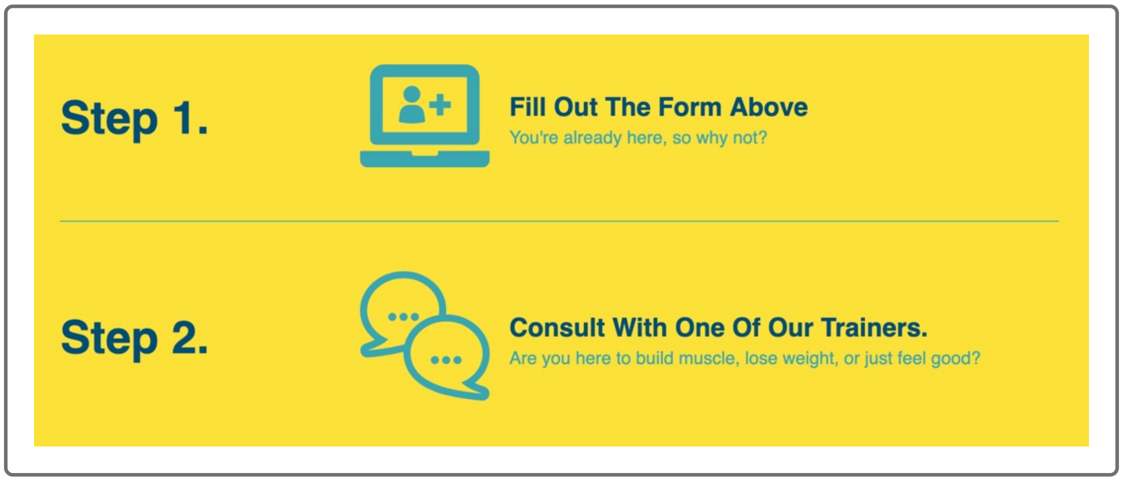
2.2.9 Nesting and Flexing

Part of designing and building a webpage is understanding how to make good use of the horizontal space so the page does not run too long vertically. The idea behind this is that some users might be too impatient to scroll down to get to all the information. Some designers try to over-correct this issue by cramming as much up top as possible, but that can make the page top-heavy and confusing to read.

We'll try to strike a nice balance by reworking the "What You Do" steps so that each step is still on its own row but has a left-to-right layout, like the following image shows:

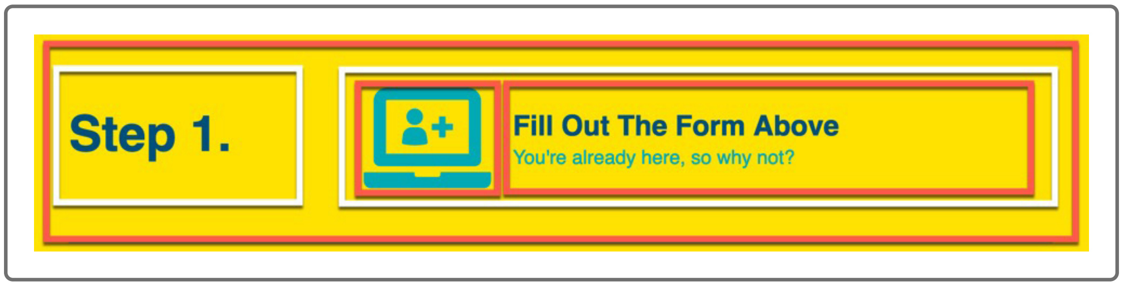

Step 1.
Fill Out The Form Above
You're already here, so why not?
Step 2.
Consult With One Of Our Trainers.
Are you here to build muscle, lose weight, or just feel good?

Each step will take up less vertical space and have a more interesting layout while still keeping the content together in a meaningful way.

Before we begin coding, let's consider how many containers we need.

SHOW HINT

Looking at the mock-up, we can see that we can make the entire step a flexbox with two children—the step's number on the left and information on the right. Then we'll make the step information's `<div>` its own flexbox with two children (for the icon and the text):



Like previous sections, we'll start by restructuring the HTML a bit and removing some CSS styles.

