Week 3: CSS Bootcamp

Agenda

- Overview
  - What's Due?
- CSS Bootcamp
  - What is CSS?
  - How can we add styles to a page?
  - Specificity in CSS
  - CSS Box Model
  - Positioning in CSS
    - Float
    - Flexbox
  - Class Examples
- CSS Exercise
- CSS Research (two posts)
- Next Week

Overview

This week, we will have a lengthy recap of CSS and how it is used to add styles to our page in addition to positioning elements to create various layouts. After reviewing information about CSS syntax, selectors, and properties, students will add styles to a page and submit it for review.

What's Due

- CSS Exercise
- CSS Research (two posts)

CSS Bootcamp

There is a lot to consider when it comes to CSS. This week will serve as a refresher of the major concepts just like we did last week with HTML. There is a lot of content in this document and on Blackboard. If you're familiar with CSS already, you don't have to read all of it. However, if you're not familiar with CSS or if you feel rusty, it's good to review everything.

What is CSS?

CSS stands for Cascading Style Sheets. Unlike HTML, which is used to represent the structure of a web page, CSS is the language for the determining the presentation of a page in terms of colors, fonts, positioning, etc. We use CSS to select specific HTML elements to give them styles.

After selecting specific elements (either directly or based on certain criteria), we can then choose whatever properties we'd like and give them different values. An example would be targeting paragraph tags and giving them a red font color. We can do that with the code below:

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

"p" is the selector (all paragraph tags), "color" is the property, and "red" is the value we are assigning to the property. The selector comes first followed by curly brackets "{}". The curly brackets represent a block of CSS declarations. Inside of the curly brackets is where we choose our properties and values, and each set is a CSS declaration. The properties and values are always separated by a colon and the whole statement (color: red) is ended with a semicolon.

It is **best practice** to separate your HTML and CSS by using an **externally placed stylesheet**. The reason for this is because it is easier to update your code if it is in a different location. If you have a consistent banner style on many pages of your website and use inline/internal styling, you would have to go into the code for each page and edit it. If you use external styling, you can edit the code in one place and the changes would persist across all pages.

How can we add styles to a page?
There are three main ways for implementing CSS on a web page and each one is explained below with a link to an example. To see the code in the example, right-click the page and select "View Source" or "Inspect Element" and look around the window that appears.

Inline CSS

This is typed directly into the HTML tag as part of the style attribute. The line below will do the same thing as the CSS demonstrated above. However, it will only apply to the HTML tag where it resides.

This is a red paragraph.

Internal CSS

This is CSS typed between <style></style> tags which are located inside of the <head></head> tags of your HTML document. I've used this in some of the assignments this semester to give our tables styles.

In the CSS declaration below, I select the table, tr, th, and td tags to give them all a solid black border that is 1px in width. I then collapse the border, so it appears as one line and then add a little padding, so the text is easier to read.

```
<style>
   table, tr, th, td {
      border: 1px solid black;
      border-collapse: collapse;
      padding: 5px;
   }
</style>
```

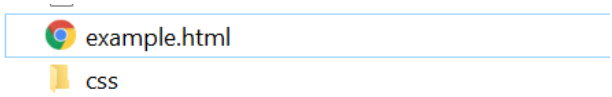
External CSS – BEST PRACTICE

This is CSS placed in a separate document with a ".css" extension. It is connected to our HTML through the usage of a <link> tag which is placed inside of the <head></head> tags. The link tag uses two attributes which are rel and href.

The rel is required to explain the relationship between our resource and the HTML. Here, we are saying it's a stylesheet. The href section is where you link your CSS stylesheet. Like working with local images, you need to include folder/file names accurately to link them correctly. In the example below, my CSS file is located in a css folder which I refer to before the file itself.

