How to Code a Website for Free in 6 Easy Steps

Stephen Roddewig

Published: April 03, 2024

As someone who's navigated the complexities of web development, I understand the challenge of learning to code a website for free. The internet is where I turn for everything from finding the best local eateries to mastering the art of web design. Now, I want to guide you through this journey.

Embarking on the path to coding a website for free is more than a cost-saving venture; it‘s empowering yourself in our digital world. Whether you’re a small business owner dreaming of an online presence or a curious learner like I once was, I‘m here to share the insights I’ve gained along the way.

Table of Contents

1. Pick your code editor.

2. Write your HTML.

3. Create your CSS stylesheet.

4. Put your HTML and CSS together.

5. Code a responsive website or a static website.

6. Code a simple website or an interactive website.

Let's start by tackling two common questions that many often wonder when beginning their web development adventure.

Frequently Asked Questions About Coding a Website

1. How long does it take to code a website?

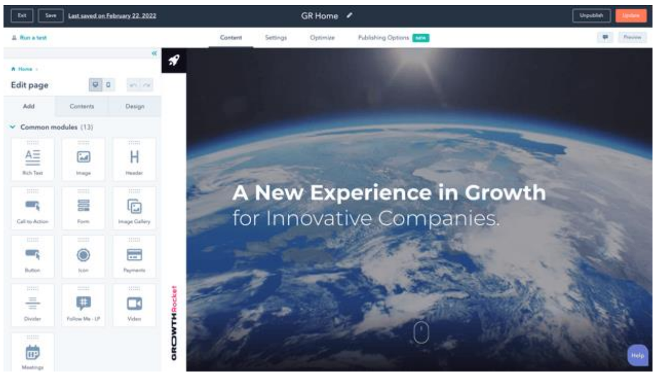
Answer: It varies greatly based on complexity. For a basic homepage, it might take a few hours, while an e-commerce site could take weeks or more. Remember, a website can evolve

2. Do I need to code my website?

Answer: Not necessarily. You've got two main paths to choose from when building a website: using a website builder or coding from scratch. Let me break down the differences for you.

Website Builders: The Quick and User-Friendly Route

CMS platforms like HubSpot's Content Hub offer user-friendly interfaces. Many CMSs offer drag-and-drop capabilities so that you can easily position elements (e.g. a button) where you want them on the page. However, understanding the basics of HTML and CSS can be incredibly empowering and offer more customization options.



Start building your website with HubSpot's Content Hub.

Coding from Scratch: The Road to Customization

On the other side, coding your website from scratch with HTML, CSS, and possibly JavaScript is like cooking a meal from scratch. It takes more time and effort, but the result is a website tailored exactly to your taste and needs.

Learning to code gives you complete control over your website. You're not limited to the templates or features of a website builder. This path is incredibly empowering and allows for greater customization and uniqueness in your design.

Website Builders vs. Coding From Scratch

If you're in a hurry or not keen on diving into code, check out our list of the best free website builders. They're a great starting point.

However, if you're up for a challenge and want to craft a site that's truly your own, then coding is the way to go. If you're ready to learn some code, let's walk through this process together.

How to Code a Website

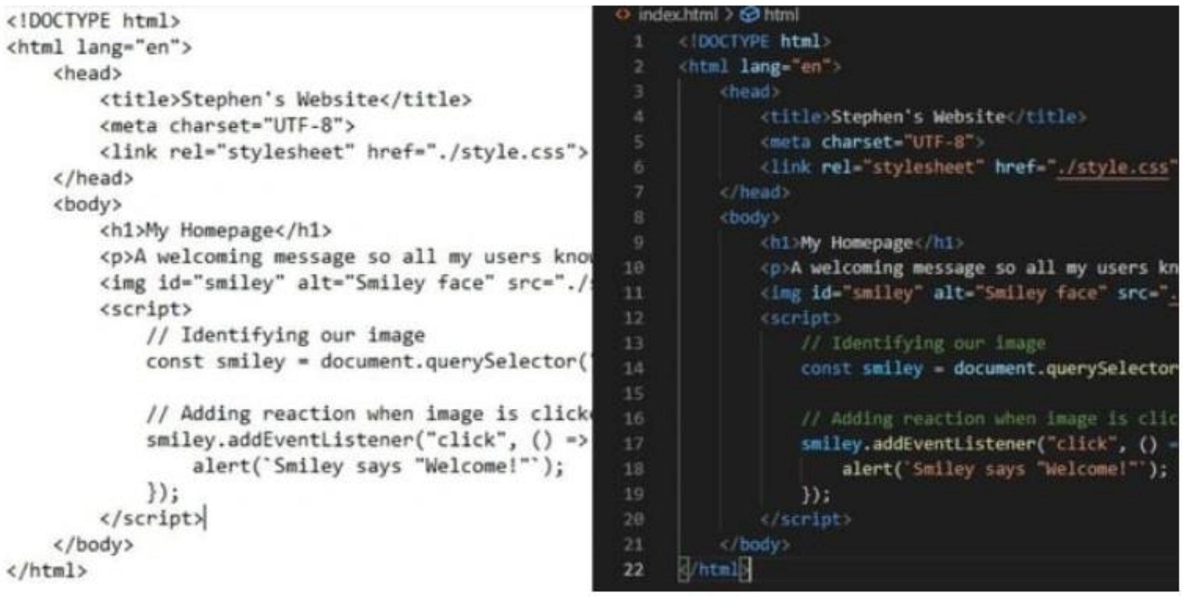
1. Pick your code editor.
2. Write your HTML.
3. Create your CSS stylesheet.
4. Put your HTML and CSS together.
5. Code a responsive website or a static website.
6. Code a simple website or an interactive website.

1. Pick your code editor.

Let me share a bit of my coding journey with you. When I first started as a new developer, picking a code editor felt like a maze. It's like finding that perfect pair of running shoes – it just makes the run smoother.

I remember the satisfaction when I found Visual Studio Code. It offers syntax suggestions so you can avoid simple typos that cause issues in your code. On top of that, it will autocomplete closing tags for HTML and add visual markers to your files so that you can easily tell different pieces of code apart.

Below I highlighted the difference between a regular text editor and a code editor. The code works the same in both documents, but one is much easier to understand at a glance than the other.



```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Stephen's Website</title>
        <meta charset="UTF-8">
        <link rel="stylesheet" href="./style.css">
    </head>
    <body>
        <h1>My Homepage</h1>
        <p>A welcoming message so all my users kno
        <img id="smiley" alt="Smiley face" src="./
        <script>
            // Identifying our image
            const smiley = document.querySelector('

            // Adding reaction when image is click
            smiley.addEventListener("click", () =>
                alert(`Smiley says "Welcome!"`);
            });
        </script>
    </body>
</html>
```

index.html > html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Stephen's Website</title>
        <meta charset="UTF-8">
        <link rel="stylesheet" href="./style.css"
    </head>
    <body>
        <h1>My Homepage</h1>
        <p>A welcoming message so all my users kn
        <img id="smiley" alt="Smiley face" src="._
        <script>
            // Identifying our image
            const smiley = document.querySelector

