META+LAB PRESENTS:

Clearing up Confusion on CSS

By Andrew Grano

CSS is deceiving.

It seems so simple.

It’s not even a programming language.

I just want to change the background color of this alert box.

It’s not working.

Why isn’t it working?

CSS can be a source of frustration and confusion for many web developers because it is *deceivingly* simple.

If you plan on continuing to pursue web development there are a couple of things you can do to approach learning CSS:

1: Remove mental roadblocks
CSS is not going away - it's the only way to style a website or web app! No more thinking negatively about it! CSS is *awesome*!

2: Learn the fundamentals of CSS
So that you understand WHY that red background isn't getting applied.

I'm going to try to help you with #2 right now, by clearing up confusion on...

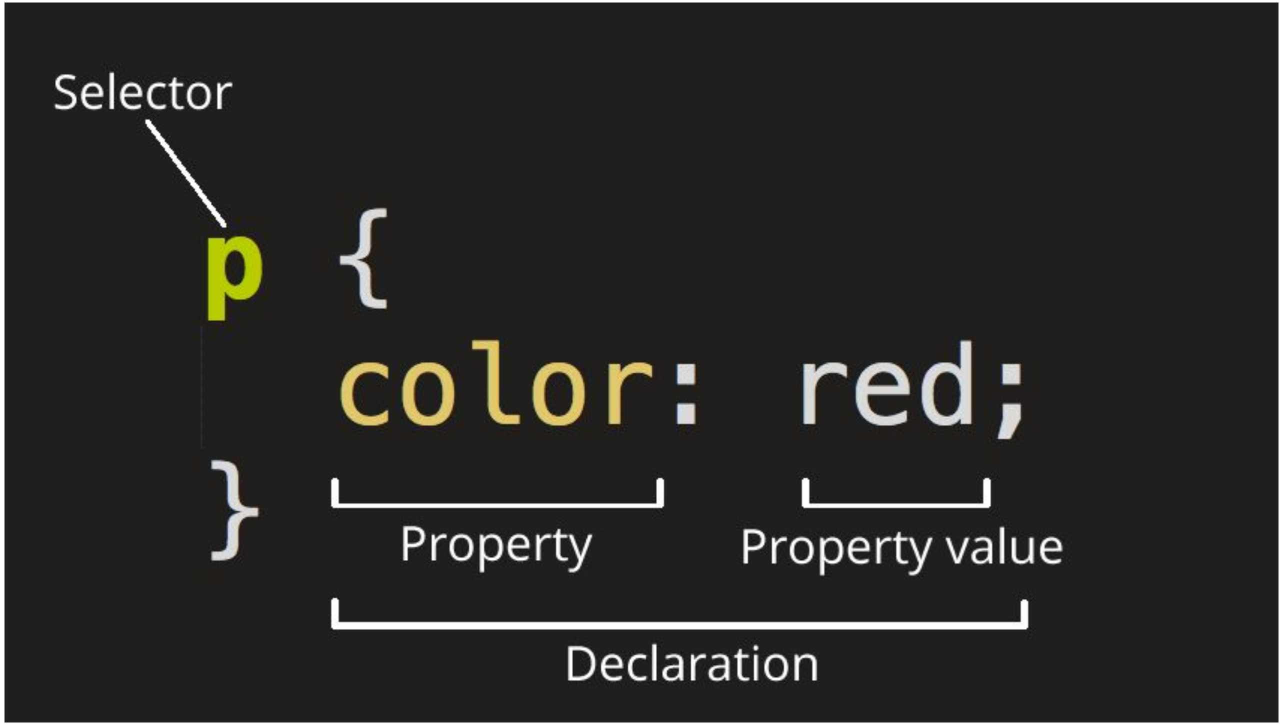
META
+LAB

Clearing Up Confusion on...

1. Selectors
2. The Cascade
3. Shorthand Properties
4. The Box Model

But first, let’s take a look at the Anatomy of a CSS rule set, so that we’re all on the same page with terminology:

Anatomy of a CSS rule set



```
p {
  color: red;
}
```

Questions?

META
+LAB

Clearing up Confusion on CSS

SELECTORS

Different Types of Selectors

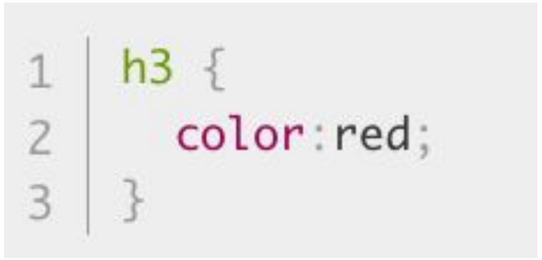
There are many different types of selectors which we can utilize, giving us different ways to **target** specific elements on the page.

Here are some of the common types of selectors:

Element selector:

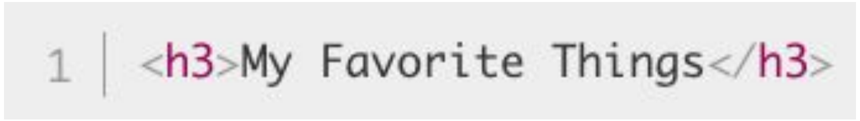
Selects all HTML element(s) of the specified type.

CSS Selector:



```
h3 {
  color:red;
}
```

Selects:



```
<h3>My Favorite Things</h3>
```

ID selector:

Selects the element on the page with the specified ID.
(On an HTML page, any given ID can only appear once)

CSS Selector:

```css
#favorite-movies {
   color: red;
}
```

Selects:

```html
<div id="favorite-movies">Some content...</div>
```

Class selector:

Selects the element(s) on the page with the specified class
(On an HTML page, multiple elements may share the same class)

CSS Selector:

```css
.big-text {
  font-size:48px;
}
```

Selects:

```html
<a class="big-text">Click here</a>
```

```html
<h3 class="big-text">My Favorites</h3>
```

```html
<p class="big-text">Some content...</p>
```

Attribute selector:

Selects the element(s) on the page with the specified attribute.

CSS Selector:

```css
img[src]{
  color:red;
}
```

Selects:

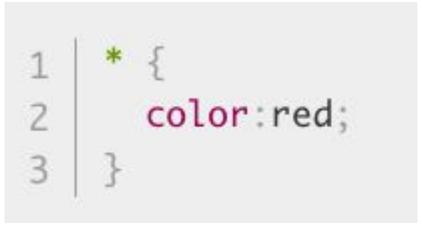
```html
<img src="images/car.jpg">
```

Does not Select:

```html
<img>
```

Universal selector:

CSS Selector:



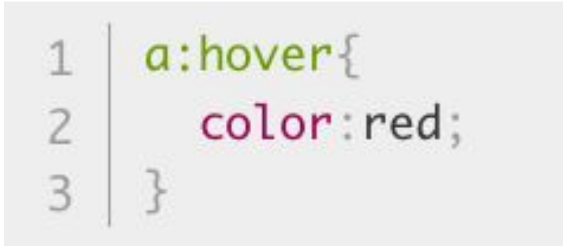
```
* {
  color:red;
}
```

Selects: All elements on the page!

Pseudo-class selectors:

A **CSS pseudo-class selector** is a keyword added to the end of a selector, preceded by a colon, which is used to specify that you want to style the selected element but **only when it is in a certain state.**

CSS Selector:



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

Selects: An <a> tag **only** when the mouse pointer is hovering over the link.

Pseudo-class selectors

There are tons more pseudo-class selectors, as well as some **pseudo-element selectors** which target a specific part of an element as opposed to an element in a specific state.

But we aint got time for that right now!

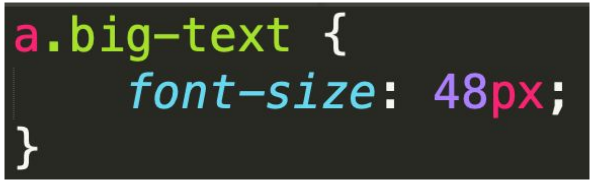
In your free time, check out the full list of psuedo-class selectors and psuedo-element selectors from the MDN Web Docs:

https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes#Index_of_standard_pseudo-classes

https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-elements#Index_of_standard_pseudo-elements

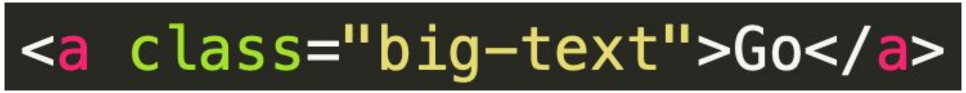
More on the CSS class selector...

We can make more specific selections based on an element’s class as well!

