Exploration — CSS Methods

Introduction

Now that you've drafted a basic external stylesheet, you can begin to experiment with the block/box model, page layout grids, color options, special selectors, and pseudo-classes. These concepts make use of the 298+ selectors and properties available in the HTML/CSS language. For comprehensive use of each, refer to **CSS Tricks Almanac** **(https://css-tricks.com/almanac/)** .

In this Exploration, we'll learn about the 7 parts of the block model, how to specify dimensions for the inside and outside of blocks, and how to position them:

- outline
- margin
- border
- padding
- background
- content (such as text and images)
- line-height

This new knowledge will help you revise the page layout started in the previous Exploration. Then, we'll look at color systems and how to define color variables for reuse in selector rules. We'll also improve the usability of form elements and improve the look of tables. And, finally, we'll explore how to debug CSS problems.

The Block Model

Nearly all HTML elements display as an empty block. The visual representation of **spacing** for these empty blocks is created by manipulating their outline, margin, border, padding, content, background, and line height properties. **Hover** over the diagram below, **starting at the top left corner**, to see where in an element these properties reside. Work your way down to see each element revealed:

Margin **and outline**

Border

Padding **and background**

Content **Line-height**

This video covers some but not all concepts related to the block model.

When hovering over the diagram, you'll see the `outline` (https://css-tricks.com/almanac/properties/o/outline/) property appear as a blue dotted line. Outline is typically seen when an element such as an anchor or form input has `:focus` via the mouse. Its properties include a width, style, and color, written in shorthand, like this:

```
outline: 1px dotted blue;
```

Inside of the outline is the `margin` (https://css-tricks.com/almanac/properties/m/margin/), which allows developers to separate one element or block from another. For example, if you had two articles side by side, you would separate them with margin left and right, so they don't touch each other, like this:

```
article {margin:0 2%;}
```

Margin is made up of the four sides of a block...top, right, bottom, and left. Their values are specified in that order (clockwise). Margin can be specified with one value for all four sides: `margin:2%;`, two values for top/bottom and left/right: `margin:0 2%;`, or all four values: `margin:0% 2% 0 2%;`. Their **typical units of measurement** include:

- percentage (%), relative to viewport spacing.
- ems (em), relative to font size spacing.
- pixels (px), for absolute spacing.

Margins can mix and match units of measurement on each side (whereas padding cannot). Many other relative and absolute units exist for the screen as well as for printing. **Read more (https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Values_and_units)**.

Margins can be specified in negative values, which allows a block to move up, down, left, or right of their normal spacing. For example, if you want a navigational block to overlap the header area, the `main` block can be moved up, like this:

```
main {margin-top:-30px;}
```

Centering an element inside of another element may require use of the `auto` value. For example, to center an article in the middle of a section, define a rule with 0 margin top to bottom and a percentage right to left, like this:

```
article {
  margin:0 auto;
}
```

The next element down is the `border` (https://css-tricks.com/almanac/properties/b/border/), which is the space that separates margin from padding. Borders have a solid style, by default, but have 0 width and color until otherwise specified. All three values are needed to render a border, and most developers specify all three in a shortcut, like this:

```
border: 3px dotted red
```

Sometimes, it is necessary to specify the border for only one side of a block, which then requires -top, -right, -bottom, or -left suffixes. This allows different design for each side of a block; different widths, types, and colors, like this:

```
article {
  border-top: 4px dotted purple;
  border-right: 6px dashed red;
  border-bottom: 8px solid orange;
  border-left: 10px groove yellow;
}
```

Padding (https://css-tricks.com/almanac/properties/p/padding/) is the space between the border and the content. This padded area can include a background color and multiple background images, which display behind content. Padding can specify one, two, or four values in clockwise order just like Margin, but it doesn't allow mixing of different units of measurement. Headings, paragraphs, and other elements have padding by default. The Normalize stylesheet sets padding for some elements to 0, typically, so the developer can specify a more pleasing amount of spacing. For example, a heading 2, followed by a paragraph, looks better when there is no margin or padding between them, like this:

```
h2 {margin:0; padding:0;}
p {margin-top: 0; }
```

