CSS & SASS

Advance

**CSS & SASS** (Three Pillars to write good HTML and CSS)

* **Responsive web-design**
  + Building one website that works beautiful on all screen-size & devices
  + What makes responsive web-design?

A. Fluid layouts

B. Media queries

C. Responsive images

D. Correct units

E. Desktop-first or mobile-first approach

* **Maintainable & scalable code**
  + Writing clean, easy to understand & supports future growth
  + Most importantly making sure codes ***Reusable***
  + How to write maintainable & scalable code?

A. Organized files

B. Properly named classes

C. Well structured HTML

* **Web performance** 
  + Improving web performance means to make it faster; reduce its size so the user has to download less data
  + Way we can improve performance?

A. Less HTTP request & code

B. Compress code

C. Use preprocessor like ***SCSS***

* + Biggest impact to improve performance is to use ***less images*** and make sure images are compressed

**CSS behind the scenes**

* **What happens when page loads?**
  + First the page loads and the browser parse HTML which means it reads the code line by line
  + In the Parse phase the browser build the Document Object Model also known as the ***DOM***
  + When the browser create the ***DOM*** it describes the HTML like a family tree with ***parents***, ***children*** & ***sibling*** elements
  + When the HTML is getting parse the CSS also gets parsed but the parse of css is bit more complexed
* **How is CSS Parsed?**
  + First the cascade process resolves conflicting CSS declarations
  + The way is does that by ***importance***, ***specificity*** & ***source order***
  + The order of ***importance?***

1. User marked !important declaration

2. Author marked !important declaration

3. Regular Author declarations

4. Regular User declarations

5. Default browser declarations

* + The order of ***specificity?***

1. Inline styles

2. IDs

3. Classes, pseudo-classes & attributes

4. Elements, pseudo-elements

* + The ***source order*** gets the last declaration in the code and will override all the other declarations and will apply it
* **How are CSS values processed?**
  + When CSS processed it returns values if applicable every CSS property must have a value.

1. Declared value (author declared)

2. Cascaded value (after the cascade)

3. Specified value (default, if there’s not cascade value)

4. Computed value (converting relative values to absolute)

5. Used value (final calculations, based on layout)

6. Actual value (browser and device restrictions)

* + Units are converted from relative to absolute in the ***computed*** state
  + Fonts ***percentages*** compute the parents size
  + Lengths expressed in ***percentages*** the reference is always the parent elements width
  + ***EM*** & ***REM*** are font based units
  + ***EM*** uses the parents for reference
  + ***REM*** uses the root font size as the reference
  + When ***EM*** is used for ***length*** is uses the current element for reference instead of the parent
  + ***VH*** & ***VW*** are are simply percentage measurement of the viewports height and width
* **Inheritance in CSS!** 
  + CSS passes properties from parent element to child
  + There are some properties that can be inherited but some that can not
  + If property is inheritable the ***computed*** value gets passed
  + Properties related to ***text*** are inherited
  + Using the ***inherit*** keyword to force inheritance
  + The ***initial*** keyword used to reset to initial value
* **The Visual Formatting Model**
  + When a website is rendering a algorithm called ***Visual formatting model*** run that calculates boxes and determines the layout of these boxes
  + It’s done for each render tree in order to determine the final layout of the page
  + Things that the algorithm takes into account

1. Dimensions of the boxes (the box model)

2. Box type (inline, block & inline-block)

3. Positioning scheme: floats and positioning

4. Stacking contexts

5. Other elements in the render tree

6. Viewport size, dimensions of images, etc

* + The ***Box Model*** is one of the factors that define how our elements are going to display on webpage
  + When ***box-sizing*** is set to ***border-box*** when padding or margin are added to box it will not get added to its dimensions
  + What are the different type of box types?

1. ***Block*** 100% of parents width & stack vertically

2. ***Inline*** have no height or width occupies only contents space. Padding & and margins are only horizontal.

3. ***Inline-block*** Occupies only content’s space, no line-breaks and box-model applies.

* **CSS Architecture, Components & BEM**
  + When thinking about the design we should use ***component-driven design***
  + Building with meaningful class names, a good structor for naming classes is Block Element modifier ***BEM***
  + ***BEM*** is a good design pattern for maintainable and scaleable code
  + ***BEM*** has a ***low-specificity***, it’s better to try to ***never nest*** in ***SASS*** file
  + What is ***BEM*** structor ?

