API Endpoints using Chai-HTTP

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Replace assert.fail(). Test the status and the text.response. Make the test pass. Don't send a name in the query, the endpoint with responds with 'hello Guest'.

Instructions

Challenge Seed

Solution

```
// solution required
```

20. Run Functional Tests on API Endpoints using Chai-HTTP II

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Replace assert.fail(). Test the status and the text.response. Make the test pass. Send you name in the query appending ?name=, the endpoint with responds with 'hello'.

Instructions

Challenge Seed

Solution

```
// solution required
```

21. Run Functional Tests on an API Response using Chai-HTTP III - PUT method

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). In the next example we'll see how to send data in a request payload (body). We are going to test a PUT request. The '/travellers' endpoint accepts a JSON object taking the structure : {surname: [last name of a traveller of the past]} , The route responds with : {name: [first name], surname:[last name], dates: [birth - death years]} see the server code for more details. Send {surname: 'Colombo'}. Replace assert.fail() and make the test pass. Check for 1) status, 2) type, 3) body.name, 4) body.surname Follow the assertion order above, We rely on it.

Instructions

Challenge Seed

Solution

```
// solution required
```

22. Run Functional Tests on an API Response using Chai-HTTP IV - PUT method

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). This exercise is similar to the preceding. Look at it for the details. Send {surname: 'da Verrazzano'}. Replace assert.fail() and make the test pass. Check for 1) status, 2) type, 3) body.name, 4) body.surname Follow the assertion order above, We rely on it.

Instructions

Challenge Seed

Solution

```
// solution required
```

23. Run Functional Tests using a Headless Browser

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). In the next challenges we are going to simulate the human interaction with a page using a device called 'Headless Browser'. A headless browser is a web browser without a graphical user interface. These kind of tools are particularly useful for testing web pages as they are able to render and understand HTML, CSS, and JavaScript the same way a browser would. For these challenges we are using Zombie.js. It's a lightweight browser which is totally based on JS, without relying on additional binaries to be installed. This feature makes it usable in an environment such as Glitch. There are many other (more powerful) options.

Look at the examples in the code for the exercise directions. Follow the assertions order, We rely on it.

Instructions

Challenge Seed

Solution

```
// solution required
```

24. Run Functional Tests using a Headless Browser II

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). This exercise is similar to the preceding. Look at the code for directions. Follow the assertions order, We rely on it.

Instructions

Challenge Seed

Solution

```
// solution required
```

Advanced Node and Express

1. Set up a Template Engine

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). A template engine enables you to use static template files (such as those written in *Pug*) in your app. At runtime, the template engine replaces variables in a template file with actual values which can be supplied by your server, and transforms the template into a static HTML file that is then sent to the client. This approach makes it easier to design an HTML page and allows for displaying of variables on the page without needing to make an API call from the client. To set up *Pug* for use in your project, you will need to add it as a dependency first in your package.json. "pug": "[^]0.1.0" Now to tell Node/Express to use the templating engine you will have to tell your express app to set 'pug' as the 'view-engine'. app.set('view engine', 'pug') Lastly, you should change your response to the request for the index route to res.render with the path to the view views/pug/index.pug. If all went as planned, you should refresh your apps home page and see a small message saying you're successfully rendering the Pug from our Pug file! Submit your page when you think you've got it right.

Instructions

Challenge Seed

Solution

```
// solution required
```

2. Use a Template Engine's Powers

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). One of the greatest features of using a template engine is being able to pass variables from the server to the template file before rendering it to HTML. In your Pug file, you're about to use a variable by referencing the variable name as # {variable_name} inline with other text on an element or by using an equal sign on the element without a space such as p= variable_name which sets that p elements text to equal the variable. We strongly recommend looking at the syntax and structure of Pug [here](#) on their Githubs README. Pug is all about using whitespace and tabs to show nested elements and cutting down on the amount of code needed to make a beautiful site. Looking at our pug file 'index.pug' included in your project, we used the variables title and message To pass those alone from our server, you will need to add an object as a second argument to your res.render with the variables and their value. For example, pass this object along setting the variables for your index view: {title: 'Hello', message: 'Please login'} It should look like: res.render(process.cwd() + '/views/pug/index', {title: 'Hello', message: 'Please login'}); Now refresh your page and you should see those values rendered in your view in the correct spot as laid out in your index.pug file! Submit your page when you think you've got it right.

