basic accessibility: the problem and how to fix it

The WebAIM Million presents a sad report of where the top one million websites on the web from an accessibility perspective. The results are sad.

In the table below, we see the most common accessibility issues and what percentage of the top one million websites exhibit the issue.

WCAG Failure Type	% of home pages in 2022	% of home pages in 2021	% of home pages in 2020	% of home pages in 2019
Low contrast text	83.9%	86.4%	86.3%	85.3%
Missing alternative text for images	55.4%	60.6%	66.0%	68.0%
Empty links	50.1%	51.3%	59.9%	58.1%
Missing form input labels	46.1%	54.4%	53.8%	52.8%
Empty buttons	27.2%	26.9%	28.7%	25.0%
Missing document language	22.3%	28.9%	28.0%	33.1%

To me this is appalling. These are not complicated things to fix or automate a solution for.

This post will look at some of the problems and how to fix them manually. In a future post we'll look at tooling to automate a11y testing.

Manual fixes

The manual fixes require working with CSS and HTML. It also requires external tools and the built-in browser developer tools.

Low contrast text

The problem is that the lower contrast the harder it is to read. This is mostly a

design decision… developers have less influence in the design of a page or app than designers do.

That said we can manually check the contrast of foreground over background text using tools like the Contrast Checker or Deque's Deque Color Contrast Analyzer.

The goal is to reach the following levels under the following conditions:

Specification	Level	Normal Text	Large Text
WCAG 2.0	AA	4.5:1	3:1
WCAG 2.0	AAA	7:1	3:1
WCAG 2.1	AA	4.5:1	3:1
WCAG 2.1	AAA	7:1	4.5:1

WCAG 2.1 AAA requires a constrast level of 3:1 for graphics and user interface components such as form input borders.

Large text is defined as 14 point (typically 18.66px) and bold or larger, or 18 point (typically 24px) or larger.

You can extract colors from the page you're viewing using extensions like Colorzilla (available for Chrome and Firefox).

Since version 89, Chrome has a different alorithm for calculating contrast. In New color contrast calculation - Advanced Perceptual Contrast Algorithm (APCA) (part of what's new in DevTools for Chrome 89), the DevTools team explains what APCA is and how it works.

Since this is an experimental feature, I would hold off on using it until it's available in other browsers or becomes part of WCAG 3.0 or later.

Missing alternative text for images

This one is another low hanging fruit that can easily be fixed manually just by being careful.

When writing the code for an image, whether it's a standalone figure or a responsive image with multiple srcsets, you can add the `alt` attribute to the

image, whether it's the default image for a responsive set or a standalone image.

```
  <img
    src="https://assets.codepen.io/32795/santiago002.jpg"
    alt="View of the mountains from the northern part of Santiago, Chile"
```

The following responsive image shows the inclusion of the `alt` attribute as a normal image would:

```
<img srcset="safari-photo-480w.jpg 480w,
             safari-photo-800w.jpg 800w"
     sizes="(max-width: 600px) 480px, 800px"
     src="safari-photo-800w.jpg"
     alt="Safari photo of a male lion approaching">
```

This is as easy as it gets. There is no excuse why we can't add this when adding the tags, if we're writing HTML or templates for our CMS content, or when entering data into a CMS.

TO check that the alt attribute is present, you can do a global search for the images on the page and check that the `alt` attribute is present and has a meaningful value.

If you use `figure` elements with the optionsal `figcaption` child, you should still use the `alt` attribute in the `img` child. Figure and figcaption – extended alternate text for screen readers? shows how different screen readers handle the figcaption attribute. the TL;DR is that the `figcaption` attribute is not a substitute for the `alt` attribute.

Empty links

A lot of times when I see a link with no text, either because the author forgot it or because it includes an icon or an image like a social media icon.

```
<a href="https://facebook.com" id="myLink"></a>
```

or

```
<a
  href="https://facebook.com"
  id="myLink">
    <img
      width="24"
      height="96"
      src="https://upload.wikimedia.org/wikipedia/commons/5/51/Facebook_f
      alt=""></a>
```

The problem with this is that screen readers and assistive technologies don't have a way to derive context from the link alone. They use the content of the alt attribute as the content that they'll show the user.

According to Hidde de Vries:

> If you were to replace your image with a square, what would you write in the square for it to still be useful?
>
> Source: Common accessibility issues that you can fix today

Leaving it empty is an option if that's the most useful alternative for the image.

