1. What happens in the background.
2. Three pillars of writing good html and css.  
   Responsive design :   
   Writing maintainable and scalable code :   
   Web Performance :
3. Responsive web design:   
   one website that works beautifully on all screen sizes and all devices.

Here we need to think about responsive images, using correct units for font-sizes or element dimensions, even includes a desktop first or mobile first strategy.

1. Writing maintainable and scalable code:  
   clean, easy to understand, supports future growth and most importantly re-usable.  
   This also includes how to organize files, how to name classes, how to structure html.
2. Web Performance: less code , compress code, less http requests, use css preprocessor,mostly reduces the use of images by using the images which are really necessary for a website + compressing those images so that they consume less bandwidth for the user.
3. HOW CSS WORKS BEHIND THE SCENES.
4. What happens to our css code when we load a webpage.  
   - browser loads html  
   - browser parses html + loads the stylesheets and parses it which is a bit different than  
    how html is parsed.  
   - browser build the DOM like a family tree.
5. Two main steps in CSS parsing phase :   
   8.a : Conflicting css declarations are resolved in a process called cascade.  
   8.b : process final css values like converting a margin defined in percentage units to pixel   
    units.

After all of this is done final css is stored in a tree like structure called the CSS object model.

1. After the html and css are parsed and stored : these two together form the render tree.
2. Then the page is rendered. The browser uses the visual formatting model to render the page.  
     
    ---------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*---------
3. CSS parsing phase.  
   step 1: cascade : it is the process of combining different stylesheets and resolving conflicts between different css rules and declarations – when more than one rule applies to a css element.  
   CSS can come from different sources.  
   **Developer written CSS**: they are called author declarations.

**User declarations**: CSS coming from the user i.e in the browser.  
**Default browser declarations**: user agent CSS as it is set by the browser.

1. But how does CSS resolve conflicts.

It looks at the IMPORTANCE > *selector* SPECIFICITY > SOURCE ORDER of conflicting declarations in order to determine which one takes precedence.

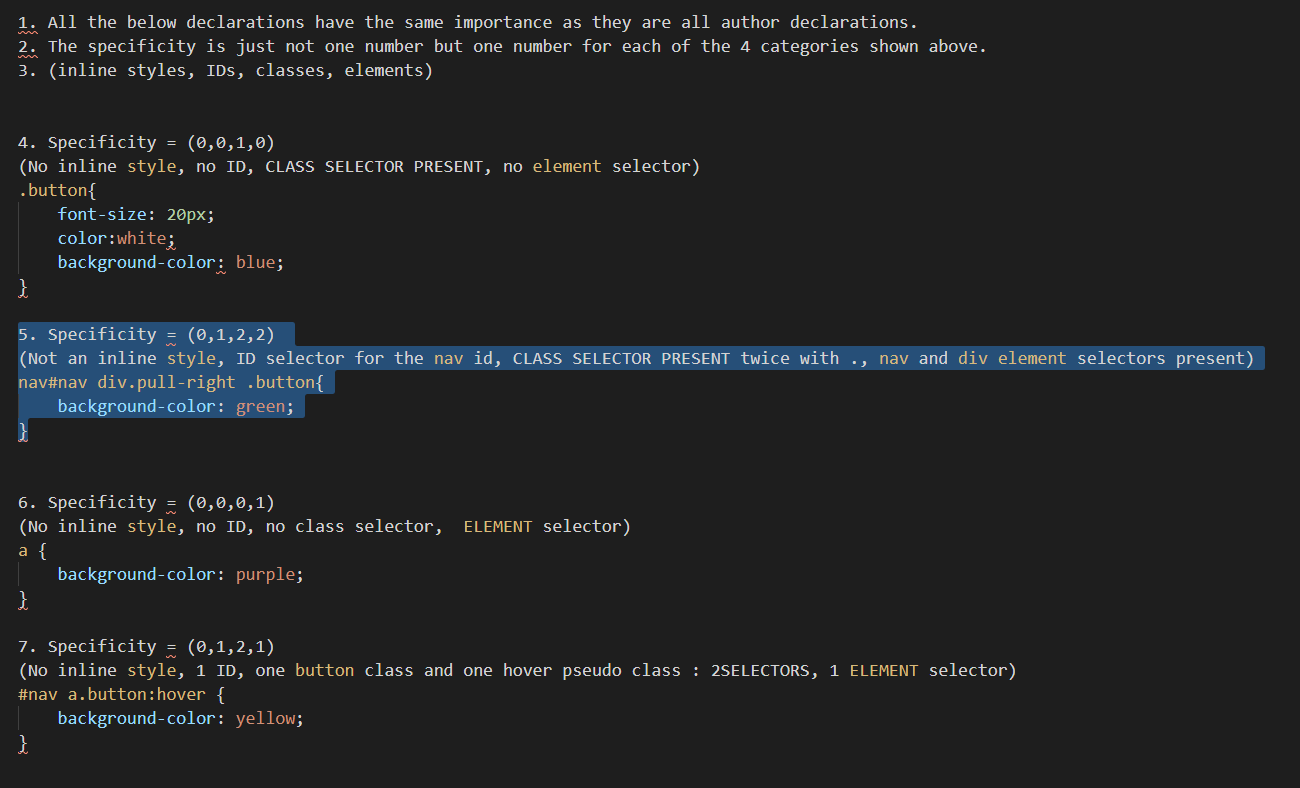
1. First the cascade starts by giving conflicting declarations different **importance** based on where they are declared / based on their source. The order is below