In `index.html`, go through each `<div>` element that contains the step information (e.g., "Step 1: Fill Out The Form Above") and change it to this:

```
<div class="step">
  <h3>Step 1.</h3>
  <div class="step-info">
    <div class="step-img">
      <img src="./assets/images/step-1.svg" alt="" />
    </div>
    <div class="step-text">
      <h4>Fill Out The Form Above</h4>
      <p>You're already here, so why not?</p>
    </div>
  </div>
</div>
```

SHOW HINT

Note how we added `class="step"` to each `<div>` element. We also created child containers for each piece of content in a step so now it will be easier to move the content around by adding style rules to the containers instead of the content itself.

SHOW PRO TIP

Let's move on to the next part of our code cleanup and remove any existing CSS styles that might conflict with our new layout:

1. Remove the entire CSS rule for `.steps div`, `.steps img`, and `.steps span`.
2. Remove the `margin-top` declaration from `.steps h3` and rename the selector to `.step h3` to tighten the relationship of the selector and make it a bit quicker for the browser to set the rule.

IMPORTANT

The browser reads CSS selectors from right to left (or innermost element to outermost element). Given a selector of `header nav ul li a`, we can think of it as the browser saying to itself "Find every `<a>` element that's inside an `<li>` element, but the `<li>` element has to be in a `<ul>` element, and the `<ul>` element has to be in a `<nav>` element,

> and so on." This means that the browser has to do a few pass-throughs and work its way up the HTML structure to ensure the CSS styles are only being applied to those distinct `a` elements.
>
> While the selector `header nav ul li a` is fine, having a whole page of that may prove to be not as performant as the page gets larger. A way to improve this is to provide a class to the elements you want to style and select just that class. But that could prove tedious and end up creating unnecessary amounts of extra HTML by adding a class attribute to every element.
>
> Situations like this are common. There will likely be more than one way to solve every problem you come across and it will be up to you to decide which route is best. In this case, it's a choice between making the code slightly more performant versus slightly easier to read. The answer usually lies somewhere in the middle with something like `.nav-class a`, which is very easy to read and understand but is also specific enough for the browser to read efficiently.

While we're at it, let's re-style the text content elements:

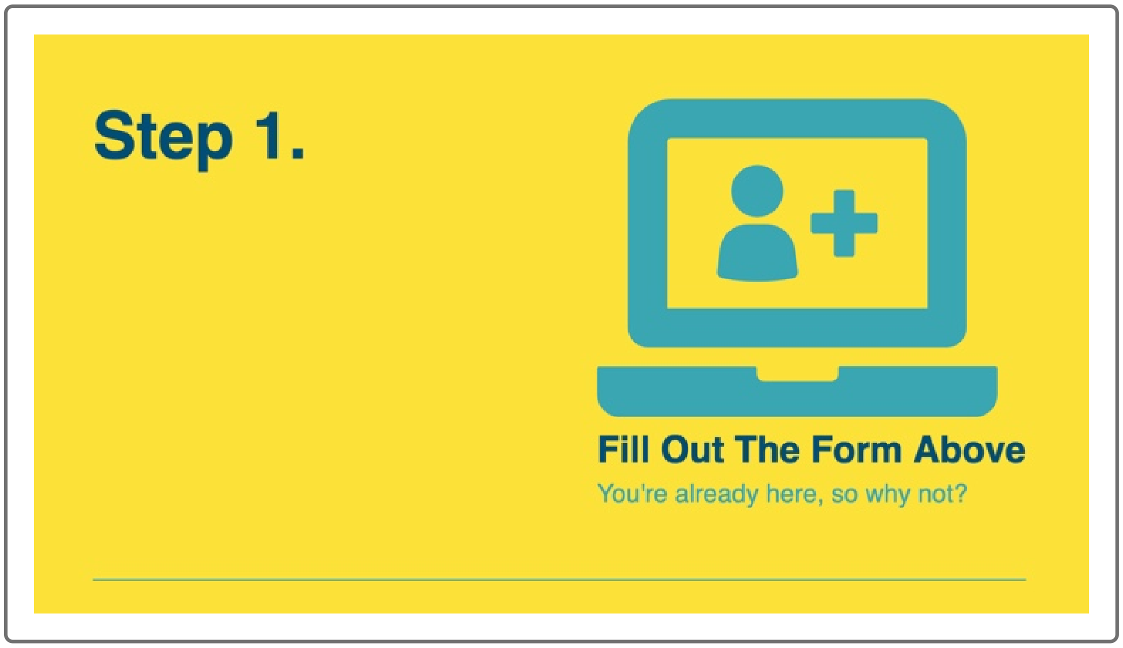
1. Change the `.steps p` selector to say `.step-text p` and change its `font-size` to be 18px instead of 23px
2. Create a new CSS style rule for the `h4` element as such:

```
.step-text h4 {
  font-size: 26px;
  line-height: 1.5;
  color: #024e76;
}
```