An annoying feature of padding is that its values are added to the width and/or height of a block. For example, if an article's width is set to 400px, and 10px of padding have been added to the right and left, the total width will be 420px. To eliminate this, the `box-sizing` property is added to elements that need it, like this:

```
article {
  padding: 20px;
  width: 400px;
  box-sizing: border-box;
}
```

Some elements already include box-sizing via the Normalize.css file.

Content, such as text, inherits the `body {}` element's margin, padding, and line-height. The width and height of the parent block will adjust based on the amount of content, unless otherwise specified. Content sits on top of background colors and images.

And, finally, the Line Height (https://css-tricks.com/almanac/properties/l/line-height/) property specifies the height of one line of content such as text or an image. Line-height is necessary to improve readability of long passages of text and typically supports vertical alignment in the middle of a parent block, like this:

```
figure img {
  line-height: 1;
  vertical-align: middle;
}
```

Grids and Positioning (optional)

In this course, you won't need to use grid layouts or position specific elements. However, it is important that you understand the many options for placing elements on a page and when to use them: float, columns, position, flex, and grid. (Click on each name to view examples.)

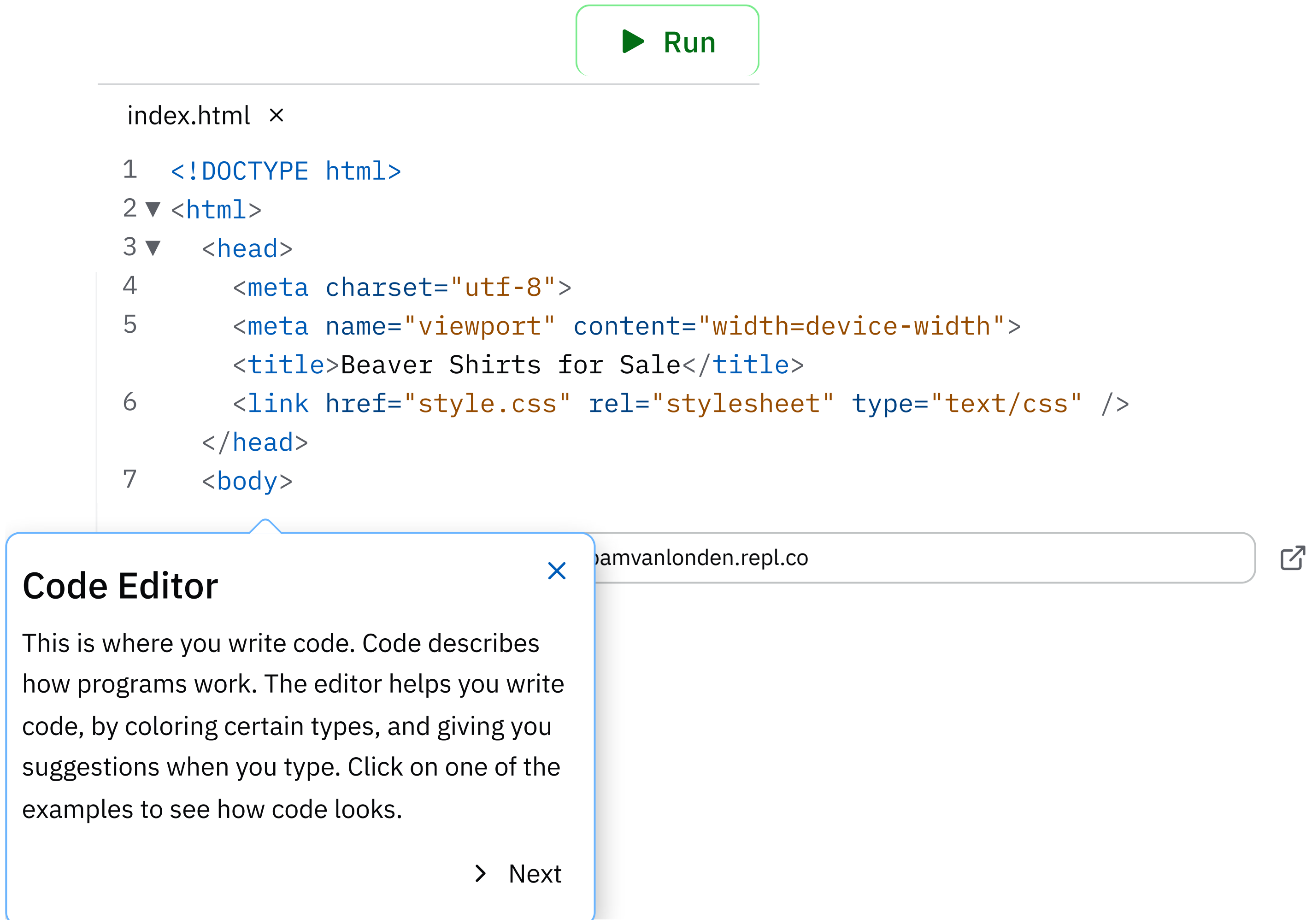
- **Float** (https://css-tricks.com/all-about-floats/), allows one block to float right or left of another block. Requires specifying width and clear. Float is typically applied to aside and figure blocks within an article. They are sometimes used to define two-or three-column page layouts.
- **Columns** (https://css-tricks.com/almanac/properties/c/column-count/), allows easy masonry-style gallery layout. May require column-count, column-gap, column-fill, etc.
- **Position** (https://css-tricks.com/almanac/properties/p/position/), using absolute, relative, fixed, or sticky placement. Fixed and sticky are helpful for keeping menus at the top, bottom, or side when scrolling.
- **Flex** (https://css-tricks.com/snippets/css/a-guide-to-flexbox/), defines how child blocks relate spatially to each other within a flexible/responsive parent block. The space is distributed evenly to fill available space or shrinks to prevent content from spilling out of the parent block. It has a one-dimensional flow, and wrapping is at the will of the browser's viewport size.
- **Grid** (https://css-tricks.com/snippets/css/complete-guide-grid/), defines how a parent block is laid out in two-dimensions, then considers how child blocks are oriented within the parent grid's cells. This method often eliminates the need for media queries for complex layouts. Cells can be animated.

