1.7.6 Add Style

Let's jump right in and add our `style.css` file to this HTML document. Do you remember which HTML tag we used in `index.html` to bring in our style sheet? Go ahead and add that to the privacy policy document's `<head>`.

PAUSE

Which one of the following uses relative pathing and which one uses absolute pathing? Which one is preferred?

```
<link rel="stylesheet" href="./assets/css/style.css" />

<link rel="stylesheet" href="/Users/<username>/Desktop/run-b
```

The first is relative pathing and the second is absolute pathing. Relative is preferred because no matter where the project folder ends up, the files will stay related to one another. Absolute pathing means it only works at that exact path and if the project folder is relocated, the path will have to change.

Hide Answer

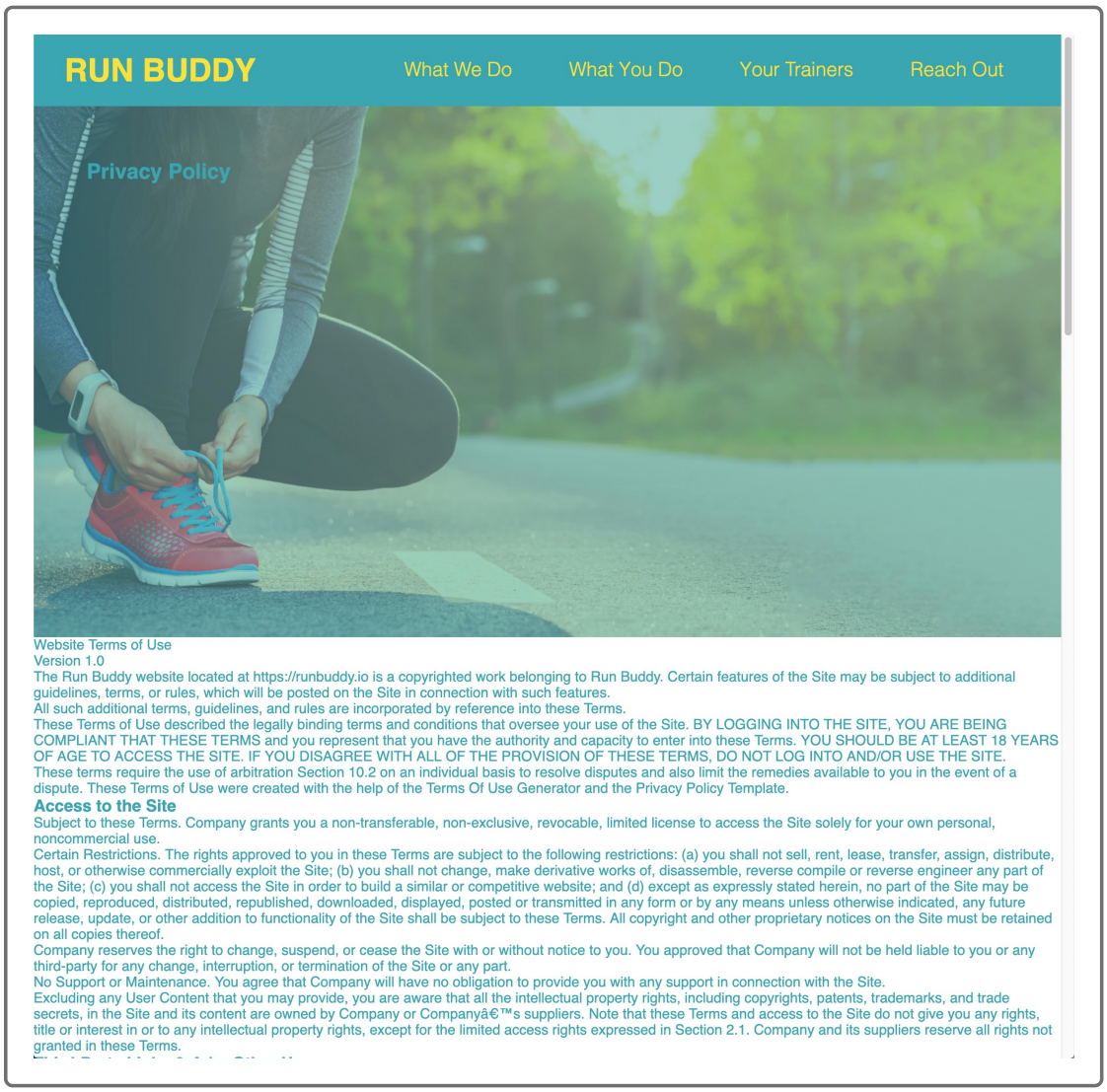
Now because the `<head>` tag of our HTML document has the exact same `<link>` tag as `index.html`, the styles from one CSS file are now being applied to both pages!

