HTML5 & CSS3

Presentation by Uplatz

Contact Us: https://training.uplatz.com/
Email: info@uplatz.com
Phone:+44 7836 212635

Uplatz

Table Of Contents:

Cascading and Specificity:

Calculating Selector Specificity:

- Each individual CSS Selector has its own specificity value.
- Every selector in a sequence increases the sequence's overall specificity.
- Selectors fall into one of three different specificity groups: A, B and c.
- When multiple selector sequences select a given element, the browser uses the styles applied by the sequence with the highest overall specificity.

Group	Comprised of	Examples
A	id selectors	#foo
B	class selectors attribute selectors pseudo-classes	.bar

[title], [colspan="2"]

:hover, :nth-child(2)

c

type selectors **div,**

pseudo-elements **::before, ::first-letter**

- Group A is the most specific, followed by Group B, then finally Group c.
- The universal selector (*) and combinators (like > and ~) have no specificity.

The !important declaration

- The !important declaration is used to override the usual specificity in a style sheet by giving a higher priority to a rule.
- Its usage is: property : value !important;

```
#mydiv {
 font-weight: bold !important; /* This property won't
be overridden
 by the rule below */
}
#outerdiv #mydiv {
 font-weight: normal; /* #mydiv font-weight won't be
set to normal
 even if it has a higher specificity because
 of the !important declaration above */
}
```

- Avoiding the usage of !important is strongly recommended (unless absolutely necessary), because it will disturb the natural flow of css rules which can bring uncertainty in your style sheet.

- Also it is important to note that when
multiple !important declarations are applied to the same rule on a certain element, the one with the higher specificity will be the on a applied.
- Here are some examples where using !important declaration can be justified:
- If your rules shouldn't be overridden by any inline style of the element which is written inside style attribute of the html element.
- To give the user more control over the web accessibility, like increasing or decreasing size of the font-size, by overriding the author style using !important.
- For testing and debugging using inspect element.

Colors

currentColor:

- currentColor returns the computed color value of the current element.

Use in same element:

Here currentColor evaluates to red since the color property is set to red:

```
div {
 color: red;
 border: 5px solid currentColor;
 box-shadow: 0 0 5px currentColor;
}
```

- In this case, specifying currentColor for the border is most likely redundant because omitting it should produceidentical results.

- Only use currentColor inside the border property within the same element if it would be overwritten otherwise due to a more specific selector.
- Since it's the computed color, the border will be green in the following example due to the second rule overriding the first:

```
div {
 color: blue;
 border: 3px solid currentColor;
 color: green;
}
```

Inherited from parent element

- The parent's color is inherited, here currentColor evaluates to 'blue', making the child element's border-color blue.

```
. parent-class {
 color: blue;
}
.parent-class .child-class {
 border-color: currentColor;
}
```

- currentColor can also be used by other rules which normally would not inherit from the color property, such as background-color.
- The example below shows the children using the color set in the parent as its background:

```
.parent-class {
 color: blue;
}
.
```

