Responsive Web Design Notes

Patrick Adams

November 1, 2020

Contents

1 Basic Html And Html5

1.1 Say Hello to HTML Elements

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.

1.2 Headline with the h2 Element

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.

1.3 Inform with the Paragraph Element

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>

1.4 Fill in the Blank with Placeholder Text

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".

1.5 Uncomment HTML

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 start with <!-- and end with a -->

1.6 Comment out HTML

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.

1.7 Delete HTML Elements

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

1.8 Introduction to HTML5 Elements

HTML5 introduces more descriptive HTML tags. These include main, header, footer, nav, video, article, section and others.
These tags give a descriptive structure to your HTML, make your HTML easier to read, and 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.
Example usage, a main element with two child elements nested inside it:
"'html
Hello World
Hello Paragraph
"'
Note: Many of the new HTML5 tags and their benefits are covered in the Applied Accessibility section.

1.9 Add Images to Your Website

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:
<img src="https://www.freecatphotoapp.com/your-image.jpg">
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:
<img src="https://www.freecatphotoapp.com/your-image.jpg" alt="A business cat wearing a necktie.">

1.10 Link to External Pages with Anchor Elements

You can use a (anchor) elements to link to content outside of your web page.
a elements need a destination web address called an href attribute. They also need anchor text. Here's an example:
<a href="https://freecodecamp.org">this links to freecodecamp.org</a>
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.

1.11 Link to Internal Sections of a Page with Anchor Elements

a (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:
"'html
Contacts
...
Contacts
"'
When users click the Contacts link, they'll be taken to the section of the webpage with the Contacts header element.

1.12 Nest an Anchor Element within a Paragraph

You can nest links within other text elements.
"'html
Here's a link to freecodecamp.org for you to follow.
"'
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: ' ... '
The text, "link to freecodecamp.org", within the a 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.

1.13 Make Dead Links Using the Hash Symbol

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.

1.14 Turn an Image into a Link

You can make elements into links by nesting them within an a element.
Nest your image within an a element. Here's an example:
<a href="#"><img src="https://bit.ly/fcc-running-cats" alt="Three kittens running towards the camera."></a>
Remember to use # as your a element's href property in order to turn it into a dead link.

1.15 Create a Bulleted Unordered List

HTML has a special element for creating unordered lists, or bullet point style lists.
Unordered lists start with an opening <ul> element, followed by any number of <li> elements. Finally, unordered lists close with a </ul>
For example:
"'html

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

1.16 Create an Ordered List

HTML has another special element for creating ordered lists, or numbered lists.
Ordered lists start with an opening <ol> element, followed by any number of <li> elements. Finally, ordered lists are closed with the </ol> tag.
For example:
“‘html
Garfield
Sylvester
“‘
would create a numbered list of "Garfield" and "Sylvester".

1.17 Create a Text Field

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.

1.18 Add Placeholder Text to a Text Field

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">
Note: Remember that input elements are self-closing.

1.19 Create a Form Element

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>

1.20 Add a Submit Button to a Form

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>

1.21 Use HTML5 to Require a Field

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>

1.22 Create a Set of Radio Buttons

You can use radio buttons 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:
"'html
Indoor
"'
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:
"'html
Indoor
"'

1.23 Create a Set of Checkboxes

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>

1.24 Use the value attribute with Radio Buttons and Checkboxes

When a form gets submitted, the data is sent to the server and includes entries for the options selected. Inputs of type radio and checkbox report their values from the value attribute.
For example:
"'html
Indoor
Outdoor
"'
Here, you have two radio inputs. When the user submits the form with the indoor option selected, the form

data will include the line: indoor-outdoor=indoor. This is from the name and value attributes of the "indoor" input.
If you omit the value attribute, the submitted form data uses the default value, which is on. In this scenario, if the user clicked the "indoor" option and submitted the form, the resulting form data would be indoor-outdoor=on, which is not useful. So the value attribute needs to be set to something to identify the option.

1.25 Check Radio Buttons and Checkboxes by Default

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>

1.26 Nest Many Elements within a Single div Element

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>.

