Test your skills: CSS and JavaScript accessibility

- CSS accessibility 1
- CSS accessibility 2
- JavaScript accessibility 1

The aim of this skill test is to help you assess whether you've understood our CSS and JavaScript accessibility best practices article.

> **NOTE:** To get help, read our Test your skills usage guide. You can also reach out to us using one of our communication channels.

CSS accessibility 1

In the first task you are presented with a list of links. However, their accessibility is pretty bad — there is no way to really tell that they are links, or to tell which one the user is focused on. We'd like you to assume that the existing ruleset with the `a` selector is supplied by some CMS, and that you can't change it.

To complete the task, create new rules to make the links look and behave like links, and for the user to be able to tell which link they are focused on in the list.

```
body {
  background-color: white;
  color: #333333;
  font:
    1em / 1.4 "Helvetica Neue",
    "Helvetica",
```

```
    "Arial",
    sans-serif;
  padding: 1em;
  margin: 0;
}

* {
  box-sizing: border-box;
}
```

The starting point of the task looks like this:

Here's the underlying code for this starting point:

```
<ul>
  <li><a href="">Animals</a></li>
  <li><a href="">Computers</a></li>
  <li><a href="">Diversity and inclusion</a></li>
  <li><a href="">Food</a></li>
  <li><a href="">Medicine</a></li>
  <li><a href="">Music</a></li>
</ul>
```

```
a {
  text-decoration: none;
  color: #666666;
  outline: none;
}

/* Don't edit the above code! */

/* Add your code here */
```

When the task is complete, the links should look something like this:

Click here to show the solution

Your finished CSS could look something like this:

```
/* ... */
/* Don't edit the above code! */

li a {
  text-decoration: underline;
  color: rgb(150 0 0);
}

li a:hover,
li a:focus {
  text-decoration: none;
  color: red;
}
```

```
a {
  text-decoration: none;
  color: #666666;
  outline: none;
}

li a {
  text-decoration: underline;
  color: rgb(150 0 0);
}

li a:hover,
li a:focus {
  text-decoration: none;
  color: red;
}
```

CSS accessibility 2

In this next task you are presented with a simple bit of content — just headings and paragraphs. There are accessibility issues with the colors and sizing of the text, and we'd like you to fix them.

To complete the task:

1. Think about what the problems are, and which guidelines state the acceptable values for color and sizing.
2. Update the CSS with new values for the color and font-size to fix the problem.
3. Test the code to make sure the problem is now fixed. Explain what tools or methods you used to select the new values and test the code.

The starting point of the task looks like this:

Here's the underlying code for this starting point:

```
<main>
  <h1>I am the eggman</h1>

  <p>
    Prow scuttle parrel provost Sail ho shrouds spirits boom mizzenmast yardarm.
    Pinnace holystone mizzenmast quarter crow's nest nipperkin grog yardarm
    hempen halter furl.
  </p>

  <h2>They are the eggman</h2>

  <p>
    Swab barque interloper chantey doubloon starboard grog black jack gangway
    rutters.
  </p>

  <h2>I am the walrus</h2>

  <p>
    Deadlights jack lad schooner scallywag dance the hempen jig carouser
    broadside cable strike colors.
  </p>
</main>
```

```
/* Edit the CSS to fix the a11y problems */

main {
  padding: 20px;
  background-color: red;
}
```

```
h1,
h2,
p {
  color: #999999;
}

h1 {
  font-size: 2vw;
}

h2 {
  font-size: 1.5vw;
}

p {
  font-size: 1.2vw;
}
```

The updated content should look like this:

Click here to show the solution

1. The problems are:
   - The color contrast is not acceptable, as per WCAG criteria 1.4.3 (AA) and 1.4.6 (AAA).
   - The text is sized using `vw` units, which means that it is not zoomable in most browsers. WCAG 1.4.4 (AA) states that text should be resizable.
2. To fix the code, you need to
   - Choose a better contrasting set of background and foreground colors.
   - Use some different units to size the text (such as `rem` or even `px`), or even implement something that uses a combination of `vw` and other units, if you want it resizable but still relative to the viewport size.
3. For testing:
   - You can test color contrast using a tool such as aXe, The Firefox Accessibility Inspector, or even a simple standalone web page tool like the WebAIM Contrast Checker.
   - For text resizing, you'd need to load the example in a browser and try to resize it. Resizing `vw` unit-sized text works in Safari, but not in Firefox or Chromium-based browsers.

For the updated code, something like this would fix the color contrast:

```
main {
  padding: 20px;
  background-color: red;
}

h1,
h2,
p {
  color: black;
}
```

And something like this would work for the font sizing:

```
h1 {
  font-size: 2.5rem;
}

h2 {
  font-size: 2rem;
}

p {
  font-size: 1.2rem;
}
```

Or this, if you want to do something a bit cleverer that gives you resizable viewport-relative text:

```
h1 {
  font-size: calc(1.5vw + 1rem);
}

h2 {
  font-size: calc(1.2vw + 0.7rem);
}

p {
```

```
  font-size: calc(1vw + 0.4rem);
}
```

JavaScript accessibility 1

In our final accessibility task, you have some JavaScripting to do. We have an app that presents a list of animal names. Clicking one of the animal names causes a further description of that animal to appear in a box below the list.

But it is not very accessible — in its current state you can only operate it with the mouse. We'd like you to add some HTML and JavaScript to make it keyboard accessible too.

The starting point of the task looks like this:

Here's the underlying code for this starting point:

```
<section class="preview">
  <div class="animal-list">
    <h1>Animal summaries</h1>

    <p>
      The following list of animals can be clicked to display a description of
      that animal.
    </p>

    <ul>
      <li
        data-description="A type of wild mountain goat, with large recurved horns, found in
Eurasia, North Africa, and East Africa.">
        Ibex
      </li>
      <li
        data-description="A medium-sized marine mammal, similar to a manatee, but with a
Dolphin-like tail.">
        Dugong
      </li>
      <li
        data-description="A rare marsupial, which looks rather like a tiny kangaroo, measuring
around 50 to 75 centimeters.">
```

```
        Quokka
      </li>
    </ul>
  </div>

  <div class="animal-description">
    <h2></h2>

    <p></p>
  </div>
</section>
```

```
p {
  color: purple;
  margin: 0.5em 0;
}

li {
  cursor: pointer;
}
```

```
const listItems = document.querySelectorAll("li");
const descHeading = document.querySelector(".animal-description h2");
const descPara = document.querySelector(".animal-description p");

listItems.forEach((item) => {
  item.addEventListener("mouseup", handleSelection);
});

function handleSelection(e) {
  const heading = e.target.textContent;
  const description = e.target.getAttribute("data-description");
  descHeading.textContent = heading;
  descPara.textContent = description;
}
```

We've not provided finished content for this task, as it looks the same as the starting point.

Click here to show the solution

1. To begin with, you'll need to add `tabindex="0"` to the list items to make them focusable via the keyboard.
2. Then you'll need to add in another event listener inside the `forEach()` loop, to make the code respond to keys being pressed while the list items are selected. It is probably a good idea to make it respond to a specific key, such as "Enter", in which case something like the following is probably acceptable:

```
item.addEventListener("keyup", (e) => {
  if (e.key === "Enter") {
    handleSelection(e);
  }
});
```

□□□ #3
□ □□□ □□□ 2026-10-31 11:27:53 UTC
□ cocobook-sync □□□ 2026-11-04 17:32:31 UTC