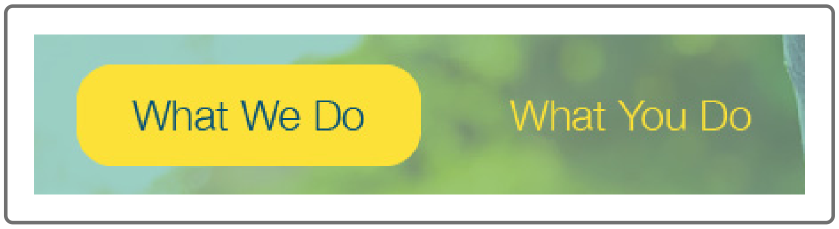
2.5.6 Style the States

Run Buddy has also requested that the page be more visually interactive. They'd like buttons and links to change when the user points the cursor at them and form elements to change when the user clicks on them.

For example, the "What We Do" link below shows what a visually interactive link could look like when the user moves the cursor over the link:



Complex interactivity requires JavaScript, which we haven't covered yet. But we can still fulfill Run Buddy's requests with CSS and pseudo-classes.

Pseudo-classes let us style different states of an HTML element. For example, can you think of some different states that a button might have? You can **hover** over a button. You can be **actively** clicking on a button. And when clicked, the button becomes the HTML element that's in **focus**.

Pseudo-classes can be added to any CSS selector using a colon and the name of the state, as shown here:

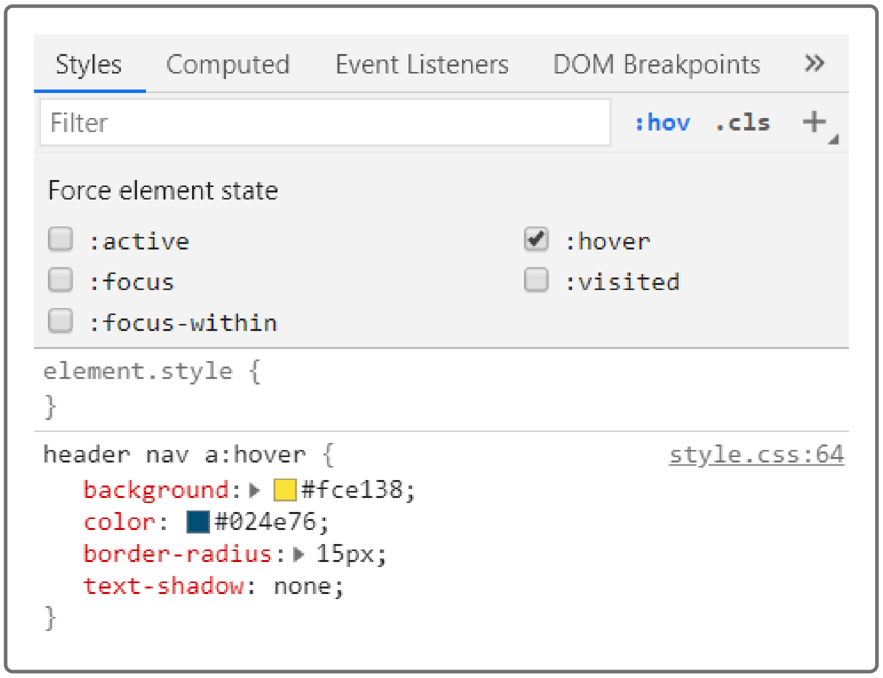
```
/* default state */
button {
  background: white;
}

/* mouse is hovering over button */
button:hover {
  background: red;
}

/* button is in focus because it was the most recently clicked element
button:focus {
  background: green;
}

/* user is actively pressing the cursor down on the button */
button:active {
  background: blue;
}
```

With this new knowledge, let's add some hover effects to the links and buttons on the landing page. To help keep the style sheet organized, add each of these new rules next to their non-hover counterparts:

```
header nav ul li a:hover {
  background: #fce138;
  color: #024e76;
  border-radius: 15px;
  text-shadow: none;
}

.hero-form button:hover {
  background-color: #39a6b2;
}

.contact-form button:hover {
  color: #fce138;
  background: #39a6b2;
}
```

HIDE PRO TIP

Chrome's DevTools lets you toggle these states on and off, making it easier to style and debug them:



