Exploration — Introduction to CSS and Frontend Design

Introduction

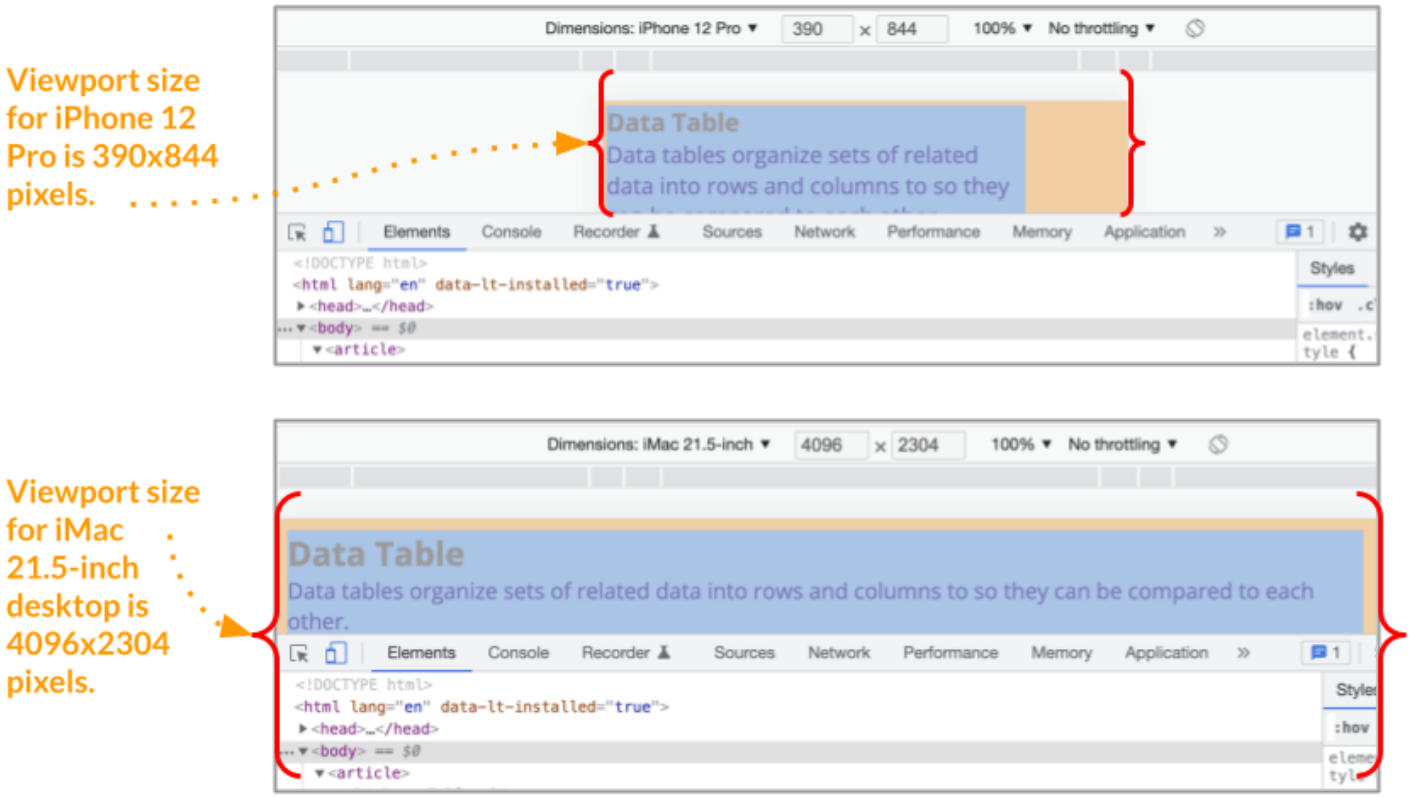
While HTML **semantic** (logical) markup gives webpages basic structure and organized content, and JavaScript gives them dynamic components, it is **Cascading Style Sheets** (CSS) that redefine how that structure, the content, and the components should look and behave to improve usability, readability, legibility, and adhere to brand requirements.

HTML elements are predefined with style, which the CSS language can override. CSS uses its own syntax to make those revisions and can be incorporated in a variety of ways. In this exploration and the next, we'll explore how to redefine many of the default HTML elements with the syntax of CSS rules while incorporating typical methods for page design, table design, and form design.

Browser Viewport

The first visible HTML element in a web page is the `<body>` element. The `<body>` displays the contents of the browser's ***viewport***, which is the portion of a browser's window/tab below the address and bookmark bar. It appears inside of the window frame. The viewport area changes size depending on the device being used to view web pages.

For example, when selecting the `<body>` element via the browser's **Web Dev/Inspect** tools, the viewport is highlighted and dimensions displayed. These two devices, iMac and iPhone have different viewport sizes:



Front-end web developers typically inspect the viewports for different devices to ensure their apps/sites display as they intended.

Default HTML Styles

The `<body>`'s default styles include:

- Times New Roman font
- 8 pixels of margin
- 0 vertical height (vh)