Exercise: Margin, Border, Padding

Improve spacing of the previous exercise (previous Exploration) by adding margin, border, and padding to the header, nav, section, article, and footer.

1. In the CSS file, add 2% margin and 2% padding to each page layout block element.
2. Remove the default margin and padding from headings 1, 2, and 3.
3. To the navigational anchors, add a 5px border-radius, 1% padding, and separate them with 2% left margin.

Edit this Replit or your existing Replit from the previous Exploration. Click the file icon to access the CSS file:



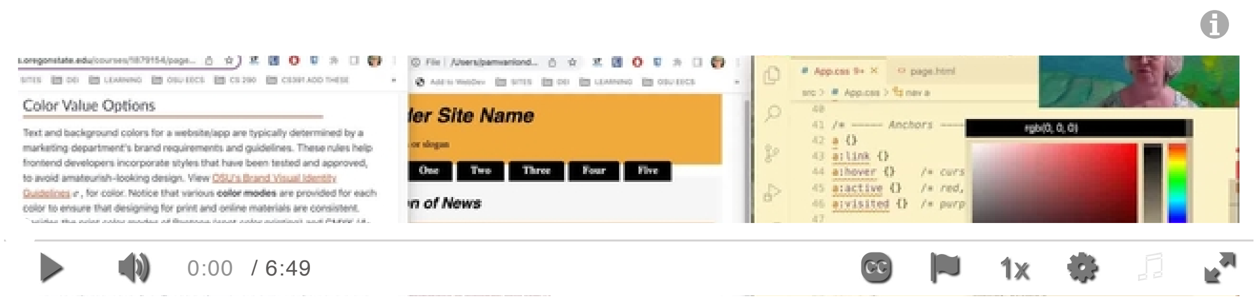
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>Beaver Shirts for Sale</title>
    <link href="style.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
```

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

Color Value Options



This video covers some but not all concepts related color value options.

Text and background colors for a website/app are typically determined by a marketing department's brand requirements and guidelines. These rules help frontend developers incorporate styles that have been tested and approved, to avoid amateurish-looking design. View **OSU's Brand Visual Identity Guidelines (https://communications.oregonstate.edu/brand-guide/visual-identity/colors)**, for color. Notice that various **color modes** are provided for each color to ensure that designing for print and online materials are consistent. Besides the print color modes of Pantone (spot color printing) and CMYK (4-color printing), online color values can be included for the following modes (click on each name to view examples):

- **Names (https://css-tricks.com/snippets/css/named-colors-and-hex-equivalents/)**, 256 pre-defined HTML colors which have names. Many of the names represent overly-bright colors, which are often not part of a brand and may make a website/app look amateurish. Here is the color named Orange: `orange`
- **Hexidecimal and Hexa (https://www.digitalocean.com/community/tutorials/css-hex-code-colors-alpha-values)**, made up of 2-digit values for red, green, and blue. Black is #000000 or #000 and white is #ffffff or #fff. Adjusting the RGB values will render up to 256^3 colors! In addition, an alpha channel allows transparency with an addition of two more digits #RRGGBBAA. Here is the OSU Beaver Orange color and then again with 80% transparency applied: `#D73F09` and `#D73F09cc`
