# Week 7: CSS Part II

**Agenda**

- Overview
    - Previous Participation Questions/Responses
    - What's Due?
- CSS Continued (Lecture/Class Activity)
    - CSS Basics Review
    - More CSS Selectors
    - Box Model
    - Positioning with CSS
- CSS Positioning Exercise (CW/HW)
- Participation Questions (4)
- Next Week

## Overview

Last week, we started learning about CSS and how it can be added to our pages to give them some styles. This week, we will review some of that information and learn more about applying CSS to our pages. By the end of this week, students will be able to apply CSS to position various elements on a webpage.

**Previous Participation Questions/Responses**

These are the participation questions from last week:
https://forms.gle/rMaLpfDhz1gT7UgJ7

This link has the responses to the fun question from last week. Feel free to look over the responses from myself and your peers.
https://docs.google.com/spreadsheets/d/1QLoQzbO-Er7ARIECl0aPvY1lKUyZTdYeuwsn4g0XEAk/edit?usp=sharing

**What's Due?**

- Participation Questions (4)
- First CSS Exercise (CW/HW)
- Project Proposal (tonight at midnight)

## CSS Continued (Lecture/Class Activity)

**CSS Basics Review**

In CSS, there are many methods for selecting HTML elements for styling. The ones we saw in last week's lecture notes were based on selecting HTML elements directly or classes. In the example below, we are **selecting** all p elements and changing their color **property** to a **value** of red.

The text on the outside of the brackets is the selector, color is the property, and red is the value. The property is separated from the value by a colon and the whole statement is ended with a semicolon.

```
<div class="container content">
	<p>This will be red.</p>
</div>
```

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

Another way to target that paragraph inside of the div would be to use the class attribute value of the div. In our example, our div tag has two classes; one is "container" and the other is "content." In HTML you can separate class names with a space.

We can target that class by putting a period before the name of the class. However, we don't want to target the div but the paragraph inside of it. We can add a space after the class name followed by the name of the element as shown below. This can be described as descendent selection because we are selecting the descendent of another element.

```
.container p {styles in here}
.content p {styles in here}
```

Both methods above are valid since each one uses a class from the div. The reason you would do something like this is because you might want to apply specific styles to one or only a few elements rather than all of them. You don't need to put classes on every HTML element but the major containers and minor containers in your pages are good places to start.

We can also target elements with their id attribute value. Let's say we have a paragraph with an id of paragraph: <p id="paragraph">Content in here.</p>

We could target that specific paragraph with a hashtag/pound symbol in front of the name of the id.

```
#paragraph {styles in here}
```

The three main ways to target elements are through the element names, the classes of those elements, and the ids of those elements. However, I strongly advise against using ids to target elements. Ids are meant to be used as a hook for JavaScript functionality. Later this semester, we will make use of ids but for now, try to target items on your pages using only element and class names.

**More CSS Selectors**
We can target groups of elements at the same time by using a comma in the selector list.

Example: p, span {styles here would apply to all p and span tags}

We can target elements that are direct children of other elements with the greater than symbol “>”.

Example: div > p {styles in here would apply to only p tags directly inside of div tags}

The child combinator (>) is like selecting a descendent element. However, descendant selection applies to all elements inside of another element, regardless of where they are in the nesting structure. The direct child selection applies only if they are directly inside of it.

We can select elements that are siblings of one another using the tilde “~”. The second element follows the first and shares the same parent container.

Example: h2 ~ p {styles in here affects p tags that follow h2 tags in the same parent container}

We can select elements that are adjacent siblings using the plus sign “+”. It’s like the sibling selector above except the second element must follow the first one **immediately.**

Example: h2 + p {styles in here only affect p tags that are directly after an h2 tag}

We can select elements based on information that is not available in the document structure. To target those elements or classes, we follow the html tag, class, etc. with a colon and then the name of the pseudo selector.

Examples:
p:hover {styles in here apply to all p tags when they are hovered over}
input:focus {styles in here apply to any input tag when it gains focus}

There are other selectors and they’re useful to understand because you can select HTML elements in a wide variety of ways. Most of the time, you’ll only need basic selectors, but the other ones offer powerful solutions when you’re otherwise stuck.