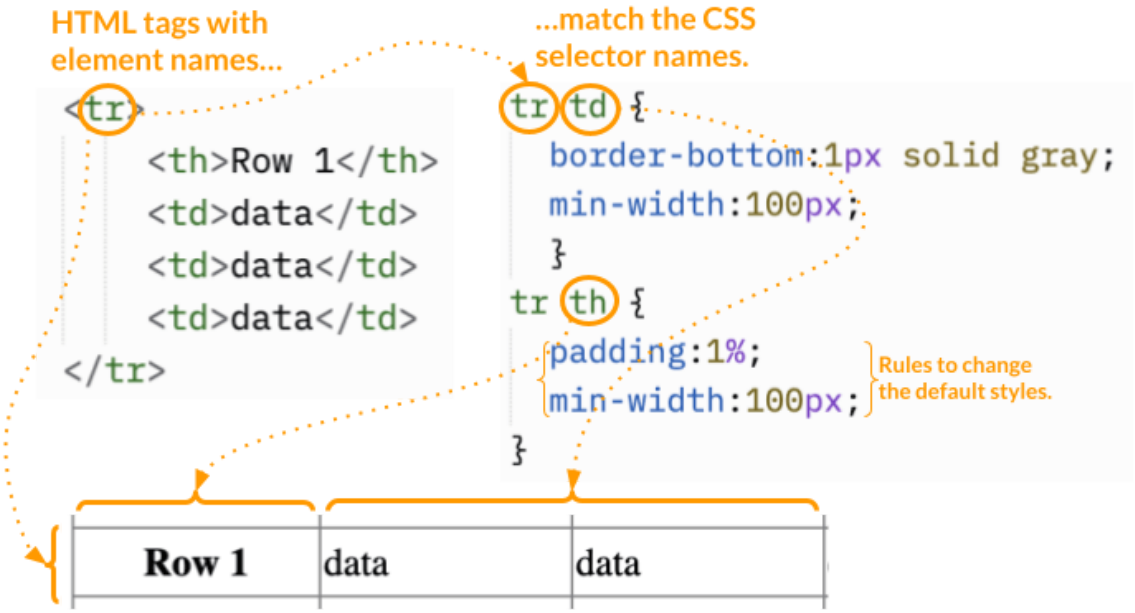
Since HTML headings and the paragraph element inherit the `<body>` style, their default fonts are also set to `Times New Roman`. The heading `<h1>` also includes a larger font size of `2em`*, a top and bottom margin of `.67em`s, and, a font-weight of `bold`. The remaining headings `<h2>` to `<h6>` inherit those rules but decrease in size. Some HTML elements, such as the semantic page layout tags `<header>`, `<nav>`, `<main>`, `<section>`, `<article>`, `<aside>`, and `<footer>` are empty 'blocks' with no dimensions. It is the job of the web designer/developer to give them dimension with placement, spacing, color, imagery, and font updates.

(To learn about each tag and its default style, visit **W3Schools.com (https://www.w3schools.com/tags/tag_aside.asp)**. At the bottom of each HTML tag page is a list of default styles for that tag.)

* The unit *em* is the width of the letter *m* as represented by a font. We'll learn more about units in the next Exploration.

CSS Selectors and Syntax

A CSS selector, in its most basic form, is the element name used in an HTML tag. Rules for changing the default style of that element are provided with the selector, like this:



Other selector types are available for more complex rule changes, which you can see here in **Mozilla's Reference: (https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors#reference_table_of_selectors)**

Selectors and Examples	
Type selector (https://developer.mozilla.org//en-US/docs/Web/CSS/Type_selectors)	`h1 { }`
Universal selector (https://developer.mozilla.org//en-US/docs/Web/CSS/Universal_selectors)	`* { }`
Class selector (https://developer.mozilla.org//en-US/docs/Web/CSS/Class_selectors)	`.box { }`
ID selector (https://developer.mozilla.org//en-US/docs/Web/CSS/ID_selectors)	`#unique { }`
Attribute selector (https://developer.mozilla.org//en-US/docs/Web/CSS/Attribute_selectors)	`a[title] { }`

Selectors and	Examples
Pseudo-class selectors (https://developer.mozilla.org//en-US/docs/Web/CSS/Pseudo-classes)	`p:first-child { }`
Pseudo-element selectors (https://developer.mozilla.org//en-US/docs/Web/CSS/Pseudo-elements)	`p::first-line { }`
Descendant combinator (https://developer.mozilla.org//en-US/docs/Web/CSS/Descendant_combinator)	`article p`
Child combinator (https://developer.mozilla.org//en-US/docs/Web/CSS/Child_combinator)	`article > p`
Adjacent sibling combinator (https://developer.mozilla.org//en-US/docs/Web/CSS/Adjacent_sibling_combinator)	`h1 + p`
General sibling combinator (https://developer.mozilla.org//en-US/docs/Web/CSS/General_sibling_combinator)	`h1 ~ p`

We'll use a few of these in the next Exploration.

Basic syntax for stylesheets requires the following:

- Selectors are followed by a set of curly braces {}: `selector {}`.
- Inside the curley braces are pairs of properties separated from values with a colon: `selector {property:value;}`.
- Sets of property:value pairs are separated with semicolons: `selector {property:value; property:value;}`.
- Multiple selectors are separated with commas but the last one in the list must not have a comma after it: `selector1, selector2, selector3 {property:value;}`.
- Values with multiple subvalues are separated with commas, such as in a font-family list `font-family:'Ubuntu', sans-serif;` and color values `color:rgb(000,000,000)`.
- Single `'` or double-quotes `"` are allowed when specifing resources such as font names and image files.
- Comments are wrapped with `/*` and `*/`.

5 Ways to Incorporate Styles

External CSS rules for redefining an element's selector are typically done in one or more external files which have a .css file extension. A stylesheet file is `<link>`ed in the global `<head>` area of a

website. For example, an `index.html` file can link to this main/global stylesheet, like this:

```
<link rel="stylesheet" href="App.css" />
```

...and provide styles to all pages of the website that link to that single, global stylesheet.

Other stylesheets for specific components can be <link>ed after the global .css file, or **imported** in that global stylesheet, like this:

```
@import "component1.css";
```

Externally linked .CSS files are typically the most efficient and prefered method to use for an app or website.

It is also possible to embed styles directly in HTML and JavaScript files, though that method is typically for one-off style changes only and will not be allowed in this course:

1. **Embedded** within a `<style><style>` tag in the area, like this:

```
<style>
    h1 {color:orange;}
</style>
```

2. **Inline** within an element using an attribute and value, like this:

```
<h1 style="color:orange">
<h1 color="orange" /> <!-- Too old to use! -->
```

