Checkpoint 9

Deeper with CSS

Now that you've got some experience with CSS, it's time to deepen your knowledge. Specifically, you're ready to learn about classes and the box model.

The content in this checkpoint may be a bit more complicated, but as a web developer, you'll regularly be challenging yourself as you learn increasingly sophisticated skills. With this more comprehensive understanding of CSS, you'll be able to build web pages that are even more complex, vibrant, and visually compelling.

Outline

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

- Describe and apply the cascading nature of CSS
- Use classes to dynamically and specifically add styles
- Structure content with the box model

The cascade in CSS

You know that CSS stands for *Cascading Style Sheets*. But you haven't yet explored what that really means. To better understand the

possibilities presented by CSS, you first need to know where you can put it. CSS code can be placed in three locations, which you'll learn about in more detail below:

- An *external* style sheet
- An *internal* style sheet
- *Inline* styles, which sit alongside HTML code

External

To properly utilize CSS to its full power, you will typically apply styles to your HTML code by linking to one or more *external style sheets* in your web page. An external style sheet contains style rules that are applied to every HTML page that links to it. These links are created automatically for you in Repl.it, but you'll need to know how to do this on your own once you start using a text editor. Take a look at the below.


```
<head>
  <link rel="stylesheet" type="text/css" href="style.
</head>
```

It's important to note that for most of your web projects, you will use an external style sheet that contains all of the CSS rules that apply to that project. In other words, all the CSS code that you want to apply will be placed inside the single `style.css` file.

The other two methods discussed below are important, but you're discouraged from using them unless you have a specific, comp g reason to do so.

Internal

The second method of applying CSS styles is through an *internal style sheet*. This approach allows you to write CSS rules within individual web pages, but those rules can only be referenced by that individual HTML page. This can make it harder to update and apply your CSS styles to your web pages. The code below shows how internal CSS is structured:

```
<head>
  <style>
    /* Internal CSS Rules go here. */
  </style>
</head>
```

Inline

Outline

The third method, *inline styles*, allows you to write CSS rules on specific HTML elements. You've seen this approach a bit in previous checkpoints, but it's actually quite limiting in real-world programming work. Because the CSS code is written alongside the HTML element where it applies, the rule works *only* for that single HTML element. Check out the structure of this technique below:

```
<body>
  <p style="color: red; font-size: 24px;">
    Example of an inline style.
  </p>
</body>
```

How does CSS cascade?

Now, take a moment to focus on the keyword in CSS: *cascade*. What does this really mean? The concept of *cascading* helps determine how rules will be applied based on when they appear in the code. It helps address the issue that can occur when the same property, but with a different value, is added to a project. In this case, which style will actually be applied? Fortunately, the cascade can help.

It works by giving more *importance* (described in more detail below) to the rules that are closer in proximity to the actual content that is being styled. In other words, if a style rule is written quite close to the HTML code where it applies, it is considered more important than a rule that is "farther away" from that HTML code.

Consider these examples. A CSS rule at the bottom of an external style sheet has more importance than one at the top of that same style sheet. A rule in an internal style sheet has more importance than any rule in an external style sheet. And an inline style has more importance than a rule in an internal style sheet.

Are you beginning to see the hierarchy? Here it is laid out: `External > Internal > Inline`. An inline style rule is the most specific and closest to the code, and will therefore override a rule from an internal style sheet. And an internal style sheet rule is more specific and closer than an external rule, so it will override a rule from an external style sheet.

For obvious reasons, the styles on smaller websites are far easier to maintain. But in large, complex websites—especially those with many contributors—you often end up with multiple style rules coming from multiple places, all targeting a specific element. If there are conflicts for

a given property, the browser will choose the rule with *higher specificity*, following the cascade above.

At some point in the future, you'll find yourself debugging a style setting, certain that a rule you wrote should be causing the style of an element to change. But if you find that the change isn't happening (or another change is happening instead), it's often a sign that a higher priority rule is overriding the one you're working with at the moment.

Here is a code sample that you can play around with to learn more about CSS specificity. Try removing the inline style and then the internal style, making sure to run the code after each change. What happens?

CSS Cascade Code Sample

The `!important` option

Take a moment to revisit the word *important* from the definition of cascade. Even with the hierarchy outlined above, there's a way to circumvent it if absolutely necessary. CSS allows you to supply the keyword `!important` in order to make a rule that overrides others. By inserting `!important`, you're telling the code that this rule, which might otherwise be lower priority in the cascade, should override other rules. Here's an example:

```
p {
  color: red !important;
}
```

A quick disclaimer: You should know about `!important` and how to use it, but try to avoid using it in your CSS. There are, of course, rare

occasions in which it's the right move. But typically, if you have to use `!important`, it's a sign that there are problems with the application of your style rules. For instance, you may just need to use a more specific selector!

Demo: Using `!important`

The Repl.it below contains a code sample that showcases the use of `!important`. Play around with it to see how it works.

Outline

Outline

CSS classes

So far in your CSS work, you've styled all HTML headings and paragraphs using *element selectors* like these ones:

```
h1 {
  color: #242424;
  font-family: "Times New Roman", Times, serif;
}

p {
  color: #757575;
  font-family: Arial, Helvetica, sans-serif;
}
```

But what if you want to style certain paragraphs and headings in different ways? In this case, you'll use a CSS class. *Classes* allow you to apply CSS properties to any HTML element and as many times as needed throughout a web page. They are very useful, offering you a convenient method of diversifying your styles.

Demo: Classes

Outline

Take a moment to review the code sample in the Repl.it below. Then, try to answer these questions without reading ahead:

- In the CSS panel, how is a class *written*?
- In the HTML panel, how is a class *assigned* to an HTML element?

