Front-end TECH Talks

sprinklr®

HTML

5

&

CSS

3

Who Am I?

My name is Ankita Mehta

I am a Front End engineer

I am from Goa and a graduate from DA-IICT

I have been working in Sprinklr for 3 years now

I love travelling, watching movies and tv series

and .. I am also a big foodie

/Ankita-Mehta /mehtaankita /AnkitaMehta16 /ankita

Day 1

Let’s Start …

Learnlayout.com

Your guide to learn CSS Layout

Document Flow

Document Flow

What are different types of html elements? How are they displayed by default?

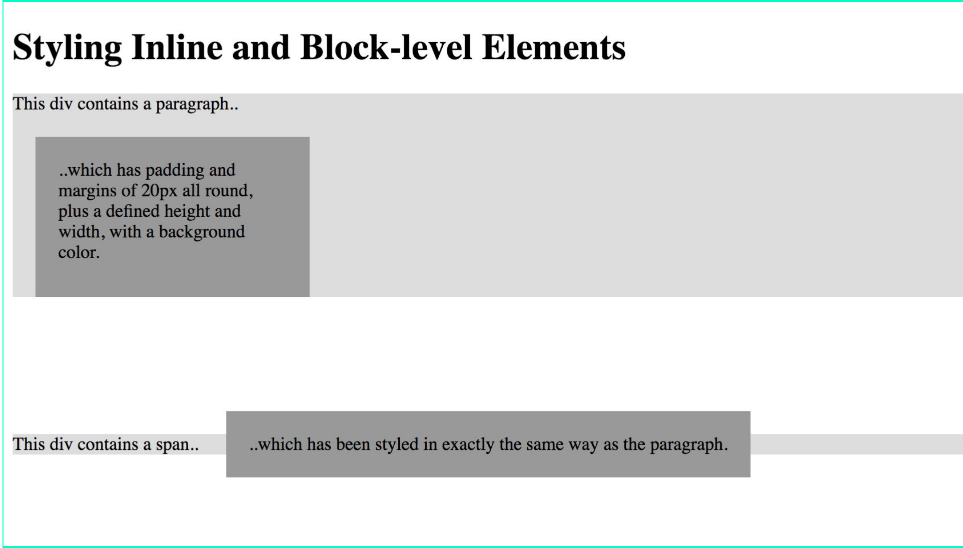
- **Inline Elements:**
  - Elements like: anchor tags, images and spans
  - Demo
- **Block-level Elements**
  - divs and paragraphs, and newer HTML5 members like article and section
  - these will normally, but not always, contain inline elements or other block-level elements
  - Browsers will (by default) format block-level elements with a line break before and after
  - will always start on a new line unless otherwise styled

Styling Elements

We're free to determine the width and height of a block-level element, but that is not the case with inline elements.

Padding and margins can be applied to inline elements, but they won't influence the dimensions of the containing element.

Demo



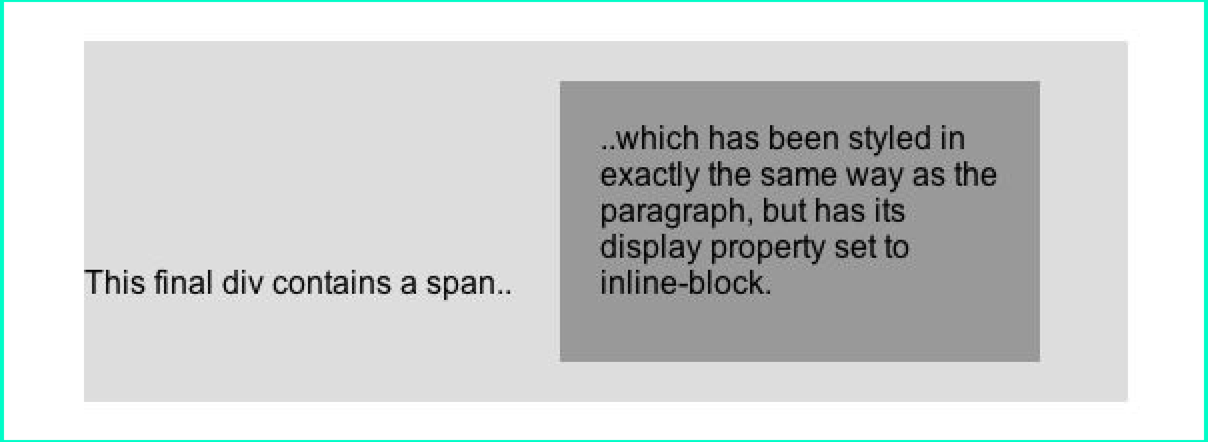
Styling Inline and Block-level Elements

This div contains a paragraph..

..which has padding and margins of 20px all round, plus a defined height and width, with a background color.

This div contains a span.. ..which has been styled in exactly the same way as the paragraph.

What if you want to achieve this?

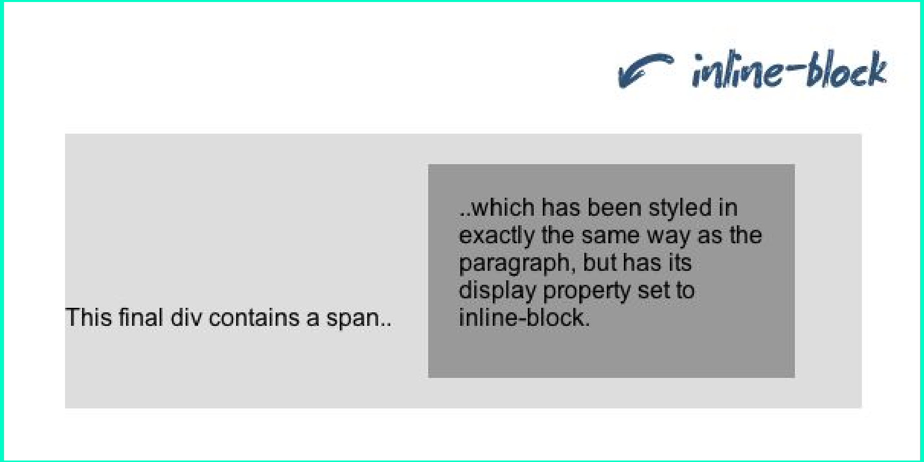


This final div contains a span..

..which has been styled in exactly the same way as the paragraph, but has its display property set to inline-block.

Inline block

Inline-block elements preserve properties such as width, height, margin and padding as they're applied to block level elements, whilst maintaining the structural qualities of an inline element.



This final div contains a span..

..which has been styled in exactly the same way as the paragraph, but has its display property set to inline-block.

Other Display Types

- List-item

  Each element with the list-item value applied is displayed vertically, as you'd expect in a list, but they also have bullet points next to them.

- None

  It's visually (along with all elements contained within it) removed from the document

What is Normal Document Flow?

It's how a page is presented when you do nothing to it with regard to structural styling.

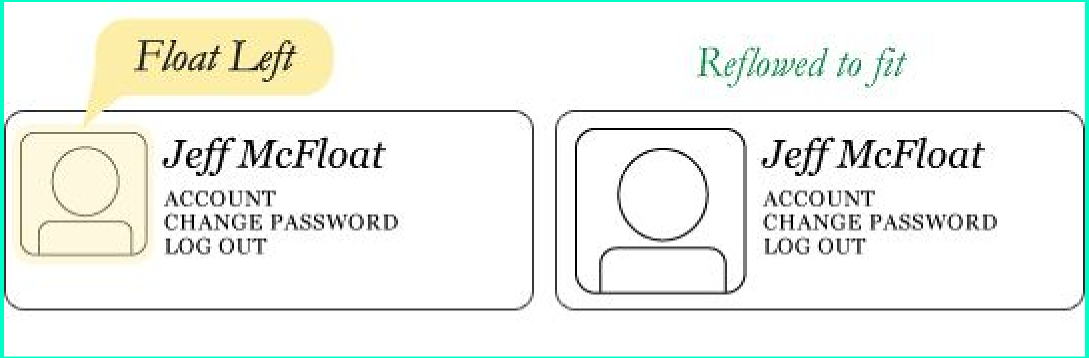
Browsers display content in the order that it's encountered. Top to Bottom.