            // Adding reaction when image is clic
            smiley.addEventListener("click", () -
                alert(`Smiley says "Welcome!"`);
            });
        </script>
    </body>
</html>
```

Here's a personal tip: while I lean towards Visual Studio Code, the tech world is your oyster. There are numerous free code editors out there, each with unique features. Some have amazing extensions, like the dark-to-light mode switch in Visual Studio Code. The key is to find one that resonates with your style of learning and coding.

I will be sharing screenshots from Visual Studio Code, so it will be easy to follow along in this tool as we dive into coding in the next step.

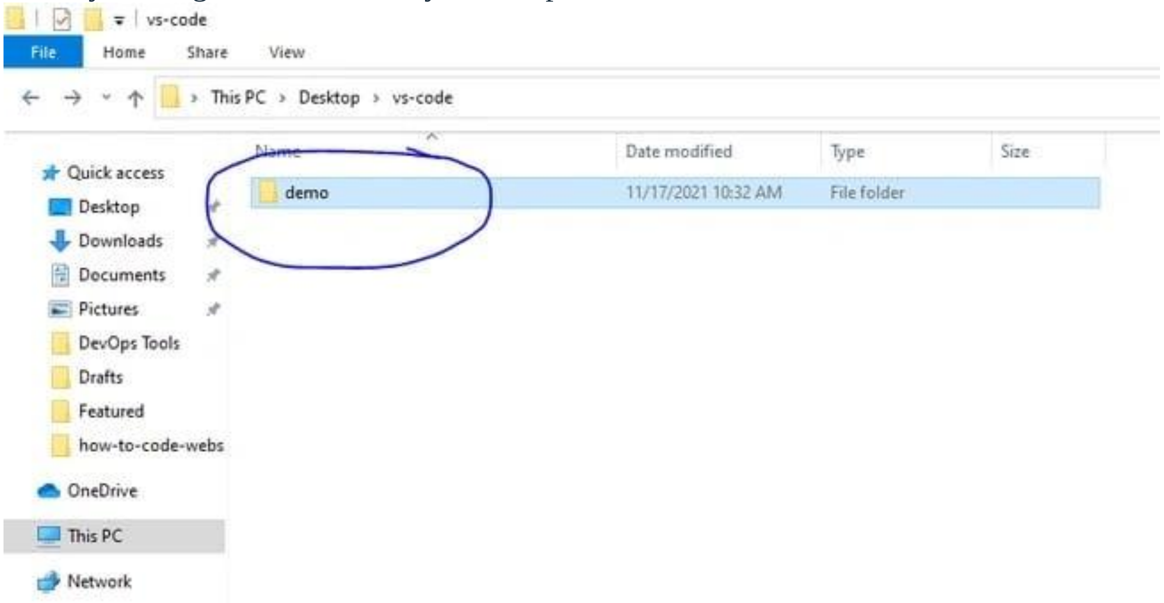
2. Write your HTML.

Time to roll up our sleeves and dive into HTML, or as I like to think of it, the building blocks of the web. My first experience with HTML was like learning a new language, but it's really about building the structure of your home, piece by piece. This is a metaphor I use when I lead HTML and CSS training, so we'll repurpose it for this tutorial as well.

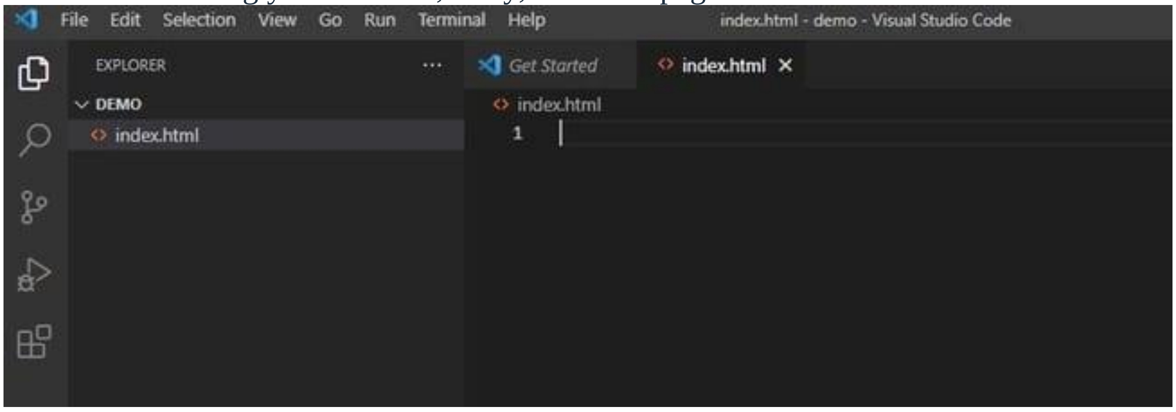
Now that we have the right tool for the job and understand our goal, it‘s time to dive into the reason you’re here: coding.

Create an HTML doc.

- Start by making a new folder on your computer. I named mine "demo".



- Open this folder in your code editor.
- Create a new file and name it something like "index.html". Remember, ".html" is the key here. It's like telling your browser, "Hey, I'm a webpage!"



Structure your document.

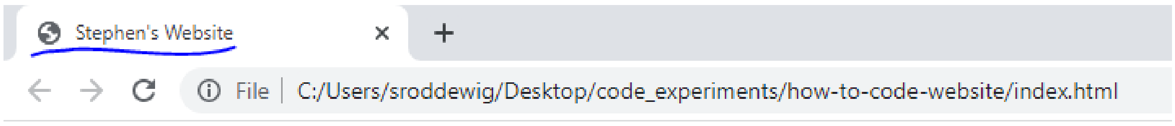
Our file is currently blank, so let's add the HTML document structure. I have laid everything out in the snippet below so you can copy and paste it into your file (or write it out).

