13. Responsive Design & SASS

Github repo

Responsive Design

- From Wikipedia: > Responsive web design (RWD) is an approach to web design that makes web pages render well on a variety of devices and window or screen sizes.

CSS Responsive Design Features

- CSS provides us with some tools to help make our designs more responsive
- They include:

Relative Units

- Instead of specifying element dimensions using fixed units (eg. pixels), we can use relative units to help things scale appropriately for various display sizes

Percentage

- Width, height, font-size, and a variety of other dimensions can be specified as a percentage
- Bear in mind that the percentage is based on the dimensions of the parent element, not the webpage itself
- eg. If the parent is `300px` wide and the child has a width of `50%`, then the child will be `150px` wide

`vh` and `vw`

- One `vh` is equal to `1%` of the viewport height
- An element with a style of `height: 50vh;` will be 50% the height of the screen
- `vw` works the same way except it's `1%` of the viewport width

`em` and `rem`

- An `em` is a relative measure based on the font-size of the parent component
- eg. If the parent has a font-size of `24px` and the child is `3em` wide, then it will be `72px` wide
- A `rem` is a **root** *em*, instead of being based on the parent's font-size, it is based on the font-size of the root element (html)

```
/* pixels */
p.pixel {
  width: 200px;
  height: 400px;
}

/* vh and vw */
p.viewport {
  width: 25vw;
  height: 50vh;
  font-size: 10vh;
}

/* em and rem */
p.relative {
  width: 25em;
  height: 40rem;
  border-width: 2em;
}
```

`max-width && min-width`

- `max-width` and `min-width` are used to set a maximum and minimum width respectively
- The element will not grow beyond the `max-width` nor shrink below the `min-width`
- Useful for making sure that your responsive elements don't grow or shrink to a point where they break the layout

Viewport Meta Tag

- We can add `meta` tags to the `head` element of our html
- In order to make sure that the user's browser displays our page correctly, we want to target the `viewport` meta tag

```
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
```

- The `content` portion is made up of two key/value pairs: `width` and `initial-scale`
- `width=device-width` tells the browser to set the width of the page to the width of the device
- `intial-scale=1.0` sets the initial zoom level of the page to `1.0` (or 100%)

Media Queries

- Media queries allow us to make changes to our design based on the user's device
- There are two parts to a media query: a **media type** and a **media feature**
- The options for *media types* are `screen`, `print`, `speech`, and `all`
- *Media features* include things like `aspect-ratio`, `device-height`, and `orientation`
- We can use multiple media queries to target various device sizes and orientations

```
@media only screen and (max-width: 500px) {
  /* these styles will be applied if the screen width is less than 500px
*/
  body {
    background-color: lightblue;
  }
}
```

CSS Preprocessors

- A CSS preprocessor generates CSS using a Domain Specific Language
- Styles are written in this *language* and then transpiled into CSS before being served to the client
- Popular preprocessors include Sass, LESS, Stylus, and PostCSS

Intro to Sass

- **S**yntactically **A**wesome **S**yle **S**heets
- Sass gives us some useful features to make writing our CSS easier
- **SCSS** or *Sassy CSS* is a superset of CSS
- A superset is a language that extends another language by adding new features
- But the browser doesn't understand SCSS, so we have to transpile our SCSS into CSS before serving it

Variables

- Sass utilizes variables like any other programming language: store a value and retrieve it later using the variables name

```
// variables
$font-color: lightblue;
$font-size: 1.2rem;

p {
  color: $font-color;
}
h1 {
  font-size: $font-size;
}
```

Nesting

- Nesting styles inside one another can help improve the readability and logical flow of our code

```
// basic css
.container p {
  color: magenta;
  text-decoration: underline;
}
.container div {
  border: 1px solid black;
}

// using nesting
.container {
  p {
    color: magenta;
    text-decoration: underline;
  }
  div {
    border: 1px solid black;
  }
}
```

Partials and `@import`

- We can use partials to store small amounts of code
- The convention for naming partials is to prepend the filename with an underscore (eg. `_variables.scss` or `_nav.scss`)
- Partials can be included into other Sass files using the `@import` syntax
- When importing, the leading underscore can be omitted from the filename

```
// inside _variables.scss
$border-width: 2px;
$border-color: red;

// inside styles.scss
@import 'variables';
p {
  border: $border-width solid $border-color;
}
```

`@extend`

- When you have two or more elements that have very similar styles, you could style one and use it as the basis for the other element(s)
- Styles can be combined into other styles using `@extend`

```
.header-text {
  font-size: 2em;
  font-family: 'sans-serif';
}

.heading {
  @extend .header-text;
  color: rebeccapurple;
}
```

Mixins

- A **mixin** is like a function that returns a group of styles
- The *mixin* can be included in any other style by using `@include`

```
// declare the mixin
@mixin header-styles {
  height: 50px;
  background-color: $header-bg;
}

// include it in another style
header {
  @include header-styles();
}

// mixins can take parameters as well
@mixin box-sizes($n) {
  height: $n;
  width: $n;
  line-height: $n;
}

.box {
  @include box-sizes(15px);
  border: 1px solid green;
}
```

Useful Links

- MDN: CSS Preprocessor
- What is the viewport?
- W3 Schools: Meta Tags
- MDN: CSS Values and Units
- Sass CLI

View on Compass | Leave Feedback