Week 4: Responsive Web Design (RWD) Part 1

Agenda

- Overview
  - What's Due?
- Responsive Web Design (RWD)
  - What is it?
  - How do we use it?
    - Width
    - Max-width
    - Font
    - @media
    - Flexbox Properties
    - Float
  - Class Examples
- First RWD Exercise
- Next Week

Overview

This week, we will be learning about Responsive Web Design and how it's used to help pages work on multiple mediums regardless of device, screen size, orientation, browser, etc. By the end of this week, students will become familiar with various methods for ensuring their pages are fluid and responsive. Specifically, we will cover some CSS concepts such as media queries and the viewport metatag.

What's Due

- First RWD Exercise
- Project Proposal

Your **Project Proposal** is due this week. Please review the requirements inside of the "Final Project Description" document on Blackboard → Final Project. In total, you should be submitting one needs assessment, three wireframes, and one style tile to describe your website. Make sure to include as many details as possible so that your project idea is clear to me. Project proposals lacking details or missing components may not be approved.

Responsive Web Design (RWD)

What is it?

Responsive web design (RWD) is an approach to web design that incorporates a variety of techniques to help pages render on numerous devices depending on screen sizes, orientation, browser, etc. Ideally, the page layout would change based on the previously mentioned factors without forcing a specific view on the user.

How do we use it?

Responsive design is implemented primarily through CSS. By using certain properties and values, we can make the content on our pages enjoyable on any device or screen. Below is a list of properties and methods for using them on your page to increase its responsiveness.

Width

You can use several types of units with the width property to create interesting layouts. However, percentages are the easiest to use when it comes to container widths. If you have a two-column layout, 50% widths on each container will ensure that the content from each container only takes up half of the screen.

Using pixel-based widths isn't recommended. This is because containers with pixel-based widths will not adjust to the size of the screen and will always take up the same screen size. If the provided width value is smaller than the width of the screen, a horizontal scrollbar will appear, and this is bad for usability.

For example, if you have a container that is 900px wide, once the screen size is below 900px, content will be displayed off screen. The user would have to zoom out on their screen or scroll to the right. Both of those options result in poor user experience.

Max-width

This property is great when paired with width for containers. You can give a container a width of 100% so it's always 100% of its container or the screen. Adding a max-width of 900px would make it so that the content is responsive but won't be wider than 900px at any point.

For images, this property is incredibly useful. You can use img {max-width: 100%} and this will almost guarantee that all images on your page are responsive. They won't go beyond the width of their container and will shrink as the page shrinks as well.

Font Styles

The font on a page may initially be very large. However, as the screen size decreases, it's important to resize your fonts. Font-size is best for altering the size when a page increases or decreases. It's a good idea to do your font sizes in ems since that is a relative unit. Pixels are okay, but relative units scale depending on the structure of your HTML.

Line-height and letter-spacing can also be useful for making fonts more readable on various screen sizes. A good value for line-height is 1.3 or 1.4, but you can play around with values until you see something you like.

@media()

Media queries are an important part of responsive design. More modern properties such as flex and grid make it so that they aren't entirely necessary, but most responsive websites make usage of media queries. Media queries are rules with styles we can place in our CSS that allow us to change elements on our page. For example, you can have a two-column layout that changes to a one-column layout when there isn't enough room on the screen.

To use media queries, you need to place the following tag in the head tag of your HTML document. It's known as the viewport meta tag and without it, your web pages wouldn't look so good depending on the device/browser being used.

```
<meta name="viewport" content="width=device-width, initial-
scale=1">
```

Without this tag, we cannot use **media queries**. Media queries are useful for changing styles on your page as necessary. This can include hiding content, making content appear, or altering content that's already visible.

In the image below, we have a basic p tag with some content inside of it. With a media query, we can change how this paragraph looks based on a screen size we choose.

