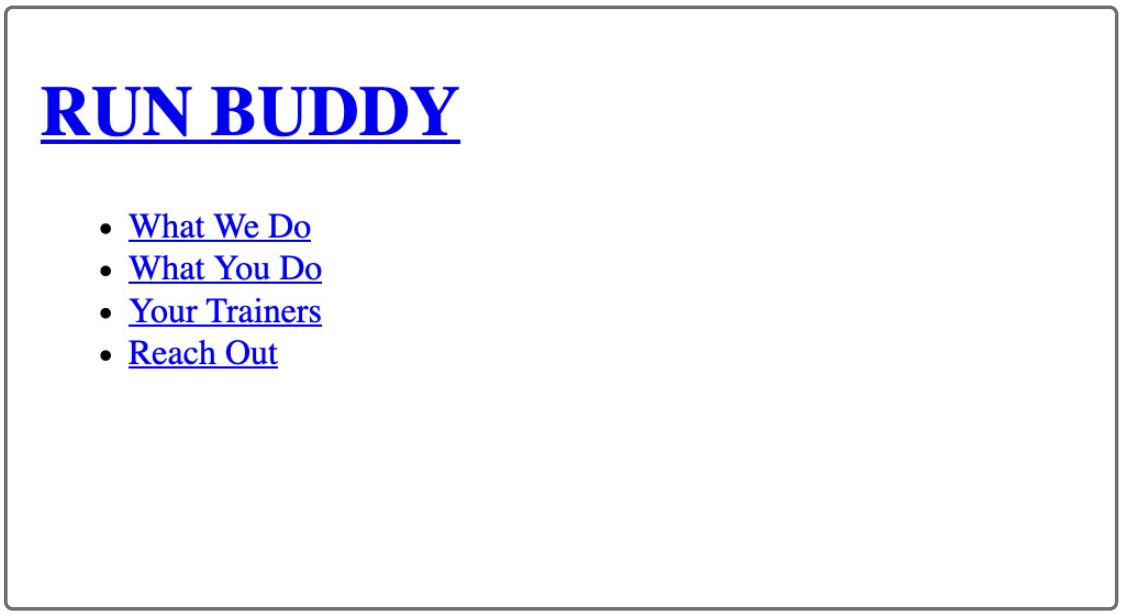
1.2.3 Add Content to the Header

This image shows the basic content that we'll add to the `<header>` element:



We'll begin by adding two things that most modern webpages have: **branding** and **navigation**.

These two important components are typically contained in the header and footer so that they appear on every page in a uniform way. Think about a website you're familiar with: does the top of the site look the same

on every page you visit? You betcha! (That's branding.) And where do you look when you want to navigate to another page? Typically, you look for a menu on the top and/or bottom of the webpage. (That's navigation.)

Add Branding

Branding refers to the logo, colors, and general style (aka "brand") of a company or organization. Think about Target's red bullseye or Amazon's orange arrow; if you visit these websites, you will always see their brand loud and clear at the top (and usually bottom) of the page.

We started to add branding content in Lesson 1 when we added `<h1>RUN BUDDY</h1>` to the `<header>`. This put the Run Buddy name at the top of every page. We will style it later in this lesson.

Add Navigation

Okay, so branding content is taken care of, but what about navigation?

Open `index.html` in VS Code, and add `<nav>` and `</nav>` immediately below `<h1>` but above the closing `</header>` tag.

It should look something like this:

```
<header>
  <h1>RUN BUDDY</h1>
  <nav></nav>
</header>
```

The `<nav>` element is a newer **semantic element** introduced in HTML5. The browser can read these elements and get an idea about the information that's inside, which aids both accessibility and SEO.

What are accessibility and SEO, you ask? Read on:

- **Accessibility** is the practice of adding HTML elements in a way that allows tools such as screen readers to explain the page's content in a meaningful way to those who have a disability such as vision or hearing loss.

- **SEO**, or **search engine optimization**, is the process of maximizing the number of visits a website receives and how frequently it shows up in search results.

DEEP DIVE

DEEP DIVE

There are a number of tools for adding accessibility to webpages. The most prominent and easy to implement is HTML5's semantic elements. Before HTML5, the `<div>` element was used for all blocks of content on a webpage. This forced developers to work extra hard to identify content for screen readers. Now, semantic elements like `<header>`, `<footer>`, and `<nav>` work just like a `<div>` element, but they have more meaningful names that tell the browser about the content inside them.

For more information, see the **MDN web docs on web accessibility (https://developer.mozilla.org/en-US/docs/Web/Accessibility)**.

There are a number of ways to organize navigation. We'll use the popular method of creating a list of navigation links.

