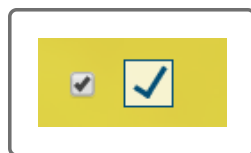


## 2.5.7 Customize the Checkboxes and Radio Buttons

The Run Buddy folks like the changes we've been making, but they're also wondering when we're going to update those ugly default checkboxes and radio buttons. In the following picture, the browser's default styling for a checkbox is on the left while the requested styling is on the right:



The one on the right is definitely an improvement, but getting there isn't exactly intuitive. Browsers don't let us change much about these elements. Some developers remove them entirely and use JavaScript to simulate their behavior, but it would be better practice to keep the radio buttons and checkboxes on the page so assistive technologies can still read them.

What we can do, though, is make the default buttons invisible and use CSS to create and position our own checkboxes and radio buttons on top of them. This is a good compromise because screen readers will ignore the CSS while still picking up the real buttons underneath.

Let's first restructure the HTML in the hero form to accommodate this approach. Replace the two `<p>` elements underneath the phone number

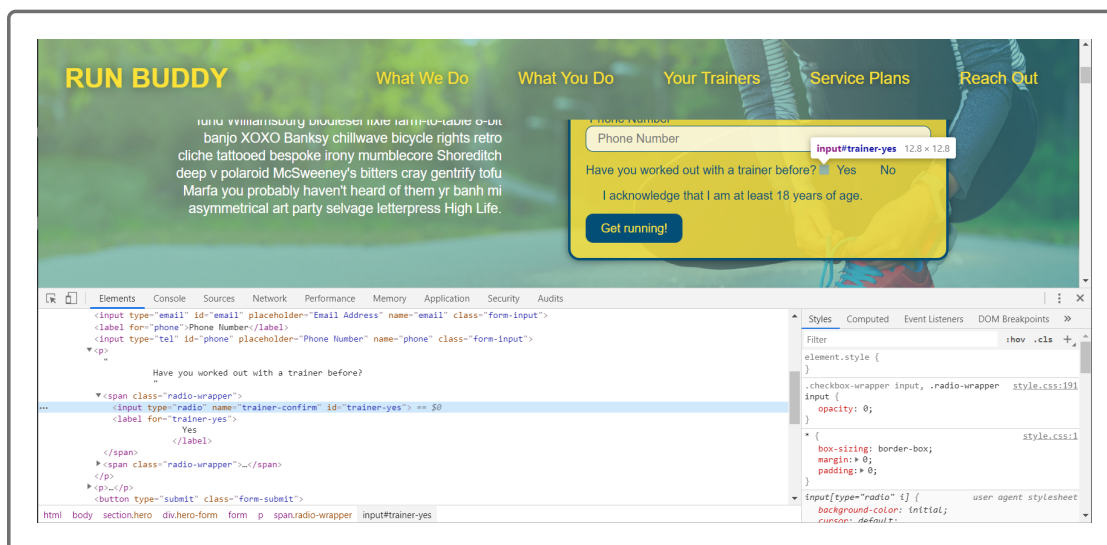
field with the following:

```
<p>
  Have you worked out with a trainer before?
  <span class="radio-wrapper">
    <input type="radio" name="trainer-confirm" id="trainer-yes" />
    <label for="trainer-yes">Yes</label>
  </span>
  <span class="radio-wrapper">
    <input type="radio" name="trainer-confirm" id="trainer-no" />
    <label for="trainer-no">No</label>
  </span>
</p>
<p>
  <span class="checkbox-wrapper">
    <input type="checkbox" name="age-confirm" id="checkbox" />
    <label for="checkbox">I acknowledge that I am at least 18 years of
  </span>
</p>
```

Note that we wrapped each of the `<input>` elements and their labels in a `<span>` element. Next, perform a little CSS magic and make the inputs inside of these wrappers disappear:

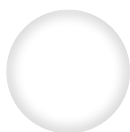
```
.checkbox-wrapper input, .radio-wrapper input {
  opacity: 0;
}
```

The `opacity` property works much like the alpha value in `rgba()`, where transparency is defined on a scale of 0.0 to 1.0. This made all of the inputs invisible, but if you inspect the page with the Chrome DevTools, you can see that the elements are still there:



Before we add the custom buttons, we'll need to prep the `<label>` elements so the new buttons can fit inside:

```
.checkbox-wrapper label, .radio-wrapper label {
  position: relative;
  left: 10px;
  margin: 10px;
  line-height: 1.6;
}
```



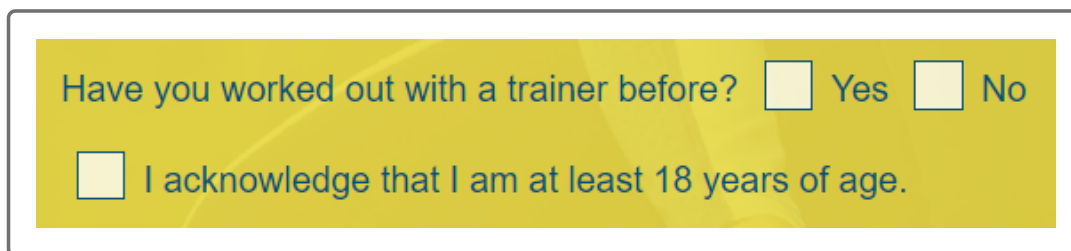
## REWIND

Earlier in our Run Buddy career, we needed to apply relative positioning to the hero section so the form inside could be positioned absolutely. The same principle applies here. The custom radio/checkbox buttons are going to sit absolutely inside of the labels.