1.27 Declare the Doctype of an HTML Document

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:
“‘html

“‘

1.28 Define the Head and Body of an HTML Document

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:
“‘html

“‘

2 Basic Css

2.1 Change the Color of Text

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 ; .

2.2 Use CSS Selectors to Style Elements

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:
"'html
"'
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:
"'html
"'
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.

2.3 Use a CSS Class to Style an Element

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

2.4 Style Multiple Elements with a CSS Class

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.

2.5 Change the Font Size of an Element

Font size is controlled by the font-size CSS property, like this:
"'css
h1 {

font-size: 30px;
}
“‘

2.6 Set the Font Family of an Element

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:
“‘css
h2 {
font-family: sans-serif;
}
“‘

2.7 Import a Google Font

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.

2.8 Specify How Fonts Should Degrade

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:
“‘css
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.

2.9 Size Your Images

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:
“‘html
“‘

2.10 Add Borders Around Your Elements

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:
“‘html
“‘

2.11 Add Rounded Corners with border-radius

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

2.12 Make Circular Images with a border-radius

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

2.13 Give a Background Color to a div Element

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:
“‘css
.green-background {
background-color: green;
}
“‘

2.14 Set the id of an Element

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">

2.15 Use an id Attribute to Style an Element

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:
“‘css
#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.

2.16 Adjust the Padding of an Element

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, border, and margin.
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.

2.17 Adjust the Margin of an Element

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.

2.18 Add a Negative Margin to an Element

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.

2.19 Add Different Padding to Each Side of an Element

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.

2.20 Add Different Margins to Each Side of an Element

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.

2.21 Use Clockwise Notation to Specify the Padding of an Element

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.

2.22 Use Clockwise Notation to Specify the Margin of an Element

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.

2.23 Use Attribute Selectors to Style Elements

You have been adding 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:
“‘css

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

“‘

2.24 Understand Absolute versus Relative Units

The last several challenges all set an element's margin or padding with pixels (px). Pixels are a type of length unit, which is what 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.

2.25 Style the HTML Body Element

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

2.26 Inherit Styles from the Body Element

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.

2.27 Prioritize One Style Over Another

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?

2.28 Override Styles in Subsequent CSS

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?

2.29 Override Class Declarations by Styling ID Attributes

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.

2.30 Override Class Declarations with Inline Styles

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?

2.31 Override All Other Styles by using Important

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.

2.32 Use Hex Code for Specific Colors

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.
“‘css
body {
color: #000000;
}
“‘

2.33 Use Hex Code to Mix Colors

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.

2.34 Use Abbreviated Hex Code

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.

2.35 Use RGB values to Color Elements

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.
“‘css
body {
background-color: rgb(255, 165, 0);
}
“‘

2.36 Use RGB to Mix Colors

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

2.37 Use CSS Variables to change several elements at once

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.

2.38 Create a custom CSS Variable

To create a CSS variable, you just need to give it a name with two hyphens in front of it and assign it a value like this:
“‘css
--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.

2.39 Use a custom CSS Variable

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

2.40 Attach a Fallback value to a CSS Variable

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:
“‘css
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.

2.41 Improve Compatibility with Browser Fallbacks

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.

2.42 Inherit CSS Variables

When you create a variable, it is available for you to use inside the selector in which you create it. It also is available in any of that selector's descendants. This happens because CSS variables are inherited, just like ordinary properties.
To make use of inheritance, 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 html element. By creating your variables in :root, they will be available globally and can be accessed from any other selector in the style sheet.

2.43 Change a variable for a specific area

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.

2.44 Use a media query to change a variable

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.

3 Applied Visual Design

3.1 Create Visual Balance Using the text-align Property

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.

3.2 Adjust the Width of an Element Using the width Property

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:

```
“‘css
img {
width: 220px;
}
“‘
```

3.3 Adjust the Height of an Element Using the height Property

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:

```
“‘css
img {
height: 20px;
}
“‘
```

3.4 Use the strong Tag to Make Text Bold

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.

3.5 Use the u Tag to Underline Text

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.

3.6 Use the em Tag to Italicize Text

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.

3.7 Use the s Tag to Strikethrough Text

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.

3.8 Create a Horizontal Line Using the hr Element

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.