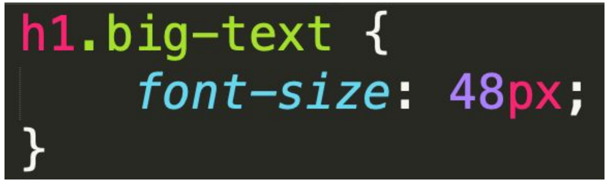


```css
a.big-text {
    font-size: 48px;
}
```

This example will target only **`<a>`** tags that have a class of “big text”

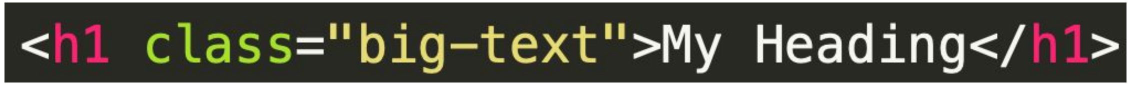


```html
<a class="big-text">Go</a>
```



```css
h1.big-text {
    font-size: 48px;
}
```

While this example will target only **`<h1>`** tags that have a class of “big text”



```html
<h1 class="big-text">My Heading</h1>
```

More on the CSS class selector...

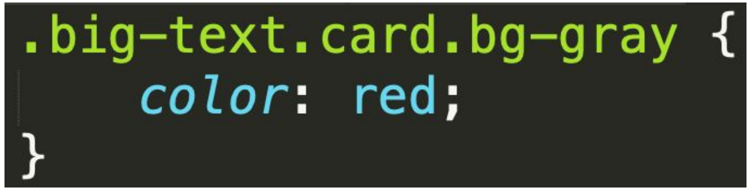
```
.big-text.card {
    color: red;
}
```

This example will target only elements that have a class of “big-text” **and** a class of “card”. Such as:

```
<span class="big-text card">hello there</span>
```

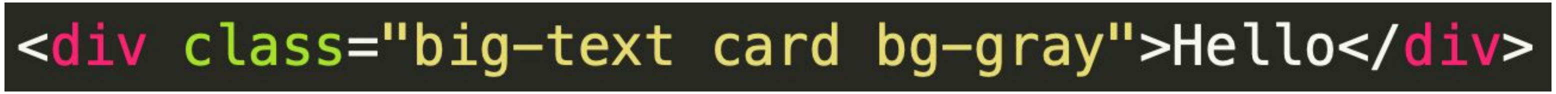
```
<h1 class="big-text card pageHeading">My Heading</h1>
```

More on the CSS class selector...



```css
.big-text.card.bg-gray {
    color: red;
}
```

This example will target only elements that have a class of “big-text” **and** a class of “card” **and** a class of “bg-gray”



```html
<div class="big-text card bg-gray">Hello</div>
```

You could continue to chain class selectors together in this fashion.

More on the CSS class selector...

```css
a.big-text.card {
    color: red;
}
```

Finally, This example will target only **`<a>`** tags that have a class of “big-text” **and** a class of “card”

```html
<a class="big-text card">Go</a>
```

CSS Selectors: Combinators

Combinators allow you to select elements based on how they are related to one another, in order to perform more fine-grained selections.

META
+LAB

CSS Selectors: Combinators

These are the four combinators:

```
A B {
	color: red;
}
```

```
A > B {
	color: red;
}
```

```
A + B {
	color: red;
}
```

```
A ~ B {
	color: red;
}
```

A and **B** as a "placeholders" for any simple-selector

Element selector
- h1
- p
- strong
- div

ID selector
- #favorite-movies
- #mainSection
- #menu
- #andrew

Class selector
- .big-text
- div.card
- .bg-gray.red
- .andrew

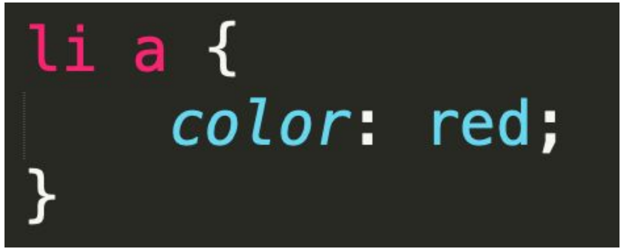
Attribute selector
- img[src]
- a[href]
- a[href="https://google.com"]
- [alt]

Descendant Combinator

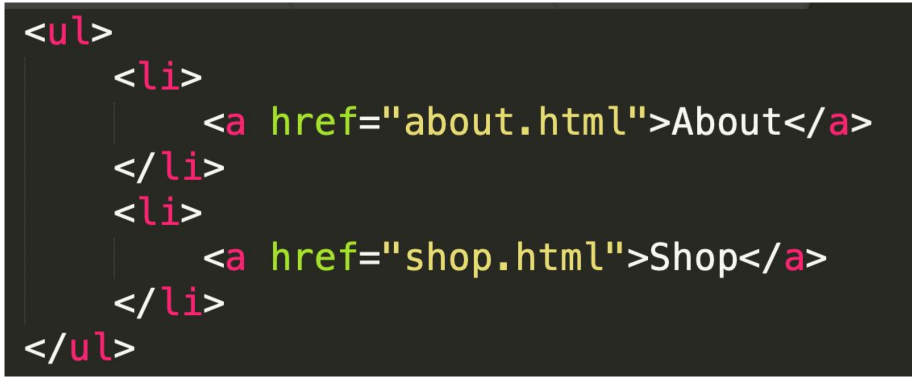
Syntax: A B

Selects: Any element matching B that is a **descendant** of an element matching A

Example:



```css
li a {
    color: red;
}
```



```html
<ul>
    <li>
        <a href="about.html">About</a>
    </li>
    <li>
        <a href="shop.html">Shop</a>
    </li>
</ul>
```

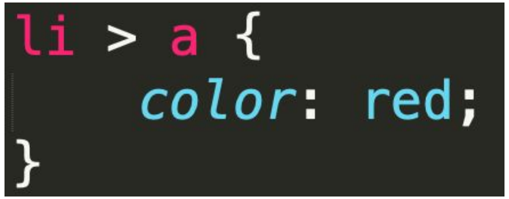
This CSS code will apply red text to any **<a>** element that is inside an **<li>** element

Child Combinator

Syntax: A > B

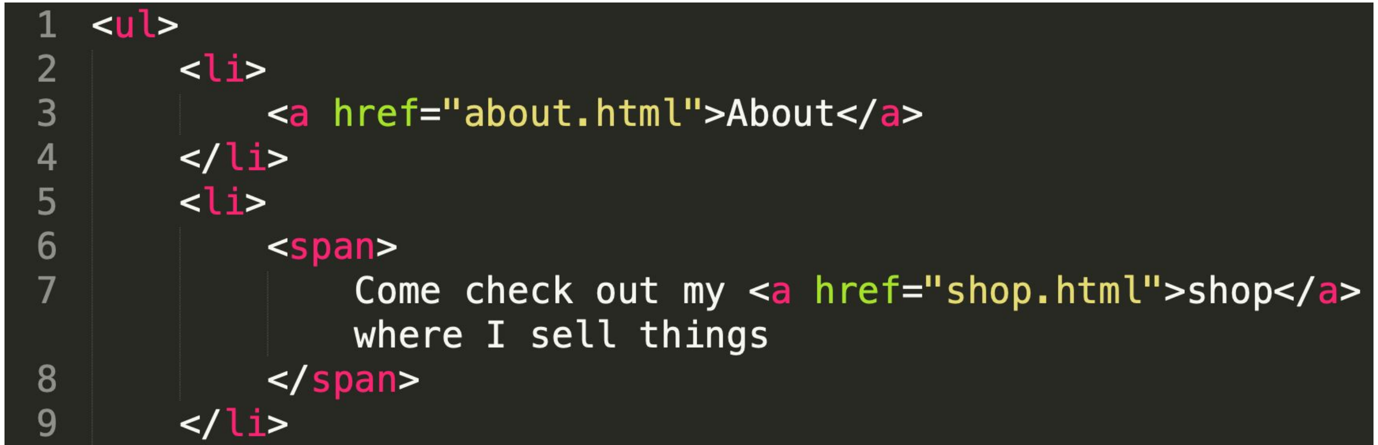
Selects: Any element matching B that is a **direct child** of an element matching A.