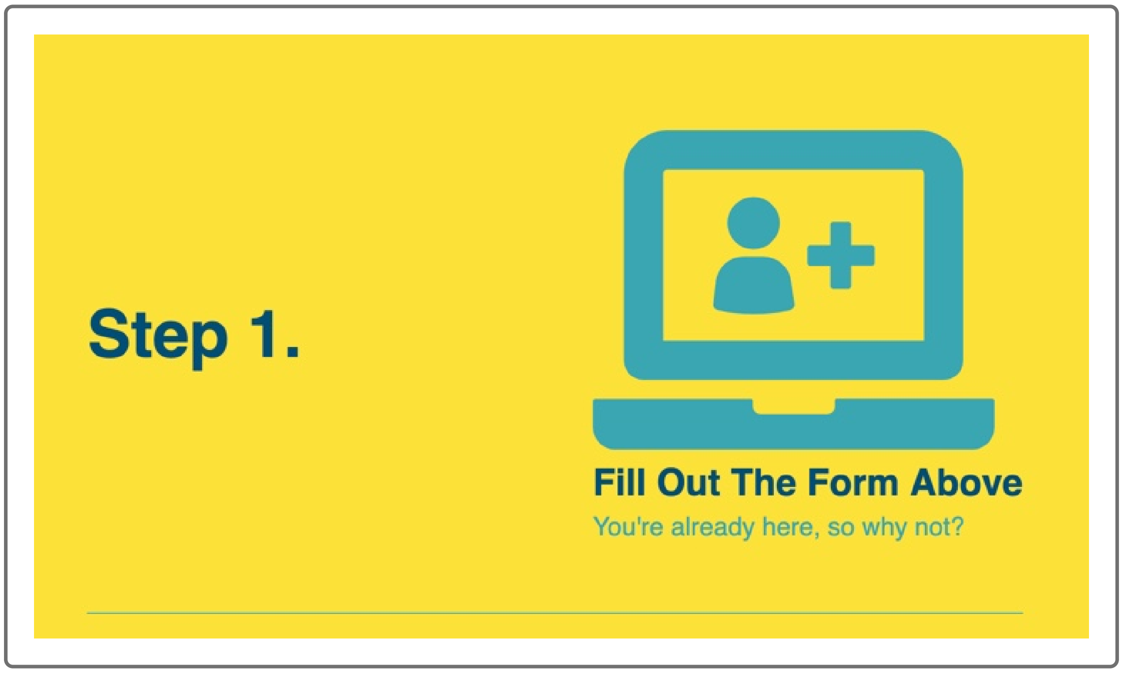
Now that the HTML is ready and the CSS has been cleaned up, we can start adding our flexbox styles to each step. Let's start with the step as a whole by creating a CSS rule for anything with a class of `.step`:

```
.step {
  margin: 50px auto;
  padding-bottom: 50px;
  width: 80%;
  border-bottom: 1px solid #39a6b2;
  display: flex;
  flex-wrap: wrap;
  align-items: center;
  justify-content: space-between;
}
```