3.9 Adjust the background-color Property of Text

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.

3.10 Adjust the Size of a Header Versus a Paragraph Tag

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.

3.11 Add a box-shadow to a Card-like Element

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
color, in that order.
The blur-radius and spread-radius values are optional.
Multiple box-shadows can be created by using commas to separate properties of each box-shadow element.
Here's an example of the CSS to create multiple shadows with some blur, at mostly-transparent black colors:
"'css
box-shadow: 0 10px 20px rgba(0,0,0,0.19), 0 6px 6px rgba(0,0,0,0.23);
"'

3.12 Decrease the Opacity of an Element

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.

3.13 Use the text-transform Property to Make Text Uppercase

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-transformvalues change the example text "Transform me". ValueResultlowercase"transform me"uppercase"TRANSFORM ME"capitalize"Transform Me"initialUse the default valueinheritUse the text-transform value from the parent elementnoneDefault: Use the original text

3.14 Set the font-size for Multiple Heading Elements

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.

3.15 Set the font-weight for Multiple Heading Elements

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.

3.16 Set the font-size of Paragraph Text

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

3.17 Set the line-height of Paragraphs

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.

3.18 Adjust the Hover State of an Anchor Tag

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:
“‘css
a:hover {
color: red;
}
“‘

3.19 Change an Element's Relative Position

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:
“‘css
```
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.

3.20 Move a Relatively Positioned Element with CSS Offsets

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.

3.21 Lock an Element to its Parent with Absolute Positioning

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.

3.22 Lock an Element to the Browser Window with Fixed Positioning

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.

3.23 Push Elements Left or Right with the float Property

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.

3.24 Change the Position of Overlapping Elements with the z-index Property

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.

3.25 Center an Element Horizontally Using the margin Property

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.

3.26 Learn about Complementary Colors

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.

3.27 Learn about Tertiary Colors

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:
ColorHex Codeorange#FF7F00cyan#00FFFFraspberry#FF007F

3.28 Adjust the Color of Various Elements to Complementary Colors

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.

3.29 Adjust the Hue of a Color

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:
ColorHSLredhsl(0, 100%, 50%)yellowhsl(60, 100%, 50%)greenhsl(120, 100%, 50%)cyanhsl(180, 100%, 50%)bluehsl(240, 100%, 50%)magentahsl(300, 100%, 50%)

3.30 Adjust the Tone of a Color

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.

3.31 Create a Gradual CSS Linear Gradient

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 horizontal gradient (from left to right) 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));

3.32 Use a CSS Linear Gradient to Create a Striped Element

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.

3.33 Create Texture by Adding a Subtle Pattern as a Background Image

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.

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

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:
“‘css
p {
transform: scale(2);
}
“‘

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

The transform property has a variety of functions that let 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:
“‘css
p:hover {
transform: scale(2.1);
}
“‘
Note: Applying a transform to a div element will also affect any child elements contained in the div.

3.36 Use the CSS Transform Property skewX to Skew an Element Along the X-Axis

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

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

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.

3.38 Create a Graphic Using CSS

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.

3.39 Create a More Complex Shape Using CSS and HTML

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:
```css
.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.