Demo

Properties which impact the normal flow:

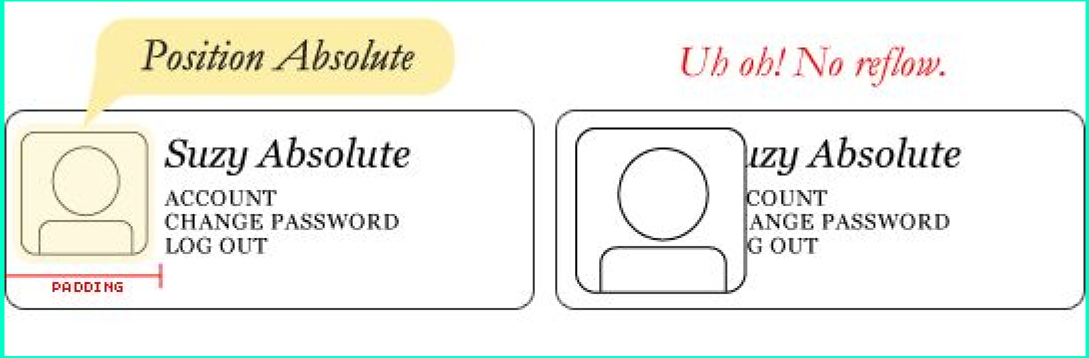
- Float: left, right, none (default), inherit
- Position: absolute, relative, fixed, static (default), and inherit

You'll generally want to use floats when developing a layout and use positioning on specific elements that you want to break out of the layout.



Floated elements remain a part of the flow of the web page.

Absolutely positioned page elements are removed from the flow of the webpage. Absolutely positioned page elements will not affect the position of other elements and other elements will not affect them, whether they touch each other or not.



POSITIONING ELEMENTS

Positioning Properties

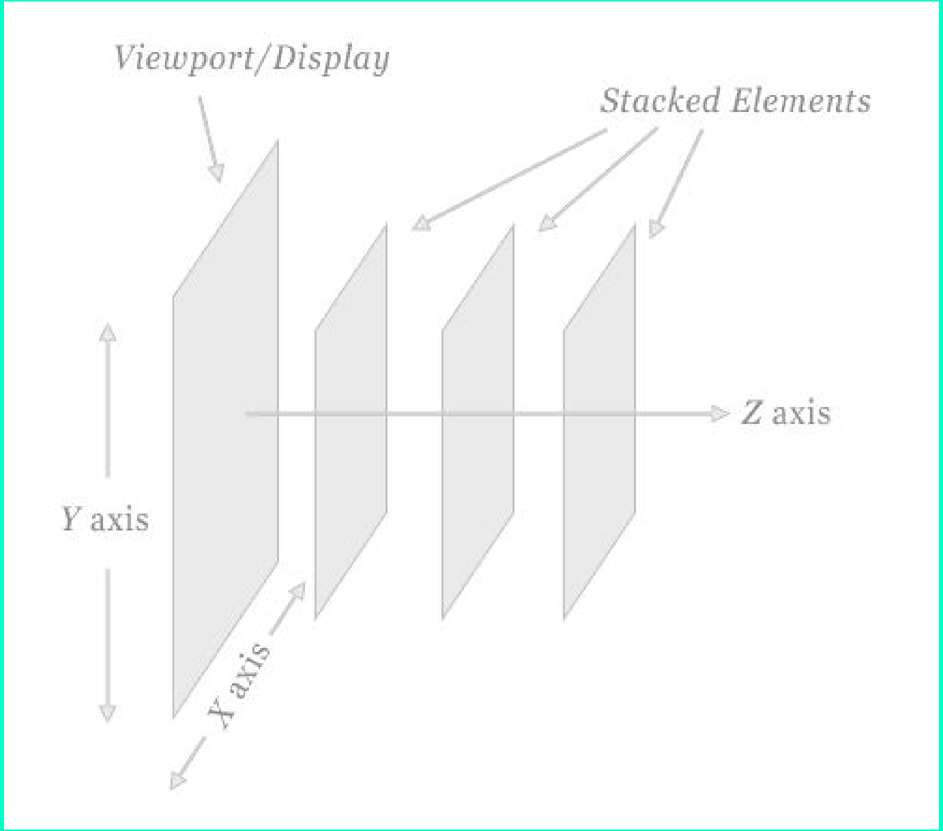
- position
- bottom
- left
- top
- right
- z-index

Demo

Z-INDEX

What is z-index?

The z-index property determines the stack level of an HTML element. The “stack level” refers to the element’s position on the Z axis (as opposed to the X axis or Y axis). A higher z-index value means the element will be closer to the top of the stacking order. This stacking order runs perpendicular to the display, or viewport.



Time for small Quiz

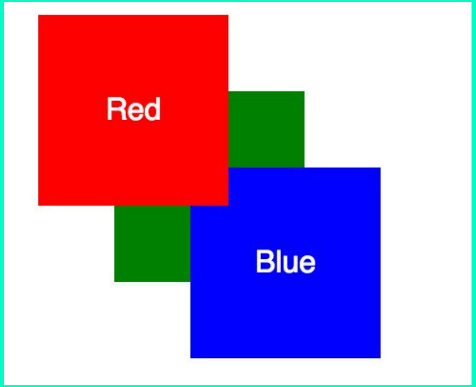
`z-index`

HTML

```html
<div>
  <span class="red">Red</span>
</div>
<div>
  <span class="green">Green</span>
</div>
<div>
  <span class="blue">Blue</span>
</div>
```

CSS

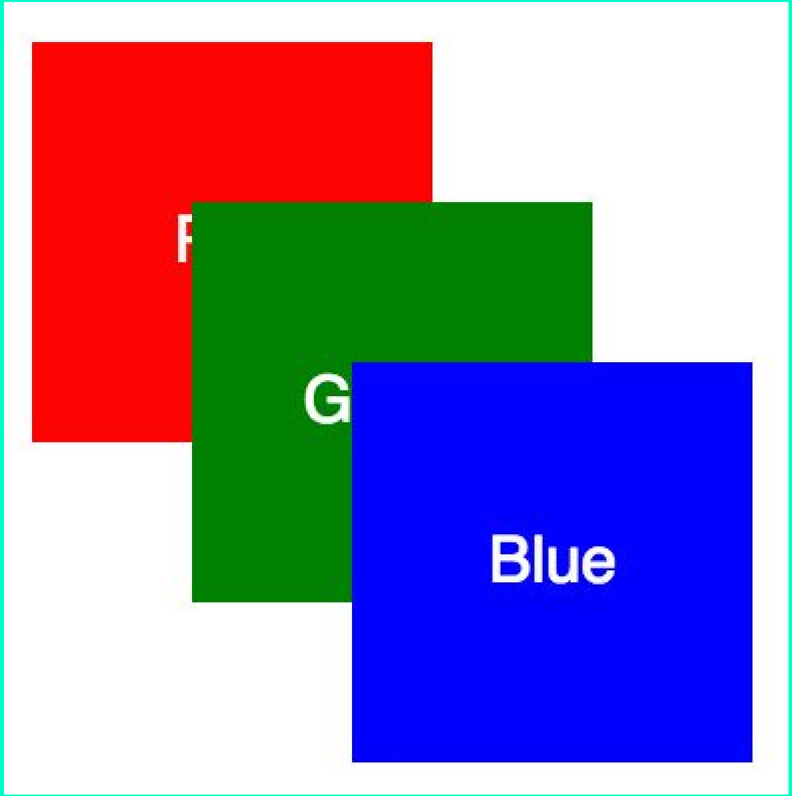
```css
.red, .green, .blue {
  position: absolute;
}
.red {
  background: red;
  z-index: 1;
}
.green {
  background: green;
}
.blue {
  background: blue;
}
```



Here’s the challenge: try to see if you can make the red <span> element stack behind the blue and green <span> elements without breaking any of the following rules:

- Do not alter the HTML markup in any way.
- Do not add/change the z-index property of any element.
- Do not add/change the position property of any element.