Instructions

Challenge Seed

Solution

```
// solution required
```

3. Set up Passport

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). It's time to set up *Passport* so we can finally start allowing a user to register or login to an account! In addition to Passport, we will use Express-session to handle sessions. Using this middleware saves the session id as a cookie in the client and allows us to access the session data using that id on the server. This way we keep personal account information out of the cookie used by the client to verify to our server they are authenticated and just keep the *key* to access the data stored on the server. To set up Passport for use in your project, you will need to add it as a dependency first in your package.json. "passport": "^0.3.2" In addition, add Express-session as a dependency now as well. Express-session has a ton of advanced features you can use but for now we're just going to use the basics! "express-session": "^1.15.0" You will need to set up the session settings now and initialize Passport. Be sure to first create the variables 'session' and 'passport' to require 'express-session' and 'passport' respectively. To set up your express app to use use the session we'll define just a few basic options. Be sure to add 'SESSION_SECRET' to your .env file and give it a random value. This is used to compute the hash used to encrypt your cookie!

```
app.use(session({
  secret: process.env.SESSION_SECRET,
  resave: true,
  saveUninitialized: true,
}));
```

As well you can go ahead and tell your express app to use 'passport.initialize()' and 'passport.session()'. (For example, app.use(passport.initialize());) Submit your page when you think you've got it right. If you're running into errors, you can check out the project completed up to this point [here](#).

Instructions

Challenge Seed

Solution

```
// solution required
```

4. Serialization of a User Object

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Serialization and deserialization are important concepts in regards to authentication. To serialize an object means to convert its contents into a small *key* essentially that can then be deserialized into the original object. This is what allows us to know who communicated with the server without having to send the authentication data like username and password at each request for a new page. To set this up properly, we need to have a serialize function and a deserialize function. In passport we create these with `passport.serializeUser(OURFUNCTION)` and `passport.deserializeUser(OURFUNCTION)`. The `serializeUser` is called with 2 arguments, the full user object and a callback used by passport. Returned in the callback should be a unique key to identify that user- the easiest one to use being the users _id in the object as it should be unique as it generated by MongoDB. Similarly `deserializeUser` is called with that key and a callback function for passport as well, but this time we have to take that key and return the users full object to the callback. To make a query search for a Mongo _id you will have to create `const ObjectId = require('mongodb').ObjectId;`, and then to use it you call `new ObjectId(THE_ID)`. Be sure to add MongoDB as a dependency. You can see this in the examples below:

```
passport.serializeUser((user, done) => {
  done(null, user._id);
});
```

```

passport.deserializeUser((id, done) => {
  db.collection('users').findOne(
    {_id: new ObjectID(id)},
    (err, doc) => {
      done(null, doc);
    }
  );
});

```

NOTE: This deserializeUser will throw an error until we set up the DB in the next step so comment out the whole block and just call `done(null, null)` in the function `deserializeUser`. Submit your page when you think you've got it right.

Instructions

Challenge Seed

Solution

```
// solution required
```

5. Implement the Serialization of a Passport User

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Right now we're not loading an actual user object since we haven't set up our database. This can be done many different ways, but for our project we will connect to the database once when we start the server and keep a persistent connection for the full life-cycle of the app. To do this, add MongoDB as a dependency and require it in your server. (`const mongo = require('mongodb').MongoClient;`) Now we want to connect to our database then start listening for requests. The purpose of this is to not allow requests before our database is connected or if there is a database error. To accomplish this you will want to encompass your serialization and your app listener in the following:

```

mongo.connect(process.env.DATABASE, (err, db) => {
  if(err) {
    console.log('Database error: ' + err);
  } else {
    console.log('Successful database connection');

    //serialization and app.listen
  }
});

```

You can now uncomment the block in `deserializeUser` and remove your `done(null, null)`. Be sure to set `DATABASE` in your `.env` file to your database's connection string (for example: `DATABASE=mongodb://admin:pass@mLab.com:12345/my-project`). You can set up a free database on [mLab](#). Congratulations- you've finished setting up serialization! Submit your page when you think you've got it right. If you're running into errors, you can check out the project completed up to this point [here](#).