3. In **JavaScript template literals** within a JavaScript function, like this:

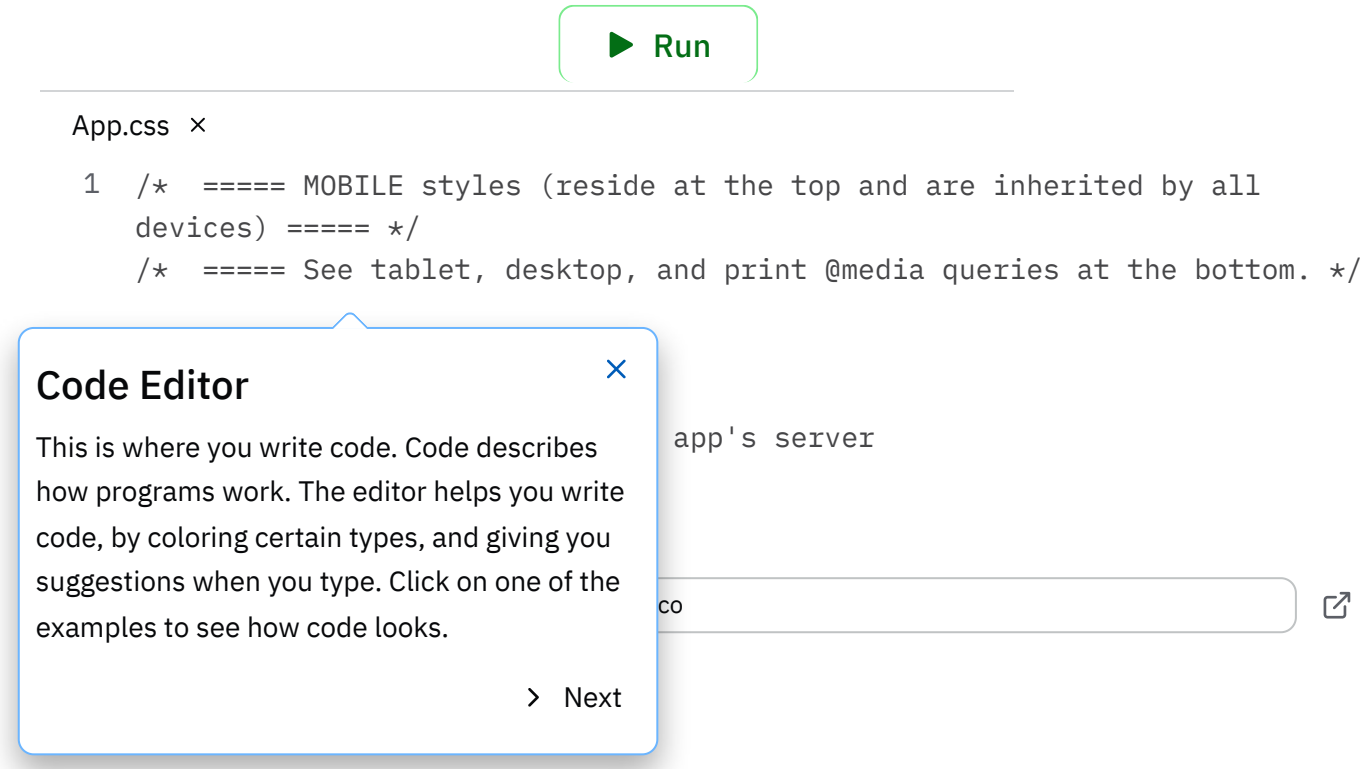
```
const orangeHeading = styled.h1`
      color: orange;
      `;
```

4. In **regular JavaScript**, like this:

```
document.getElementsByTagName('h1').style.color = 'orange';
```

CSS File Structure

A typical sheet includes the following basic selectors (in the Replit). You won't need all of them, but you can see the possibilities:



Access Denied

At the top of an **external** .css file, front-end developers typically import some dependencies, such as fonts from the server or host, the "normalize.css" stylesheet, component-specific stylesheet files, and

device-specific stylesheet files, like this:

```
@font-face {
    font-family: 'FontName';
    src: url('FontName.woff2') format('woff2');
  }

  @import url(//fonts.googleapis.com/css?family=Font+Name);

  @import-normalize;

  @import "component1.css";
```

Google Fonts (https://fonts.google.com/) is the most commonly-used web font service, with more than 1000 choices. Use of unique fonts support the brand of a site/app by differentiating it from others. Fonts can be served from a host or from the local server.

The **Normalize.css** (https://www.npmjs.com/package/modern-normalize) (updated link) file is an open-source set of style rules that update browser selector defaults, so that new and old browsers can start with the same rule conventions, creating cross-browser consistency. It also corrects bugs and improves usability.

The next rules of a global external stylesheet file typically include definitions for cellphone viewports via the `:root {}, html {}` and/or `body {}` rules. `@media` queries are typically added to the bottom of the file to accommodate other devices such as tablets, laptops, desktop computers, printers, projectors, etc. These queries help developers improve a site/app's accessibility and responsiveness. A typical set of queries starts like this, with new rules overriding previous rules within the correlating set of curly braces {}:

```
@media all and (min-width: 600px) {
  /* override rules above */
}
@media all and (min-width: 1080px) {
  /* override rules above */
}
@media print {}
```

Learn more at **A Complete Guide to CSS Media Queries** (https://css-tricks.com/a-complete-guide-to-css-media-queries/) and **Media Queries for Standard Devices** (https://css-tricks.com/snippets/css/media-queries-for-standard-devices/).

The `:root` controls the browser viewport (the <html> tag) but also allows rules for variables that can be reused in other rules, such as colors, sizes, borders, boxes, shadows, etc. Their syntax includes dashes and looks like this:

```
:root {
  --custom-color:orange;
  --custom-size:200%;
}
```

The `body {}` typically defines the page margin and padding around the perimeter of the viewport. You'll learn more about margin and padding in the next exploration. By default, the content will be left-justified just 8 pixels from all four sides of the viewport.

In the `body {}`, we also typically define rules for paragraphs and paragraph-level elements, which are inherited by headings, lists, figcaptions, and tables. Once a font is defined, its default size must be defined, then the height of one line of text, so we end up with a comfortable reading experience, like this:

```
body {
  color:var(--custom-color);
  font-family:'Font Name', sans-serif;
  font-size:1.2em;
  line-height:1.15;
}
```

Font sizes can be declared with a variety of units of measurement:

- **Pixels** (px), which are the number of pixels tall.
- **Absolute and Relative keywords**, such as small, medium, and large, or smaller, larger.
- **Percentage** (%) of the parent's font-size.
- **em, ex**, or **ch** units, which are based on the height of an "m", "x", or "0" zero in the font family. They are also relative to the parent element font-size.
- **rem** (relative units) are dependent on the :root, html, or body font size.
- **Viewport height and width** (vh and vw) are relative to the dimensions of the viewport, so that `1vw = 1% of viewport width` and `1vh = 1% of viewport height`

Learn more about **Font-size.** **(https://css-tricks.com/almanac/properties/f/font-size/)**