Force element state

- [ ] :active
- [ ] :focus
- [ ] :focus-within
- [x] :hover
- [ ] :visited

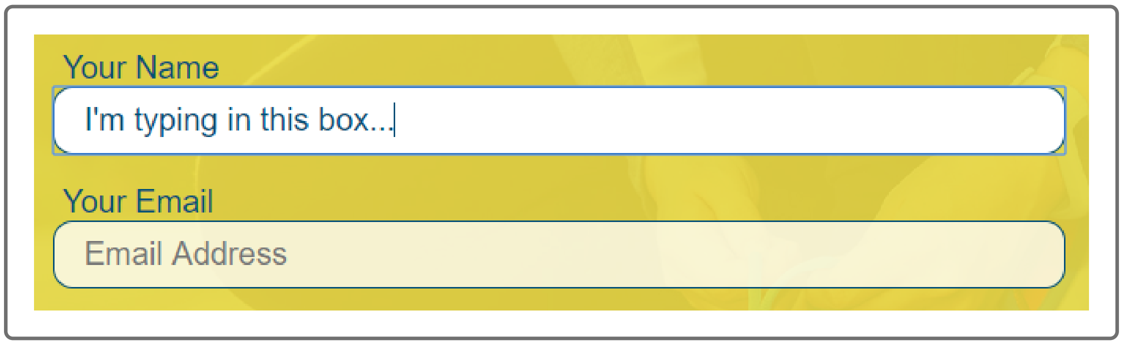
```
element.style {
}
header nav a:hover {                    style.css:64
  background: ▶ #fce138;
  color: #024e76;
  border-radius: ▶ 15px;
  text-shadow: none;
}
```

Using the `:focus` state, we can also visually highlight the form input that the user is currently typing in. Let's try this by adding the following rules to our style sheet:

```
/* update existing rule with this declaration */
.form-input {
  background-color: rgba(255,255,255, 0.75);
}
```

```
/* create a new rule for focus state */
.form-input:focus {
  background-color: rgba(255,255,255, 1);
}
```

This makes the form input's default state to be slightly transparent, then switches to full opaqueness when the input is in focus:

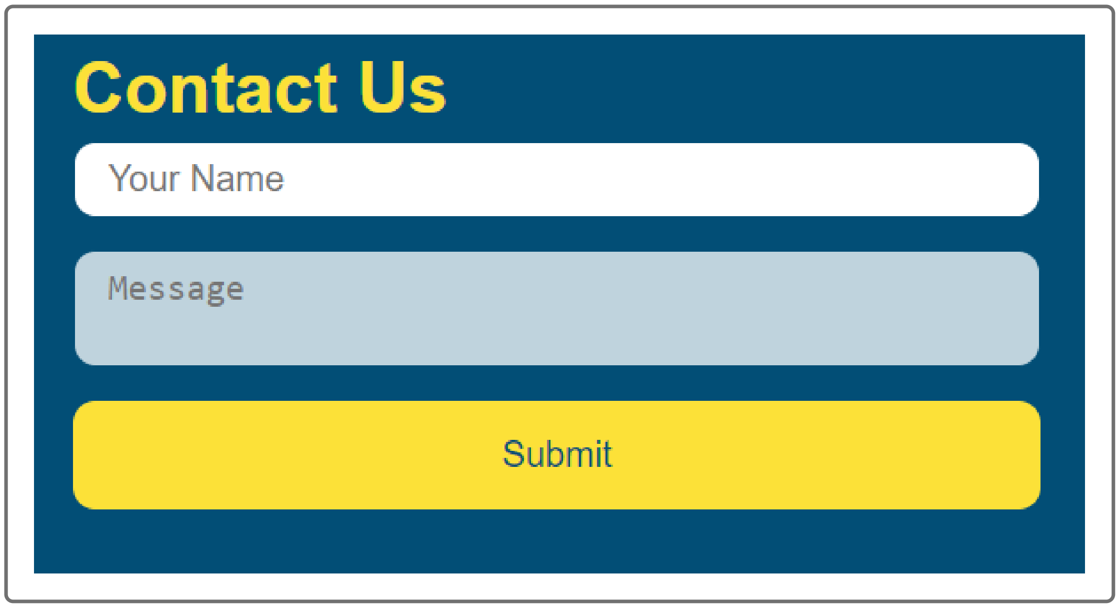


That looks pretty nifty, but it's not quite right yet. Take a closer look at the input that's in focus. There's an extra blue border around it, which looks terrible when paired with the rounded corners! This is the browser's default styling at play. So how do we fix it?

