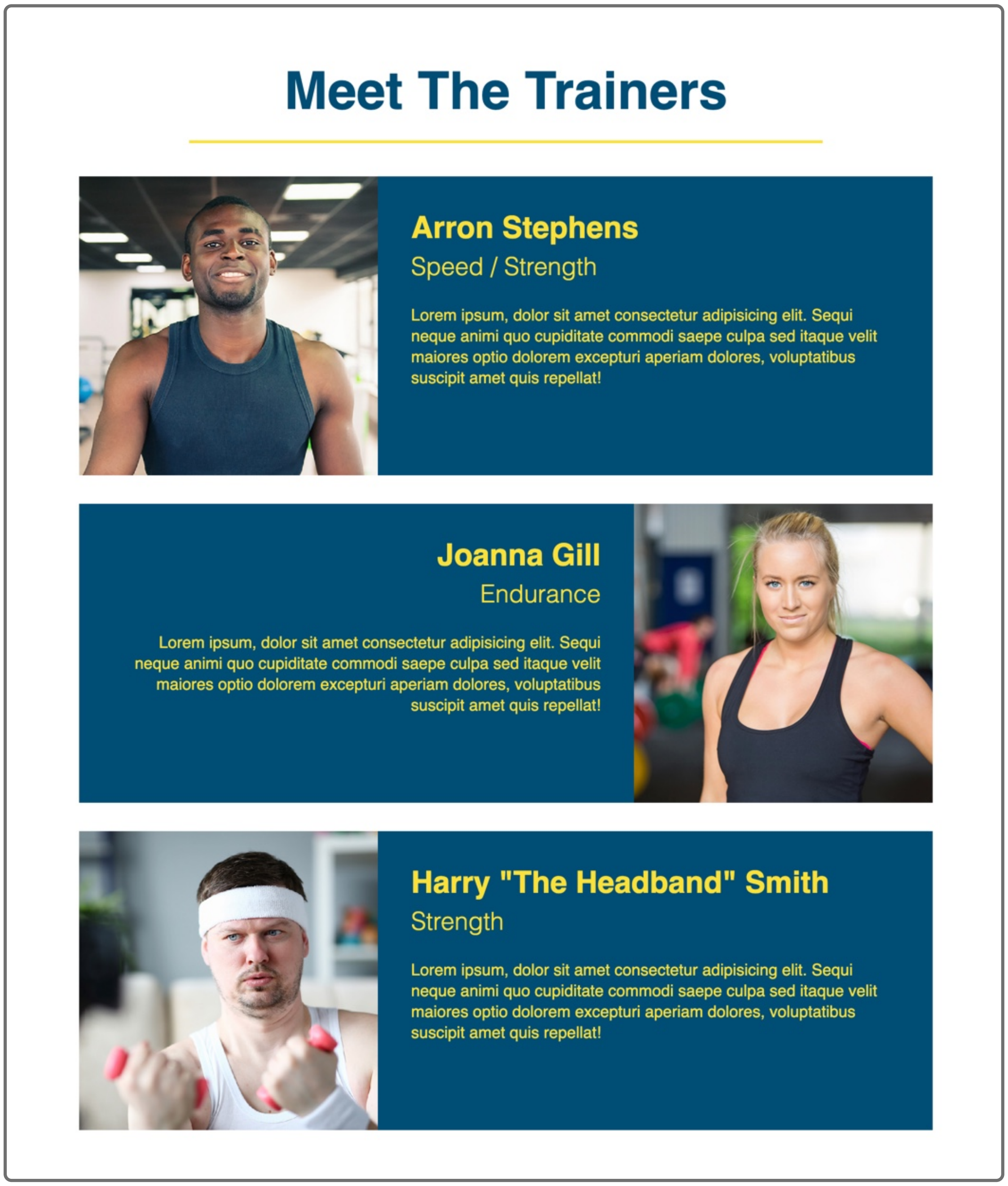
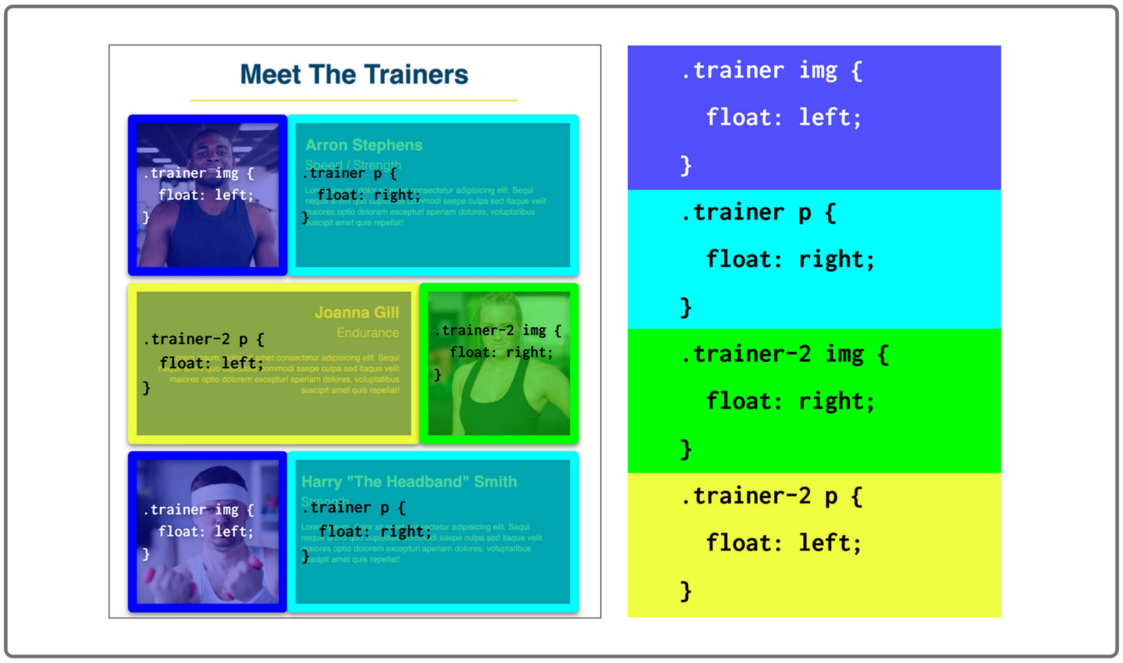
1.5.5 Add CSS to the First Trainer