Example:



```css
li > a {
    color: red;
}
```

This CSS code will apply red text to any **<a>** element that is directly inside an **<li>** element...



```html
1 <ul>
2     <li>
3         <a href="about.html">About</a>
4     </li>
5     <li>
6         <span>
7             Come check out my <a href="shop.html">shop</a>
              where I sell things
8         </span>
9     </li>
```

...such as the <a> element on on line 3,

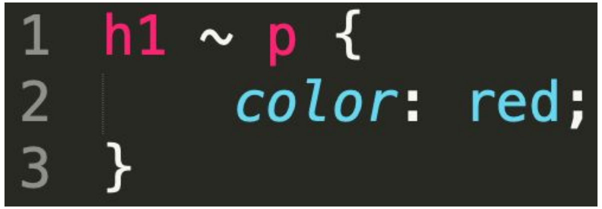
but **not** the <a> element on line 7

General Sibling Combinator

Syntax: A ~ B

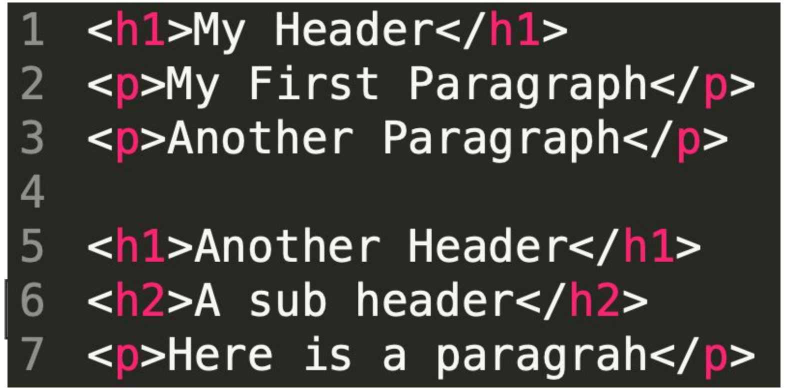
Selects: Any element matching B that is **one of the next siblings** of an element matching A

Example:



```css
h1 ~ p {
    color: red;
}
```

This CSS code will apply red text to any **<p>** tag that follows an **<h1>** tag...



```html
<h1>My Header</h1>
<p>My First Paragraph</p>
<p>Another Paragraph</p>

<h1>Another Header</h1>
<h2>A sub header</h2>
<p>Here is a paragrah</p>
```

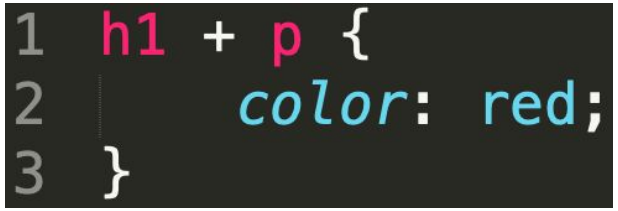
...In this example, all of the <p> tags will have red text applied.

Adjacent Sibling Combinator

Syntax: A + B

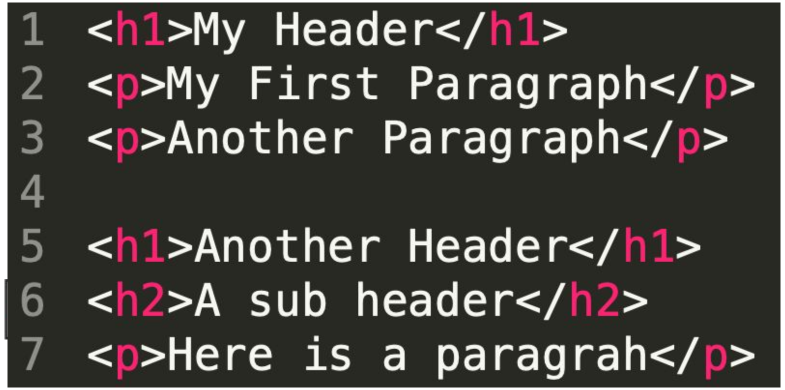
Selects: Any element matching B that is the **next sibling** of an element matching A

Example:



```css
h1 + p {
    color: red;
}
```

This CSS code will apply red text to **only** the first **<p>** tag after each **<h1>** tag...



```html
1  <h1>My Header</h1>
2  <p>My First Paragraph</p>
3  <p>Another Paragraph</p>
4
5  <h1>Another Header</h1>
6  <h2>A sub header</h2>
7  <p>Here is a paragrah</p>
```

...In this example, only the <p> tag on line 2 will have red text applied.

Combining Combinators

You can chain together multiple combinators in a single ruleset in order to make an even more specific selection. For example, you can:

```
.card p strong {
```

Use the descendant combinator twice to select all **<strong>** tags that are descendants of **<p>** tags that are descendants of **any element with a class of "card"**

```
li > a .big-text {
```

Use the descendant combinator and the Child Combinator to select all **elements with a class of "big-text"** that are descendants of **<a>** tags that are direct descendants of **<li>** tags

```
.mainContent > .card h1 ~ p {
```

Use a bunch of combinators to make a very specific selection

Questions?

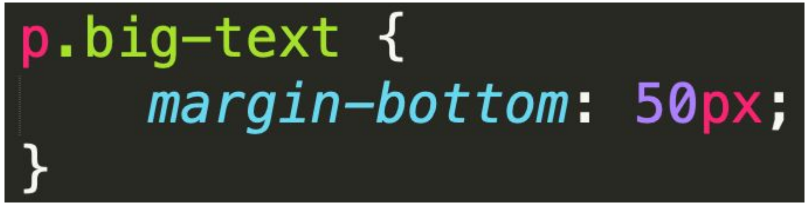
Clearing up Confusion on CSS

THE CASCADE

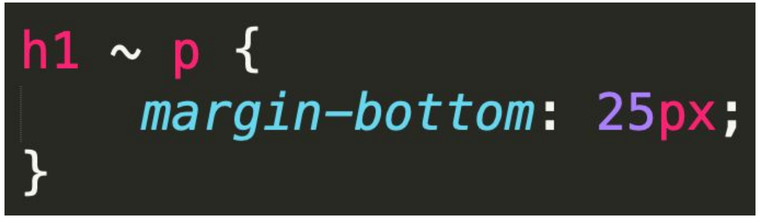
META
LAB

CSS Cascade

We now know a lot of different ways to select elements on the page.
At some point, we'll find ourselves in a situation where two of our CSS rules target the same element.



```
p.big-text {
    margin-bottom: 50px;
}
```



```
h1 ~ p {
    margin-bottom: 25px;
}
```

In such cases, which CSS rule "wins" and ends up being applied to the element?

This answer is determined by a mechanism called the **Cascade**;
(Remember that CSS stands for **Cascading** Style Sheet)

META
+LAB

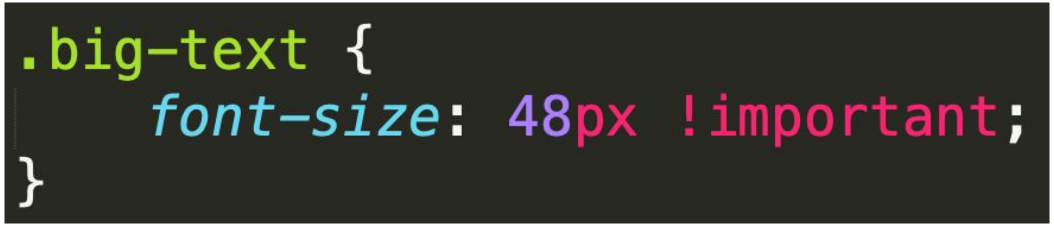
CSS Cascade

Which selectors win out in the cascade depends on three factors:

1. Importance ⟵ Carries the **most** weight
2. Specificity
3. Source Order ⟵ Carries the **least** weight

CSS Cascade - Importance

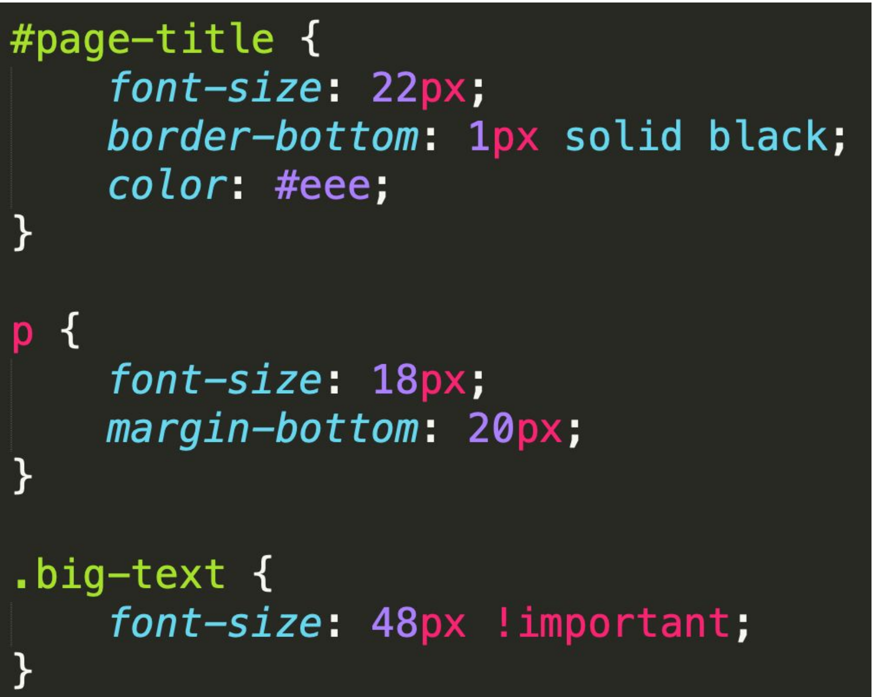
In CSS, there is a special piece of syntax that you can add to the end of any declaration to make it **always** win over others: `!important`



```
.big-text {
    font-size: 48px !important;
}
```

CSS Cascade - Importance

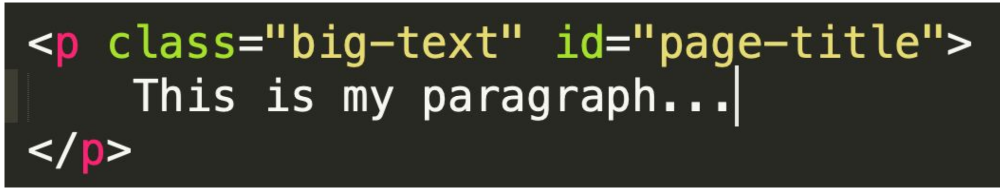
In this example, any element with a class of “big-text” will always have a font-size of 48px, even if there are other rulesets that compete with it:



```css
#page-title {
    font-size: 22px;
    border-bottom: 1px solid black;
    color: #eee;
}

p {
    font-size: 18px;
    margin-bottom: 20px;
}

.big-text {
    font-size: 48px !important;
}
```



```html
<p class="big-text" id="page-title">
    This is my paragraph...
</p>
```

This is because the `!important` tag carries the most weight in the cascade.

