**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