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

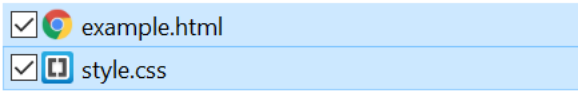
View of my folder structure (style.css is inside of the css folder):



Since I use "css/style.css" my CSS file should be inside of that folder. If I wanted to have my HTML and CSS files be in the same folder, the link tag would be changed so that the href had style.css as shown below.

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

View of my folder structure (they're blue because I selected both items in my folder):



A lot of students tend to put both files in the same folder. I would recommend keeping them in separate areas like they are in the first screenshot. This will keep your files organized and is helpful for assignments/the final project. When you move your files to FileZilla, the folder organization there should be the same as it is on your local machine. **If your CSS doesn't work at all with your HTML, it's probably because the href value is incorrect.**

Online Resources (Not required reading, only meant to help you if you need it)

- https://www.htmldog.com/guides/css/ - CSS Tutorials for all levels
- https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps - Excellent resource with guides/tutorials for CSS
- https://www.tutorialspoint.com/css/css_syntax.htm - Specific syntax examples
- https://css-tricks.com/ - Cool website for code snippets and articles

Specificity with CSS

In CSS, styles are generally applied in order from the top of the CSS file to the bottom. However, if there are multiple styles which target the same element, browsers will follow some

rules to decide which style is the most specific and is therefore applied. Specificity is essentially a score or rank that determines what styles are applied to an element. Each selector has its own "score" and depending on how close that selector is to the element, various things can happen.

Overall, there are four categories which define the score of a potential selector. I've listed them below along with the value they receive when the total score is calculated. Pretend that all the CSS is in an externally placed CSS file (except for inline styles).

!important – This overrides any style, regardless of where it is placed. The only way to override this style is to use another !important style later in the document or with greater specificity. **I would recommend avoiding the usage of !important. You can always find a way to style things with just classes/ids and direct element names.**

Inline Styles (1000) - Any CSS that is directly inside of an HTML tag. This is the closest you can get to an HTML tag, and it is the most specific besides !important.

This sentence will be red.

IDs (100) – An ID is a unique identifier for elements on a page and uses a pound/hashtag sign. You should only use an ID value once per HTML document.

#redPara {color: red;}
This sentence will be red.

Class, Attribute, and Pseudo-Classes (10) – A class can identify multiple elements and uses a period to select elements. You can target an element based on its attribute too ([type="href"] would target all tags with an href attribute). Lastly, pseudo-classes such as :hover, :focus can be targeted as well and receive the same point value.

.class {color: red;}
This sentence will be red.

Type Selectors and Pseudo-Elements (1) – You can target an HTML element directly or use something such as :nth-child() or :first-child.
p {color: red;}
This sentence will be red.

If you wanted to calculate the score or value of a group of selectors, you could simply add them up. The CSS declaration below uses 3 type selectors and would have a specificity of 3.

div p span {color: blue;}

In the CSS declaration below, the total specificity is 12 since we are using a class and two type selectors.

.content p span {color: green;}

.content is worth 10. P and span are worth 1 each. It's important to note that if you have two groups of selectors with the same score, the one placed later in the document will be applied.

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

Since the two lines above both have a specificity of 1, the paragraph would display with green text. This is because the later declaration gets applied. If you have styles in an external style sheet and styles inside of an internal style sheet with the same specificity, the styles in the internal style sheet will be applied since it is closer to the element it is styling. Inline styles would beat internal styles since they are directly in the HTML element.

If a style isn't being applied as you expect, inspect the element in your browser. You'll see a list of styles and you can identify which ones are being applied and adjust your code accordingly.

