Responsive Web Design (RWD)

Responsive design is design which changes across different screen sizes. A responsive website layout will shift and items rearrange themselves based on the screen width. How this works is through the concept of breakpoints. A breakpoint is the screen width at which the layout or styling (e.g. font sizes) will change.

The typical breakpoints used are:

- 1200px: this is now a common breakpoint since 1366 x 768 became a popular screen resolution
- 1024px: this breakpoint and larger is typically a desktop layout (under is a larger tablet layout)
- 768px: under this point is the tablet layout (portrait)
- 480px: this is usually the landscape width of a phone screen
- 320px: this is usually the portrait width of a phone screen

(For phone layouts, you may use 480px or 320px or both. It depends on your design.) Currently, with 1366 x 768 as the most common resolution, 1200px is quite common for desktop.

You may find that sometimes, in between breakpoints, the design looks a little messed up. Usually, this means that you will need to add a breakpoint somewhere in between or make your design shift gracefully or consider the units used. Are you using relative units where possible? If the bulk is in pixels, consider using percentages and ems or vw/vh (viewport width/viewport height, respectively).

To add styling based on breakpoints, you use what is called a CSS media query. Media queries were added to CSS3 to allow you to target certain screen widths (or browser window width) to style.

To ensure your media queries work as intended, add the following in the <head> of your HTML file:

```
<meta name="viewport" content="width=device-width">
```

or (setting the initial zoom to 1)

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

This line just tells the browser to **scale appropriately for the device**. This is important because 1920 x1080 on a phone is a much different screen size (physically) from 1920 x 1080 on a desktop due to pixel size and density. (Smaller screens are held closer to the face so pixel sizes are smaller.) Without this line in your HTML, only your desktop layout will show in your phone since newer phones have high resolution.

You may come across maximum-scale=1 in older resources. Do not include this because this disables pinch to zoom. Initial-scale=1 just sets the initial zoom level to 1.

Read more

https://css-tricks.com/snippets/html/responsive-meta-tag/

Now, in your CSS, you can add a CSS media query:

```
@media screen and (max-width:768px) {

  /* add your styles here */
  /* these are styles which work for any screen 768px wide and
     under */
  /* adjust your styles to shift for tablet */

}
```

Another example:

```
@media all and (min-width:481px) and (max-width:768) {
  ...
}
```

The `all` keyword means that the query targets all media types (screen, handheld, projector, etc.).

Note that depending on how you use max- or min- width, you may have to be careful about the order. Remember that CSS is read top-down.

Tip!

You can do a `@media print` query to create styles for printing. This is how printer-friendly versions of pages are created.

```
@media print {

 /* styles for printer-friendly version – you can use display:none for things that don't need to be printed */

}
```

Styles outside the media queries are your default styles. This means if you use max-width for your media query conditions, your default styles outside the media queries would be your desktop styles. If you use min-width only, your default styles would be mobile styles and the media queries would be for tablet and desktop layout. Anything you set in your default styles will still apply so you don't and **should not** redo all styles in the media queries. **The media query styles are for overrides you want to apply for the specific screen widths!**

Note

I usually stick with either min-width or max-width and **don't bother with both** because depending on how a browser calculates width, you may end up with a specific width which falls on the breakpoint that

breaks oddly.

Example

The following will show how media queries work. In tablet mode, we want the second sidebar to move below the main content. In phone mode, the first sidebar should also move beneath the main content. In phone mode, the main menu links should also be displayed one-per-line.

1. Use the 3-column flex layout example we did in week 4: (or use any other layout you want).
2. Add the following media query to the end of the *CSS* file:

```
@media all and (max-width:768px) {
}
```

3. To move the second sidebar below the main content, add the following within the media query:

```
@media all and (max-width:768px) {
  #sidebar-two {
    flex-basis: 100%; /* force to full width */
  }
}
```

4. Save the CSS file and view the changes in your browser.
5. Resize your browser window width to view your new styles at work.

> *Note*
>
> Notice that the first sidebar and the main content will stay in proportion and stretch to fill the gap where the sidebar was before. This is because we used proportions and flexbox stretches to fill the container. If using percentages for the flex-basis property with 0 for flex-grow, the flex items will not grow to fill the space. (Note that if you've used margin between flex items, you'll need to remove the stray margin next to the main. If you used gap to add spacing between flex items, you don't need to worry.)
>
> If you've used a margin, the #main may also be touching the sidebar below. You would also need to add a bottom margin to #main in that case. If using gap, this is not a problem because gaps will be added in between any flex items within the flex container with a gap defined.