We can't add a `border-radius` to this outline, but we can remove it altogether by including a second declaration:

```
.form-input:focus {
  background-color: rgba(255,255,255, 1);
  outline: none;
}
```

That's much better! Let's do something similar with the form input and text area in the Reach Out section. Go ahead and do this on your own until the Contact Us form looks like this:



Remember to add a transparent default state and a separate `:focus` rule.

HIDE HINT

You can consolidate your `:focus` rules with comma-separated selectors: `.contact-form input:focus, .contact-form textarea:focus`.

Here's how this CSS could look:

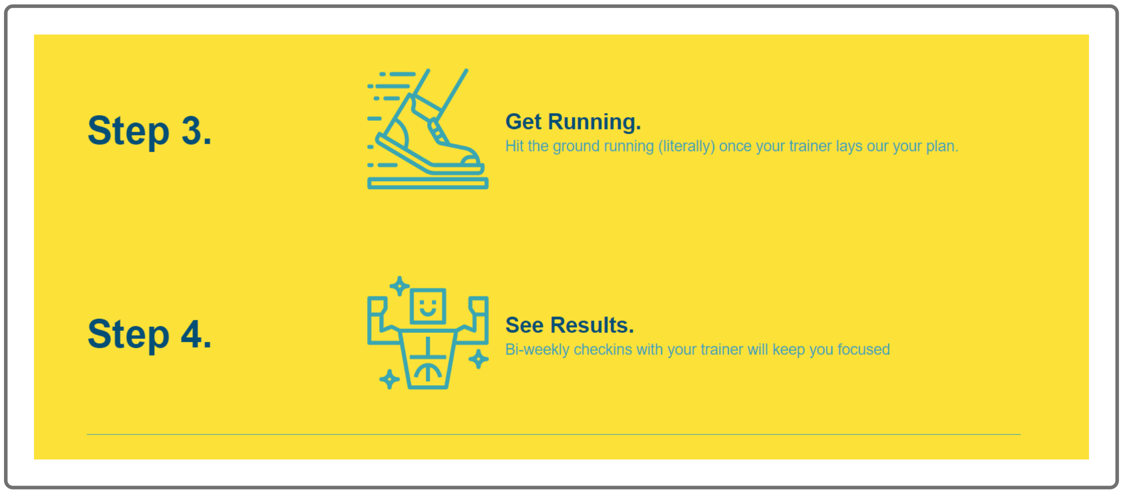
```
.contact-form input, .contact-form textarea {
  /* add a transparent background here */
}

/* create a new css rule for focus state */
.contact-form input:focus, .contact-form textarea:focus {

}
```

So far, we've looked at hover, active, and focus states that depend on user interactivity. There are also other states an element can be in that have more to do with where it is in relation to others.

Earlier, we added a `border-bottom: 1px solid #39a6b2` declaration to `.step`. Delete that declaration for now and add it to a brand new CSS rule instead:

```
.step:last-child {
  border-bottom: 1px solid #39a6b2;
}
```

Look at the page in the browser, and you'll notice that a border only appears under the last step now:


Step 3.
Get Running.
Hit the ground running (literally) once your trainer lays our your plan.
Step 4.
See Results.
Bi-weekly checkins with your trainer will keep you focused