Run Code

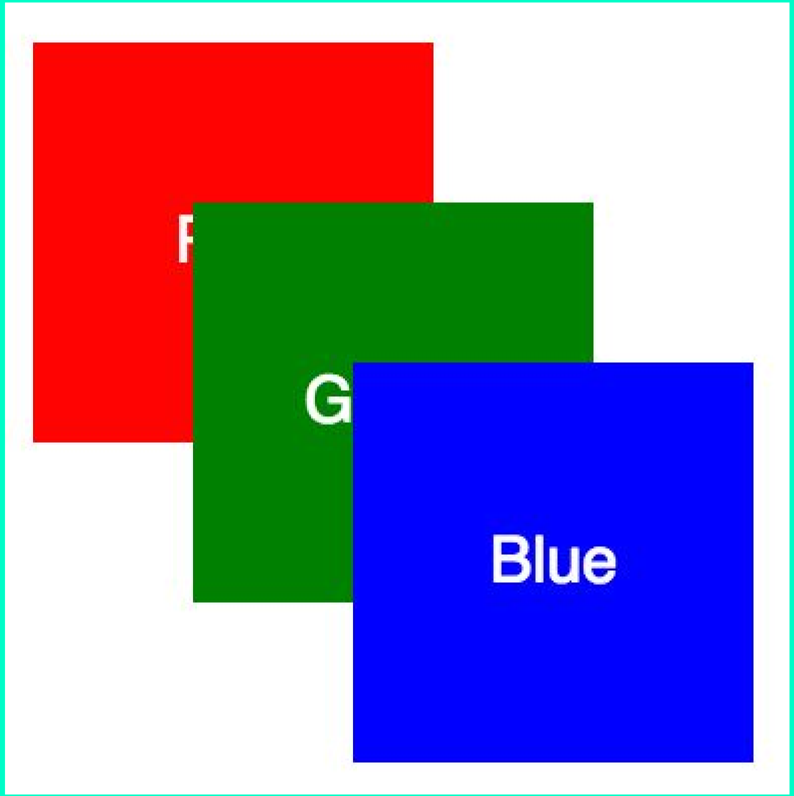


Ready for the solution?

The solution is to add an opacity value less than 1 to the first <div> (the parent of the red <span>). Here is the CSS that was added to the Code above:

```
div:first-child {
  opacity: .99;
}
```

Wondering why this is happening?



When you introduce the position property into the mix, any positioned elements (and their children) are displayed in front of any non-positioned elements. (To say an element is “positioned” means that it has a position value other than static, e.g., relative, absolute, etc.)

Finally, when z-index is involved, things get a little trickier. At first it’s natural to assume elements with higher z-index values are in front of elements with lower z-index values, and any element with a z-index is in front of any element without a z-index, but it’s not that simple. First of all, z-index only works on positioned elements. If you try to set a z-index on an element with no position specified, it will do nothing. Secondly, z-index values can create **stacking contexts**, and now suddenly what seemed simple just got a lot more complicated.

Stacking Contexts

Groups of elements with a common parent that move forward or backward together in the stacking order make up what is known as a stacking context.

Every stacking context has a single HTML element as its root element. When a new stacking context is formed on an element, that stacking context confines all of its child elements to a particular place in the stacking order. That means that if an element is contained in a stacking context at the bottom of the stacking order, there is no way to get it to appear in front of another element in a different stacking context that is higher in the stacking order, even with a z-index of a billion!

Stacking Contexts

New stacking contexts can be formed on an element in one of three ways:

- When an element is the root element of a document (the <html> element)
- When an element has a position value other than static and a z-index value other than auto
- **When an element has an opacity value less than 1**

Stacking Order Within the Same Stacking Context

Here are the basic rules to determine stacking order within a single stacking context (from back to front):

1. The stacking context's root element
2. Positioned elements (and their children) with negative z-index values (higher values are stacked in front of lower values; elements with the same value are stacked according to appearance in the HTML)
3. Non-positioned elements (ordered by appearance in the HTML)
4. Positioned elements (and their children) with a z-index value of auto (ordered by appearance in the HTML)
5. Positioned elements (and their children) with positive z-index values (higher values are stacked in front of lower values; elements with the same value are stacked according to appearance in the HTML)

z-index will only work on an element whose position property has been explicitly set to **absolute, fixed, or relative**.

Stacking Order Within the Same Stacking Context

Note: positioned elements with negative z-indexes are ordered first within a stacking context, which means they appear behind all other elements. Because of this, it becomes **possible for an element to appear behind its own parent**, which is normally not possible. This will only work if the element's parent is in the same stacking context and *is not the root element* of that stacking context.

Wrapping up the solution

This order is assuming the original CSS.

```
<div><!-- 1 -->
  <span class="red"><!-- 6 --></span>
</div>
<div><!-- 2 -->
  <span class="green"><!-- 4 --><span>
</div>
<div><!-- 3 -->
  <span class="blue"><!-- 5 --></span>
</div>
```

When we add the opacity rule to the first <div>, the stacking order changes like so:

```
<div><!-- 1 -->
  <span class="red"><!-- 1.1 --></span>
</div>
<div><!-- 2 -->
  <span class="green"><!-- 4 --><span>
</div>
<div><!-- 3 -->
  <span class="blue"><!-- 5 --></span>
</div>
```

I've used dot notation to show that a new stacking context was formed and span.red is now the first element within that new context.

CSS UNITS

Meet the CSS Units

Absolute units include:

- pt (points) - represents 1/72 of an inch
- px (pixels)
- cm (centimeters)
- mm (millimeters)
- in (inches)
- pc (picas) - represents 1/6 of an inch

Absolute units are a digital representation of actual measurements in the physical world. These units are not related to the size of the screen or its resolution.

Absolute units Demo

Meet the CSS Units

Relative units include:

- em (named after print ems, but not the same) - if the font size of parent element is 20px then the value of 1em will compute to 20px for all immediate child element
- rem (root em) - The value of a rem always stays equal to the font size of the root element.
- ex (x-height) - 1ex is equal to the size of the lowercase ‘x’ in the font being used
- ch (character) - it is related to the ‘0’ character. 1ch is the advance measure of the character ‘0’ in a font.

These values are relative to some other predefined value or feature. Relative units make it easy to properly size elements since we can relate their width, height, font-size, etc. to some other base parameter.

Meet the CSS Units

Viewport-relative units include:

- vh (viewport height)
- vw (viewport width)
- vmin (viewport minimum)
- vmax (viewport maximum)

All units Demo

Day 2

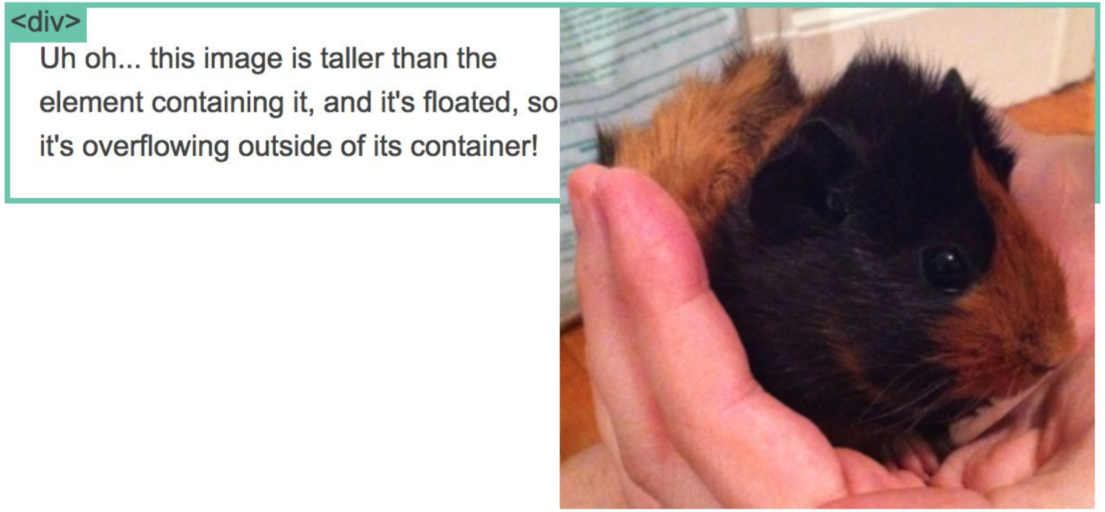
Let’s Start ...

CLEARFIX

Clearfix Hack

Bad thing that can sometimes happen when using floats:

```
img {
 float: right;
}
```

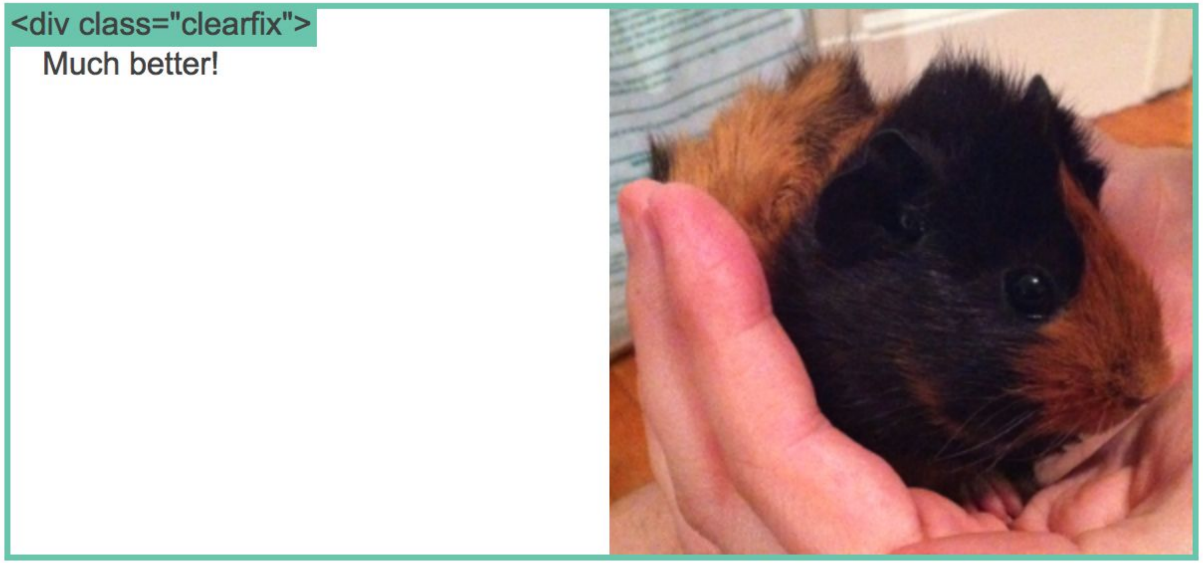


Uh oh... this image is taller than the element containing it, and it's floated, so it's overflowing outside of its container!