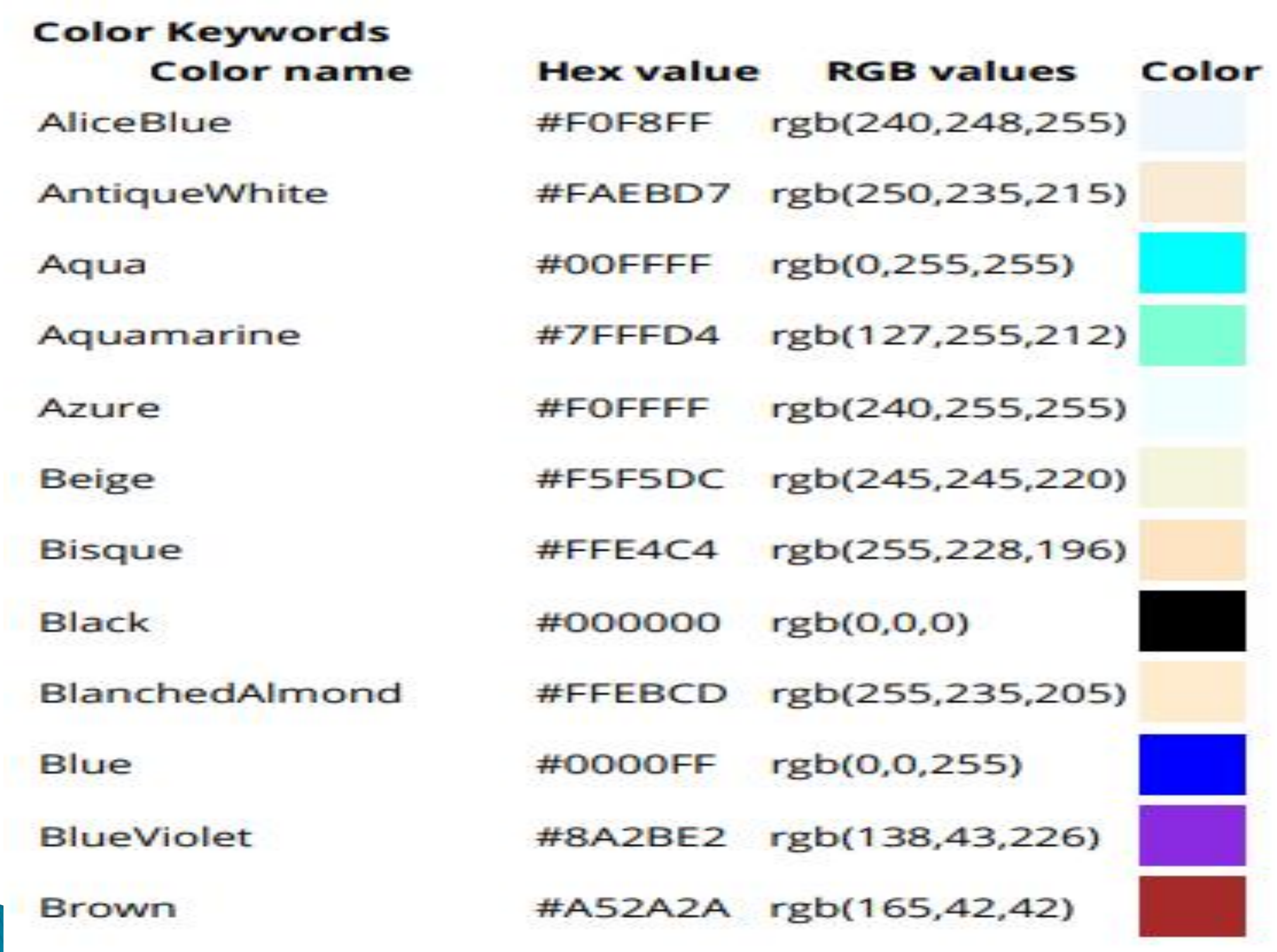
```
.parent-class .child-class {
 background-color: currentColor;
}
```

Color Keywords:

- Most browsers support using color keywords to specify a color.
- For example, to set the color of an element to blue, use the blue keyword:

```
.some-class {
 color: blue;
}
```

- CSS keywords are not case sensitive—blue, Blue and BLUE will all result in #0000FF.



Color Keywords

Color name	Hex value	RGB values	Color
AliceBlue	#F0F8FF	rgb(240,248,255)	⬜
AntiqueWhite	#FAEBD7	rgb(250,235,215)	⬜
Aqua	#00FFFF	rgb(0,255,255)	🟦
Aquamarine	#7FFFD4	rgb(127,255,212)	🟩
Azure	#F0FFFF	rgb(240,255,255)	⬜
Beige	#F5F5DC	rgb(245,245,220)	⬜
Bisque	#FFE4C4	rgb(255,228,196)	⬜
Black	#000000	rgb(0,0,0)	⬛
BlanchedAlmond	#FFEBCD	rgb(255,235,205)	⬜
Blue	#0000FF	rgb(0,0,255)	🟦
BlueViolet	#8A2BE2	rgb(138,43,226)	🟪
Brown	#A52A2A	rgb(165,42,42)	🟫

Hexadecimal Value

Background

- CSS colors may also be represented as a hex triplet, where the members represent the red, green and blue
- components of a color.
- Each of these values represents a number in the range of 00 to FF, or 0 to 255 in decimal notation.
- Uppercase and/or lowercase Hexadecimal values may be used (i.e. #3fc = #3FC = #33ffCC).
- The browser interprets #369 as #336699.
- If that is not what you intended but rather wanted #306090, you need to specify that explicitly.
- The total number of colors that can be represented with hex notation is 256 ^ 3 or 16,777,216.

