Lesson 8: CSS

[Overview](#h.f7tzhx1vpft)

[Review of Front-End](#h.jt5ad7wlezkp)

[What is CSS?](#h.t4h8flrbe6dv)

[Syntax](#h.utvhgnchme0j)

[We Do: Intro CSS](#h.jys3z9dia4l7)

[More On Selectors](#h.g0hzb0im4hd7)

[Classes and IDs](#h.vkv56enupzqe)

[Multiple Selectors](#h.9xbeoppf8m43)

[Pseudo-Classes](#h.yp0jpygbkiis)

[Font/Text Style](#h.xpzfvn7tb8mc)

[Size Units](#h.atpis96q4sa1)

[Chrome Developer Tools](#h.egy9qz1dkm4f)

[The Box Model](#h.ty8f6mk0b0xk)

[CSS](#h.b2xczox9yec5)

[Borders and Border Radius](#h.8vwmt2f8jywr)

[Colors](#h.l49wwb2xwzql)

[Background](#h.d8uck247u5v2)

[Display](#h.av88ghcf0wfl)

[Box-Sizing](#h.q6guve5gticf)

[Floats](#h.1hxutju01fq1)

# Overview

# Review of Front-End

(This is a repeat of yesterday)

Front-end development is the development of those elements of a website that the user sees and interacts with directly. It is a combination of programming skills and aesthetics (understanding element arrangements on the screen, the color and font choices).

Front-end development uses 3 key languages or tools: HTML, CSS, and JavaScript.

The browser:

* parses the **HTML** to build a hierarchical model of the document;
* applies the **CSS** rules to the elements to render the model as a page on-screen; and
* runs the **Javascript** in the page to animate or manipulate both the model and the CSS on-screen in reaction to the user's clicks and keypresses.

One way to think of this is with a grammar analogy:

* HTML does the nouns: <p></p> means "paragraph";
* CSS does the adjectives: p {color:#000;} means that paragraph text is black;
* Javascript does verbs: window.open() means "open a window"

Today, we are going to focus on CSS.

# What is CSS?

CSS stands for **Cascading Style Sheets** and is a markup language used exclusively to style html. A **style sheet** is a separate file that holds all of your css, and is called upon in your HTML.

We say these style sheets are **cascading** because the sheets can apply formatting when more than one style applies. If we say all paragraphs should be blue, except for one paragraph which should be red. We can easily do that.

We hold our css in a **different file** for a couple of reasons.

* We can apply the same styling to several html elements without rewriting code over and over.
* We can do this from one single file.
* More organized
* Easier to keep consistent

Basically, it saves us from writing the same code, over and over and over again.

We’ve actually already seen some **inline CSS** (yesterday). We could do this all the time, but it would get very tiring and make your HTML file very long and difficult to read.

We **reference** our style sheets in the **head** of our html document:

<link type=”text/css” rel=”stylesheet” href=”stylesheet.css” />

## Syntax

CSS is generally written like this:

Selector {  
 Property: Value;

Another-Property: Another-Value;  
}

The **selector** can be an html tag, a class, or an id.

The **property** is what we actually want to change, like background color, font size, etc.

The **value** is what we are doing to that property, like blue, 12px, etc.

We have to use **curly brackets** around everything for a selector, then **colons** after each property, and **semicolons** at the end of the value. We can set **multiple properties** inside the curly brackets for one selector. For example:

p {  
 Color: red;  
 Font-family: Arial;  
 Font-size: 20px;  
}

## We Do: Intro CSS

Create a new folder in your front-end folder called intro-css. Create an html file called index.html and a css file called style.css.

1. Add the link to your css file in the head of the html file.
2. In the html file, add content:
   1. h1 - “My favorite restaurants”
   2. div
      1. h2 - “Kokkari”
      2. h3 - “Greek food in SF
      3. p - “review…”
3. Open in Chrome to see what it looks like.
4. Add style:
   1. Change the default font (body)
   2. Add an id to h1 like “title”, then change the h1 color using id selector
   3. Add a class to div, like restaurant-group, then change the background color to #CCC.

<head>

<title>Sia's Favorite Restaurants</title>

<link rel="stylesheet" type="text/css" href="style.css">

</head>

<body>

<h1 id="title">Sia's Favorite Restaurants</h1>

<div class="restaurant-group">

<h2>Kokkari Estiatorio</h2>

<h3>Greek restaurant in San Francisco, CA</h3>

<p>This is my favorite Greek restaurant in the world. It takes classic Greek recipes and updates them based on local ingredients available in California.</p>

</div>

</body>

body {

font-family: "Lucida Sans Unicode", "Lucida Grande", sans-serif;

}

#title {

color: blue;

}

.restaurant-group {

background-color: #CCC;

}

Classroom challenge:

1. Add another one of your favorite restaurants to the html content using the first as a guide. Partner up if you have an issue you can’t figure out. You can find more web-safe fonts [here](http://www.w3schools.com/cssref/css_websafe_fonts.asp).
2. Commit your changes

## More On Selectors

### Classes and IDs

An **id** should be unique within a page, so the id selector is used if you want to select a single, unique element. We call it using a # before the ID name.

A **class** is similar except that by convention, it can be used more than once on a page to help identify several things that should be treated in the same way. We call it using a period before the class name.

### Multiple Selectors

We can chain together selectors in order to separate out styling. For example, maybe I use h2 in multiple places, but I only want to change it for the restaurant-group. I can do this:

.restaurant-group h3 {

color: #FFF;

font-weight: normal;

font-style: italic;

}

### Pseudo-Classes

Pseudo-class selectors are a way of accessing html items that aren’t part of the document. Two very big examples of this are **:hover** and **:visited**.

* **Hover** can allow you to change an item’s appearance when someone points their mouse over the item.
* **Visited** allows you to change the styling of a link that has been visited before.

Classroom challenge: Pair-program to add a link to your restaurants’ websites. Then, modify the hover styling. (Who remembers how we make it open in a new tab?)

a {

background-color: blue;

color: white;

text-decoration: none;

}

a:hover {

background-color: #555;

}

## Font/Text Style

Here are some common font CSS properties and values:

p {  
 font-size: 1.2em;  
 font-family: 'helvetica', 'arial', 'sans-serif';  
 font-weight: 100; /\* 100-900 \*/  
 color: #777;  
 line-height: 125%;  
 letter-spacing: 2px;

text-align: center;  
}

## Size Units

CSS has several different units for expressing a length, used in properties like width, margin, padding, font-size, border-width, etc. The length is a number followed by a length unit, such as 10px, 2em, etc. without any space between the number and the unit. However, if the value is 0, the unit can be omitted.

There are two types of length units: **relative** and **absolute**.

* Relative length units specify a length relative to another length property. Relative length units scales better between different rendering mediums.
  + **em** - Relative to the font-size of the element (2em means 2 times the size of the current font)
  + **%** - tbc.
* The absolute length units are fixed and a length expressed in any of these will appear as exactly that size. Absolute length units are not recommended for use on screen, because screen sizes vary so much, though pixels do vary some by screen.
  + **px** - pixels (1px = 1/96th of 1in)

Classroom challenge: Make sure you have used each of those properties at least one time in your CSS.

# Chrome Developer Tools

As you can see, our page isn’t too impressive yet. Let’s take a step back to understand 2 things that will help you out in your front-end development: Chrome Developer Tools and the Box Model.

Chrome Developer Tools, are a set of web authoring and debugging tools built into Google Chrome. DevTools provide web developers deep access into the internals of the browser and their web application to efficiently track down layout issues, set JavaScript breakpoints, and get insights for code optimization.

DevTools are really powerful, but today we’re just going to focus on… **Inspecting the DOM and styles**

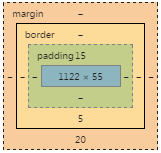
The [**Elements**](https://developer.chrome.com/devtools/docs/dom-and-styles)panel lets you see everything in one DOM tree, and allows inspection and on-the-fly editing of DOM elements. You will often visit the Elements tabs when you need to identify the HTML snippet for some aspect of the page. For example, you may be curious if an image has an HTML id attribute and what the value is.

You can learn more about DevTools here: <https://developer.chrome.com/devtools>

Access by right-clicking then selecting Inspect Element. Or, ctrl+shift+i or cmd+option+i. (r is reload)

# The Box Model

Now that we are in DevTools, we can easily take a look at the box model of any element.



All elements on the page can be considered boxes!

* Each box has internal content
* Boxes come with styles that adjust the amount of space they take up
* If you're ever in doubt, check the inspector!

**Margin**: the space outside the box

**Border**: the space around the box (the edge or border thickness)

**Padding**: the space between the border and the content

**Content**: the actual stuff in the box - like text or a shape or image.

# CSS

Now that we understand DevTools and the Box Model, let’s make our links and divs a bit prettier.

Add margin & padding to div and padding to link:

.restaurant-group {

background-color: #CCC;

padding: 10px 10px 20px 10px;

margin: 10px;

}

a {

background-color: blue;

color: white;

text-decoration: none;

padding: 10px;

}

## Borders and Border Radius

We can set a border

​border: 1px solid black;

Then border-radius: will round the edges out. Great for making buttons. It has been proven a button with rounded edges gets more clicks!

​border-radius: 10px;

Add border & radius to the div and link. Note we are essentially turning our link into a button, which should normally be called with the button tag.

.restaurant-group {

background-color: #CCC;

padding: 10px 10px 20px 10px;

margin: 10px;

border-radius: 10px;

}

a {

background-color: blue;

color: white;

text-decoration: none;

padding: 10px;

border: 2px solid #000084;

border-radius: 10px;

}

a:hover {

background-color: #555;

border: 2px solid #333;

}

That was a good segue to colors!

## Colors

Colors are based on a Red, Green, and Blue scale (aka, the light color spectrum).

* RGBA Style Declaration - three values between 0 and 255, and one for opacity between 0 and 1
  + rgba(rrr, ggg, bbb, a)
* ​Hex Style Declaration - six digits with values between 0 and f, corresponding to RGB
  + ​#rrggbb
  + Can be simplified to #rgb if digits in the same scale are the same.
* ​Web colors - "named" colors that are universal to all browsers, but very limited

do an activity

show tools for color

show gradually modifying in chrome?

## Background

* You can use **background-color:** and set a color
* **Background-image:** and set an image based on a file or url(“example.com/example.jpg”)
* Or simplify both and just call **background:**

(show each of these) - see webkit thing

## Display

To learn about display properties, let’s first set up our html and css with some basics:

<br>

<h2>Playing with display styles</h2>

<div class="play" id="one"></div>

<div class="play" id="two"></div>

<div class="play" id="three"></div>

<div class="play" id="four"></div>

div.play {

height: 50px;

width: 100px;

border: 1px solid black;

border-radius: 5px;

}

#one {

background-color: red;

}

#two {

background-color: blue;

}

#three {

background-color: green;

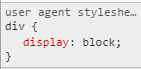
}

#four {

background-color: purple;

}

Now, inspect element, and notice that the display property is set to block by default:



Normally an element’s outermost block extends all the way across the page. That’s why everything so far we have built has been stacked on top of each other.

Our first Positioning property is **Display**. This changes how an element is seen.

* **Block**: This makes the element a block and won’t let anything sit next to it on a page
* **Inline-Block**: this makes the element a block but will allow other elements to sit next to it on the same line
* **Inline**: this makes the element sit on the same line as another element but without formatting it like a block
* **None**: this make everything disappear

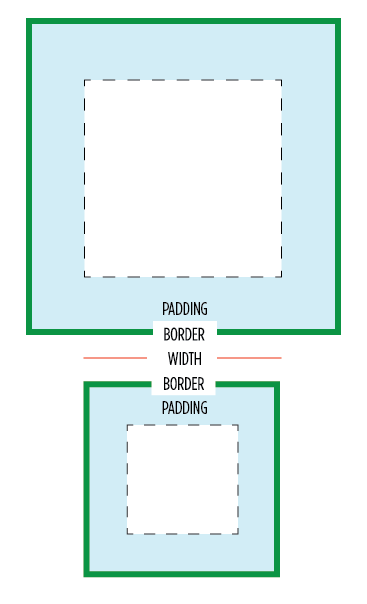
Let’s try this together:

* Set all of those divs to inline block by setting the base div
* Set to inline
* Set back to inline block and then change only one of them to none
* Change the height and width to use % instead of px. Change your screen size to see the impact
* Add some text and edit padding and margins then play with box-sizing (below, optional)

## Box-Sizing

The box-sizing property is used to tell the browser what the sizing properties (width and height) should include. Should they include the border-box? Or just the content-box (which is the default value of the width and height properties)?

|  |  |
| --- | --- |
| content-box | Default. The width and height properties (and min/max properties) includes only the content. Border, padding, or margin are not included |
| border-box | The width and height properties (and min/max properties) includes content, padding and border, but not the margin |



See this post for more in-depth explanation: <https://css-tricks.com/box-sizing/>

## Floats

With CSS float, an element can be pushed to the left or right, allowing other elements to wrap around it. Float is often used with images, but it is also useful when working with layouts.

Elements are floated horizontally, this means that an element can only be floated left or right, not up or down. A floated element will move as far to the left or right as it can. Usually this means all the way to the left or right of the containing element.

* The elements after the floating element will flow around it.
* The elements before the floating element will not be affected.
* If an image is floated to the right, a following text flows around it, to the left:

If you place several floating elements after each other, they will float next to each other if there is room.

1. Add images before your restaurant headings. Give them a class of “rest-photo” and in style.css, a width of 50%.
2. Change your restaurant group class to have a width of 40%.
3. Now, on rest-photo, add “float: right;” and notice what happens.

Maybe you don’t want the paragraph to float around the photo. To avoid this, use the clear property. The clear property specifies which sides of an element other floating elements are not allowed.

Add:

.restaurant-group p {

clear: both;

}

(comment it out if you don’t like it)

Now let’s try floating left instead. It doesn’t look so great. How could we fix it so there is some space? … Let’s wrap all the text stuff into a div with a new class (rest-text) and then add a left margin of 52% to our still for .rest-text.

Note:

* removes the display property from elements
* each element only takes up the necessary space on the page for its box
* breaks box-model of non-floated wrappers