There is a way to fix this, but it's ugly. It's called the **clearfix hack**.

Clearfix Hack

```
.clearfix {
 overflow: auto;
}
```



Clearfix Hack

But .. Overflow auto generates a side scroll in your div when its contents are larger than the div's desired size. So another popular solution is:

HTML

```
<div class="clearfix">
 <div style="float:left">Hello!</ div>
 <div style="float:left">Whatsup?</ div>
</div>
```

CSS

```
.clearfix:after {
 clear: both;
 content: '';
 display: block;
}
```

Further Read here.

Pseudo-Classes And Pseudo-Elements

Pseudo-Class

- A pseudo-class is basically a phantom state or specific characteristic of an element that can be targeted with CSS. A few common pseudo-classes are `:link`, `:visited`, `:hover`, `:active`, `:focus`, `:first-child` and `:nth-child`.
- Also, pseudo-classes are always preceded by a colon (:). Then comes the name of the pseudo-class and sometimes a value in parentheses. `:nth-child` anyone?
- A state pseudo-class usually come into play when the user performs an action. An “action” in CSS could also be “no action”, as in the case of a link that hasn’t been visited yet.

Let’s check them out.

:LINK

- The `:link` pseudo-class represents the “normal” state of links and is used to select links that have not yet been visited.
- Declaring the `:link` pseudo-class before all other pseudo-classes in this category is recommended. The order of all four is this: `:link, :visited, :hover, :active`

```
a:link {
 color: orange;
}
```

If you use it as follows, it can be omitted:

```
a {
 color: orange;
}
```

:VISITED

- The `:visited` pseudo-class is used in links that have been visited.
- Position `:visited` pseudo-class second in order (after the `:link` pseudo-class).

```
a:visited {
 color: blue;
}
```

:HOVER

- The `:hover` pseudo-class is used to style an element when the user’s pointer is above it.
- It doesn’t have to be restricted to links, although that is the most common use case.
- It should appear third in order (after the `:visited` pseudo-class).

```
a:hover {
 color: orange;
}
```

:ACTIVE

- The `:active` pseudo-class is used to style an element that has been “activated” either by a pointing device or by a tap on a touchscreen device.
- It can also be triggered by the keyboard, just like the `:focus` pseudo-class.
- It works very similarly to `:focus`, the difference being that the `:active` pseudo-class is an event that occurs between the mouse button being clicked and being released.
- It should appear fourth in order (after the `:hover` pseudo-class).

```
a:active {
 color: rebeccapurple;
}
```

:focus

- The `:focus` pseudo-class is used to style an element that has gained focus via a pointing device, from a tap on a touchscreen device or via the keyboard.
- It's used a lot in form elements.

```
a:focus {
 color: green;
}

input:focus {
 background: #eee;
}
```

:FIRST-CHILD

- The `:first-child` pseudo-class represents the first child of its parent element.

  In the following example, the first li element will be the only one with orange text.

  HTML

```
<ul>
   <li>This text will be orange.</li>
   <li>Lorem ipsum dolor sit amet.</li>
   <li>Lorem ipsum dolor sit amet.</li>
</ul>
```

  CSS

```
li:first-child {
 color: orange;
}
```

:LAST-CHILD

- The `:last-child` pseudo-class represents the last child of its parent element.

  In the following example, the last li element will be the only one with orange text.

  HTML

  ```
  <ul>
     <li>Lorem ipsum dolor sit amet.</li>
     <li>Lorem ipsum dolor sit amet.</li>
     <li>This text will be orange.</li>
  </ul>
  ```

  CSS

  ```
  li:last-child {
   color: orange;
  }
  ```

:NOT

- The `:not` pseudo-class is also known as the negation pseudo-class. It accepts an argument — basically, another “selector” — inside parentheses. The argument can actually be another pseudo-class.

  In the following example, the `:not` pseudo-class matches an element that is not represented by the argument.

  HTML

  ```
  <ul>
     <li class="first-item">Lorem ipsum dolor sit amet. </li>
     <li>Lorem ipsum dolor sit amet. </li>
     <li>Lorem ipsum dolor sit amet. </li>
     <li>Lorem ipsum dolor sit amet. </li>
  </ul>
  ```

  CSS

  ```
  li:not(.first-item) {
   color: orange;
  }
  ```

:NTH-CHILD

- The `:nth-child` pseudo-class targets one or more elements depending on their order in the markup.
- All of the `:nth` pseudo-classes take an argument, which is a formula that we type in parentheses. The formula may be a single integer, a formula structured as an+b or the keyword `odd` or `even`.
- In the `an+b` formula:
    - the `a` is a number (called an integer);
    - the `n` is a literal n (in other words, we will actually type the letter `n` in the formula);
    - the `+` is an operator that may be either a plus sign (`+`) or a minus sign (`-`);
    - the `b` is an integer as well but is only required if an operator is being used.
- Demo

:NTH-OF-TYPE

- The `:nth-of-type` pseudo-class works basically the same as `:nth-child`, the main difference being that it's more specific because we're targeting a specific element relative to like elements contained within the same parent element.

HTML

```html
<article>
   <h1>Heading Goes Here</h1>
   <p>Lorem ipsum dolor sit amet.</p>
   <a href=""><img src="images/rwd.png" alt="Mastering RWD"></a>
   <p>This text will be orange.</p>
</article>
```

CSS

```css
p:nth-of-type(2) {
 color: orange;
}
```

:CHECKED

- The `:checked` pseudo-class targets radio buttons, checkboxes and option elements that have been checked or selected.

  Demo

:DISABLED

- The `:disabled` pseudo-class targets a form element in the disabled state.
- An element in a disabled state can’t be selected, checked or activated or gain focus.

  Demo

:EMPTY

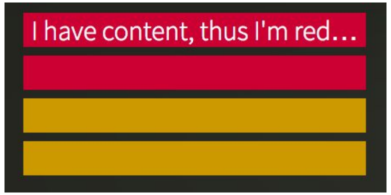
- The `:empty` pseudo-class targets elements that have no content in them of any kind at all.
- If an element has a letter, another HTML element or even a space, then that element would not be empty.
- An HTML comment inside an element does not count as content in this case.

HTML

```html
<div>This box is orange</div>
<div> </div>
<div></div>
<div><!-- This comment is not considered content --></div>
```

CSS

```css
div {
 background: orange;
 height: 30px;
 width: 200px;
}

div:empty {
 background: yellow;
}
```



Pseudo-Element

- Pseudo-Elements are like virtual elements that we can treat as regular HTML elements.
- They don't exist in the document tree or DOM. This means we don't actually type the pseudo-elements, but rather create them with CSS.
- A few common pseudo-elements are `:after, :before` and `:placeholder`

:BEFORE

- The `:before` pseudo-element, like its sibling `:after`, adds content (text or a shape) to another HTML element.
- Again, this content is not actually present in the DOM but can be manipulated as if it were. And the content property needs to be declared in the CSS.

HTML

```
<h1>Ankita</h1>
```

CSS

```
h1:before {
 content: "Hello "; /* Note the space after the word Hello. */
}
```

Output

```
Hello Ankita!
```

:AFTER

- The `:after` pseudo-element is used to add content (either text or a shape) to another HTML element.
- This content is not actually present in the DOM, but it can be manipulated as if it were.
- In order for it to work, the content property needs to be declared in the CSS.