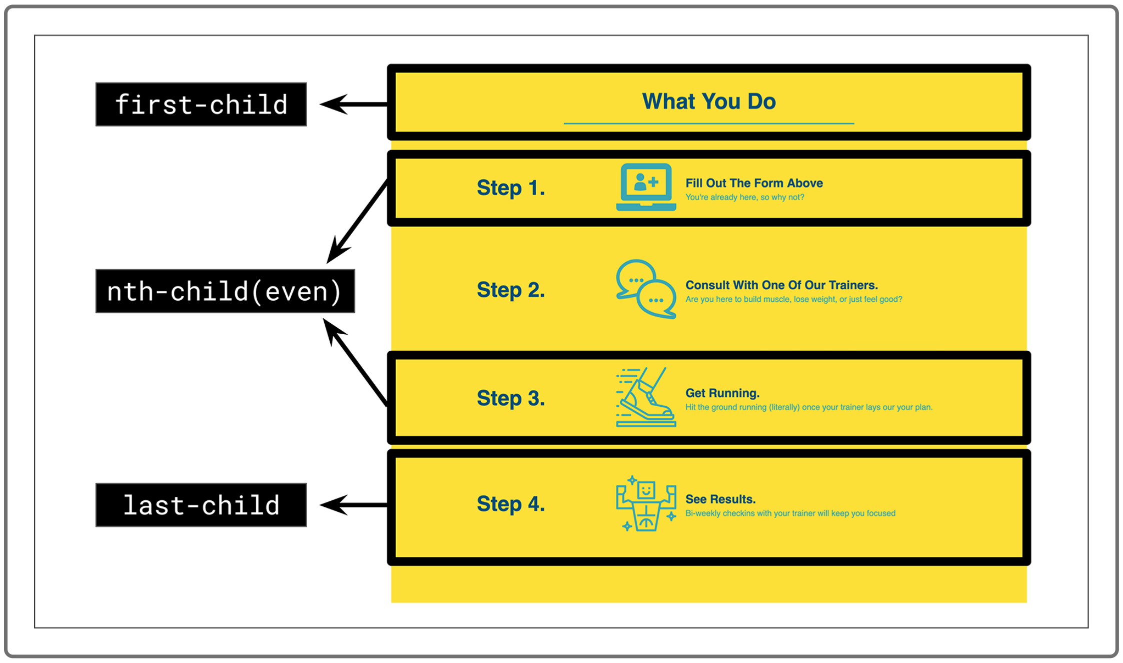
The `.step:last-child` selector means the CSS declarations will only apply to an element with class `step` that's also the last element in whatever container it's in. Let's revisit the HTML, where we have a `<section>` element with multiple child elements:

```
<section id="what-you-do" class="steps">
  <div class="flex-row">
    ...
  </div>
  <div class="step">
    <h3>Step 1.</h3>
    ...
  </div>
  <div class="step">
    <h3>Step 2.</h3>
    ...
  </div>
  <div class="step">
    <h3>Step 3.</h3>
    ...
  </div>
  <div class="step">
    <h3>Step 4.</h3>
    ...
  </div>
</section>
```

The `<div>` with "Step 4" in it is the last child of the `<section>` element. The `<div>` with class `flex-row` is the first child. This parent/child relationship lets us do some interesting things with pseudo-classes. Try this one instead, replacing the `.step:last-child` declaration with it:

```
.step:nth-child(even) {
  border-bottom: 1px solid #39a6b2;
}
```

This adds a border to every even-numbered child of the `<section>` that also has the class `step`. In this case, that's Step 1 and Step 3. That probably seems confusing, because Step 1 is not an even number! But keep in mind that `<div class="flex-row">` is the first official child, so Step 1 is actually the second child (an even number).


first-child
nth-child(even)
last-child
What You Do
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?
Step 3.
Get Running.
Hit the ground running (literally) once your trainer lays our your plan.
Step 4.
See Results.
Bi-weekly checkins with your trainer will keep you focused

For Run Buddy, we need a bottom border applied to every step except the last one. We could approach this in a few ways. One solution is to apply the border to all steps and then remove it from the last one:

```css
.step {
  /* re-add alongside existing declarations */
  border-bottom: 1px solid #39a6b2;
}

.step:last-child {
  border-bottom: none;
}
```

Or we could combine `:last-child` with another pseudo-class, `:not`, to check for the inverse of that being true:

```css
.step:not(:last-child) {
  border-bottom: 1px solid #39a6b2;
}
```

The best solution, of course, is the one that makes the most sense to you!

DEEP DIVE ▲

DEEP DIVE

Skim over the **MDN web docs on pseudo-classes (https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes)** to see what else is available.

This is a great stopping point to `add`, `commit`, and `push` these changes to your branch using the following commands:

```
git add .
git commit -m "<your commit message>"
git push origin feature/aesthetics
```