Online Resources (Not required reading, only meant to help you if you need it)

- https://css-tricks.com/specifics-on-css-specificity/
- https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity (has demonstrations of calculations)

CSS Box Model

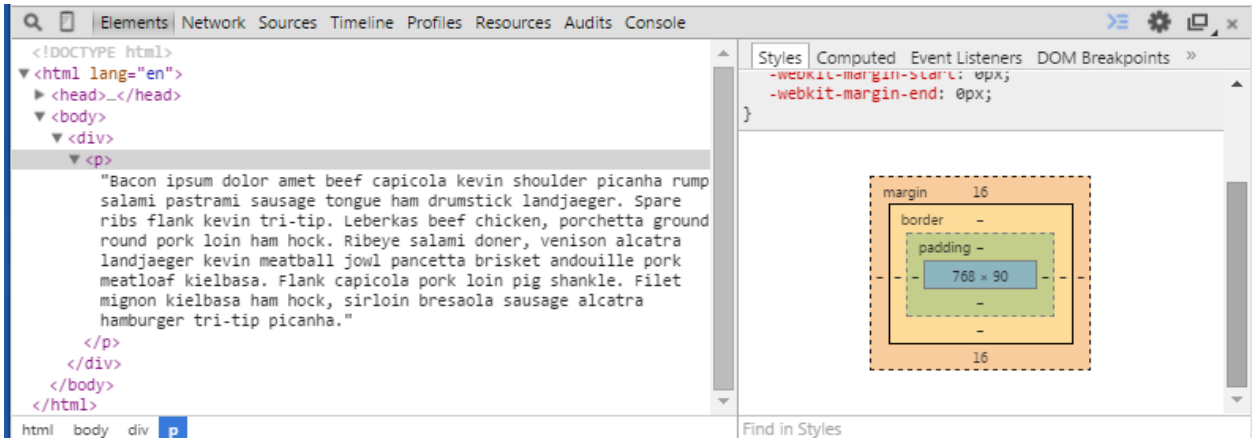
Every element in HTML can be considered as a box. Each of these boxes takes up a certain amount of space on our page relative to the other boxes. Also, the spacing inside of these boxes can vary depending on the content inside of them.

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.



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. The blue area is the content.

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.

TIPS FOR CENTERING CONTENT

There will be times where you want to center an element on your page. If it is a block level element and has a width, you can use the margin property to center it. For example, if we had a table that we wanted to center, we could use the following code to do it:

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

I'm using the margin shorthand to apply 0 margin to the top and bottom of the table. The auto part is applied to the left and right sides of the table. Since it has a width based on its content, the browser will automatically figure out how much margin should be applied on both sides.

If you want to center an inline element such as an image, you need to change its display value to block. After that's done, you can use margin: 0 auto to center it.

```
img {display: block; margin: 0 auto;}
```

That would make the image display just like other block level elements and then center it.

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 (Not required reading, only meant to help you if you need it)

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

Positioning in 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 float and flexbox properties and provide links to some of the other ones as well.

For this course, most of you will probably use the float or flexbox properties since position is harder to implement for a site with more content. Flexbox and grid are great methods and were more recently implemented in the CSS specs. They can accomplish more things but can be complicated in some instances. **I would recommend flex vs float as it's easier to implement, however, you are free to use whatever method you'd like.**

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") impacts the usability of your page.

For most browsers rendering CSS styles, the default width of an element is equal to the width of the content plus the padding and border. When trying to figure out how much margin or width is needed for elements to space them out, you would have to consider the widths of the border or padding as well.

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.

In the example code below, I create a three-column layout based on percentages (fluid-width). I float my columns to the left and give them 30% width, with 1.66% margin on the left and right, and 10px of padding. I got those numbers because I want to use 100% of the width of my screen. 30% for each column is 90%. The last 10% can be divided by 6 since I want equal amounts of space on each side of my divs. That spacing or "gutter" percentage can be up to you.

```
14
15  <div class="row">
16      <div class="one-third-column">
17          <p>Cool Content</p>
18      </div>
19      <div class="one-third-column">
20          <p>Cool Content</p>
21      </div>
22      <div class="one-third-column">
23          <p>Cool Content</p>
24      </div>
25  </div>
```

```
1   * {
2       box-sizing: border-box;
3   }
4   .row {
5       width: 100%;
6   }
7   .row:before,
8   .row:after {
9       content: "";
10      display: table;
11      clear: both;
12  }
13  .one-third-column {
14      float: left;
15      width: 30%;
16      margin: 0 1.66%;
17      padding: 10px;
18      text-align: center;
19      background: black;
20      color: white;
21  }
```

The asterisk on line 1 means that all elements on the page ("*" targets everything) will have the border/padding added into width calculations.