HTML

```
<h1>Ankita</h1>
```

CSS

```
h1:after {
 content: ", Front End Developer!";
}
```

Output

```
Ankita, Front End Developer!
```

:PLACEHOLDER

- The `:placeholder` pseudo-element targets placeholder text used in form elements via the placeholder HTML attribute.

HTML

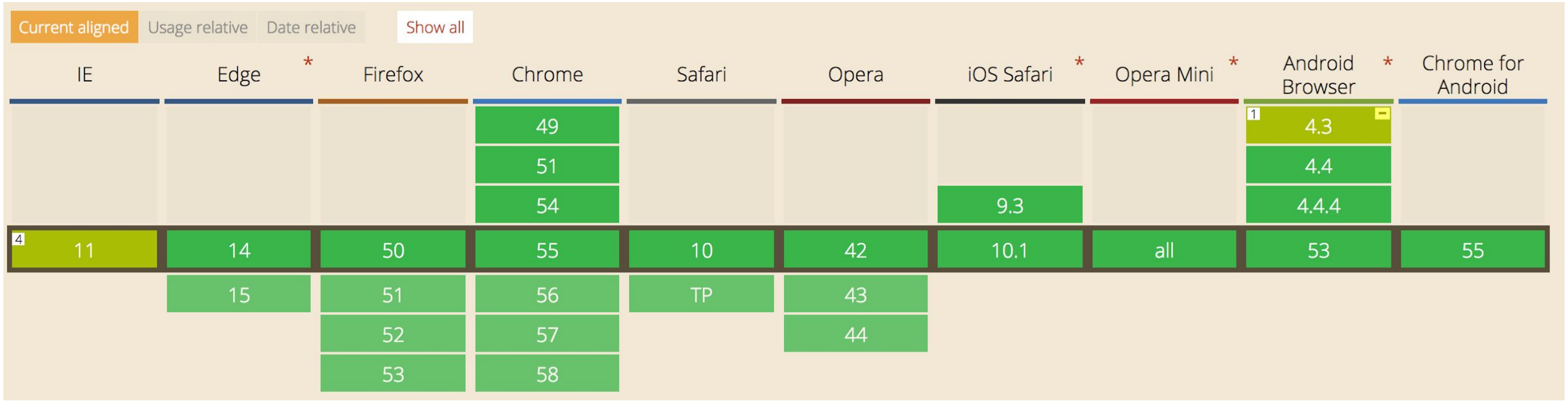
```
<input type="email" placeholder="name@domain.com">
```

CSS

```
input::-moz-placeholder {
 color:#666;
}

input::-webkit-input-placeholder {
 color:#666;
}

/* IE 10 only */
input:-ms-input-placeholder {
 color:#666;
}

/* Firefox 18 and below */
input:-moz-input-placeholder {
 color:#666;
}
```

Flex

Flex Support


Current aligned
Usage relative
Date relative
Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			49					[1] 4.3	
			51					4.4	
			54			9.3		4.4.4	
[4] 11	14	50	55	10	42	10.1	all	53	55
	15	51	56	TP	43				
		52	57		44				
		53	58						

For the latest support, refer here.

FLEX VISUAL GUIDE

Link

Time for small Quiz

Ids vs Classes

HTML

```html
<div class="content large-box" id="media-box">
  Hello
</div>
```

CSS

```css
#media-box {
  color: red;
}

.content.large-box {
  color: green;
}
```

What is the color of the text? Option 1 or 2?

Option 1:

Hello

Option 2:

Hello

The color of the text is going to be red.

Wondering why this is happening?

Option 1:

Hello

CSS Specificity

CSS Specificity

- Specificity determines which CSS property declaration is applied when two or more declarations apply to the same element with competing property declarations.
- The most specific selector takes precedence.
- If specificity is equal, the last declaration in the source order takes precendence.

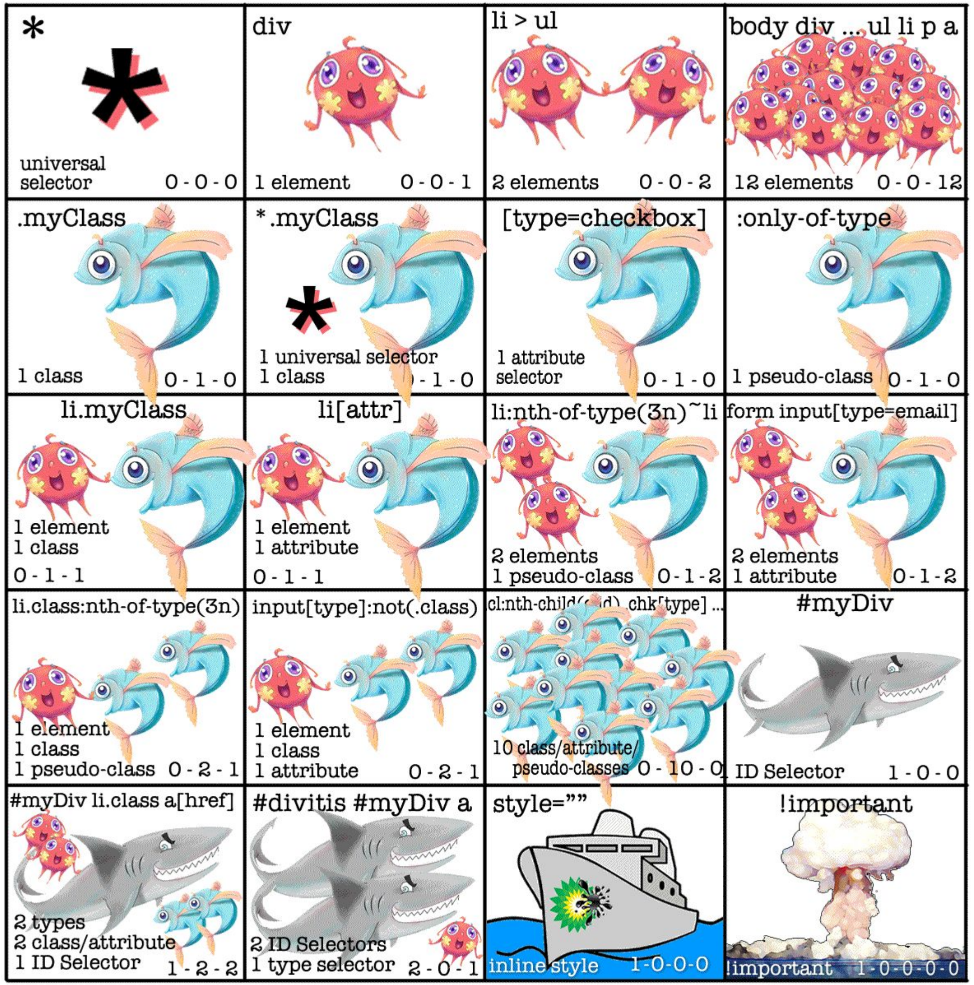
- **!important:** Any property declaration with the term !important takes highest precendence, even over inline styles. It’s as if the weight of the selector with the !important declaration were 1-X-A-B-C, for that property only (where A, B and C are the actual values of the parent selector as described below). Because of this, important should not be used, but can be handy in debugging.
- **style="":** If the author includes style attribute on an element, the inline style will take precedence over any styles declared in an external or embedded stylesheet, other than !important declarations. Note that styles added with the JavaScript style (i.e. document.getElementById('myID').style.color = 'purple';) property is in effect creating an inline style (... id="myID" style="color: purple"...). The weight of style="" is 1-0-0-0.
- **id:** Among the selectors you’ll actually be including in your stylesheet, ID’s have the highest specificity. The weight of an id selector is 1-0-0 per id.
- **class/pseudo-class/attributes:** As shown in the second code block above, class selectors, attribute selectors and pseudo-classes all have the same weight of 0-1-0.
- **type:** Element type selectors and pseudo-elements, like :first-letter or :after have the lowest value in terms of specificity with 0-0-1 per element.