Instructions

Challenge Seed

Solution

```
// solution required
```

6. Authentication Strategies

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). A strategy is a way of authenticating a user. You can use a strategy for allowing users to authenticate based on locally saved information (if you have them register first) or from a variety of providers such as Google or GitHub. For this project we will set up a local strategy. To see a list of the 100's of strategies, visit Passports site [here](#). Add `passport-local` as a dependency and add it to your server as follows: `const LocalStrategy = require('passport-local');` Now you will have to tell passport to `use` an instantiated LocalStartegy object with a few settings defined. Make sure this as well as everything from this point on is encapsulated in the database connection since it relies on it!

```
passport.use(new LocalStrategy(
  function(username, password, done) {
    db.collection('users').findOne({ username: username }, function (err, user) {
      console.log('User ' + username + ' attempted to log in.');
      if (err) { return done(err); }
      if (!user) { return done(null, false); }
      if (password !== user.password) { return done(null, false); }
      return done(null, user);
    });
  }
));
```

This is defining the process to take when we try to authenticate someone locally. First it tries to find a user in our database with the username entered, then it checks for the password to match, then finally if no errors have popped up that we checked for, like an incorrect password, the users object is returned and they are authenticated. Many strategies are set up using different settings, general it is easy to set it up based on the README in that strategies repository though. A good example of this is the GitHub strategy where we don't need to worry about a username or password because the user will be sent to GitHub's auth page to authenticate and as long as they are logged in and agree then GitHub returns their profile for us to use. In the next step we will set up how to actually call the authentication strategy to validate a user based on form data! Submit your page when you think you've got it right up to this point.

Instructions

Challenge Seed

Solution

```
// solution required
```

7. How to Use Passport Strategies

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). In the `index.pug` file supplied there is actually a login form. It has previously been hidden because of the inline JavaScript `if showLogin` with the form indented after it. Before `showLogin` as a variable was never defined, it never rendered the code block containing the form. Go ahead and on the `res.render` for that page add a new variable to the object `showLogin: true`. When you refresh your page, you should then see the form! This form is set up to **POST** on `/login` so this is where we should set up to accept the POST and authenticate the user. For this challenge you should add the route `/login` to accept a POST request. To authenticate on this route you need to add a middleware to do so before

then sending a response. This is done by just passing another argument with the middleware before your `function(req,res)` with your response! The middleware to use is `passport.authenticate('local')`. `passport.authenticate` can also take some options as an argument such as: `{ failureRedirect: '/' }` which is incredibly useful so be sure to add that in as well. As a response after using the middleware (which will only be called if the authentication middleware passes) should be to redirect the user to `/profile` and that route should render the view `'profile.pug'`. If the authentication was successful, the user object will be saved in `req.user`. Now at this point if you enter a username and password in the form, it should redirect to the home page / and in the console of your server should be 'User {USERNAME} attempted to log in.' since we currently cannot login a user who isn't registered. Submit your page when you think you've got it right. If you're running into errors, you can check out the project completed up to this point [here](#).

Instructions

Challenge Seed

Solution

```
// solution required
```

8. Create New Middleware

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). As in, any user can just go to `/profile` whether they authenticated or not by typing in the url. We want to prevent this by checking if the user is authenticated first before rendering the profile page. This is the perfect example of when to create a middleware. The challenge here is creating the middleware function `ensureAuthenticated(req, res, next)`, which will check if a user is authenticated by calling `passports isAuthenticated` on the `request` which in turn checks for `req.user` to be defined. If it is then `next()` should be called, otherwise we can just respond to the request with a redirect to our homepage to login. An implementation of this middleware is:

```
function ensureAuthenticated(req, res, next) {
  if (req.isAuthenticated()) {
    return next();
  }
  res.redirect('/');
};
```

Now add `ensureAuthenticated` as a middleware to the request for the profile page before the argument to the get request containing the function that renders the page.

```
app.route('/profile')
  .get(ensureAuthenticated, (req,res) => {
    res.render(process.cwd() + '/views/pug/profile');
  });
};
```

Submit your page when you think you've got it right.