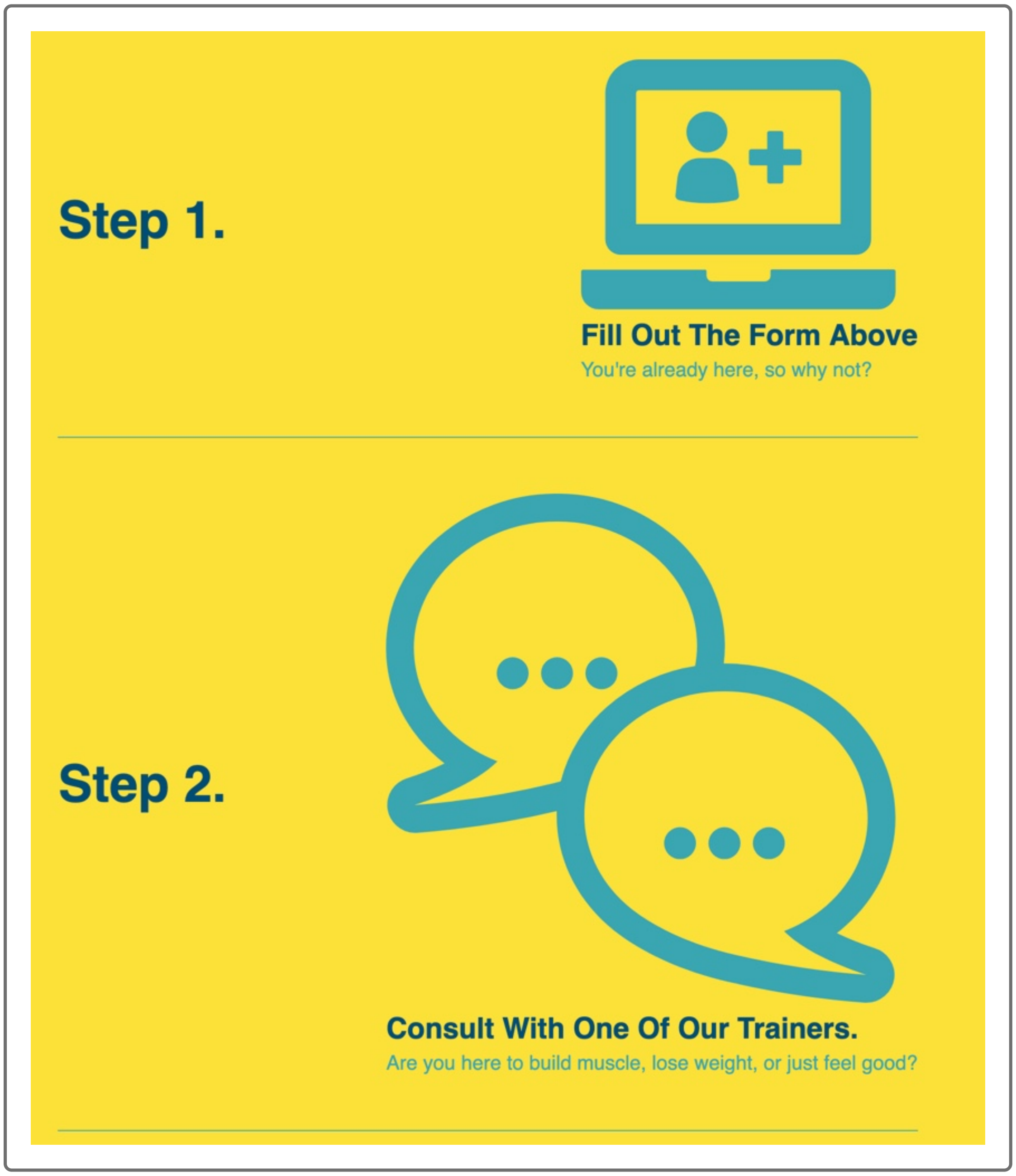
We've seen and used all of these properties a fair amount at this point, but this will be the first time we see the `align-items` property make a difference. If we didn't tell the two children to be center-aligned, the finished product would look like this:



By centering them, they look like this:



We aren't quite there yet for the `.step` flexbox, as our current layout doesn't match the mock-up. Right now both children of the `.step` flexbox (`<h3>` and `<div class="step-info">`) share a horizontal line, but their widths vary based on the length of the content inside them. As we can see in this image, the content for Step 1 is less than Step 2, so the box holding that content is narrower:


Step 1.
Fill Out The Form Above
You're already here, so why not?
Step 2.
Consult With One Of Our Trainers.
Are you here to build muscle, lose weight, or just feel good?

Because we can't rely on the content being uniform enough to provide a consistent layout, we need to tell the containers they're in to have some rules for how big they can be.

Use the Flex Property for Widths

In previous sections, we used the `width` property to give flexbox child elements dimensions and rules to follow. This would work here, but instead we'll use the `flex` property to help the steps scale with the page:

- To `.step h3`, add `flex: 1 30%;`
- Create a new rule for `.step-info` and add `flex: 2 70%;`

The `flex` property can be challenging to understand. It's used when we need to apply more specific instructions to how flexbox children should be displayed on the page in relation to sibling elements. It accepts up to three values in its declaration because it rolls up the following three `flex` properties into one:

- **`flex-grow`**: A numeric value that's used to determine how much of the flexbox's unused space can be spread out to its children. The number provided is used as a ratio compared to the other child's `flex-grow` value. The higher the number, the more unused space that child will be given. To learn more, see the **MDN web docs on the flex-grow property (https://developer.mozilla.org/en-US/docs/Web/CSS/flex-grow)**.