Reference to the diagram:

1. X-0-0: The number of ID selectors, represented by Sharks
2. 0-Y-0: The number of class selectors, attributes selectors, and pseudo-classes, represented by Fish
3. 0-0-Z: The number of type selectors and pseudo-elements, represented by Plankton a la Spongebob
4. *: The universal selector has no value
5. +, >, ~: combinators, although they allow for more specific targeting of elements, they do not increase specificity values
6. :not(x): The negation selector has no value, but the argument passed increases specificity

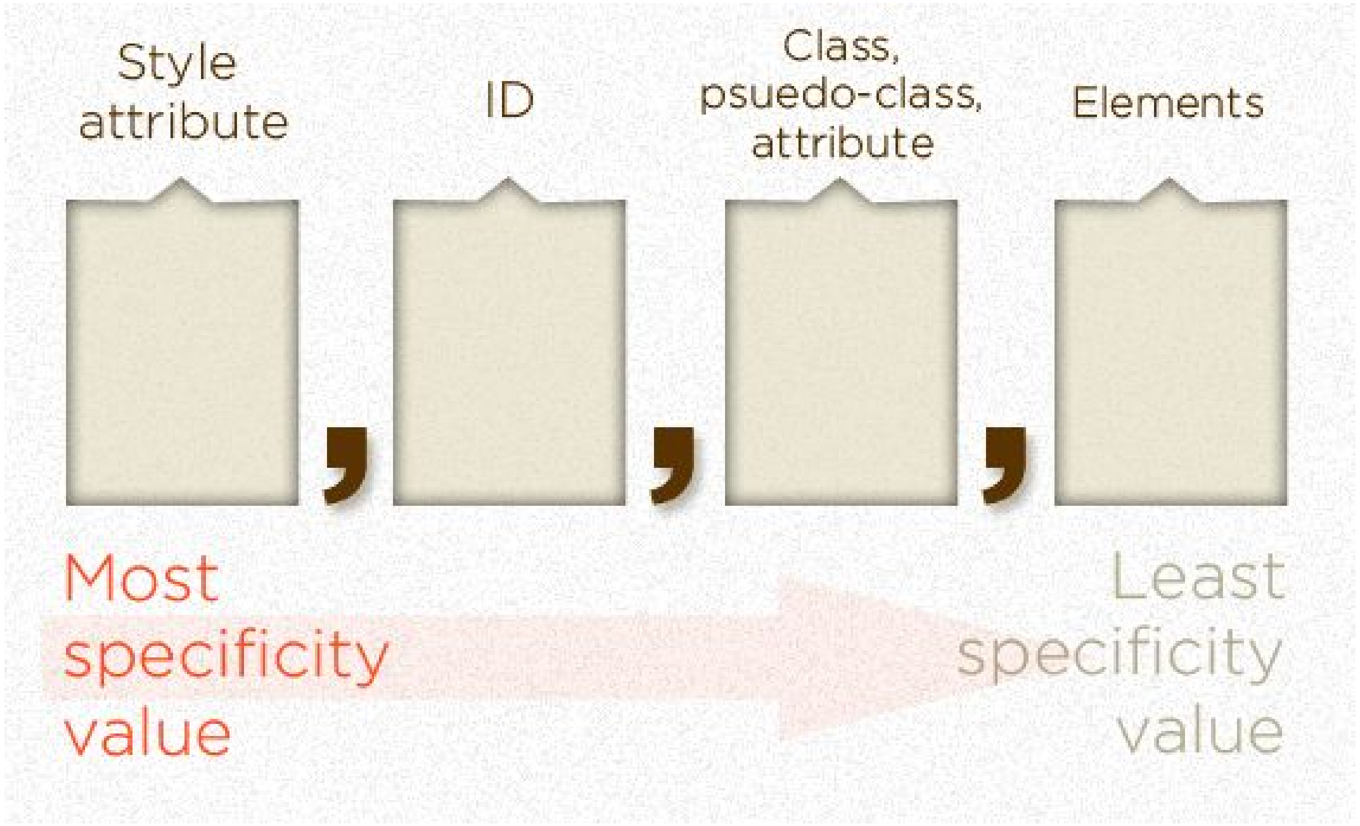
CSS SpeciFISHity

with Plankton, Fish and Sharks

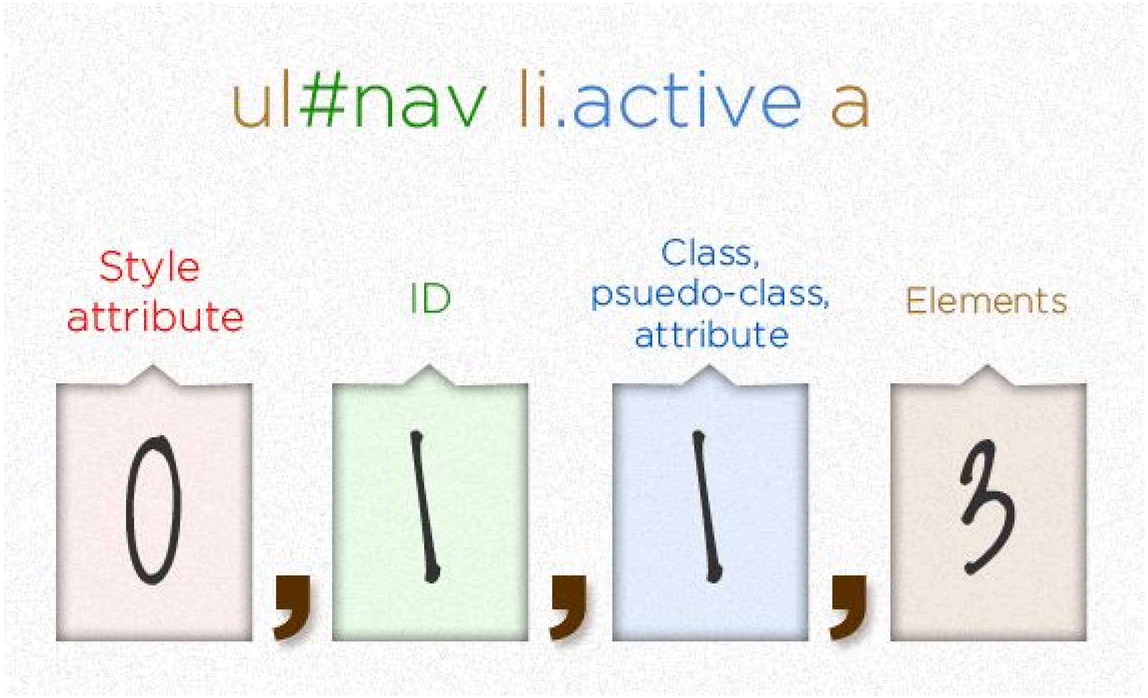


* universal selector 0 - 0 - 0	div 1 element 0 - 0 - 1	li > ul 2 elements 0 - 0 - 2	body div ... ul li p a 12 elements 0 - 0 - 12
.myClass 1 class 0 - 1 - 0	*.myClass 1 universal selector 1 class 0 - 1 - 0	[type=checkbox] 1 attribute selector 0 - 1 - 0	:only-of-type 1 pseudo-class 0 - 1 - 0
li.myClass 1 element 1 class 0 - 1 - 1	li[attr] 1 element 1 attribute 0 - 1 - 1	li:nth-of-type(3n)~li 2 elements 1 pseudo-class 0 - 1 - 2	form input[type=email] 2 elements 1 attribute 0 - 1 - 2
li.class:nth-of-type(3n) 1 element 1 class 1 pseudo-class 0 - 2 - 1	input[type]:not(.class) 1 element 1 class 1 attribute 0 - 2 - 1	cl:nth-child(odd) .chk[type] ... 10 class/attribute/ pseudo-classes 0 - 10 - 0	#myDiv ID Selector 1 - 0 - 0
#myDiv li.class a[href] 2 types 2 class/attribute 1 ID Selector 1 - 2 - 2	#divitis #myDiv a 2 ID Selectors 1 type selector 2 - 0 - 1	style="" inline style 1-0-0-0	!important !important 1-0-0-0-0

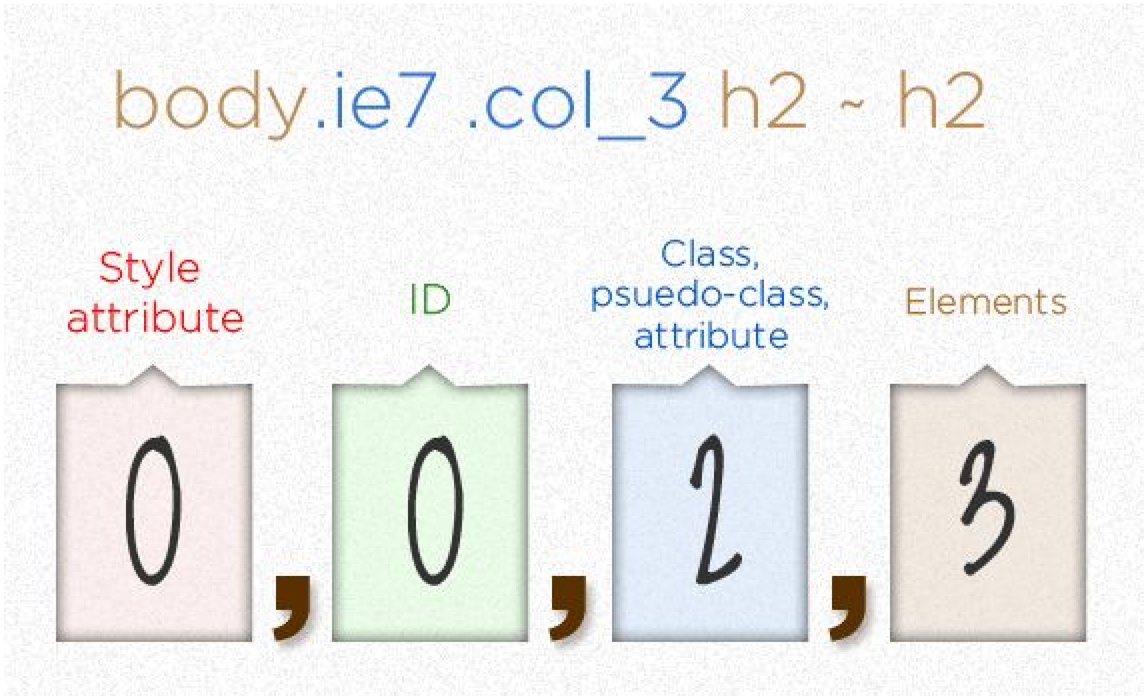
Let's take a look at how the numbers are actually calculated:



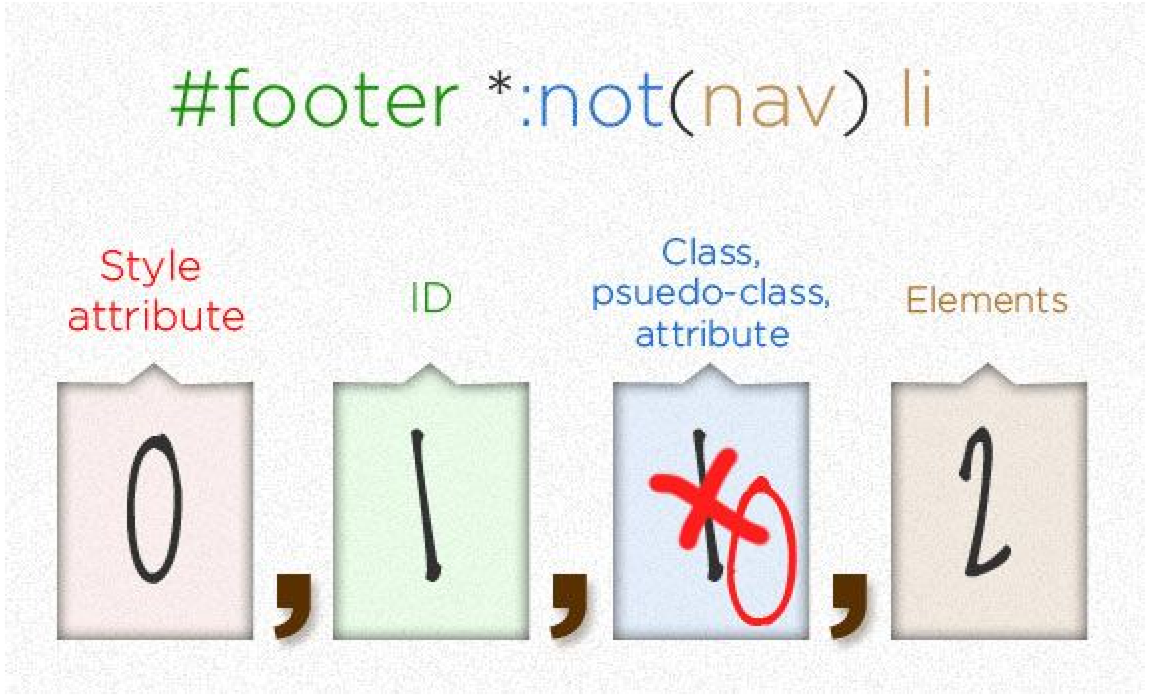
Sample Calculation



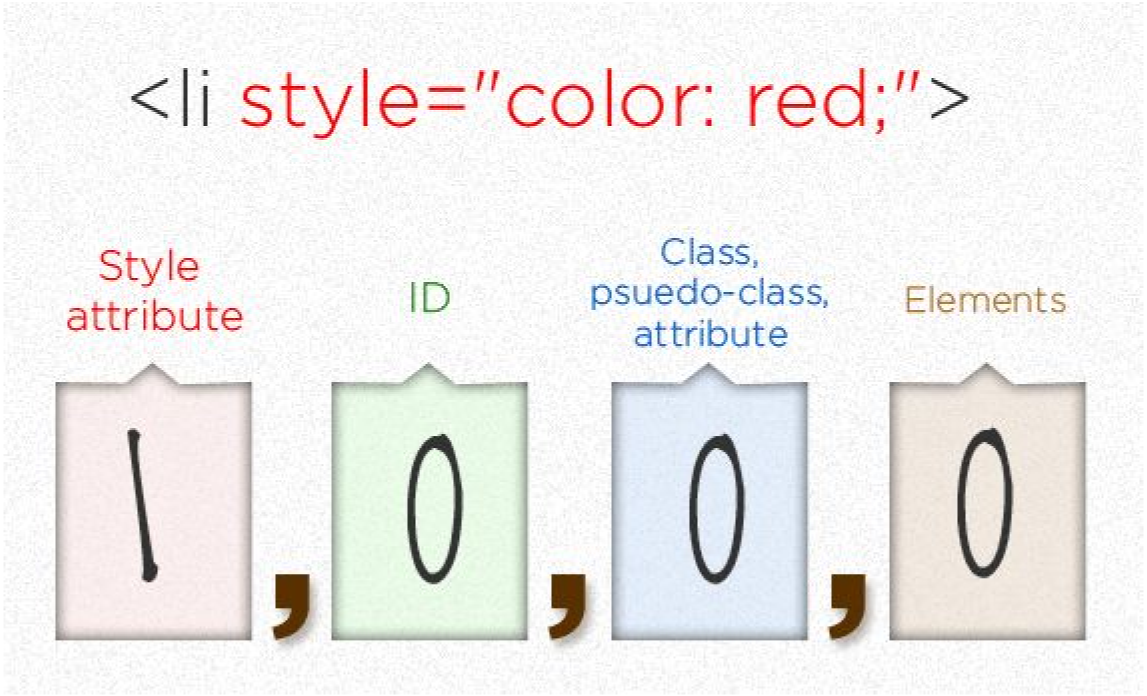
Sample Calculation



Note: The :not() sort-of-pseudo-class adds no specificity by itself, only what's inside the parens is added to specificity value.



Sample Calculation



CSS Specificity - Important Notes

- The universal selector (*) has no specificity value (0,0,0,0)
- Pseudo-elements (e.g. :first-line) get 0,0,0,1 unlike their psuedo-class brethren which get 0,0,1,0
- The pseudo-class :not() adds no specificity by itself, only what's inside it's parentheses.
- The !important value appended a CSS property value is an automatic win. It overrides even inline styles from the markup. The only way an !important value can be overridden is with another !important rule declared later in the CSS and with equal or great specificity value otherwise. You could think of it as adding 1,0,0,0,0 to the specificity value.

CSS Specificity - Important Notes

- Specificity is not inherited. If you declare 27 values on a parent of a paragraph, and even add !important, but declare the paragraph separately, the property declared on the element will be applied. Inheritance does not trump such declarations.

```
<div id="x" class="y"><p>Hi</p><div>
div#x.y {color:red;}           1-1-0
p {color: blue;}               0-0-1
```

The paragraph will be blue not red, as although the first declaration is more specific, and colors are inherited, the second declaration is actually applied to the element, whereas the first is applied to the parent. The color is only inherited if not specifically declared on the descendant. And, in this case, it is declared.

Repaint & Reflow

CSS PERFORMANCE MAKING YOUR JAVASCRIPT SLOW?

A lot of developers are guilty of adding superficial CSS3 animations or manipulating multiple DOM elements without considering the consequences.

Let’s dive into CSS performance optimisation.

Repaints

- A repaint occurs when changes are made to elements that affect visibility but not the layout.
- For example, opacity, background-color, visibility, and outline.
- Repaints are expensive because the browser must check the visibility of all other nodes in the DOM — one or more may have become visible beneath the changed element.

Reflows

- Reflows have a bigger impact.
- This refers to the re-calculation of positions and dimensions of all elements, which leads to re-rendering part or all of the document.
- Changing a single element can affect all children, ancestors, and siblings

Impact of Reflows and Repaint

- Both are browser-blocking; neither the user or your application can perform other tasks during the time that a repaint or reflow occurring.
- In extreme cases, a CSS effect could lead to slower JavaScript execution. This is one of the reasons you encounter issues such as jerky scrolling and unresponsive interfaces.