```
<!DOCTYPE html> <html lang="en"> <head> <title>Stephen's Website</title>
</head> <body> <!-- Content goes here --> </body> </html>
```

Let's review each of these pieces briefly:

- **<!DOCTYPE html>**: A declaration to the browser that this is an HTML document type which will tell it how to process our code

- **<html lang="en">**: An opening tag to wrap all of our HTML so the browser knows where our code starts and where it ends (note the closing </html> tag)
- **<head>**: This section doesn't show up directly on the webpage but holds important info like your page title, also known as your metadata.

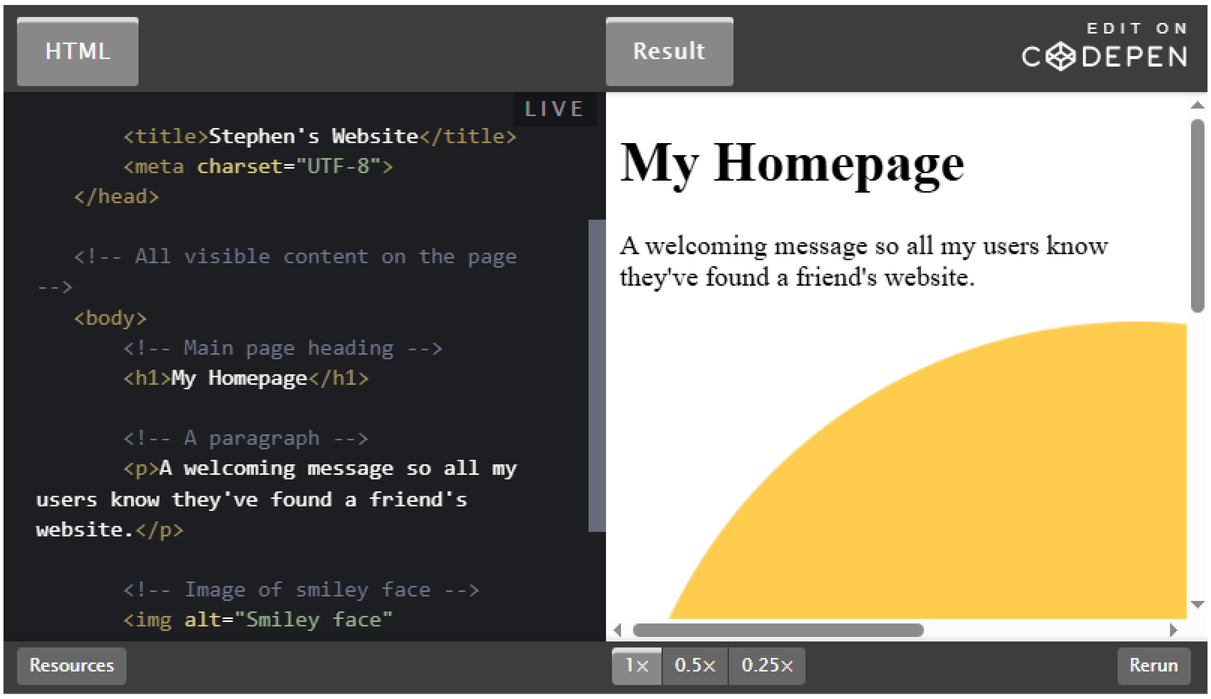

Stephen's Website
File | C:/Users/sroddewig/Desktop/code_experiments/how-to-code-website/index.html

- **<body>**: This is where all your content lives. We'll add elements to this next.

Tip: If you're not as familiar with HTML syntax, I recommend our HTML guide as a good way to brush up on your skills before continuing.

Add content to your body.

Time to add some personality! I've gone ahead and added content to our web page so we (finally) have something to look at. I usually start with a heading (`<h1>`) and a paragraph (`<p>`). You can see the updated HTML under the <body> on the left and what the user will see on the right.



```
        <title>Stephen's Website</title>
        <meta charset="UTF-8">
    </head>

    <!-- All visible content on the page
-->
    <body>
        <!-- Main page heading -->
        <h1>My Homepage</h1>

        <!-- A paragraph -->
        <p>A welcoming message so all my
users know they've found a friend's
website.</p>

        <!-- Image of smiley face -->
        <img alt="Smiley face"
```

My Homepage

A welcoming message so all my users know they've found a friend's website.

HTML:

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

  <!-- Metadata for the page -->
  <head>
    <!-- Title of webpage in the browser -->
    <title>Stephen's Website</title>
    <meta charset="UTF-8">
  </head>
  <!-- All visible content on the page -->
  <body>
    <!-- Main page heading -->
    <h1>My Homepage</h1>

    <!-- A paragraph -->
    <p>A welcoming message so all my users know they've found a friend's website.</p>

    <!-- Image of smiley face -->
    <img alt="Smiley face" src="https://assets.codepen.io/6093409/smiley-face.png">
  </body>
</html>
```

Result:

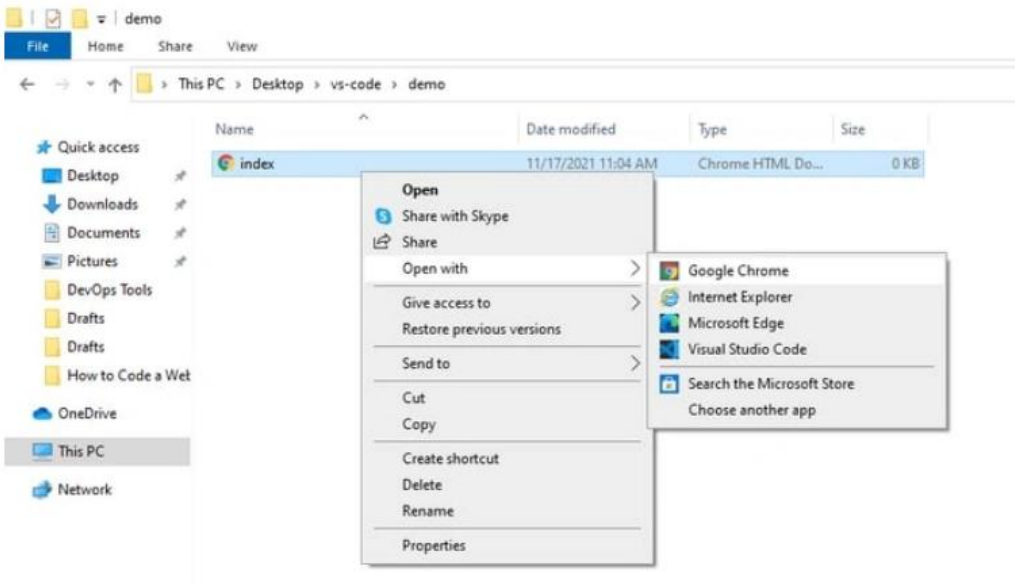
My Homepage

A welcoming message so all my users know they've found a friend's website.

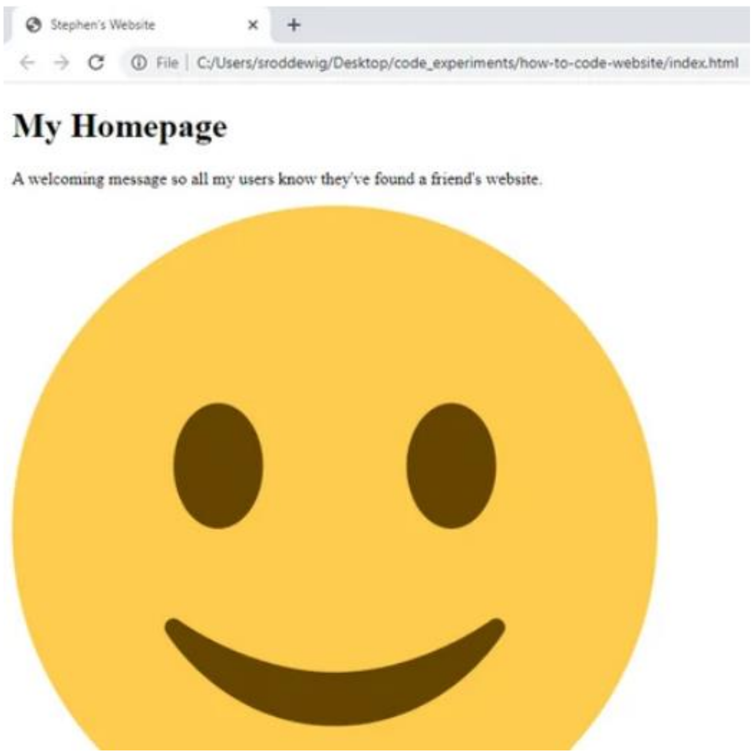
Pro Tip: the order of the attributes inside your HTML tag does not matter as long as they are all inside the tag.

Check your work.

