1.2.7 Add Detailed CSS Styles

Now that we've created our base `<header>` styles, let's adjust some of the elements nested inside of it, starting with the `<h1>`:

```
header h1 {
  font-weight: bold;
  font-size: 36px;
  color: #fce138;
  margin: 0;
}
```

Here, we're implementing a more specific selector pattern. This one ensures that we are only applying styles specific to a particular `<h1>` element: the one that lives **inside** a `<header>` element. This is a great method for adding specificity to our styles to keep them scoped to particular section and context. We'll do more of this next with some different combinations of specificity and leverage the "C" in CSS, which stands for **Cascade**.

The CSS Cascade

The cascade follows three factors:

1. **Importance**: When you add `!important` to the end of a property declaration, it will override any conflicting style declarations for that element. This is not recommended because overriding the default "cascading" behavior of CSS will make your site harder to maintain.
2. **Specificity**: CSS actually weighs the importance of different types of selectors used by how specific they are. If we were to apply a style by selecting `h1`, it will apply to all `<h1>` elements. But if we were to then apply a style by selecting `header h1`, it will ignore conflicting property declarations in the `h1` definition and apply `header h1` instead because it is a more specific selection.
3. **Source Order**: There is nothing that will stop us from accidentally selecting and defining styles to the same element more than once, but CSS is read top-down. This means that if we select `h1` and give it a color of red on line 1, then select it again and give it a color of blue on line 4, our `<h1>` element is going to be blue because it was defined later.

CSS styles are also applied through something known as **inheritance**, which means that if a style isn't explicitly defined for a child element, it will use the style being applied to the parent element.

To learn more,see **the MDN web docs on cascade and inheritance (https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/Cascade_and_inheritance)** .

Now let's turn our attention to the property declarations in our code:

- `font-size`: The size of the font in the `<h1>` element.
- `font-weight`: Sets the font to `lighter`, `normal`, or `bold`. There are other values associated with this property, but there's no need to dive into them now.

The cascade in CSS can be tricky to understand at first; so before we jump back into our styling, let's learn a bit more about it with this video:

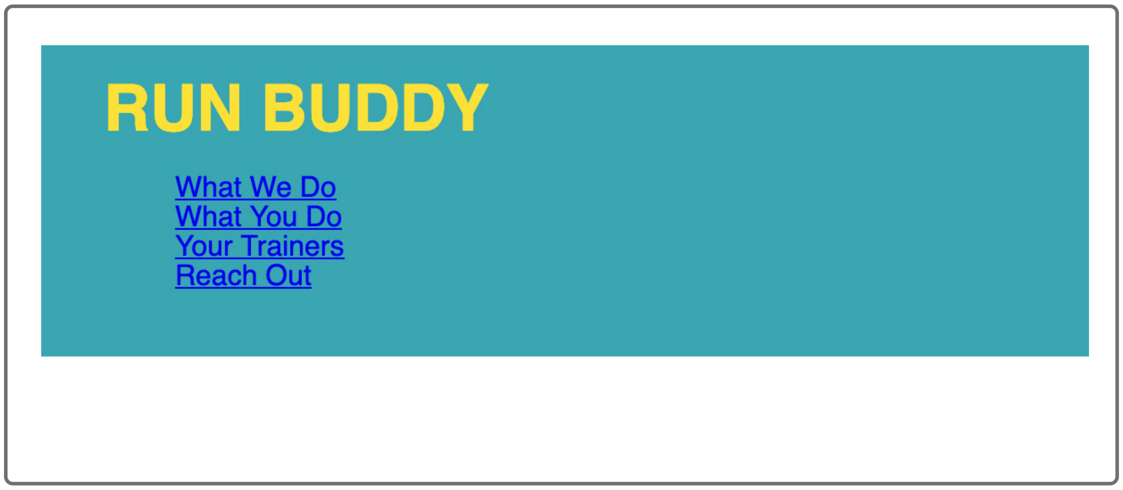
IMPORTANT

In HTML, there will be many cases where the same elements are used for very different reasons in a document. This will typically mean that the CSS applied to them needs to be different as well. How CSS determines what styles are applied to specific elements when there are multiple instances of them on a page can be described by using a word in its name: "cascading."

The **cascade** is a set of rules CSS follows when determining the order of importance when it comes to applying styles. Say, for instance, we have multiple `<a>` elements in the `<header>` that we want to make yellow, but we want to make the `<a>` element in the `<footer>` blue. This can be achieved by being more specific in our selection of elements and saying "let's select all `<a>` that are in `<header>` and do this with them," meaning we can only focus on elements inside another element.

- `color`: Sets the color for this particular `<h1>` element (notice how it overrides the `color` we set for `<body>`; this is because it is applied directly to the element).

Reload your page in the browser. The header should now look like this image:



If your header doesn't look like this, go back and review the CSS and see where things went awry. If it does, great job! Let's start moving that navigation bar over to the right.