Instructions

Challenge Seed

Solution

```
// solution required
```

9. How to Put a Profile Together

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Now that we can ensure the user accessing the `/profile` is authenticated, we can use the information contained in `'req.user'` on our page! Go ahead and pass the object containing the variable `username` equaling `'req.user.username'` into the render method of the profile view. Then go to your `'profile.pug'` view and add the line `h2.center#welcome Welcome, # {username}!` creating the `h2` element with the class `'center'` and id `'welcome'` containing the text `'Welcome,'` and the `username!` Also in the profile, add a link to `/logout`. That route will host the logic to unauthenticate a user.

`a(href='/logout') Logout` Submit your page when you think you've got it right.

Instructions

Challenge Seed

Solution

```
// solution required
```

10. Logging a User Out

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Creating the logout logic is easy. The route should just unauthenticate the user and redirect to the home page instead of rendering any view. In passport, unauthenticating a user is as easy as just calling `req.logout();` before redirecting.

```
app.route('/logout')
  .get((req, res) => {
    req.logout();
    res.redirect('/');
  });

```

You may have noticed that we're not handling missing pages (404), the common way to handle this in Node is with the following middleware. Go ahead and add this in after all your other routes:

```
app.use((req, res, next) => {
  res.status(404)
    .type('text')
    .send('Not Found');
});

```

Submit your page when you think you've got it right.

Instructions

Challenge Seed

Solution

```
// solution required
```

11. Registration of New Users

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Now we need to allow a new user on our site to register an account. On the res.render for the home page add a new variable to the object passed along- showRegistration: true . When you refresh your page, you should then see the registration form that was already created in your index.pug file! This form is set up to **POST** on `/register` so this is where we should set up to accept the POST and create the user object in the database. The logic of the registration route should be as follows: Register the new user > Authenticate the new user > Redirect to /profile The logic of step 1, registering the new user, should be as follows: Query database with a findOne command > if user is returned then it exists and redirect back to home OR if user is undefined and no error occurs then 'insertOne' into the database with the username and password and as long as no errors occur then call *next* to go to step 2, authenticating the new user, which we've already written the logic for in our POST /login route.

```
app.route('/register')
  .post((req, res, next) => {
    db.collection('users').findOne({ username: req.body.username }, function (err, user) {
      if(err) {
        next(err);
      } else if (user) {
        res.redirect('/');
      } else {
        db.collection('users').insertOne(
          {username: req.body.username,
           password: req.body.password},
          (err, doc) => {
            if(err) {
              res.redirect('/');
            } else {
              next(null, user);
            }
          }
        );
      }
    });
  }),
  passport.authenticate('local', { failureRedirect: '/' }),
  (req, res, next) => {
    res.redirect('/profile');
  }
);
```

Submit your page when you think you've got it right. If you're running into errors, you can check out the project completed up to this point [here](#).

Instructions

Challenge Seed

Solution

```
// solution required
```

12. Hashing Your Passwords

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Going back to the information security section you may remember that storing plaintext passwords is *never* okay. Now it is time to implement BCrypt to solve this issue.

Add BCrypt as a dependency and require it in your server. You will need to handle hashing in 2 key areas: where you handle registering/saving a new account and when you check to see that a password is correct on login. Currently on our registration route, you insert a user's password into the database like the following: `password: req.body.password`. An easy way to implement saving a hash instead is to add the following before your database logic `var hash = bcrypt.hashSync(req.body.password, 12);` and replacing the `req.body.password` in the database saving with just `password: hash`. Finally on our authentication strategy we check for the following in our code before completing the process: `if (password !== user.password) { return done(null, false); }`. After making the previous changes, now `user.password` is a hash. Before making a change to the existing code, notice how the statement is checking if the password is NOT equal then return non-authenticated. With this in mind your code could look as follows to properly check the password entered against the hash: `if (!bcrypt.compareSync(password, user.password)) { return done(null, false); }` That is all it takes to implement one of the most important security features when you have to store passwords! Submit your page when you think you've got it right.