The moment of truth: if you‘ve made it this far, you’re probably excited to see your hard work on the big screen (AKA web browser). The simplest way to do this is to return to your File Explorer and right-click your HTML file:



Once you‘ve selected your web browser of choice (I recommend Chrome, but any will work), you should see your HTML file displaying like it’s a live web page. Don't worry, only you can see it for now.


Stephen's Website
File | C:/Users/sroddewig/Desktop/code_experiments/how-to-code-website/index.html

My Homepage

A welcoming message so all my users know they've found a friend's website.

Pro Tip: If you make additional changes, you will need to save your file and then refresh the page to see them take effect in the browser. This will be more relevant as we review how to change the aesthetics of our page in the next section.

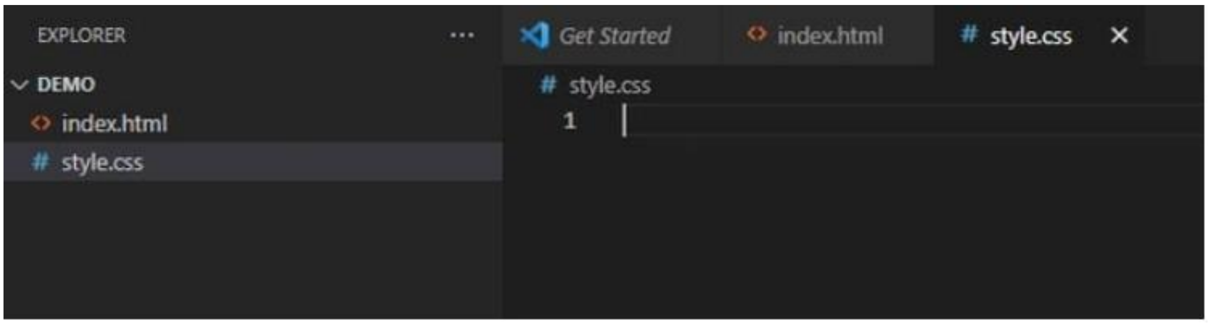
3. Create your CSS stylesheet.

Now, let's give your website its unique style with CSS. Imagine your HTML as a house's structure; CSS is what turns this structure into a home with color and personality. The moment I understood this, everything about web design clicked for me.

Tip: If you're unfamiliar with CSS, I recommend starting with our CSS guide to review the finer points of properties, syntax, selectors, and specificity.

Create the CSS doc.

- Just like with HTML, create a new file for CSS. I usually call mine "style.css", but feel free to name it whatever you like. The important part is the ".css" extension.
- Place this file in the same folder as your HTML file. It's like keeping all your design tools in one toolbox.



Add a CSS rule.

Let's start simple. I remember the first rule I ever wrote was to change the text color. It felt like a magic wand.

Here's an example:

This rule targets all paragraph elements (`<p>`) in your HTML and turns their text color red. It's amazing how a bit of color can transform a page.

```
p { color: red; }
```

You may notice that nothing happens on the page after you‘ve added this CSS rule to your file. That’s because we need to link our HTML and CSS together, which we'll show in the next section.

4. Put your HTML and CSS together.

Alright, it's time to bring your HTML and CSS into harmony. This is where your website starts to get its unique personality.

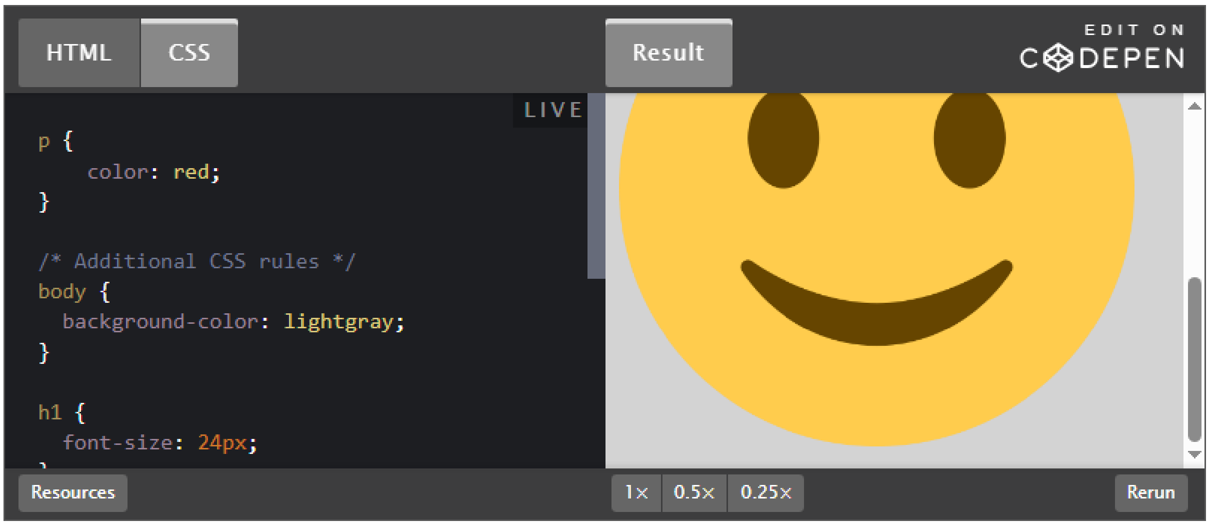
Linking CSS to HTML

To make your CSS styles apply to your HTML, you need to link them. This is done with a simple but crucial line in your HTML file's <head> section. Here’s how it's done:

```
<link rel="stylesheet" href="style.css">
```

More CSS Rules

Once linked, your webpage will start to reflect the styles you defined in your CSS file. I like to think of it as a black-and-white drawing being filled with color.



```
p {
    color: red;
}

/* Additional CSS rules */
body {
  background-color: lightgray;
}

h1 {
  font-size: 24px;
```

HTML:

```
<!DOCTYPE html>

<html lang="en">

    <head>

        <link rel="stylesheet" href="style.css">

        <title>Stephen's Website</title>

        <meta charset="UTF-8">

    </head>

    <body>

        <h1 class="text">My Homepage</h1>

        <p class="text">A welcoming message so all my users know they've found a friend's
website.</p>

        <img id="smiley" alt="Smiley face" src="https://assets.codepen.io/6093409/smiley-
face.png">

    </body>

</html>
```

CSS:

```
p {

    color: red;
}


/* Additional CSS rules */
body {
  background-color: lightgray;
}


h1 {
  font-size: 24px;
}


.text {
  color: green;
}


#smiley {
  width: 300px;
  height: auto;
}
```

Result:

My Homepage

A welcoming message so all my users know they've found a friend's website.

Here's an example of adding more personality to your site:

- Change the background color of your webpage
- Adjust the font size of your headings
- Apply different colors to text using class selectors

For me, this step was the most influential. It transformed my basic web pages into something I was proud to show off. Now, let’s see how you can make your webpage pop with these new styles.

