Custom Web Part I: Responsive in a less SASSy Way

People love using the Bootstrap framework because it makes responsive web design fast and easy. As a developer, it is important to be able to create your own frameworks to improve the development experience and speed up the process in the future. For this lab, you are going create a CSS file that serves a similar function as Bootstrap's CSS along with a documentation site on how to use it. This project should not have any JavaScript in it (notice this is only Part I). You will be using a CSS pre-processor called Sass, just like Bootstrap does, to make this more fun. No other libraries or frameworks should be used.

Requirements:

- Must be built to handle responsive websites (Read these two articles and apply the concepts as they do it or in your own way: https://developers.google.com/web/fundamentals/getting-started/your-first-multi-screen-site/responsive?hl=en and http://learn.shayhowe.com/advanced-html-css/responsive-web-design/)
- Use a CSS Reset. I recommend the Meyer Reset.
- Watch the appropriate section from the LinkedIn Learning course titled - Sass Essential Training. https://www.linkedin.com/learning/sass-essential-training/welcome.
- Using Sass, build every aspect of your CSS to make life easier for the developer
- colors should all be defined as constant variables
- have one to three 'theme colors' that all other colors are based on using Sass functions or operations
- set customized default styles for standard HTML elements like headings (1–6), fonts, links, tables, forms, code, buttons, images, headers, footers, navs, etc.
- create several custom classes like Bootstrap has with it's wells, alerts, pagination, etc.
- keep the styling consistent
- always consider good padding and margins for elements
- add in several CSS transitions/transforms that are made easy to modify with Sass (Transitions w3schools, Transitions on LinkedIn Learning, Transforms w3Schools)
- use mixins to make support for webkit and mozilla easy
- create default values whenever possible but give the ability to make changes easy as well
- using `@use`, split your Sass into multiple easy to manage files, like: colors, fonts, ui-elements, main-styles, css-reset, etc.
- only have one, final CSS file even though you have several Sass files. Its ok to have more than one CSS file if it makes sense. (e.g. one CSS for all element styles, one for layout styles)
- Create a documentation website that explains how to use every aspect of your framework
- this site should actually be using your framework
- code examples should be included to show how to implement features (fix any special characters that don't show by default in html)
- visual output for each code example should be included
- should be multiple pages to make it user-friendly with solid design principles, keeping in mind that more content is coming (again, this is only Part I)
- Ultimately, you are building a framework were someone can construct a responsive website without really needing to do anything to the CSS other than possibly altering a few color variables or other constants in a Less or Sass file.

Submission

One zip file containing the html pages at the top level and three directories for other files: images, sass, css

Rubric: Custom Web 1

- Responsive design
  - Must be built to handle responsive websites. Should scale with different page sizes
- Uses CSS Reset
  - Should use Meyer CSS Reset. Either imported in html page, or as a SCSS file.
- Uses Sass
  - Should use Sass and compile down to a CSS file. More then one is allowed as long as it makes sense.
- 1-3 theme colors as constants
  - Site should use 1-3 theme colors that everything is based on and be defined as constant variables in the SCSS.
- Set default style for many of HTML elements
  - Set customized default styles for standard HTML elements like headings 1–6, fonts, links, tables, forms, code, buttons, images, headers, footers, navs, etc.
- Custom element classes
  - Create several custom classes like Bootstrap wells, alerts, pagination, etc.
- 2 or more CSS transitions/trasnforms that can be modified through Sass
- Include at least one mixin to handle vendor prefixes
  - Vendor prefix example is `-moz` and `-webkit`.
- Use at least 2 shared/extended properties
  - Shared properties use the `%variable-name` then `@extend` to use it.
- Sass is split into multiple files
  - Sass should be split into multiple files. Separation of concerns just like in programming. Then imported using the `@use` keyword to the main SCSS file.
- Create a Document Website
  - Create a document website to demonstrate your framework. It should include several pages defining each element you modified, how to use it with code examples and real examples similar to bootstraps documentation.
- Document Website should use your framework.
  - Each page of your site should import your compiled CSS file and utilize your framework.
- Documentation website code examples
  - Pages should show how to use your elements through code blocks
- Documentation website Visual Examples
  - Pages should include visual live examples of the elements to show what they look like when used.