HIDE HINT

If the above didn't work, make sure you saved the file and refreshed the page in the browser.

This is great news, as we now don't have to repeat our style definitions and can reuse them wherever we need to. This is also one of the main reasons why in Lesson 2 we created a separate CSS file for our style definitions instead of including them in `index.html` itself.

There are some styles that are a little out of sorts, however. The `<header>` and `<footer>` styles look great, but the hero section needs to be fixed and the page's content in the `<article>` tag doesn't have any styling at all, as you can see in the below image:



Both of these issues are easy to solve. Looking back at the mock-up at the top of this lesson, you can see that there are only a few small differences between the appearance of the Run Buddy homepage and the Privacy Policy page.

To fix the styling of the Privacy Policy page, you'll need to do two things:

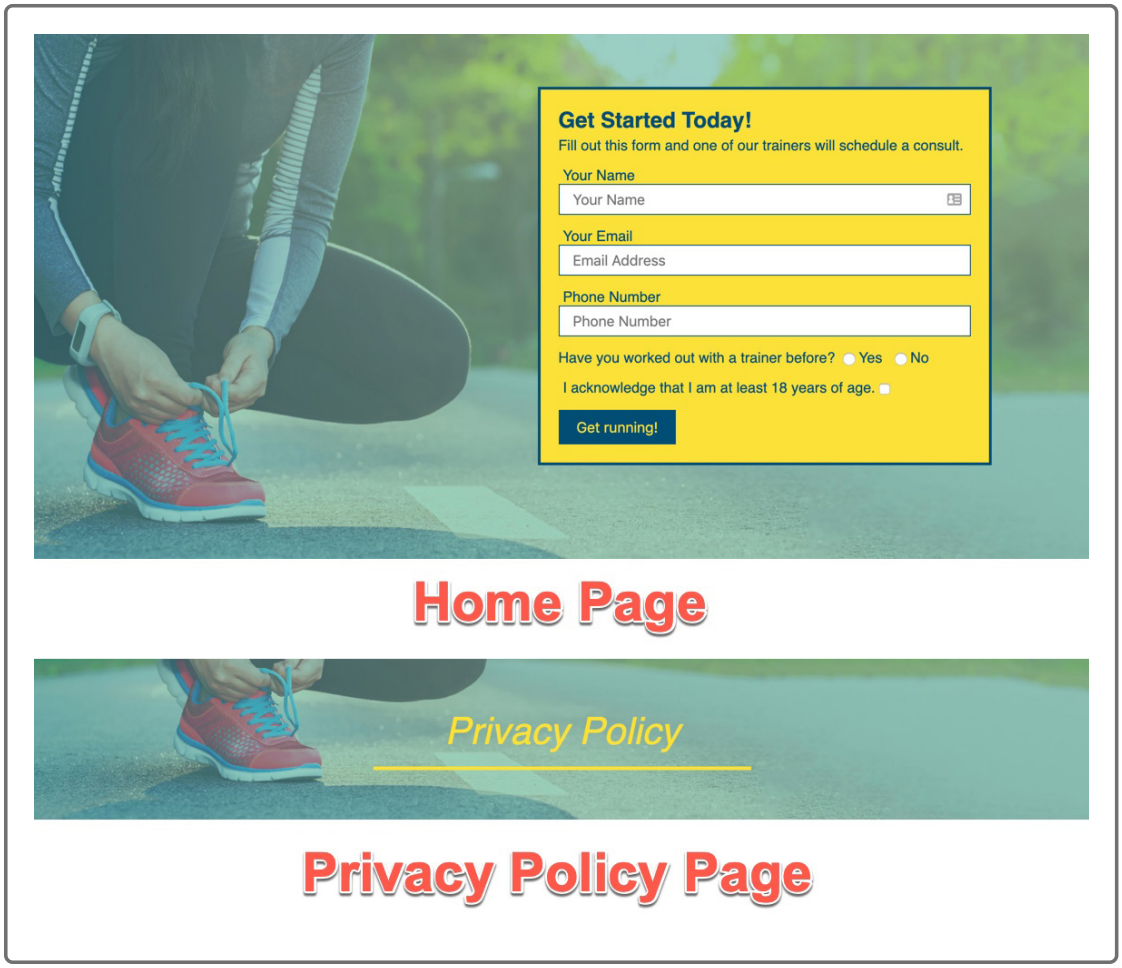
1. Keep most of the existing styles for the hero section and make just a couple of edits to them.