* Most important declarations are the user declarations marked with !important keyword.
* Author declarations marked with an !important keyword.
* Author declarations
* User declarations
* Browser declarations.

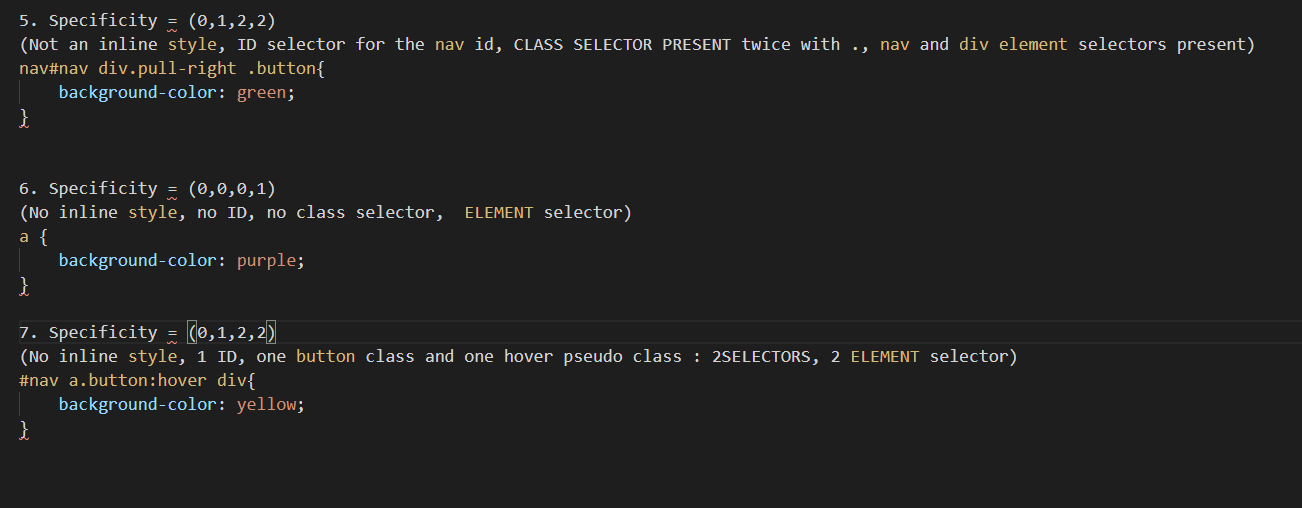
1. When we have conflicting declarations with the same importance we calculate the selector specificity based on the priorities shown below.

Now cascade calculates and compares the specificities of the declaration selectors.

* The specificity order is below.
* Inline styles : they have the highest specificity.
* IDs
* Classes , Pseudo classes, attribute selectors
* Elements, pseudo elements.

Example : How to calculate specificities.  


* Selector number 2 is the most specific selector of all.

1. Now let us say of if point 5 and 7 have the same specificity like below ..i.e if all the declarations have the same specificity.. then the last css declaration written in the code is the one that will apply.( SOURCE ORDER)  
     
   Universal Selector has zero specificity (0,0,0,0)  
   Put your stylesheets in the last of all the included stylesheets whichmay come from 3rd parties.   
    ---------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*--------
2. How values are processed in the parsing phase + how units like em,rem,%,pixels are calculated.
3. How declared values are processed in six different steps starting from the declared values to the final actual value.  
   Lets analyze the ‘width’ ‘padding’ and ‘font-size’.
4. Lets begin with width-paragraph.  
   Declared Value : 140px via ‘p’ element selector  
    : 66% via ‘amazing’ class selector  
   Cascaded Value : As the class selector has more specificity 66% is picked.