In the `index.html` file, type the following right after the opening `<nav>` you just created:

```
<!-- Unordered list element -->
<ul>
  <!-- List item element-->
  <li>
    <!-- Anchor element -->
    <a href="#what-we-do">What We Do</a>
  </li>
  <li>
    <a href="#what-you-do">What You Do</a>
  </li>
  <li>
    <a href="#your-trainers">Your Trainers</a>
  </li>
  <li>
    <a href="#reach-out">Reach Out</a>
  </li>
</ul>
```

Let's unpack what we just created, starting with describing what a list is.

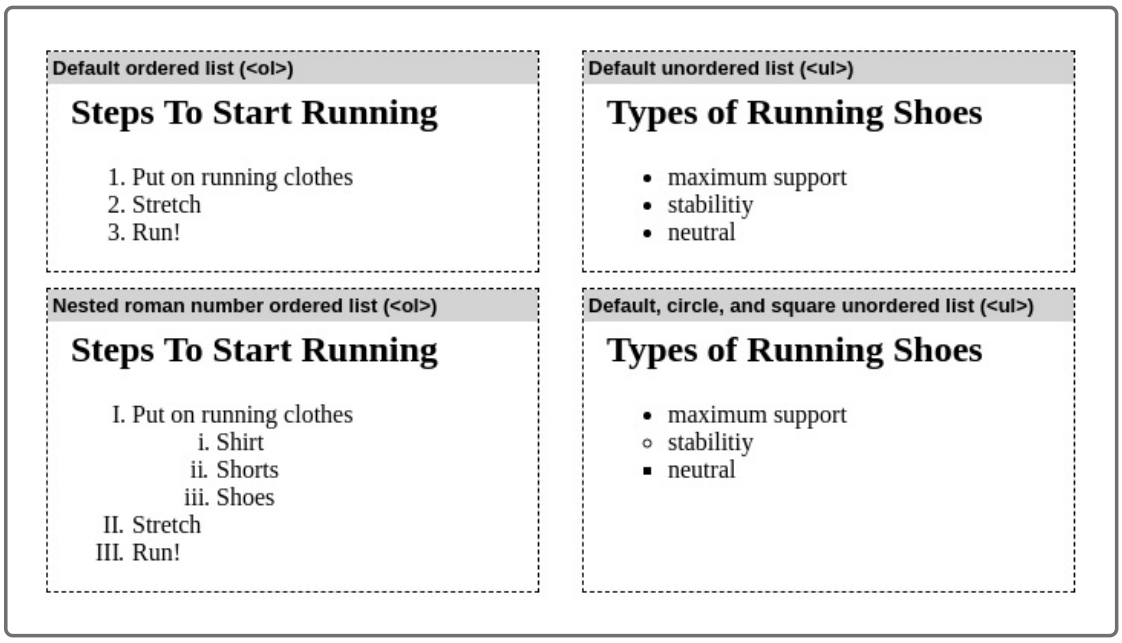
Lists

In HTML, there are two types of lists: ordered and unordered.

- **Ordered lists**, denoted with the `<ol>` element, tell the browser to interpret any nested list item (denoted with the `<li>` element) in numeric order (1, 2, 3, etc.).

- **Unordered lists** (`<ul>`) allow for a more loosely formatted list by marking each nested list item with a symbol (such as a bullet point or square).

The following image shows examples of ordered and unordered lists:



The relationship between a `<ul>` or `<ol>` element to an `<li>` is a direct parent/child relationship, meaning that an `<li>` (or child) should be nested or contained within the parent element's opening and closing tags—in this case, the `<ol>` or `<ul>` elements.

Within reason, anything can be nested in an `<li>` element. In our case, we created a link in each list item using an anchor element (`<a>`). Anchor elements are used to create links that take us to other destinations when clicked. These destinations can be within the same page, on another page in our website, or on another website entirely.

CONNECT THE DOTS

The `<a>` element is a prime example of the "hypertext" in Hypertext Markup Language (HTML). Hypertext is defined as text that links to other text. Since this term was coined in the early 1960s, its definition has expanded to include other types of media as well (such as images and videos).

Here are a few examples of the `<a>` element being used to create links:

```
<!-- When you click "Go to Google," it will take you to Google -->
<a href="https://google.com">Go to Google</a>

<!-- This will take us to our own site's privacy policy HTML page -->
<a href="./privacy-policy.html">View Our Privacy Policy</a>

<!-- This does nothing at all -->
<a>This does nothing</a>
```

Attributes

Did you notice that the first two examples above have an `href=` inside their opening tags? This is an HTML **attribute**.

Attributes can be used to give an HTML element a unique identity, create relationships with other elements, and provide design changes using CSS. They give functionality, meaning, and context to HTML elements.

Attributes aren't necessary for every element, but some do require them. One of those is the `<a>` element. If we were to omit the `href` attribute in the examples above, the links would not work.

Here are some popular attributes:

- `id`: This is a unique identifier for an HTML element. The value of this can only be used once per HTML document.
- `class`: This is another way of identifying an HTML element, but its value is expected to be more general and can be reused across multiple HTML elements on a page.
- `title`: Not to be mistaken for the `<title>` element, the `title` HTML attribute holds a value that appears as a small pop-up (known as a tool-tip) when the mouse is hovered over an element for a period of time.