When reflows are triggered

- **Adding, removing or changing visible DOM elements**
  The first is obvious; using JavaScript to change the DOM will cause a reflow.

- **Adding, removing or changing CSS styles**
  Similarly, directly applying CSS styles or changing the class may alter the layout. Changing the width of an element can affect all elements on the same DOM branch and those surrounding it.

- **CSS3 animations and transitions**
  Every frame of the animation will cause a reflow.

When reflows are triggered

- **Using offsetWidth and offsetHeight**

  Bizarrely, reading an element's offsetWidth and offsetHeight property can trigger an initial reflow so the figures can be calculated.

- **User actions**

  Finally, the user can trigger reflows by activating:

  - `:hover` effect
  - entering text in a field
  - resizing the window
  - changing the font dimensions
  - switching stylesheets or fonts

Tips to enhance performance

1. **Use Best-Practice Layout Techniques**
   - Don’t use inline styles or tables for layout!
   - An inline style will affect layout as the HTML is downloaded and trigger an additional reflow.
   - Tables are expensive because the parser requires more than one pass to calculate cell dimensions.
2. **Minimize the Number of CSS Rules**
   - The fewer rules you use, the quicker the reflow.
   - You should also avoid complex CSS selectors where possible.
   - Tools like Unused CSS or browser profiling can help you in identifying these unused CSS.
3. **Minimize DOM depths**
   - Reduce the size of your DOM tree and the number of elements in each branch.
   - The smaller and shallower your document, the quicker it can be reflowed.

Tips to enhance performance

4. **Remove Complex Animations From the Flow**
    - Ensure animations apply to a single element by removing them from the document flow with `position: absolute;` or `position: fixed;`.
    - This permits the dimensions and position to be modified without affecting other elements in the document.
5. **Modify Hidden Elements**
    - Elements hidden with `display: none;` will not cause a repaint or reflow when they are changed.
    - If practical, make changes to the element before making it visible.

Tips to enhance performance

6. **Update Elements in Batch**

   Performance can be improved by updating all DOM elements in a single operation.

   JS

```
let myelement = document.getElementById('myelement');

myelement.width = '100px';

myelement.height = '200px';

myelement.style.margin = '10px';
```

   We can reduce this to a single reflow which is also easier to maintain, e.g.

   JS

```
let myelement = document.getElementById('myelement');

myelement.classList.add('newstyles');
```

   CSS

```
.newstyles {
 height: 200px;
 margin: 10px;
 width: 100px;
}
```

Time for small Quiz

Updating Elements in Batch

How will you create the following bullet list with JS and then insert it in DOM?

- item 1
- item 2
- item 3

How many reflows will be caused?

INITECH

Adding each element one at a time causes up to seven reflows — one when the `<ul>` is appended, three for each `<li>` and three for the text. However, a single reflow can be implemented using a DOM fragment and building the nodes in memory first, e.g.

JS

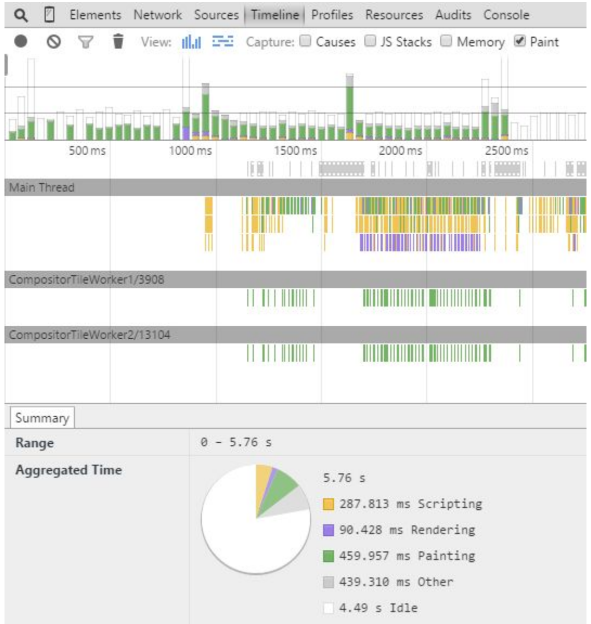
```
var frag = document.createDocumentFragment (),
 ul = frag.appendChild (document.createElement ('ul')),
 index, li;

for (index = 1; index <= 3; index++) {
 li = ul.appendChild (document.createElement ('li'));
 li.textContent = 'item ' + index;
}

document.body.appendChild (frag);
```

Tips to enhance performance

7. **Trade smoothness for speed**
    - By moving an animation one pixel at a time, animation and subsequent reflows use 100% of the CPU the animation will seem jumpy as the browser struggles to update the flow.
    - Moving the animated element by four pixels at a time may seem slightly less smooth on very fast machines, but it won't cause CPU thrashing on slower machines and mobile devices.

Tips to enhance performance

8. **Analyze Repaint Issues with Browser Tools**
   - All mainstream browsers provide developer tools that highlight how reflows affect performance.
   - In Blink/Webkit browsers such as Chrome, Safari, and Opera, open the Timeline panel and record an activity:



Elements Network Sources Timeline Profiles Resources Audits Console
View:
Capture: Causes JS Stacks Memory Paint
500 ms
1000 ms
1500 ms
2000 ms
2500 ms
Main Thread
CompositorTileWorker1/3908
CompositorTileWorker2/13104
Summary

Range	0 – 5.76 s
Aggregated Time	5.76 s
	🟨 287.813 ms Scripting
	🟪 90.428 ms Rendering
	🟩 459.957 ms Painting
	⬜ 439.310 ms Other
	⬜ 4.49 s Idle

BEM

BEM - naming convention

BEM stands for “Block”, “Element”, “Modifier”. It is a front-end methodology: a new way of thinking when developing Web interfaces.

Further Read: Official BEM Website

Block

A block is an independent entity, a “building block” of an application. A block can be either simple or compound (containing other blocks).

Block names must be unique within a project to unequivocally designate which block is being described. Only instances of the same block can have the same names.

A block name follows the block-name scheme and defines a namespace for elements and modifiers.

Example

`menu`

`lang-switcher`

HTML

```
<div class="menu">...</div>
```

CSS

```
.menu { color: red; }
```

Element

An element is a part of a block that performs a certain function. Elements are context-dependent: they only make sense in the context of the block that they belong to.

An element name is delimited by a double underscore (__).

Element names must be unique within the scope of a block. An element can be repeated several times.

Example

`menu__item`

`lang-switcher__lang-icon`

HTML

```
<div class="menu">

        ...

        <span class="menu__item"></span>

</div>
```

CSS

```
.menu__item { color: red; }
```

Modifier

A modifier defines a different state or version of a block or an element.

A modifier name is delimited by double hyphens (--).

Example

`menu--hidden`

HTML

```
<div class="menu menu--hidden">...</div>
```

Incorrect notation

```
<div class="menu--hidden">...</div>
```

Here the notation is missing the block that is affected by the modifier.

CSS

```
.menu--hidden {
  @include hidden; /* Using Bootstrap */
}
```

BEM - naming convention

The naming convention follows this pattern:

```css
.block {}
.block__element {}
.block--modifier {}
```

.block represents the higher level of an abstraction or component.

.block__element represents a descendent of .block that helps form .block as a whole.

.block--modifier represents a different state or version of .block.

BEM - naming convention

The reason for double rather than single hyphens and underscores is so that your block itself can be hyphen delimited, for example:

```css
.site-search {} /* Block */
.site-search__field {} /* Element */
.site-search--full {} /* Modifier */
```

A comparison of using and not using BEM

```html
<form class="site-search full">
    <input type="text" class="field">
    <input type="Submit" value ="Search" class="button">
</form>
```

```html
<form class="site-search site-search--full">
    <input type="text" class="site-search__field">
    <input type="Submit" value ="Search" class="site-search__button">
</form>
```

Style Guide

CSS Style guide

Link to CSS Style Guide

- Naming Conventions and Methodologies
- Syntax and Formatting
- Comments
- ID Selectors
- Shorthand Notation
- Pixels vs rem
- Color Units
- Border
- Quotes
- !important
- Vertical Margins
- Calculations

Sass Style guide

Link to Sass Style Guide

- Syntax
- Internal Order
- Ordering of property declarations
- Variables
- Mixins
- Placeholders

Useful links

Chrome Extension to Install to keep yourself daily updated: Panda 5

Websites to follow

- Codrops

Other

- Awesome Github Library
- Writing Scalable Code

Thank You! :)