6. Now make the first sidebar move below #main for a phone. For a phone, both the first sidebar and `#main` should stretch all the way across the screen. Make sure you put the following **below** the above media query (because we're using max-width).

```
@media all and (max-width:480px) {
  #sidebar-one, #main {
    flex-basis: 100%;
  }
  #main {
    order:-1; /* alternatively, set #sidebar-one to have order:0 */
```

```
  }
}
```

> *The order of your breakpoints*
>
> In this example, we're adding our media queries such that the breakpoints are going from the largest width to the smallest width. This is because we are using max-width only. If you're using max-width but the order goes from smallest to largest, some mobile CSS rules may be overwritten by tablet CSS rules because phone widths will still be smaller than a max-width of 768px (tablet).

7. Now for the main menu links. We want our main menu links stacked in mobile.

```
@media all and (max-width:480px) {
  #sidebar-one, #main {
    flex-basis: 100%;
    margin-right:0; /* don't need if gap property used */
  }
  #main {
    order:-1; /* alternatively, set #sidebar-one to have order:0 */
  }
  #main-menu li {
    display: block;
    width: 100%;
  }
}
```

Some Common RWD Ideas

There are some common strategies and things to do to make your responsive website more usable:

1. Make your *images* responsive as well so that they do not stretch beyond the screen size.
2. In phone size, have an element which acts as a menu toggle to display or hide the main menu.
3. Think of font sizes and how readable they are.
4. Think of link sizes and paddings (especially in footer menus) and about how easy it will be to tap links.
5. Think about how much scrolling is involved.
6. Have a "jump to top" at the bottom of the page so that users don't have to scroll all the way up.
7. Think about whether or not there is any unnecessary information which can be hidden for smaller screens.
8. Start thinking in ems, rems, and percentages (which is why I had you start with percentages for layouts from the get-go). Interesting units are **vw** and **vh** which can be useful for text which is always large to ensure they don't shrink too small. (Note: vh can also be used for creating sections with distinct backgrounds that should always fill 100% height of the viewport.)
9. If there's a lot of stuff before the main content, you can (and should) add a "jump to main content."

Responsive Images

To make images responsive, use the following CSS:

```
img {
  max-width: 100% !important;
  height: auto;
}
```

The above `max-width` means that the image cannot be any larger than 100% of its container.

The `!important` keyword means that even if you style an `img`'s max-width later (perhaps by accident), the style declaration will not be overridden. Essentially `!important` means that you cannot change the style because it has the highest priority. For this reason, you should not use `!important` often. You want to be able to override any style to take advantage of the cascading nature of CSS.

Using `!important` too much will make styling very difficult. In fact, the only time I ever use `!important` is for this purpose only (responsive images). In most cases, you probably won't even need it for responsive images.

Menu Toggles

This is simply an HTML element in the page which you use to toggle the menu in smaller screens. (You will hide this in larger screens and make it appear in smaller screens using CSS.)

This toggle button should reside within <nav> because it is navigation-related.

Example:

```
<button class="menu-toggle">Menu</button>
<ul id="main-menu">
  <li><a href="/">Home</a></li>
  <li><a href="#">Shop</a></li>
  <li><a href="#">About</a></li>
  <li><a href="#">Contact</a></li>
</ul>
```

Keep the toggle hidden in larger screens, so in your non-media-query CSS, put:

```
.menu-toggle {
  display: none;
}
```

In your CSS, you will need the following for small screens (i.e. in the media query):

```
.menu-toggle {
  display: block;
}
#main-menu {
  display: none;
```

```
}
#main-menu.show-small {
  display:block;
  position: absolute;
  left: 0;
  width:100%;
}
```

You will then toggle the menu via Javascript or jQuery. This toggle button **should be within the <nav>** but outside the <ul>. (**The toggle function will hide/show the <ul>.**) The reason for this is because the menu toggle is still navigation-related so it belongs in the <nav>. This is very important for accessibility. If you hide the <nav> element itself, that means that—to a screen-reader user—your page has no navigation available at all.

In your Javascript file, you will need:

```
function toggleMenu() {
  var menu = document.querySelector("#main-menu"); //or getElementById
  menu.classList.toggle("show-small"); //toggle class on and off
}
```

You may need to adjust your CSS to adjust the position and styling of the links but the above is the gist.

Now, you can add the Javascript function to the menu toggle to be toggled on click. (Note: You can put a menu icon within the button opening and closing tags.)

```
<button class="menu-toggle" onclick="toggleMenu()">Menu</button>
```

Note

If you're using jQuery, the above Javascript can be shortened to:

```
function toggleMenu() {
  $("#main-menu").toggleClass('show-small');
}
```

Viewport-Sized Text

You can also base your font size on the viewport width using the unit "vw" where 1vw is equal to 1% of the browser width. https://css-tricks.com/fun-viewport-units/ (note the CSS calc() function in the linked page; this function can sometimes come in handy when positioning things)

Caution!