These next steps introduce more selectors. Keep in mind that this is just one of many ways to select and apply styles. Over time as you hone your skills, you can move on to more complex selectors, but for now we'll stick with the simpler ones.

We'll explain these properties when we're all done, so don't worry if it's confusing at first. In between applying these styles, make sure to save and refresh the page in the browser to see the changes happening!

Here's a rundown of what we'll be doing:

1. Apply styles to every `<a>` element in the header.

2. Apply styles to the `<nav>` element.

3. Apply styles to the `<nav>` element's `<li>` elements.
4. Apply styles to the `<a>` elements inside of the `<li>` elements.
5. Marvel at the progress we've made!

Let's start with applying styles to every `<a>` element in the header. To begin, add the following CSS:

```
header a {
  text-decoration: none;
  color: #fce138;
}
```

In the above code, we are styling all `<a>` elements inside `<header>`, including those in `<h1>` and `<nav>`. We used a new property here as well:

- **`text-decoration`**: Applies `underline`, `strikethrough`, or `overline` styles to the text. By default, the value is `none`, so this is usually not something we have to explicitly tell it not to do. With `<a>` elements, however, the browser automatically applies a blue color and an underline, and we don't want our links to look like that.

Now we'll move on to styling the `<nav>` element.

```
header nav {
  float: right;
  margin: 7px 0;
}
```

Here, we've done something fairly drastic. We took the `<nav>` element and moved it to the right side.

As we've seen so far, most HTML elements position themselves along the left side of the page with one following the other. So we've taken one totally out of the normal "flow" of the page. The property used here is called `float`:

- `float`: Think of this as similar to the text-wrapping property in Microsoft Word, which takes elements that want to take up 100% of its parent's width by default (known in CSS as **block elements**) and pushes everything after it below it—even if it physically isn't 100% of its parents width—and allows other elements to come along side of it or wrap around it (known in CSS as **inline elements**). This property is used when we have HTML elements that would look better side by side, and we want to use our horizontal space in a more meaningful way. There are other CSS properties that allow us to turn block elements into inline elements, but using `float` in this case made more sense because we needed to turn this element into an `<inline>` element and also move it to the right. `float` let's us do both at once.

DEEP DIVE

DEEP DIVE

The browser wants to interpret and position certain HTML elements in a specific way. This concept is called **flow**. Normal flow in HTML is a page with no CSS overriding default layout styles. This flow follows two directions: **block** (top to bottom) and **inline** (left to right)

In HTML, certain elements are designed by default to take up 100% of the width of whatever the parent element is. If the parent element is 800px wide, then the child is 800px wide and won't allow anything to the left or right of it. This is what's known as a block-level element. Popular elements that have a default block styling are all `<h1>`–`<h6>`

elements, `<div>`, `section`, `<nav>`, `<header>`, `<footer>`, and `<li>`.

The other type of element default is an inline element. This means that the element will only take up the space it needs to take up and not demand 100% width. These are used to allow elements to the left or right of them. The most popular element that is an inline element is the `<a>` element, but there will be more that we get into later.

CSS allows us to override these elements' default layout definitions through a few different ways, but the most on-the-nose one is to apply a `display` property to that element. For instance, `display: block` is used to take an element and force it to take up 100% of the width of its parent. On the other hand, `display: inline` makes an element only take up the space it needs and allows other elements to flow "in line" with it horizontally.

We've also added in a little bit of a top and bottom `margin` here too:

- **`margin`**: We've discussed `margin` before, but just to reiterate, this one has a value of `7px 0`, which means it has `7px` of space added to the top and bottom, but `0px` to the left and right.

Now that we have our two main pieces (`<h1>` and `<nav>`) in position, let's move into the `<nav>` elements.

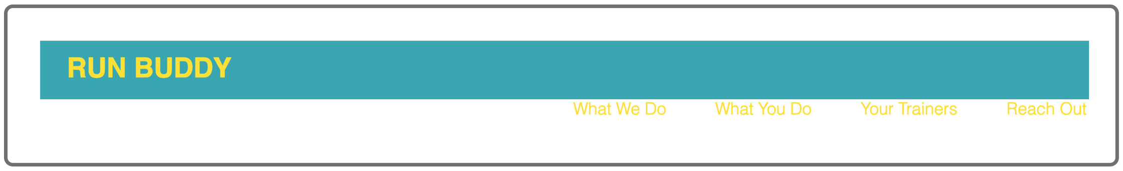
```
header nav ul li {
  display: inline;
}
```

Did you notice that before we added `display: inline`, the list looked like, well, a list? This is because each `<li>` is a block element, meaning the browser lets it take up 100% of the width of whatever parent element it's in. As mentioned above, block elements will always force the next element to be on the next line, so we had to make it an inline element. This is another case of us overriding a default style that the browser provides `<li>` elements.

