Intro to Web Development

An introduction to designing and building websites

Written by Stephany Lopez

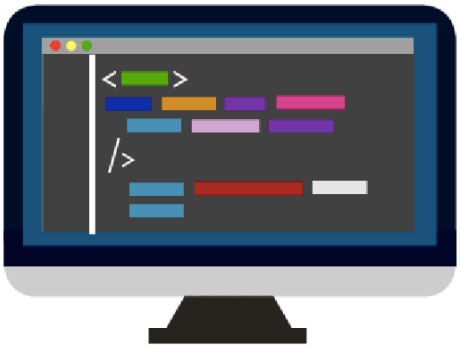
INTRODUCTION

What is Web Development?

Web development is a broad term for the work involved in developing a website from the ground up. There are generally two parts to web development:

Front End

Back End



Front End development uses HTML, CSS, and JavaScript to create an interactive user experience. Everything a computer user sees or clicks is the work of a front-end developer who creates client-side software that brings the site's design to life.

Back End development uses languages like Java, Python, PHP, and Ruby to build the behind-the-scenes functionality of a website, like storing and retrieving a user's information, securely storing and accepting credit card numbers, and more!

A combination of both of front-end and back-end is what we call *full-stack development*.

What you will learn

Throughout this bootcamp you'll learn the fundamental blocks of web development using HTML, CSS, and JavaScript. Using these technologies, you'll build a series of projects that will expand your knowledge in web development.

HTML

Content & Structure
Headings
Paragraphs
Lists

CSS

Presentation
Font
Background color
Border

JavaScript

Behavior
Dynamic Display
User Interaction
Widgets

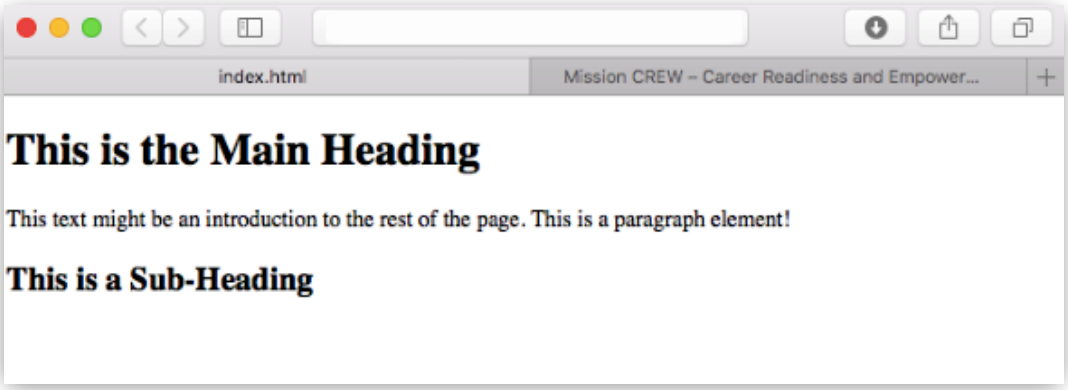
Even if you're not interested in being a programmer or studying Computer Science, have no fear! The skills you'll learn in this course will teach you how to pay great attention to detail, think abstractly, and sharpen your problem solving skills- which are desired skills throughout any job field.

HTML

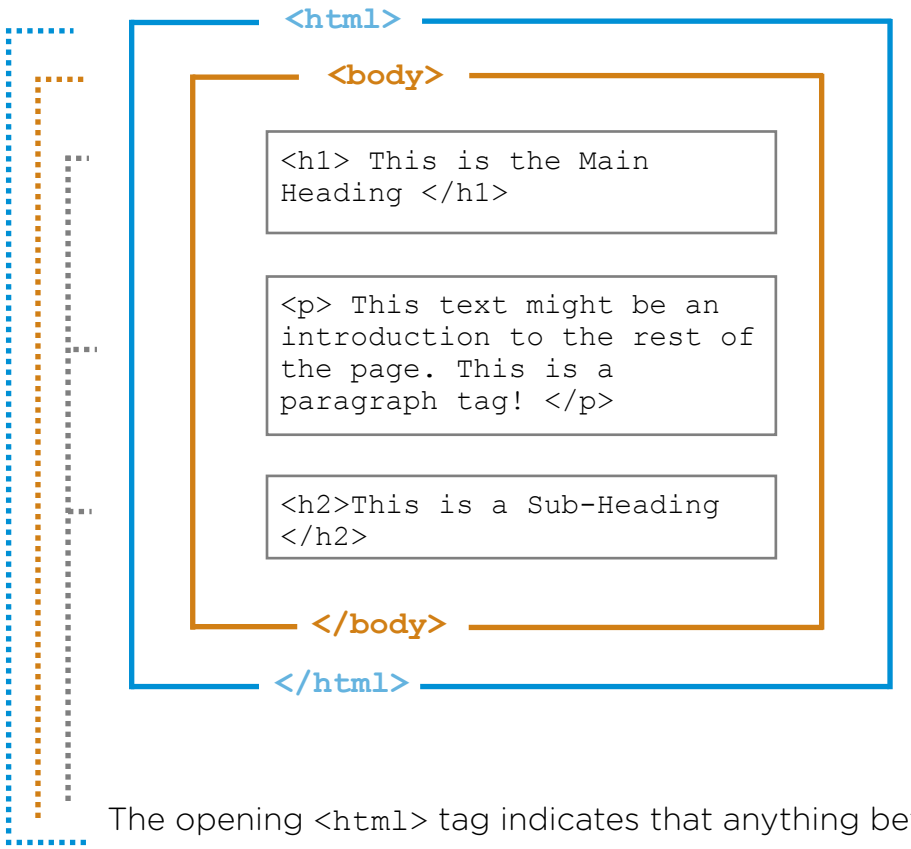
In this first section, we will look at how HTML is used to create web pages. HyperText Markup Language (HTML) defines the content of web pages. It's used to write and create content like text, photos, images, videos, widgets, and other content **elements** on a web page. Let's look at some HTML code, and then see how information is displayed in a web browser.

```
1 <!DOCTYPE html>
2 <html>
3  <body>
4       <h1> This is the Main Heading </h1>
5       <p>  This text might be an introduction to
6.           the rest of the page. This is a
7            paragraph element! </p>
8       <h2> This is a Sub-Heading </h2>
9   </body>
10 </html>
```

The HTML code (in blue) is an element, which is typically made up of two **tags**: an opening tag and a closing tag. Each of the elements in the example above tell the browser something about the information that sits between its opening and closing tags. Let's look at how this information is displayed in a web browser.



Let's look closer at the code from the last page. There are several different elements.



Tags act like containers. They tell you something about the information that lies between their opening and closing tags.

The opening `<html>` tag indicates that anything between it and a closing `</html>` tag is HTML code.

The `<body>` tag indicates that anything between it and the closing `</body>` tag should be shown inside the main browser window. The closing `</body>` tag indicates the end of what should appear in the main browser window.

Words between <h1> and </h1> are a main heading.

A paragraph of text appears between these and tags.

Words between <h2> and </h2> form a sub-heading.

ELEMENT HALL OF FAME

The Element Hall of Fame recognizes the best-known and resourceful HTML elements used in introductory web development.

<head>

The `<head>` element is one of the most important elements of an HTML document. This element can include a title for a document, meta information, scripts, styles, and more.

<body>

Like an English paper, the HTML `<body>` element defines the main content of the HTML document, or section of the HTML document that will be directly visible on a web page.

<title>

The `<title>` element is typically shown at the top of a web browser, and is most commonly known as the “tab name”.

<p>

To create a paragraph, surround the words that make up the paragraph between an opening `<p>` tag and closing `</p>` tag.

The `<strong>` element indicates that the content between the tags has strong importance. By default, the contents of a `<strong>` element will be in **bold**.

The `<em>` element indicates emphasis that subtly changes the meaning of a sentence. By default, the contents of a `<em>` element will be in *italics*.

ELEMENT HALL OF FAME

<h1>
<h2>
<h3>
<h4>
<h5>
<h6>

HTML defines six “levels” of headings.

`<h1>` defines the most important heading, and should be used for main headings followed by `<h2>`, then less important by `<h3>`, and so on.

`<h6>` defines the least significant heading.

Search engines use heading to index the structure and content of your web pages.

Each heading has a default size, however, they can be modified with the style attribute (which you’ll learn more about later).

```
<h1> Heading 1 </h1>
<h2> Heading 2 </h2>
<h3> Heading 3 </h3>
<h4> Heading 4 </h4>
<h5> Heading 5 </h5>
<h6> Heading 6 </h6>
```

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

ELEMENT HALL OF FAME

The <ol> element creates a list, with the assistance of the <li> element, where each item in the list is numbered. (The ol stands for ordered list.)

Each item in the list is placed between an opening <li> tag and a closing <li> tag. (The li stands for list item.)

Here's an example:

```
<ol>
  <li> Wake up </li>
  <li> Eat breakfast </li>
  <li> Shower </li>
  <li> Brush teeth </li>
  <li> Get dressed </li>
  <li> Pack homework </li>
  <li> Get backpack </li>
  <li> Drive to school </li>
</ol>
```

1. Wake up
2. Eat breakfast
3. Shower
4. Get dressed
5. Pack homework
6. Get backpack
7. Drive to school

An unordered list is created with the <ul> element. Like ordered lists, unordered lists require the use of the <li> element.

```
<ul>
  <li> 1 cup sugar </li>
  <li> 1/2 cup butter </li>
  <li> 2 eggs </li>
  <li> 2 tsp vanilla </li>
  <li> 1/2 cup milk </li>
  <li> 1 1/2 cup flour </li>
</ul>
```

- 1 cup sugar
- 1/2 cup butter
- 2 eggs
- 2 tsp vanilla
- 1/2 cup milk
- 1 1/2 cup flour

ELEMENT HALL OF FAME

<a>

The HTML <a> element (or *anchor* element) creates a hyperlink to other web pages, files, address a new email to someone, or any other URL. This element uses a the href attribute*, which indicates the link's destination.


```
<p> Social Media Websites:
<ul>
  <li><a href="https://www.facebook.com"Facebook</a></li>
  <li><a href="https://www.instagram.com"Instagram</a></li>
  <li><a href="https://www.twitter.com"Twitter</a></li>
</ul>
</p>
```

Social Media Websites:

- Facebook
- Instagram
- Twitter

* Attributes provide additional information about the contents of an element.