Online Resources

- https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors
- https://www.w3schools.com/css/css_pseudo_classes.asp
- https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-elements

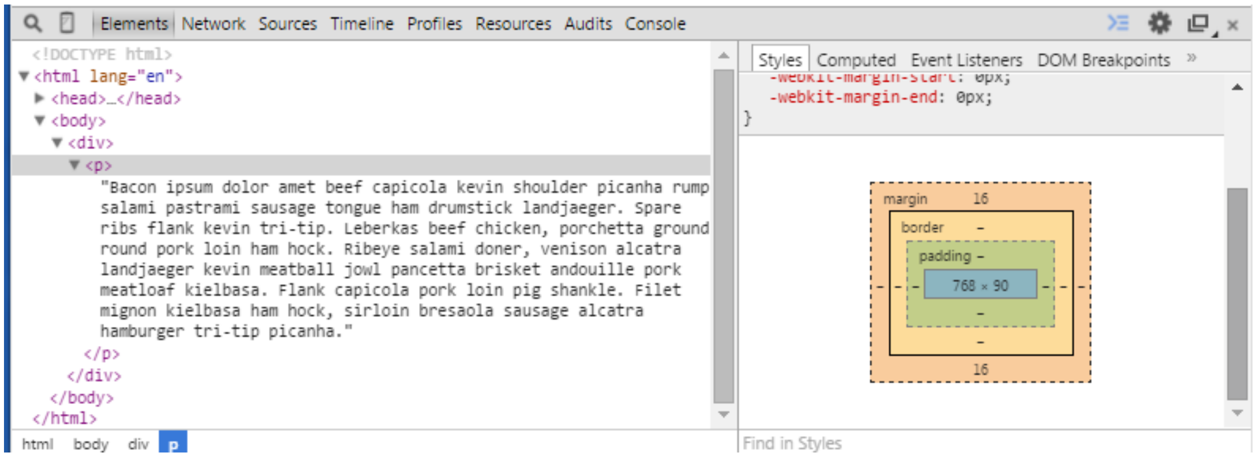
**Box Model**
Every element in HTML can be considered as a box. Each of these boxes takes up a certain amount of space on our page relative to the other boxes. Also, the spacing inside of these boxes can vary depending on the content inside of them.

The box model in CSS is a way to understand how HTML is laid out and uses certain properties to help achieve this goal. It is comprised of **three** main properties. The properties are *padding*, *border*, and *margin* which all impact how elements are spaced out on a page:

The padding property handles the space between the content and the border around that content.

The border property handles borders which wraps around the padding and content.

The margin property handles space outside of the border and wraps up the border, padding, and content.



```
<!DOCTYPE html>
▼<html lang="en">
  ▶<head>…</head>
  ▼<body>
    ▼<div>
      ▼<p>
          "Bacon ipsum dolor amet beef capicola kevin shoulder picanha rump
          salami pastrami sausage tongue ham drumstick landjaeger. Spare
          ribs flank kevin tri-tip. Leberkas beef chicken, porchetta ground
          round pork loin ham hock. Ribeye salami doner, venison alcatra
          landjaeger kevin meatball jowl pancetta brisket andouille pork
          meatloaf kielbasa. Flank capicola pork loin pig shankle. Filet
          mignon kielbasa ham hock, sirloin bresaola sausage alcatra
          hamburger tri-tip picanha."
        </p>
      </div>
    </body>
</html>
```

```
  -webkit-margin-end: 0px;
}
```

In the image above, I have used Google Chrome's "Inspect Element" feature (F12 or right-click and select it) to view the HTML/CSS in a page. On the left side of the inspector, the HTML structure is viewable. On the right side, there is a multi-colored box which depicts the CSS box model in action.

The orange area represents the margin (16px default for paragraphs in Chrome). The border is depicted by the yellow area. Lastly, the green area is the padding which separates the content from the border.

CSS Shorthand

If you wanted to manipulate the padding or margins of an element for different sides, you could target the sides with the margin or padding side property. I use px in these examples in these examples, but you could use any unit of measurement.

margin-left: 10px;
margin-right: 10px;
margin-bottom: 10px;
margin-top: 10px;

You could achieve the same effect in one declaration. The line below this would apply 10 pixels of margin to all sides of an element.

margin: 10px;

If you wanted to target the top/bottom and left/right values, you could use two numerical values separated by a space. The line below would add 0 margin to the top and bottom but 20px to the left and right sides.

margin: 0 20px;

You could target the top, left/right sides, and bottom separately as well. The line below does the same thing as the line above but the first and last value represent the top and bottom margins respectively.

```
margin: 0 20px 0;
```

Lastly, you can target each individual side with one declaration. The order is top, right, bottom, left and is always clockwise. The line below would give an element 10px of padding on top, 20px on the right, 30px on the bottom, and 40px on the left side.

```
Padding: 10px 20px 30px 40px;
```

There is shorthand for padding, margin, borders, and other CSS properties so experiment with them when adding your own styles to a page. The less CSS you write, the better your page will perform.