Outline

Class syntax

Reviewing the code sample above, you probably noticed a few things. In the CSS in the `style.css` file, a class is written much like other CSS rules are written, but with a key difference: the class is identified by

a period, `.`. That period is very important; every CSS class *requires that period*. Here it is in action: `.align-right`.

Unlike your work with HTML elements such as `<p>`, `<h1>`, or `<ul>`, in which you had to assign CSS to style a specific HTML element name, the class names can be anything you want them to be. However, it helps to give each class a name that is descriptive and informative, such as `.large-text`. You don't want a class name to be *so* specific that it wouldn't get reused, such as `.font-size-72-pixels`. Once you have your descriptive class name, you can add an attribute to the element.

```
<p class="large-text">This text is large!</p>
```

Outline

It's important to notice that there is no period `.` written in the `index.html` file. The `.` only shows up in the CSS, not in the HTML.

Drill: Class practice

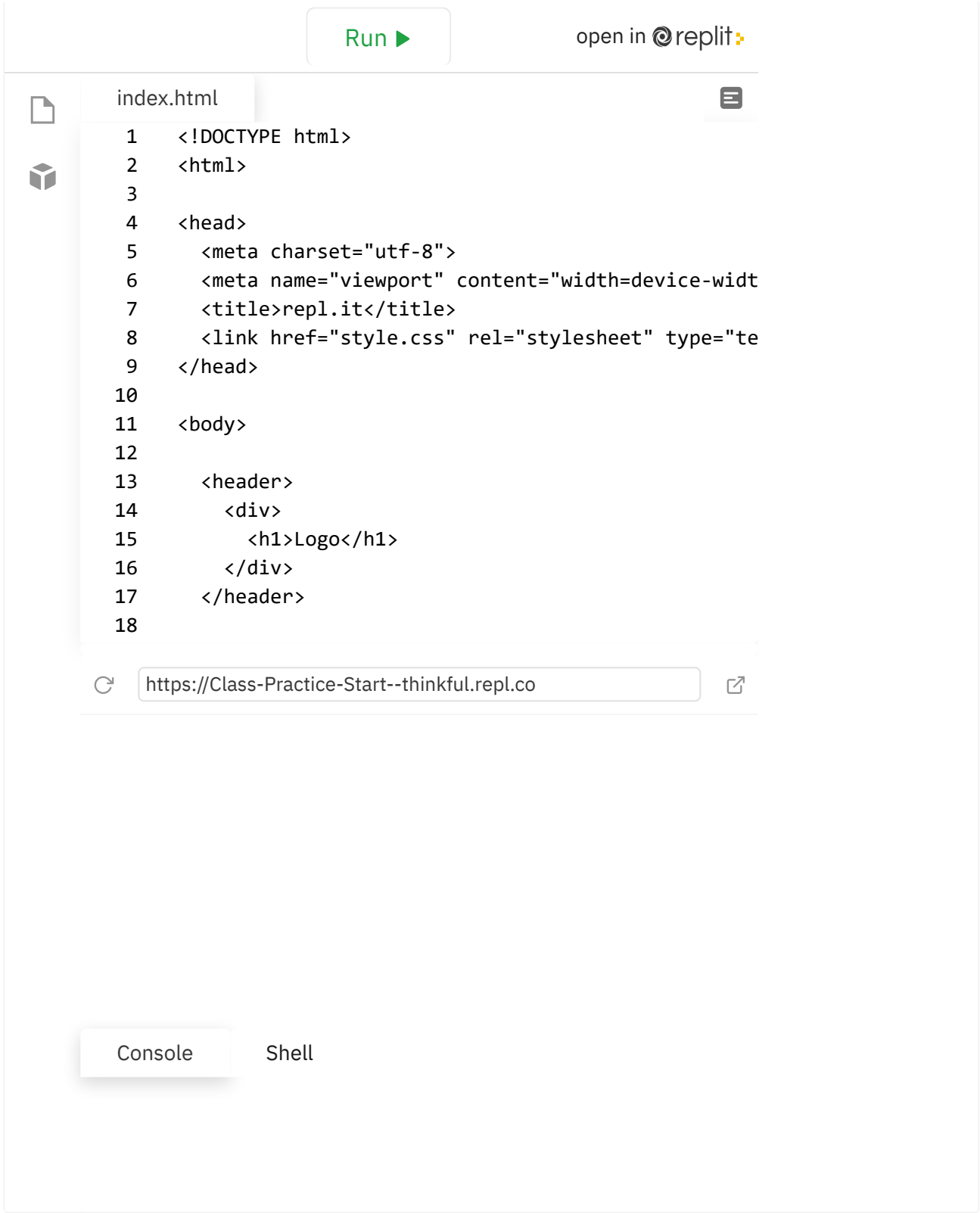
You're ready for a drill! In the Repl.it below, modify the code sample to get the result to present like the image below.

Follow these tasks to complete this drill.

1. **Center the text**: Create a class with a relevant name, like `text-center`, to center certain text. Assign the class to both the logo and the copyright text.

2. **Create container background colors**: Create two classes: one that gives a container a dark background with light text, and one that gives a container a light background with dark text. Assign the dark background class to the `header` and `footer` of the code sample, and assign the light background class to the `<main>` HTML element.

3. **Add a wrapper**: The *wrapper* is a common class that is often used by developers in web page layouts. The wrapper wraps around the content within primary containers to center that content within the page. In this drill, create a class with the name `.wrapper`. This class should have a set width of 300 to 400 pixels and the rule `margin: auto;`. (Don't worry, you'll learn what this means during the box model discussion). Assign this class to each of the three `<div>` containers inside of `<header>`, `<main>`, and `<footer>`.



Outline

When you finish, compare your code to this completed sample. Don't look at the code unless you are done or you're totally stuck.