ELEMENT HALL OF FAME

In HTML, images are defined with the `<img>` element. It contains attributes only, and is an empty element (which means there is no closing tag). The `<img>` element must carry the two attributes: **src** (an attribute that tells the browser where to find the image file) and **alt** (an attribute that provides a description of the image).

In this example, the value for the source attribute is an image saved as "ffc.jpg". This example assumes that this image is located in a folder named "photos", and is stored in the same directory as your HTML file.

```
<p> <img src="photos/feminstFightClub.jpg"
alt="Feminist Fight Club by Jessica Bennett">
In Feminist Fight Club, acclaimed journalist Jessica
Bennett blends the personal stories of her retail-life
fight club with research, statistics, and no bullsh*t
advice for how to combat today's sexism.
</p>
```

If you'd like to link an *external* image (an image from another website) for the source attribute, you can use an absolute URL.

```
<p> <img src="https://s3-us-west-2.amazonaws.com/
tabs.web.media/c/2/c2sl/c2sl-square-orig.jpg"
alt="Feminist Fight Club by Jessica Bennett">
In Feminist Fight Club, acclaimed journalist Jessica
Bennett blends the personal stories of her retail-life
fight club with research, statistics, and no bullsh*t
advice for how to combat today's sexism.
</p>
```

ELEMENT HALL OF FAME

<table>

The <table> element is used to create a table (how original...). The contents of the table are written out row by row.

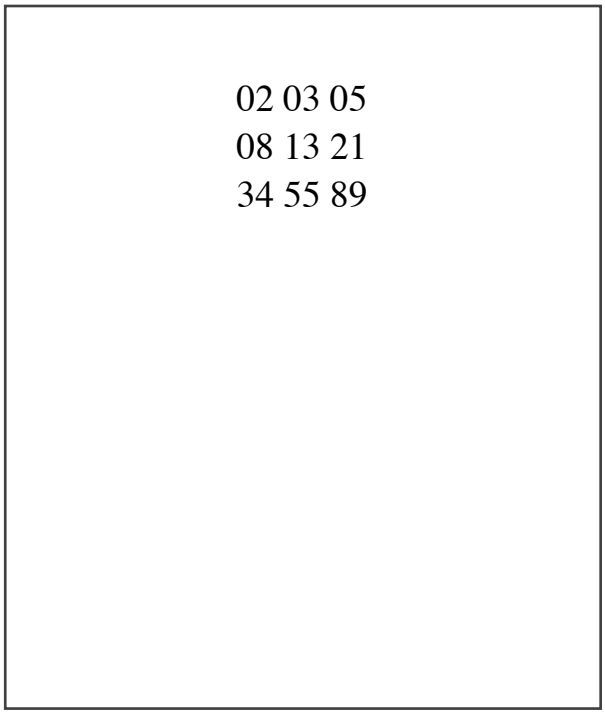
<tr>

The start of each row for a table is used with the opening <tr> tag (The tr stands for table row). Typically, a <tr> element is followed by one or more, and at the end of the row a closing </tr> tag is needed.

<td>

Each cell of a table is represented using a <td> element (The td stands for table data). Don't forget the closing </td> tag!

```
<table>
<tr>
  <td> 02 </td>
  <td> 03 </td>
  <td> 05 </td>
</tr>
<tr>
  <td> 08 </td>
  <td> 13 </td>
  <td> 21 </td>
</tr>
<tr>
  <td> 34 </td>
  <td> 55 </td>
  <td> 89 </td>
</tr>
</table>
```



And the `<table>` element marks our last inductee for our Element Hall of Fame! Later throughout the bootcamp, you might become acquainted with more elements like forms, buttons, and input types, but for now, these are the fundamental elements you'll use throughout your project.

Your First Website

Now that you've become acquainted with the HTML syntax, it's time for you to create your very first website! Using any technology Glitch, you can start with the skeleton template below:

HTML

```
<!DOCTYPE html>
<html>
 <head>
     <title> My First Website </title>
   <body>
    <h1> Hello World! </h1>
   </body>
</html>
```

With the elements and material you know, you have the tools to create a captivating webpage.

CHALLENGE

In this challenge, you'll build a pure HTML page from scratch. You'll create a profile page for your immediate neighbor. Some things you can include on the profile page include name, nickname, favorite social media website, short bio, favorite books, shows, movies, etc.

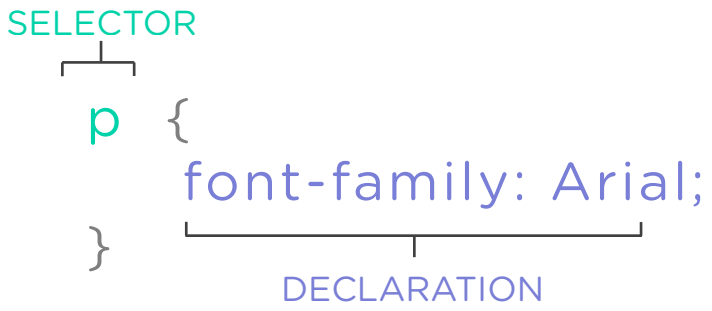
Requirements:

- Skeleton HTML (doctype, html, head, title, body)
- At least two header elements (h1, h3, etc.)
- At least one image
- At least one paragraph
- At least one anchor (a) element linking to another page

CSS

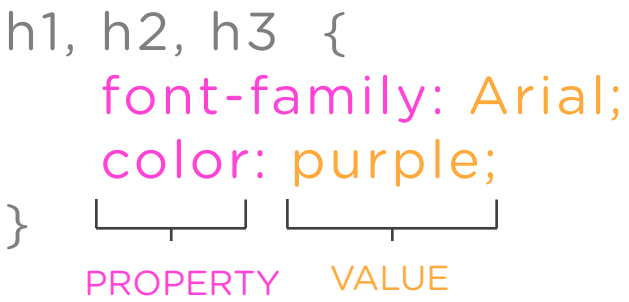
Cascading Style Sheets (CSS) is one of the core technologies for designing and building websites. CSS is a stylesheet language that describes the presentation of web pages, whereas HTML provides the structural foundation. With CSS, you can specify whether the background of your website is pastel pink, or if all paragraphs should use the Didot font, or that all headings should be red— the possibilities are endless.

CSS works by associating style rules with HTML elements. These rules depict how the content of specified elements should be displayed. A CSS rule contains two parts: a selector and a declaration.



Selectors indicate which element the rule applies to, whereas declarations indicate how the elements should be styled. This example indicates that all <p> elements should be shown in the Arial typeface.

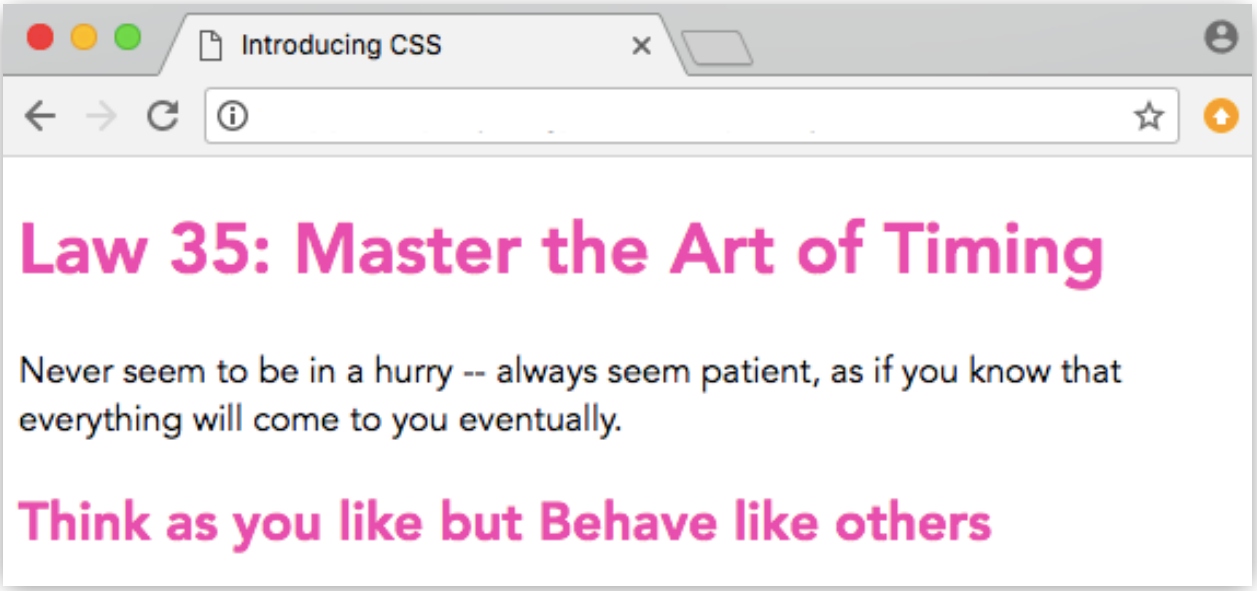
CSS declarations sit inside curly brackets, and each is made up of two parts: a property and a value, separated by a colon. Several properties can be specified in a single declaration, each separated by a semi-colon.