Instructions

Challenge Seed

Solution

```
// solution required
```

13. Clean Up Your Project with Modules

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Right now everything you have is in your `server.js` file. This can lead to hard to manage code that isn't very expandable. Create 2 new files: `Routes.js` and `Auth.js`. Both should start with the following code:

```
module.exports = function (app, db) {
```

Now in the top of your `server.js` file, require these files like such: `const routes = require('./routes.js');` Right after you establish a successful connect with the database instantiate each of them like such: `routes(app, db)`. Finally, take all of the routes in your `server.js` and paste them into your new files and remove them from your `server.js` file. Also take the `ensureAuthenticated` since we created that middleware function for routing specifically. You will have to now correctly add the dependencies in that are used, such as `const passport = require('passport');`, at the very top above the `export` line in your `routes.js` file. Keep adding them until no more errors exist, and your `server.js` file no longer has any routing! Now do the same thing in your `auth.js` file with all of the things related to authentication such as the serialization and the setting up of the local strategy and erase them from your `server.js` file. Be sure to add the dependencies in and call `auth(app, db)` in the `server.js` in the same spot. Be sure to have `auth(app, db)` before `routes(app, db)` since our registration route depends on `passport` being initiated! Congratulations- you're at the end of this section of Advanced Node and Express and have some beautiful code to show for it! Submit your page when you think you've got it right. If you're running into errors, you can check out an example of the completed project [here](#).

Instructions

Challenge Seed

Solution

```
// solution required
```

14. Implementation of Social Authentication

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). The basic path this kind of authentication will follow in your app is:

1. User clicks a button or link sending them to our route to authenticate using a specific strategy (EG. GitHub)
2. Your route calls `passport.authenticate('github')` which redirects them to GitHub.
3. The page the user lands on, on GitHub, allows them to login if they aren't already. It then asks them to approve access to their profile from our app.
4. The user is then returned to our app at a specific callback url with their profile if they are approved.
5. They are now authenticated and your app should check if it is a returning profile, or save it in your database if it is not.

Strategies with OAuth require you to have at least a *Client ID* and a *Client Secret* which is a way for them to verify who the authentication request is coming from and if it is valid. These are obtained from the site you are trying to implement authentication with, such as GitHub, and are unique to your app- **THEY ARE NOT TO BE SHARED** and should never be uploaded to a public repository or written directly in your code. A common practice is to put them in your `.env` file and reference them like: `process.env.GITHUB_CLIENT_ID`. For this challenge we're going to use the GitHub strategy. Obtaining your *Client ID* and *Secret* from GitHub is done in your account profile settings under 'developer settings', then '[OAuth applications](#)'. Click 'Register a new application', name your app, paste in the url to your glitch homepage (**Not the project code's url**), and lastly for the callback url, paste in the same url as the homepage but with '/auth/github/callback' added on. This is where users will be redirected to for us to handle after authenticating on GitHub. Save the returned information as '`GITHUB_CLIENT_ID`' and '`GITHUB_CLIENT_SECRET`' in your `.env` file. On your remixed project, create 2 routes accepting GET requests: `/auth/github` and `/auth/github/callback`. The first should only call `passport` to `authenticate 'github'` and the second should call `passport` to `authenticate 'github'` with a failure redirect to '/' and then if that is successful redirect to `'/profile'` (similar to our last project). An example of how `'/auth/github/callback'` should look is similar to how we handled a normal login in our last project:

```
app.route('/login')
  .post(passport.authenticate('local', { failureRedirect: '/' }), (req, res) => {
    res.redirect('/profile');
  });

```

Submit your page when you think you've got it right. If you're running into errors, you can check out the project up to this point [here](#).