Now here's the crazy part. We're not going to add our new buttons in the HTML code. We're going to use CSS and **pseudo-elements** to do this. Add the following CSS rule to your style sheet:

```
.checkbox-wrapper label::before, .radio-wrapper label::before {  
  content: "";  
  height: 20px;  
  width: 20px;  
  background: rgba(255, 255, 255, 0.75);  
  border: 1px solid #024e76;  
  position: absolute;  
  top: -4px;  
  left: -30px;  
}
```

This should produce the following result in the browser:



The `::before` pseudo-element lets us inject content into an element using CSS! It places this content before everything else that currently resides in the element. Pretty cool, right?

Here, we left the actual `content` property empty but used additional CSS properties to create a white square with a blue border. We then used absolute positioning to move this box on top of the invisible, original input element.

Unfortunately, we've made our new radio buttons square. Let's fix this with another CSS rule that targets only the `radio-wrapper` class:

```
.radio-wrapper label::before {  
  border-radius: 50%;  
}
```

## HIDE PRO TIP

Setting the `border-radius` to `50%` is a quick way to turn an element into a perfectly round circle or oval.

Awesome, we're halfway there! Next, we need to define what the buttons look like when checked, or clicked. To do this, we'll need another pseudo-element to overlay on top of our first one. We can't have two `::before` pseudo-elements in the same place, but we can use the `::after` pseudo-element, like this:

```
.radio-wrapper label::after {  
  content: "";  
  width: 20px;  
  height: 20px;  
  border-radius: 50%;  
  background: #024e76;  
  position: absolute;  
  left: -29px;  
  top: -3px;  
}  
  
.checkbox-wrapper label::after {  
  content: "";  
  height: 6px;  
  width: 14px;  
  border-left: 2px solid #024e76;  
  border-bottom: 2px solid #024e76;  
  position: absolute;  
  left: -26px;
```

```
top: 1px;  
}
```

Similar to `::before`, we didn't put any text content in the `::after` pseudo-element. Instead, we used it to create a colored circle for the radio buttons and some weird L-shaped thingy for the checkbox:

Alas, making an actual checkmark requires a bit of tinkering. Here, we created a rectangle that only has two borders, which makes it look like an L. With another CSS property, though, we can rotate this L just enough to make it look more like a checkmark.

Add this declaration to the `.checkbox-wrapper label::after` rule:

```
transform: rotate(-58deg);
```

We can also make the radio buttons look a little nicer by replacing the solid background color with a gradient. Add this to the `.radio-wrapper label::after` rule:

```
background-image: radial-gradient(circle, #39a6b2, #024e76);
```

These are looking pretty good:

Have you worked out with a trainer before? ☐ Yes ☐ No

☒ I acknowledge that I am at least 18 years of age.

The only problem is that they're stuck in the "checked" state. But these are pseudo-elements—they don't have a checked state! However, the invisible inputs that we covered up do. Remember how clicking on a `<label>` element checks the input that's tied to it? That's still working. We just don't see it anymore. However, we can still tap into the state of these inputs using a pseudo-class:

```
.checkbox-wrapper input:checked, .radio-wrapper input:checked {  
  opacity: 1;  
}
```

In the browser, click on the custom checkbox and notice how the visibility of the real checkbox is toggled on and off:

☒ I acknowledge that I am at least 18 years of age.

Obviously, this behavior isn't desirable. We want the state of this input to either show or hide the `::after` pseudo-element. Fortunately, CSS has sibling selectors that we can use to target an element that sits directly after another element.

Delete the `input:checked` rule we just wrote and replace it with this one:

```
.checkbox-wrapper input:checked + label,  
.radio-wrapper input:checked + label {
```

```
font-weight: bold;
}
```

## DEEP DIVE ▲

### DEEP DIVE

---

To learn more, see the [MDN web docs on sibling selectors](https://developer.mozilla.org/en-US/docs/Web/CSS/Adjacent_sibling_combinator) [\\_\(\[https://developer.mozilla.org/en-US/docs/Web/CSS/Adjacent\\\_sibling\\\_combinator\]\(https://developer.mozilla.org/en-US/docs/Web/CSS/Adjacent\_sibling\_combinator\)\)\\_](https://developer.mozilla.org/en-US/docs/Web/CSS/Adjacent_sibling_combinator).

Now the boldness of the `<label>` element is dependent on the `:checked` state of its sibling element! That's still not quite what we wanted, though. Let's give this one last shot.