Specified Value : It is the default value of a certain css property.

In this case its ir-relevant as we have a cascaded value already.

Computed value : In this step values with relative units are converted to pixels, So  
 that they can be inherited.  
 In this case we have a percentage value which is technically not a  
 unit.So nothing happens in this step.  
Used value : In this step the css engine uses the rendered layout to figure out some   
 of the remaining values.Ex percentage values that depend on the   
 layout. The 66% we specified is in relation to its parent element.  
 So the parser needs to know that width, in order to calculate the  
 paragraph width.  
 The parent element i.e ‘section’s ’ width is defined as 280px.

(66\*280)/100 = 184.8px – this is now our used value.

Actual Value : Browsers can not usually display 184.8 pixels and so they are rounded  
 of to another value 185pixels.This value will be used in the layout.

1. Padding property for the paragraph.  
   There is no declaration for the padding property for the paragraph .  
   Each and every CSS property should have a value even though it is not declared.And thus no cascade value.  
   Declared Value : X  
   Cascaded Value : X  
   Specified Value : Each CSS property has a initial value, which is used when there is no  
    cascaded value.i.e if we/browser/user don’t declare a value then   
    the initial value is used.[inheritance should be taken into   
    account].The initial value for padding is 0 pixels.  
   Computed + Used + Actual Value : There is not more calculation to do as its already   
    an absolute unit.  
     
   
2. Now the font size property of the root element.  
   Declared Value : We have not defined it anywhere. So no declared value.  
   Cascaded value : The browser has a default value of 16 pixels.We know that CSS   
    could come from different sources.In this case the font-size is a   
    user agent declaration.  
   Specified Value + further : no more calculations needed and 16 pixels is used.
3. Font-size of an element having section class.

Declared-value : 1.5 rem which is a relative unit.

Cascaded-value : 1.5rem  
Specified-value : 1.5 rem

Computed Value : 24pixel  
 1.5 \* 16pixels.  
 rem unit is always relative to the root font-size.  
 Used + Actual Value : 24 pixels.

1. Font size of the paragraph element.

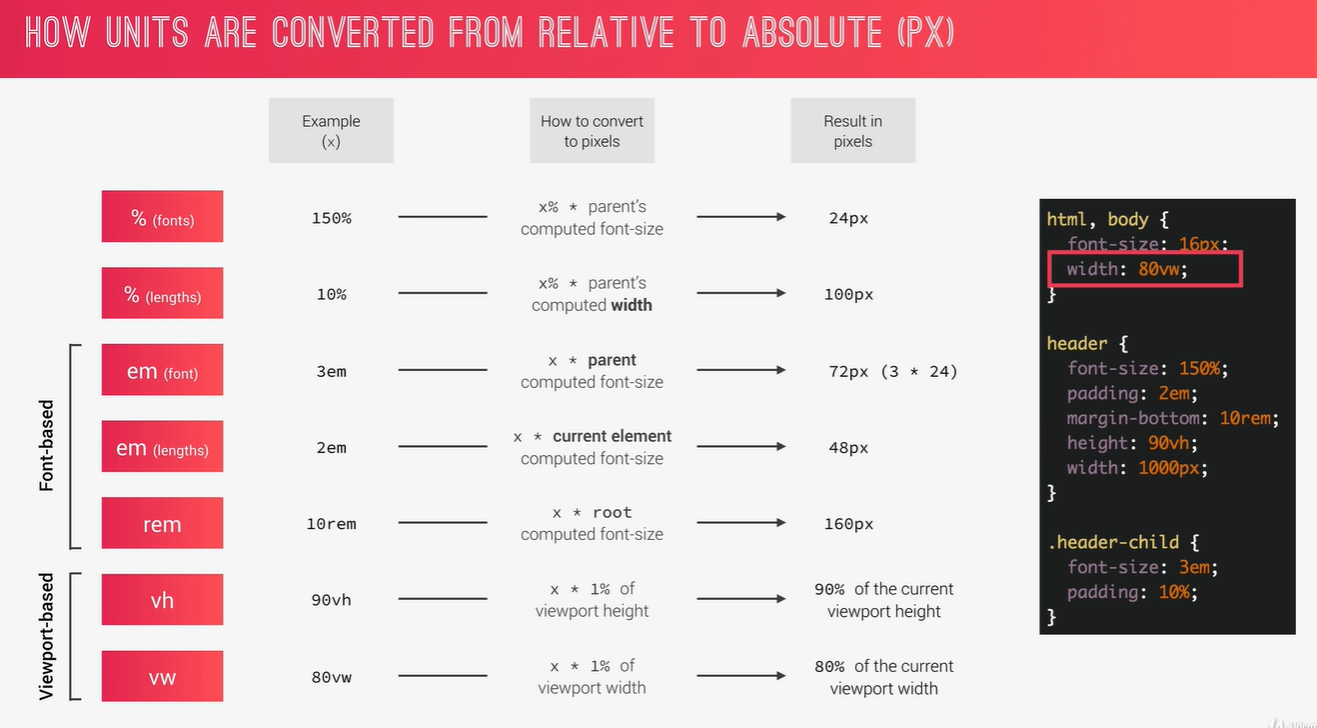
Declared Value : X.  
Cascaded Value : X