As we just mentioned, we have everything we need to begin applying styles to this section. And we have everything in place so when we add the next two trainers' HTML content, the styles will automatically be applied to them and we can see it happening as we go.

In this section, we'll bring back `float`, something we used in the `<header>` and `<footer>` elements to render HTML block elements side by side. Based on the finished mock-up below, which elements do we think we'll need to `float`?


Meet The Trainers
Arron Stephens
Speed / Strength
Lorem ipsum, dolor sit amet consectetur adipisicing elit. Sequi neque animi quo cupiditate commodi saepe culpa sed itaque velit maiores optio dolorem excepturi aperiam dolores, voluptatibus suscipit amet quis repellat!
Joanna Gill
Endurance
Lorem ipsum, dolor sit amet consectetur adipisicing elit. Sequi neque animi quo cupiditate commodi saepe culpa sed itaque velit maiores optio dolorem excepturi aperiam dolores, voluptatibus suscipit amet quis repellat!
Harry "The Headband" Smith
Strength
Lorem ipsum, dolor sit amet consectetur adipisicing elit. Sequi neque animi quo cupiditate commodi saepe culpa sed itaque velit maiores optio dolorem excepturi aperiam dolores, voluptatibus suscipit amet quis repellat!

The answer is somewhat of a trick. We could get away with floating the first and third images left and the text content `<div>` to the right and that would work fine. Then for the second one we'd just need to swap the two, floating the image right and the text content `<div>` left. This image illustrates this strategy:



```
.trainer img {
  float: left;
}
.trainer p {
  float: right;
}
.trainer-2 img {
  float: right;
}
.trainer-2 p {
  float: left;
}
```

Again, this is fine, but it also involves writing more code than we need to.

Instead, we'll float both pieces to the left for all three trainers and apply a specific `width` value to them so they will all come nicely side-by-side with one another. Be warned—some interesting "gotcha" issues will come up in this section, but we'll get through them just fine!

