Handout

Responsive Web Development
Lesson 5 - Step 2
Guilherme Muller

With the main structure done and files created, I'd like to show you some very simple functions and mixins that can help us better develop the CSS of the project. If you are not into programming, don't worry. Here we'll work mainly with CSS, but with a different wrapping.

Sass has three structures that may seem similar, but have different use cases: **extends**, **mixins**, and **functions**.

Extends are used to join selectors that have common code. Sass will group everything together. And while that may seem like a good idea at first, at the end of the day it poses some problems. In the compiled file, it will join in unrelated selectors that can also break other parts of the code that are less specific.

A **mixin** is a block of code that's been included in a Sass file, like a common piece of code that you don't want to type over and over again. Keep in mind that every time you use a mixin, Sass outputs this code in that part of the file, that is, the code will be duplicated. While some may think that the final CSS file will be bloated because of the duplications, if used carefully, mixins are more reliable than joining unrelated selectors through extends. Another advantage of mixins is that they can take arguments. That means that a mixin can receive an input and change the block of code into an output, based on the argument supplied.

Finally, **functions** were later introduced in Sass, as they supply the important functionality not covered by mixins. Functions are great for returning values based on the arguments provided.

Let's see how all that works in practice. Begin by editing the mixin's general file. I find it useful to have functions that generate `em` or `rem` units based on pixel values. Create an function that has two arguments, `pixels` and `ref` for reference:

```
//
//-- General mixins and functions
//

@function em($pixels, $ref: 16) {
    @return ($pixels / $ref) * 1em;
}

@function rem($pixels, $ref: 16) {
    @return ($pixels / $ref) * 1rem;
}
```

The `:16` means that 16 is the default value for the argument, so we don't need to specify it when calling the function, unless we want to change it. The `@return` directive returns the value we are looking for, dividing the pixels value by the reference and multiplying it by `1em`, so that the result already comes with that unit.

Now let's create some mixins that will help us to configure the responsiveness of our page. These mixins will be general media queries. In the *_config.scss* file:

```
//
//-- Config
//

// colors
$color-base: #15263e;
$color-base-light: #374362;
$color-gradient1: #b119ec;
$color-gradient2: #f8027e;
$color-highlight: #a91aef;

// widths
$smallest-width: 20em;
$very-small-width: 30em;
$small-width: 48em;
$medium-width: 60em;
$large-width: 75em;
$very-large-width: 100em;

$site-width: 75em;
```

Now tweak *mixins/_mixins_breakpoints.scss*:

```
//
//-- RWD mixins
//

@mixin phone-land {
    @media (min-width: $very-small-width) {
        @content;
    }
}

@mixin tablet-port {
    @media (min-width: $small-width) {
        @content;
    }
}

@mixin tablet-land {
    @media (min-width: $medium-width) {
        @content;
    }
}

@mixin wide {
    @media (min-width: $large-width) {
        @content;
    }
}

@mixin very-wide {
    @media (min-width: $very-large-width) {
        @content;
    }
}

@mixin retina {
    @media
        only screen and (-webkit-min-device-pixel-ratio: 2),
        only screen and (min--moz-device-pixel-ratio: 2),
        only screen and (-o-min-device-pixel-ratio: 2/1),
        only screen and (min-device-pixel-ratio: 2),
        only screen and (min-resolution: 192dpi),
        only screen and (min-resolution: 2dppx) {
            @content;
    }
}
```

This is an example of media query nest in Sass. We insert the media query or mixin, that's generate a media query in this case, directly inside the element. Sass will generate the final CSS with the media query right after the element. When developing, this approach is much easier to maintain because all responsive styles are bundled within the element.

As for the example itself, we are using a mobile first approach. When there's no media query applied, that is when the device has a very small screen, the background will be red. When it reaches the 30em mark or 480 pixels, the background changes to green. The content inside the mixin substitutes the @content directive in the definition of the mixin.

So, we have set up general functions and responsive design mixins that will save us a lot of time later in the development phase.