Syntax

color: #rrggbb;

color: #rgb

Value   Description

rr	00 - FF for the amount of red
gg	00 - FF for the amount of green
bb	00 - FF for the amount of blue

```
.some-class {
 /* This is equivalent to using the color keyword 'blue' */
 color: #0000FF;
}
.also-blue {
 /* If you want to specify each range value with a single
number, you can!
This is equivalent to '#0000FF' (and 'blue') */
 color: #00F;
}
```

rgb() Notation:

- RGB is an additive color model which represents colors as mixtures of red, green, and blue light.
- In essence, the RGB representation is the decimal equivalent of the Hexadecimal Notation.
- In Hexadecimal each number ranges

from 00-FF which is equivalent to 0-255 in decimal and 0%-100% in percentages.

```
.some-class {
 /* Scalar RGB, equivalent to 'blue'*/
 color: rgb(0, 0, 255);
}
```

```
.also-blue {
  /* Percentile RGB values*/
  color: rgb(0%, 0%, 100%);
}
```

Syntax:

rgb(<red>, <green>, <blue>)

Value	Description
<red>	an integer from 0 - 255 or percentage from 0 - 100%
<green>	an integer from 0 - 255 or percentage from 0 - 100%
<blue>	an integer from 0 - 255 or percentage from 0 - 100%

rgba() Notation:

- Similar to rgb() notation, but with an additional alpha (opacity) value.

```
.red {
 /* Opaque red */
 color: rgba(255, 0, 0, 1);
}
.red-50p {
 /* Half-translucent red. */
 color: rgba(255, 0, 0, .5);
}
```

Syntax

rgba(<red>, <green>, <blue>, <alpha>);

Value	**Description**
<red>	an integer from 0 - 255 or percentage from 0 - 100%

<green> an integer from 0 - 255 or percentage from 0 - 100%

<blue> an integer from 0 - 255 or percentage from 0 - 100%

<alpha> a number from 0 - 1, where 0.0 is fully transparent and 1.0 is fully opaque

hsl() Notation:

- HSL stands for hue ("which color"), saturation ("how much color") and lightness ("how much white").
- Hue is represented as an angle from 0° to 360° (without units), while saturation and lightness are represented as percentages.

```
p {
 color: hsl(240, 100%, 50%); /* Blue */
}
```

hsla() Notation

- Similar to hsl() notation, but with an added alpha (opacity) value.

hsla(240, 100%, 50%, 0) /* transparent */

hsla(240, 100%, 50%, 0.5) /* half-translucent blue */

hsla(240, 100%, 50%, 1) /* fully opaque blue */

Syntax

hsla(<hue>, <saturation>%, <lightness>%, <alpha>);

Value **Description**

<hue> specified in degrees around the color wheel (without units), where 0° is red, 60° is yellow, 120° is green, 180° is cyan, 240° is blue, 300° is magenta, and 360° is red

<saturation> percentage where 0% is fully desaturated (grayscale) and 100% is fully saturated (vividly colored)

<lightness> percentage where 0% is fully black and 100% is fully white

<alpha> a number from 0 - 1 where 0 is fully transparent and 1 is fully opaque

Opacity:

Opacity Property:

- An element's opacity can be set using the opacity property.
- Values can be anywhere from 0.0 (transparent) to 1.0 (opaque).

Example Usage

```
<div style="opacity:0.8;">
 This is a partially transparent element
</div>
```

Property Value	Transparency
opacity: 1.0;	Opaque
opacity: 0.75;	25% transparent (75% Opaque)
opacity: 0.5;	50% transparent (50% Opaque)
opacity: 0.25;	75% transparent (25% Opaque**)**
opacity: 0.0;	Transparent

IE Compatibility for `opacity`:

- To use opacity in all versions of IE, the order is:

.transparent-element {
/* for IE 8 & 9 */
-ms-filter:"progid:DXImageTransform.Microsoft.Alpha(Opacity=60)"; // IE8
/* works in IE 8 & 9 too, but also 5, 6, 7 */

```
filter: alpha(opacity=60); // IE 5-7
 /* Modern Browsers */
 opacity: 0.6;
}
```

Length Units:

Creating scalable elements using rems and ems:

- You can use rem defined by the font-size of your html tag to style elements by setting their font-size to a value of rem and use em inside the element to create elements that scale with your global font-size.