The links below cover the CSS Box Model in greater detail and explain things like inline vs block level elements as well. The CSS slides on Blackboard also provide a quick glimpse into these topics and should be reviewed.

## Positioning with CSS

There are numerous ways to position HTML elements using CSS. Position, float, flexbox, grid, and other properties allow us to place our elements, but each one has its strengths and weaknesses. I will explain the position and float properties and provide links to some of the other ones as well.

For this course, most of you will probably use the float property since position is harder to implement for a site with more content. Flexbox and grid are also great methods and were more recently implemented in the CSS specs. They can accomplish more things but can be complicated in some instances.

### Width Calculations

Before we get into positioning elements, it is important to understand how widths work in CSS. When you move things around on your page, the amount of space you use for the content and space between content (sometimes referred to as “gutters”) impact the readability and usability of your page.

In CSS, the full width of an element is equal to the width of the content plus the margins, padding, and border. When trying to figure out how much margin or width you have to provide for elements to space them out, you would have to consider the widths of the border or padding as well.

For example, if we have two divs with a width of 40% that are floated to the left, with 5% margin on the left and right sides. In total, the width would be 100% since the divs take up 80% of the width and the margins take up 20% of the width (5% for each left/right of each div). Once we add padding or a border to either div, the width goes over 100% and one of the divs will be forced to the next line.

The reason for this is because padding and border are separate parts of the width for calculation purposes. There is a property which is useful for reducing the amount of math you'll need to do for these calculations. It's called box-sizing and when used with a value of border-box, it automatically adds in the padding and border widths when calculating the width of an element. I've included it below, and I would recommend that you use it on your pages for all assignments and the final project.

```
* {box-sizing: border-box;}
```

It means that all elements on the page ("*" targets everything) will have the border/padding added into width calculations. If you had the same CSS as mentioned above (40% width on two divs with 5% margins on left/right), and used this declaration, adding padding or a border wouldn't force content to the next line. Instead, the padding and border amounts would get added in and the content itself would become thinner.

The links below cover the CSS Box Model in greater detail and explain things like inline vs block level elements as well. The CSS slides on Blackboard also provide a quick glimpse into these topics and should be reviewed.

Online Resources

- https://www.sitepoint.com/set-css-margins-padding-cool-layout-tricks/
- https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/The_box_model
- https://css-tricks.com/the-css-box-model/

Float Property

This property takes content and aligns it to either the left or right side of a container. The content is removed from the normal document flow, but text content and inline elements will wrap around it. Block level elements will appear on the next available line.

To prevent unwanted wrapping of content, you would have to clear your floats. This means you use the clear property on the element which wraps your floated elements. There are many methods to clear floats but the best one I have found is here:

https://css-tricks.com/snippets/css/clear-fix/

To use the clear fix in the link above, you need to wrap all floated elements in a parent container (div usually). Then you target that parent container's before/after and apply the styles as depicted. The example link provided below demonstrates how to do this with multiple rows of content.

Position Property

This property allows us to alter the location of an element in a web page. The six values to consider are static, relative, absolute, fixed, sticky, and inherit. By combining these values with the top, right, bottom, left, and z-index properties, we can move items around and layer them as needed.

By default, every element has a static position and so top, left, etc. will not work on them. When you give an element a position value of relative, those other properties can be manipulated. Absolute positioning will remove an element from the flow of the document. The positioning of these elements would be relative to other containers that have a “relative” position value.

Unlike floats, the other content will behave as if the absolutely positioned content doesn’t exist. Fixed does the same thing as absolute except the content is always relative to the viewport or screen. Sticky is experimental, and I wouldn’t recommend using it for the time being. It behaves similarly to fixed positioning though.

Online Resources
You **do not** have to read through every single link below. They are only here to help you understand the concepts explained above in addition to other methods for positioning content on a web page. Feel free to use them or not but the first ones on Float may be useful for helping you with the assignment.

