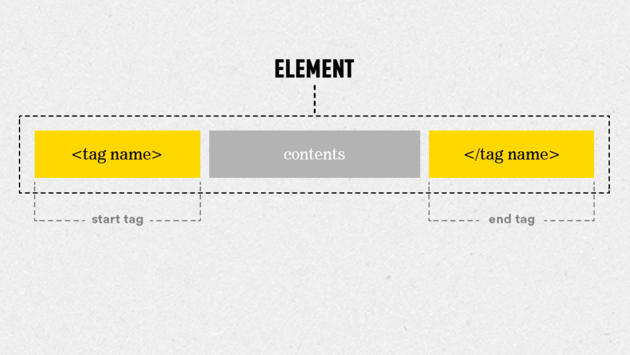
**Review Guide: Introduction to HTML and CSS**

Let’s review some of the key ideas introduced in this unit. You can also download all this content for future reference by clicking on the attached file.

**Key Definitions**

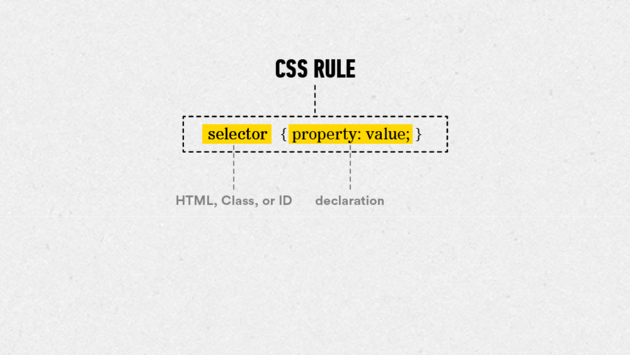
* **World Wide Web:** An information ­sharing model built on top of a global system of **inter**connected computer **net**works called the Internet.
* **Websites:** Collections of files, accessible through the world wide web.
* **Clients:** Personal devices such as computers, phones, or tablets, used to access the web.
* **Servers:** Special computers that host massive amounts of the files that make up websites. They “serve” those files to end users.
* **HTML Elements:** The building blocks of HTML. Consists of a start tag, content, and end tag.
* **HTML Tags**: HTML syntax used to indicate to a web browser how to present content. Bookended by angle brackets.



* **HTML Attribute:** Adds information to an HTML element. An HTML tag can have one or many attributes. Attributes are always included in the opening tag. They are preceded by a space, include the name of the attribute, an equal sign, and a value in quotes.



* **CSS Rule:** The building block of a CSS stylesheet. A rule consists of a selector and a declaration block (one or more declarations).
* **CSS Declaration:** A declaration is made up of a property name and a value, separated by a colon.
* **CSS Selector:** The actual HTML object the declaration(s) apply to.
* **CSS Property:** The characteristic of the selector that will be changed.
* **CSS Value:** The amount to change corresponding property of the matched selector.



HTML vs. CSS

So what’s the difference between HTML and CSS? And when do I use each?

**HTML**, or **H**yper **T**ext **M**arkup **L**anguage, is the standard language used to code web pages. All of the content on a website is added with HTML.

**CSS** or **C**ascading **S**tyle **S**heets, decides how the content written in an HTML document looks, such as the layout, colors, and fonts.

**Common HTML Elements**

**Headline Elements:** are meant to be used for important text that titles the content that comes after it. They range from —the largest—to —the smallest.

<h1>This would make a big headline</h1>

**Paragraph Elements:** are one of the most basic tags in HTML. They indicate blocks of text.

<p>This would add a block of text to a page</p>

**Anchor Elements:** are the tags we use to create links. In order to make a working link, we have to add more information to the anchor tag, using an href attribute.

<a href="www.generalassemb.ly>This would make this whole

sentence a link to General Assembly's home page.</a>

**HTML Boilerplate**

In order to organize tags properly, start with a set of structural elements called the HTML boilerplate. It should look like this:

***<!DOCTYPE html>***

<html>

<head>

</head>

<body>

</body>

</html>

**Common CSS Properties and Values**

**Property #1:**  
**color —** This CSS property allows you to adjust the *text* color of the selected element.

For example, to turn the text of an element green, you would create the following CSS rule:

h1 {

**color:** #00ff00;

}

**Property #2:**  
**background-color —** This CSS property allows you to adjust the *background* color of the selected element.