Class Practice: Complete

Multiple classes

You can add as many CSS classes to an HTML element as you'd like. This allows you to build classes that have multiple utilities and are not overly specific. For example, take a look at the following two classes:

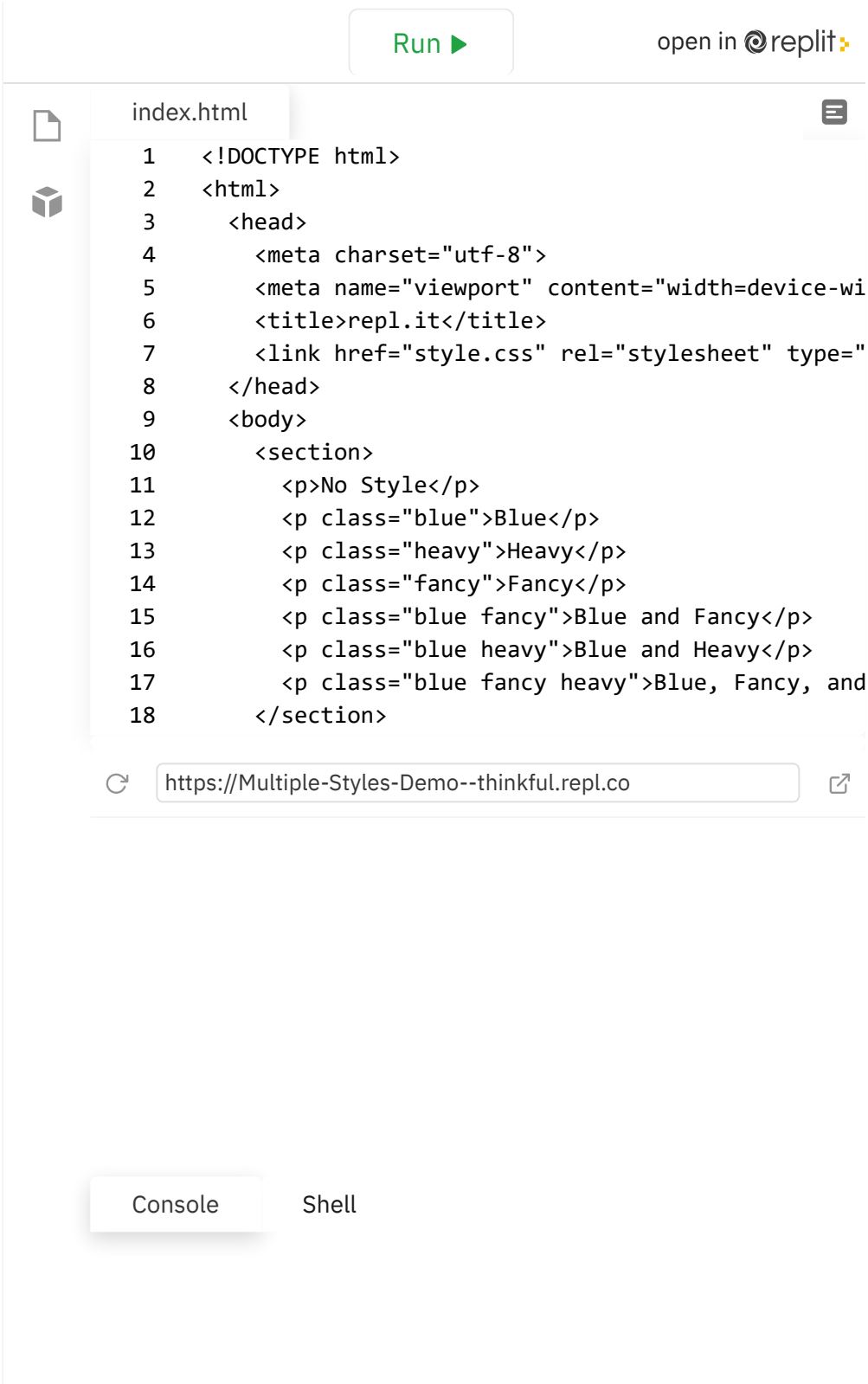
```
.warning {
  color: red;
}

.large {
  font-size: 35px;
}
```

In this case, you could apply each class individually to give a particular element one style, such as just red text or just large text. But you can also apply both classes together to give an element both styles, creating text that is red *and* large. The resulting HTML code would look like the following:

```
<p class="warning large">Internal server error.</p>
```

Demo: Multiple classes

Review and experiment with the code sample in the Repl.it below to learn more about applying multiple classes.