3.40 Learn How the CSS @keyframes and animation Properties Work

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:
“‘css
```
#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%.

3.41 Use CSS Animation to Change the Hover State of a Button

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:
“‘html
“‘

3.42 Modify Fill Mode of an Animation

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;

3.43 Create Movement Using CSS Animation

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.
“‘css
```
@keyframes rainbow {
0% {
background-color: blue;
top: 0px;
}
50% {
background-color: green;
top: 50px;
}
100% {
background-color: yellow;
top: 0px;
}
}
```
“‘

3.44 Create Visual Direction by Fading an Element from Left to Right

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.

3.45 Animate Elements Continually Using an Infinite Animation Count

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.

3.46 Make a CSS Heartbeat using an Infinite Animation Count

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.

3.47 Animate Elements at Variable Rates

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.

3.48 Animate Multiple Elements at Variable Rates

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.

3.49 Change Animation Timing with Keywords

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.

3.50 Learn How Bezier Curves Work

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.

3.51 Use a Bezier Curve to Move a Graphic

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.

3.52 Make Motion More Natural Using a Bezier Curve

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 a 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.

4 Applied Accessibility

4.1 Add a Text Alternative to Images for Visually Impaired Accessibility

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:
<img src="importantLogo.jpeg" alt="Company logo">
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.

4.2 Know When Alt Text Should be Left Blank

In the last challenge, you learned that including an alt attribute when using 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:
<img src="visualDecoration.jpeg" alt="">
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.

4.3 Use Headings to Show Hierarchical Relationships of Content

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.

4.4 Jump Straight to the Content Using the main Element

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.

4.5 Wrap Content in the article Element

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 divThe 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.
“‘html
- groups content
- groups related content
- 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.
Tests
“‘yml
tests:
- text: Your code should have three article tags.
testString: assert($('article').length == 3);
- text: Your code should not have any div tags.
testString: assert($('div').length == 0);
“‘
Challenge Seed
“‘html
Deep Thoughts with Master Camper Cat
The Garfield Files: Lasagna as Training Fuel?
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...
Defeating your Foe: the Red Dot is Ours!
Felines the world over have been waging war on the most persistent of foes. This red nemesis combines both cunning stealth and lightning speed. But chin up, fellow fighters, our time for victory may soon be near...
Is Chuck Norris a Cat Person?
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?...
“‘

Solution
“‘html
Deep Thoughts with Master Camper Cat
The Garfield Files: Lasagna as Training Fuel?
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...
Defeating your Foe: the Red Dot is Ours!
Felines the world over have been waging war on the most persistent of foes. This red nemesis combines both cunning stealth and lightning speed. But chin up, fellow fighters, our time for victory may soon be near...
Is Chuck Norris a Cat Person?
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?...
“‘

4.6 Make Screen Reader Navigation Easier with the header Landmark

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: The 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.

4.7 Make Screen Reader Navigation Easier with the nav Landmark

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.

4.8 Make Screen Reader Navigation Easier with the footer Landmark

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.

4.9 Improve Accessibility of Audio Content with the audio Element

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:
"'html

"'
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.

4.10 Improve Chart Accessibility with the figure Element

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:
"'html

Master Camper Cat demonstrates proper form of a roundhouse kick.

"'

4.11 Improve Form Field Accessibility with the label Element

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:
"'html
Name:
"'

4.12 Wrap Radio Buttons in a fieldset Element for Better Accessibility

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:
"'html

Choose one of these three items:
Choice One
Choice Two
Choice Three
“‘

4.13 Add an Accessible Date Picker

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:
“‘html
Enter a date:
“‘

4.14 Standardize Times with the HTML5 datetime Attribute

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>

4.15 Make Elements Only Visible to a Screen Reader by Using Custom CSS

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:
“‘css
.sr-only {
position: absolute;
left: -10000px;
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

4.16 Improve Readability with High Contrast Text

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.

4.17 Avoid Colorblindness Issues by Using Sufficient Contrast

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.

4.18 Avoid Colorblindness Issues by Carefully Choosing Colors that Convey Information

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.

4.19 Give Links Meaning by Using Descriptive Link Text

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.

4.20 Make Links Navigable with HTML Access Keys

HTML offers the accesskey attribute to specify a shortcut key to activate or bring focus to an element. This can make navigation more efficient for keyboard-only users.
HTML5 allows this attribute to be used on any element, but it's particularly useful when it's used with interactive ones. This includes links, buttons, and form controls.
Here's an example:
<button accesskey="b">Important Button</button>

4.21 Use tabindex to Add Keyboard Focus to an Element

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.

4.22 Use tabindex to Specify the Order of Keyboard Focus for Several Elements

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>

5 Responsive Web Design Principles

5.1 Create a Media Query

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.

5.2 Make an Image Responsive

Making images responsive with CSS is actually very simple. You just need to add these properties to an image:
“‘css
img {
max-width: 100%;
height: auto;
}
“‘
The ‘max-width‘ of ‘100%‘ will make sure the image is never wider than the container it is in, and the ‘height‘ of ‘auto‘ will make the image keep its original aspect ratio.

5.3 Use a Retina Image for Higher Resolution Displays

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 that 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:
“‘html
“‘

5.4 Make Typography Responsive

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 (viewport width): 10vw would be 10% of the viewport's width.vh (viewport height): 3vh would be 3% of the viewport's height.vmin (viewport minimum): 70vmin would be 70% of the viewport's smaller dimension (height or width).vmax (viewport maximum): 100vmax would be 100% of the viewport's bigger dimension (height or width).
Here is an example that sets a body tag to 30% of the viewport's width.
body { width: 30vw; }

6 Css Flexbox

6.1 Use display: flex to Position Two Boxes

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.

6.2 Add Flex Superpowers to the Tweet Embed

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.

6.3 Use the flex-direction Property to Make a Row

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.

6.4 Apply the flex-direction Property to Create Rows in the Tweet Embed

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.

6.5 Use the flex-direction Property to Make a Column

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.

6.6 Apply the flex-direction Property to Create a Column in the Tweet Embed

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.

6.7 Align Elements Using the justify-content Property

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. This is the default alignment if no justify-content is specified.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 remaining space is distributed evenly among the other items.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 with a half space on either end of the flex container.space-evenly: Distributes space evenly between the flex items with a full space at either end of the flex container

6.8 Use the justify-content Property in the Tweet Embed

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.

6.9 Align Elements Using the align-items Property

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.

6.10 Use the align-items Property in the Tweet Embed

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.

6.11 Use the flex-wrap Property to Wrap a Row or Column

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.

6.12 Use the flex-shrink Property to Shrink Items

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.

6.13 Use the flex-grow Property to Expand Items

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.

6.14 Use the flex-basis Property to Set the Initial Size of an Item

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.

6.15 Use the flex Shorthand Property

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;.

6.16 Use the order Property to Rearrange Items

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.

6.17 Use the align-self Property

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.

7 Css Grid

7.1 Create Your First CSS Grid

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.

7.2 Add Columns with grid-template-columns

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:
“‘css
```
.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.