For example, to turn the background of a web page yellow, you would enter the following CSS rule:

body {

background**-color:** #ffff00;

}

**Common Values:**  
There are 17 standard color values that you can pair with the color and background-color properties — aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white, and yellow.

Additionally, there are 123 more colors that are generally accepted as standard. You can find the entire list [here](http://www.w3schools.com/cssref/css_colornames.asp).

You can also create custom colors using RGB and hexadecimal codes, which we'll discuss in Unit 3.

**Questions to Ask Your Mentor**

1. How did you first get introduced to HTML and CSS?
2. Can you show me some examples of websites with basic HTML and CSS code?
3. What are your favorite web designers and websites?
4. Before you sit down to create a website, what steps do you take to prepare?
5. What are some good resources that you typically use for learning about web design?

**Resources**

* [Sublime Text](http://www.sublimetext.com/3) — A free text editor used for building HTML and CSS files. It's available for both Mac OS X and Windows.
* [Google Chrome](https://www.google.com/chrome/browser/desktop/index.html) — The primary web browser used to demonstrate rendered code throughout this course. It's available for both Mac OS X and Windows.
* [wireframe.cc](https://www.wireframe.cc/) — A free tool for creating web design mockups. It operates in your web browser, so you don't even need to download it. You'll learn how to create design mockups with wireframe.cc in our next unit.

**Further Reading**

Looking for more help with the Basics of HTML? Here are a couple places you can check out:

More basics on HTML and CSS (read up until the section called "In Practice"):  
<http://learn.shayhowe.com/html-css/terminology-syntax-intro>

More info on getting started:  
[http://www.htmldog.com/guides/html/beginner/gettingstarted](http://www.htmldog.com/guides/html/beginner/gettingstarted/)

Intro to HTML tags:  
<http://www.htmldog.com/guides/html/beginner/tags/>

Intro to title elements:  
<http://www.htmldog.com/guides/html/beginner/titles/>

Intro to paragraph elements:  
<http://www.htmldog.com/guides/html/beginner/paragraphs/>

Intro to heading elements:  
<http://www.htmldog.com/guides/html/beginner/headings/>

The first web page ever posted on the Web:  
<http://info.cern.ch/hypertext/WWW/TheProject.html>

Also, please feel free to reach out to your mentor to discuss any of the topics in this unit further.

[←](https://circuits.generalassemb.ly/student/209/pages/785?direction=back) [→](https://circuits.generalassemb.ly/student/209/quizzes/232)

**Downloads**

[Summary](https://ga-sprites.s3.amazonaws.com/uploads/production/page/downloadable_content/137/Summary.zip)

© 2016 General Assembly. All rights reserved. [Regulatory Information](https://generalassemb.ly/regulatory-information) [Terms](https://generalassemb.ly/terms_and_conditions) [Privacy](https://generalassemb.ly/privacy_policy) [Assembled](http://nytm.org/made-in-nyc)

# Project: Part 1 — Creating Your First Web Page

Welcome to your first Circuits project!

Throughout the course, we’ll guide you through the creation of your very own website. The content of your website will revolve around a business. It can be a real business that you own or work at, a fictional business that you'd like to create, or a personal brand page.

If you’re interested in seeing some previous students' final projects to get some inspiration, you can do so [here](https://gallery.generalassemb.ly/WDC?metro=).

In this week's project, you'll create a basic web page and host it online. The following steps will help guide you:

## Step 1

Create a new folder on your computer.

To create a new folder, go to the location where you want to create a new folder (such as your documents folder or the desktop) and right-click any blank area (or press CTRL + click). This will create a pop-up menu from which you can select **new folder**.

Name your new folder **unit\_1**.

## Step 2

Open Sublime Text and create a new file. Name the file **index.html** and save it in the **unit\_1** folder you created in Step 1. This will tell Sublime Text that the file you're working on is an HTML file.

## Step 3

Incorporate the HTML boilerplate elements into your index.html file:

<!DOCTYPE html>

<html>

<head>

<title>

<body>

The title for this webpage should be: “[the name of your business]”

## Step 4

Populate the body element with the <h1> text: “[the name of your business]” and the paragraph text: “This is the [business type] of [your name here]”

For example:

<body>

<h1>Batcave</h1>

<p>This is the secret headquarters of Batman</p>

</body>

You probably have a lot more to say than that, but don't worry! You'll have plenty of time to upgrade the text on your site later. Our main focus for now is getting your page up and running.