If **background** **(https://css-tricks.com/almanac/properties/b/background/)** images, gradients, or single colors are required behind the text, then they are typically defined in the :root, html {}, or `body{}` as well. For example:

```
body {
  background-attachment: fixed;
  background-image: url(img/grand-canyon.jpg);
  background-position: top left;
  background-repeat: no-repeat;
  background-size: cover;
  ...
}
```

Once the `body` has defined the font treatment, most other selectors will inherit that same family, size, color, and line height. Some selectors, such as a **form'** **(https://developer.mozilla.org/en-US/docs/Learn/Forms)** s `input, textarea, select`, and `button` *do not inherit* the font, so should be repeated in a section of the file related to forms, like this:

```
input, textarea, select, button {
  font-family:'Font Name', sans-serif;
```

```
  font-size:1.2em;
  color:var(--custom-color);
  line-height:1.15;
}
```

Other form selector rules can be defined as well, such as the `fieldset, legend, label, meter,` and `progress`. In some cases, a developer will want to change how default input types are displayed, so they apply the attribute pseudo-class, like this:

```
label input[type="checkbox"],
label input[type="radio"] {
  display:inline!important;
  min-width:15px;
}
```

This video covers some but not all concepts related to anchors and navigation.

The hallmark of the Internet and websites and apps is their ability to jump from one part in a document to another, as well as externally to other websites and apps, via hyperlinks or **anchors (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/a)** (the <a> element). Default anchors have pseudo-states that represent mouse/touch actions. The `a:link` provides the default blue and underline (called text-decoration) and represents a normal hyperlink. The `a:hover` provides a cursor change from default arrow to pointer (hand) when a mouse hovers over the hyperlinked words. The `a:active` state changes to red to alert the user that their action will change their location. The `a:visited` state changes to purple, so it is obvious to the user that they have already visited that link. Even though these typical state rules can be changed, it is important that they stay **obvious**, so users can quickly find them without struggling to see them.

In an external stylesheet, anchor rules are defined below the `body {}` selector, so it will inherit the its property values. Anchors are typically defined for paragraph and list text:

```
a:link {}
a:hover {}
```

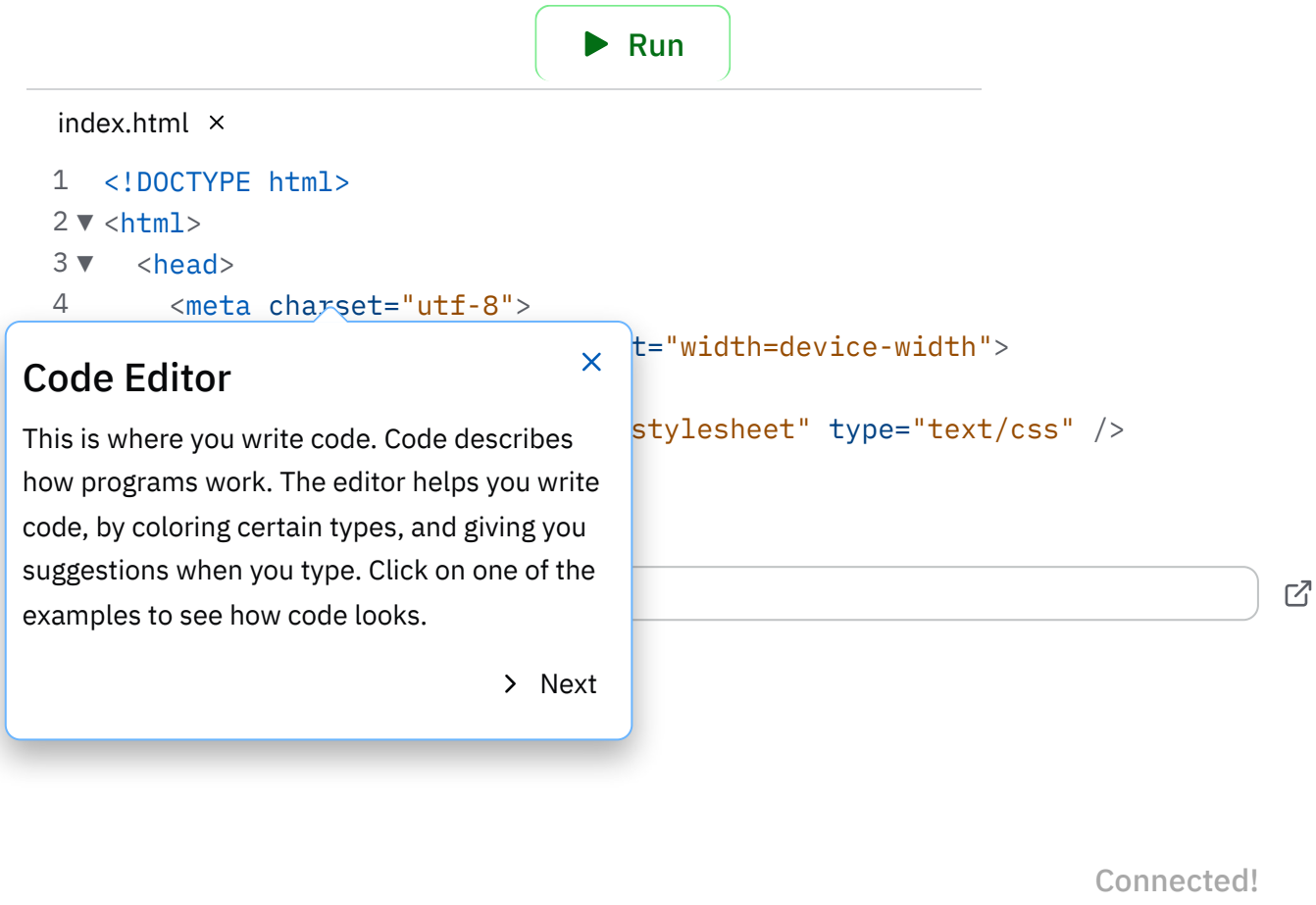
```
a:active {}
a:visited {}
```

Then, when developers want to define rules for page-to-page **navigation (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/nav)**, they place new anchor rules underneath the previous anchor rules, so they can inherit but override those styles. Navigational blocks of anchors typically look like buttons or tabs and include, at the very least, a `:hover` change, so users understand they could activate a change in location. In addition, the a:visited colors must be updated so they are visible on their parent block's background color. In other words, the dark purple of the visited state is not legible on top of a black background, so should get updated for high-contrast. For example:

```
nav {
  text-align:center;
}
nav a {
  background-color:black;
  border:2px solid black;
  border-radius:5px 5px 0 0;
  color:white;
  font-size:1.1em;
  font-weight:bold;
  margin-right:3px;
  padding: 3px 20px 5px 20px;
  text-decoration:none;
}
nav a:hover {
  background-color:white;
  color:black;
}
```