Let's start by adding styles to the entire `<section>` tag by selecting it by its class name, `trainers`. In `style.css`, add the following code:

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

By adding that to the entire `<section>` element with a class attribute of `"trainers"`, our `<h2>` element that says "Meet the Trainers" should now be centered. This gives us a good start, but now our trainer's content is also centered. This isn't a problem though, as we will be fixing that soon.

Before we fix it, let's turn our attention to adding styles to the `<article>` element for our trainer. Add the following CSS rule to your `style.css`:

```
.trainer {
  width: 900px;
  margin: 0 auto 30px auto;
  background: #024e76;
  color: #fce138;
}
```

So now we've added some CSS styles to the entire `<article>` tag with a class of `trainer`. Let's review what we've done.

We don't want the content to span the full width of the page because it will run a little too long, so we've given it a set width value of 900px.

By using `auto` as the values for the left and right margin, we're telling the browser to take whatever space is unused and evenly distribute it on those sides, centering the element. An example of this would be if we had a screen that was 1000 pixels wide and an element that was only 800 pixels wide, the unused space would be 200 pixels; and if both left and right margins were set to `auto`, each side would get 100 pixels applied to it.

Lastly, we provided some spacing below it so the next trainers don't run up against one another.

IMPORTANT

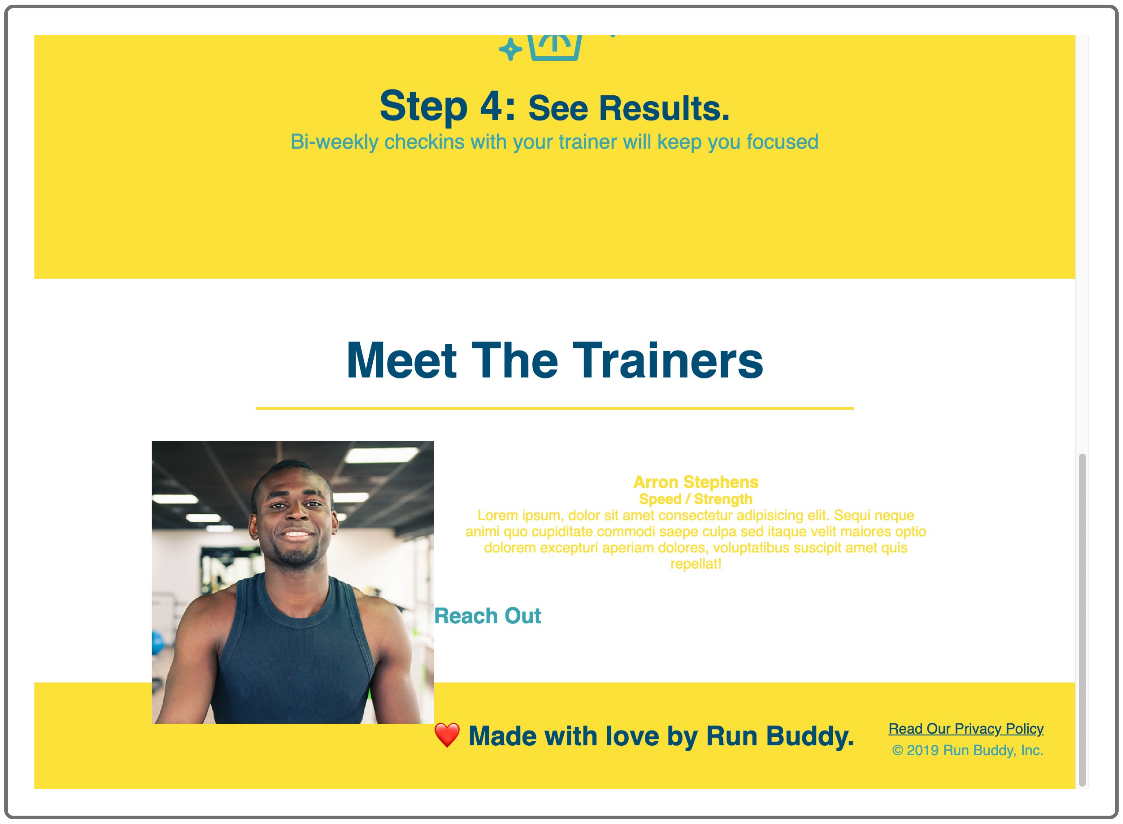
> Remember, the order for labeling `margin` and `padding` values is clockwise (top, right, bottom, left). So when you see:
>
> ```
> margin: 0 auto 30px auto;
> ```
>
> You can assume this is saying `"margin-top: 0"`, `"margin-right: auto"`, `"margin-bottom: 30px"`, and `"margin-left: auto"`.