Delete the `input:checked + label` rule and replace it with this one:

```
.checkbox-wrapper input + label::after,
.radio-wrapper input + label::after {
  content: none;
}

.checkbox-wrapper input:checked + label::after,
.radio-wrapper input:checked + label::after {
  content: "";
}
```

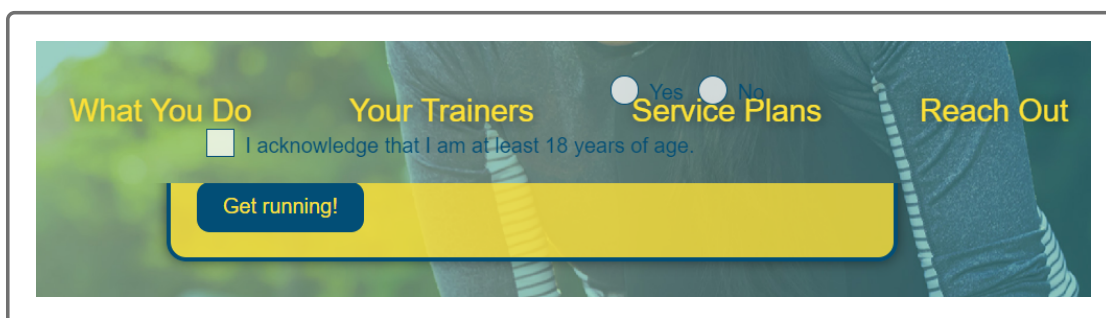
If the radio buttons or checkbox are in their default state, they will have no `::after` pseudo content. As soon as an input becomes `:checked`, however, the content and all of the other styles defined earlier will kick in.



**IMPORTANT**

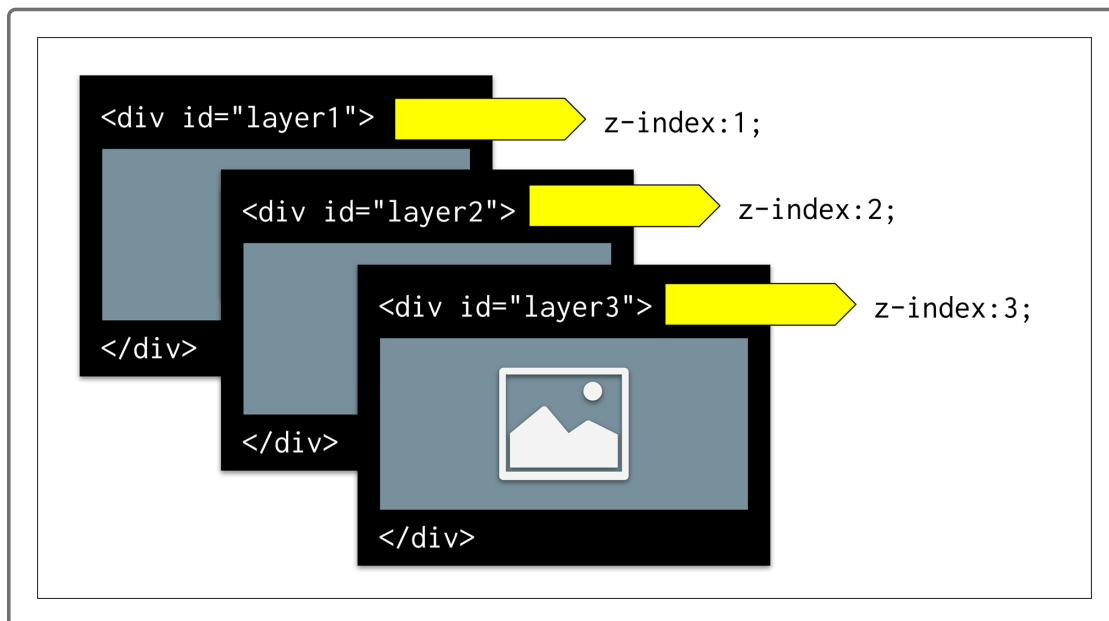
In programming, there's a big difference between empty text ("" ) and nothing at all. This will become doubly important once we get into JavaScript.

It would seem that we're done, but as you scroll up and down on the page again, you might notice some weird overlapping with the header:



Hmm. Why do the custom buttons and labels appear in front? The reason has to do with something called **stacking order**. As soon as you give an HTML element positioning (relative, absolute, sticky), stacking order comes into play.

Think of HTML elements like pieces of paper that stack on top of each other, as this image shows:



In our Run Buddy webpage, the header was the first element to receive positioning, so it sat on top. When we gave the form elements positioning, however, those "pieces of paper" were brought to the front and covered up the header. Fortunately, with the `z-index` property, we can change this stacking order, or "reshuffle the papers," so to speak.

Add the following declaration to the `header` rule to keep this element on top:

```
z-index: 9999;
```

A higher `z-index` brings the element to the front while a lower `z-index` pushes it to the back. Does the value `9999` seem like overkill in this case? A value of `z-index: 1;` would still fix our header/form problem. We might end up needing to add `z-index` to other elements, though, and we want to ensure the header is always on top, so we'll go with `9999` to avoid future conflicts.

Save your code if you haven't already and check your work in the browser. To truly appreciate `z-index`, open Chrome's DevTools and toggle the declaration on and off!

© 2020 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.