(Note that the `nav a` is the selector for all of the pseudo states, such as `:link`, `:active`, and `:visited`)

View the HTML, CSS, and output below, or **view the output in the browser. (https://navigation.pamvanlonden.repl.co)**



Access Denied

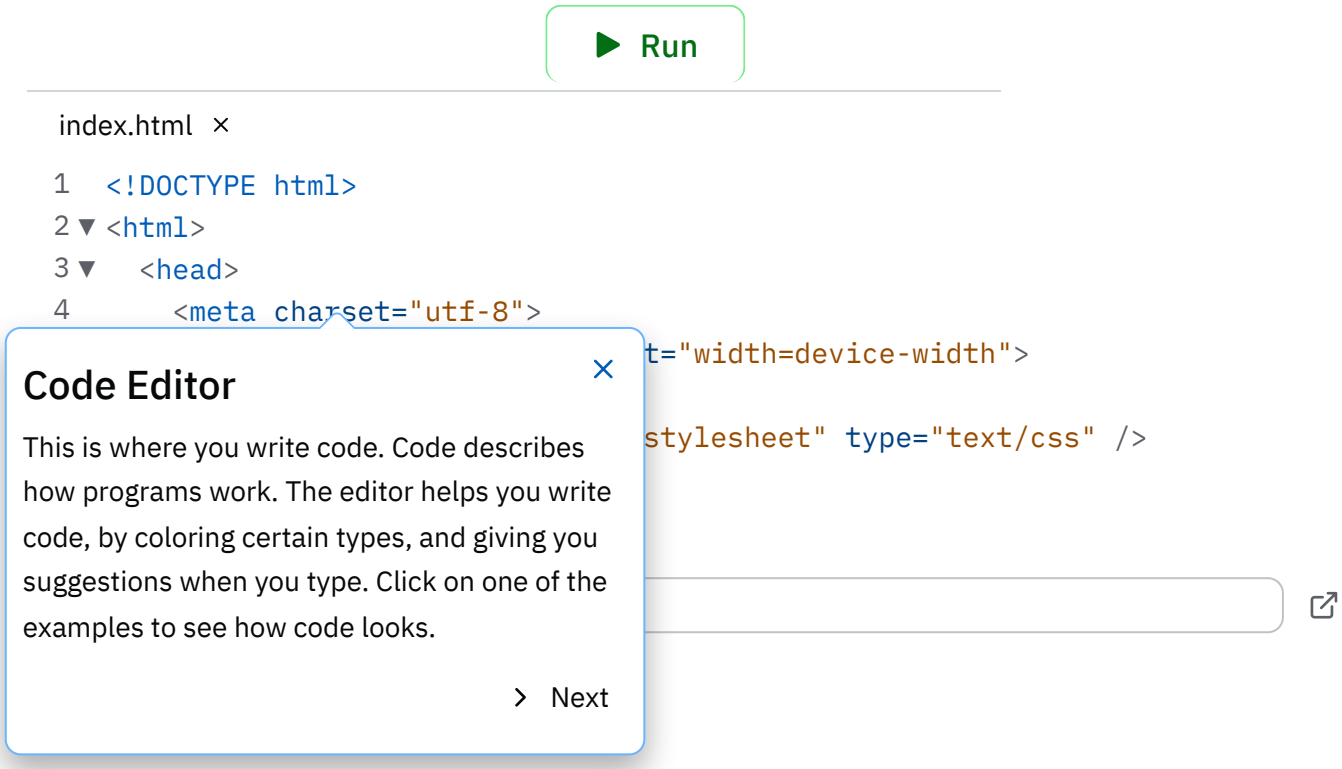
This video covers some but not all concepts related to page layout elements and selectors.

The placement of navigation will be determined by the **page layout (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/main)** ...above or below the `header` and/or `footer`, or beside the `main` block of content. Defining those areas of the page typically go next in the stylesheet file structure. HTML 5 introduced new names for areas of a page, so that developers were not stuck using a `<div>` for everything. A division block is typically used for embedding dynamic content. Page layout blocks help machines (such as search engine robots and screen readers) to understand which area of the page includes which components. For example, a typical page includes a `header, main, section` of `articles, asides`, and a `footer`. The stylesheet can define the perimeter of those blocks, as well as background color, text color, borders, shadows, etc.

When multiple `article` tags are displayed on a page in a section, they are typically marked with **ID** selectors (`#topic`), so they can be styled uniquely from another. Those IDs also help create a place for anchors to land when jumping internally within a page. Here is a simple set of page layout styles:

```
header, footer {
  background-color:orange;
  margin:0;
  padding:1.5%;
}
main {margin:0;}
section {
  background-color:whitesmoke;
  padding:3%;
}
article {border-top: 2px solid orange;}
#topic1 h3 {color:brown;}
```

Compare the page layout HTML with its correlating CSS and **view the output (https://page-layout.pamvanlonden.repl.co)** .



Typography (https://css-tricks.com/typography-for-developers/) is the next section of selectors to define in the stylesheet. Typography includes paragraph-level, phrase-level, headings, code- and instructional-writing selectors. There are 50+ selectors in this group, so we won't list them all here. At the very least, the paragraph, lists, and headings should be styled. The paragraph is typically styled like the font-related properties in the `body {}`. **Definition lists** dl {}, dt {}, dd {}, **ordered** ol {} and **unordered** ul {} **list items** li {} typically get bottom margin added to help separate them from the next item in the list. Definition list descriptions `dl dt {}` are often used for article titles with anchors and are typically styled with bold or a color change, like this:

```
li, dd {margin-bottom:7px;}
dt {
  font-weight:bold;
  color:orange;
  }
```

Access Denied

This video covers some but not all concepts related to Table elements and selectors.