## Step 5

Within your **unit\_1** folder, create a subfolder called **css**.

Back in Sublime Text, create a new file, name it **style.css**, and save it in your new **css** folder. This will be your external style sheet.

In this file, set the text color to any color besides black, and the background color to any color besides white.

For example:

body {

background-color: black;

}

h1, p {

color: gold;

}

When you're done, make sure to link to your new style sheet to your web page inside the <head> section of your HTML file using the following element:

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

Don't forget to save **both** your HTML and CSS files once you've reached this point.

## Step 6

Congrats! You've created the files necessary to create a basic web page. Next, you're going to take these files and host them online using a service called GitHub.

To learn how, please continue to the next activity, which is a step-by-step slideshow to help you through this process.

[←](https://circuits.generalassemb.ly/student/209/quizzes/232?direction=back) [→](https://circuits.generalassemb.ly/student/209/slideshows/198)

© 2016 General Assembly. All rights reserved

text-align: center;

text-decoration: none;

text-transform: uppercase;

background: #000

color: #fff

font-family: Arial, sans-serif;

size:24px;

font-style: normal;

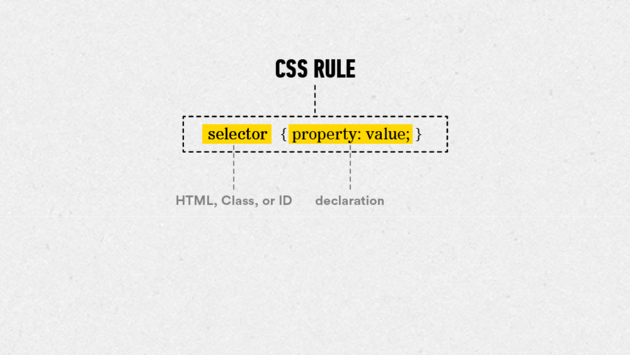
font-weight: bold;

**Review Guide: Principles of HTML & CSS**

Let’s review some of the key ideas introduced in this unit. You can also download all this content for future reference by clicking on the attached file.

**Key Definitions**

* **Rule**: The building block of a CSS stylesheet. A rule consists of a selector and a declaration block (one or more declarations).
* **Declaration**: A declaration is made up of a property and a value, separated by a colon and punctuated by a semi-colon."
* **Selector**: The actual HTML object the declaration(s) apply to.
* **Property**: The characteristic of the selector that will be changed.
* **Value**: The amount or type of change to be applied to the corresponding property of the matched selector.

  
- **Marking-Up**: The process of assigning HTML tags to given text content in order to indicate its relation to the rest of the text or dictate how it should be displayed.  
- **Serif Font**: One of two general categories of fonts (typefaces) that uses marks (called “serifs”) to embellish characters. A common serif font is “Times New Roman”.  
- **Sans-Serif Font**: One of two general categories of fonts that have cleaner line due to not using marks to embellish characters. (Sans Serif literally meaning “Without serif”) A common sans-serif font is “Helvetica”.  
- **Class**: A class attribute is added to an HTML element in order to give you a “hook” to refer to that element in your CSS. CSS class selectors begin with a “.”. Classes can be used multiple times per page.  
- **ID**: An id attribute is added to an HTML element in order to give you a “hook” to refer to that element in your CSS. CSS id selectors begin with a “#”. IDs can only be used one time per page.  
- **Horizontal Rule:** Add a horizontal line across your page using the hr tag.  
- **Line Break:** Break up a block of text using the br tag.  
- **Image:** Add images to your HTML using the img tag. Tell the browser the source of the image file with an src attribute.  
- **Absolute File Path**: A path to a website or file that includes a full web address (starting with “http”) that the browser loads from the remote location directly. For example:

**<**img src**=**"http://imgur.com/awesomedog.jpg"**>**

* **Relative File Path**: A path to a website or file that gives you the path to the resource you are looking for as it relates to your website's local file structure. For example, if you wanted to retrieve an image called *newlogo.png* that resides in a file called *img*, you would enter the following relative address:

**<**img src**=**"images/awesomedog.jpg"**>**

**Why Separate HTML from CSS?**

Separating HTML from CSS offers you scalability and versatility. If you separate how your site looks from what your site says, things become more flexible.