Our trainer's container is ready to go, so now let's add some of the styles for the text and image content. Add the following to `style.css`, save the file, and refresh the page to see what happens:

```
.trainer img {
  width: 35%;
  float: left;
}

.trainer-bio {
  padding: 35px;
  float: left;
  width: 65%;
}
```

Okay, so don't panic. Those two style additions just set the whole page on fire, as this image shows us:


Step 4: See Results.
Bi-weekly checkins with your trainer will keep you focused

Meet The Trainers

Arron Stephens
Speed / Strength
Lorem ipsum, dolor sit amet consectetur adipisicing elit. Sequi neque animi quo cupiditate commodi saepe culpa sed itaque velit maiores optio dolorem excepturi aperiam dolores, voluptatibus suscipit amet quis repellat!

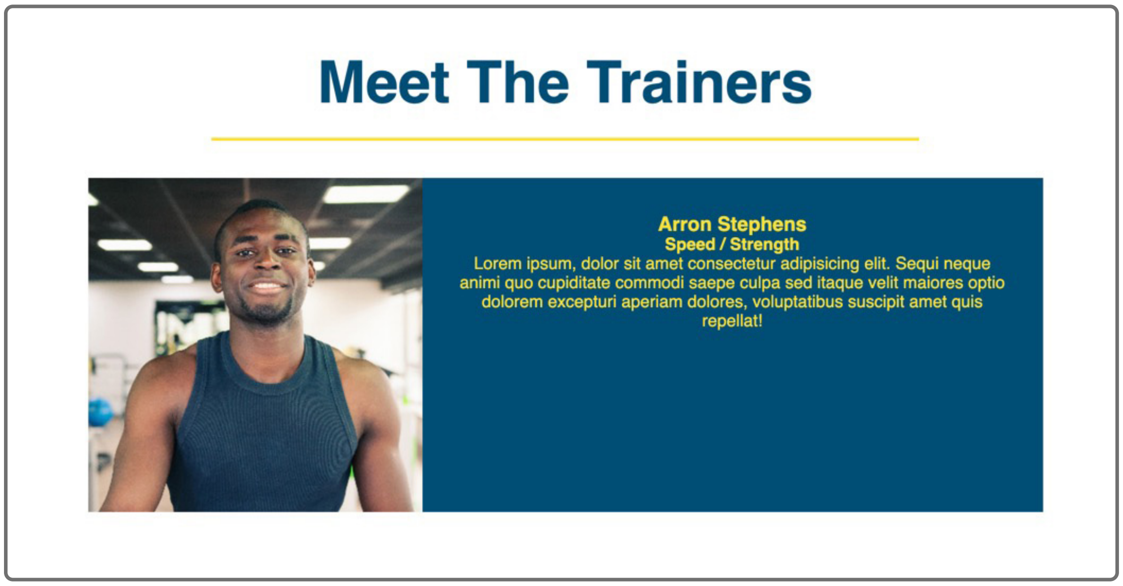
Reach Out

❤️ Made with love by Run Buddy.