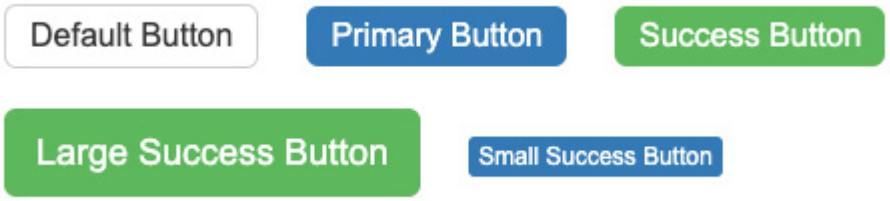


Outline

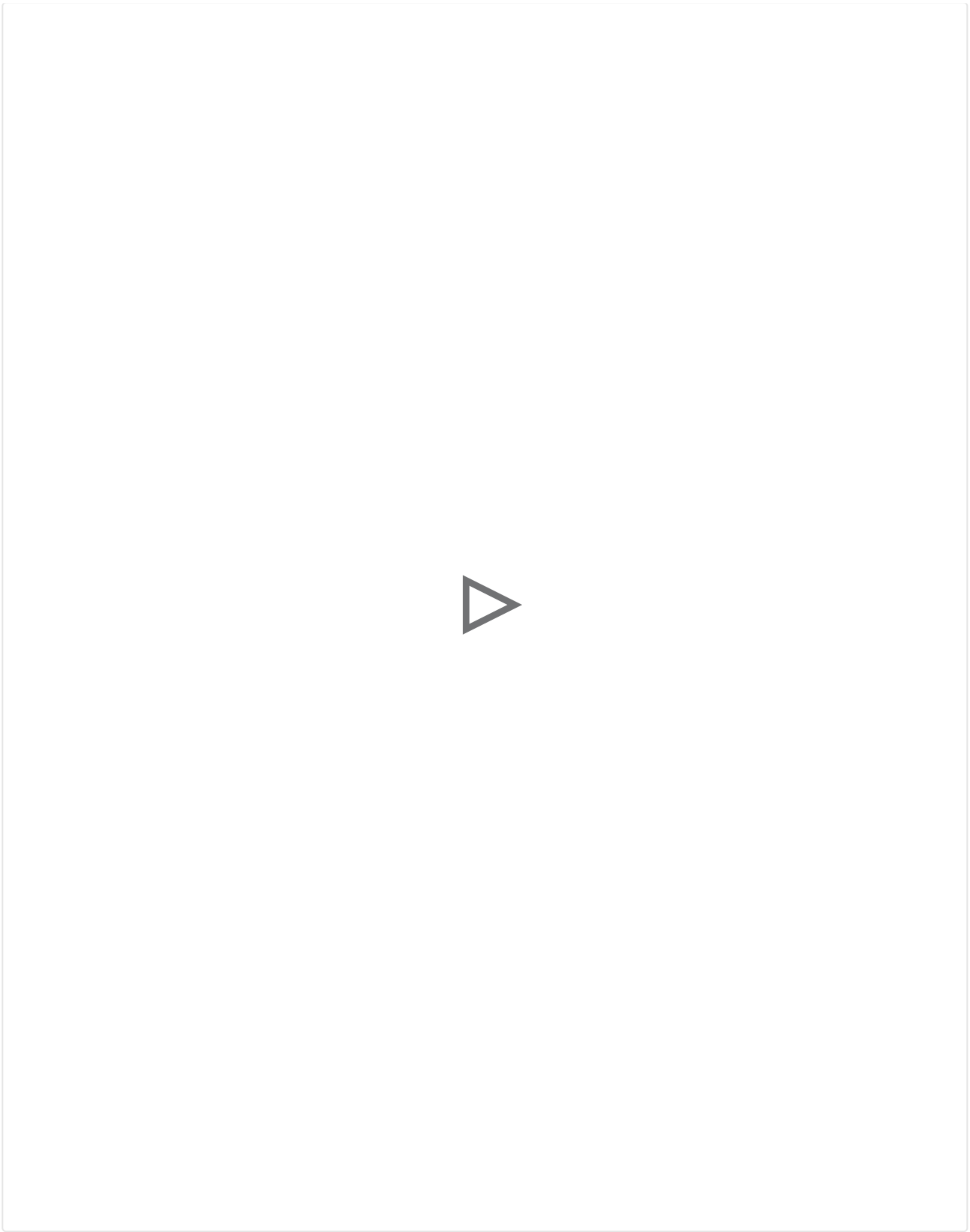
Drill: Styled link practice

In this drill, you'll get a bit more practice. The following Repl.it already has multiple classes in the style sheet. Apply the classes to the HTML code to create the buttons seen in the image below.

Note: In this drill, you only need to style the links. Don't worry if the links in the image are arranged differently from your resulting code sample.



Outline



Outline

When you finish, you can compare your code to this completed sample. But remember, don't look at the code unless you are done or yo[illegible] totally stuck.

Styled Button Practice: Complete

Specific classes

When assigning CSS rules to HTML, you can assign classes to specific HTML elements using a *combination selector*. With the combination selector, a class will only be assigned if it follows the specific rules. This can help you avoid mistakes because the classes will not be assigned if they *don't* follow the specific rules. Take a look at the example below. What do you think this is doing?

```
p.center {
  text-align: center;
}
```

Outline

In this example, the `.center` class applies *only* to paragraphs. In fact, because of the `p.` in `p.center`, this class will not center any text other than paragraphs, even if this class is assigned to other HTML elements.

Take a look at the following Repl.it for an example. What happens when you change the `p` to `h1`?

Specific Classes

Grouping CSS selectors

When multiple CSS classes share the same properties, they can be grouped together. To group them together, you simply need to separate them with a comma `,`. This technique can simplify your code a bit, as you can see below.

This code does *not* have grouped classes.

```
h1 {
  color: green;
  font-family: Arial, sans-serif;
}

h2 {
  color: green;
  font-family: Arial, sans-serif;
}

.green-text {
  color: green;
  font-family: Arial, sans-serif;
}
```