Let's add some internal CSS to a simple web page.

HTML

```
<!DOCTYPE html>
<html>
 <head>
      <title> Introducing CSS </title>
      <style type="text/css">
         body {
              font-family: Avenir;}
         h1, h2 {
              color: #ff69b4;}
      </style>
</head>
   <body>
   <h1> Law 35: Master the Art of Timing </h1>
   <p> Never seem to be in a hurry – always seem patient,
       as if you know that everything will come to you
       eventually.
   </p>
   <h2> Think as you like but Behave like
        others </h2>
  </body>
</html>
```



This example uses a single document: an HTML file. With help from the `<style>` element, we are able to place CSS rules into our HTML page. The `<style>` element uses the **type** attribute, which indicates that the styles are specified in CSS. The value should be *text/css.*

If you are creating a single web page, you might decide to put the CSS rules in the HTML file to keep everything in one place. However, many developers consider it better practice to keep CSS in a separate file. Let's see how we can incorporate an external style sheet into our HTML.

HTML

```
<!DOCTYPE html>
<html>
 <head>
      <title> Using External CSS </title>
      <link href="css/styles.css" type="text/css" rel=
      "stylesheet">
</head>
  <body>
   <h1> Law 25: Re-Create Yourself </h1>
    <p> Do not accept the roles that society foists on you.
        Re-create yourself by forging a new identity, be
        the master of your own image rather than letting
        others define it for you.
    </p>
    <h2> Law 9: Win through your Actions, Never through
         Argument </h2>
   </body>
</html>
```

CSS

```
body {
       font-family:Avenir;
}
h1, h2 {
         color:#ff69b4;
}
```

The `<link>` element can be used in an HTML document to tell the browser where to find the CSS file used to style the page. It is an empty element, and is placed inside the head element. The `<link>` element uses the following three attributes:

href This attribute specifies the path to the CSS file (which is often placed in a folder called css or styles).

type This attribute specifies the type of document being linked to. The value should be text/css.

rel This attribute specifies the relationship between the HTML page and the file it is linked to. The value should be stylesheet when linking to a CSS file.

Using both **internal** and **external** style sheets have several advantages and disadvantages to both of these options. Here are some factors to consider:

- Multiple web pages can share the same style sheet, which is achieved with the `<link>` element on each HTML page of your site to the same CSS document. Unlike internal style sheets, this means that the same code does not need to be repeated in every page.

- If you have one page which requires a few extra rules that are not used by the rest of your site, you might consider using internal style sheets in the same page.

- Convenience: For instance, if you need to update the style of every `<h1>` element, you can do so by modifying the one CSS style sheet, rather than changing the CSS rules on every page.

- HTML code will be easier to read and edit without CSS rules in the same document, and will have a lot less code.

Now that you've had the change to become familiar with CSS rules, let's practice adding some color to our elements.

COLOR

The color property is used to set the color of the text. The color can be specified by:

RGB values

RGB values refer to a system that expresses how much red, green, and blue are used to make up a color. Example: `rgb(255, 59, 162)`

HEX values

HEX values are six-digit number that represent the amount of red, green, and blue in a color preceded by a pound sign (#). Example: `(#C90D00)`

Color name

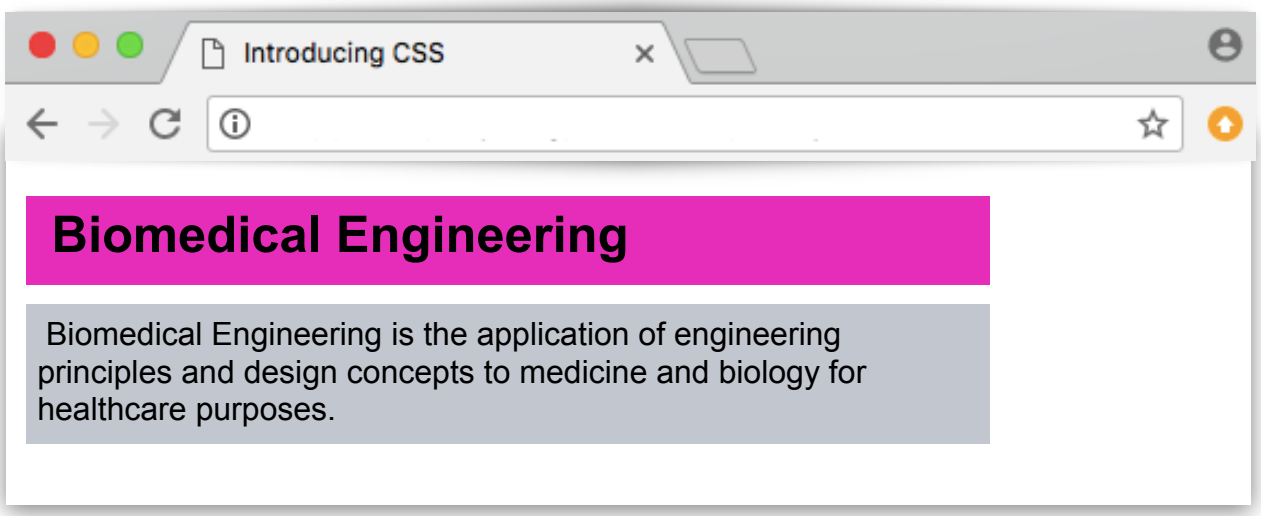
There are a set of roughly 150 predefined color names that are recognized by browsers. For example: `yellow`.

You can use any of the three ways to specify a color for a property.

CSS

```css
p {
    background-color:#c2c6ce;
}

h1 {
    background-color: rgb(229,45,186);
}
```



In the example above, we are able to to change the color of a property using both a *HEX code* and *RGB* value. If you don't know the value of a specific color, you can always use a color picker online.

The formatting of text on a web page can have a significant effect on how readable your pages are. When choosing a typeface, there are several ways to use fonts other than the one thats used with standard HTML.

The font-family property allows you to specify the font that should be used for any text inside an element. The value of this property is the name of the typeface you want to use. If a name is made up of more than word, be sure to use double quotes!

font-family

When choosing typefaces, it is important to remember that the people who are visiting your website need the typeface you have specified installed on their computer in order for it to be displayed.

Considering this factor, you can specify a list of fonts separated by commas so that, if the user does not have your preferred choice of typeface installed, you'll have a backup of your choosing.

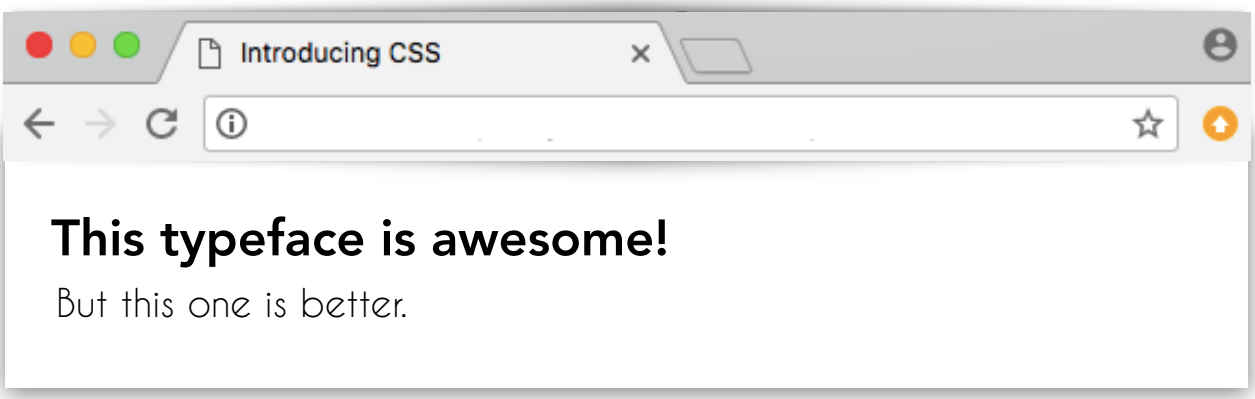
Some typefaces that most computers support include:

Georgia
Times
Times New Roman
Arial
Courier
Courier New
Comic Sans
Snell Roundhand

CSS

```css
body {
    font-family: Avenir, Verdana;
}

p {
    font-family: "Caviar Dreams", Impact;
}
```



In the example above, the typefaces displayed are "Avenir" and "Caviar Dreams", however, if a user did not have these typefaces installed on their computers, "Verdana" and "Impact" would have been substitutes.

To ensure a particular font of your choosing is available on your website, let's introduce our newest addition to the toolkit: **Google Fonts.**

Google Fonts is a library composed of over 900+ different fonts that you can import into your HTML/CSS code. By using Google Fonts, you'll have access to a wide range of fonts that you can use for your website without worrying if a user has your preferred font installed on their computer. Let's add Google Fonts to an HTML file!

To add Google Fonts to your HTML file, refer to the following steps:

1. Visit https://fonts.google.com/
2. Browse the directory of different fonts and select the one you would like to add to your website by pressing the + button. In this example, we will be selecting the font Playfair Display.

3. Navigate to the bottom of the screen and press on the button that states “1 Family Selected”.

4. Copy the code below into the <head> of your HTML document.

```
<link href="https://fonts.googleapis.com/css?family=Playfair+Display&disp
lay=swap" rel="stylesheet">
```

HTML

```
<!DOCTYPE html>
<html>
 <head>
     <title> Using External CSS </title>
     <link href="css/styles.css" type="text/css" rel=
     "stylesheet">
     <link href="https://fonts.googleapis.com/css?family=
      Playfair+Display&display=swap" rel="stylesheet">
</head>
```

5. Lastly, specify the font-family for the element you would like to style in your CSS document.

```
font-family: 'Playfair Display', serif;
```

CSS

```
body {
    background-color: pink;
}

h1 {
    font-family: 'Playfair Display', serif;
}
```

---

font-size

The `font-size` property allows you to specify the size that should be used for any text inside an element. There are several ways to specify the size of a font including pixels (px), percentages (ex: 75%, 150%), and ems.