2. Add some new style definitions for the page's `<article>` tag by targeting its `secondary-content` class.

Let's discuss the first point. If you compare the finished product of `index.html` with the screenshot of the finished product we're working

towards for `privacy-policy.html`, you'll notice that the hero sections have the same background but the privacy policy page has a centered title and is much shorter.

Here's an image to see how our landing page's hero section looks compared to what we'll have in our privacy policy hero section:



To adjust the hero section of the Privacy Policy page, we'll edit some styles and have them only apply to the privacy policy's hero. There are a couple of ways to do this:

- **Change the class names**: In this approach, we reuse most of the current CSS rule blocks, make only a few edits to the style declarations, and give the CSS selector a new class name. This way, we can target different sections but still have a similar look. An

example of this would be if we wanted both paths to have the same margins and font sizes but different colors:

```
/* homepage hero */
.hero {
  margin: 20px;
  background-color: white;
  font-size: 20px;
  color: blue;
}

/* secondary page hero */
.hero-secondary {
  margin: 20px;
  background-color: black;
  font-size: 20px;
  color: red;
}
```

- **Keep the class names, but override some of the declarations**: This choice seems to be the most efficient with less duplicate code being written, because most of the declarations will be the same except for a few changes. In the example below, an HTML element gets most of its styles from one class but then has a couple of them overridden by a second one:

```
<div class="hero hero-secondary"></div>
```

```
.hero {
  margin: 20px;
  font-size: 20px;
  background-color: white;
  color: blue;
}

/* these two colors will override the ones above in .hero */
.hero-secondary {
```

```
  background-color: black;
  color: red;
}
```

Overriding CSS declarations can be tricky. Take care not to accidentally reassign working declarations in the hero section.

The way to avoid this is to create a second CSS file and have it accessible only to the `privacy-policy.html` file. You can do this by placing the secondary style sheet link in `privacy-policy.html` and not `index.html`. This is also useful because the folks at Run Buddy are currently reviewing the work you did on the landing page, so making edits to `style.css` at this moment could break things during the review, and an unhappy client makes for an unhappy developer.

IMPORTANT

> Not every HTML page needs a unique style sheet attached to it. In many cases, an entire site uses a single style sheet. Having separate style sheets can be useful when there are too many styles to keep organized in a single sheet and/or when multiple CSS developers are handling different parts of the site and don't want to step on each other's toes.

As with most problems you will face in programming, there will usually be a number of possible solutions. You might not always lean towards one over the other, and the option you choose should always depend on the problem at hand. The key is to not get overwhelmed by these decisions. You can always go back and change it if you don't like how it turns out (one of the many reasons we use Git!), and you'll never know which solutions you like until you try them.

For the sake of seeing how it works, let's go with the second option. The first one is a more than acceptable solution, but the second lets us see two style sheets in action.