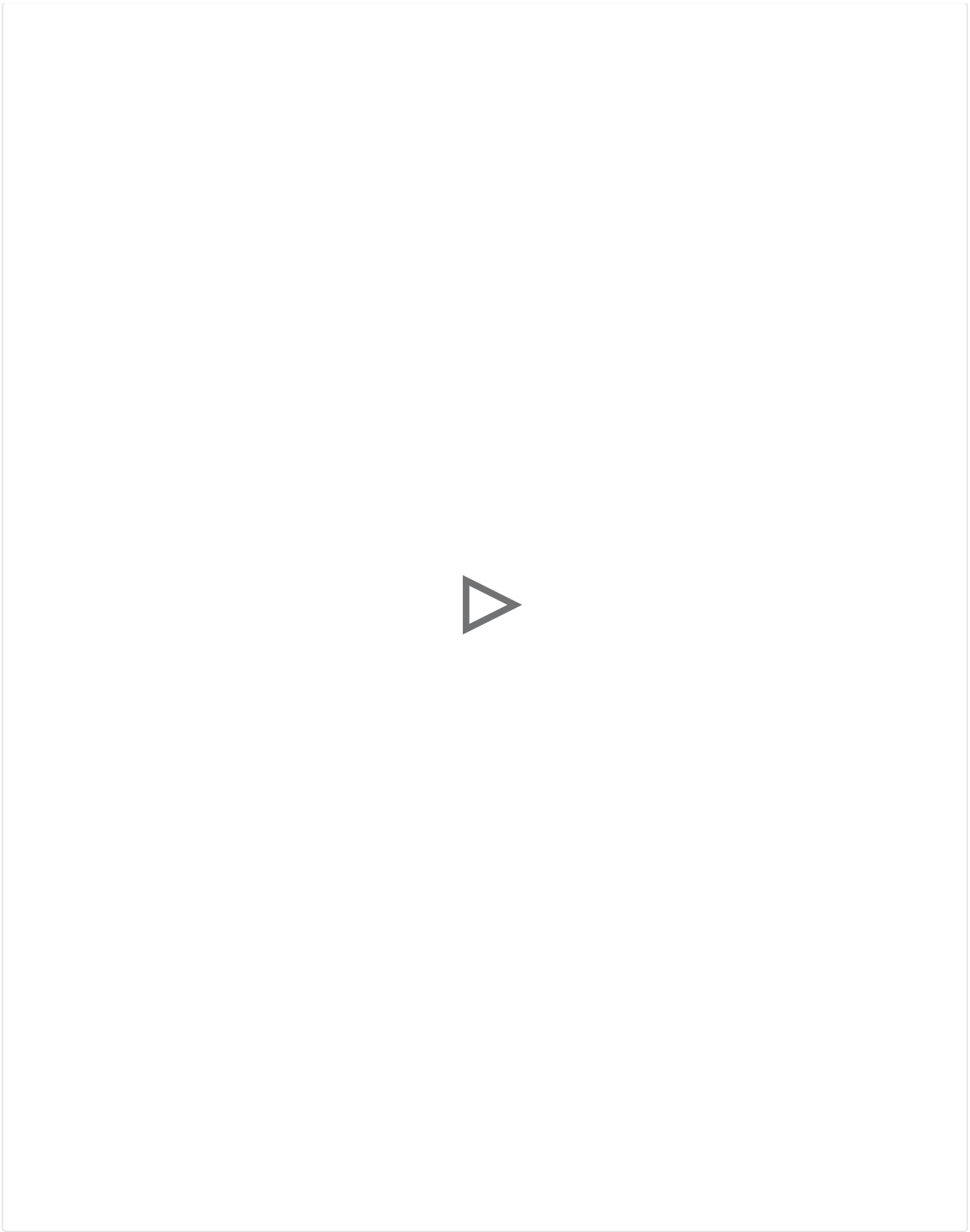
But the code below does! See how much cleaner that is?

Outline

```
h1, h2, .green-text {
  color: green;
  font-family: Arial, sans-serif;
}
```

Drill: Grouped class practice

In this drill, your goal is to group and condense the CSS rules in the Repl.it below.



Outline

When you are finished, feel free to compare your code to this completed project. Don't look at it before you're done unless you're completely stuck.

Grouped Class Practice: Complete

Nesting CSS selectors

Now, you're ready for another technique: nesting CSS. Nesting CSS selectors is extremely useful when you want different sections of your website to style common HTML elements in different ways. Rather than assigning classes that would then need to be assigned to specific HTML elements, CSS can be nested to target certain HTML elements within other HTML elements.

As you may have noticed by this point, one of the keys to writing code is writing the least amount of code possible. Using nested CSS selectors, you have a lot of control over how you style your page without having to add unique classes.

Outline

Take a look at the example below. What do you notice?

```
/* All paragraphs within main are black. */
main p {
  color: black;
}

/* All paragraphs within footer are white. */
footer p {
  color: white;
}
```

The styles assigned in this code sample will be applied to any paragraphs inside of the main container or footer container, regardless of whether they are inside additional containers. If you wanted t more specific, you could use a descendant selector to target *only*

sectors that are immediately within a parent element, as seen below. (But this will get covered in more depth later on.)

```
main > p {
  color: white;
}
```

```
<main>
  <p>
    This text should be white since it is directly wi
  </p>
  <div class="group">
    <p>
      This text will not be targeted since it is not
      container.
    </p>
  </div>
</main>
```


Pseudo-elements: `::before` and `::after`

You're ready for another concept: pseudo-elements. A *pseudo-element* lets you style a specific part of the selected HTML element, such as the first letter or line. It involves adding a keyword to a CSS selector, and takes the following structure: `selector::pseudo-element`. Here, you'll learn about two specific pseudo-elements: `::before` and `::after`.

The `::before` and `::after` pseudo-elements allow you to add content to an HTML element either just before or just after the content of the element. This technique is great for adding certain types of content, such as creating smart quotes around blockquotes. And more

broadly, writing `::before` and `::after` style rules can be a good way to handle repeated visual content that surrounds an element.