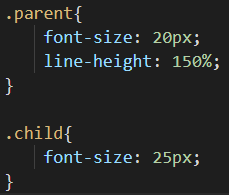
Computed Value : Some properties like the ones related to text such as the font-size inherit the computed value from their parent elements.24 pixels.

1. How does the CSS engine convert relative units to pixels to calculate computed and used values.

Relative units are most important for building responsive websites.

How different units work in different situations.

* First lets start with percentages.
* There is a difference in using percentages for fonts and lengths/distance measurements.
* %+font : Now the header has a font-size of 150%. This means that the header will have a font-size 150% larger than its parent element which is the body element with a font-size of 16.
* %+Length : When we express a length measurement in percentages, like height,padding,margin – the reference is always the parent elements width. For Example consider the element which has this class .header-child, it has a padding of 10%.As its parent element has 1000px of width, 10% of 1000px is 100px.Remeber that the parents width is taken as a reference.
* Next are font based relative units em,rem. Its different to use em for fonts and for lengths.  
  Both ems and rems are font based.
* Em – uses the parent element or the current element as the reference.
* Rem – uses the root font size as the reference.
* For example (font) : There is a 3em font size in the .header-child element. This is 3 times the parents font-size.3\*150%. 3 \* ((16\*150)/100). 3\*24. i.e 72 pixels.
* For example (length like padding) : For length it’s a bit different : There is a 2em padding in the header. Since it’s a length measurement, it uses the font-size of the current element as a reference, which is 24 pixels. 2(24px). 48px pading.
* When measuring in terms of em :   
   For fonts the reference is the parent.  
   For lengths the reference is the current element.  
    
    
  
* Next about the rem : It works the same way for both font-sizes and length. Because it always uses the root font size as reference. The .header has a margin-bottom of 10rem. 10\* root font size = 10\*16px = 160px
* Why should we size stuff with ems and rems, if they are based in font size : but using them we can build more robust responsive layouts. By changing font-sizes we will automatically change length, since they depend on the font size. It gives us a lot of flexibility and it is just a great technique.
* 2 more relative units: vh vw. These are based on the browsers viewport.
* Summary :   
  About percentages :   
   \* They are measured relative to their   
   parents font size – if used to specify font-sizes.  
   \* They are measured relative to their   
   parents width – if used to specify length.  
  About em:  
   \* ems are measured relative to their parents font-size,  
   if used to specify font sizes.  
   \* they are measured relative to their current font size,   
   if they are used to specify length.  
  About rems:  
   \* They are always measured relative to the documents root font size.  
     
   ---------\*\*\*\*\*\*\*\*\* INHERITANCE \*\*\*\*\*\*\*---------

1.   
   lets determine what the line-height of the child will be.  
   There is no declared and hence no cascaded value.  
   Can this property be inherited ? yes.  
   Then .child will inherit the computed value of 150%.  
   In this case its 150% of 20px = 30px.  
   SO the line-height of .child will be 30px.  
     
   Now if it’s a property that is not inherited like padding, then the **specif**ied value will become the initial value – which is also specific to each property, which is 0 pixels in the case of padding.
2. Properties related to text are inherited. Other properties like margin, padding are not inherited, obviously.
3. What gets inherited is the computed value of the property and not the declared value.
4. We can use the **inherit** keyword to force inheritance of a certain property.
5. We can use the initial keyword to reset the property to its initial value

\*\*\*\*\* Visual Formatting Model \*\*\*\*\*\*\*\*\*\*\*

1. Sd
2. dsfs

\*\*\*\*\* CSS architecture ,components and BEM\*\*\*\*\*\*\*\*\*\*\*

1. **THINK** about the layout of the web page before you write the code.