HTML:

```
<input type="button" value="Button">
<input type="range">
<input type="text" value="Text">
```

Relevant CSS:

```
html {
 font-size: 16px;
}
input[type="button"] {
 font-size: 1rem;
 padding: 0.5em 2em;
}
input[type="range"] {
 font-size: 1rem;
width: 10em;
}
input[type=text] {
 font-size: 1rem;
 padding: 0.5em;
}
```

Font size with rem:

- CSS3 introduces a few new units, including the rem unit, which stands for "root em".
- First, let's look at the differences between em and rem.
- **em**: Relative to the font size of the parent.
- This causes the compounding issue

rem: Relative to the font size of the root or <html> element.

This means it's possible to declare a single font size for the html element and define all rem units to be a percentage of that.

- The main issue with using rem for font sizing is that the values are somewhat difficult to use.
- Here is an example of some common font sizes expressed in rem units, assuming that the base size is 16px :

10px = 0.625rem

12px = 0.75rem

14px = 0.875rem

16px = 1rem (base)

18px = 1.125rem

20px = 1.25rem

24px = 1.5rem

30px = 1.875rem

32px = 2rem

```
html {
 font-size: 16px;
}
h1 {
 font-size: 2rem; /* 32px */
}
```

```
p {
 font-size: 1rem; /* 16px */
}
li {
 font-size: 1.5em; /* 24px */
}
```

vmin and vmax:

- **vmin**: Relative to 1 percent of the viewport's smaller dimension
- **vmax**: Relative to 1 percent of the viewport's larger dimension
- In other words, 1 vmin is equal to the smaller of 1 vh and 1 vw
- 1 vmax is equal to the larger of 1 vh and 1 vw

Note: vmax is not supported in:

any version of Internet Explorer

Safari before version 6.1

vh and vw:

- CSS3 introduced two units for representing size.
- vh, which stands for viewport height is relative to 1% of the viewport height
- vw, which stands for viewport width is relative to 1% of the viewport width

```
div {
 width: 20vw;
 height: 20vh;
}
```

- Above, the size for the div takes up 20% of the width and height of the viewport

using percent %:

- One of the useful unit when creating a responsive application.
- Its size depends on its parent container.

Equation:

(Parent Container`s width) * (Percentage(%)) = Output

For Example:

- Parent has 100px width while the Child has 50%.
- On the output, the Child's width will be half(50%) of the Parent's, which is 50px.

HTML

```
<div class="parent">
 PARENT
 <div class="child">
 CHILD
 </div>
</div>
CSS
<style>
*{
 color: #CCC;
}
.parent{
 background-color: blue;
 width: 100px;
}
```

```
.child{
 background-color: green;
 width: 50%;
}
</style>
```

Pseudo-Elements

- Pseudo-elements are added to selectors but instead of describing a special state, they allow you to style certain parts of a document.
- The content attribute is required for pseudo-elements to render; however, the attribute can have an empty value (e.g. content: "").

```
div::after {
 content: 'after';
 color: red;
border: 1px solid red;
}
div {
 color: black;
 border: 1px solid black;
 padding: 1px;
}
div::before {
 content: 'before';
 color: green;
 border: 1px solid green;
}
```

Pseudo-Elements in Lists:

- Pseudo-elements are often used to change the look of lists (mostly for unordered lists, ul).

The first step is to remove the default list bullets:

```
ul {
 list-style-type: none;
}
```

- Then you add the custom styling. In this example, we will create gradient boxes for bullets.

```
li:before {
 content: "";
 display: inline-block;
 margin-right: 10px;
 height: 10px;
 width: 10px;
 background: linear-gradient(red, blue);
}
```

HTML:

```
<ul>
 <li>Test I</li>
 <li>Test II</li>
</ul>
```

Overlapping Elements with z-index:

- To change the default stack order positioned elements (position property set to relative, absolute or fixed), use the z-index property.
- The higher the z-index, the higher up in the stacking context (on the z-axis) it is placed.

Example

In the example below, a z-index value of 3 puts green on top, a z-index of 2 puts red just under it, and a z-index of 1 puts blue under that.

HTML

```
<div id="div1"></div>
<div id="div2"></div>
<div id="div3"></div>
```

CSS

```
div {
 position: absolute;
 height: 200px;
 width: 200px;
}
div#div1 {
 z-index: 1;
 left: 0px;
;
top: 0px;
 background-color: blue;
}
div#div2 {
 z-index: 3;
 left: 100px;
 top: 100px;
 background-color: green;
}
div#div3 {
 z-index: 2;
 left: 50p;x
top: 150px;
 background-color: red;
}
```

Absolute Position:

- When absolute positioning is used the box of the desired element is taken out of the Normal Flow and it no longer affects the position of the other elements on the page. Offset properties:

1. top
2. left
3. right
4. bottom

- specify the element should appear in relation to its next non-static containing element.

```
.abspos{
 position:absolute;
 top:0px;
left:500px;
}
```

- This code will move the box containing element with attribute class="abspos" down 0px and right 500px relative to its containing element.