Read Our Privacy Policy
© 2019 Run Buddy, Inc.

Before we explain what happened, let's make ourselves feel better and fix this mess. Go ahead and add the following property declaration to the `.trainer` styles: `overflow: auto`.

After saving the file and refreshing, the page should look like this image:


Meet The Trainers
Arron Stephens
Speed / Strength
Lorem ipsum, dolor sit amet consectetur adipisicing elit. Sequi neque animi quo cupiditate commodi saepe culpa sed itaque velit maiores optio dolorem excepturi aperiam dolores, voluptatibus suscipit amet quis repellat!

Much better! Now that we can relax a second, let's go over what just happened.

We worked with floats and manipulating a page's default "flow" in Lesson 2, but in those cases we only needed to float one of the two elements. When we floated the `<nav>` element inside the `<header>`, the `<header>` lost its ability to interpret how much room the `<nav>` element needed. This is okay, though, because the `<header>` still gets to interpret how much space the `<h1>` element needed because it wasn't floated and the `<header>` displays at the right size.

In this case, however, we floated both elements (the `<img>` and `<div>`), which means its parent element—the `<article>` element—cannot interpret how much space its inner HTML content needs and assumes there's nothing inside it at all. This means the `<article>` element's styles, like `background-color`, don't show up because the `<article>` element itself is 0 pixels tall.

This a common issue for float-based layouts. It involves not only moving the elements we want to move, but also tweaking elements around it to tell it to understand that there may be some floated elements it needs to account for. There are a few ways to make these tweaks to fix the problem we just had—we chose the `overflow` property.

DEEP DIVE ▲

DEEP DIVE

This is just one use of the `overflow` property and it will come up again in different use cases. For more information, read the [MDN web docs on the overflow property (https://developer.mozilla.org/en-US/docs/Web/CSS/overflow)](https://developer.mozilla.org/en-US/docs/Web/CSS/overflow).

The `overflow` fix we applied told the trainer's `<article>` element that it does in fact have content inside of it and that it needs to look for it and account for those two floated HTML elements' sizes. This is what's known as providing **block formatting context** to the element.

DEEP DIVE ▲

DEEP DIVE

To learn more, read the [MDN web docs on the block formatting context (https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Block_formatting_context#Examples)](https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Block_formatting_context#Examples).

We'll fix this by making sure the first and third trainer `<article>` elements have left-aligned text and the second trainer `<article>` element has right-aligned text. Rather than style each one separately, we'll take an approach that makes our code less repetitive.

We'll start by making two class style definitions that aren't unique to our trainers, but more of a utility class (think about how we created `primary-border` and `secondary-border`) that can be applied anywhere in our page to fix a style.

Go ahead and add the following two classes to `style.css`:

```
.text-left {
  text-align: left;
}

.text-right {
  text-align: right;
}
```

Notice how these classes have generic names that do not indicate exactly where they belong on the page. This is done on purpose so that they can applied to any HTML tag that needs a quick fix to their `text-align` properties, rather than having to create more style rules for each one individually.

DEEP DIVE

DEEP DIVE

CSS classes that are not specific to any part of an HTML document and that do only one thing are what's known as a **utility classes**. The idea behind them is instead of creating very specific classes that only fit the needs of a couple of HTML elements, you create more CSS classes that apply one property declaration and are applied by adding multiple classes to an HTML element instead.

To learn more, read this **article about CSS utility classes (https://blog.mariano.io/css-utility-classes-how-to-use-them-effectively-d61ee00dad2d)** .