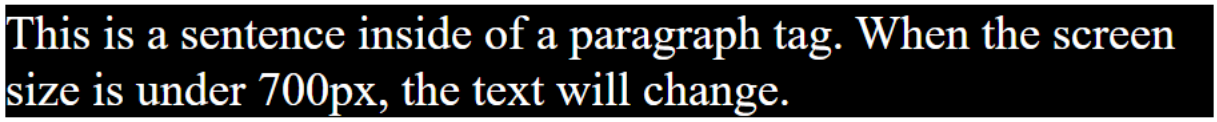
> This is a sentence inside of a paragraph tag. When the screen size is under 700px, the text will change.

The syntax for media queries is: @media(some rule) {styles in here}

We replace "some rule" for something like max-width, min-width, etc. and then provide a screen size which will act as a trigger for the styles inside. In the screenshot below, we have a media query set to max-width of 700px. If the screen width is 700px or below, the styles inside of the curly brackets will execute.

```
@media(max-width: 700px) {
    p {
        color: white;
        background: black;
        font-size: 1.5em;
    }
}
```

Once my screen width gets below 700px, the content will look as it is shown below.



The "700px" portion of my code is known as my "breakpoint." This is a point in the page where things typically begin to break in terms of appearance. We can then apply styles to fix the look of the page. Determining your breakpoints shouldn't be hard. All you do is resize your page in a browser and look for times where content becomes squished or inaccessible. You can then add a breakpoint near that width and add your styles.

In the examples above, I used "max-width" with my media query. Max-width is typically used for when you're going from a desktop screen to a smaller screen. It essentially says, from 0 up until this screen size, apply these styles. Min-width works from smaller screens to larger ones. It is like saying, from this minimum width and higher, apply these styles.

Flexbox Properties

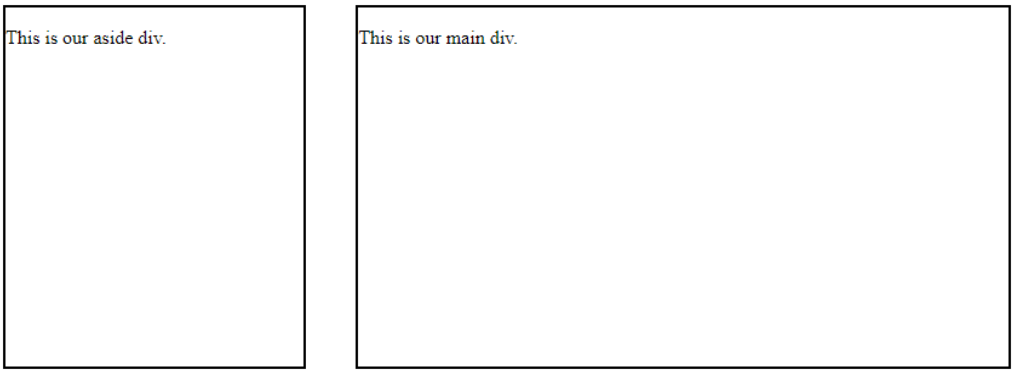
As you saw in last week's lecture notes, flexbox can be used to create quick multi-column layouts. Flexbox has many properties which allow us to also build responsive layouts with ease.

If you recall, flexbox works by using "display: flex;" on the parent container of content you want to appear in one row. Creating a two-column layout on a page would look something like what is shown in the screen shots below. Flex-wrap will wrap our content onto the next line automatically if it's needed. When using percentages, this won't have an effect. However, if you used pixel-based widths, content will move to the next line automatically if there isn't enough space.

```html
<div class="flex">
    <div class="aside">
        <p>This is our aside div.</p>
    </div>
    <div class="main">
        <p>This is our main div.</p>
    </div>
</div>
```

```css
* {box-sizing: border-box;}
.flex {
    display: flex;
    flex-flow: row wrap;
    width: 100%;
    max-width: 900px;
    margin: 0 auto;
}
.aside {
    width: 30%;
    margin-right: 5%;
    height: 300px;
    border: 2px solid black;
}
.main {
    width: 65%;
    height: 300px;
    border: 2px solid black;
}
```