By separating your HTML and CSS, you can make the change in one place and have it apply to your whole site. If you separate what your site says from how it looks, you can apply any number of different styles to the same content.

**CSS Color Treatment**

While color names are fine when you're just beginning, there's a number of reasons you'll want to switch over to something more advanced.

First, color names are rendered differently by different browsers. Secondly, there are only 147 color names accepted as standard, meaning your options are going to be pretty limited.

Instead, you'll want to use either RGB or hexadecimal codes. Both of these are built on a system of entering values for the colors red, green, and blue.

By mixing different intensities of these three colors, you can create millions of different colors and shades. Intensity values range from 0 (no intensity) to 255 (full intensity) in the RGB system.

In hex, they range from 0-9, then continue from A-F, with two characters each for red, green, and blue.   
This is clearer with examples, so [here are some common colors with their RGB and Hex equivalents](http://www.w3schools.com/tags/refcolortryit.asp?color=White).

The format for color names, RGB, and hexadecimal should look like the following, respectively:

p {

**color:** red;

}

p {

**color:** (255,0,0);

}

p {

**color:** #FF0000;

}

**CSS Text Treatment**

* **font-family**

To adjust the font of your selected text element, use the font-family property. For the value, enter the name of the font to which you’d like to alter your text.

To be safe, try putting a comma after your selected font and enter a generic family as a fallback. If the web browser doesn’t support the font you selected, it will choose the fallback.

h1 {

font**-family:** Arial, sans**-**serif;

}

If your selected font is more than one word, capitalize both words and put them in quotation marks.

h1 {

font**-family:** "Courier New", monospace;

}

* **font-size**

To increase or decrease the spacial dimensions of your chosen text, use the font-size property. As a beginner, you’ll want to enter pixel values for your font-size values.

h3 {

font**-size:** 24px;

}

As you become more advanced, try using percentages or ems instead of pixels. These can be pretty tricky, though, so you may want to wait until we cover them in Unit 9.

* **font-weight**

To adjust the thickness of your selected text, use the font-weight property.

As a beginner, you can enter values like “normal” to make your text thin and “bold” to make your text thick. As these values aren’t very specific, different browsers may interpret their display with slightly different outputs.

h1 {

font**-weight:** normal;

}

h1 {

font**-weight:** bold;

}

As you become more advanced, try using the numbers 100, 200, 300, 400, 500, 600, 700, 800, and 900 as values to gain more granular control. With this system, 400 is roughly equivalent to “normal” and 700 roughly equals “bold.”

h1 {

font**-weight:** 400;

}

h1 {

font**-weight:** 700;

}

* **font-style**

To make normal text italic, use the property font-style and the value “italic.” To reverse this effect, use the value “normal.”

a {

font**-style:** italic;

}

a {

font**-style:** normal;

}

* **text-align**

To adjust the positioning of a text element, use the CSS property text-align and one of the following values: left, right, center, or justify.

body {

text**-align:** center;

}

* **text-decoration**

To add an underline to normal text, use the CSS property text-decoration and the value “underline.”

h1 {

text**-decoration:** underline;

}

To remove underlines, use the value “none.” This declaration is often applied to anchor tags.

a {

text**-decoration:** none;

}

Less commonly used values include “overline” to add a line above text and “line-through” which strikes a line through your text.

* **text-transform**

To adjust capitalization in a selected text element, use the text-transform property.

Values for this property include “uppercase” to make every letter capitalized, “lowercase” to make every letter uncapitalized, and “capitalize” to make the first letter of every word in the selected text uppercase.

h1 {

text**-transform:** uppercase;

}

**Sublime Text Shortcuts**

* **Quick Save:** ⌘+S (for Mac) or CTRL+S (PC)
* **Instant Boilerplate:** Type “html” + TAB (Mac and PC)
* **Instant Lorem Ipsum:** Type “Lorem” + TAB (Mac and PC)
* **Add Comment:** ⌘+/ (for Mac) or CTRL+/ (for PC)
* **Undo:** ⌘+Z (for Mac) or CTRL+Z (for PC)
* **Redo:** ⌘+Y (for Mac) or CTRL+Y (for PC)

**Classes vs. IDs**

Classes and IDs, also called selectors, are ways of targeting the style of specific HTML elements on your page.

So what’s the difference between them?