Missing form input labels

39% of the 4.4 million form inputs identified were not properly labeled either via <label> child element, aria-label, aria-labelledby, or title.

```
<form action="" method="get">
  <div class="form-example">
    <label for="name">
      Enter your name:
    </label>
    <input type="text"
      name="name"
      id="name"
      required>
  </div>
```

```
  <div class="form-example">
    <label for="email">
      Enter your email:
    </label>
    <input type="email"
      name="email"
      id="email"
      required>
  </div>
  <div class="form-example">
    <input type="submit"
      value="Subscribe!">
  </div>
</form>
```

To me, this is just a design and development decision. You should make this part of your reviews: all form field should have a label and indicate what element they are intended for.

We can also use aria-label to indicate the label for the form element or aria-labelledby to indicate the element that describes the form element.

Headings

The number of headings and the sequence of headings matter.

Headings are the primary mechanism used by screen reader users to navigate content so their proper implementation is important.

According to the WebAIM

> 19.6% of home pages had more than one <h1> – an increase from 18.4% in 2021. There were 1,092,097 instances of skipped heading levels (e.g., jumping from <h2> to <h4>) and 1 in every 21 headings was improperly structured. Skipped headings were present on 40.4% of all pages (up from 38.4% in 2021), and 9.9% of pages had no headings present at all (down from 10.6% in 2021).

> Source: WebAIM One Million

Here are a few tips on how to use headings in an accessible way.

- Only use one <h1> element per page. This will usually be the site title
- Use headings sequentially. If the first element is a <h1> it's children should be <h2>, its children should be a <h3> and so on
- Don't skip levels. Don't go from <h2 to <h4>. Use the sequence as it was intended

Empty buttons

Just like links, not having content between the button opening and closing tags hurts accessibility. By default, a button's accessible name is the content between the opening and closing <button> tags so not having any content or having content that can't be read by assistive technology makes the button inaccessible.

Even though this buttom is perfectly usable as a search button for abled users, it won't work with screen readers.

```
<button type="submit">
   <svg id="search" viewBox="0 0 16 16.9">
      <path d="M16, 15.7L11.3,11C12.4,9.8,13, 8.2,13,6.5C13"></path>
   </svg>
</button>
```

The first way to fix this is to save the SVG as an image and load it like a typical image with an `alt` attribute.

```
<button type="submit">
   <img src="/search-icon.svg" alt="Search"/>
</button>
```

Depending on what your SVG does, it may not work in browserrs.

If this is the case then the second option is to keep it as an inline SVG and modifying it with accessibility attributes.

We add `aria-describedby` to the SVG element pointing to the element that

describes the button, `search` in this case.

Then, we add the `title` element as a child of SVG and with an `ID` attribute that matches the name we used in the `aria-describedby` attribute of the parent SVG element.

```
<button type="submit">
  <svg id="search" role="img" aria-describedby="searchIcon" viewBox="0 0 1
    <title id="searchIcon">Search</title>
    <path d="M16, 15.7L11.3,11C12.4,9.8,13, 8.2,13,6.5C13"></path>
  </svg>
</button>
```

You can read about possible strategies and really go in depth on proper markup in images and SVG graphics in Contextually Marking up accessible images and SVGs by Scott O'Hara.

Naming things to improve accessibility gives pointers on how to name things to make them more accessible.

Accessible Name and Description Computation 1.1 is the specification that explains how browsers compute the accessible name and description of elements.

Missing document language

This one was interesting to me. The `html` root element needs to have a language content.

Unless it's overriden. Setting the `lang` attribute in the `html` element will set the default language for the page.

```
<html lang="en">
```

This will help screen readers load language-appropriate pronuciation and other rules thaat will help present content in different languages.

If a page uses multiple languages you can set the `lang` attribute for each part that uses a different language.

```
<html lang="en">
  <head>
    <title>My Page</title>
  </head>
  <body>
    <h1>Hello World</h1>

    <aside lang="es">
      <h2>Información para hispanohablantes</h2>
    </aside>

    <aside lang="fr">
      <h2>Informations pour les francophones</h2>
    </aside>
  </body>
```

Links and resources

- The WebAIM Million 2022 report
- One million broken web sites - and a way to prevent that
- How to Fix Your Low-Contrast Text
- Common accessibility issues that you can fix today
- Figure and figcaption – extended alternate text for screen readers?
- How The HTML Lang Attribute Helps Accessibility
- Understanding Success Criterion 3.1.2: Language of Parts