Check out the examples below.

```
div::before {
  content: "before";
}

div::after {
  content: "after";
}
```

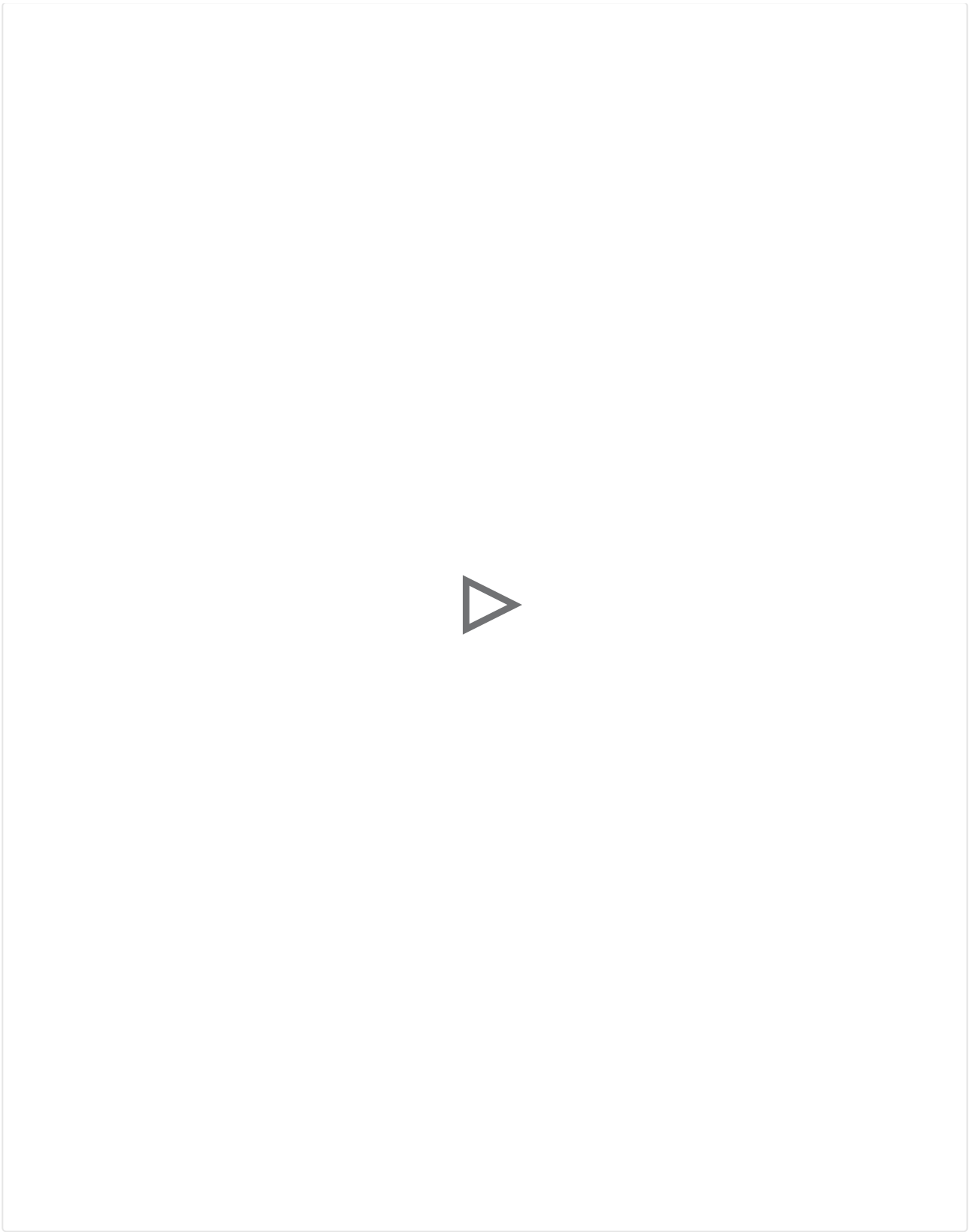
```
<div>
  before
  <!-- Rest of stuff inside the div -->
  after
</div>
```

Outline

Keep in mind this added content is still *inside* the specified element. The names `::before` and `::after` sound a bit like the pseudo-elements would add content outside of the element, either before or after. But actually, they add content before or after the *content* of the element. The new content is still inside the element itself.

Drill: Nested styles

In this code sample, you'll see blocks of HTML and CSS code. Your goal in this drill is to add specific nested CSS styles at the bottom of the CSS style sheet. There are four changes required for this web page.



