CHALLENGE FRONTEND DEVELOPER BERTELSMANN & UDACITY

**SUMÁRIO**

[1 HTML 4](#_Toc124607734)

[1.1 Text Elements I 4](#_Toc124607735)

[1.1.1 Text Element Structures 4](#_Toc124607736)

[1.2 Text Elements II 5](#_Toc124607737)

[1.2.1 Headings 5](#_Toc124607738)

[Observable Universe 5](#_Toc124607739)

[Milky Way Galaxy 5](#_Toc124607740)

[Earth 5](#_Toc124607741)

[USA 5](#_Toc124607742)

[Norfolk, VA 5](#_Toc124607743)

[Main Street 5](#_Toc124607744)

[1.2.1.1 Accessibility with Headers 6](#_Toc124607745)

[1.2.2 Paragraphs 6](#_Toc124607746)

[1.2.3 Spans 6](#_Toc124607747)

[1.2.4 Blockquotes 7](#_Toc124607748)

[1.2.5 Line Breaks 7](#_Toc124607749)

[1.3 HTML Lists 8](#_Toc124607750)

[1.3.1 Unordered Lists 8](#_Toc124607751)

[1.3.2 Ordered Lists 8](#_Toc124607752)

[1.4 The DOM 9](#_Toc124607753)

[1.4.1 The Head 9](#_Toc124607754)

[1.4.2 Metadata 9](#_Toc124607755)

[1.4.3 Title 10](#_Toc124607756)

[1.4.4 The Body 10](#_Toc124607757)

[1.4.5 Hierarchy 10](#_Toc124607758)

[1.4.6 Semantic Elements 11](#_Toc124607759)

[1.5 Forms 12](#_Toc124607760)

[1.5.1 Input Types, Select and Textarea 13](#_Toc124607761)

[1.5.2 More on forms 14](#_Toc124607762)

[2 CSS 15](#_Toc124607763)

[2.1 The Box Model 15](#_Toc124607764)

[2.2 Problems in set Height 16](#_Toc124607765)

[2.3 Inheritance 17](#_Toc124607766)

[2.4 Values em and rem explained 17](#_Toc124607767)

[2.5 The Cascade 18](#_Toc124607768)

[2.6 Origin and Importance 19](#_Toc124607769)

[2.7 Origin and Importance when animations & transitions come to play 20](#_Toc124607770)

[2.8 Specificity 20](#_Toc124607771)

[2.9 CSS Selectors 21](#_Toc124607772)

[2.9.1 Tags 21](#_Toc124607773)

[2.9.2 Classes 22](#_Toc124607774)

[2.9.3 Ids 22](#_Toc124607775)

[2.9.4 Pseudo-classes 22](#_Toc124607776)

[2.9.5 Attributes 23](#_Toc124607777)

[2.9.6 Multiple Selectors 23](#_Toc124607778)

[2.10 Linking CSS 24](#_Toc124607779)

[2.10.1 linking a CSS file to another CSS file 24](#_Toc124607780)

[2.10.2 Specificity 25](#_Toc124607781)

[2.11 Absolute vs Relative Units 27](#_Toc124607782)

[3 CSS FLEXBOX 29](#_Toc124607783)

# HTML

## Text Elements I

### Text Element Structures

An HTML element is a unit of content in an HTML document that is formed using HTML tags.

The basic structure of an element is composed of 4 key items, one of which is optional.

<p class="dog-breed">Labrador Retriever</p>

1. The opening tag is the first HTML tag used to start an element.
2. The content is the info contained between the opening and closing tags. Only this content inside the opening and closing body tags is displayed to the screen.
3. The closing tag is the second tag used to define the end of a single element. Closings tags have a forward slash / inside of them, always after the left angle bracket.
4. (Optional) The attribute name and value.

Notice both tags are always surrounded by opening and closing angle brackets <>.

Note: developers use the terms "left bracket" and "opening bracket" interchangeably. SImilarly, you can use either "right bracket" or "closing bracket".

There are over 100 different types of HTML tags that each serve a specific use case.

In HTML, you’ll mainly come across 2 types of HTML elements.

Block elements are meant to structure the main parts of your page, by dividing your content in coherent blocks.

**Block elements** are:

* paragraphs <p>
* lists:
  + unordered (with bullet points) <ul>, or
  + ordered - lists (with numbers) <ol>
* headings: from 1st level <h1> to 6th level headings - <h6>
* articles <article>
* sections <section>
* long quotes <blockquote>

Inline elements are meant to differentiate part of a text, to give it a particular function or meaning. Inline elements usually comprise a single or few words.

**Inline elements** are:

* links <a>
* emphasized words <em>
* important words <strong>

## Text Elements II

### Headings

Headings in HTML are comparable to headings in other media types. In journals, for instance, big headings are typically used to catch the attention of a reader. Other times, headings are used to define material, such as a film's title or an instructional article.

Headings are the primary way to outline the content of your webpage. They define the outline of your web page as both humans and search engines see it, which makes selecting relevant headings essential for a high-quality web page.

There are six distinct headings or heading components in HTML. Headings can be used for a multitude of reasons, such as titling segments, journals, or other types of content.

One way to think about headings on a web page is like headings in a book..

The <h1>, like the book title, introduces the topic that the web page is all about.

The <h2>, like book chapters, describe the main topics covered on the web page

Smaller headers like the <h3> to <h6> serve as other sub-headings that can be used within each section, just like a book chapter can be as a book chapter may be split up by multiple sub-topics.

Headings are ordered from the biggest to the smallest size. There are 6 levels of headings available, ranging from <h1> to <h6>, 1 being the most important one.

H1 is used for the primary headings. For subheadings, all other lower headings are used.

Here's an example of headers:

<h1>Observable Universe</h1>

<h2>Milky Way Galaxy</h2>

<h3>Earth</h3>

<h4>USA</h4>

<h5>Norfolk, VA</h5>

<h6>Main Street</h6>

Which results in:

# Observable Universe

## Milky Way Galaxy

### Earth

#### USA

##### Norfolk, VA

###### Main Street

#### Accessibility with Headers

For people who are blind or visually impaired, screen reading software is used to parse through text on a web page. A common technique these folks will use to navigate the page is to jump from heading to heading to determine the overall content of the page more easily. That's why it's best practice to not skip one or more heading levels. If you did skip headings and went from <h1> to <h3>, you may cause confusion since the user has to deal with a missing heading. Don't bum out any of your users - structure your headings properly!

### Paragraphs

Paragraphs <p> are the most used HTML element, as they act as the default block-level element and are quick to write.

Below is the HTML code, as well as the “paragraphs” of text - note that there is nothing particularly special with the formatting of these.

<p>

The sweet-faced and loving Labrador Retriever is actually one of the most

popular dog breeds.

</p>

<p>

Labs are extremely friendly with an easygoing and high-spirited personality

which is great for bonding with the whole family.

</p>

The sweet-faced and loving Labrador Retriever is actually one of the most popular dog breeds.

Labs are extremely friendly with an easygoing and high-spirited personality which is great for bonding with the whole family.

### Spans

The HTML <span> element is like a generic wrapper that is used to group text, mostly for styling purposes. Consider the following code:

<style>

p {

color: black;

}

.red {

color: red;

}

</style>

<p>

This sentence needs some <span class=”red”>visual emphasis</span> to really bring home the point.

</p>

And its result:

This sentence needs some visual emphasis to really bring home the point.

In this code, the words visual emphasis have been put inside the span with the class red, so that those individual words can be styled separately from the rest of the p element. In this instance, the span words would be red, while the rest of the words would be black.

### Blockquotes

Blockquotes are used to identify a citation.

<blockquote cite="https://www.wikiwand.com/en/Scooby-Doo\_(character)">

<p>Ruh-roh--RAGGY!!!</p>

<footer>—Scooby Doo, <cite>Mystery Incorporated</cite></footer>

</blockquote>

Ruh-roh--RAGGY!!!

—Scooby Doo, Mystery Incorporated

### Line Breaks

The spacing between code in an HTML file doesn’t affect the positioning of elements in the browser.

If you are interested in modifying the spacing in the browser, you can use HTML’s line break element.

<p>

I jump in delight<br />

I run off in frenzy<br />

For now I have just realized<br />

that the fun has arrived<br />

the fun has begun<br />

jumping all on one piece<br />

almost feeling like I can't breathe<br />

blood rushing through me<br />

a second, a beat<br />

I feel the air on my face<br />

My fur rising up<br />

Free as free as it can be<br />

That's what you feel<br />

When your owner has arrived<br />

</p>

I jump in delight  
I run off in frenzy  
For now I have just realized  
that the fun has arrived  
the fun has begun  
jumping all on one piece  
almost feeling like I can't breathe  
blood rushing through me  
a second, a beat  
I feel the air on my face  
My fur rising up  
Free as free as it can be  
That's what you feel  
When your owner has arrived

## HTML Lists

You can also organize content in list format.

Although I noted 2 lists in HTML, there are actually 3 total types of lists in HTML:

1. unordered - lists in no specific order
2. ordered - lists in a specific order
3. description - lists with name/value pairs

Depending on the use case, you may want to use one over the other. Just then, I used an ordered list because there was a specific number of lists I wanted to showcase.

Description lists are out of scope for this lesson, but you can learn more about them [here](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/dl).

### Unordered Lists

If you want items in no particular order, like with a shopping list, you use the unordered list HTML tag - <ul>.

An unordered list outlines individual list items with a bullet point with each individual bullet added using the list item or <li> tag.

<p>New puppy shopping list</p>

<ul>

<li>Treats</li>

<li>Dog food</li>

<li>Leash</li>

<li>Collar</li>

<li>Dishes</li>

<li>ID tag</li>

</ul>

Which results in:

New puppy shopping list

* Treats
* Dog food
* Leash
* Collar
* Dishes
* ID tag

### Ordered Lists

Ordered lists <ol> are like unordered lists, except that each list item is numbered.

They are useful when you need to list different steps in a process or rank items for first to last given the order of items is relevant.

Just like with unordered lists, you can add individual list items to the list using <li> tags.

<p>Steps after adopting a puppy</p>

<ol>

<li>Spoil the puppy</li>

<li>Be happy with your puppy</li>

<li>Repeat</li>

</ol>

Which results in:

Steps after adopting a puppy

1. Spoil the puppy
2. Be happy with your puppy
3. Repeat

Ordered lists are automatically numbered by the browser, so the numbers don’t need to be included in your HTML.

## The DOM

**Note:** If you’d like to inspect the HTML of a website as Daniel does in the video, you can use [Chrome’s Developer Tools](https://developers.google.com/web/tools/chrome-devtools/) to do so.

Now that you’ve learned about some of the most common HTML elements, it’s time to learn how to set up an HTML file. HTML files require certain elements to set up the document properly. You can let web browsers know that you are using HTML by starting your document with a document type declaration. The declaration looks like this:

<!DOCTYPE html>

This declaration is an instruction, and it must be the first line of code in your HTML document. It tells the browser what type of document to expect, along with what version of HTML is being used in the document. For now, the browser will correctly assume that the html in <!DOCTYPE html> is referring to HTML5, as it is the current standard.

In the future, however, a new standard will override HTML5. To make sure your document is forever interpreted correctly, always include <!DOCTYPE html> at the very beginning of your HTML documents.

### The Head

So far you’ve done two things to set up the file properly:

1. Declared to the browser that your code is HTML with <!DOCTYPE html>
2. Added the HTML element (<html>) that will contain the rest of your code.

Remember the <body> tag? The <head> element is part of this HTML metaphor. It goes above our <body> element.

### Metadata

The <head> element contains the metadata for a web page.

Metadata is information about the page that isn’t displayed directly on the web page.

Unlike the information inside of the <body> tag, the metadata in the head is information about the page itself.

### Title

A browser’s tab displays the title specified in the <title> tag. The <title> tag is always inside of the <head>.

**<!DOCTYPE html>**

<html>

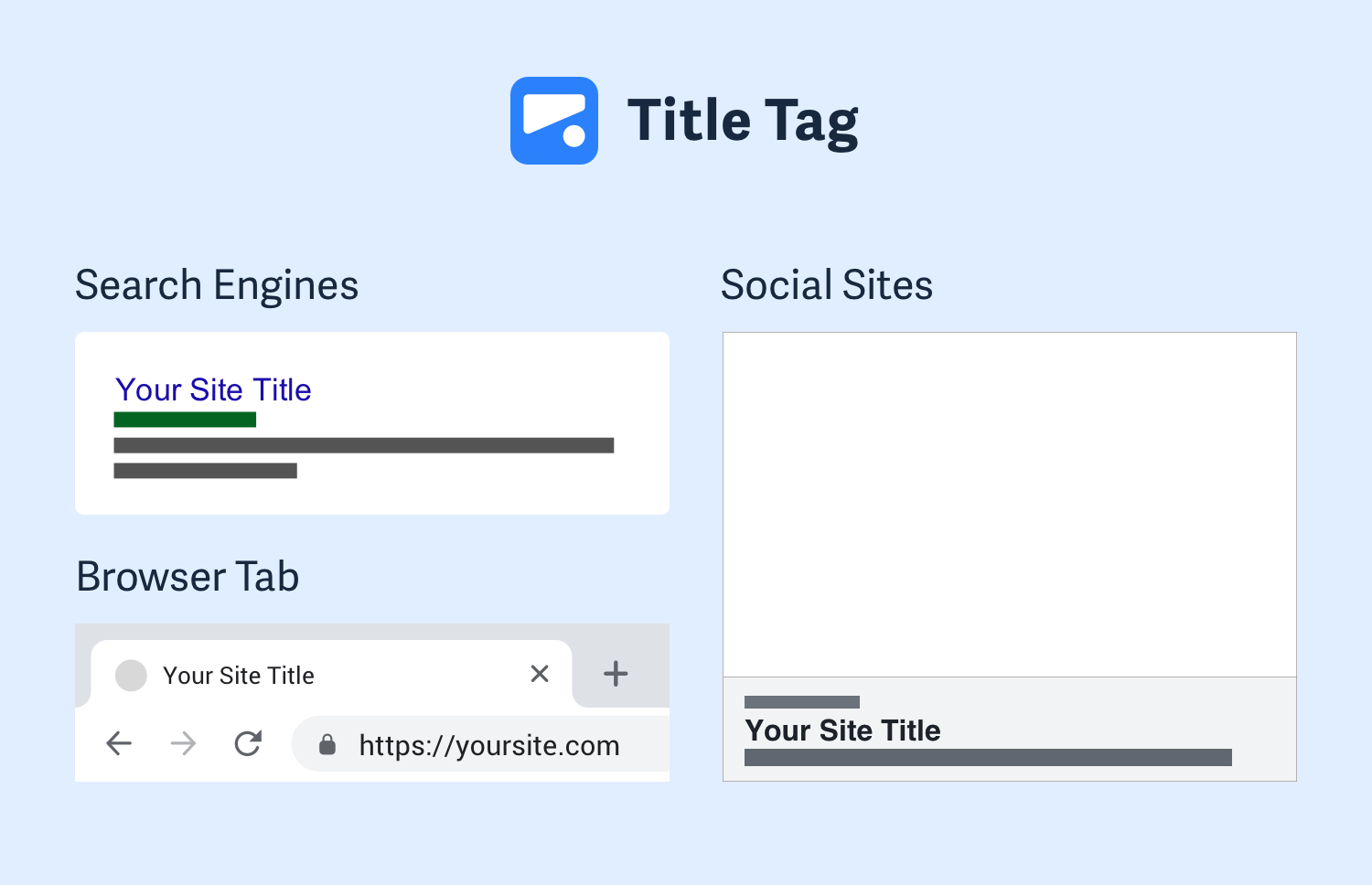
<head>

<title>My Coding Journal</title>

</head>

</html>

If we were to open a file containing the HTML code in the example above, the browser would display the words “My Coding Journal” in the title bar (or in the tab’s title).



An example of titles

### The Body

<body></body>

Once the file has a body, many different types of content – including text, images, and buttons – can be added to the body.

### Hierarchy

HTML is organized as a collection of family tree relationships. As you saw in the last exercise, we placed <p> tags within <body> tags. When an element is contained inside another element, it is considered the child of that element. The child element is said to be nested inside of the parent element.

<body>

<p>This paragraph is a child of the body</p>

</body>

In the example above, the <p> element is nested inside the <body>element. The <p> element is considered a child of the <body> element, and the <body> element is considered the parent. You can also see that we’ve added two spaces of indentation (using the space bar) for better readability.

Since there can be multiple levels of nesting, this analogy can be extended to grandchildren, great-grandchildren, and beyond. The relationship between elements and their ancestor and descendent elements is known as hierarchy.

Let’s consider a more complicated example that uses some new tags:

<body>

<div>

<h1>Sibling to p, but also grandchild of body</h1>

<p>Sibling to h1, but also grandchild of body</p>

</div>

</body>

In this example, the <body> element is the parent of the <div> element. Both the <h1> and <p> elements are children of the <div> element. Because the <h1> and <p> elements are at the same level, they are considered siblings and are both grandchildren of the <body> element.

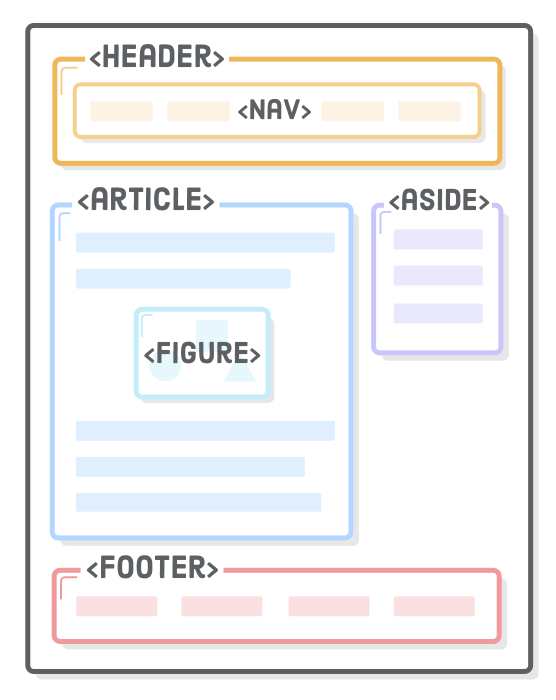
Again, understanding HTML hierarchy is important because child elements can inherit behavior and styling from their parent element. You’ll learn more about webpage hierarchy when you start digging into CSS.

### Semantic Elements

Structure elements allow you to organize the main parts of your page. They usually contain other HTML elements.

Here’s what a typical webpage could include:

* <header> as the first element of the page, that can include the logo and the tagline.
* <nav> as a list of links that go to the different pages of the website.
* <h1> as the title of the page.
* <article> as the main content of the page, like a blog post.
* <footer> as the last element of the page, located at the bottom.



An example of the structure of semantic elements

## Forms

Forms are a part of everyday life. When we use a physical form in real life, we write down information and give it to someone to process. Think of the times you’ve had to fill out information for various applications like a job, or a bank account, or dropped off a completed suggestion card — each instance is a form!

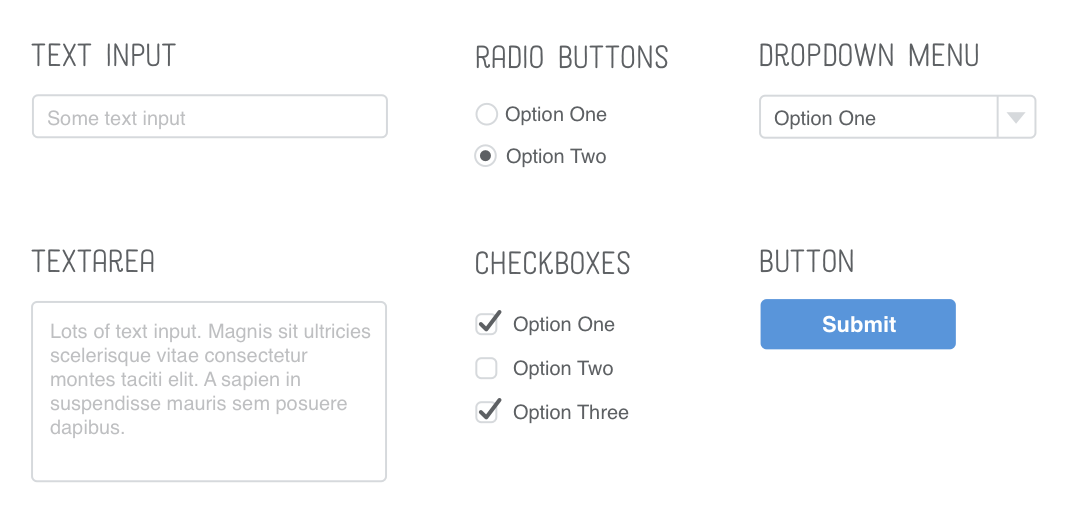
Just like a physical form, an HTML <form> element is responsible for collecting information to send somewhere else. Every time we browse the internet we come into contact with many forms and we might not even realize it. There’s a good chance that if you’re typing into a text field or providing an input, the field that you’re typing into is part of a <form>!

In this lesson, we’ll go over the structure and syntax of a <form> and the many elements that populate it.

HTML form elements let you collect input from your website’s visitors. Mailing lists, contact forms, and blog post comments are common examples for small websites, but in organizations that rely on their website for revenue, forms are sacred and revered.

Forms are the “money pages.” They’re how e-commerce sites sell their products, how SaaS companies collect payment for their service, and how non-profit groups raise money online. Many companies measure the success of their website by the effectiveness of its forms because they answer questions like “how many leads did our website send to our sales team?” and “how many people signed up for our product last week?” This often means that forms are subjected to endless A/B tests and optimizations.

There are multiple types of HTML forms, such as text input, text areas, radio buttons, checkboxes, dropdown menus, and buttons.



Examples of forms

### Input Types, Select and Textarea

Text, checkbox and radio button forms are specified by an input type.

*<!-- A text input -->*

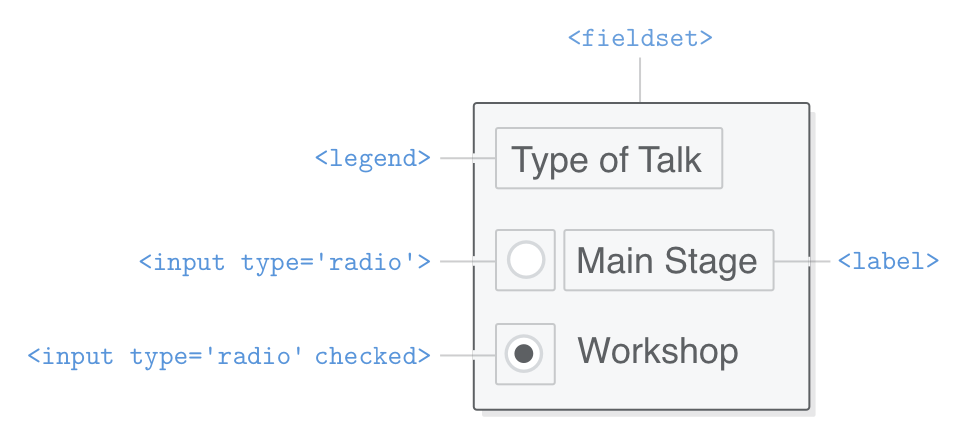
<input type="text" />

*<!-- A checkbox -->*

<input type="checkbox" />

*<!-- A radio button -->*

<input type="radio" />



An example of a ‘radio’ input type

Separately, a dropdown menu can be created using select.

<label for="color-select">Choose a color:</label>

<select id="color-select">

<option value="">--Please choose an option--</option>

<option value="blue">Blue</option>

<option value="red">Red</option>

<option value="green">Green</option>

<option value="yellow">Yellow</option>

<option value="orange">Orange</option>

<option value="pink">Pink</option>

</select>

Choose a color:                    

Last is textarea, which creates a more free-form text field for the user to enter information.

<label for="learn">What do you hope to learn today?</label>

<textarea id="learn" name="learn" rows="5" cols="30">

I hope to learn about...

</textarea>

What do you hope to learn today? 

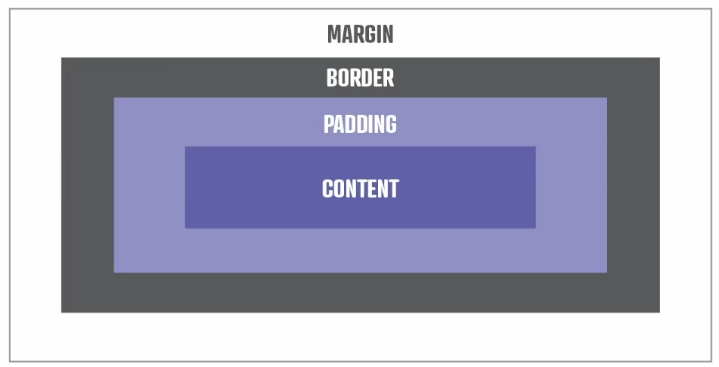
### More on forms

Follow these links to learn more on [HTML forms](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form), [select](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/select), and [textarea](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/textarea).

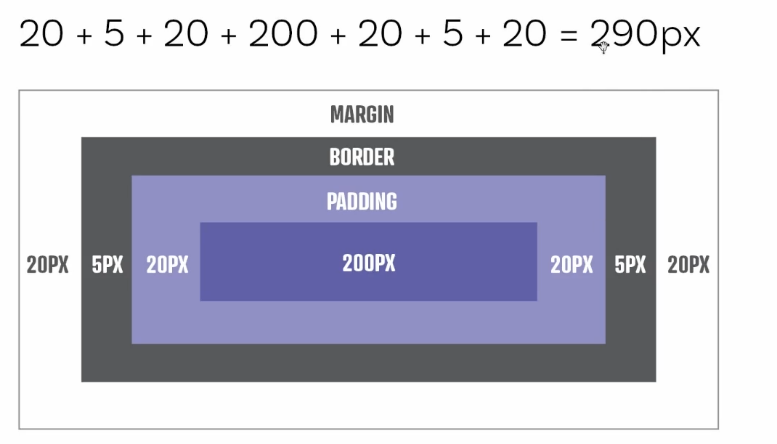
# CSS

## The Box Model

When we declare a width to an element, that width is setting the width of the contend:



But the width is referent only to content by itself, if you define others measurements like, padding, border, margin the values will be added to total, and the width will be different, like this:



In this case above the width wouldn’t be 200px otherwise will be = 290px

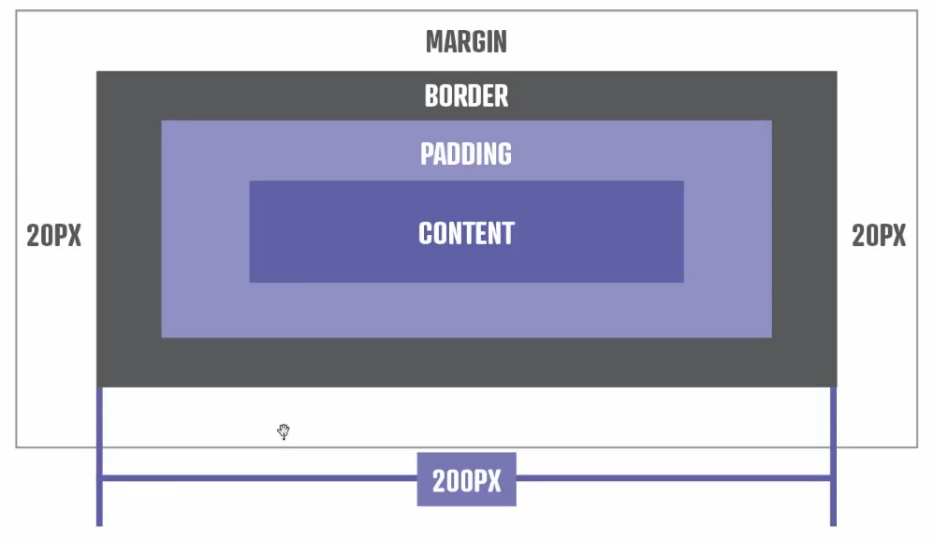
To help make our lives easier, it’s very common to change the box-sizing of an element to border-box:



With that when we declare a width or height, the number we declare includes the content, but also includes the padding and border. Let’s see an example below:

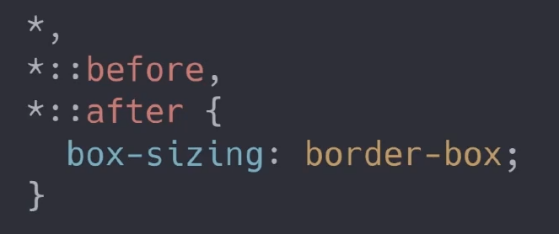


That would result in the following:



Now the 200px will be part of total, all box includes all other elements in the box. One thing is that margin isn’t part of the total because margin is spacing between elements, by only adding the padding and border to the total, our width and height now give us the total size of thevisual parts of our element.

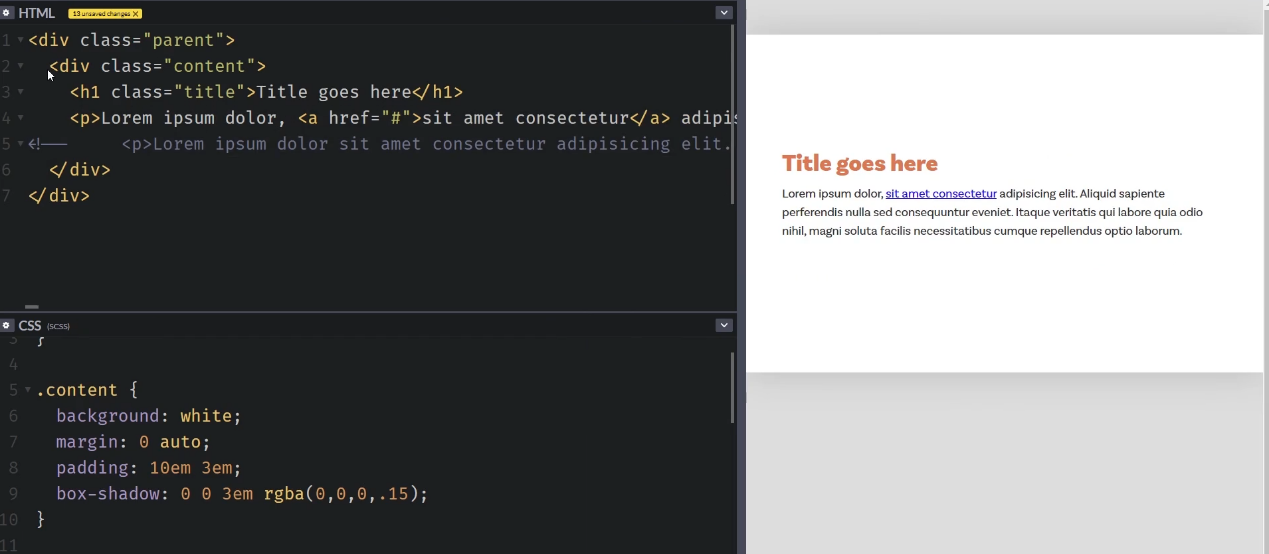
Is common to use a universal selector for use box-sizing in all document, like this:



## Problems in set Height

Is trouble setting heights to a property, is wise to avoid use height, and use min-height instead, because if you set a fixed height and when space is shrinking the content inside the box will leak out the box. With min-height you fixed a height but when shrinking the height will change.

You can use paddings instead use height to create more space in your box.



## Inheritance

Anything related to typography is inherited, like:

font-size | font-family | text-decoration | color…

Everything we see on the page is a child of the HTML element, so they inherit those properties.

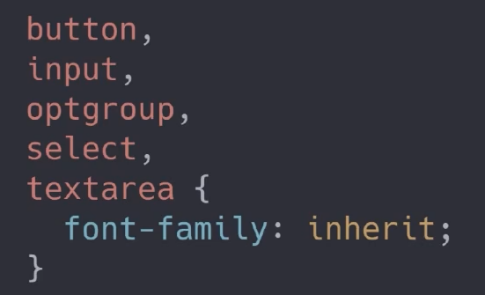
Nothing related to Layout is inherited, like:

margin | padding | height | width | position

The following don’t inherit things like you’d expect them to, because the browser have this elements by default:

<button> | <input> | <optgroup> |<select> | <textarea>

But you can make them inherit if you want, something like this:



## Values em and rem explained

1em is equal to 16px, that is the browser default font size. This value is always related to the parent, if you don’t set a value to your parent the value will be browser default, which is 16px, but if you set a value in an parent box, when you set a value using em in children boxes these values will be related to the parent, and will increase. Look at this example:

<div *class*="col--em">

    <h1>em</h1>

    <p>Lorem ipsum dolor sit amet.</p>

</div>

*.col--em* {

    font-size: 20px;

}

*.col--em* h1 {

    font-size: 2em;

}

The class .col—em have a font size value equal to 20px it is 1em, when we set the value for the children h1 in 2em this mean: 20 x 2 = 40px.

Values rem work a little different, rem is related to root values, not to elements by itself, rem is always related to root values.

## The Cascade

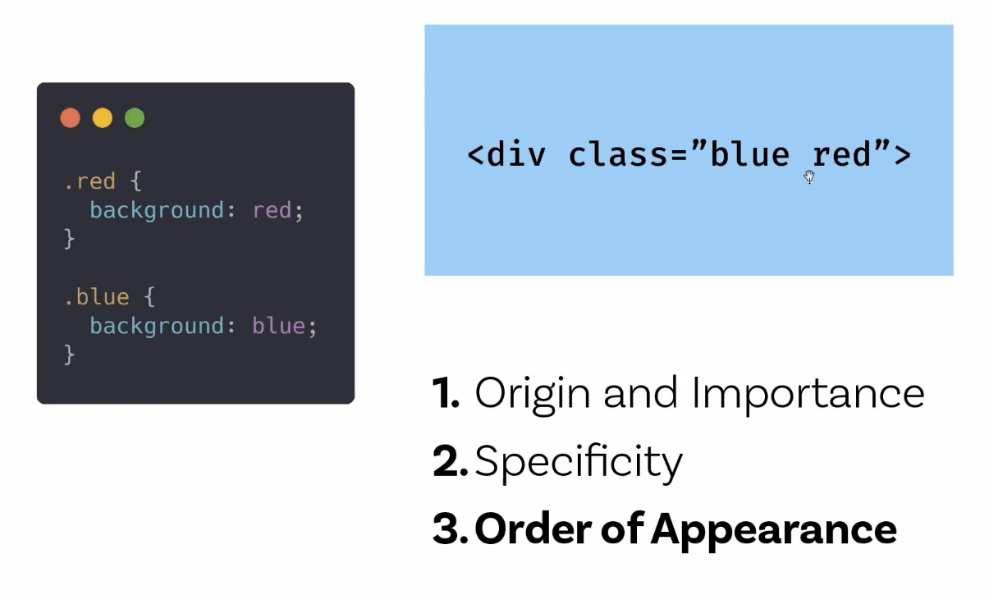
Look at this example below:



To figure out what rule it’s going to use, the CSS cascade algorithm use the following:

1. Origin and Importance
2. Specificity
3. Order of Appearance

In this image below you will see the these rules in action:



Take a look that the order of appearance is mandatory here.

## Origin and Importance

There is more than just !important take look at this example of use:



These are all based on author declarations:

1. Linked style sheets
2. Embedded styles (the <style> attribute)
3. Inline styles

These are things we write. It could be from our stylesheet, JavaScript adding an inline style, or even using an **@import**. Whatever the situation, we have control over these things. And really, these all fall under order of appearance.

Origin is a deeper look at things. Another origin is one you’re probably already familiar with, user agent declarations. These are the browser defaults. So, we have:

* Author declarations (what we control)
* User agent declarations (the browser defaults)

But we also have User declarations, which are the styles that the user imposes. The order of origin is:

1. Author declarations
2. User declarations
3. User agent declarations

When **!important** comes to play

1. **Important user declarations**
2. Important author declarations
3. Author declarations
4. User declarations
5. User agent declarations

Take a look that important user declarations are more important than author declarations, but only some people do this, the user have the ultimate control at the end of day.

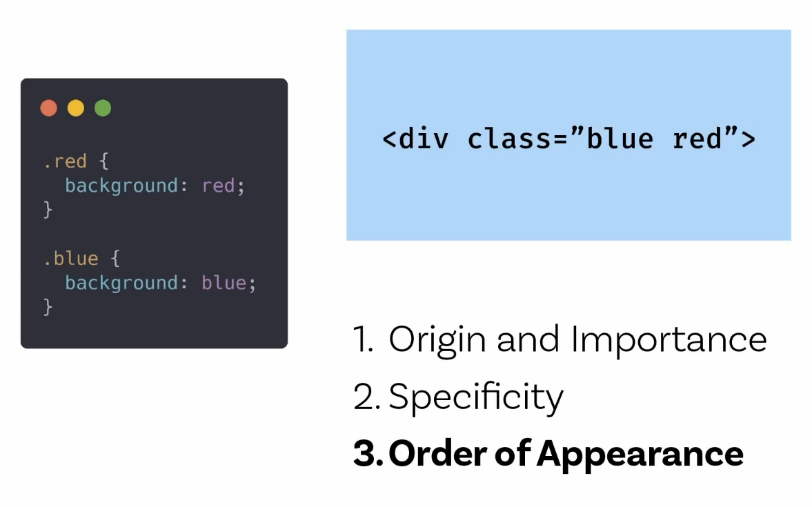
## Origin and Importance when animations & transitions come to play

When animations & transitions come to play there are:

1. Important user declarations
2. Important author declarations
3. **Animations**
4. **Transitions**
5. Author declarations
6. User declarations
7. User agent declarations

## Specificity

Specificity is one of the ways that the browser decides which declaration to use on a given element (if there are 2 or more conflicting styles). The higher the specificity, the more important that selector is considered to be, so it will win.





If we change the order of appearance, that won’t affect, take a look:



The problem with raising specificity is, if you need to overwrite that in the future, it can be frustrating. So by keeping specificity flat, we eliminate one of the things the browser uses to decide which rule to apply.

And if we don’t have specificity wars, that more or less removes the need for ever using !important.

## CSS Selectors

### Tags

In this section, you'll learn how to use different visual CSS guidelines to style elements individually and by group.

CSS can select HTML elements by using an element’s tag name. A tag name is the word (or character) between HTML angle brackets.

For example, in HTML, the tag for a paragraph element is <p>. The CSS syntax for selecting <p> elements is:

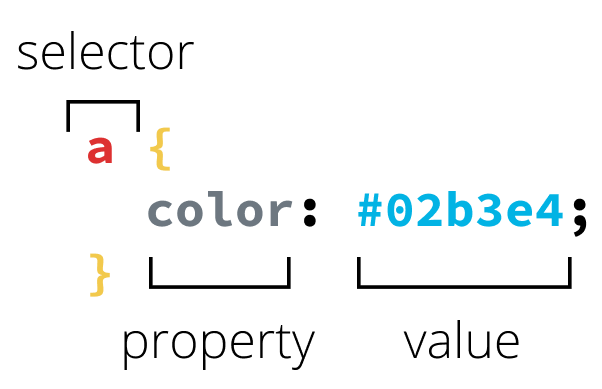
p {

color: red;

}

In the example above, all paragraph elements will be selected using a CSS selector. The selector in the example above is p. Note that the CSS selector matches the HTML tag for that element, but without the angle brackets.

In addition, two curly braces follow immediately after the selector (an opening and closing brace, respectively). Any CSS properties will go inside of the curly braces to style the selected elements.



An example of a CSS selector, property and value.

### Classes

CSS is not limited to selecting elements by tag name. HTML elements can have more than just a tag name; they can also have attributes. One common attribute is the class attribute. It’s also possible to select an element by its class attribute.

For example, consider the following HTML:

<p class="brand">Sole Shoe Company</p>

The paragraph element in the example above has a class attribute within the <p> tag. The class attribute is set to "brand". To select this element using CSS, we could use the following CSS selector:

.brand {

}

To select an HTML element by its class using CSS, a period (.) must be prepended to the class’s name. In the above case, the class is “brand”, so the CSS selector for it is .brand.

### Ids

For situations where you need more specificity in styling, you may also select elements for CSS using an id attribute. You can have different ids associated with a class (although a class is not required). For example, consider the following HTML:

<p id=”solo” class="brand">Sole Shoe Company</p>

The id attribute can be added to an element, along with a class attribute. On the CSS side, the delineation is made by using # to represent an id, the same way . is used for class. The CSS to select and style the HTML element above could look like this:

**#solo** {

color: purple;

}

### Pseudo-classes

A CSS pseudo-class is a keyword added to a selector that specifies a special state of the selected element(s). For example, :hover can be used to change a button's color when the user's pointer hovers over it.

selector:pseudo-class {

property: value;

}

For more information on pseudo-classes, [see the Mozilla Dev Docs here.](https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes)

### Attributes

Attribute selectors are a special kind of selector that will match elements based on their attributes and attribute values.

Their generic syntax consists of square brackets ([]) containing an attribute name followed by an optional condition to match against the value of the attribute.

Attribute selectors can be divided into two categories depending on the way they match attribute values:

1. Presence and value attribute selectors and
2. Substring value attribute selectors.

These attribute selectors try to match an exact attribute value:

* [attr] This selector will select all elements with the attribute attr, whatever its value.
* [attr=val] This selector will select all elements with the attribute attr, but only if its value is val.
* [attr~=val] This selector will select all elements with the attribute attr, but only if val is one of a space-separated list of words contained in attr's value. (This one is a bit more complex, so checking some [documentation](https://developer.mozilla.org/en-US/docs/Web/CSS/Attribute_selectors) might be helpful.)

### Multiple Selectors

What if we want to add some styles to all our headings? We don’t want to have redundant rules, since that would eventually become a nightmare to maintain and is not scalable at all.

h1 {

font-family: "Helvetica", "Arial", sans-serif;

}

h2 {

font-family: "Helvetica", "Arial", sans-serif;

}

h3 {

font-family: "Helvetica", "Arial", sans-serif;

}

h4 {

font-family: "Helvetica", "Arial", sans-serif;

}

h5 {

font-family: "Helvetica", "Arial", sans-serif;

}

h6 {

font-family: "Helvetica", "Arial", sans-serif;

}

Instead, we can select multiple HTML elements in the same CSS rule by separating them with commas. Add this to our styles.css file:

h1,

h2,

h3,

h4,

h5,

h6 {

font-family: "Helvetica", "Arial", sans-serif;

}

This defines the font to use for all of our headings with a single rule. That’s great, because if we ever want to change it, we only have to do so in one place. Copying and pasting code is usually a bad idea for web developers, and multiple selectors can help reduce that kind of behavior quite a bit.

## Linking CSS

Inline styles have the biggest precedence in a style tree, for example if you have a html document with:

<p style="color: red; font-size: 20px;">I'm learning to code!</p>

And in the future you want add an text-decoration: none in an <style> element or external stylesheet, it wouldn’t have any effect, because the inline style have more precedence than the others. Because of that inline styles must be avoided at all costs, it’s a bad practice.

### linking a CSS file to another CSS file

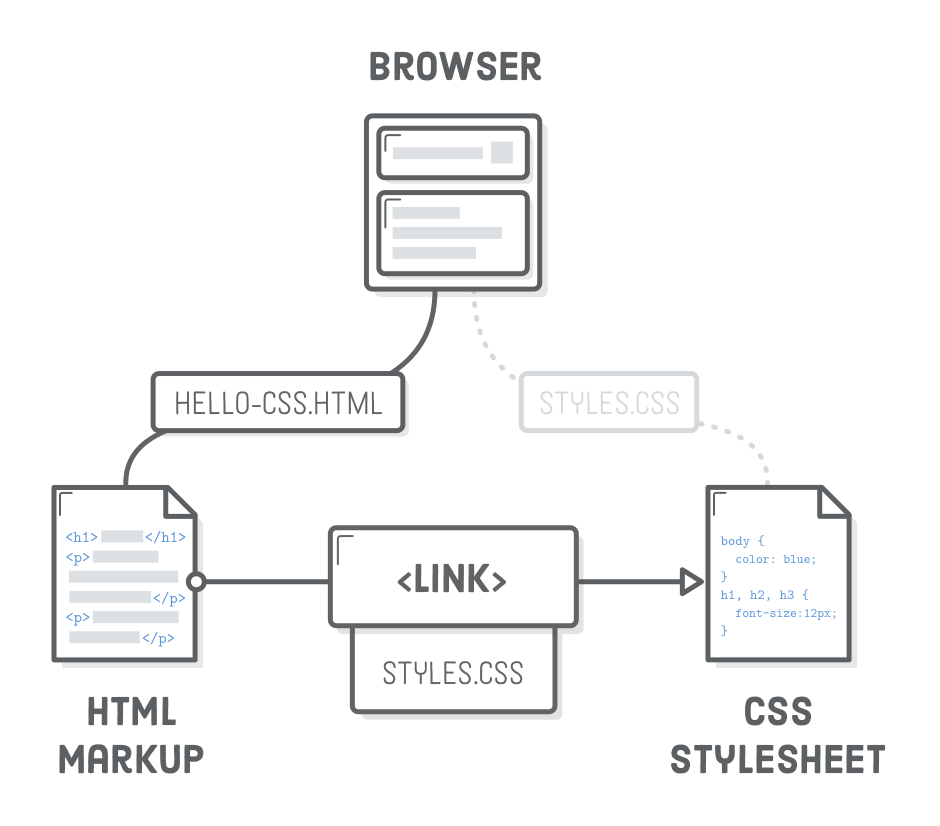
What about linking a CSS file to another CSS file? You can have all your styles living inside one main CSS file, or you can use @import to break your styles (one for layout, one for images, one for blog cards, etc.) into a number of smaller, focused files. This makes it a lot easier to manage the styles they contain and your code is more scalable and modular!

// at the top of your main CSS file

@import “./layout”;

@import “./images”;

@import “./blog-cards”;



Relationship between HTML and CSS files

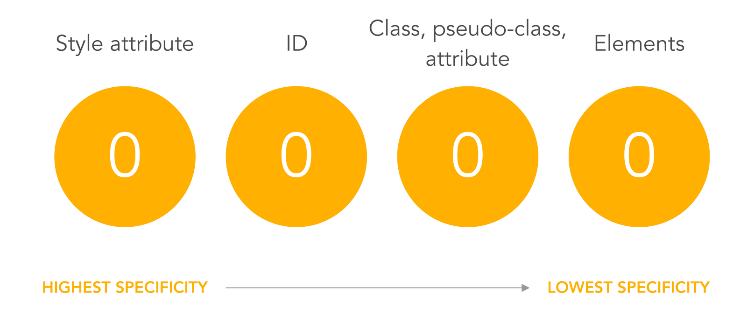
### Specificity

Because elements can have multiple CSS selectors, there is a hierarchy for the weight given to each type of selector. Here is the logical order of selectors from least to most weight assigned:

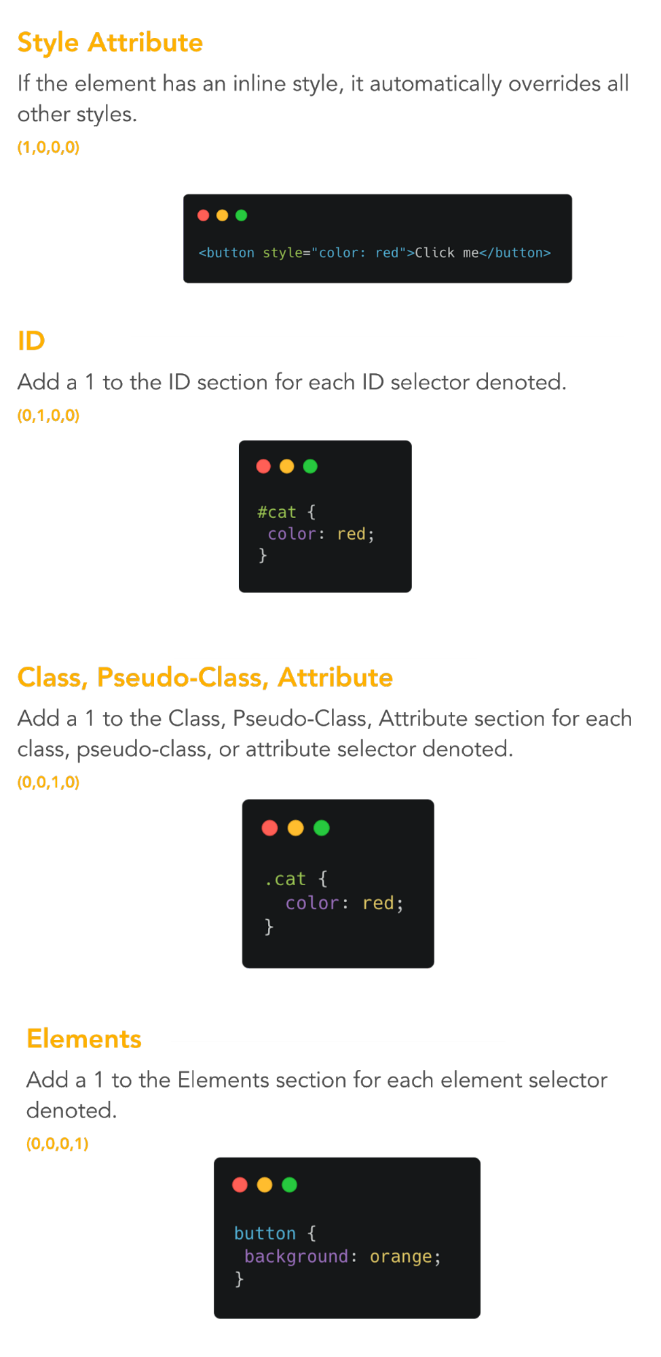
* Type selectors (e.g., h1) and pseudo-elements (e.g., ::before).
* Class selectors (e.g., .example), attributes selectors (e.g., [type="radio"]) and pseudo-classes (e.g., :hover).
* ID selectors (e.g., #example).

This concept can help you understand why your styles aren't being applied in the way you expect. There is a way to escape or override the specificity evaluation of elements using the !important keyword after an individual CSS property rule, but a couple important reminders:

Always look for a way to use specificity before even considering !important. Never use !important on site-wide CSS.



Specificity levels



Specificity explanation - example



Specificity explanation – example

Take a look in these references here:

<https://developer.mozilla.org/en-US/docs/Web/CSS/Attribute_selectors>

<https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes>

## Absolute vs Relative Units

Many CSS properties require a unit of measurement. There’s a lot of units available, but the most common ones you’ll encounter are px (pixel) and em (pronounced like the letter m). The former is what you would intuitively call a pixel, regardless of whether the user has a retina display or not, and the latter is the current font size of the element in question.

There are broadly two types of units of measurement for CSS properties, absolute and relative.

**Absolute**

* px
* in
* mm
* cm

Absolute measurements are set values regardless of anything having to do with your program or the browser. px is the most common absolute unit of measurement, and many font sizes on the web for example, are set to somewhere between 12px-30px, A font size set to 16px will appear the same size no matter how big the screen. If however, you wanted to set a CSS property value based on some dynamic value, the width of a browser window for example, there are also relative units of measurement that can be used to define CSS properties.

**Relative**

* % - percentage of something, such as screen width
* em - A unit equivalent to the current font size - if 12px font, 2em would be 24px
* vw - units of viewport width (essentially the browser’s rendering space). Each unit is 1/100th of width
* vh - the same as above but for viewport height

The em unit is very useful for defining sizes relative to some base font. For example, if you set the font-size of body to 16px, you could then set other element’s font-size value relative to that 16px. Here’s what that could look like:

body {

font-size: 16px;

}

**#one** {

font-size: 1.5em

}

**#two** {

font-size: 0.5em

}

In this example, one would have font bigger than 16px, and two would have font smaller than 16px.

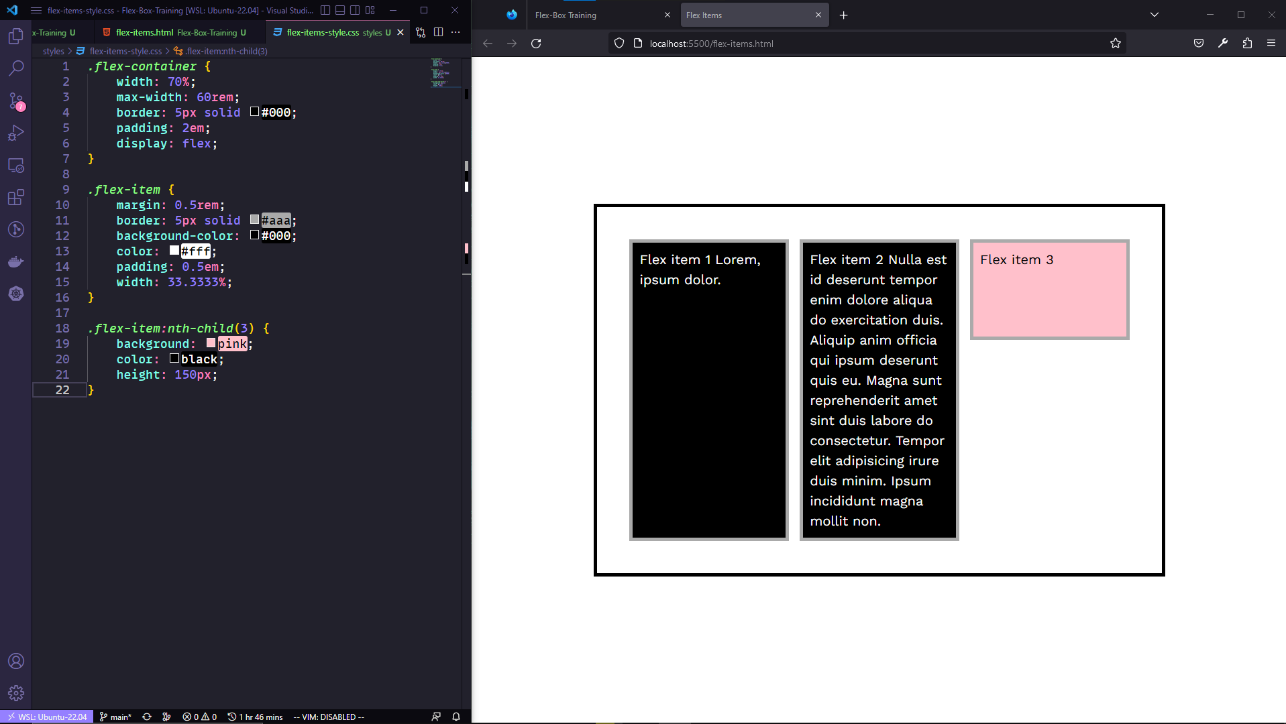
# CSS FLEXBOX

To use flexbox set the display property of a div to flex. The items inside that element will automatically become flex items, and you can then use the flexbox syntax in your CSS code.

When a container receives the property display:flex only the child components are affected. All components inside the parent component have their behavior changed to flex and all child elements behave in the same way.

After researching more about this attribute: .class:nth-of-type(2){}

The .class:nth-of attribute is used to capture attributes by position. There are diferente types of this attribute.



Another behavior is that each child item receives the height of the tallest item inside the container. This rule can be broken when you set the height of an element that is shorter than the others, as in the example above, the pink box received a smaller value compared to the other boxes. But if a box is taller, they will all behave based on the tallest height and all be the same height.

# CSS GRID

Lorem

# DISPLAY TYPES AND NORMAL FLOW

Lorem

# UTILIZANDO CONVENÇÕES DE NOMES

É frequente a utilização de padrões de nomenclaturas para distinguir uma aplicação de regras no CSS, para facilitar o entendimento do que está sendo feito no código, duas delas são:

Uso de duplo underscore:

<div *class*="plan ribbon bg-accent">

    <h2 *class*="plan\_\_title">super</h2>

    <p *class*="plan\_\_price">

Perceba que no container da DIV temos a classe plan, que corresponde a estilização do bloco, e dentro da DIV temos as classes de estilização do bloco que contém a classe plan seguida de dois underscores e a descrição do que está sendo estilizado, essa técnica é utilizada para descrever estilização de bloco especifica que está dentro da div pai.

Outra técnica é a do duplo traços, veja:

<div *class*="team-member bg-dark team-member--mirrored">

Acima temos a classe team-member e logo em seguida também temos a mesma classe seguida de dois traços mais um identificador, isso é um modificador de classe, usado para modificar o comportamento de uma classe, essa modificação de classe não está dentro, não se comporta como um bloco dentro da div pai, mas se comporta como um modificador da classe.

Outra forma de evitar problemas com escopo do CSS é utilizar esse tipo de convenção:

*.team-member* img {

    width: 10rem;

    border-radius: 0.25em;

    margin-right: 1em;

}

*.team-member* *.name* {

    font-size: 1.75rem;

    line-height: 1;

    font-weight: 900;

    color: var(--pop);

    margin: 0;

}

*.team-member* *.position* {

    margin-top: 0;

    font-weight: 700;

    opacity: 0.8;

}

Uma classe de bloco seguida de uma classe interna, para limitarmos o escopo e dizer que queremos estilizar apenas um bloco que esta dentro de um outro bloco especifico.

S