DEEP DIVE

DEEP DIVE

Having a reference to all the HTML attributes and their uses and limitations will come in handy. For more information, see **the MDN web docs on HTML attributes (https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes)** and consider bookmarking the page for future reference.

The values associated with the `href` attributes in the navigation we just added give us the ability to jump to a certain spot on our current page. The syntax `href="#what-we-do` tells the browser that when that `<a>` element is clicked, go find another HTML element on the page with the attribute `id="what-we-do"`. We haven't added that attribute yet, but we will soon.

Now that we know what `<a>` elements are and how they're used, let's implement a best practice and make the branding in the `<header>` of our page clickable to take the user back to the homepage. We can do this by wrapping the content between the `<h1>` tags with its own `<a>` having an `href` value of `"/"`, like this:

```
<h1>
  <a href="/">RUN BUDDY</a>
</h1>
```

The value of the `href` used here—a forward slash (`"/"`)—will always represent the path to the topmost directory of an application or project. In this case, when a user clicks the `<a>` element, they will be taken to the

topmost directory. And because there is no file specified, the `index.html` file will be loaded.

REWIND

Think back to Lesson 1 when you learned that if no specific file is being looked for, a computer will try to provide an "index" page. In web development, this is why the homepage has a filename of `index.html`.

If the code inside the `<header>` element now looks like the following, you're ready to move on and make the landing page look good!

```
<header>
  <h1>
    <a href="/">RUN BUDDY</a>
  </h1>
  <nav>
    <ul>
      <li>
        <a href="#what-we-do">What We Do</a>
      </li>
      <li>
        <a href="#what-you-do">What You Do</a>
      </li>
      <li>
        <a href="#your-trainers">Your Trainers</a>
      </li>
      <li>
        <a href="#reach-out">Reach Out</a>
      </li>
    </ul>
  </nav>
</header>
```

DEEP DIVE ▲

DEEP DIVE

Learn more about the HTML elements we just used:

- MDN web docs on the anchor element <a> (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/a)
- MDN web docs on heading elements <h1>–<h6> (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements)
- MDN web docs on navigation <nav> (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/nav)
- MDN web docs on unordered list <ul> (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/ul)
- MDN web docs on list item <li> (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/li)

Commit to Git and Push to GitHub

You just did a decent amount of work so you should save it and push it up to GitHub for safekeeping. Using the commands you learned in the last

lesson, go ahead and do this now.

Here's a refresher of the Git commands you should use to commit your code locally:

1. `git add -A`: Use this to tell Git to pick up (also known as "track") any new, edited, or removed files in your project.

2. `git commit -m "commit message"`: Then tell Git what you added or changed (replace `"commit message"` with a short note about what changed).

SHOW HINT

3. `git push origin master`: Use this command to take all of the code and file edits from your recent `commit` and push it all up to GitHub.

Now your code is saved locally on your machine and on your GitHub profile. If you visit your GitHub Pages link at `<username>.github.io/run-buddy` (replace `<username>` with your GitHub username), you'll be able to see how the new HTML looks when viewed in a browser!

NERD NOTE

We've seen the `#` syntax in action with `href` values, but you will use it in many more places throughout your career as a programmer. It is what's known as an **octothorpe**. For more information, see **Wikipedia's page on the many uses of the # (https://en.wikipedia.org/wiki/Number_sign#In_computing)** .

The HTML elements we've implemented so far do a great job at two things:

- Organizing content (e.g., the content between the `<h1>` tags in `<h1>RUN BUDDY</h1>`)
- Providing interactivity (e.g., the `<a>` element's `href` attribute taking the user somewhere else when they click on the link)

But there is more we need to do: make our content and interactivity look good!

Looks Matter

One question that HTML used to be able to answer was "How do I look?". Before CSS, HTML handled all of the design aspects of a website in addition to the content and structure. Developers had a very limited set of options—they could change the color of text, add a background color, give images height and width dimensions, but not a heck of a lot else.

Most advanced (for the time) designs were achieved by taking a mock-up of the design and slicing it into a bunch of small images, then laying them all into an HTML table element to build the image on the webpage. Think about how difficult it would be to create a complex design by chopping up an image and placing the pieces into Microsoft Excel cells. As you can imagine, this wasn't very much fun.

LEGACY LORE

Most older website designs have been removed from the internet, but we do know of one that remains: **the website for the 1996 movie "Space Jam" (https://www.spacejam.com)**. This website attempts a creative layout, but the limitations of HTML can't take it any further than static content and a rudimentary design.

As developers began to want more control over their page designs, they realized that HTML might not be capable of doing the job. Thus, a new language was created to take some of the burden off of HTML when it came to presentation and design, and that language is CSS.