Instructions

Challenge Seed

Solution

```
// solution required
```

15. Implementation of Social Authentication II

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). The last part of setting up your GitHub authentication is to create the strategy itself. For this, you will need to add the dependency of '`passport-github`' to your project and require it as `GithubStrategy` like `const GitHubStrategy = require('passport-`

`github').Strategy;`. To set up the GitHub strategy, you have to tell `passport` to use an instantiated `GitHubStrategy`, which accepts 2 arguments: An object (containing `clientID`, `clientSecret`, and `callbackURL`) and a function to be called when a user is successfully authenticated which we will determine if the user is new and what fields to save initially in the user's database object. This is common across many strategies but some may require more information as outlined in that specific strategy's `github README`; for example, Google requires a scope as well which determines what kind of information your request is asking returned and asks the user to approve such access. The current strategy we are implementing has its usage outlined here, but we're going through it all right here on freeCodeCamp! Here's how your new strategy should look at this point:

```
passport.use(new GitHubStrategy({
  clientID: process.env.GITHUB_CLIENT_ID,
  clientSecret: process.env.GITHUB_CLIENT_SECRET,
  callbackURL: /*INSERT CALLBACK URL ENTERED INTO GITHUB HERE*/
},
function(accessToken, refreshToken, profile, cb) {
  console.log(profile);
  //Database logic here with callback containing our user object
})
);
```

Your authentication won't be successful yet, and actually throw an error, without the database logic and callback, but it should log to your console your GitHub profile if you try it! Submit your page when you think you've got it right.

Instructions

Challenge Seed

Solution

```
// solution required
```

16. Implementation of Social Authentication III

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). The final part of the strategy is handling the profile returned from GitHub. We need to load the user's database object if it exists, or create one if it doesn't, and populate the fields from the profile, then return the user's object. GitHub supplies us a unique id within each profile which we can use to search with to serialize the user with (already implemented). Below is an example implementation you can use in your project- it goes within the function that is the second argument for the new strategy, right below the `console.log(profile);` currently is:

```
db.collection('socialusers').findAndModify(
  {id: profile.id},
  {},
  {$setOnInsert:{ 
    id: profile.id,
    name: profile.displayName || 'John Doe',
    photo: profile.photos[0].value || '',
    email: profile.emails[0].value || 'No public email',
    created_on: new Date(),
    provider: profile.provider || '' 
  },$set:{ 
    last_login: new Date()
  },$inc:{ 
    login_count: 1 
  },
  {upsert:true, new: true},
  (err, doc) => {
    return cb(null, doc.value);
  }
);
```

With a `findAndModify`, it allows you to search for an object and update it, as well as insert the object if it doesn't exist and receive the new object back each time in our callback function. In this example, we always set the `last_login` as now, we always increment the `login_count` by 1, and only when we insert a new object(new user) do we populate the majority of the fields. Something to notice also is the use of default values. Sometimes a profile returned won't have all the information filled out or it will have been chosen by the user to remain private; so in this case we have to handle it to prevent an error. You should be able to login to your app now- try it! Submit your page when you think you've got it right. If you're running into errors, you can check out an example of this mini-project's finished code [here](#).

Instructions

Challenge Seed

Solution

```
// solution required
```

17. Set up the Environment

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Add `Socket.IO` as a dependency and require/instantiate it in your server defined as '`io`' with the `http` server as an argument. `const io = require('socket.io')(http);` The first thing needing to be handled is listening for a new connection from the client. The `on` keyword does just that- listen for a specific event. It requires 2 arguments: a string containing the title of the event that's emitted, and a function with which the data is passed through. In the case of our connection listener, we use `socket` to define the data in the second argument. A socket is an individual client who is connected. For listening for connections on our server, add the following between the comments in your project:

```
io.on('connection', socket => {
  console.log('A user has connected');
});
```

Now for the client to connect, you just need to add the following to your `client.js` which is loaded by the page after you've authenticated:

```
/*global io*/
var socket = io();
```

*The comment suppresses the error you would normally see since '`io`' is not defined in the file. We've already added a reliable CDN to the `Socket.IO` library on the page in `chat.pug`. Now try loading up your app and authenticate and you should see in your server `console` 'A user has connected'! **Note***

`io()` works only when connecting to a socket hosted on the same url/server. For connecting to an external socket hosted elsewhere, you would use `io.connect('URL');`. Submit your page when you think you've got it right.

Instructions

Challenge Seed

Solution

```
// solution required
```

18. Communicate by Emitting

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Emit is the most common way of communicating you will use. When you emit something from the server to 'io', you send an event's name and data to all the connected sockets. A good example of this concept would be emitting the current count of connected users each time a new user connects!