Fixed position

- Defining position as fixed we can remove an element from the document flow and set its position relatively to the browser window.
- One obvious use is when we want something to be visible when we scroll to the bottom of a long

page.

```
#stickyDiv {
 position:fixed;
```

top:10px;
 left:10px;
}

Relative Position

- Relative positioning moves the element in relation to where it would have been in normal flow .Offset properties:

1. top
2. left
3. right
4. bottom

- are used to indicate how far to move the element from where it would have been in normal flow.

.relpos{
 position:relative;

```
 top:20px;
 left:30px;
}
```

- This code will move the box containing element with attribute class="relpos" 20px down and 30px to the right from where it would have been in normal flow.

Static positioning

- The default position of an element is static. To quote MDN:
- This keyword lets the element use the normal behavior, that is it is laid out in its current position in the flow.
- The top, right, bottom, left and z-index properties do not apply.

```
.element{
 position:static; }
```

Layout Control:

The display property:

- The display CSS property is fundamental for controlling the layout and flow of an HTML document.
- Most elements have a default display value of either block or inline (though some elements have other default values).

Inline

- An inline element occupies only as much width as necessary.
- It stacks horizontally with other elements of the same type and may not contain other non-inline elements.

<span>This is some <b>bolded</b> text!</span>

- As demonstrated above, two inline elements, <span> and <b>, are in-line (hence the name) and do not break the flow of the text.

Block

- A block element occupies the maximum available width of its' parent element.
- It starts with a new line and, in contrast to inline elements, it does not restrict the type of elements it may contain.

<div>Hello world!</div><div>This is an example!</div>

- The div element is block-level by default, and as shown above, the two block elements are vertically stacked and,unlike the inline elements, the flow of the text breaks.

Inline Block:

- The inline-block value gives us the best of both worlds:
- it blends the element in with the flow of the text while allowing us to use padding, margin, height and similar properties which have no visible effect on inline elements.
- Elements with this display value act as if they were regular text and as a result are affected by rules controlling the flow of text such as text-align.
- By default they are also shrunk to the the smallest size possible to accommodate

their content.

<!--Inline: unordered list-->

```
<style>
li {
 display : inline;
 background : lightblue;
 padding:10px;
 border-width:2px;
 border-color:black;
 border-style:solid;
 }
</style>
<ul>
<li>First Element </li>
<li>Second Element </li>
<li>Third Element </li>
</ul>
```

<!--block: unordered list-->

```
<style>
li {
 display : block;
 background : lightblue;
 padding:10px;
 border-width:2px;
 border-color:black;
 border-style:solid;
 }
</style>
<ul>
<li>First Element </li>
<li>Second Element </li>
<li>Third Element </li>
</ul>
```

```
<!--Inline-block: unordered list-->
<style>
li {
 display : inline-block;
 background : lightblue;
 padding:10px;
 border-width:2px;
 border-color:black;
 border-style:solid;
 }
</style>
<ul>
<li>First Element </li>
<li>Second Element </li>
<li>Third Element </li>
</ul>
```

none

- An element that is given the none value to its display property will not be displayed at all.
- For example let's create a div-element that has an id of myDiv:

<div id="myDiv"></div>

This can now be marked as not being displayed by the following CSS rule:

```
#myDiv {
 display: none;
}
```

To get old table structure using div:

- This is the normal HTML table structure

```
<style>
 table {
 width: 100%; }
```

```
</style>
<table>
 <tr>
 <td>
 I'm a table
 </td>
 </tr>
</table>
```

- You can do same implementation like this

```
<style>
 .table-div {
 display: table;
 }
```

```
. table-row-div {
 display: table-row;
 }
 .table-cell-div {
 display: table-cell;
 }
</style>
<div class="table-div">
 <div class="table-row-div">
 <div class="table-cell-div">
 I behave like a table now
 </div>
 </div>
</div>
```

Summary:

- Cascading and Specificity
- Colors
- Opacity
- Length Units
- Pseudo-Elements & Positioning

Thank You……….

If you have any quries please write to   info@uplatz.com".