7.3 Add Rows with grid-template-rows

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.

7.4 Use CSS Grid units to Change the Size of Columns and Rows

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:
“‘css
```
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.

7.5 Create a Column Gap Using grid-column-gap

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:
“‘css
```
grid-column-gap: 10px;
```

“‘
This creates 10px of empty space between all of our columns.

7.6 Create a Row Gap using grid-row-gap

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.

7.7 Add Gaps Faster with grid-gap

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.

7.8 Use grid-column to Control Spacing

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 lines1234row lines1234
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:
“‘css
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.

7.9 Use grid-row to Control Spacing

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.

7.10 Align an Item Horizontally using justify-self

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.

7.11 Align an Item Vertically using align-self

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.

7.12 Align All Items Horizontally using justify-items

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.

7.13 Align All Items Vertically using align-items

Using the align-items property on a grid container will set the vertical alignment for all the items in our grid.

7.14 Divide the Grid Into an Area Template

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:
“‘css
```
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.

7.15 Place Items in Grid Areas Using the grid-area Property

After creating an area's 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:
“‘css
```
.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.

7.16 Use grid-area Without Creating an Areas Template

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:
“‘css

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:
“‘css
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.

7.17 Reduce Repetition Using the repeat Function

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.
Let’s 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.
“‘css
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:
“‘css
grid-template-columns: repeat(2, 1fr 50px) 20px;
“‘
This translates to:
“‘css
grid-template-columns: 1fr 50px 1fr 50px 20px;
“‘
Note: The 1fr 50px is repeated twice followed by 20px.

7.18 Limit Item Size Using the minmax Function

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:
“‘css
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.

7.19 Create Flexible Layouts Using auto-fill

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, like this:
“‘css
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.

7.20 Create Flexible Layouts Using auto-fit

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.

7.21 Use Media Queries to Create Responsive Layouts

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.

7.22 Create Grids within Grids

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.

8 Responsive Web Design Projects

8.1 Build a Tribute Page

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.

8.2 Build a Survey Form

Objective: Build a CodePen.io 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.

8.3 Build a Product Landing Page

Objective: Build a CodePen.io 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).
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.

8.4 Build a Technical Documentation Page

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.

8.5 Build a Personal Portfolio Webpage

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.