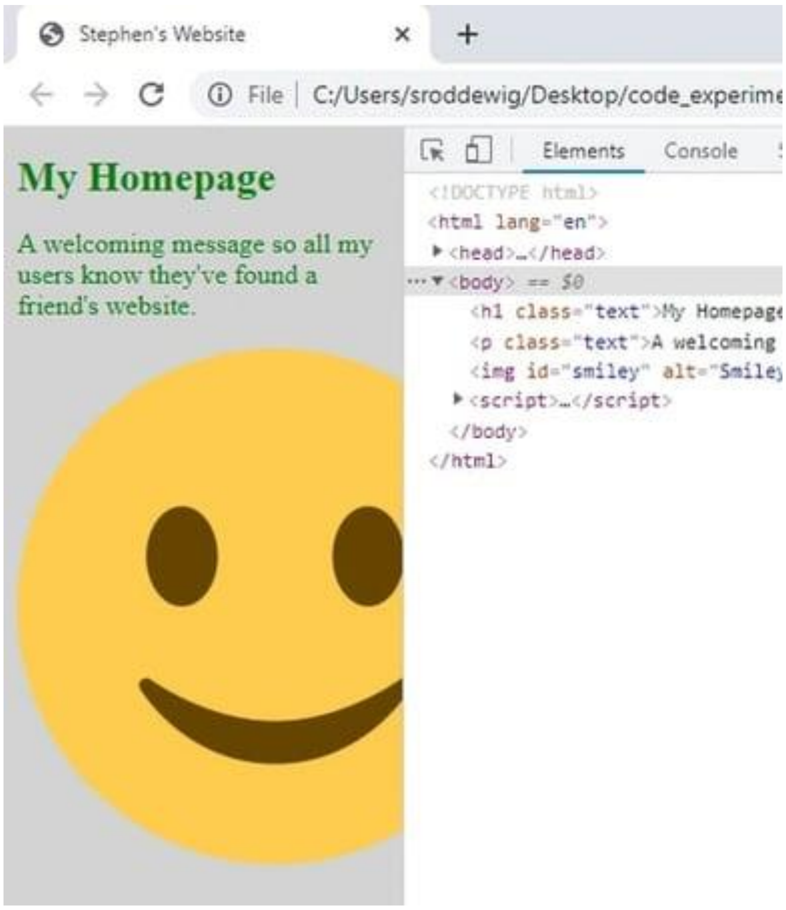
Pro Tip: Remember, for any specific styles you want to apply using classes or IDs in your CSS, you'll need to add those classes or IDs to the corresponding elements in your HTML.

5. Code a responsive website or a static website.

To understand web responsiveness, we will turn once again to a metaphor. On one hand, we have a pond. The water inside the pond is always the same shape because the edges of the pond never change. On the other hand, we have a stream. The edges of the stream are always changing as it flows, and the water is always expanding and narrowing to fit that shape.

I like to think of static and responsive websites in the same way. The static website does not react to a change in the browser's size. Its contents will always be the same shape. Meanwhile, the responsive website constantly adapts to different browsers and screen sizes.

Until now, we have created a static website. The text elements will resize by default, but other elements will keep their original shape and size no matter how the display changes. Below, we can see our smiley face's right side is cut off because the viewport is now narrower than its fixed width of 300 pixels (note that our paragraph has wrapped its text to fit).



My Homepage

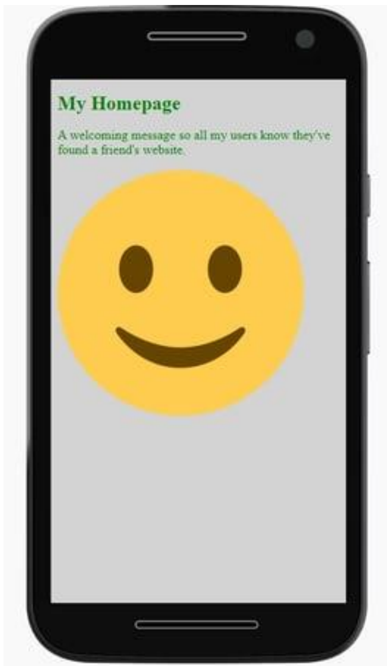
A welcoming message so all my users know they've found a friend's website.

```
<!DOCTYPE html>
<html lang="en">
 ▶<head>…</head>
…▼<body> == $0
    <h1 class="text">My Homepage
    <p class="text">A welcoming
    <img id="smiley" alt="Smile
   ▶<script>…</script>
  </body>
</html>
```

Meta Viewport Tag

Why does responsiveness matter? The first time I viewed my site on a phone and had to zoom in to read anything, I knew something had to change. That's when I learned about the **<meta name="viewport" content="width=device-width, initial-scale=1">** tag.

Add this tag to your HTML's <head>. It ensures that your site scales correctly on any device.


My Homepage
A welcoming message so all my users know they've found a friend's website.

This is already looking better, and now we can be sure our page will behave consistently across devices and screen sizes, which will save us a lot of frustration. We can now optimize our web page for mobile using media queries and fluid design practices.

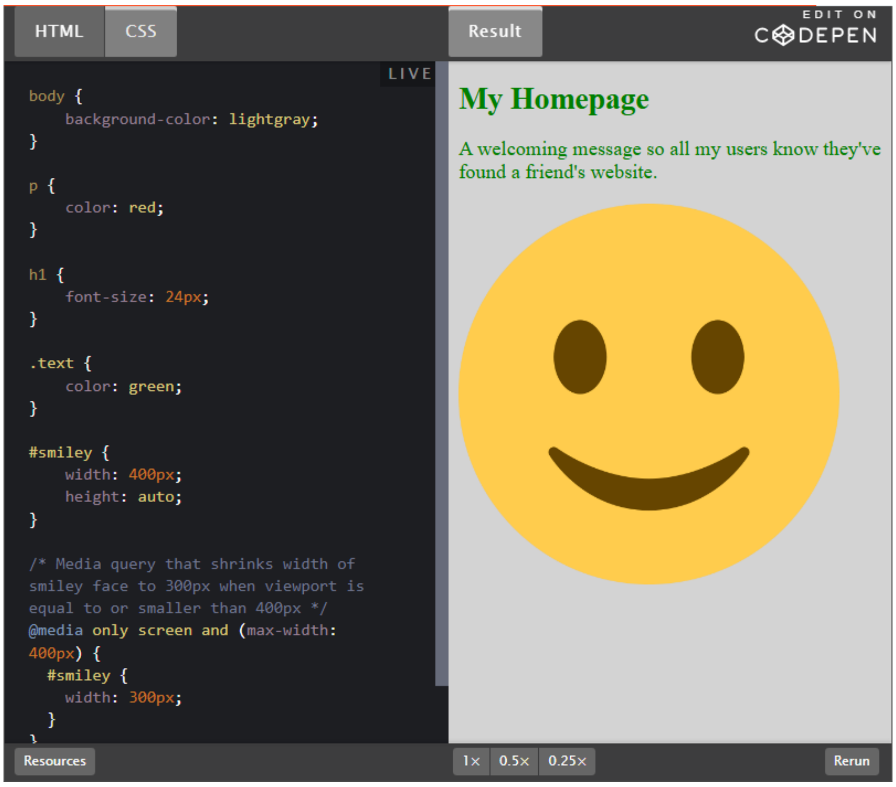
Media Queries

A media query is a CSS rule that instructs the CSS to behave differently based on the width of the viewport (AKA the screen/browser width the web page is displaying in). You define that width and how the CSS changes.