- **`flex-shrink`**: This is used to determine how to size the flexbox's children when the flexbox container shrinks. This property is cool, but it isn't used as much as `flex-grow`. To learn more, read the **MDN web docs on the flex-shrink property (https://developer.mozilla.org/en-US/docs/Web/CSS/flex-shrink)**.

- **`flex-basis`**: This works similar to setting a `width` value for a child element, but it is used more as a baseline value that at the very least will let the child be that size no matter what and grow or shrink accordingly. To learn more, see the **MDN web docs on the flex-basis property (https://developer.mozilla.org/en-US/docs/Web/CSS/flex-basis)**.

There's no benefit to using `flex` instead of listing these properties separately except that it saves a little space in the style sheet.

The `flex` property can read values in a few different ways, based on the type of value entered:

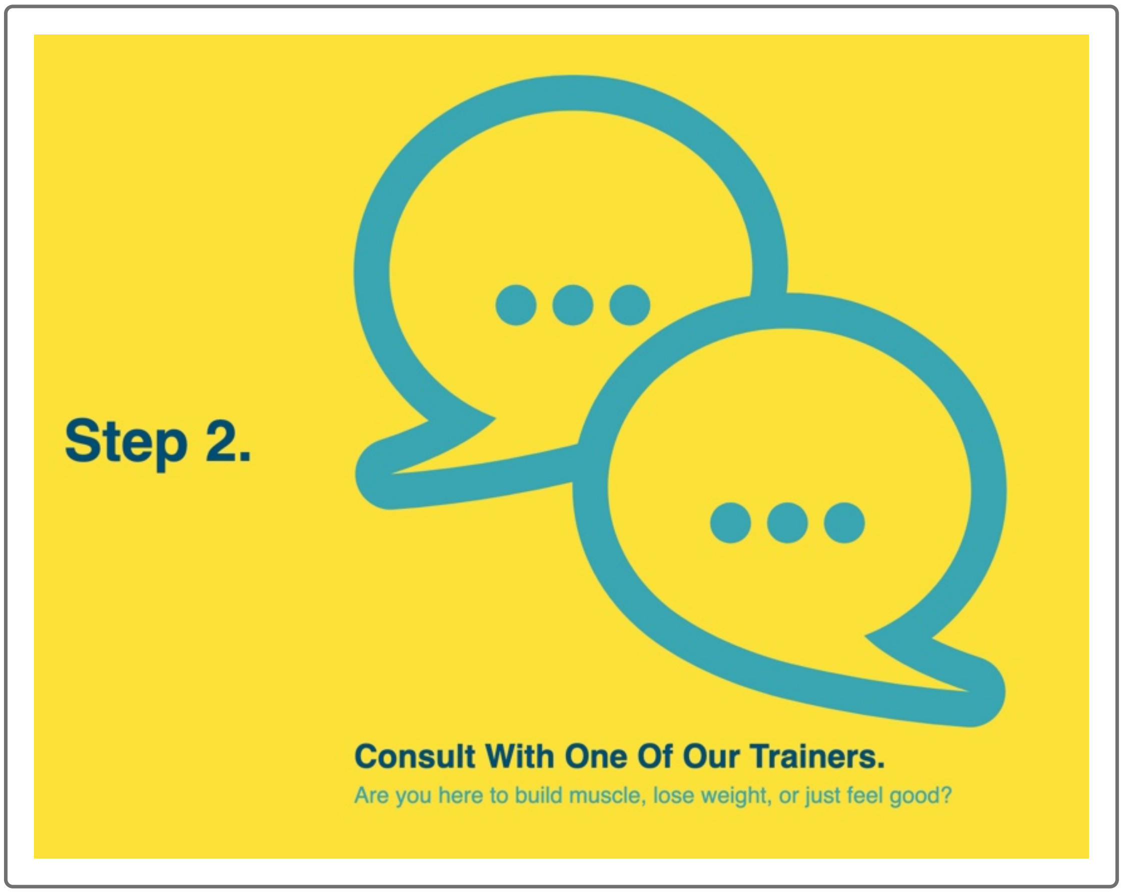
```
flex: <flex-grow value> <flex-shrink value> <flex-basis value>;

flex: <flex-grow value> <flex-basis value>;

flex: <flex-grow value> <flex-shrink value>;
```

DEEP DIVE

The value we provided to our step's flexbox children uses the `<flex-grow value> <flex-basis value>` syntax, meaning that:

- `.step h3` will be at least 30% wide and will receive any extra space in the container.
- `.step-info` will be at least 70% wide, but will receive any extra space in the container at a 2-to-1 ratio, meaning it will receive 2 more units of unused space than a sibling with a `flex-grow` value of 1.