CSS

```
h1 {
    font-size: 100px;
}

p {
    font-size: 5em;
}
```

font-weight

The `font-weight` property allows you to create bold text. There are two values that this property commonly takes: normal and **bold.**

text-align

The `text-align` property specifies the horizontal alignment of text in an element. Some common values for this property include left, right, and center.

text-shadow

The `text-shadow` property adds shadow to text. This property requires at least three values: horizontal-shadow, vertical-shadow, and color.

Now that we've introduced a few CSS properties we can change, let's practice with a heading element and see how this changes its default styling.

CSS

```
h1 {
    font-family: 'Playfair Display', serif;
    font-size: 5em;
    text-align: center;
    text-shadow: 5px 5px pink;
}
```

This is our result:

POINCARÉ CONJECTURE

Borders

CSS border properties allow you to create and specify the style, width, and color of an element's border. Here are the different border properties you will need to use:

`border-style`

The `border-style` property specifies what kind of border to display. There are several values allowed for this property, including:

```
border-style: dotted;
```

I am a dotted border!

```
border-style: dashed;
```

I am a dashed border!

```
border-style: solid;
```

I am a solid border!

```
border-style: double;
```

I am a double border!

```
border-style: inset;
```

I am an inset border!

And there are several other border styles you can use for your website—feel free to experiment and search for other styles that are most appropriate for your website.

border-color

The border-color property specifies what color is used to set the color of the four borders.

```
border-color: #9e9fff;
```

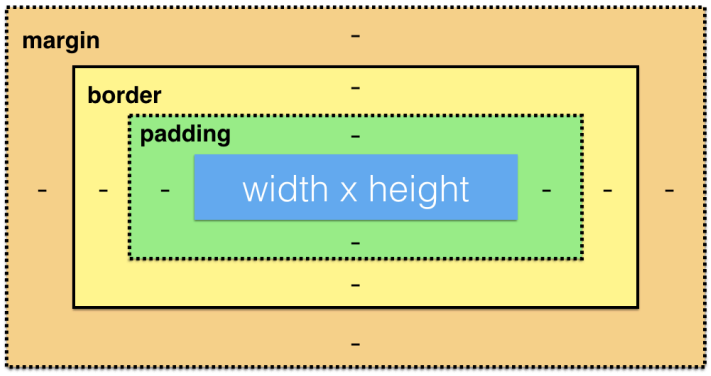
This is a single color border!

```
border-color: red green blue yellow
```

This is multi-colored border!

Margins and Padding

Margin properties in CSS are used to create space around elements, outside of any defined borders. There are properties for setting the margin for each side of an element (top, right, bottom, and left).



The CSS properties to modify the margins of each elements side are: `margin-top`, `margin-bottom`, `margin-left`, and `margin-right`.

Let's apply some of these properties to a sample paragraph and see how they transform the element!

CSS

```css
p {
    margin-top: 100px;
    margin-bottom: 100px;
    margin-left: 80px;
    margin-right: 150px;
}
```

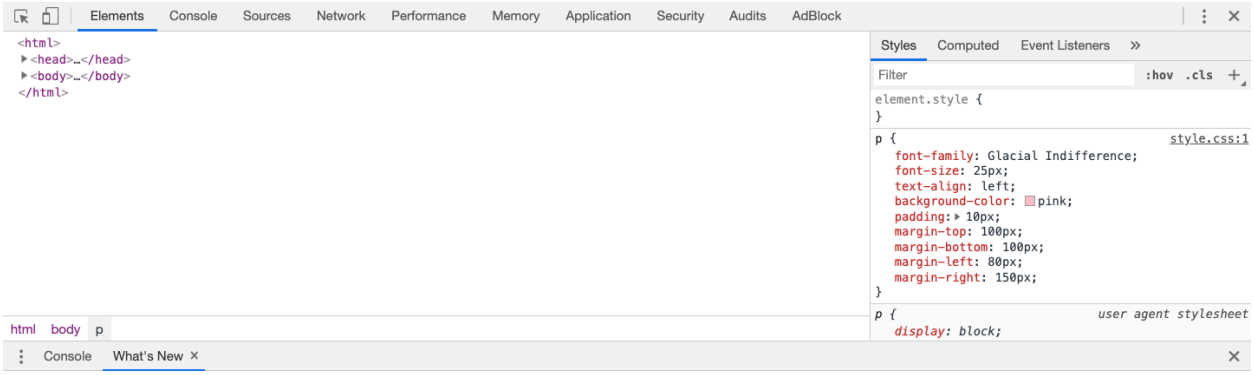
This p element has a top margin of 100px, a right margin of 150px, a bottom margin of 100px, and a left margin of 80px.

Viewing the element in your browser may be a little difficult, so we'll introduce a new addition to your toolkit: Google Chrome **Inspect**.

Google Chrome's Inspect Element is a developer tool that allows developers like you to view and modify CSS and JavaScript elements. This tool will be especially helpful when we begin to create *dynamic* webpages! To access Google Chrome Inspect, you'll need to:

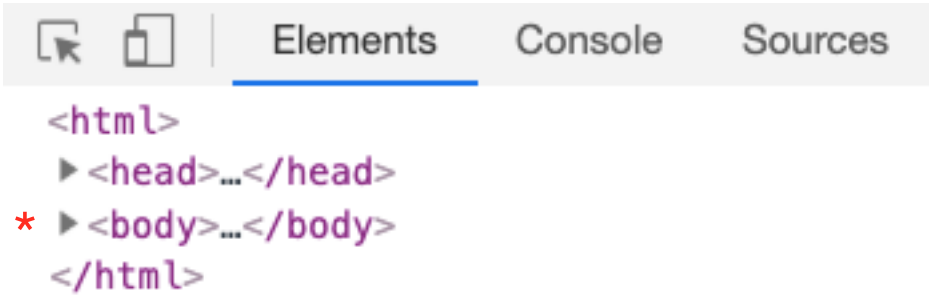
- Right click on the web page you would like to inspect
- Select "Inspect" from the dropdown menu

From here, you should see something that looks similar to this:

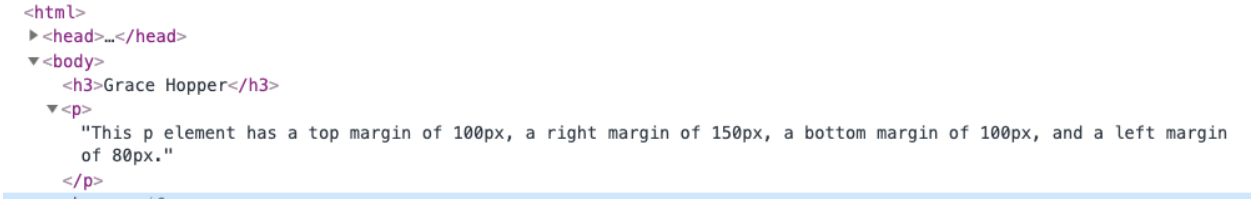


The purpose of using the Inspect element at this moment is to observe our elements now that we've added margins. To do this, we'll need to select the element that needs to be inspected in our code.

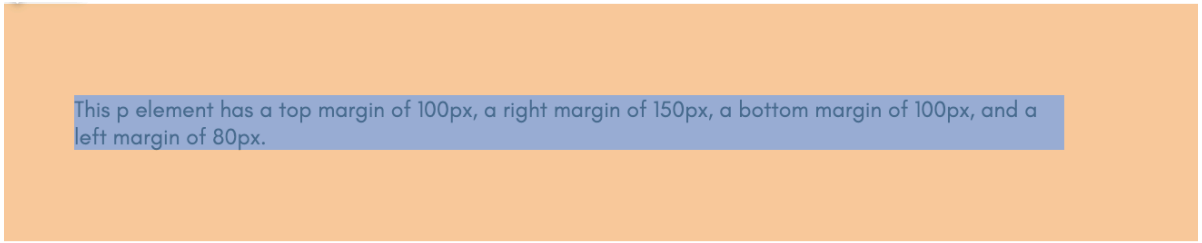
In your Inspect window, press on the light grey arrow next to the opening tag.



This will expand and display all the contents inside the pressed element, and this particular example, we can see that inside our body tag we have one <h3> and one <p> element.



To observe the elements we've added margins to, all we need to do is hover over the particular element with the cursor to see the margins.



Perfect! Now we are able to view our margins with the Google Chrome Inspect tool and visualize the space around our elements.

Another property available to specify margin values that requires less code is the `margin` property, which takes four values: top, right, bottom, and left.