This is what the content looks like when we use box-sizing: border-box as shown in the code above in lines 1-3. This is because the width is calculated like this:

- 30% width for each column's content minus padding or borders (10px padding, 0 border and 90% total)
- 1.66% for each side of each column (~10% total)

When you remove "* {box-sizing: border-box;}" in lines 1-3, the same page looks like the screenshot below. This is because the width is calculated differently as follows:

- 30% width for each column's content (90% total)
- 1.66% for each side of each column (~10% total)
- 10px padding for each side of each column (60px total)
- Any border being added (0 in this case)

That extra 60px of padding will make our total over 100%. Rather than having to think about what widths we can use for padding/borders, we can use the box-sizing property. Box-sizing: border-box is very useful is making it easier to position elements on your page. You don't have to use it, but I strongly recommend it as it is widely used in the web development field and it's more intuitive in general.

The links provided in this document 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.

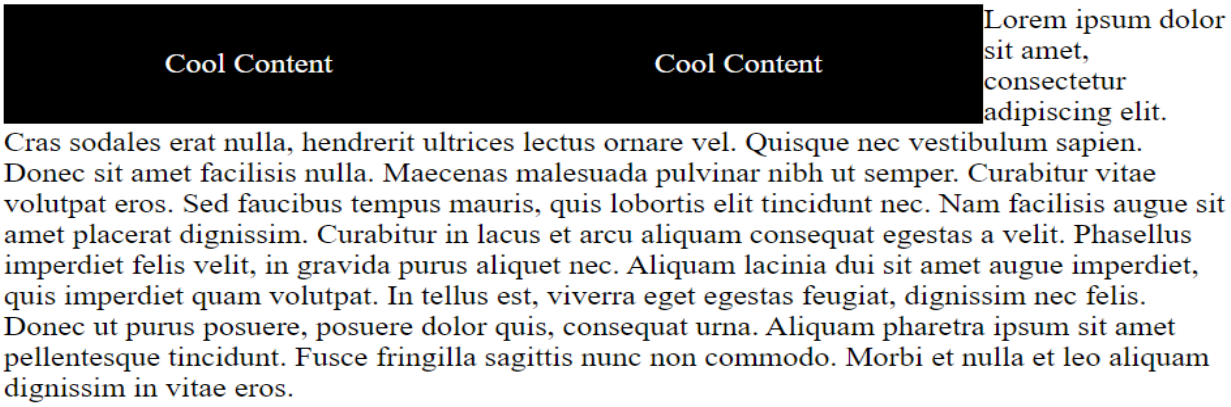
I would recommend percentages for your container widths, and ems or pixels for your padding/margin widths. Em is a relative unit though and is superior for ensuring your page uses a consistent layout. If your body has a font-size of 16px. The other em sizes will be based on 16px. 1.5em would then be 24px since 16 times 1.5 is 24.