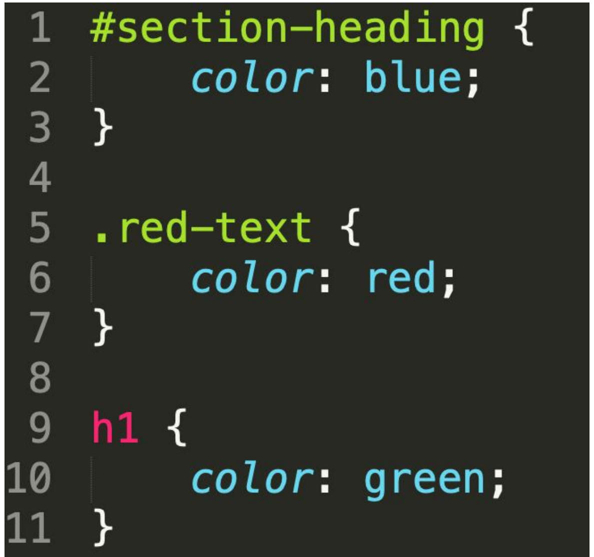
CSS Cascade - Importance

It is useful to know that `!important` exists, however I strongly recommend that you never use it unless you absolutely have to, as it will make your CSS much harder to scale and maintain.

CSS Cascade - Specificity

Specificity is basically a measure of how specific a selector is — **how many elements it could match**.

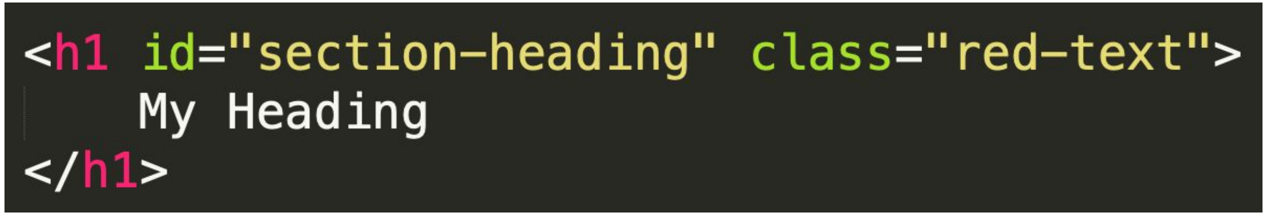
- **Element selectors** have low specificity (they could match many elements).
- **Class selectors** have a higher specificity, so will win against element selectors.
- **ID selectors** have an even higher specificity, so will win against class selectors.
    - The only way to win against an **ID selector** is to use !important.



```css
#section-heading {
    color: blue;
}

.red-text {
    color: red;
}

h1 {
    color: green;
}
```



```html
<h1 id="section-heading" class="red-text">
    My Heading
</h1>
```

What color will this element have applied to it?

CSS Cascade - Specificity

In general, it is considered good practice to **avoid using id selectors** because they will often lead to issues with scalability and maintainability due to their increased specificity.

(It is difficult to override any of the properties that are declared within an ID selector. So we can avoid that issue by just using **class selectors** instead).

CSS Cascade - Specificity

When using combinators, things get a little more complicated, because your combinators might mix **element selectors** with **class selectors** and/or **id selectors**.

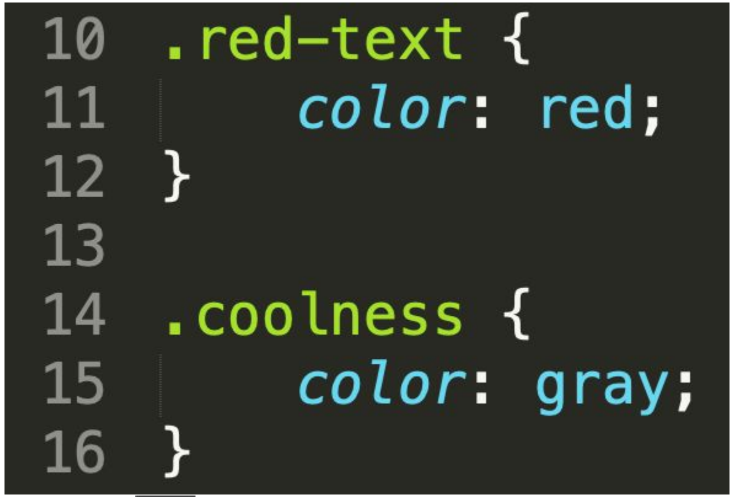
In these cases, the browser will look at how many of each type of selector is used in the combinator and assign each selector a certain amount of points:

- Each ID selector gets **100 points**
- Each class selector, attribute selector or pseudo-class selector gets **10 points**
- Each element selector or pseudo-element selector gets **1 point**

The browser adds up all of the points in the combinator to determine that combinator's "**specificity score**". A combinator with a higher specificity score will win out over a combinator or simple selector with a lower specificity score.

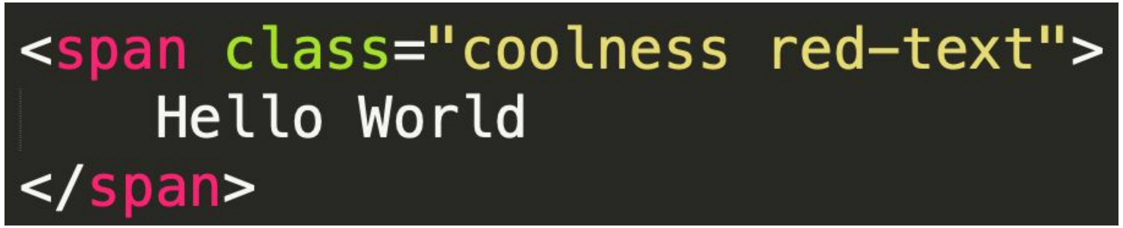
CSS Cascade - Source Order

If multiple competing selectors have the same **importance** and **specificity score**, the third factor that comes into play to help decide which rule wins is **source order** — rulessets at the bottom of the stylesheet will win over rulessets higher up in the stylesheet.



```css
.red-text {
    color: red;
}

.coolness {
    color: gray;
}
```



```html
<span class="coolness red-text">
    Hello World
</span>
```

What color will this element have applied to it?

CSS Inheritance

CSS Inheritance is the last piece of the puzzle when investigating what styles actually get applied to an element.

Some properties, like **font** and **color** are inherited by its children.
(Meaning that the **color** or **font** that you set on an element will **also be applied to that element's children**).

But not all properties are inherited...

META
+LAB

CSS Inheritance

Which properties **are** inherited and which **aren't** is largely down to common sense.

For example, it makes sense for **color** to be inherited, as it allows you to set a site-wide “default”. You can then override the color on individual elements where needed. It would be really annoying to have to set the base color separately on every element.

As another example, it makes sense for **border** to NOT be inherited. It would be really weird if you set a border for one element, and then every element inside that element also had a border!

So, be aware that any given element may have some properties applied to it that were never **explicitly set**, but instead were **inherited**.

Quick Specificity Test:

https://codepen.io/andrewMETALAB/pen/OqNLKw?editors=1100

Questions?