The last one we need to hit is the `<nav>`'s `<a>` elements. Notice how we've already applied styles to the `<a>` elements in the `<header>`, but now we need to be more specific and give only these particular `<a>` elements styles that the other `<a>` element doesn't need. So now these `<a>` elements will receive not only the styles we added earlier, but these styles as well.

```
header nav ul li a {
  margin: 0 30px;
  font-weight: lighter;
  font-size: 22px;
}
```

There are still a few tweaks we need to add to our CSS to get it aligned perfectly. This will require us to override some browser quirks. For now, our `<header>` should look something like this image:



PAUSE

Considering the above, what do these other selectors say:

- `header nav ul li`
- `header nav`

- Select all `<li>` elements inside of a `<ul>` element inside of a `<nav>` element inside of a `<header>` element
- Select all `<nav>` elements inside of a `<header>` element

Hide Answer

Okay, this has been a lot to take in, but hopefully it has given you a basic understanding of how things want to behave and how you can undo that if you want to. With that said, let's head back and take a look at fixing the little issue in the `<header>`.

Think about that list of block level elements above and see if there's any element in the `<header>` that is taking up more width than it needs to.

HIDE HINT

It's on the lefthand side and it's not the `<nav>` element!

Now that we've figured out that our `<h1>` element is the wrench in the works, let's add another style property to our already existing style. In particular, let's set that element's `display` property to `<inline>`:

```
header h1 {
  font-weight: bold;
  font-size: 36px;
  color: #fce138;
  margin: 0;
  display: inline;
}
```

So just by adding that one property declaration and setting the `<h1>` to `inline` instead of `block`, we moved the `<nav>` on the righthand side up—but not as far as we thought it would. What gives? If we took the `<h1>` element out of `block` styling, shouldn't that allow whatever is coming to the right of it to be on the same line?

This is happening for the same reason that the teal background of the `<header>` doesn't come flush up against the top-left of the page, which is what we were hoping for. The good news is that we can fix both of these problems at once, while also preventing similar future problems as well.

As we've mentioned, the browser has some thoughts of its own about styling some elements. Without CSS, browsers give heading elements (`<h1>`–`<h6>`) some default styling, such as making them bold and adding padding around them to, well, make them look like headings! As a matter of fact, the browser provides built-in margins and padding to a lot of elements by default. This is a remnant of a world before CSS, when the browser did more of the heavy lifting when it came to styling text.

Before we can start styling a page using CSS, we need to remove these default browser-enforced styles. To do this, we apply some default CSS values for every element in the page to level the playing field all at once.

Let's go head and do that by adding this to the very top of our style sheet:

```
* {
  margin: 0;
  padding: 0;
}
```

We just told every element on the page to not have any margin or border unless we explicitly tell it to. Now we don't have to concern ourselves with undoing built-in browser styles one by one.

The asterisk `*` we used here is used quite often in programming. It is typically called a wildcard, but in CSS it is known as a **universal selector**.

This is essentially a catch-all selector that says, "I won't match one thing—I'll match everything!"

DEEP DIVE

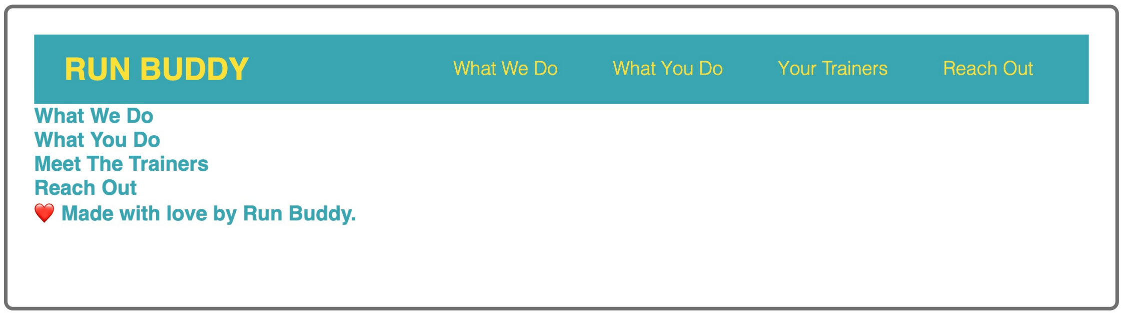
DEEP DIVE

The `*` selector is powerful. To learn more, see **the MDN web docs on universal selectors (https://developer.mozilla.org/en-US/docs/Web/CSS/Universal_selectors)**.

Okay, so now we're looking good, right? The header is flush up against the top-left corner of the page, so there's no weird white gap. The navigation is nice and directly to the right of the `<h1>`.

We can safely say at this point that we've finished our `<header>`! Woo-hoo!

Refresh the page in the browser. It should look like this image:



Awesome job! It's time to move on to styling the `<footer>`. Don't worry—we went through so much in this one section that a lot of the work we do next won't be as long or difficult.

Don't forget to save your work and push to GitHub!