- **RGB and RGBa (https://htmlcolors.com/rgba-color)**, made up of 1, 2, or 3-digit values for red, green, and blue. Black is rgb(0,0,0) and white is rgb(255,255,255). Adjusting the RGB values will render up to millions of colors! In addition, an alpha channel allows transparency with an addition of another digit. Here is the OSU Beaver Orange color and then again with 80% transparency applied: `rgb(215,63,9)` and `rgba(215,63,9,0.8)`

- **HSL and HSLa (https://htmlcolors.com/hsla-color)**, is made up of values for hue, saturation, luminosity/lightness, and alpha transparency. Designers from the photography industry will recognize this mode. Once a single hue is chosen, then the brightness/saturation and tone (addition of gray) can be defined for variations based on that same hue. Here is the OSU Beaver Orange color and then again with 80% transparency applied: `hsl(16, 92%, 44%)` and `hsla(16, 92%, 44%, 80%)`.

An excellent method for defining variables for HSL using calculations is presented by Dieter Raber at CSS Tricks: **Creating Color Themes With Custom Properties, HSL, and a Little calc() (https://css-tricks.com/creating-color-themes-with-custom-properties-hsl-and-a-little-calc/)**

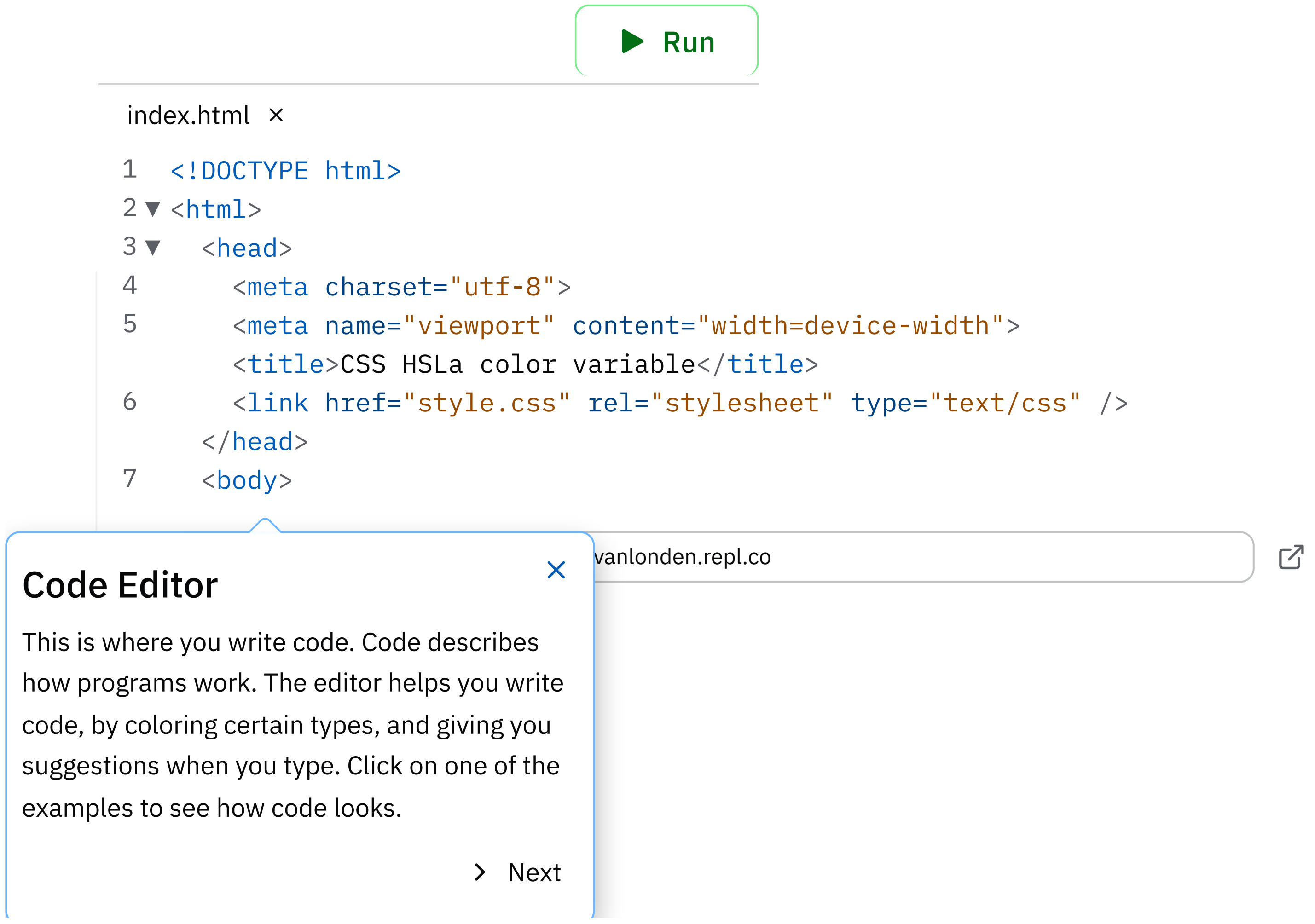
When a brand does not specify a color mode that you prefer to use, a conversion tool will provide the value. Ask Google to convert orange into Hex, RGB, or HSL to see how easy it is!

Exercise: Color Variables (optional)

Define a variable for OSU's Stratesphere blue color and use it in a button control that has improved spacing and interaction:

1. Using a color-conversion tool, convert OSU's Stratosphere blue color from RGB (0-106-142) to HLSa.
2. In the CSS file, define that new color in the :root {} rule.
3. Use that color variable for the background of the button element and add a 2px white border with 5px border radius, white text, and 2.5% padding inside.
4. Switch the colors when hovering.

Edit this Replit to change the button styles.



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>CSS HSLa color variable</title>
    <link href="style.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
```