Let's revisit our HTML and add the class `text-left` to the `<div class="trainer-bio">` tag. It should look like this when it's done:

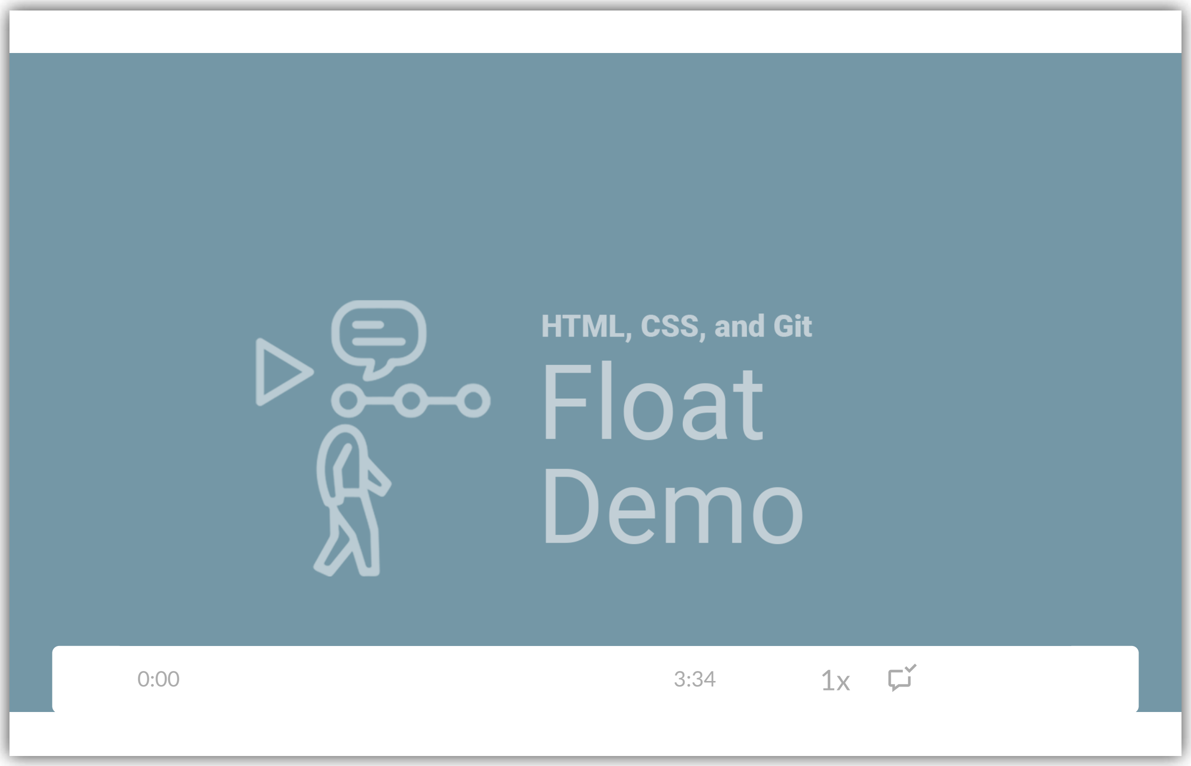
```
<div class="trainer-bio text-left">
  <h3>Arron Stephens</h3>
  <h4>Speed / Strength</h4>
  <p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Sequi r
</div>
```

Now the trainer's `<article>` element text is left-aligned because we added another class, `text-left`, to it. This class was created for the sole purpose of left-aligning text when needed.

When we add the next two trainers' `<article>` elements, we will also be applying `text-left` and `text-right` to them just as we did with this first one.

Let's add some styles that are more specific to our trainers' name heading, subheading, and paragraph text. In `style.css`, add the following code:

As it was mentioned, there are a few ways for dealing with complications floated HTML elements can bring about. We just used the `overflow` property to handle it, but there's another one we can use called `clear`. This video will walk through a similar situation, but use the `clear` property to fix the issue instead:



Style the Text Content

Now that we've given the `<article>` element that holds our trainer's information a nicer layout, let's get to applying CSS styles to the text content. As you can see, all of the text is currently center-aligned. This is because we set the `trainers` class in the parent `<section>` to be center-aligned, and the style is being applied to the child elements.