Start by adding a variable to keep track of the users just before where you are currently listening for connections. var currentUsers = 0; Now when someone connects you should increment the count before emitting the count so you will want to add the incrementer within the connection listener. ++currentUsers; Finally after incrementing the count, you should emit the event(still within the connection listener). The event should be named 'user count' and the data should just be the 'currentUsers'. io.emit('user count', currentUsers);

Now you can implement a way for your client to listen for this event! Similarly to listening for a connection on the server you will use the on keyword.

```
socket.on('user count', function(data){
  console.log(data);
});
```

Now try loading up your app and authenticate and you should see in your client console '1' representing the current user count! Try loading more clients up and authenticating to see the number go up. Submit your page when you think you've got it right.

Instructions

Challenge Seed

Solution

```
// solution required
```

19. Handle a Disconnect

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). You may notice that up to now you have only been increasing the user count. Handling a user disconnecting is just as easy as handling the initial connect except the difference is you have to listen for it on each socket versus on the whole server.

*To do this, add in to your existing connect listener a listener that listens for 'disconnect' on the socket with no data passed through. You can test this functionality by just logging to the console a user has disconnected. socket.on('disconnect', () => { /*anything you want to do on disconnect*/ }); To make sure clients continuously have the updated count of current users, you should decrease the currentUsers by 1 when the disconnect happens then emit the 'user count' event with the updated count! Note*

Just like 'disconnect', all other events that a socket can emit to the server should be handled within the connecting listener where we have 'socket' defined. Submit your page when you think you've got it right.

Instructions

Challenge Seed

Solution

```
// solution required
```

20. Authentication with Socket.IO

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Currently, you cannot determine who is connected to your web socket. While 'req.user' contains the user object, that's only when your user interacts with the web server and with web sockets you have no req (request) and therefore no user data. One way to solve the problem of knowing who is connected to your web socket is by parsing and decoding the cookie that contains the passport session then deserializing it to obtain the user object. Luckily, there is a package on NPM just for this that turns a once complex task into something simple!

Add 'passport.socketio' as a dependency and require it as 'passportSocketIo'. Now we just have to tell Socket.IO to use it and set the options. Be sure this is added before the existing socket code and not in the existing connection listener. For your server it should look as follows:

```
io.use(passportSocketIo.authorize({
  cookieParser: cookieParser,
  key:           'express.sid',
  secret:        process.env.SESSION_SECRET,
  store:         sessionStore
});
```

You can also optionally pass 'success' and 'fail' with a function that will be called after the authentication process completes when a client tries to connect. The user object is now accessible on your socket object as `socket.request.user`. For example, now you can add the following: `console.log('user ' + socket.request.user.name + ' connected')`; and it will log to the server console who has connected! Submit your page when you think you've got it right. If you're running into errors, you can check out the project up to this point [here](#).

Instructions

Challenge Seed

Solution

```
// solution required
```

21. Announce New Users

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). Many chat rooms are able to announce when a user connects or disconnects and then display that to all of the connected users in the chat. Seeing as though you already are emitting an event on connect and disconnect, you will just have to modify this event to support such feature. The most logical way of doing so is sending 3 pieces of data with the event: name of the user connected/disconnected, the current user count, and if that name connected or disconnected.

Change the event name to 'user' and as the data pass an object along containing fields 'name', 'currentUsers', and boolean 'connected' (to be true if connection, or false for disconnection of the user sent). Be sure to make the change to both points we had the 'user count' event and set the disconnect one to sent false for field 'connected' instead of true like the event emitted on connect. `io.emit('user', {name: socket.request.user.name, currentUsers, connected: true});` Now your client will have all the necessary information to correctly display the current user count and announce when a user connects or disconnects! To handle this event on the client side we should listen for 'user' and then update the current user count by using jQuery to change the text of `#num-users` to '{NUMBER} users online', as well as append a `` to the unordered list with id 'messages' with '{NAME} has {joined/left} the chat.'. An implementation of this could look like the following:

```
socket.on('user', function(data){
  $('#num-users').text(data.currentUsers+' users online');
  var message = data.name;
  if(data.connected) {
    message += ' has joined the chat.';
  } else {
    message += ' has left the chat.';
  }
  $('#messages').append($('- ').html('<b>' + message + '</b>'));
});

```

Submit your page when you think you've got it right.