Float Property

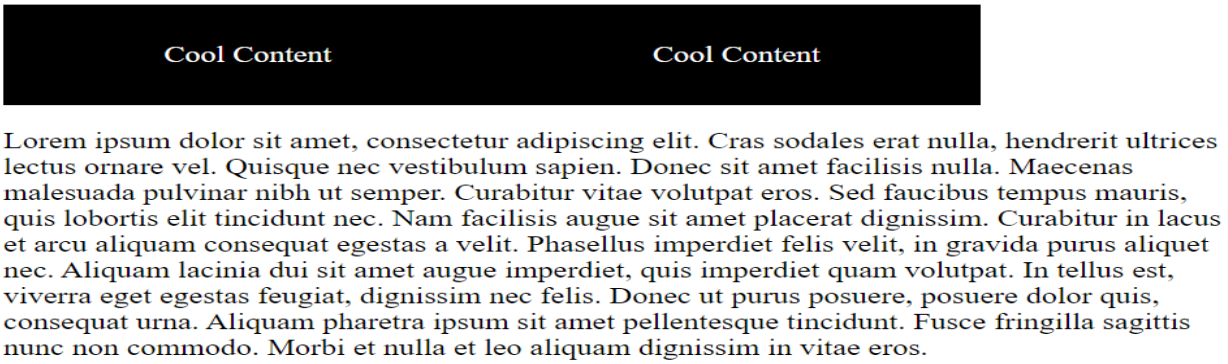
This property takes content and aligns it to either the left or right side of a container. The content is removed from the normal document flow, but text content and inline elements will wrap around it. Block level elements will appear on the next available line.

It was intended originally for having text wrap around images but has been used to manipulate the positioning of elements on a page. The problem is that if the total width of items being floated is under 100%, other items below your floated ones may move up in that space.

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. Unwanted wrapping would look something like the screenshot below.



In the image above, I have 2 divs with 40% width each floated to the left. Currently, they don't have the clearfix implemented and 0 margin to space them out. Since there is 20% of the width still available, content below these divs will try to fill that space. If you use the clearfix method appropriately, the content can have whatever widths you'd like, and content won't move to its side as shown in the next image.



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 pseudo property and apply the styles as depicted. The before/after pseudo selectors allow us to add extra content to our HTML without using anything extra in the HTML markup. However, in our case, we don't display any content (that's why it has empty quotation marks). We display it like a table and clear both sides of it.

If you recall the width example in the screenshots from earlier, I used a div with a class of row to wrap my divs with a class of "one-third-column." This declaration allows us to float content inside of any div with a class of row and give them any width we'd like without having other elements wrap around them. You don't have to use it for the class assignments, but it is helpful if you're using floats to position elements.

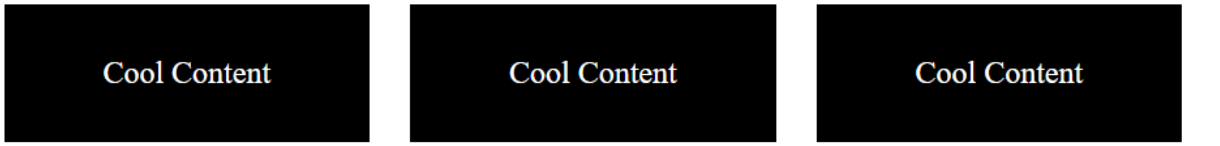
Flexbox

Flexbox is an awesome property which allows you to create layouts without using floats or positioning. Similar to floats, it requires the usage of a parent container which has other items/containers inside of it. Unlike floats, you don't need to worry about clearing anything and it's easier to implement. If we used the same HTML code as we did previously to create a 3-column layout, the CSS would look like this instead:

```
<div class="row">
    <div class="one-third-column">
        <p>Cool Content</p>
    </div>
    <div class="one-third-column">
        <p>Cool Content</p>
    </div>
    <div class="one-third-column">
        <p>Cool Content</p>
    </div>
</div>
```

```
* {
    box-sizing: border-box;
}
.row {
    display: flex;
    width: 100%;
}
.one-third-column {
    width: 30%;
    margin: 0 1.66%;
    padding: 10px;
    text-align: center;
    background: black;
    color: white;
}
```

Rather than using the row class for clearing, just simply give it a display of flex and a width of 100%. From there, we apply all the same styles to the children containers (one-third-column) with the exception of floating it. The final result would look pretty much the same, but with less effort as far as clearing floats and choosing float directions:



As a quick recap, for flexbox, you need a flex container and flex items. The flex container is just the parent container which wraps the elements you want to move around. The flex items are the elements inside of the container which will be displayed horizontally or some other way.