Only use viewport width **for large text such as banner text**. For more regularly-sized text, you will end

up with overly small text for smaller screens. Although you can add breakpoints to change the font-size to some other unit for smaller screens this is more inefficient styling than it needs to be.

Beyond media queries, there are a few things you can use to make your responsive page work better. The most useful because of wider support are image sets.
min(), max() and clamp() are for **very new browsers only**.

Image sets

When dealing with responsive webpages and images, in some cases it makes more sense to serve a different-sized image (file size-wise) based on device size. This can save on bandwidth.
In other cases, you may want to show *different* images depending on device size. For example, you want a cropped image only showing the important parts. This is useful because some images don't look appropriate if just adjusted smaller. For example the subject in the image is somewhat small, so if you shrink this for mobile, the subject may look too small.
Both the above cases will require different strategies outlined below. In the case of both strategies, the images are defined in HTML and the browser will select the most appropriate.
The content below are simple use cases. If you want to dive even deeper, take a look at the following resources:

- https://css-tricks.com/a-guide-to-the-responsive-images-syntax-in-html/
- https://www.smashingmagazine.com/2014/05/responsive-images-done-right-guide-picture-srcset/

img srcset

This strategy is for providing the same image but at different resolutions. The src can be the lowest resolution while srcset provides alternative resolutions. Notice there is a #x providing pixel density. A browser will determine which image is most appropriate for the current device's pixel density and serve the best choice out of the images.

```
<img alt="A cat staring at a mouse"
  src="cat_lower-resolution.jpg"
  srcset="cat_high-resolution.jpg 2x"
>
```

You can provide more pixel densities by separating the value within srcset (only ONE srcset) with commas, but after a certain point this becomes pointless.

```
srcset="cat_high-resolution.jpg 2x, cat_4k-resolution.jpg 4x"
```

picture and source

When you need to provide slightly different images based on device size, you can use the <picture> element with different <source> elements defined, where each <source> defines an image. (See next page.)

```
<picture>
  <!-- for screens larger than 768px wide -->
  <source media="(min-width:768px)" srcset="cat-on-beach.jpg" />
```

```
  <!-- for anything else - in this case, smaller than 768px wide -->
  <source srcset="cat-on-beach-cropped.jpg" />

  <!-- required - no images show if there's no img -->
  <img alt="Tuxedo cat sitting on beach" src="cat-on-beach.jpg">
</picture>
```

min()

This CSS function is used where you are setting some sort of length (e.g. width property) and chooses the smaller of the two. Effectively, this sets the **maximum** length the element can grow.
Example. For the width of an element, you want the element (e.g. named #box) to be the smaller of 500px or 30% of its parent container.

```
#box {
  width: min(500px, 30%);
}
```

This is the equivalent of:

```
#box {
  width: 30%;
  max-width: 500px;
}
```

Notice that in the equivalent rule, we're using max-width (the maximum length the element can grow).

max()

max() is similar but it chooses the larger value. This effectively sets a **minimum** length an element should have.

```
#box {
  width: max(300px, 50%);
}
```

This can also be useful for setting responsive font sizes where you want the text to shrink but only up to a certain point.

Example. If you have large text over a hero image, you want the large text to shrink with the image but not get too small.

```
#banner .banner-title {
  font-size: max(5vw, 2em);
}
```

clamp()

clamp() allows you to set a minimum value, ideal value, and maximum value in one line.
For example, using the example of a large title, you don't want your banner text to get too large nor do you want to let it get too small, so you need to clamp it down to a minimum and maximum, but ***ideally*** you want 5vw as the font size.

```
.banner-title {
  font-size: clamp(2em, 5vw, 4em);
}
```

More responsive mobile menu considerations

The link below has a good rundown of some of the considerations for mobile menus with submenus.
https://webdesign.tutsplus.com/articles/best-practices-for-responsive-dropdown-menus--cms-35212

Future CSS: Container queries (perhaps not quite ready for production sites)

Container queries are currently being worked on so I wouldn't completely rely on them yet, but if you're interested, check out the links below. Container queries are similar to media queries, but where media queries are based on the viewport size, container queries are based on the **container size**. This is extremely useful because sometimes you just want to tweak the display of a container's contents once it's below a certain width threshold. Container queries will allow you to make individual containers responsive depending on its contents.
At the moment, container queries are only an experimental feature in Chrome Canary, so if you'd like to try them out, you need to enable them in flags. Even if you don't try them out, the links below are interesting reads.

Good reads about container queries:

- https://www.smashingmagazine.com/2021/05/complete-guide-css-container-queries/
- https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Container_Queries