NERD NOTE

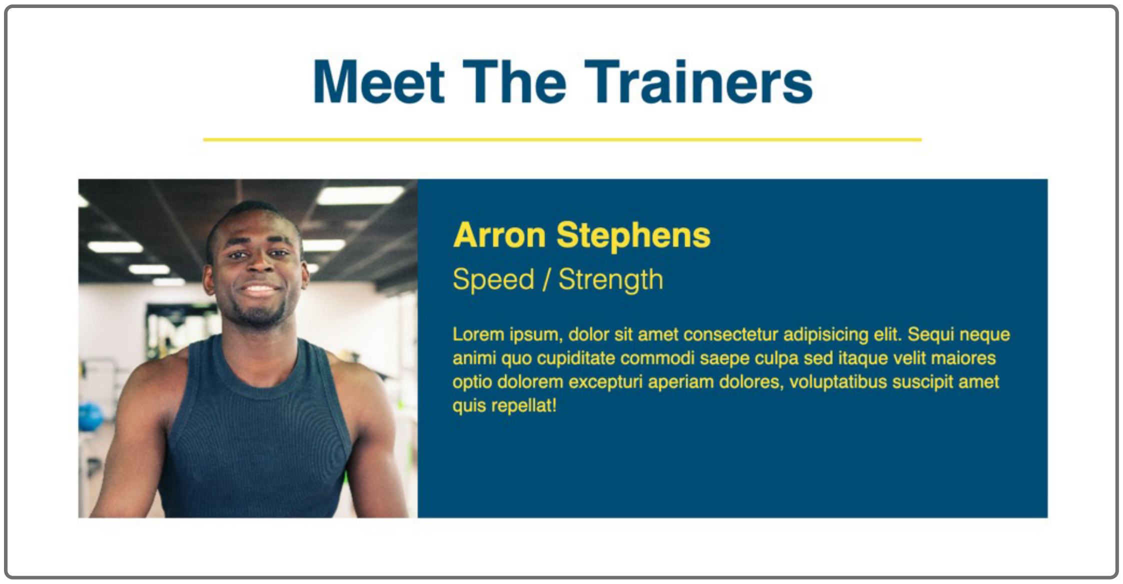
When a parent HTML element's style is applied to any child element, it is called **inheritance**.

```
.trainer-bio h3 {
  font-size: 32px;
  margin-bottom: 8px;
}

.trainer-bio h4 {
  font-weight: lighter;
  font-size: 26px;
  margin-bottom: 25px;
}

.trainer-bio p {
  font-size: 17px;
  line-height: 1.3;
}
```

And there we go! Our trainer's styles are now set up and everything should be looking good, just as this image shows us:


Meet The Trainers
Arron Stephens
Speed / Strength
Lorem ipsum, dolor sit amet consectetur adipisicing elit. Sequi neque animi quo cupiditate commodi saepe culpa sed itaque velit maiores optio dolorem excepturi aperiam dolores, voluptatibus suscipit amet quis repellat!

IMPORTANT

> For some users—especially Windows users—the font we're using here, Helvetica, might be replaced with Arial. This is because most Windows computers do not come with Helvetica installed on them. Helvetica and Arial look similar, but there are differences between the two, namely the

> variations of the `font-weight` property. Arial does not have a weight for the value `lighter`, so it uses the closest weight (in this case, `normal`).
>
> This is something developers deal with on a regular basis, but the nice thing is that the fall-back weight that's provided doesn't break the page. It will just look a little different than it does for MacOS users.
>
> To learn more, read the **MDN web docs on the font-weight property (https://developer.mozilla.org/en-US/docs/Web/CSS/font-weight)** .

Let's go ahead and add the other two trainers to the page. Then we'll be done with this section!