The reason you would use float over flexbox is because float is more universally implemented by browsers. Some older browsers don't support the flexbox property, and then your styles wouldn't work as expected. It's important to understand your audience and the types of devices/browsers they are using. For the purposes of this class, flexbox and floats are acceptable for assignments and the final project.

There are many other properties for the flex containers/items, so make sure to review them to fully understand the possibilities with flexbox. My example above is useful for creating quick multi-column layouts.

Online Resources (Not required, but can be helpful for guidance)

- 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)

Class Examples
All the links below point to pages that I created to give you an example of the concepts mentioned above. Feel free to right-click the page and view the source. You can then click the link to the CSS file to see my code which has comments placed inside of it.

This link covers the box model, floats, and how to apply basic styles.
https://www.albany.edu/~cv762525/cinf362/examples/box-model-float-example.html

This link is an example of a basic page with comments throughout it explaining how/why I did it that way.
https://www.albany.edu/~cv762525/cinf362/examples/basic-page-example.html

This link shows how to use some basic flexbox features to structure your content.
https://www.albany.edu/~cv762525/cinf362/examples/flex-box.html

This link is an example of a page which uses floats to structure content. The content is the same as the flexbox example but done with floats instead.
https://www.albany.edu/~cv762525/cinf362/examples/floats.html

Here's a link to all course examples:
https://www.albany.edu/~cv762525/cinf362/examples/ (coded examples)
https://www.albany.edu/~cv762525/cinf362/videos/ (videos)

There is also content on Blackboard to help you with understanding/debugging CSS code. These items are located on Blackboard → Resources → CSS and will help further explain CSS and demonstrate some best practices you should adopt moving forward.

Before completing the assignment(s) for this week, please read the "Viewing Your Web Pages.docx" file on Blackboard. You will not be able to submit anything for the assignment without completing that portion first. It is located directly in the Lecture Notes folder.

CSS Exercise

Due February 13th at midnight

Download the "CSS-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, a CSS file, and an image which depicts what the final product should be after you've completed the assignment.

Based on the instructions below, add CSS to your style sheet to alter the way the page looks. The image I've provided is an example of how the assignment should look after you are done with it. Your page should look pretty close to mine if you follow the steps outlined below. You can use different colors/fonts if you want, but the general properties should still be used.

Instructions for Adding CSS

These instructions are for those of you who choose to use floats for your page. If you want to use flexbox, you can set the display of the parent containers to flex, and then alter the widths/margins of the flex items (child elements) from there. You could realistically leave the .row styles and add display: flex to the .row CSS declaration for this part.

You can also use other methods as you see fit, but I wouldn't recommend trying to use the position property as it is cumbersome and difficult to do. Please note that this list isn't exhaustive, it will help get you started. **You may edit the HTML to better fit your positioning method. The only caveat is that the content should remain the same on the page as far as text, images, links, etc.**

General/Body Styles

- Add a font-size of 16px to the body and set the margin to 0
- Give the container a max-width of 900px, a width of 100%, and center it with margin: 0 auto
- Most containers have a padding of 10px
- Border-radius has been set to 10px for any rounded corners
- Border-radius has been set to 50% for circular images
- Link tag colors have been altered
- Images have a max-width of 100%

Header Styles

- I gave the header a height of 4.5em, a black background, a white color, and padding of 0 on top/bottom and 10px on the left/right sides
- The h2 is floated to the left with some margins on it
- The nav has a margin on top of it (1.8em) and is also floated to the left

Aside Styles

- Aside has been floated to the left and has a width of 30% and a margin on the right side of 5%
- Both images are centered, the first one has a 2px solid black border, the second one is circular