Clearing up Confusion on CSS

SHORTHAND PROPERTIES

CSS Font Properties

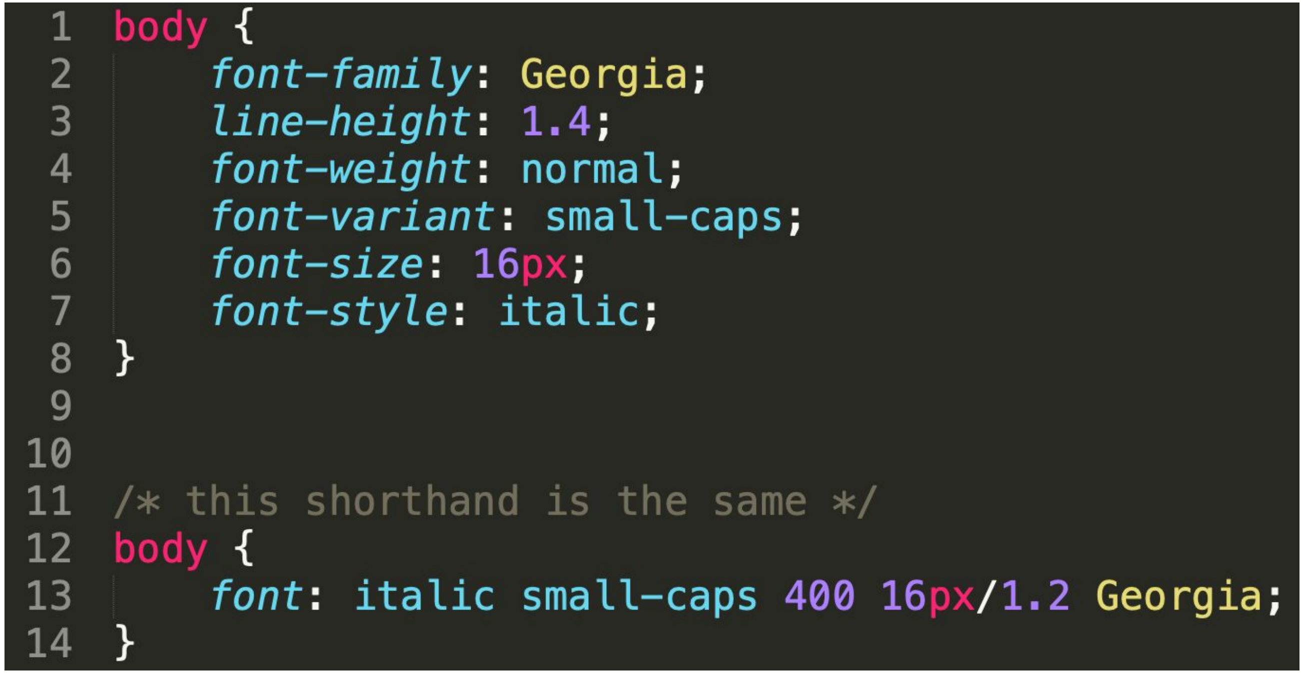
Ya'll have probably encountered some of these CSS font properties before:

- font-size
    - Sets the size of your text
- font-weight
    - Makes text bold or not bold or kinda bold
- line-height
    - Sets the amount of space between lines of text
- font-style
    - Makes text italicized or not
- font-variant
    - (less commonly used), makes lower-case letters into small capitalized letters

Did you know that there is also a CSS property called **font**

The **font** property combines **font-size**, **line-height**, **font-weight**, **font-style**, **font-family**, and **font-variant** all into one single declaration.

CSS Font Properties



```css
body {
	font-family: Georgia;
	line-height: 1.4;
	font-weight: normal;
	font-variant: small-caps;
	font-size: 16px;
	font-style: italic;
}


/* this shorthand is the same */
body {
	font: italic small-caps 400 16px/1.2 Georgia;
}
```

Example:
https://developer.mozilla.org/en-US/docs/Web/CSS/font#live_sample

CSS Font Properties

The **font** property is what’s known a **shorthand property** because it combines several CSS properties into one single declaration.

There are many other **shorthand properties** in CSS.

Can you think of any other shorthand properties?

We’ll come back to it.

Questions?

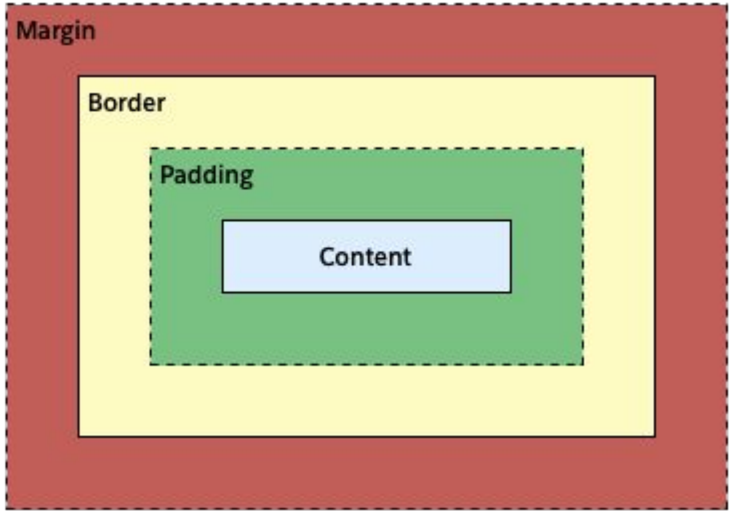
Clearing up Confusion on CSS

THE BOX MODEL

META
+LAB

Box Model

Every element on your webpage is a box.
When a browser loads a webpage, the browser renders every element on the page according to the **CSS box model** to determine its size, position, and properties.

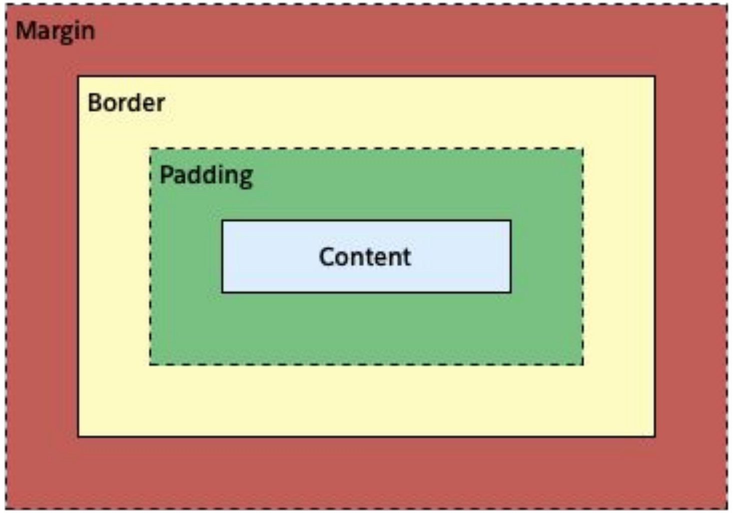


Every box is composed of four areas: the **content** area, **padding** area, **border** area, and **margin** area.

We can control the four areas of any element with various CSS properties.

META
+LAB

Box Model



The **content area** contains the "real" content of the element, such as text or an image.

The **padding area** enlarges the content area. If there is a background color applied to the element, then the background color will be applied to the padding area as well.

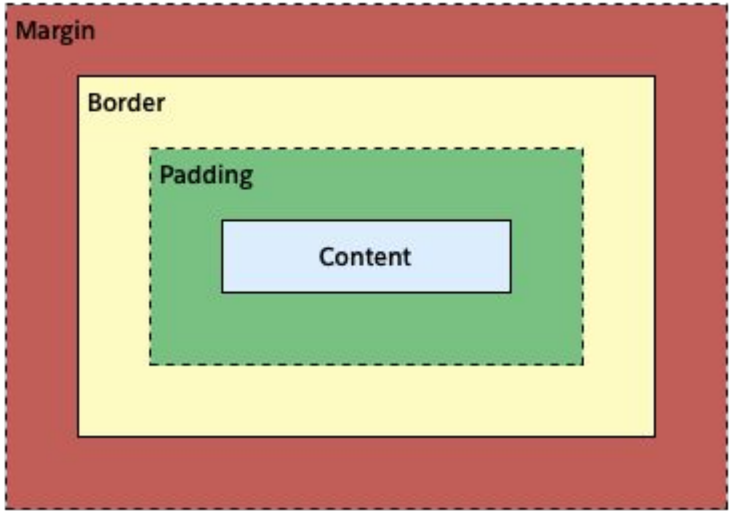
The **border area** applies a border around the edge of the padding area.

The **margin area** extends the element to include an empty area used to separate the element from its neighbors.

Box Model

The **margin area** is controlled by the following CSS properties:

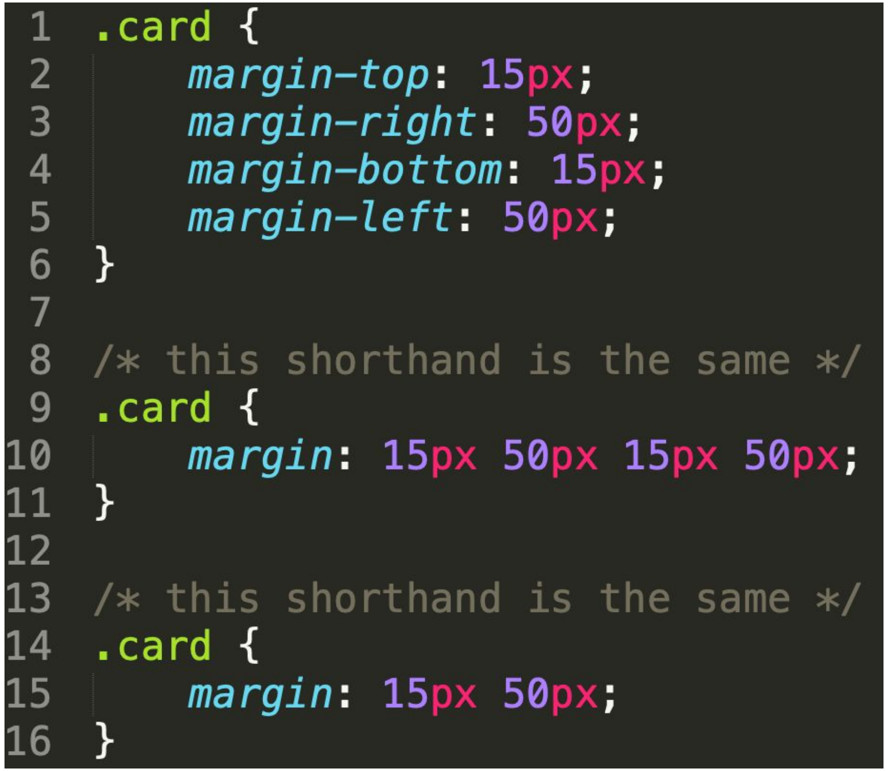
```
margin-top: [value]
margin-right: [value]
margin-bottom: [value]
margin-left: [value]
```



The shorthand for these properties is

```
margin: [top-value] [right-value] [bottom-value] [left-value]
```
or
```
margin: [top-value & bottom-value] [left-value and right-value]
```
or
```
margin: [top-value & bottom-value & left-value & right-value]
```

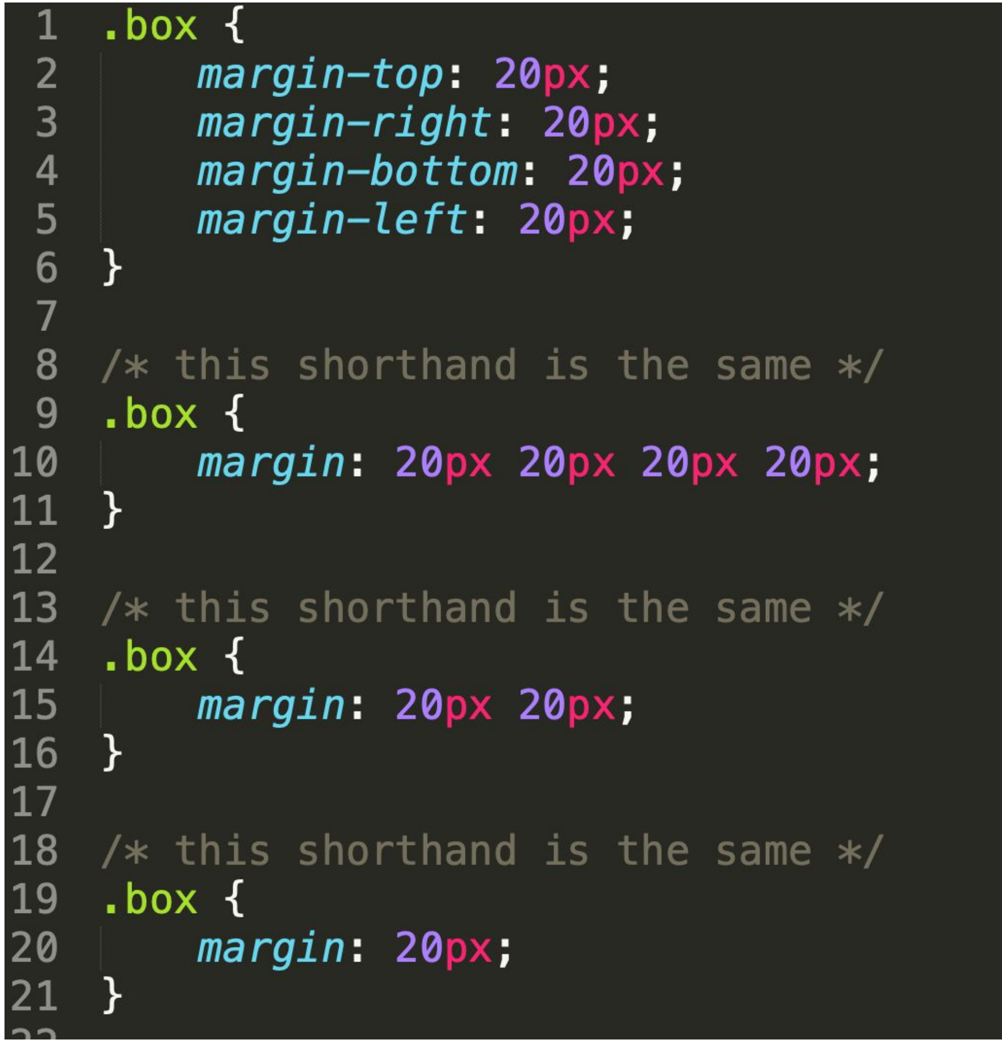
META
+LAB



```css
.card {
    margin-top: 15px;
    margin-right: 50px;
    margin-bottom: 15px;
    margin-left: 50px;
}

/* this shorthand is the same */
.card {
    margin: 15px 50px 15px 50px;
}

/* this shorthand is the same */
.card {
    margin: 15px 50px;
}
```



```css
.box {
    margin-top: 20px;
    margin-right: 20px;
    margin-bottom: 20px;
    margin-left: 20px;
}

/* this shorthand is the same */
.box {
    margin: 20px 20px 20px 20px;
}

/* this shorthand is the same */
.box {
    margin: 20px 20px;
}

/* this shorthand is the same */
.box {
    margin: 20px;
}
```

Box Model

The **padding area** is controlled by the following CSS properties:

```
padding-top: [value]
padding-right: [value]
padding-bottom: [value]
padding-left: [value]
```

The shorthand for these properties is

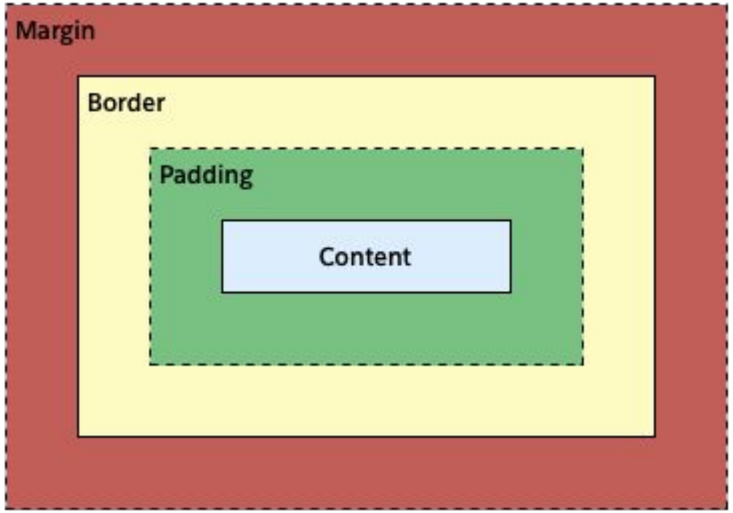
```
padding: [top-value] [right-value] [bottom-value] [left-value]
```

or

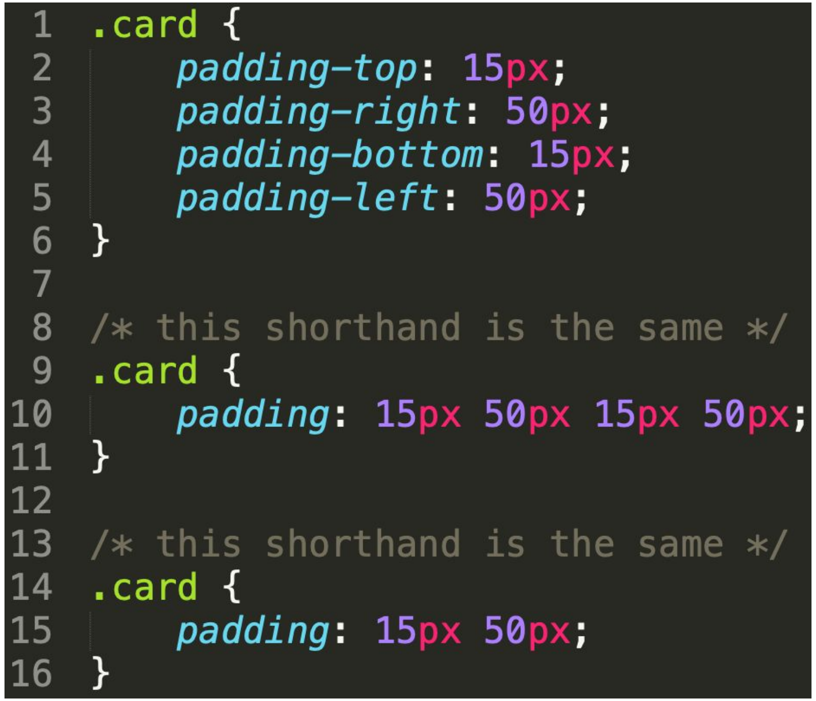
```
padding: [top-value & bottom-value] [left-value and right-value]
```

or

```
padding: [top-value & bottom-value & left-value & right-value]
```



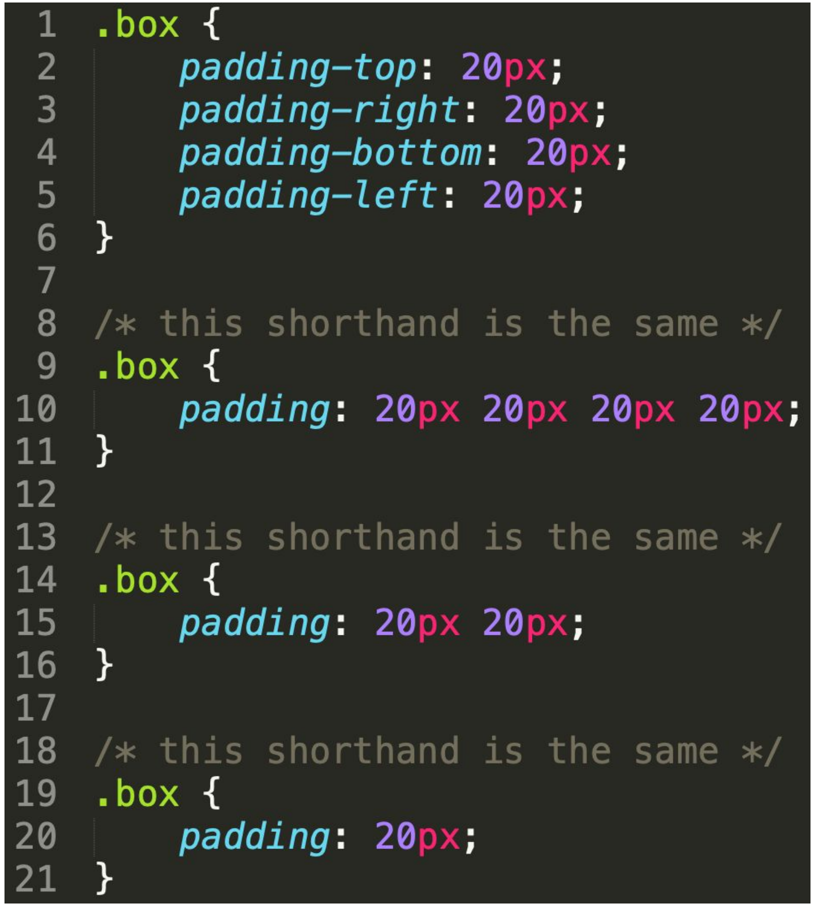
META +LAB



```css
.card {
    padding-top: 15px;
    padding-right: 50px;
    padding-bottom: 15px;
    padding-left: 50px;
}

/* this shorthand is the same */
.card {
    padding: 15px 50px 15px 50px;
}

/* this shorthand is the same */
.card {
    padding: 15px 50px;
}
```



```css
.box {
    padding-top: 20px;
    padding-right: 20px;
    padding-bottom: 20px;
    padding-left: 20px;
}

/* this shorthand is the same */
.box {
    padding: 20px 20px 20px 20px;
}

/* this shorthand is the same */
.box {
    padding: 20px 20px;
}

/* this shorthand is the same */
.box {
    padding: 20px;
}
```

META
+LAB

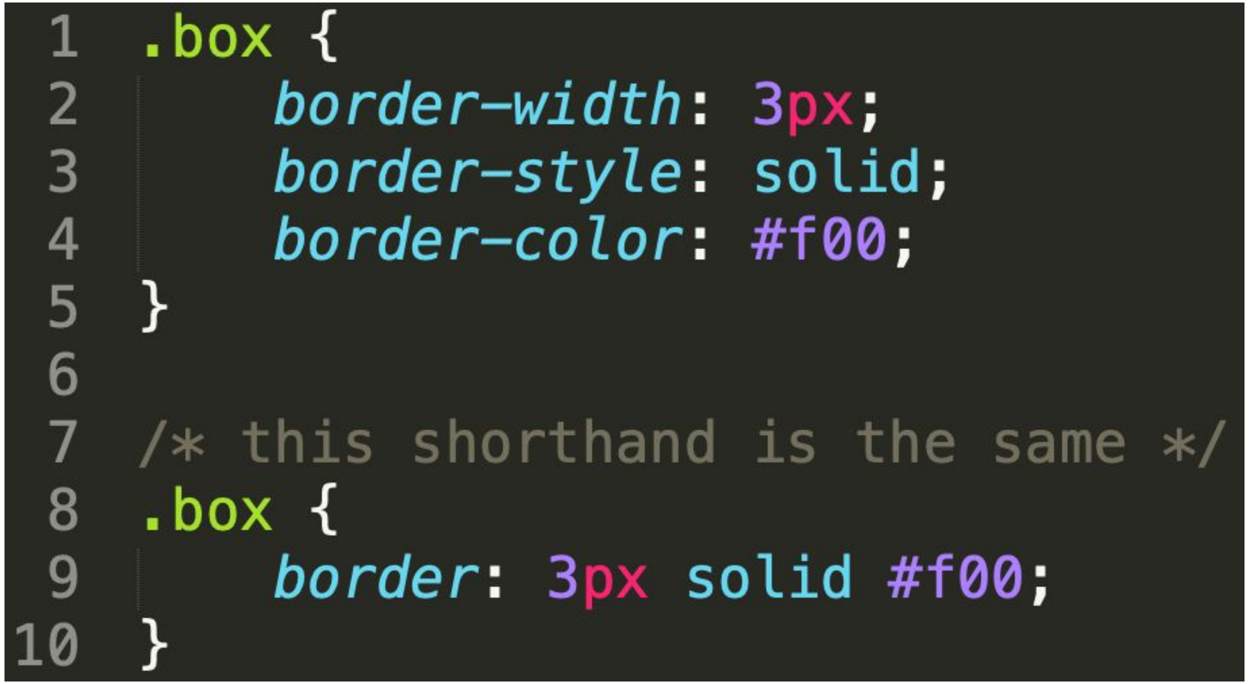
Box Model

The **border area** is controlled by the following CSS properties:

```
border-width:  [value]
border-style:  [dotted] [dashed] [double] [solid]
border-color:  [color value]
```

The shorthand for these properties is

```
border: [border-width] [border-style] [border-color]
```



```css
.box {
    border-width: 3px;
    border-style: solid;
    border-color: #f00;
}

/* this shorthand is the same */
.box {
    border: 3px solid #f00;
}
```

Box Model

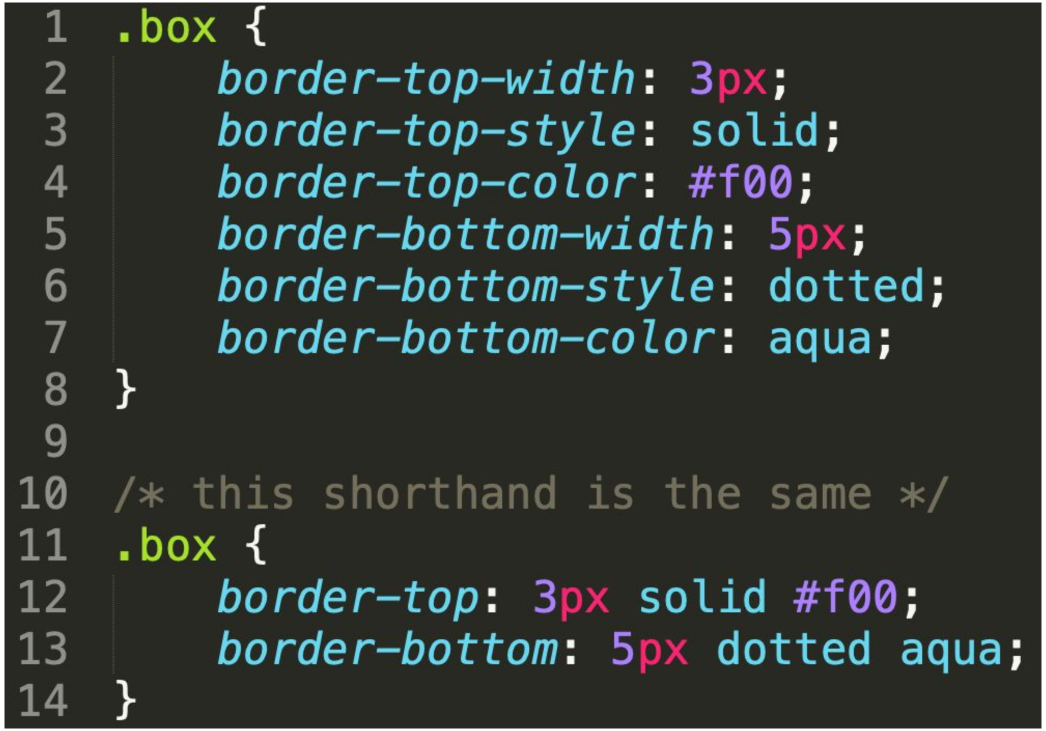
Note that the border property applies the values to all 4 sides of your box.
If you need to apply different styling to the different sides of the box, you can use:

```
border-top-width / border-right-width / border-bottom-width / border-left-width
border-top-style / border-right-style / border-bottom-style / border-left-style
Border-top-color / border-right-color / border-bottom-color / border-left-color
```

The shorthand for these properties is

```
border-top / border-right / border-bottom / border-left
```

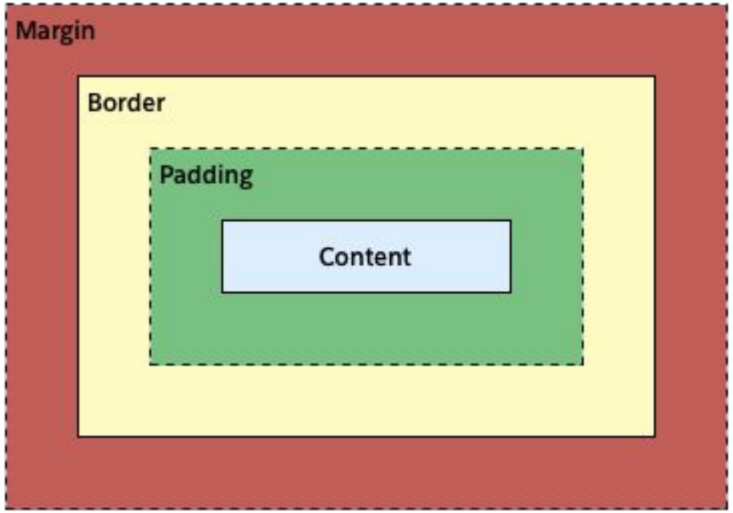
Box Model



```css
.box {
    border-top-width: 3px;
    border-top-style: solid;
    border-top-color: #f00;
    border-bottom-width: 5px;
    border-bottom-style: dotted;
    border-bottom-color: aqua;
}

/* this shorthand is the same */
.box {
    border-top: 3px solid #f00;
    border-bottom: 5px dotted aqua;
}
```

Box Model

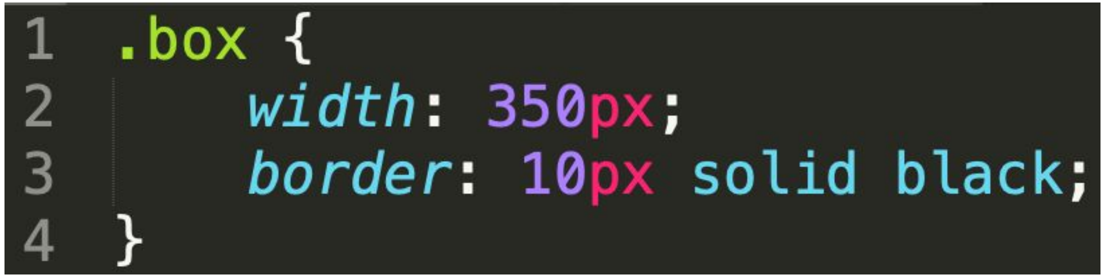


You can use the following CSS properties to change the width and the height of any element:

```
width: [value]
height: [value]
```

By default, when you set the width and height of an element, CSS applies the values to the **content box** but not the padding, border, or margin boxes.

For example, the following CSS would render a box that is 370px wide:



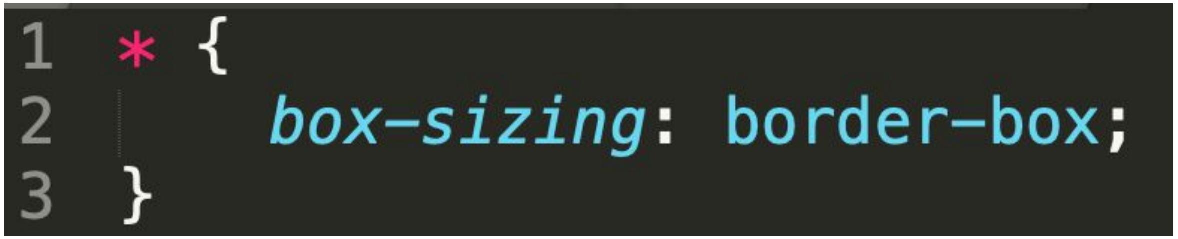
```
.box {
    width: 350px;
    border: 10px solid black;
}
```

META
+LAB

Box Model

Having the actual visible width of a box turn out differently from what you declared using the CSS width property is a bit odd.

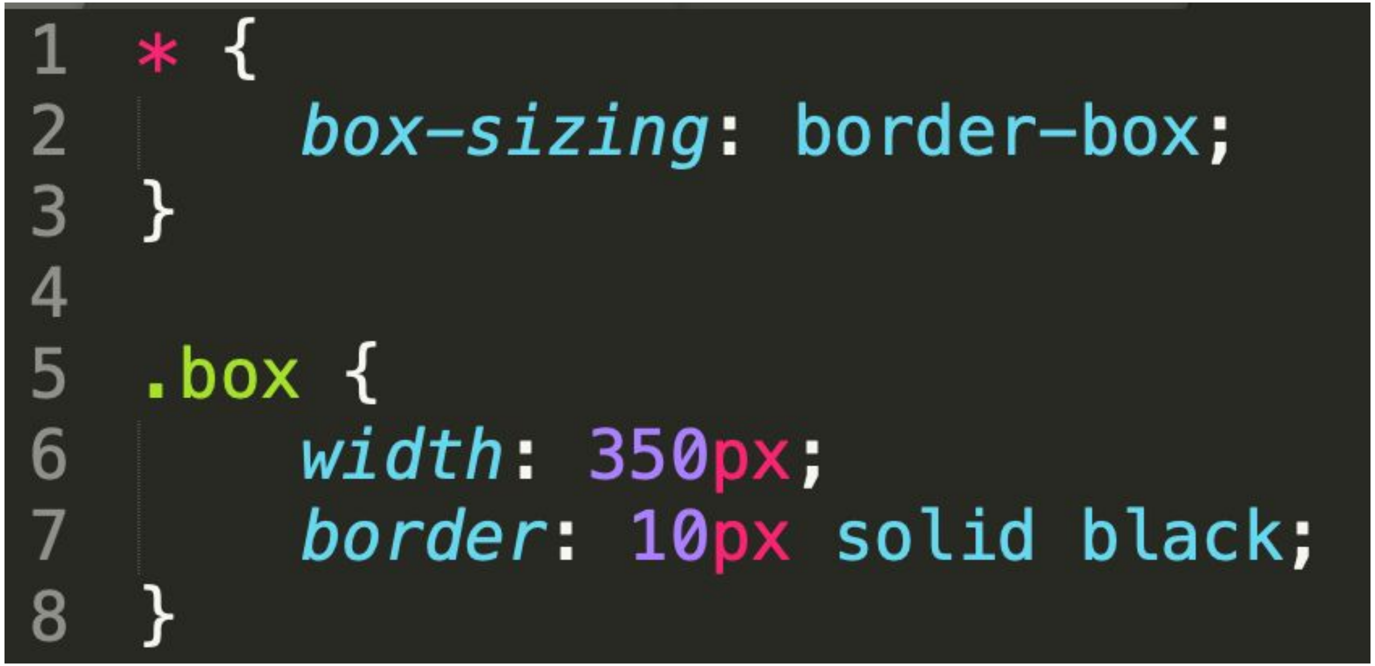
Luckily, there is a CSS property called **box-sizing** which controls how CSS will make these calculations. The **box-sizing** property defaults to a value of **content-box**, but it also accepts a value of **border-box**. We should apply this declaration to all elements on our page so that the width and height calculations are more intuitive.



```
* {
    box-sizing: border-box;
}
```

Box Model

Now, the the following CSS would render a box that is 350px wide:



```
* {
    box-sizing: border-box;
}

.box {
    width: 350px;
    border: 10px solid black;
}
```

META
+LAB

Questions?

META
+LAB

THAT’S A WRAP FOLKS,

THANK YOU!

META
+LAB