CSS

```
p {
    margin: 100px 150px 100px 80px;
}
```

This property is the same as specifying the margins for each side:

CSS

```
p {
    margin-top: 100px;
    margin-bottom: 100px;
    margin-left: 80px;
    margin-right: 150px;
}
```

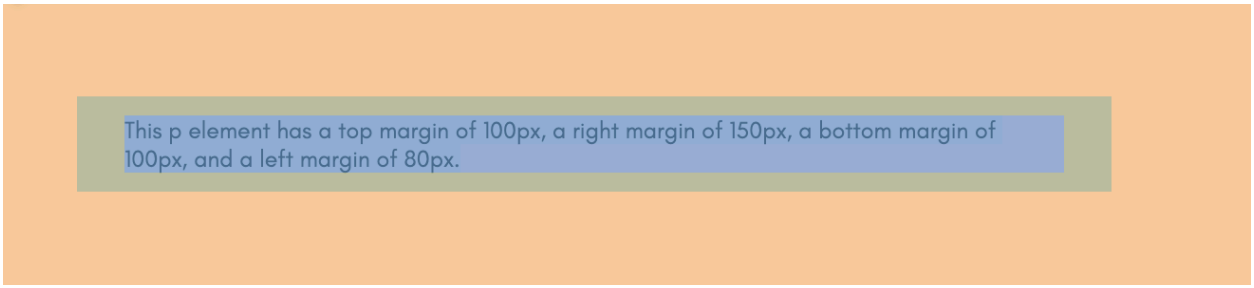
CSS padding properties are used to generate space around an element's content, inside of any defined borders. The primary difference between margins and padding is that margins add space outside an element, and padding adds space inside an element.

Similar to margins, there are properties to add padding for each side of the element with the following properties: `padding-top`, `padding-bottom`, `padding-left`, and `padding-right`.

A shortcut for adding padding on all sides is using the `padding` property with the four values in the following order: top, right, bottom, left. Let's add some padding to our paragraph element and see what it looks like when we inspect it in Google Chrome.

```css
p {
    padding: 20px 50px 20px 50px;
}
```

This p element has a top margin of 100px, a right margin of 150px, a bottom margin of 100px, and a left margin of 80px.



In the figure above, the orange space represents the margins and the green space represents the padding.

Class & ID Selectors

While we've been exploring CSS, we've primarily styled elements (headings, paragraphs, images, etc.) using element selectors. Before we move on to the next section of this bootcamp covering CSS Grid and JavaScript, we'll need to become familiar with class and id selectors. As a refresher, let's understand what selectors are and how they operate in CSS.

Selectors indicate which element the rule applies to, whereas declarations indicate how the elements should be styled.

The ***id selector*** uses the id attribute of an HTML element to select a specific element. The id of an element should be unique within a page, and is only used to select one unique element! To select an element with a specific id, write a hash (#) character, followed by the id of the element.

HTML

```
<h1 id="name-miranda"> Miranda </h1>
<h1> Emma </h1>
<h1> Anahi </h1>
```

CSS

```
h1 {
    color: purple;
}

#name-miranda {
    color: pink;
}
```

In the example above, we have three <h1> elements and they will all be the color purple— but since Miranda's name has an ID that is styled differently in the CSS, her name will be pink. Let's see how this will look in a webpage!

Miranda

Emma

Anahi

The important thing to note with ID selectors is that they:

- Begin with a hashtag/pound sign (#)
- Are used to modify an element once

The ***class selector*** uses the class attribute of an HTML element to select a specific element. An element's class should be unique within a page, and unlike id selectors, is used to style several elements at once! To select an element with a specific class, write a period (.) character, followed by the class name of the element.

In our HTML below, we have several `<h1>` elements with the names of the students enrolled in this bootcamp. Let's style only the names that start with the letter A by giving them a unique class name, then styling that class in our CSS, and finally seeing what our result looks like in our page.

HTML

```
<h1 class="a-team"> Aislyn </h1>
<h1 class="a-team"> Anahi </h1>
<h1 class="a-team"> Anahí </h1>
<h1 class="a-team"> Anapaula </h1>
<h1 class="a-team"> Andrea </h1>
<h1> Clarissa </h1>
<h1> Daisy </h1>
<h1> Daphne </h1>
<h1> Emma </h1>
<h1> Frida </h1>
```

CSS

```
h1 {
    color: blue;
}

.a-team {
    color: red;
}
```

Let's observe the final result in our web page:

As we can see from results above, every element with the unique class name has been styled differently even though they are all `<h1>` elements.

HTML `<div>`

Before we can construct website layouts with CSS grid, we'll need to add one more tag to the Element Hall of Fame: the `<div>` element.

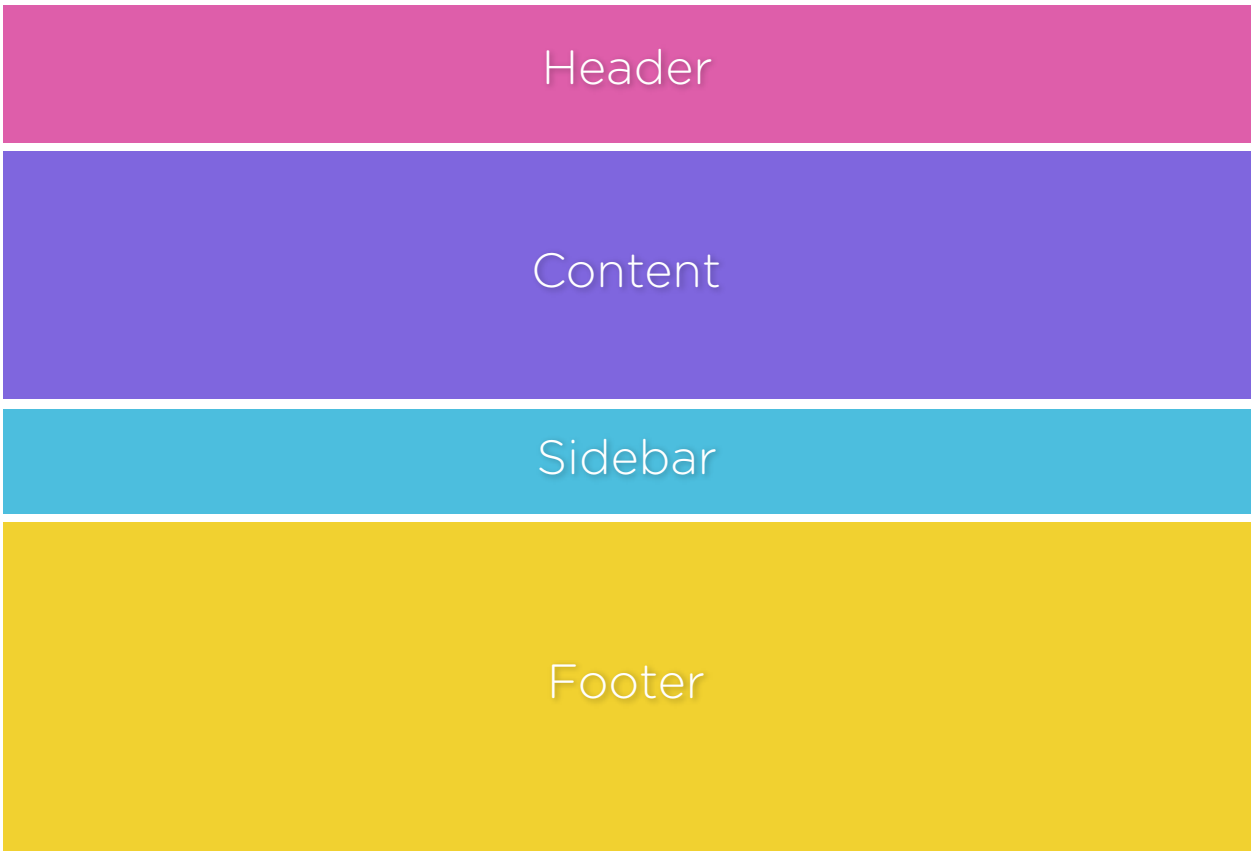
The `<div>` tag defines a division or a section in an HTML document. It's often used as a container for other HTML elements to style them with CSS or to perform certain tasks with JavaScript. Let's look at some sample code:

HTML

```
<div class="about-section">
     <h3> I'm a header inside a div! </h3>
     <p> I'm a paragraph inside a div! </p>
</div>
```

CSS Grid

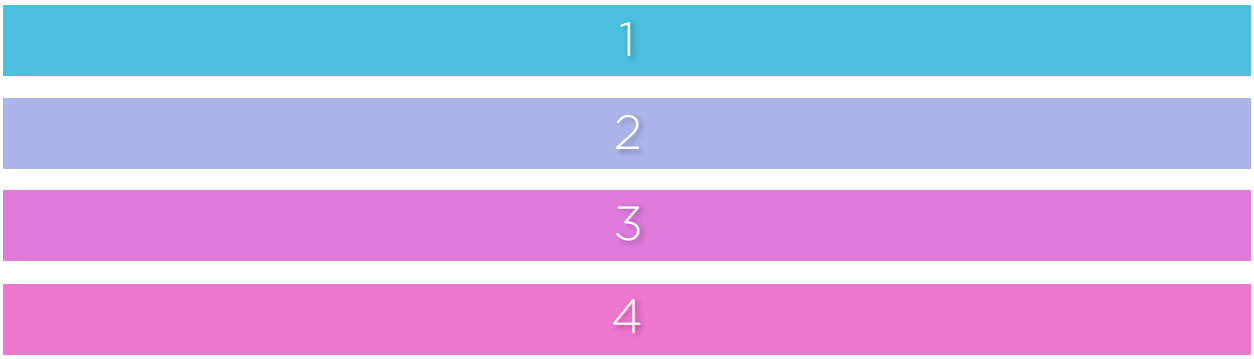
With all the tools that we've learned yesterday, we're able to layout our elements in the following style.



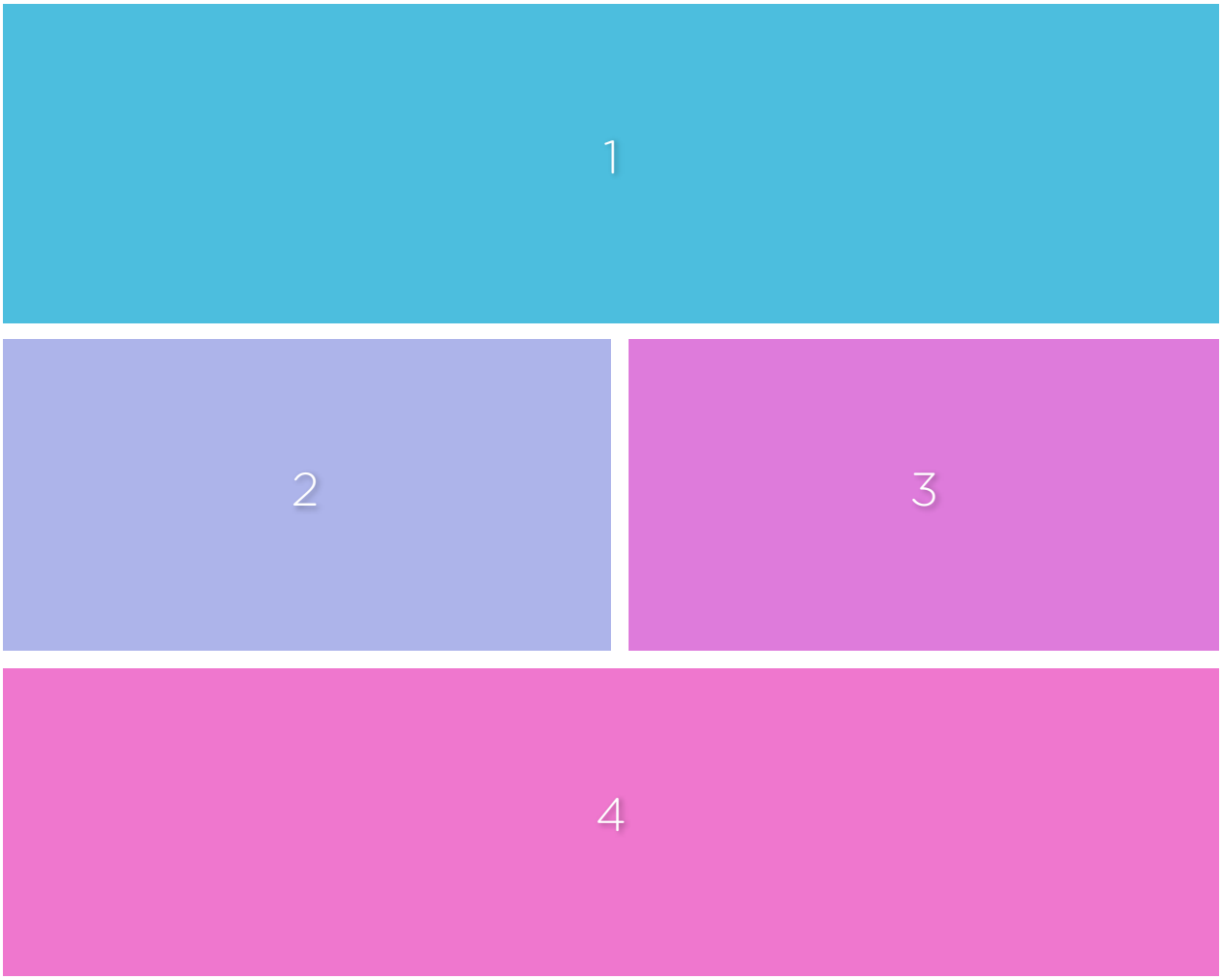
What separates a good website from a bad website is the ability to structure the contents of your website in the most approachable way to your users. The website must be clean with an easy-to-follow navigation system to contribute to a usable webpage layout.

A layout that is easy to follow will give the site's visitor easy access to the valuable and important information on your website. When content is difficult to find on a webpage, visitors get agitated and choose to leave the site with the possibility of not returning.

CSS Grid is a technique in CSS that allows developers to create responsive web design layouts more easily and consistently across browsers. With CSS Grid, we can transform our elements from this layout:



to this layout:



Let's begin! To create a grid, we'll first need to create a `<div>` element with the class name grid. The `<div>` tag is used as a container unit that divides the HTML document into sections. Web developers (yes, that's you included!) use `<div>` elements to group together HTML elements and apply CSS styles to many elements at once.

HTML

```
<div class="grid">

</div>
```

Fantastic! This `<div>` element states that in our webpage, we have a designated section named grid. Now, it's time for us to add items inside our grid that we'll use to style. We'll do this by adding smaller sections with `<div>` elements and give them individual IDs.

HTML

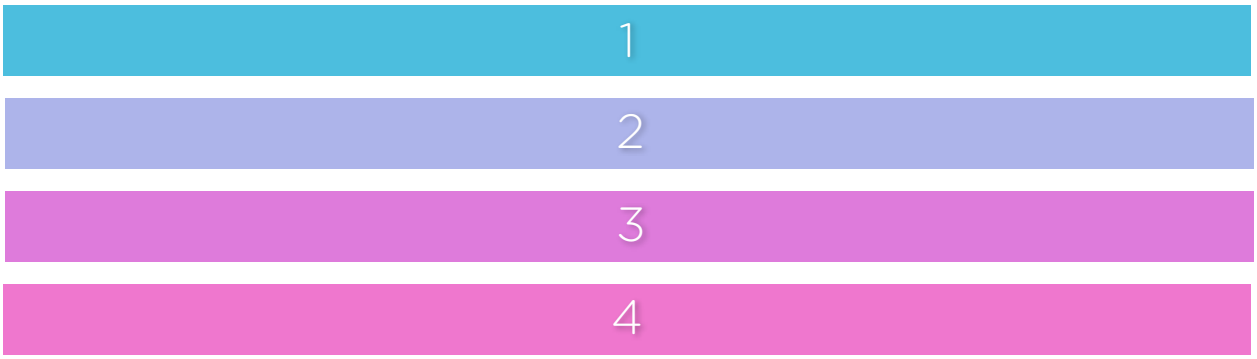
```
<div class="grid">
     <div id="item-1"> 1 </div>
     <div id="item-2"> 2 </div>
     <div id="item-3"> 3 </div>
     <div id="item-4"> 4 </div>
</div>
```

At this point, we have one large container full of smaller containers that make up a grid. Let's turn our container into a grid by modifying it in our CSS.

CSS

```
.grid {
   display: grid;
}
```

Now, let's see what our grid looks like!

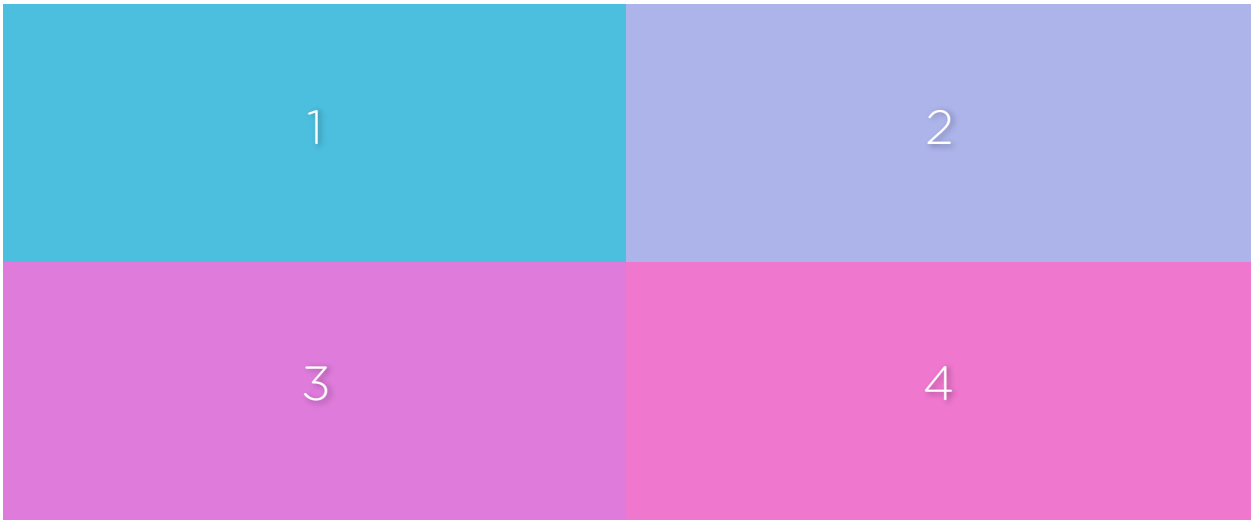


Because we haven't defined any properties yet in our grid like rows or columns, all of the elements inside of it will remain stacked on top of each other. Let's change this by adding two new properties to our grid: `grid-template-columns` and `grid-template-rows`.

CSS

```
.grid {
      display: grid;
      grid-template-columns: 100px 100px;
      grid-template-rows: 50px 50px;
}
```

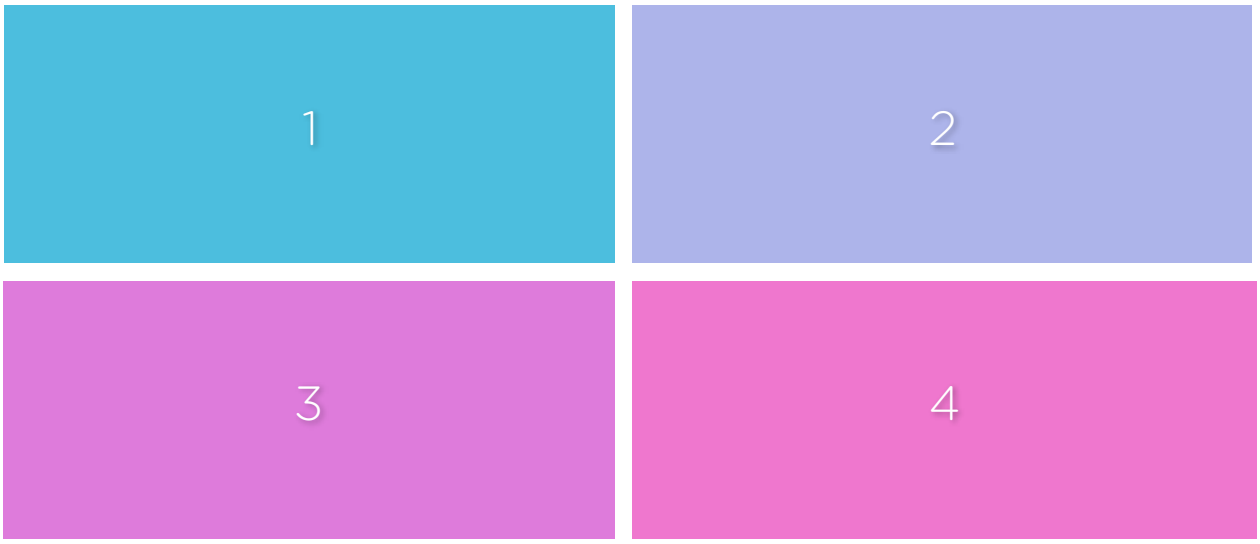
Each value provided to the `grid-template-columns` and `grid-template-rows` properties dictate how wide we want our columns (100px) and how tall we want our rows (50px).



Excellent! Finally, let's create a small gap between the elements in the grid with the `grid-gap` property.

CSS

```css
.grid {
    display: grid;
    grid-template-columns: 100px 100px;
    grid-template-rows: 50px 50px;
    grid-gap: 10px;
}
```



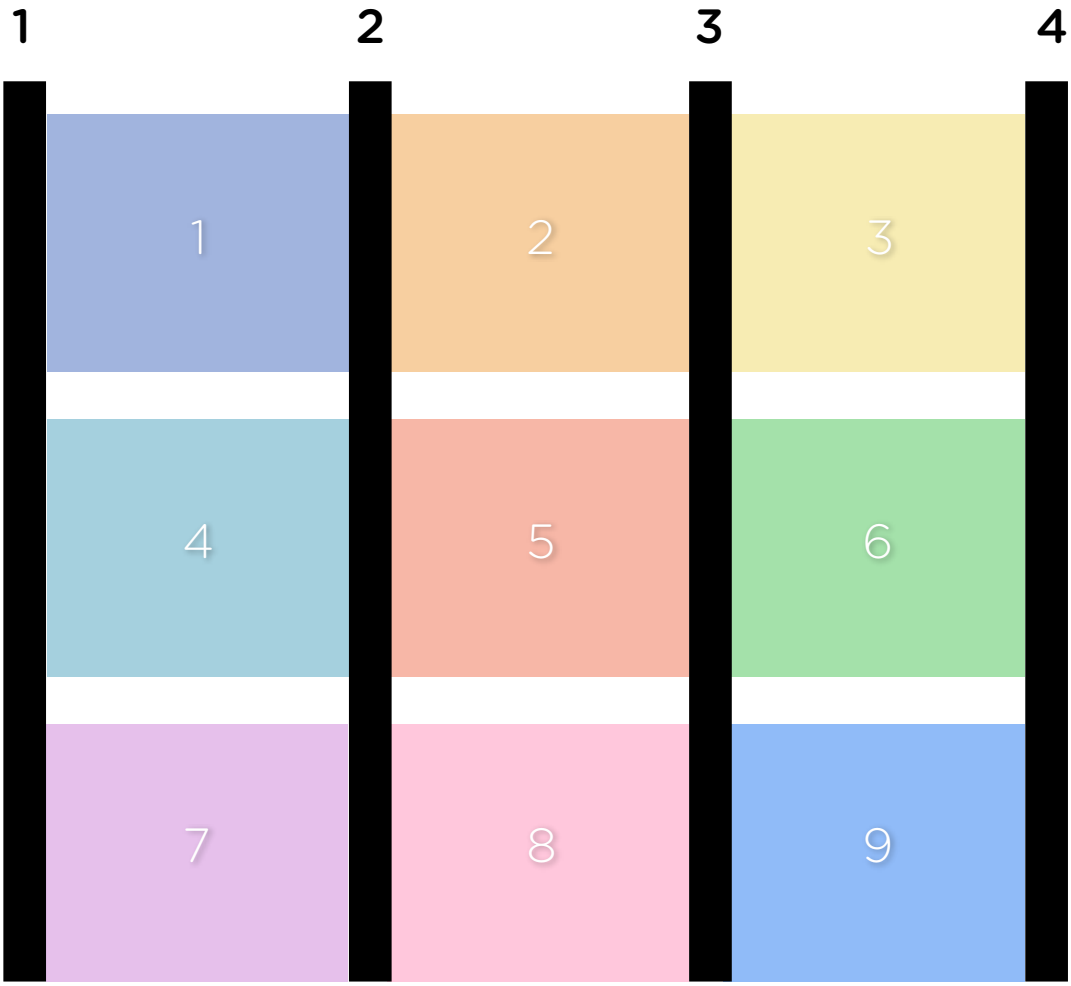
CHALLENGE

Using the same CSS properties with different values and adding more elements to your grid in your HTML, try creating a 3x3 grid!

Hint: you'll need three columns and three rows of identical values each.

Now that we've seen how to create two dimensional layouts with CSS grid, let's position and resize the items in our grid to create different layouts!

In addition to grid rows and columns, we also have grid ***lines*** that identify the start and end of a column.



From the figure above, we can see that although we have 3 columns, we have 4 grid lines where:

- Column 1 *starts* at line 1 and *ends* at line 2
- Column 2 *starts* at line 2 and *ends* at line 3
- Column 3 *starts* at line 3 and *ends* at line 4

To create multi-dimensional layouts, all we'll need to do is modify where each element in our grid starts and ends. Let's modify our layout so that the first item in our grid spans across all three columns.

HTML

```
<div class="grid">
     <div id="item-1"> 1 </div>
     <div id="item-2"> 2 </div>
     <div id="item-3"> 3 </div>
     <div id="item-4"> 4 </div>
     <div id="item-5"> 5 </div>
     <div id="item-6"> 6 </div>
     <div id="item-7"> 7 </div>
     <div id="item-8"> 8 </div>
     <div id="item-9"> 9 </div>
</div>
```

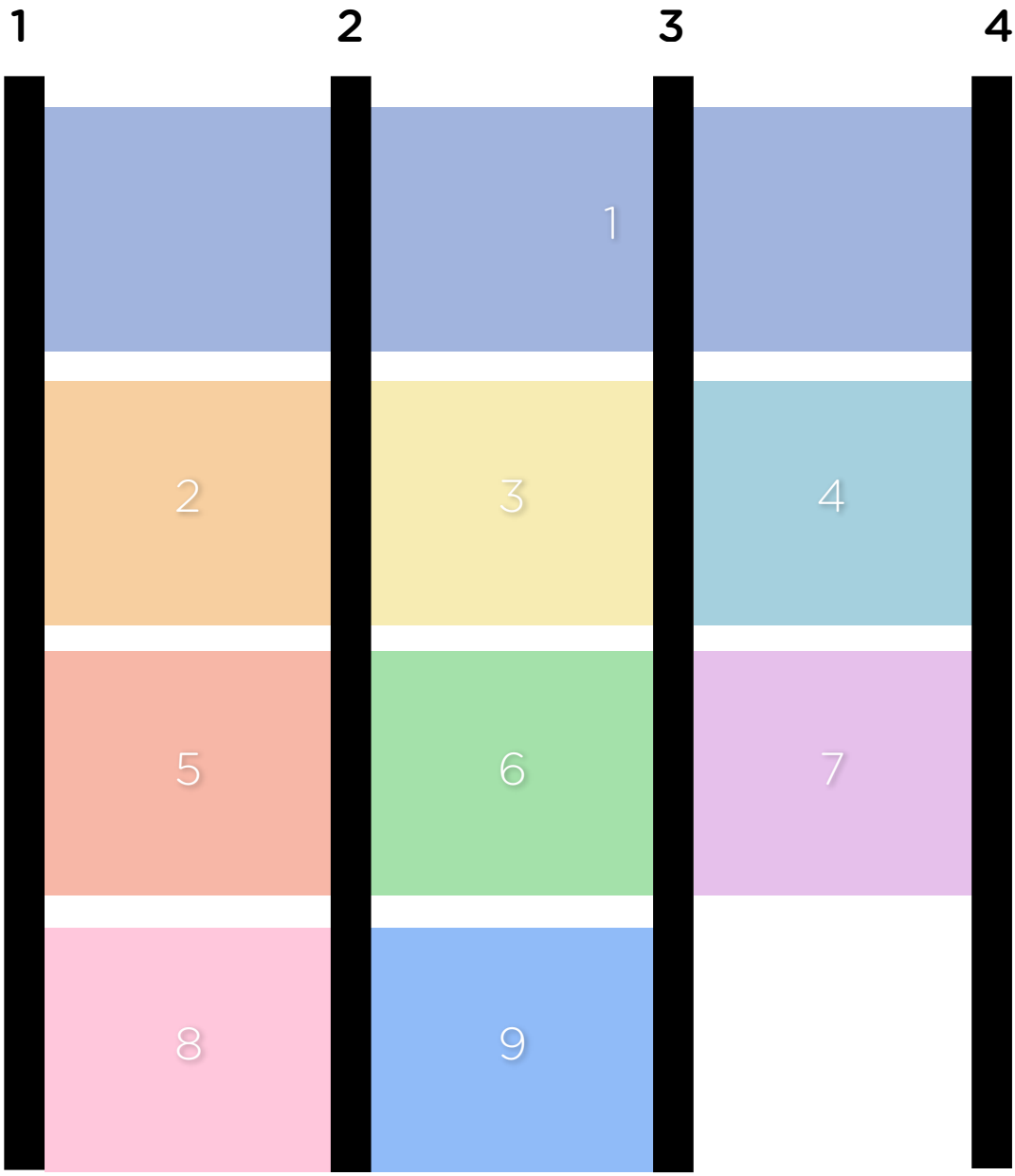
CSS

```
.grid {
   display: grid;
   grid-template-columns:300px 300px 300px;
   grid-template-rows:300px 300px 300px;
   grid-gap:15px;
}

#item-1 {
   grid-column-start:1;
   grid-column-end:4;
}
```

In our HTML file, we've created a `<div>` element with the class name grid that contains nine unique elements and matching ids inside the grid.

In our CSS file, we've styled our .grid class so that we have 3 columns that are 300px wide and 3 rows that are 300px tall. We've also specified that the first element in our grid should span over 3 entire columns by specifying where the column should start and end. Let's see what this grid looks like!



Since the first item is spanning across all 3 columns, all the other elements in the grid have been pushed over to the next available column.

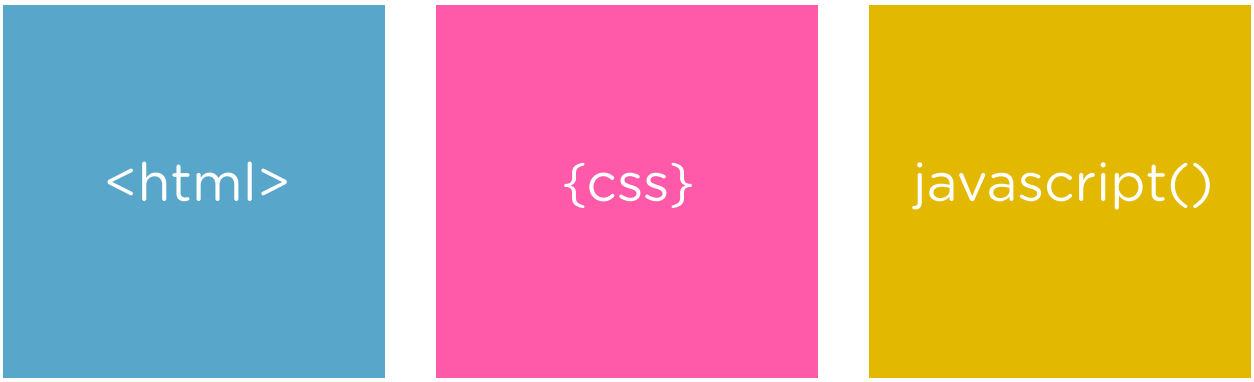
And that's the basics of CSS grid! With the properties we've gone over, you now have the availability to create multi-dimensional layouts that will best suit your website.

JAVASCRIPT

So far, we've created a content layer using HTML and a presentation layer using CSS.

Starting with the HTML layer, we are able to focus on the most important thing on a webpage: the content. This layer should be accessible to all users, available on all devices, and load in a timely manner.

Adding CSS rules in a separate file keeps rules regarding how to webpage is styled away from the content itself. Using the same stylesheet throughout your webpage will make it easier to load and maintain.



CONTENT PRESENTATION BEHAVIOR

Finally, we have our behavior layer that uses **JavaScript**, an object-oriented programming language that allows developers to create interactive effects within web browsers. With JavaScript, we can enhance the usability and and experience of the webpage. Like CSS, keeping JavaScript in a different file is good practice. By separating all files, the page will still work even if a user cannot load or run the JavaScript.

In this last section, you'll learn how to read + write JavaScript, and learn how to give a web browser instructions you want it to follow. Let's begin by understanding *how* we can use JavaScript to access and change the elements of an HTML document.

When a web page is loaded, the browser creates a **Document Object Model** (**DOM**) of the page. This HTML DOM is a standard for how to access, change, add, and delete HTML elements.

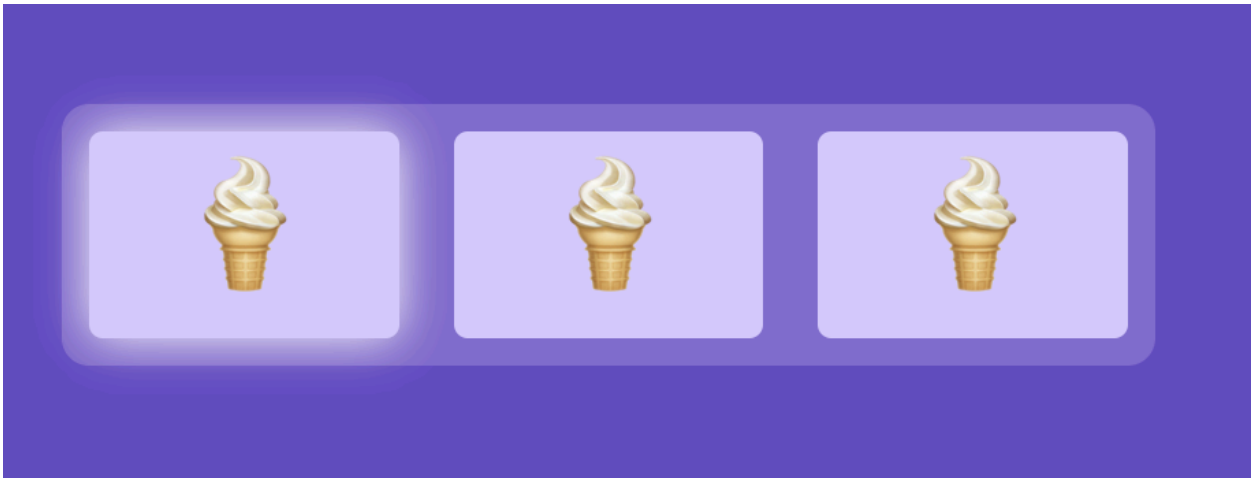
Before we begin writing any JavaScript, let's first create a JavaScript file and link it to our HTML document.

HTML

```
<!DOCTYPE html>
<html>
  <head>
    ...
  </head>
  <body>
    ...
    <script type="text/javascript" src="script.js">
    </script>
  </body>
</html>
```

In this example, we're telling our HTML document that we'll be using a JavaScript file named script.js. If you have a different name for your JavaScript file, just be sure to modify that in your HTML document.

Now we're ready to access and change our HTML elements!



HTML

```
<div class="container">
    <div id="ice-cream1" class="item dessert active”>🍦
     </div>
    <div id="ice-cream2" class="item dessert">🍦</div>
    <div id="ice-cream3" class="item dessert">🍦</div>
</div>
```

What we have above is a container `<div>` element that is holding three other `<div>` elements, where one of them has a class “active” that is styled to look like it is glowing.

CSS

```
.active {
   box-shadow: 0 0 35px white;
}
```

Our goal for this section is have the ability to click on any of the three boxes with ice cream in it and *toggle* the active class. Let's begin!

The first thing we'll need to do is identify what element we'd like to modify and select it on our JavaScript. In our case, we would like to have the ability to add and remove the glowing styling when we press on one of the ice-cream `<div>` elements.

First, we'll need to select ice-cream1 with the following command:

JS

```
let ice_cream1 = document.querySelector("#ice-cream1");
```

Let's dissect the two different parts to the command above starting with part one:

document.querySelector("#ice-cream1");

In the snippet above, we are specifying the exact element we'd like to change by its id name. Let's take a look at part two:

let ice_cream1 =

This specific command is creating a local JavaScript variable that is assigned to the value of #ice-cream1. Creating variables while coding is similar to math where you'd use the letter x to store a value of numeric value (e.g. x = 5), but in this case, we're using the variable ice_cream1 to store the element that has the id `#ice-cream1`.

Now that we've selected our element in JavaScript, we'll now focus on adding the class name "active" whenever we click our element. To have this ability, we'll need an ***eventListener***.

JS

```
ice_cream1.addEventListener("click", e => {

});
```

In the code snippet above, we're specifying what *event* or *action* needs to happen to our element. Our code says that whenever we "click" on the first ice cream element, we'll do something. Here are some other events to listen for:

- click
- mouseover
- mouseclick
- scroll

Our eventListener has nothing inside it's curly brackets, meaning whenever someone clicks on the first ice cream, nothing will happen. Let's change that so that whenever we click on the ice cream, we can add the class "active" to it!

JS

```
ice_cream1.addEventListener("click", e => {
  ice_cream1.classList.toggle("active");
});
```

Our final snippet is the *action* that takes place whenever the event occurs— so this code block is saying whenever we *click* on our first ice cream, we'll *toggle* (turn on) the class active. Now let's see what our entire JavaScript file looks like!

JS

```
let ice_cream1 = document.querySelector("#ice-cream1");

ice_cream1.addEventListener("click", e => {
  ice_cream1.classList.toggle("active");
});
```

Let's do a quick overview of our code:

1. We select our HTML element through it's given id
2. We assign the selected HTML element to a JavaScript variable
3. We specify what event needs to happen
4. We specify what action needs to take place when the event occurs

And with this code snippet, we are able to successfully click on our ice cream and turn the active class "on and off"! Woohoo!

CHALLENGE

Now that we've been able to toggle the active class for our first ice cream element, try adding it to the other two ice cream elements! Refer to the four steps we followed above this box if you're stuck, or just need a soft reminder of what we've done. Good luck!

This week has been an incredible journey. Some of you were familiar with code, but the majority were not and now, you all have the knowledge and abilities to create your very own webpage!

It was such a joy to meet and get to know each of you, and we're incredibly excited to see what you'll create by the end of the week. Thank you for joining us this week and have a wonderful, safe rest of your summer 💕 😄