For example, I‘ve increased our smiley face’s width to 400 pixels so that it has more presence on larger screens. However, I want to make sure it‘s not cut off on smaller screens. So, I wrote a media query that states that once the screen is 400 pixels wide or smaller, reduce the smiley face’s width to 300 pixels.

```
@media only screen and (max-width: 400px) { #smiley { width: 300px; } }
```

I've added the media query to the end of our CSS stylesheet. Make sure they always come *after* your regular CSS rules and group them sequentially. For example, a 500 pixel-media query comes before a 400-pixel media query.



```
body {
    background-color: lightgray;
}

p {
    color: red;
}

h1 {
    font-size: 24px;
}

.text {
    color: green;
}

#smiley {
    width: 400px;
    height: auto;
}

/* Media query that shrinks width of
smiley face to 300px when viewport is
equal to or smaller than 400px */
@media only screen and (max-width:
400px) {
  #smiley {
    width: 300px;
  }
```

My Homepage

A welcoming message so all my users know they've found a friend's website.

HTML:

```
<!DOCTYPE html>
<html lang="en">
   <head>
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <link rel="stylesheet" href="style.css">
      <title>Stephen's Website</title>
      <meta charset="UTF-8">
   </head>
   <body>
      <h1 class="text">My Homepage</h1>
      <p class="text">A welcoming message so all my users know they've found a friend's website.</p>
      <img id="smiley" alt="Smiley face" src="https://assets.codepen.io/6093409/smiley-face.png">
   </body>
</html>
```

CSS:

```
body {

    background-color: lightgray;

}



p {

    color: red;
}

h1 {
    font-size: 24px;
}

.text {
    color: green;
}

#smiley {
    width: 400px;
    height: auto;
}

/* Media query that shrinks width of smiley face to 300px when viewport is equal to or smaller than 400px */
@media only screen and (max-width: 400px) {
  #smiley {
    width: 300px;
  }
}
```

Result:

My Homepage

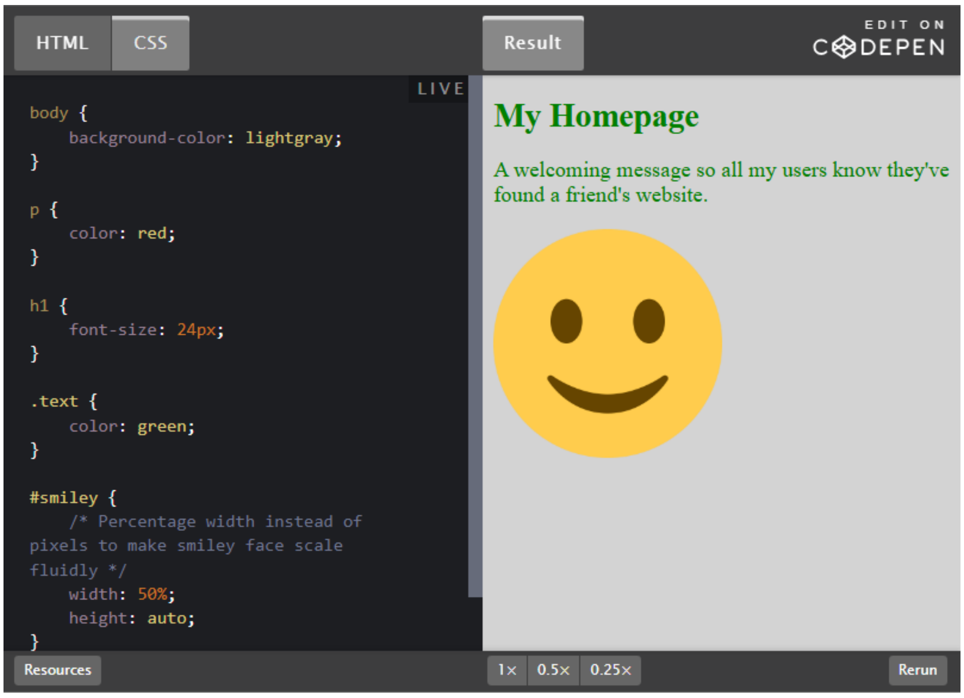
A welcoming message so all my users know they've found a friend's website.

Fluid Design

Media queries work best if we want to keep our elements at fixed widths. As you saw above, I wanted my smiley face to display 400 pixels wide and then scale back to 300 pixels. However, what happens when the screen reaches a 300-pixel width? I would need another media query to reduce its size again.

An alternative approach is fluid design. Fluid design means using relative units like percentages instead of fixed pixels. It's like making sure your picture takes up 50% of the screen, no matter how big or small that screen is. This approach saves you from writing multiple media queries. It's all about making your website flow like water, fitting into whatever space it's given.

In my code, the environment for the smiley face image element is the body, which is a container that spans the entire page. The relative unit we'll use is the percentage to set my smiley face equal to that proportion of the body's width. For example, if the body is 400 pixels wide and the smiley face is set to 50% of the container's width, then it is 200 pixels wide.



```
body {
    background-color: lightgray;
}

p {
    color: red;
}

h1 {
    font-size: 24px;
}

.text {
    color: green;
}

#smiley {
    /* Percentage width instead of
pixels to make smiley face scale
fluidly */
    width: 50%;
    height: auto;
}
```

My Homepage

A welcoming message so all my users know they've found a friend's website.

HTML:

```
<!DOCTYPE html>
<html lang="en">
   <head>
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <link rel="stylesheet" href="style.css">
      <title>Stephen's Website</title>
      <meta charset="UTF-8">
   </head>
   <body>
      <h1 class="text">My Homepage</h1>
      <p class="text">A welcoming message so all my users know they've found a friend's website.</p>
      <img id="smiley" alt="Smiley face" src="https://assets.codepen.io/6093409/smiley-face.png">
   </body>
</html>
```

CSS:

```
body {

    background-color: lightgray;

}



p {

    color: red;
}


h1 {
    font-size: 24px;
}


.text {
    color: green;
}


#smiley {
    /* Percentage width instead of pixels to make smiley face scale fluidly */
    width: 50%;
    height: auto;
}
```

Result:

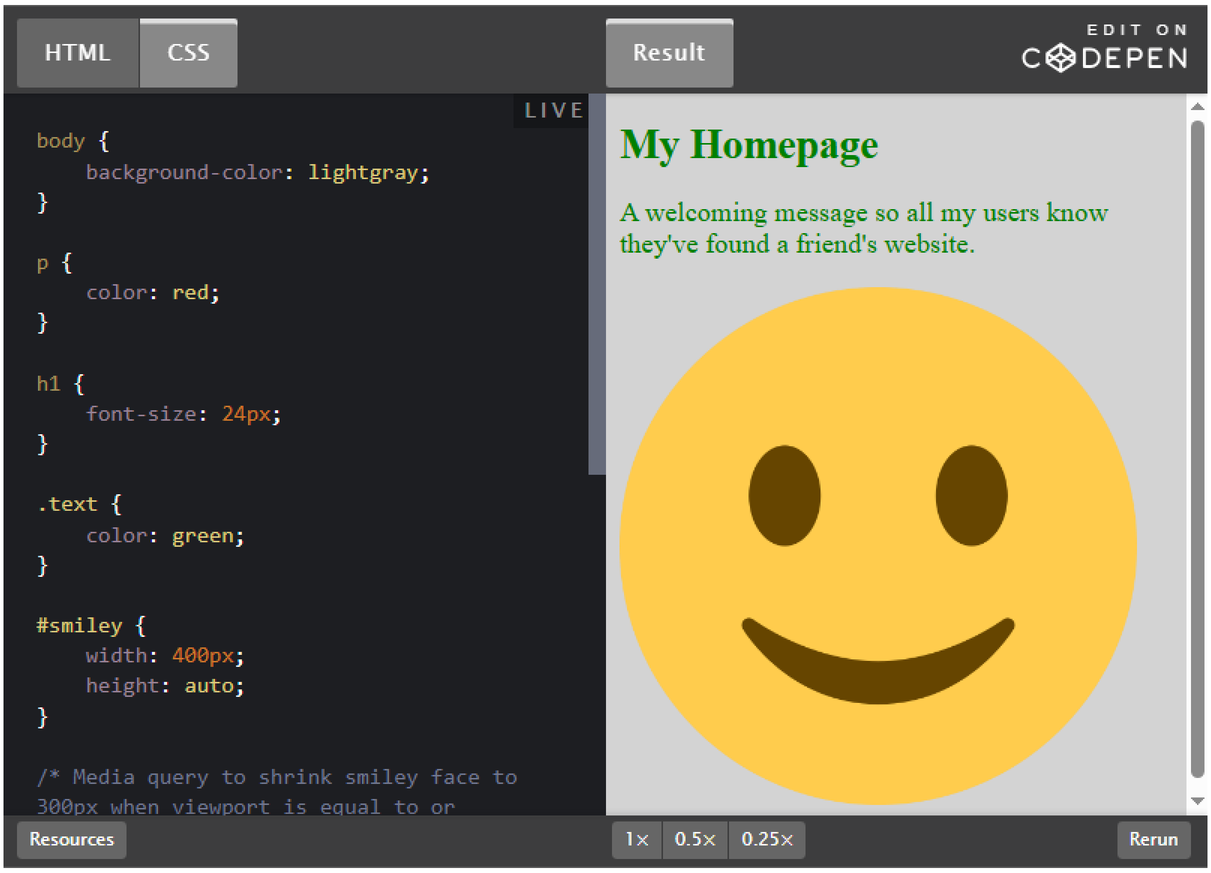
My Homepage

A welcoming message so all my users know they've found a friend's website.

Now I no longer need a media query; the smiley face will always scale to fill the container to the amount specified. In other words, it's *fluid*. However, don't write media queries off just yet.

Using Media Queries and Fluid Design

Though they may seem like opposites, media queries and fluid design pair well to achieve responsiveness. A favorite trick of mine is to use the CSS property "max-width" set to a maximum pixel count along with a "width" set to a percentage. Take the example below.



```
body {
    background-color: lightgray;
}

p {
    color: red;
}

h1 {
    font-size: 24px;
}

.text {
    color: green;
}

#smiley {
    width: 400px;
    height: auto;
}

/* Media query to shrink smiley face to
300px when viewport is equal to or
```

My Homepage

A welcoming message so all my users know they've found a friend's website.

HTML:

<!DOCTYPE html>

<html lang="en">

    <head>

        <meta name="viewport" content="width=device-width, initial-scale=1">

        <link rel="stylesheet" href="style.css">

        <title>Stephen's Website</title>

        <meta charset="UTF-8">

```
    </head>

    <body>

        <h1 class="text">My Homepage</h1>

        <p class="text">A welcoming message so all my users know they've found a friend's website.</p>

        <img id="smiley" alt="Smiley face" src="https://assets.codepen.io/6093409/smiley-face.png">

    </body>

</html>
```

CSS:

```
body {

    background-color: lightgray;

}



p {

    color: red;

}



h1 {

    font-size: 24px;

}



.text {

    color: green;

}
```

```
#smiley {

    width: 400px;

    height: auto;

}



/* Media query to shrink smiley face to 300px when viewport is equal to or smaller than
400px */

@media only screen and (max-width: 400px) {

  #smiley {

    width: 300px;

  }

}



/* Media query to set smiley face's width to percentage when viewport is equal to or smaller
than 300px */

@media only screen and (max-width: 300px) {

  #smiley {

    max-width: 300px;

    width: 100%;

  }

}
```

Result:

My Homepage

A welcoming message so all my users know they've found a friend's website.

I have my smiley face initially at a fixed width of 400 pixels. Then I scale it back to 300 pixels when the viewport reaches 400 pixels. Then I set it to a maximum width of 300 pixels but to scale down to always be 100% of the available width of the container for any screens *smaller* than 300 pixels.

6. Code a simple website or an interactive website.

We will (you guessed it) bring back our friend the metaphor for this section. Imagine you‘re standing on a street reading a flier for guitar lessons on a brick wall. You’re interested, and the flier says to call a number to sign up.

This is what we'd consider a simple website. It displays information about something, it may even ask you to take an action, but reading is the extent of what you (the user) can do on the site. Remember, you need to call the number it gives you on your phone.

We‘re back at the brick wall. Now, there’s a sign talking about guitar lessons and a door to enter the studio to sign up if you‘re interested. The wall is still displaying information, but now you can interact with it to take the next step. This is what we’d consider an *interactive* website. See where the name comes from?

Enter JavaScript

Up to this point, we've been crafting a website that's like our flyer – static and informative. But the web today is all about interaction, and that's where JavaScript comes in.

JavaScript is what adds life to your site. It's like turning a static painting into a movie. It allows your website to respond to user actions, like clicks, scrolls, or keyboard entries.

Your First JavaScript Program

Unfortunately, it is outside of the scope of this lesson to dive into the syntax and mechanics of JavaScript. Instead, I will provide a pre-written program so that we can focus on implementing it within our website.

```
const smiley = document.querySelector("#smiley"); smiley.addEventListener("click",
() => { alert(`Smiley says "Welcome!"`); });
```