Main Styles

- The main class is also floated to the left with a width of 65%

- The images in the section tags are floated with a margin around them
- The paragraphs In the section tags have no margin on top
- The blog posts have a width of 45% and margins of 2.5% on all sides
- The images/containers in the blue blog posts have a top-left and top-right border radius applied
- The blog post h4 tags have margin applied: .5em .5em 1em
- The blog post p tags have padding applied: 0 .5em

Footer Styles

- The footer has a back background with white centered text

If your CSS isn't working at all, it's probably because the href value you've provided in the link tag is incorrect. If your CSS is working, but some styles are not appearing as you expect, check for missing semicolons and make sure you aren't overriding your style in some other declaration that is more specific.

Your webpage is **due on Sunday, February 13th at midnight.** To submit the work for this exercise, visit Blackboard → Course Materials → Lectures Notes for this week's class. You can also go into the "Assignments" folder and the submission area will be titled "CSS Exercise." You should be submitting a live link to me; it can be through the UAlbany server or 000webhost. The work submitted will be evaluated based on the rubric explained below.

CSS Exercise Rubric – 10pts

- External CSS file is used – 1pt
- Live link submitted – 1pt
- CSS is organized/easy to read – 2pts
- Correct styles are applied – 6pts

CSS Research (Two Posts)

Initial Post due Thursday, February 10th

Do some research on any of the CSS properties listed below and report your findings. In addition to writing about what you discovered, you must create an original example demonstrating some capability of that CSS property and submit a link to that example with your initial post.

The example doesn't have to be very elaborate, but it should clearly depict what the CSS property you chose does and include an explanation of how it works. You can either submit a write up with a link or you can submit the write up inside of the HTML file used to create your example. I've listed some starting points, but you can include other pieces of information you find that are interesting as well. If you use code from some website for inspiration, please include a link to the page on your page or as a comment in your submission.

- What is the property?
- What does it do?
- Does it require other properties for its usage?
- What version of CSS was it introduced in?

- What browsers support this property? Which ones don't?
- Do you think this property is used very often?

Properties to Research

- Visibility
- Text-shadow
- Transform
- Clip
- Float
- Grid
- Text-transform
- Content
- Table-layout
- Animation
- Position
- Border-radius
- Box-shadow
- Flex
- Overflow
- @font-face
- Text-decoration
- Z-index
- Word-wrap
- Media queries
- Filter

The initial post is **due Thursday, February 10th at midnight**. To submit, go to Blackboard → Course Materials → Lecture Notes for this week's class. There will be a discussion area called "CSS Research" where you can post your initial post. You can also visit the Discussion Board area directly from the Course Materials folder.

Response Post due Sunday, February 13th
In your response post, choose a property done by another student that isn't the same as the one you selected initially and create an example for it. In your post, you should include a link to this new example and an explanation as to what you did differently. If you found out some piece of information the student didn't list, please share it.

The response post is **due Sunday, February 13th at midnight**. To submit, go to Blackboard → Course Materials → Lecture Notes for this week's class. There will be a discussion area called "CSS Research" where you can post your response post. You can also visit the Discussion Board area directly from the Course Materials folder.

The initial post is worth 1.5pts and the response post is worth .5pt for a total of 2 points. I will be evaluating your posts based on the following criteria:

- Did you pick a valid property from the list?
- Was a link to an example included?
- Was the example sufficient to demonstrate the property's capabilities?
- Does your response post include a link/explanation of what you did differently?

Next Week

We will learn about responsive web design and how it can be implemented on our web pages. Specifically, we will learn about the viewport meta tag along with media queries. If you'd like to get ahead on the material, feel free to skim through the links below.

1. https://www.pewresearch.org/internet/fact-sheet/mobile/
2. https://developers.google.com/web/fundamentals/design-and-ux/responsive

3. https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries