**Importance Order**

1. **!important** – User declarations
2. **!important** – Author declarations
3. **Author** **Declarations**
4. **User** **Declarations**
5. **Default Browser Declarations**

**\*NOTE:** If there is the same importance, CSS will use selector specificity to determine the most importance.

**Specificity Order**

1. Inline Styles – written in the HTML file
2. IDs
3. Classes, Pseudo-Classes, Attribute
4. Elements, Pseudo-Elements

**Specificity calculations:**

.button { //

    font-size: 20px;

    color: white;

    background-color: blue;

}

Inline 0

IDs 0

Classes 1 .button

Elements 0

Specificity = 0010

nav#nav div.pull-right .button {

    background-color: green;

}

Inline       0

IDs     1 #nav

Classes 2 .pull-right and .button

Elements     2 nav and div

Specificity = 0122

a {

    background-color: purple;

}

Inline 0

IDs 0

Classes     0

Elements 1 a tag

Specificity = 0001

#nav a.button:hover {

    background-color: yellow;

}

Inline 0

IDs 1 #nav

Classes 2 .button and :hover(pseudo-element)

Elements 1 a tag

Specificity = 0121

**\*NOTE:** If there is the same specificity:

The last declaration in the code will override all other declarations and will be applied.

**Rules for Cascade and Specificity**

* CSS declarations marked with !important have the highest priority
* Only use !important as the last resource. It’s better to use correct specificities – it makes the code more maintainable.
* Inline styles will always have priority over styles in external stylesheets
* A selector that contains 1 ID is more specific than one with 1000 classes
* A selector that contains 1 class is more specific than one with 1000 elements
* The universal selector (\*) has no specificity value (0, 0, 0, 0)
* Rely more on specificity than on the order of selectors
* Rely on the order when using 3rd-party stylesheets – always put your author stylesheet last

**CSS Value Processing**

* Each property has an initial value, used if nothing is declared (and if there is no inheritance)
* Browsers specify a root font-size for each page (usually 16px)
* Percentages and relative values are always converted to pixels
* Percentages are measured relative to their parent’s font-size, if used to specify font-size
* Percentages are measure relative to their parent’s width, if used to specify lengths
* em are measured relative to their parent font-size, if used to specify font-size
* em are measure relative to the current font-size, if used to specify lengths
* rem are always measured relative to the document’s root font-size
* vh and vw are simply percentage measurements of the viewport’s height and width

**Inheritance**

* Inheritance passes the values for some specific properties from parents to children – more maintainable code
* Properties related to TEXT ARE INHERITED: font-family, font-size, color, etc…
* Properties related to padding or margins ARE NOT INHERITED
* The computed value of a property is what gets inherited, not the declared value
* Inheritance of a property only works if no one declares a value for that property
* The *inherit* keyword forces inheritance on a certain property
* The *initial* keyword resets a property to its initial value

**Box Types**

**Block-level boxes**

* Elements (paragraphs and divs) are set to block by default. We can change it.
* Occupies 100% of parent’s width
* Creates line breaks before and after
* Vertically, one after another

display: block

(display: flex)

(display: list-item)

(display:  table)

**Inline boxes**

* Content is distributed in lines
* Occupies only the content’s space
* There are no line-breaks
* There are no heights and widths meaning we cannot use these properties
* Paddings and margins only horizontal - left and right padding and margins only

display: inline

**Inline-block boxes**

* A mix of block and inline
* Occupies only the content’s space
* There are no line-breaks

display: inline-block

**Positioning Schemes**

**Normal Flow**

* Default positioning scheme
* NOT **floated**
* NOT **absolutely positioned**
* Elements laid out according to their source order

Default

position: relative

**Floats**

* Element is removed from the normal flow
* Text and inline elements will wrap around the floated element
* The container will not adjust its height to the element – use clearfix to resolve this issue

**\*Note:** When using float, all the height and width collapse. To fix this, use the clearfix method:

.clearfix::after {

    content: "";

    clear: both;

    display: table;

}

float: left

float: right

**Absolute Positioning**

* Element is removed from the normal flow
* No impact on surrounding content or elements
* We use top, bottom, left, and right to offset the element from its relatively positioned container

position: absolute

position: fixed

**Stacking Contexts**

* Determines the order in which elements are rendered on the webpage.
* These are layers where elements with a lower z-index is placed below elements with a higher z-index.

**Think – Build – Architect Mindset**

* Think about the layout of your webpage or web app before writing code.
* Build your layout in HTML and CSS with a consistent structure for naming classes.
* Create a logical architecture for hour CSS with files and folders.

**Think**

**Component-Driven Design**

* Modular building blocks that make up interfaces.
* Held together by the layout of the page
* Re-usable across a project and between different projects
* Independent, allowing us to use them anywhere on the page

**Build**

**Block Element Modifier (BEM)**

* **BLOCK** – standalone component that is meaningful on its own
* **ELEMENT** – part of a block that has no standalone meaning
* **MODIFIER** – a different version of a block or an element

.block {}

.block\_\_element {}

.block\_\_element--modifier {}

**Architect**

* Create a logical architecture for hour CSS with files and folders.
* The 7-1 Pattern – 7 different folders for partial Sass files, and 1 main Sass file to import all other files into a compiled CSS stylesheet.
* The 7 Folders

1. base/
2. components/
3. layout/
4. pages/
5. themes/
6. abstracts/
7. vendors/

**SASS**

* SASS is a CSS preprocessor, an extension of CSS that adds power and elegance to the basic language.
* We use SASS to fix problems with CSS.
* SASS Source Code 🡪 SASS Compiler 🡪 Compiled CSS Code

**Features of SASS**

* **Variables** – reusable values such as colors, font-sizes, spacing, etc…
* **Nesting** - nest selectors inside of one another, allowing us to write less code.
* **Operators** – mathematical operations right inside of CSS
* **Partials** and **Imports** – write CSS in different files and importing them all into one single file
* **Mixins** – write reusable pieces of CSS code (@mixin then use it with @include)
* **Functions** – like mixins, with the difference that they produce a value that can then be used
* **Extends** – make different selectors inherit declarations that are common to all of them
* **Control** **Directives** – writing complex code using conditionals and loops

**Installing SASS**

Install nodejs – <https://nodejs.org/en/>

node -v – check to see if you installed nodejs by checking the version in command line/terminal

Graphical user interface, text

Description automatically generated

Create a package.json file

Navigate to the project folder

Type npm init

Text

Description automatically generated

Now there should be a package.json package in your directory:

A screenshot of a computer

Description automatically generated with medium confidence

Install SASS package

npm install node-sass --save-dev

Text

Description automatically generated

Now check the package.json package to see if sass was installed:

  "devDependencies": {

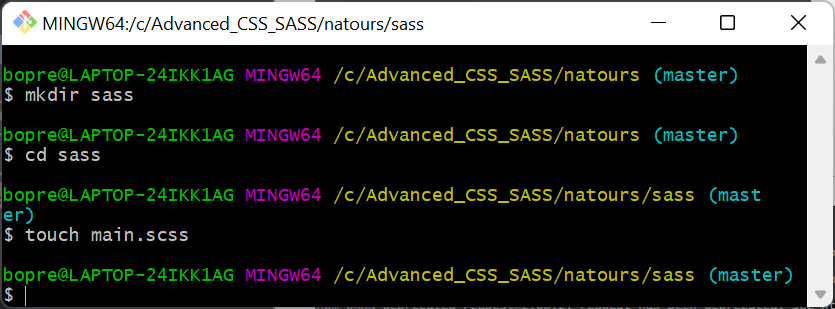
    "node-sass": "^7.0.3"

  }

Create a folder named sass

Change directory to sass

Create a main.scss file



Copy the entire contents of the CSS file into the main.scss file

Delete contents of style.css file now

To compile SASS file:

Go to package.json and add the line to the scripts:

  "scripts": {

    "compile:sass": "node-sass sass/main.scss css/style.css -w"

  },

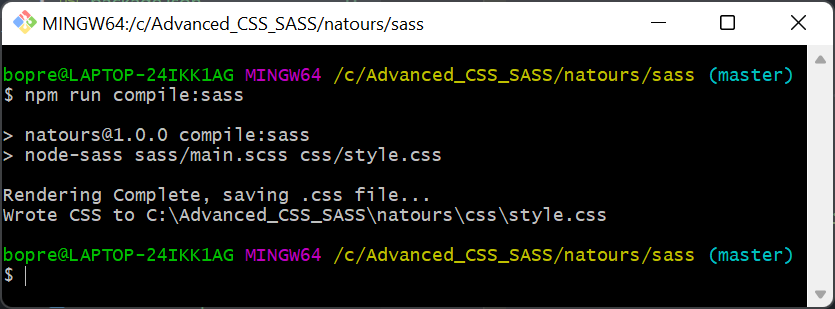
node-sass – means that we want to use sass

sass/main.scss – the sass file (input)

css/style.css – the css file (output)

-w – will automatically save the changes we made in our main.scss file to style.css file

In the terminal type the command: npm run compile:sass



Now the empty style.css file before will have the contents of the main.scss file however, the values will be converted to css in the style.css file.

\*Note: Look at the declared variable of primary-light and primary-dark in the style.css file.

style.css

rgba(40, 180, 133, 0.8)

main.scss

rgba($color-primary-light, 0.8)