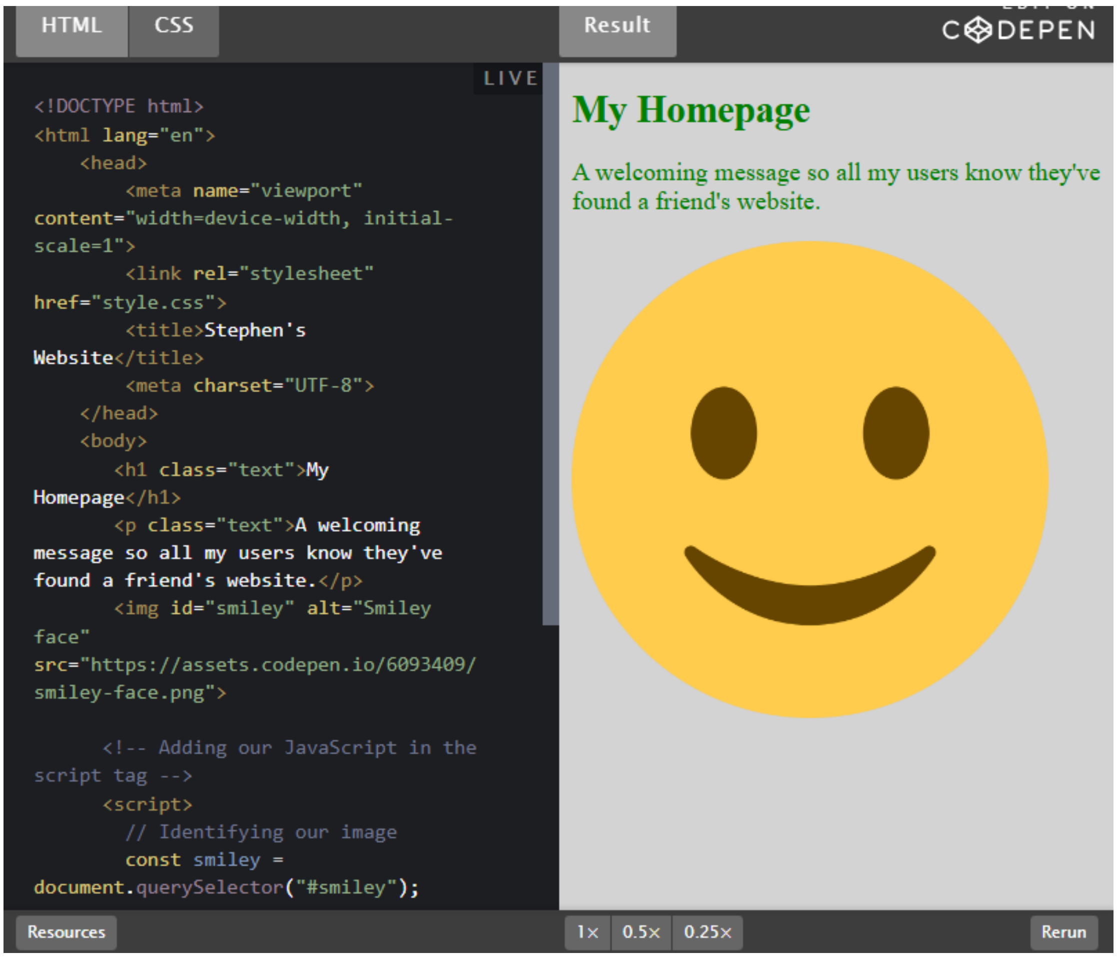
This code finds an image with the ID "smiley" and makes something happen when it's clicked – in this case, a welcoming message pops up.

Now let me show it to you in action.

Adding JavaScript to Your Page

To incorporate JavaScript, you'll use a <script> tag in your HTML. This is usually placed right before your closing body tag (</body>). It's like telling your website, "Hey, there's something more than just plain text and style here."

Here’s an example of how you might include it:



```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta name="viewport"
content="width=device-width, initial-
scale=1">
        <link rel="stylesheet"
href="style.css">
        <title>Stephen's
Website</title>
        <meta charset="UTF-8">
    </head>
    <body>
       <h1 class="text">My
Homepage</h1>
       <p class="text">A welcoming
message so all my users know they've
found a friend's website.</p>
       <img id="smiley" alt="Smiley
face"
src="https://assets.codepen.io/6093409/
smiley-face.png">

      <!-- Adding our JavaScript in the
script tag -->
      <script>
        // Identifying our image
        const smiley =
document.querySelector("#smiley");
```

My Homepage

A welcoming message so all my users know they've found a friend's website.

HTML:

```
<!DOCTYPE html>

<html lang="en">

   <head>

      <meta name="viewport" content="width=device-width, initial-scale=1">

      <link rel="stylesheet" href="style.css">

      <title>Stephen's Website</title>

      <meta charset="UTF-8">

   </head>

   <body>

     <h1 class="text">My Homepage</h1>
```

```
        <p class="text">A welcoming message so all my users know they've found a friend's
website.</p>

        <img id="smiley" alt="Smiley face" src="https://assets.codepen.io/6093409/smiley-
face.png">



      <!-- Adding our JavaScript in the script tag -->

      <script>

        // Identifying our image

        const smiley = document.querySelector("#smiley");



        // Adding reaction when image is clicked

        smiley.addEventListener("click", () => {

          alert(`Smiley says "Welcome!"`);

        });

      </script>



    </body>

</html>
```

CSS:

```
body {

    background-color: lightgray;

}



p {

    color: red;

}
```

```
h1 {
    font-size: 24px;
}

.text {
    color: green;
}

#smiley {
    width: 400px;
    height: auto;
}

@media only screen and (max-width: 400px) {
  #smiley {
    width: 300px;
  }
}

@media only screen and (max-width: 300px) {
  #smiley {
    max-width: 300px;
    width: 100%;
  }
```

}

Result:

My Homepage

A welcoming message so all my users know they've found a friend's website.

And just like that, you've stepped into the world of interactive web design. Remember, a website can start simple, but with a bit of JavaScript, you can elevate it to a whole new level of engagement. Trust me, the first time I added an interactive element to my site, it felt like unlocking a new achievement in web design.

Next Steps in Building Your Website

If you have made it this far, then take a moment to congratulate yourself. You have gone from perhaps never coding before to having the foundation of a website that is both responsive *and* interactive. We've covered a lot of ground between different concepts in HTML and CSS and even touched on programming.

Now, let me share a bit about where you can go from here. This journey is far from over, and trust me, the road ahead is exciting.

Keep learning HTML and CSS.

HTML and CSS have more complexity than what we‘ve covered today. These languages form the visible face of your website and are critical to your users’ experience. I've relinked the foundational guides to HTML and CSS below as well as added some links to more advanced CSS topics. You can always look for the best coding boot camps that can provide extra support to teach you HTML and CSS.

Foundational Guides:

- Ultimate Guide to HTML: a one-stop shop for everything you need to know to write HTML like a pro
- Ultimate Guide to CSS: a one-stop shop to start using CSS to make your website wow your visitors

Best For: Web designers and developers who want to enhance their website's visual appeal and usability. Ideal for those aiming to create more engaging and aesthetically pleasing designs.

Advanced CSS Topics:

- Float: a simplified method for arranging HTML elements to the left, right, or center of a page
- Grid: a rigid layout system for arranging content on a page in rows and columns that offers more customization than flexbox
- Flexbox: a fluid layout system for arranging content in rows *or* columns that is simpler to implement than grid but offers less control over contents' positions
- Animations: a method for adding flair to your page by animating HTML elements for different motions or visual effects

Best For: Web designers looking to add dynamic, eye-catching elements to their sites. Ideal for those wanting to learn how to create engaging user experiences.

Learn JavaScript.

JavaScript was a turning point for me. It's where websites start to come alive, moving from static pages to interactive experiences. Whether it's adding a simple animation or building an entire web app, JavaScript opens up a world of possibilities. Don't be intimidated by its power; embrace it.

Learn Git and GitHub.

Learning Git and GitHub was like learning to save my game in the coding world. Git lets you keep track of changes and experiment without fear of losing your work. And GitHub? It's like a social network for your code, where you can store it safely and collaborate with others. The first time I successfully merged a branch in Git, I felt like I unlocked a new level in my coding skills.

Remember, you're just getting started. In less than three years, I went from fumbling through my first lines of HTML to guiding others on their coding journey. Stay curious, keep experimenting, and practice regularly. Coding is a journey, not a sprint. I can't wait to see where your newfound skills take you.