Now we should have a step layout that looks like this image:


Step 2.
Consult With One Of Our Trainers.
Are you here to build muscle, lose weight, or just feel good?

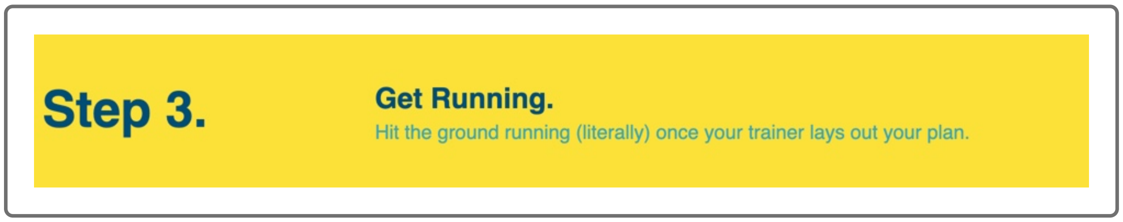
We're almost there. All we need to do now is take what we just learned and apply it to `.step-info`.

Start by adding these declarations to the `.step-info` rule:

- `display: flex;`
- `flex-wrap: wrap;`
- `align-items: center;`

Where did our icon go?! It's been totally removed from the screen for some reason. Think about how a flexbox container works if its children do not have any set `width` or `flex-basis` values. It tries to create space for both of them as evenly as possible based on the content of each child element, but what happens when one of the child element's content doesn't have a set dimension?

Take a look at the following image to see what this area looks like now:


Step 3.
Get Running.
Hit the ground running (literally) once your trainer lays out your plan.

As you can see, the `<img>` icon has no CSS rules for a width value. So when the flexbox parent (`<div class="step-info">`) attempts to calculate how much space each one of its children (`<div class="step-img">` and `<div class="step-text">`) needs, it will see that one of them has text content and the other has an image with no width explicitly set. Because of this, the flexbox parent cannot infer how big that image should be and ends up ignoring it, giving all of the space to the child element that has content.

To fix this, we need to give both children some type of value to give them their own space. Let's do that by creating a new CSS rule for `.step-img`, like this:

```
.step-img {
  flex: 1 12%;
  margin-right: 20px;
}
```

We're getting close, but it's creating some weird movement where steps with less text have a bigger image. This is because there's only one child with the `flex` property so it's ignoring the needs of the other child.

Let's fix that by creating another CSS rule for `.step-text` that looks like this:

```
.step-text {
  flex: 12;
}
```

Now both child elements have a `flex-grow` property of 1 and 12, respectively. This means that `.step-text` will be allotted 12 times more unused space than `.step-img`, but `.step-img` at the very least *must* be 12 percent of the width of `.step-info`.

Why were these values chosen? A 12-to-1 seems like a wide gap to give these two flexbox children. As we've seen before, these values can be settled on in a number of different ways, but it usually involves some form of trial and error using Chrome's DevTools until the desired layout is achieved.

Notice how we don't even need to worry about setting a `flex-basis` value for `.step-text`. When that value is omitted, the browser gives it a value of `auto`, which allows it to be whatever width is left over.

One last thing we need to do is tell the `<img>` element to limit its width to be whatever its container is. This isn't a problem with most browsers, but Microsoft Edge can be a little buggy with it, so it's always worth putting in a little extra to have a uniform design across browsers. Create this CSS rule for it:

```
.step-img img {
  max-width: 100%;
}
```

Again, we just fixed a problem that probably won't occur on any of our machines if we're using Google Chrome, but it's a good practice to stay ahead of any browser quirks. It's part of our job to prevent users from encountering issues just because they're on a different browser than we are.

SHOW PRO TIP

The rest of the page's conversion to flexbox won't be as involved as this section was, but it's cool to know that we can use this one tool to handle both simple and complex layouts.

Don't forget to save your progress with Git using the following commands:

```
git add .
git commit -m "add flexbox to the steps"
git push origin feature/flexbox
```