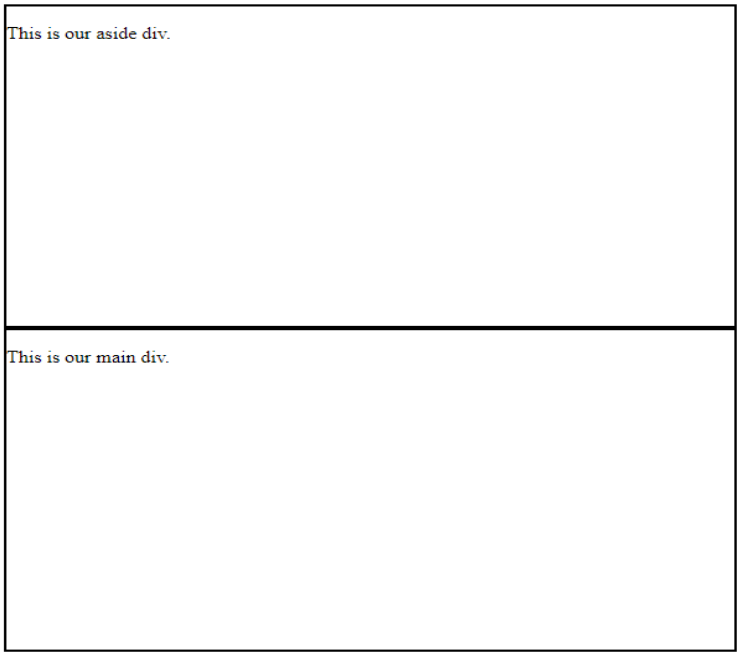
That code would produce the following layout on our page



If we wanted to turn this into a one-column layout when we got to a smaller screen size, we could use a media query combined with the flex-direction and width properties as demonstrated in the screenshot below.

```css
@media(max-width: 700px) {
    .flex {
        flex-direction: column;
    }
    .main, .aside {
        width: 100%;
    }
}
```

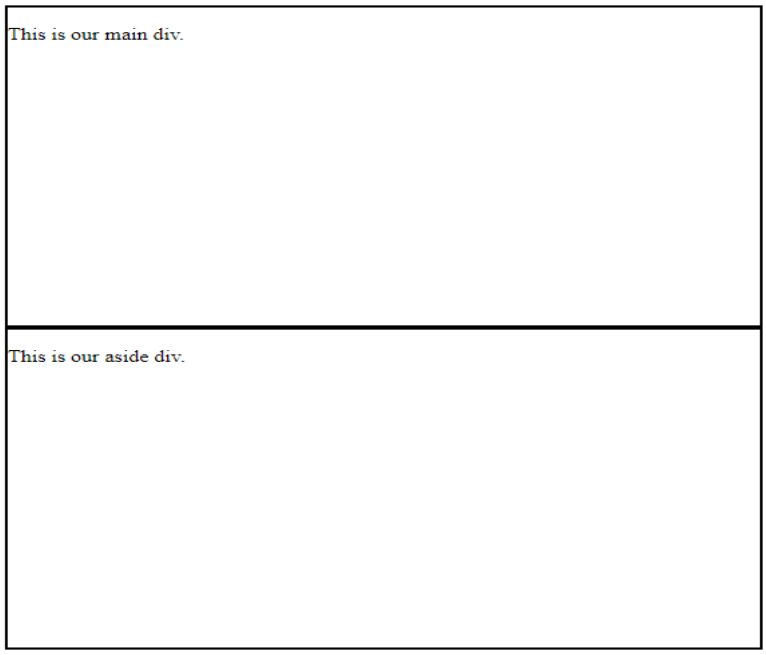
In the media query above, we change the flex direction to column. By default, flex box makes items fit in a horizontal row. By changing the direction to column, we make the boxes stack on top of each other. The only issue is that the old widths remain, so we use the next declaration to change the widths to make our containers full-width (100%). You can use less width and apply margins to alter the look as you'd like. The code above would result in the following layout once our page became smaller than 700px.



Other properties for flexbox include “order” and “justify-content.” Order allows you to change the order of content on the page and justify content allows you to alter the spacing of your elements.