In this course, you'll be rendering **Tables (https://css-tricks.com/complete-guide-table-element/)** in most of the assignments, and the default styles are...how should we say...ugly! And, difficult to read. Defining an understated but easy-to-read table style for your assignments will make the assignment more fun. At the very least, the table's row and column spacing and borders need definition, but all of its parts can be styled as needed. In the next exploration's Special Selectors section, we'll go into more detail. For now, start with the basics.

The table automatically has borders with space between each column/row, which creates too many lines, so **collapse** those borders to improve readability, like this:

```
table {border-collapse:collapse}
```

Then, since the **caption** does not inherit table styles (even though it resides within a table element) it needs spacing and perhaps borders, like this:

```
caption {
  background-color:orange;
  border:1px solid gray;
  color:white;
  font-weight:bold;
  padding:1% 0;
}
```

The **three sections** of a table can get specific styles as well, but for basic tables, that isn't necessary. One simple update is to center all the text in the table body, like this:

```
thead {}
tbody {text-align:center;}
tfoot {}
```

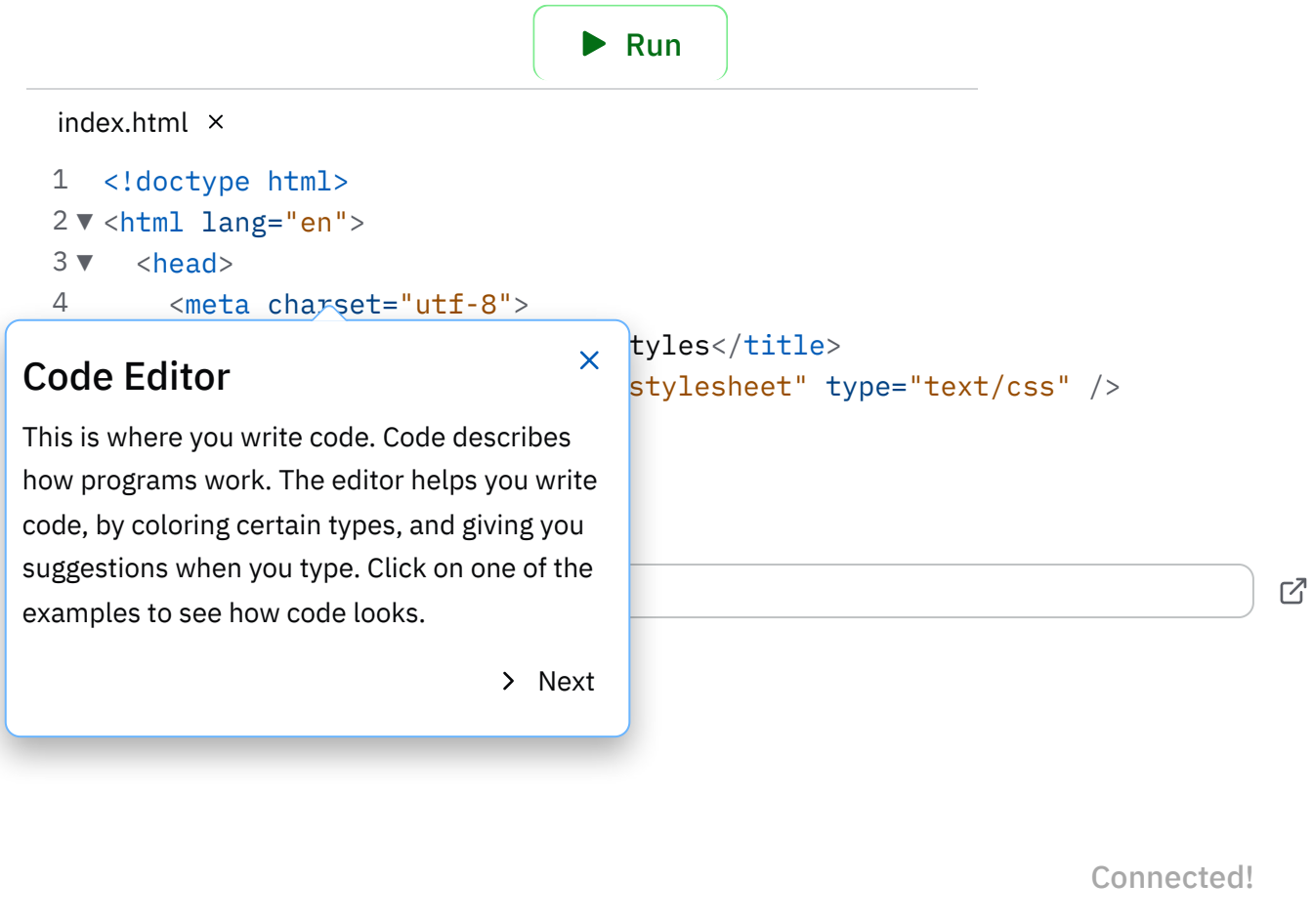
It is the row `tr {}` along with its counterparts, the `th {}` and/or `td {}` that completes the rendering of a cell in a table. It is common that the row selector will be included when specifying changes to a column, like this:

```
tr td {border-bottom:1px solid orange;}
```

The **heading** rows/columns are bold and centered by default, but you may want something more interesting, such as a subtle background color and padding. If your data is numerical, then align the columns to the right:

```
tr th {
  background-color:lightgray;
  padding:1%;
  text-align:right;
}
```

Compare the Table HTML to its correlating CSS to understand how they relate. **View the output (https://table-basic.pamvanlonden.repl.co/)** .



In the next module, you'll be working with the React framework, which includes an external, global stylesheet.

Customizing with classes and IDs

Previously, we discussed using **ID attributes** to give page layout blocks each a unique name, so we could differentiate one article from another and link to them. Those IDs can each be used *one time* per web page. Also, each `id=""` can have a single value/name; use of multiple names in the attribute is not allowed. Repeating the use of an ID on a single page is also not allowed by HTML. ID style rules also override class rules. See the next paragraph.

If we want to define a style that gets repeated on a page, then a **class attribute** is defined, rather than an ID name. For example, if you want to apply the same thick orange dotted border to each use of article, figure, and table, you could define a class (using a dot), like this:

```
.orangeBorder {
  border:5px dotted orange;
}
```

and apply it like this in HTML:

```
<article class="orangeBorder">
<figure class="orangeBorder">
```

Unlike with IDs, class attributes can list multiple class names to build upon smaller design components, like this:

```
<article class="orangeBorder boxShadow purpleText">
```

Some frameworks have their own classes (like Module 5's React framework), which can be included in HTML tags using `className=""` attributes (not to be confused with standard `class=""` attributes).

The Cascade

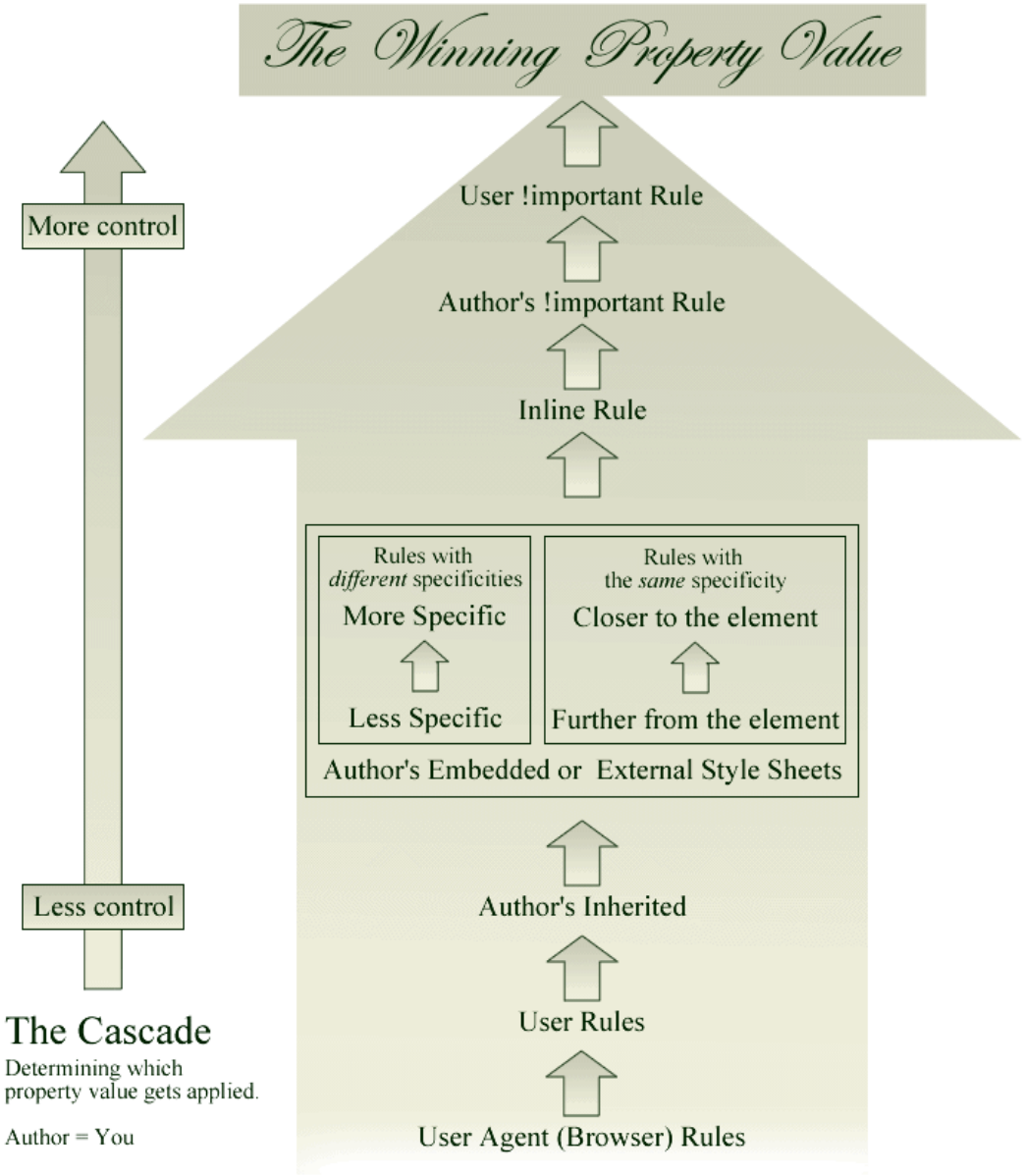
Style rules are applied using a "cascading" method. In its simplest terms, the cascade works like this:

Cascade

Linked external
stylesheet is overruled by
↓
Imported external
stylesheet, which is overruled by
↓
Embedded/internal

document level stylesheet, which
is overruled by
↓
Inline
style definition.

Dan Hitchcock of UCLA Extension explains the Cascade more specifically. Control belongs to the visitor via their browser. The next level of control belongs to the web developer when using `!important` with a definition inline, then any inline style, then any embedded style in the section of a document, then styles at the bottom of a `.css` file. Using `!important` in the `.css` file overrides the same declaration made earlier in the document.