vanlonden.repl.co

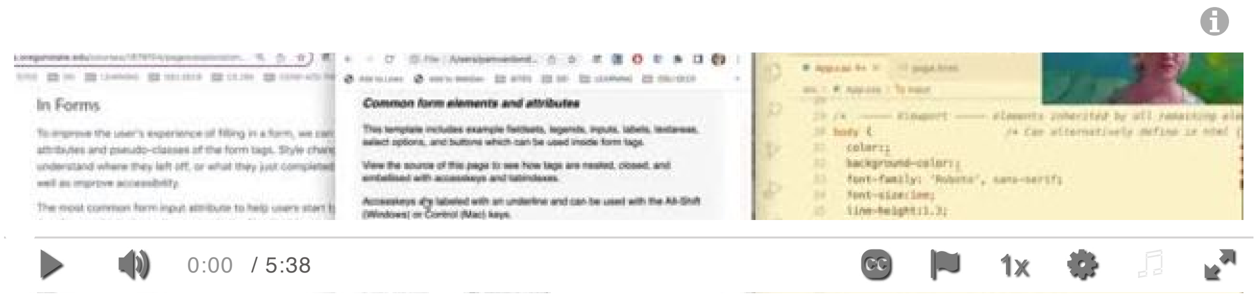
Code Editor

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.

Special selectors and pseudo-classes

The style of many attributes we include in HTML tags can be controlled via a stylesheet. The following examples are helpful to learn when working with tables and forms.

In Forms



This video covers some but not all concepts related basic form styles.

Most frontend developers will revise the default form elements so they are easier to read and interact with. Many forms are filled in via cellphone, so the height and width of inputs, selects, and buttons ought to be about the size of the tip of a finger and each group should be visually separated to reduce confusion. Consider adding these kinds of properties to improve accessibility of your controls:

Change the default gray border to some other color, type and width on the **fieldset**. Add margin to provide breathing room above each fieldset. Add padding inside the fieldset to keep the labels and controls from touching the edges:

```
fieldset {
   border:;
   margin-top:;
   padding:;
}
```

Each **fieldset** has a legend that helps discribe its group of form controls, or prompts the user. Its default styles incorporate positioning properties, which are hard to control. Consider changing its font size and color to differentiate it from labels.

```
legend {
   color:;
   font-size:;
}
```

To display labels above, rather than inline with, their form controls, use the block value, and perhaps a smaller font size:

```
label {
    display:block;
    font-size:80%;
}
```

Accommodate fat fingers and disabled hands by increasing the font size and adding padding:

```
input {
    font-size:120%;
    padding:2%;
}
```

Optional, more advanced form elements, attributes, and selectors will improve the user's experience of filling in a form. Style changes that help users understand where they left off, or what they just completed, can add interest as well as improve accessibility.

The most common form input attribute to help users start typing in a form is **autofocus**, which places the cursor in the first field for immediate typing, rather than necessitating a mouse click or touch into the input. (This property may not work inside of Canvas.) A color change will help users realize they can begin typing. And, any field that has the mouse's focus can have a color change, like this:

```
:focus{
    background-color:HoneyDew;
}
```

To help users understand whether their input meets the required `pattern`, such as enough of the right kinds of characters in an `<input type="password" />`...

```
<input type="password"
  id="pwd" name="pwd"
  size="12"
  minlength="8" maxlength="12"
  pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,12}"
  title="Include 8 to 12 upper, lower, numbers, and characters"
  placeholder="Include 8 to 12 upper, lower, numbers, and characters"
  autocomplete="current-password"
/>
```

... changing the border color from gray to red when invalid and green when valid is common, like this:

```
input:invalid {
  border:3px solid red;
}
input:valid {
  border:2px solid green;
}
```

When inputs have the mouse's focus and a color change is specified, the placeholder text may need updating to improve contrast (and ultimately better readability), so a text color change may be needed:

```
input::placeholder {
    color:darkgray.
}
```

Learn more about **Placeholder selectors.** **(https://css-tricks.com/almanac/selectors/p/placeholder/)**

Consider updating how a checked box or radio button looks when unchecked and checked. This method uses the `type="checkbox"` attribute as well as the adjacent sibling `label` and works well if the `label` tag is placed after the `input` tag in the HTML:

```
input[type=checkbox]:checked + label {
  color: green;
  font-style: italic;
}
```

Learn more about the **:checked selector.** **(https://css-tricks.com/almanac/selectors/c/checked/)**

Complex forms, where one input relies on another, often use the `:disabled` and `:enabled` attributes to keep parts of a form from being accessed until the user provides the necessary input. Styling these disabled controls often follows this pattern:

```
input:disabled {
  background: repeating-linear-gradient(142deg, #CCCCCC, #CCCCCC 4px, #999999 22px);
}
input:enabled {
  background: white;
}
```

View more form controls and their unique attributes and selectors:

▶ Run

index.html ×

```
<!doctype html >
<html lang="en">
<head>
<meta charset="UTF-8" />
<title>Common form field elements and attributes</title>
<link href="style.css" rel="stylesheet" type="text/css" />
</head>

```

pl.co

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

In Tables

When one or more columns in a table represent numerical data, that data is easier to read when aligned to the right, especially if it represents currency. This is accomplished by singling out those 'child' columns with `:nth-child(#)`, like this, when the third column requires right-alignment:

```
th:nth-child(3),
td:nth-child(3) {text-align:right;}
```

A large table of data will be easier to read if every other row is a slightly different color. This is accomplished by specifying 'odd' or 'even' rows, like this:

```
tr:nth-child(odd) {
    background-color:beige;
}
```

Play more with table styles. Click the file icon to access the CSS file:

Run

index.html ×

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
                         tyles</title>
                         stylesheet" type="text/css" />
```

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

Exercise: Debugging CSS

Chrome Inspect shows the tags and their related styles.

A single typo can render CSS inactive. Finding that typo is tedious without the help of tools. Front-end developers typically use these tools to speed up the process: their IDE's debugging features (VS Code will show typos with red squiggly underlines and in the Problems tab of the terminal), the browser's Dev Tools, and the Unicorn validator.

In FireFox or Chrome, try this to see what the browser sees:

1. Launch this **background photo demonstration (http://pamvanlonden.com/demos/techniques/background/background-color.php)** page.
2. Click on the main content area in the viewport and right-click to choose the Inspect menu (Chrome) or Dev Tools (FireFox).
3. With the `<main>` tag selected on the left window, the correlating CSS renders on the right window. You can see that the background has a 90% alpha opacity setting on the RGB color white, with a

border radius of 5px, and padding of 5%.

4. Now click on the `<body>` tag in the left window to see what styles are specified. The `<body>` has a green background color that covers the entire viewport, with top and bottom margins of 5%, and left and right margins of 10%.
5. Click on the `<nav>` tag in the left window to view its style on the right window. Notice that it has no style rules defined; it just lists the browser's default settings (which are specified in italics).

Another important tool is the **Unicorn Validator (https://validator.w3.org/unicorn/)** by W3.org. It will show warnings and errors for HTML and CSS on the same screen. If the HTML syntax is properly nested (green status) then you can review the CSS error list to determine exact line numbers for typos.

Summary

We learned to manipulate the outline, margin, border, padding, and line-height of a selector's block, the five different positioning options, how to define variables for colors from two of the four modes, and how to improve the usability and look of forms and tables. In addition, we explored how to use three debugging tools.

Additional Resources

Learn more about design topics:

- **Index of CSS Properties (https://developer.mozilla.org/en-US/docs/Web/CSS/Reference#index)** at MDN and list of most **commonly used CSS properties (https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Properties_Reference)**.
- Discussion of the box model at **MDN (https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/The_box_model)** and at **W3 School (https://www.w3schools.com/css/css_boxmodel.asp)**.
- Discussion of positioning at **MDN (https://developer.mozilla.org/en-US/docs/Web/CSS/position)** and at **W3 School (https://www.w3schools.com/css/css_positioning.asp)**.