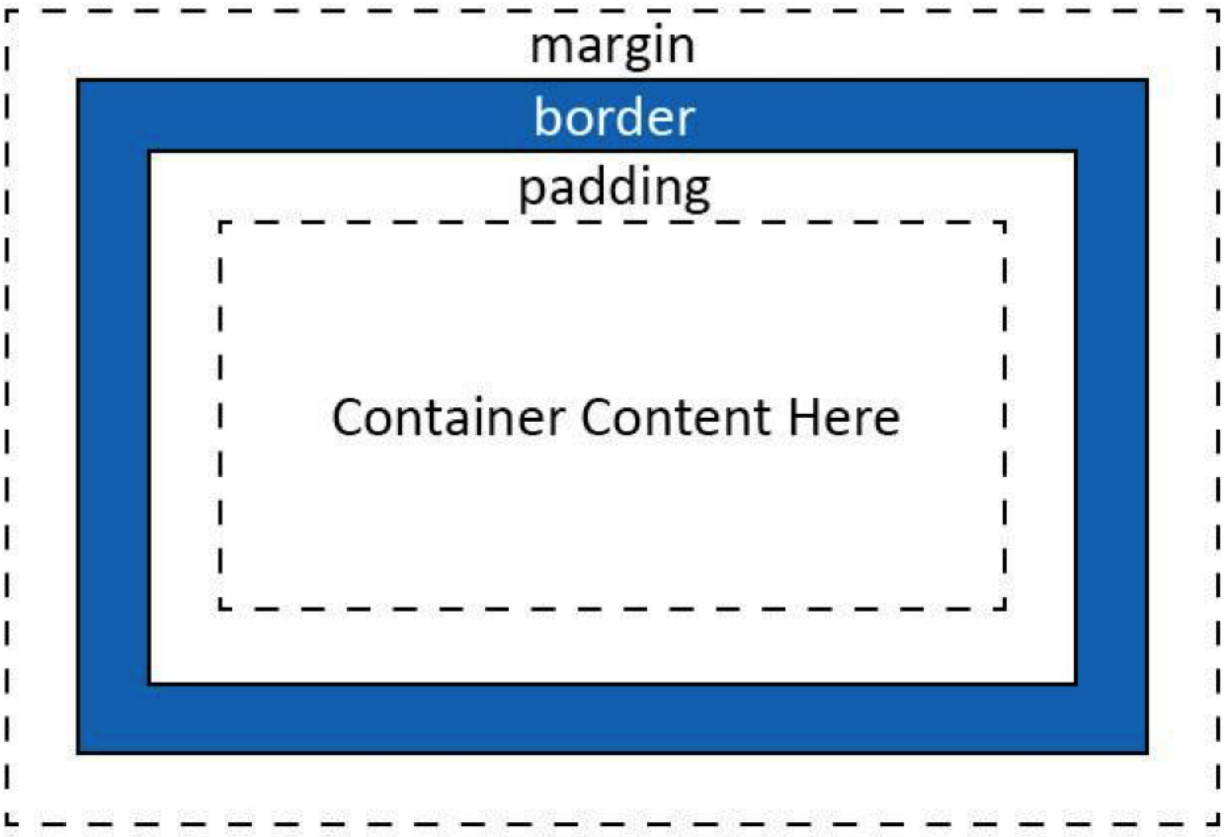
Outline

When you are finished or you wish to compare your code, you can review the completed practice project below.

Nested Practice: Complete

The box model

You're getting good! Next up, there's the box model. The *box model* is one of the most important methods of controlling the space and borders around an HTML element. Take a look at the image below. What do you notice?



Outline

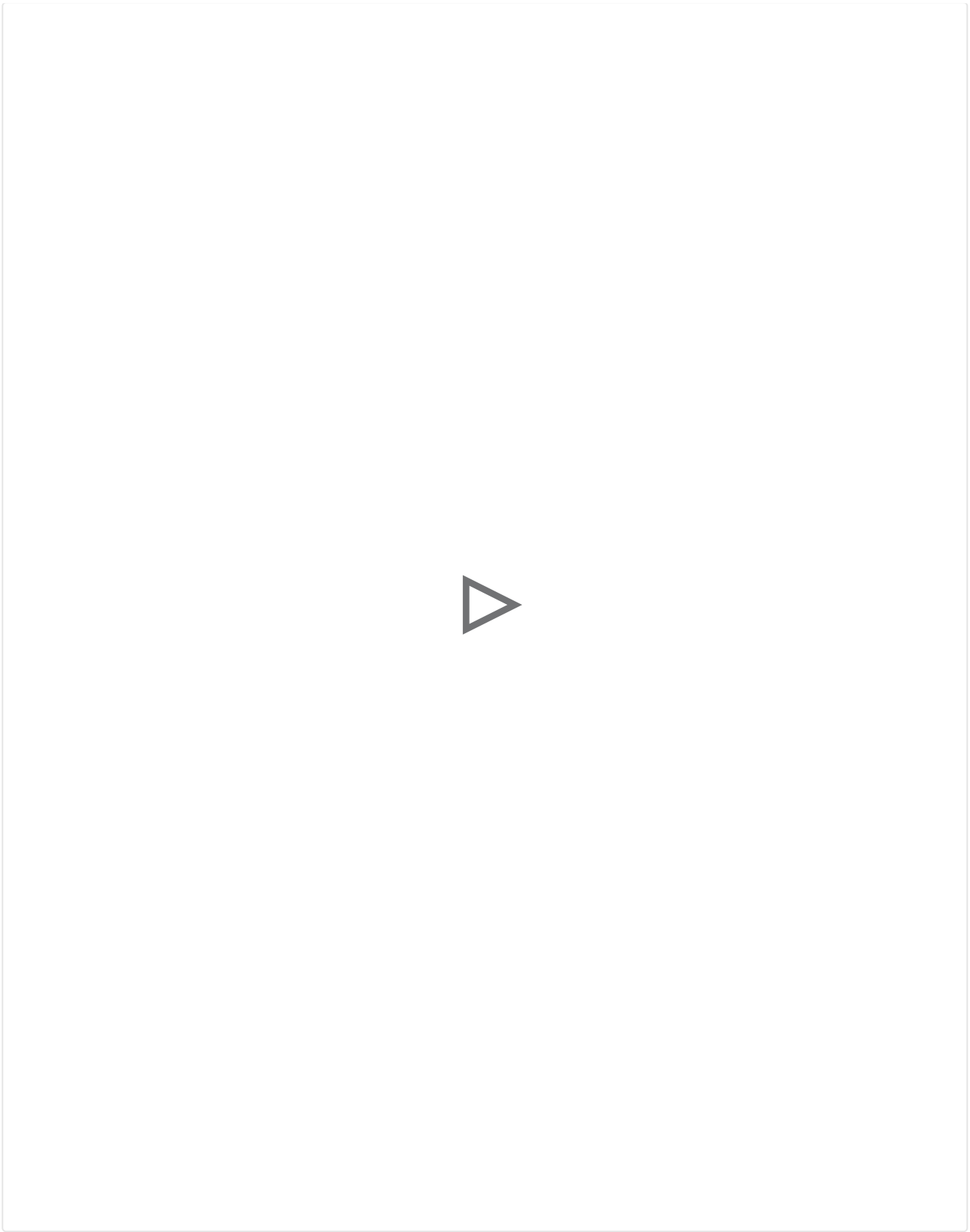
Every HTML element is considered to be a box. Additionally, each element has these four parts:

- **Content**: This is the area in each element where the text, links, and images appear.
- **Border**: This is like a "frame" around the element. Every element can have a visible border, and borders can be styled in va ways.