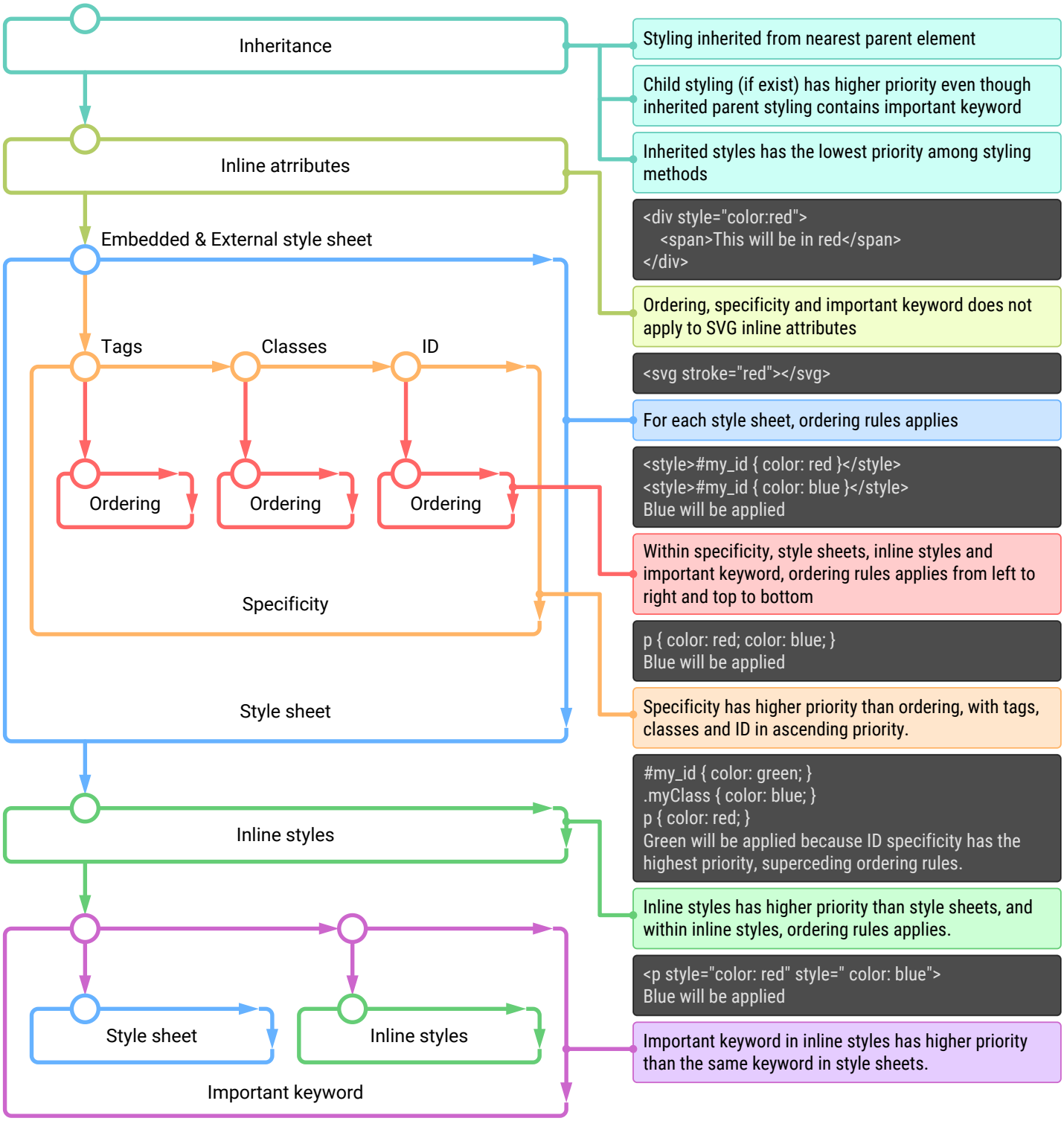
(http://www.gottheknack.com/a_htmlCss/courseDocs/cascade/cascade.html)

A more specific and current rendition of the Cascade is presented by **Thomas Yip (https://vecta.io/blog/definitive-guide-to-css-styling-order/#how-everything-works-in-a-diagram)** (2018).

The definitive guide to CSS styling order

Includes CSS stylings for SVG

vecta.io
Diagramming for teams



Reducing stylesheet load times

Stylesheets can get to be 1000s of lines of code long, which takes several seconds to load, so most front-end developers take steps to reduce that load time using minimization, Critical, and Lazyload

techniques. The stylesheets we develop for this course will not require minimization, but if you want to learn more, read: **Optimizing CSS for faster page loads (https://pustelto.com/blog/optimizing-css-for-faster-page-loads/)** .

Exercise: Define Basic Rules

Apply semantic HTML tags to the text in the following Repl, and update the color, font, font-size, and line-height of the `<body>` text and headings. Decorate the navigation `<nav>` with color:

1. In the HTML file, add page layout tags for a heading, sections, articles, second level headings, paragraphs, and a footer, to differentiate each part of the page and block of text.
2. Also add a navigation block with anchors and IDs to each section of shirts.
3. In the CSS file, import the normalize.css and import the Ubuntu font from Google Fonts.
4. Change the body to Ubuntu font, DarkSlateGray, size 1em, and a line height of 1.2.
5. Change the navigational anchor background-color to Coral with bold, White text. Reverse the colors when hovering.
6. Add DarkSlateGray to the background of the header and footer and change the text to White.
7. Change the section background color to PapayaWhip.
8. Add a 1px solid DarkSlateGray border to the article.
9. Change the heading 1, 2, and 3 sizes to 200%, 150%, and 125% and the color Coral.

Edit this Replit to add HTML tags and CSS selectors, properties, and values:

index.html ×

```
1  <!DOCTYPE html>
2 ▼<html>
3 ▼  <head>
4      <meta charset="utf-8">
       <meta name="viewport"
5  content="width=device-width">
       <title>Beaver Shirts for Sale</title>
       <link href="style.css" rel="stylesheet"
   type="text/css" />
6    </head>
```

Code Editor

This is where you write code. Code describes how programs work. The editor helps you write code, by coloring certain types, and giving you suggestions when you type. Click on one of the examples to see how code looks.

› Next

rge pocket and

We'll improve the spacing of this design in the next exploration.

Summary

In this exploration, we looked at the relationship between HTML and its defaults styles. In an external stylesheet, we added selectors with various properties and values to override those defaut HTML styles. This foundation will help you incorporate styles in the upcoming assignments to improve their visual interest and usability.

Additional Resources

Learn more about the topics we discussed in this exploration:

- MDN provides detailed **tutorials on CSS** (https://developer.mozilla.org/en-US/docs/Web/CSS).
- MDN's **reference on CSS** (https://developer.mozilla.org/en-US/docs/Web/CSS/Reference).
- The CSS Tricks websites has a useful **CSS almanac** (https://css-tricks.com/almanac/).
- A handy reference to **CSS selectors** (https://developer.mozilla.org/en-US/docs/Web/CSS/Reference#selectors).
- In addition to pseudo-class, selectors can be defined based on pseudo-elements. To learn more about this, see **the discussion on MDN** (https://developer.mozilla.org/en-

US/docs/Learn/CSS/Building_blocks/Selectors/Pseudo-classes_and_pseudo-elements) .

- W3C's online **Unicorn Validator** (https://validator.w3.org/unicorn/) can find typos that will mis-render the styles.