1. ***Block***, standalone component that can be reused anywhere in the project, Blocks can be nested ***(.block {} )***

2. ***Element***, is part of block that has no meaning on its own, if we take one of these elements of the block they wouldn’t be useful at all ***(.block\_\_element {} )***

3. ***Modifier***, a different version of a block or an element it lets you modify a already created block or element ***(.block\_\_element-modifier {} )***

* + ***Architecting*** using the 7-1 pattern. Which is using 7 different folders for partial ***SASS*** files, and importing all into one main ***SASS*** file.
  + The 7 folder of the The 7-1 Pattern

1. ***Base/***, used to place the basic project definitions

2. ***components/***, where we have 1 file for each component

3. ***Layout/***, defines the overall layout of the project

4. ***Pages/***, styles for specific pages of the project

5. ***Themes/***, used to implement different visual themes

6. ***Abstracts/***, code that doesn’t output any CSS such as variable or mix-ins

7.***vendor/***, where all third-party CSS ex.Bootstrap

**SCSS the CSS preprocessor**

* **What is SASS?**
  + ***SASS*** code that get converted to CSS, an extension of CSS that adds power and elegance to the basic language
  + Main ***SASS*** Features?

1. ***Variable***, for reusable values such as colors, font, spacing, etc

2. ***Nesting***, to next selectors inside of one another, allowing us to write less code

3. ***Operators*** for mathematical operations right inside of CSS

4. ***Partials and imports***, write CSS in different files & importing them all into one single file

5. ***Mixins***, reusable piece of CSS codes

6. ***Functions***, similar to Mixins with the difference that they produce a value that can be used.

7. ***Extends***, make different selectors inherit declarations that are common to all them

* + There are 2 ***SASS*** syntax ***SASS*** syntax & ***SCSS*** syntax SCSS is better to use

**Advanced Responsive Design**

* **Responsive design strategies**
  + There are 2 types of approach for designing your responsive layout ***Desktop-First*** & ***Mobile-First a***pproach
  + What basic of ***desktop-first***?

1. Start writing CSS for desktop large screen

2. Then use Media Queries to shrink design to smaller screen

3. This ***a***pproach uses ***max-width***

* + What basic of ***Mobile-first***?

1. Start writing CSS for mobile devices

2. Then use Media Queries to expand design to larger desktop screens using ***min-width***

3.Forces us to reduce website and app to absolute essentials

* **Breakpoints**
  + Max-width used for desktop-first what it does is checks if the current viewport is smaller or equal to the given value
  + Example if Max-width:600px means the width is <= 600px
  + Which means if the width is greater then or equal to 600px then all the css in that media will apply
  + So in this situation everything after 600px will not apply anymore
  + When we are using desktop first approach we overwrite different viewport using max-width and overwrite those properties
  + If there are 2 max-width queries one of 600px other 900px & our current viewport is 500px which means both condition are true then both queries will applied
  + If there are conflicting CSS rules in those 2 queries then the one that is last gets applied for this example it would 900px since 600px was applied first
  + A good way to select media queries is to group most used device is logical way then select the most common breakpoint <http://gs.statcounter.com/screen-resolution-stats>
* **Responsive Images**
  + Responsive images are important for design but as-well as performance
  + The goal of responsive images is to serve the ***right image*** to the ***right screen size*** and device, in order to avoid downloading unnecessary large images on smaller screens.
  + Different way to make responsive images

1.***Resolution*** ***switching,*** decreasing image resolution on smaller screen

2.***Density switch,*** Half the image resolution on @1x screen

3.***Art direction***, different images on smaller screens

**Flexbox Layouts**

* **What’s is flexbox?**
  + Flexbox makes it easy to align element to one another, in different direction and order.
  + Main idea behind flexbox is to give the container the ability to expand & to shrink elements to best use all available space.
  + Flexbox work its ***one-dimensional*** layouts
* **Main Flexbox concepts?** 
  + Flexbox makes it easy to align element to one another, in different direction and order.
  + When an element is given display ***flex property*** it becomes a ***flex container***
  + All the direct children of ***flex container*** are called ***flex items.***
  + The direction the items in the flex container are laid out are called the ***main axis*** & ***cross axis***
  + Properties of ***flex container***?

1. ***Flex-direction***, defines the direction of the container

2. ***Flex-wrap***, defines whether or not ***flex-items*** should wrap if there isn’t enough space.

3. ***Justify-content***, defines how the ***flex-items*** will be defined along the ***main axis***

4. ***Align-items***, defines how the ***flex-items*** will be defined along the ***cross axis***

*5.* ***Align-content,*** controls how the rows are aligned on the ***cross axis***

* + Properties of ***flex item***?

1. ***align-self***, just like align items but applies to single item

2. ***order***, defines the order of the item

3. ***flex-row***, defines how much an item can grow

4. ***Flex-shrink***, defines how much an item can shrink

5. ***Flex-basis***, defines it base width

6. ***Flex***, defines ***flex-row, Flex-shrink & Flex-basis***

***CSS-GRID Layouts***

* ***What’s is flexbox?***