Instructions

Challenge Seed

Solution

```
// solution required
```

22. Send and Display Chat Messages

Description

As a reminder, this project is being built upon the following starter project on [Glitch](#), or cloned from [GitHub](#). It's time you start allowing clients to send a chat message to the server to emit to all the clients! Already in your `client.js` file you should see there is already a block of code handling when the message form is submitted! (`$('form').submit(function(){ /*logic*/ })` ;)

Within the code you're handling the form submit you should emit an event after you define 'messageToSend' but before you clear the text box `#m`. The event should be named 'chat message' and the data should just be 'messageToSend'.

`socket.emit('chat message', messageToSend);` Now on your server you should be listening to the socket for the event 'chat message' with the data being named 'message'. Once the event is received it should then emit the event 'chat message' to all sockets `io.emit` with the data being an object containing 'name' and 'message'. On your client now again, you should now listen for event 'chat message' and when received, append a list item to `#messages` with the name a colon and the message! At this point the chat should be fully functional and sending messages across all clients! Submit your page when you think you've got it right. If you're running into errors, you can check out the project up to this point [here for the server](#) and [here for the client](#).

Instructions

Challenge Seed

Solution

```
// solution required
```

Information Security and Quality Assurance Projects

1. Metric-Imperial Converter

Description

Build a full stack JavaScript app that is functionally similar to this: <https://hard-twilight.glitch.me/>. Working on this project will involve you writing your code on Glitch on our starter project. After completing this project you can copy your public glitch url (to the homepage of your app) into this screen to test it! Optionally you may choose to write your project on another platform but it must be publicly visible for our testing. Start this project on Glitch using [this link](#) or clone [this repository](#) on GitHub! If you use Glitch, remember to save the link to your project somewhere safe!

Instructions

Challenge Seed

Solution

```
// solution required
```

2. Issue Tracker

Description

Build a full stack JavaScript app that is functionally similar to this: <https://protective-garage.glitch.me/>. Working on this project will involve you writing your code on Glitch on our starter project. After completing this project you can copy your public glitch url (to the homepage of your app) into this screen to test it! Optionally you may choose to write your project on another platform but it must be publicly visible for our testing. Start this project on Glitch using [this link](#) or clone [this repository](#) on GitHub! If you use Glitch, remember to save the link to your project somewhere safe!

Instructions

Challenge Seed

Solution

```
// solution required
```

3. Personal Library

Description

Build a full stack JavaScript app that is functionally similar to this: <https://spark-cathedral.glitch.me/>. Working on this project will involve you writing your code on Glitch on our starter project. After completing this project you can copy your public

glitch url (to the homepage of your app) into this screen to test it! Optionally you may choose to write your project on another platform but must be publicly visible for our testing. Start this project on Glitch using [this link](#) or clone [this repository](#) on GitHub! If you use Glitch, remember to save the link to your project somewhere safe!

Instructions

Challenge Seed

Solution

```
// solution required
```

4. Stock Price Checker

Description

Build a full stack JavaScript app that is functionally similar to this: <https://giant-chronometer.glitch.me/>. Working on this project will involve you writing your code on Glitch on our starter project. After completing this project you can copy your public glitch url (to the homepage of your app) into this screen to test it! Optionally you may choose to write your project on another platform but must be publicly visible for our testing. Start this project on Glitch using [this link](#) or clone [this repository](#) on GitHub! If you use Glitch, remember to save the link to your project somewhere safe!

Instructions

Challenge Seed

Solution

```
// solution required
```

5. Anonymous Message Board

Description

Build a full stack JavaScript app that is functionally similar to this: <https://horn-celery.glitch.me/>. Working on this project will involve you writing your code on Glitch on our starter project. After completing this project you can copy your public glitch url (to the homepage of your app) into this screen to test it! Optionally you may choose to write your project on another platform but it must be publicly visible for our testing. Start this project on Glitch using [this link](#) or clone [this repository](#) on GitHub! If you use Glitch, remember to save the link to your project somewhere safe!

Instructions

Challenge Seed

Solution

```
// solution required
```