Component driven design.

1. **BUILD :** Block Element Modifier.Clean system for marking up our layouts.

**Block :** It is a stand alone component that is meaningful on its own.  
**Element** : An element is a part of a block and has no meaning on its own.

1. ARCHITECT : 7-1 pattern. Which means we put 7 different folders where we put partial sass files, and then 1 main sass file in which we import we import all of our partial files into 1 final compiled css style sheet.  
   The 7 folders are :   
   /base : basic product definitions.  
   /components : 1 file for each component.  
   /layout : Overall layout of the project is defined.

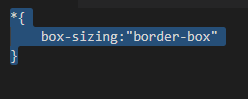
/pages : Where we have styles for specific pages of the project.

/themes : for different visual themes.

/abstracts : here we put code that doesn’t output any css – such as variables and mixins.

/vendors : all third party css goes.

\*\*\*\*\*CHEAT SHEET \*\*\*\*\*\*\*\*\*\*\*

1. Global reset  
      
     
   font-size:10px ? From this root font size font + length of other elements will be taken and we write those measurements in rem.  
     
   box-sizing:inherit ? A better practice by the community. This property is not inherited.   
   So instead of using below reset for all elements we ask each element to inherit this property from its parent.  
   
2. Global-reset to apply for the before and after pseudo elements also.  
   

\*\*\*\*\*\*SASS\*\*\*\*\*\*

1. What does sass offer us.

* Variables : This allows us to have re-usable values such as colors,font-sizes,spacing.
* Nesting : We can nest selectors inside of one another – allowing us to write less code.
* Operators : For mathematical operations inside of CSS.
* Partials and imports : to write css in different files and importing them to one single file.
* **Mixins** : to write re-usable pieces of CSS code.
* **Functions** : these are similar to mixins, with a difference that they produce a value that can be used later.
* **Extends** : to make different selectors inherit declarations that are common to all of them.
* Control Directives : write complex code using conditionals and loops.

1. Sass syntax

* Sass Syntax : Indentation sensitive. No curly brases and semi colons.
* Scss syntax : Sassy CSS. Same like writing css code.  
    
  **Mixins:** It is just a reusable piece of code that we write in a mixin.When we use that mixin, that code is put in the place where we used it. Something like a variable with huge lines of code.

How to install SASS

* npm install node-sass --save-dev
* npm script : "compile:sass":"node-sass sass/main.scss style.css -w"
* npm run compile:sass  
    
  How to automatically reload a browser
* npm install –g live-server
* And then run **live-server** in the package.json folder.  
    
  How to organize your files in SASS.
* We use the 7-1 pattern.
* We have 7 folders and 1 main sass file to import all the files that are in these folders.
* 1st Folder : /base/ : basic project definitions  
  Files under it :   
  \_base.scss : [This file is for the real low level basics such as ‘resets’ and styles for the HTML and BODY element selectors.]

\_animation.scss : For all animaions.

\_typography.scss :

\_utilities.scss :

* 2nd Folder : /abstract/ :Code that does not output any CSS. Like variables,mixins etc.  
  Files under it :   
  \_variables.scss :  
  \_mixins.scss :  
  \_functions.scss :
* 3rd Folder : /Components/ : 1 file for each of our components. These are re-usable building blocks.
* 4th folder : /layout/ : holds all the components together. For each piece of the global layout of the entire project.
* 5th folder : /pages/ : if we have very specific styles for a specific page for ex a home page, we can then create a new file for that specific page and we do that in the pages folder.
* 6th folder : /themes/ : in case we are doing a web app with different themes.
* 7th folder : /vendors/ : here we can put 3rd party CSS. For Ex : Css file of bootstrap / icon system / animation framework

BASIC PRINCIPLES OF RESPONSIVE DESIGN AND LAYOUT

* Fluid grids and layouts : use percentages rather than pixels for all layout related length and especially width : so that we can adapt correctly to the user viewport width.
* Flexible and responsive images: images should also scale accordingly to the viewport. Should mention their width in percentages. We should optimize the images for different widths.
* Media queries : allow us to change styles on certain viewport widths, which are called breakpoints.With this we can create different versions of websites for different devices.

LAYOUT TYPES

* Float based layouts : they are still used as the modern alternatives are not supported by all browsers
* Flex box : Perfect for laying out elements in a 1dimentional row.
* CSS Grid : is perfect for creating the overall layout of a page in a fully-fledged 2d grid.