If you wanted to change things so that the “main” div appears before the “aside” div content, you could use the following code in a media query which would result in the structure on the right. Notice how main appears before aside even though aside comes first in the HTML. The default is for content to stack in the order it is presented in the HTML, but order allows you to change this as you need.

```
.aside {
    order: 1;
}
.main {
    order: 0;
}
```



Floats

The same concepts in flexbox apply when using floats. When you have multiple items floated side-by-side as is done in this week's assignment, you would have to alter the float property. If something is floated left or right, you could change it to "float: none;" which removes the float on that item. You would then alter the width as we did above for flexbox. This can be seen in the class example titled "basic-responsive-example."

Flexbox is simpler when it comes to structuring content in various ways. There are many properties and I don't expect you to remember them all, but it's worth knowing that there are properties for the parent container, and a separate set of properties for the children elements. I'd recommend flexbox over floats when creating new pages.

I've provided an example link below the "Online Resources" with some basic examples of responsive design using floats and inline-block (inline-block can also be used for positioning). Please view the page source or inspect the page for comments in the HTML/CSS with explanations of what's going on.

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

- https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag (Viewport)
- https://developers.google.com/web/fundamentals/design-and-ux/responsive (General)
- https://css-tricks.com/logic-in-media-queries/ (Media Query Logic)
- https://www.smashingmagazine.com/2018/02/media-queries-responsive-design-2018/ (Alternative to floats/media queries)
- https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox (Flexbox)
- https://joshnh.com/weblog/why-you-should-use-inline-block-when-positioning-elements/ (Argument for inline-block vs floats)

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 shows how to use some basic media queries to change the structure of a page.
https://www.albany.edu/~cv762525/cinf362/examples/basic-responsive-example.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.

First RWD Exercise

Your task this week will be to take a static HTML page and make it responsive using CSS. I am providing you with a CSS file, so you only need to add styles based on the instructions I provide.

Due February 20th at midnight

Download the "First-RWD-Exercise.zip" folder from Blackboard under this week'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, and a CSS file.

Open the page in a browser and try to resize the page. You'll notice that the content doesn't resize well, and a horizontal scrollbar appears. Your task is to add or alter CSS so that this page becomes responsive down to a size of 400px. This means that the font sizes, margin, padding, positioning, etc. will have to be changed so that all the content is present on the screen at smaller sizes.

You can change the HTML/CSS as needed to complete this assignment. However, all content in terms of text/images should still be present.

There are a few ways to fix this page and I've provided a short list of tips to help get you started. The page was built with floats, so some of these tips will be related to floating. If you decide to use flexbox, that is fine. You can ignore the float portions, but the width calculations will still be relevant.

General Tips

- For containers and the row class, change the pixel-based widths to percentages.
    - For example, the .container class has a width of 900px…this is obviously not very responsive. Changing it to have a width of 100% with a max-width of 900px would be better and one way to make it responsive.
    - If you're not sure what percentages to change the widths to, based it on 900px. If you see 450px, the width could be changed to 50% or something similar.
- The total width for content in general shouldn't be above 100%, also look at margins and padding for your containers
- For images, use the max-width attribute
- Change the size of text on the page
- For containers floated side-by-side, change the float to none so they no longer float and give them a width of 100%. If you don't want to use floats, feel free to use flexbox instead. You'll have to alter the CSS a bit more to try this.
- If you use media queries going from a larger screen to a smaller one, use max-width. If you are going from a smaller screen to a larger one, use min-width

- Consider your "breakpoints" and space them appropriately. You only need a handful to make a page responsive
- Your smallest media query shouldn't be below 400px and your largest one shouldn't be above 1200px at least for this assignment
- There shouldn't be a horizontal scrollbar when I view your page in a smaller browser window

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 20th 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 "First RWD 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.

First RWD Exercise Rubric – 10pts

- External CSS file is used – 1pt
- Live link submitted – 1pt
- CSS is organized/easy to read – 2pts
- Content is responsive down to 400px – 4pts
- Content transitions are smooth (avoid having squished content, unreadable text, and horizontal scrollbars when the page gets smaller) – 2pts

Next Week

We will learn more about responsive web design and the various approaches to implementing styles, so our content works on as many devices/screens as possible. Concepts such as mobile-first design, progressive enhancement, and graceful degradation will be explored.