- https://css-tricks.com/almanac/properties/p/position/ (Position)
- https://developer.mozilla.org/en-US/docs/Web/CSS/float (Float)
- https://css-tricks.com/all-about-floats/ (Float)
- https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox (Flexbox)
- https://css-tricks.com/snippets/css/a-guide-to-flexbox/ (Flexbox)
- https://css-tricks.com/snippets/css/complete-guide-grid/ (Grid)
- https://www.w3schools.com/css/css_grid.asp (Grid) - I’m normally against using w3schools since there are inaccuracies sometimes but this tutorial is solid. Exercise caution when using this website.

## CSS Positioning Exercise (CW/HW)

Download the “CSS-Positioning-Exercise.zip” folder from Blackboard under today’s Lecture Notes or the Assignment folder. Extract everything from inside of this zip folder to some place where you will be able to find them easily. There will be an HTML file, a css folder with a css-positioning-exercise.css file in it, and an image with an example of a finished assignment.

Based on the image provided, add CSS to the css-positioning-exercise.css file so that the web page looks as close as possible to mine. I don’t care if you change colors if the text is readable. However, the items should be styled similarly in terms of their placement, individual styles (borders, padding, margins, underlines, etc.).

To get started, extract all the files from the Zip folder on Blackboard. Open the HTML/CSS files inside of a text editor and add code as necessary. All CSS should go inside of the css-positioning-exercise.css file. You don’t need to include any tags, only CSS should be inside of that document.

General Tips
- If you use floats, float everything in one direction (I personally always use left)

- To center an image, use “display: block;” and “margin: 0 auto;” (images are always inline so when we display them with block, we can add width/height, margins, etc.)
- Padding/Margin will help with spacing (sometimes needs to be set to 0)
    - Use px or em (em is a relative unit and generally, 1em = 1x whatever the default body font size is – sometimes 16px – 2em would then equal 32px)
- Use my CSS file
    - It has code so you don’t have to perform calculations for width (box-sizing: border-box which is mentioned in the reading)
    - The “.row” class declarations will clear the floats for you

Styles Used

To help get you started, I’ve listed some properties/values that were used on my page, so you can get your version looking more like mine. They aren’t listed in any order and aren’t necessarily exhaustive. Values in parenthesis are some that I have used. They might be individually used, or I might use them in shorthand. Play around with the margin, padding, and border values to understand them better.

- Properties
    - Margin (0, 10px, auto, 2em)
    - Background
    - Width (I only use percentages)
    - Max-width (1000px for container div)
    - Color
    - Padding (0, 1em, 10px, 2em)
    - Float
    - Margin-right
    - Margin-bottom
    - Border-bottom
    - Border (2px or .5em for width, solid or dashed for style)
    - Text-align
    - Text-decoration
    - Font-family
    - List-style-type
    - Display
- Colors
    - Navy
    - Red
    - Lightgreen
    - Black
    - White
    - #ccc
- Fonts
    - Verdana

    - Lucida Sans Unicode

For all assignments/project work in this course, your CSS must be **externally placed.** This is because it will be easier to manipulate styles on your page if the styles are separated. I would recommend leaving the ".css" file inside of the folder provided. Otherwise, you will have to change the href value in your link tag.

Your web page is **due October 14th at 6pm.** To submit the work for this exercise, go to Blackboard → Course Materials → Lecture Notes for this week's class. You can also visit the "Assignments" folder and the submission area will be titled "CSS Positioning Exercise" **<u>You must submit a live link to your web page</u>**. The work submitted will be evaluated based on the rubric explained below.

**<u>CSS Positioning Exercise – 10pts</u>**

- Live link was submitted: 1pt
- CSS is externally placed: 1pt
- CSS is organized/nested neatly: 1pt
- Elements on the page are positioned/styled correctly: 7pts

## Participation Questions 10/7

To receive credit for participation in today's class, please answer all the questions in the Google Form linked below. If you don't want to answer the final question (fun), simply put N/A (not applicable) for your answer and that will count.

**This Week's Participation Questions:**
https://forms.gle/D8iNAL1XPsfvGKws6

**Previous Participation Questions/Responses**
These are the participation questions from last week:
https://forms.gle/rMaLpfDhz1gT7UgJ7

This link has the responses to the fun question from last week. Feel free to look over the responses from myself and your peers.
https://docs.google.com/spreadsheets/d/1QLoQzbO-Er7ARIECl0aPvY1lKUyZTdYeuwsn4g0XEAk/edit?usp=sharing

## Next Week

Next week, we will continue practicing positioning elements on our web pages. There is no new reading but feel free to review the links above on floats and applying styles to become more comfortable with CSS.