- **Padding**: This is the space between the border and the content. It takes on the background color assigned to the element.
- **Margin**: This is the space outside of the border. It is transparent, displaying any colors or images behind it.

Demo: Box model

Review and play around with the code sample in the Repl.it below. The code and comments in the CSS help identify the border, padding, and margins. Try to change these values to visually see what is happening before reading further. What do you notice?



Outline

A note on borders

A border can be added to every element. Borders require three properties:

- `width`: The border width is typically set in pixels (`px`).
- `style`: Technically, borders can have a variety of different border styles. But be careful: from a design perspective, styled borders are pretty out of date. These styles can make your web page look antiquated, and even ugly.
- `color`: The border color can be assigned using any color technique you prefer (although a couple of these may still be unfamiliar):
  - HTML color name: `red`
  - Hex: `#ff0000`
  - RGB: `rgb(255,0,0)`
  - HSL: `hsl(0, 100%, 50%)`

Outline

When it comes to borders, the order of the values is not important. See the example below.

```
any-element {
  border: 1px solid #000;
}
```

The link below displays the eight basic HTML border styles for you to review. Used properly, borders can look fresh and modern. But if you don't use them properly, you'll build a website that looks tacky, unattractive, and outdated.

Borders

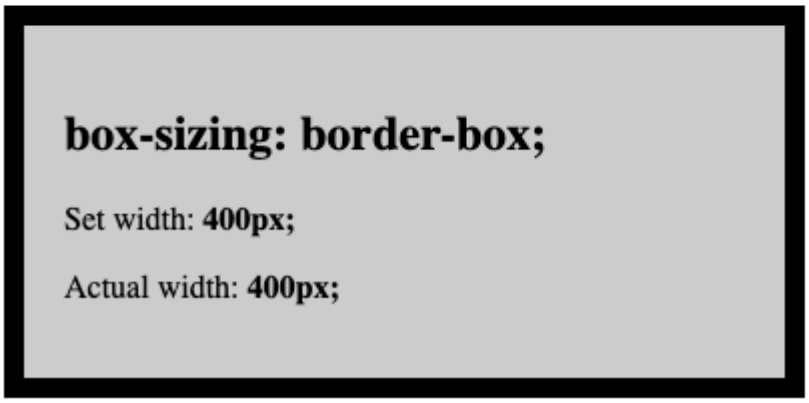
On `box-sizing`

Now, back to the boxes. The `box-sizing` property allows you to include both the padding and the border within the total width and height of an element. But you have to be deliberate about it, because the default does not include them.

There are two different ways to set `box-sizing`, but the second is what you want to work with:

1. `box-sizing: content-box;`: This is the older, default setting.
2. `box-sizing: border-box;`: This is the newer, CSS3 setting.

Take a moment to consider these options. If you apply `box-sizing: border-box;` to an element, the padding and border are included in the width and height of that element. You know what dimensions it will have. By comparison, the old method—`box-sizing: content-box;`—was a mathematical pain. With that approach, a `400px` box would actually display as `460px`, as you can see in both the image and the Repl.it below. This is why you want to assign `box-sizing: border-box;` for every HTML element in each new website you develop.



Outline

Box-sizing comparison

Because this is just an overview, you'll keep it simple for now. Just add this code into the top of your CSS page for every project you work on so that your page doesn't default to the old method of `box-sizing`.

```
/* Set ALL HTML elements with border-box sizing */
* {
  box-sizing: border-box;
}
```

If you'd like to read more about `box-sizing` in CSS, this CSS-Tricks post is an excellent starting point, as is this Mozilla developer article.

This `box-sizing` practice will work for most of the projects y complete during the program. However, as your apps become more

complex, you might find it helpful to add *inheritance*. This will be covered in more depth, but this CSS-Tricks post explains the benefits of inheriting `box-sizing`.

Assigning margins and padding

There are a few different ways to assign values to margins and padding. And with a little practice, you'll find this pretty easy.

When a single value is applied to the margin or padding of a box, the code will apply that value (measured in `px`) to *all* four sides of the HTML element: the top, right, bottom, and left sides. Here are some example values:

- `margin: 25px;`
- `padding: 25px;`

But you don't have to set one value; you can also apply a specific value to each side. When you need to set a specific value to a specific side, you can do so like this:

- `margin-top: 25px;`
- `margin-right: 15px;`
- `margin-bottom: 20px;`
- `margin-left: 12px;`
- `padding-top: 25px;`
- `padding-right: 15px;`
- `padding-bottom: 20px;`

- `padding-left: 12px;`

Or maybe you've provided only two values. In this case, the code will apply the values to the two pairs of sides as follows:

- **First value**: The first value will apply to the top and bottom sides.
- **Second value**: The second value will apply to the left and right sides.
- `margin: 25px 15px;`
- `padding: 25px 15px;`

And finally, when four values are provided, they will always apply to each side of the box in this specific order:

- **First value**: The first value will apply to the top side.
- **Second value**: The second value will apply to the right side.
- **Third value**: The third value will apply to the bottom side.
- **Fourth value**: The fourth value will apply to the left side.
- `margin: 25px 15px 20px 12px;`
- `padding: 25px 15px 20px 12px;`

Outline

Browser defaults versus `normalize.css`

And there you have it—you've learned several challenging CSS concepts in this checkpoint. The final point that's worth reitera here is the issue of browser style defaults. As you learned earlier in this

module, web browsers don't have the exact same settings for the default styles of CSS. But there is a popular and easy-to-implement solution: normalize.css. With this small CSS file, you can guarantee cross-browser consistency for default styles. Revisit the *Web page template* checkpoint to learn more.

Your work

03.11.21 ✓

Completed

Next checkpoint

Outline

How would you rate this content?

Report a typo or other issue

Go to Overview

Outline