In short, classes allow you to style many elements with a particular style, while IDs are only capable of styling a single element.

Using these selectors allows you flexibility and control of styling individual, as well as groups, of elements on your page.

**HTML and Images**

When you’d like to add images to your site, you should use the tag with the “src” attribute. “src” stands for “Source” and works just like the tag’s “href” attribute. It tells the image tag where to find the image you’d want to include on your page.

**Relative vs. Absolute Addressing**

Relative addressing basically tells your browser: “Look in our project folder for the file specified. Or, if a folder is specified, look within that folder for the file specified.”

On the other hand, when we link to a source outside of our project, it is referred to as an absolute address.

**Sublime Text Shortcuts**

Bookmark [this guide](https://docs.google.com/document/d/1LugbQxfLpG5yFNfTz22hnGclN1Fwp_PH5QZfk0KLjO0/edit?usp=sharing) or download it for review.

**Questions to Ask Your Mentor**

1. Why is it important to create separate HTML and CSS documents? Can you explain separation of concerns in greater detail?
2. How do I get started with creating an external style sheet? What elements should I define the style for first?
3. Where can I find examples of external style sheets to use as templates?
4. What are some best practices for organizing my files and folders on my computer? Should I be saving my work elsewhere?
5. Which fonts should I use for my project?

**Further Reading**

How the RGB Color System Works  
<http://en.wikipedia.org/wiki/RGB_color_model>  
<http://www.rgbworld.com/color.html>

More on Colors  
<http://www.w3schools.com/cssref/css_colornames.asp>  
<http://www.color-hex.com/>

Adobe Color CC (formerly known as Adobe Kuler)  
<https://color.adobe.com/create/color-wheel/>

More on Fonts   
<http://www.w3schools.com/css/css_font.asp>  
<http://www.w3schools.com/cssref/css_websafe_fonts.asp>

Difference Between Serif and Sans-Serif   
<http://www.urbanfonts.com/blog/2013/02/serif-vs-sans-the-final-battle>

Classes and IDs   
<http://css-tricks.com/the-difference-between-id-and-class/>

File Paths   
<http://css-tricks.com/quick-reminder-about-file-paths/>

Absolute vs. Relative Paths  
<http://www.coffeecup.com/help/articles/absolute-vs-relative-pathslinks>  
<http://www.boogiejack.com/server_paths.html>

# Building Your Home Page

In this unit, you learned how to expand your color palette by a couple million values, how to style text and font, how to label and manipulate elements with classes and IDs, how to add images, and more! Did you also know semantic HTML is a vital step in defining your content? If not, don’t worry! The Unit 3 lecture will tell you all about it.

In this project, we’ll revisit the page you made in Unit 1, and we’ll give it a few upgrades.

Let's get started!

## Step 1:

If you haven’t already created a **unit\_3** subfolder in your local GitHub folder, do so now. Within that folder, create two subfolders. Name one subfolder **images** and the other **css**.

Then, create a new file in Sublime Text and save it in this folder as **index.html**. Don’t worry that it has the same filename as the Unit 1 project. It’s okay to have multiple files in your GitHub folder named index.html as long as they’re not in the same unit subfolder. Your homepage should always be named index.html.

Next, recreate the HTML you had in your Unit 1 project. If you want to save time, feel free to copy and paste your original code. If you’d like some extra practice, you should recreate it from scratch.

If you start from scratch, try using the time-saving Sublime Text technique we showed you to create your HTML boilerplate (after saving the file as a .html file, type **html** and hit the **tab** key).

At this point, your HTML should contain at least the following:  
- The HTML boilerplate  
- A title your page "[Your business name]"  
- An <h1> with your business name  
- A <p> tag with a little information about your business

Okay, now that we’re all on the same page, let’s get building!

## Step 2:

First, let’s update the copy in your <p> element. Instead of just a description of your business, let’s make it a slogan! In fact, let’s give it the ID “slogan.” Try to come up with a fun, catchy phrase that captures the spirit of your business. If you’re short of ideas, try using one of the following slogan generators: <http://slogangenerator.co/> or <http://www.sloganizer.net/en/>

Beneath your slogan, add a hero image. This should be a large image that sets the tone of your site. For example, if your business is a restaurant, add a photo of something delicious. If it’s a personal blog, maybe add a photo of yourself or an image representative of the topics about which you’ll write.

Remember, there are two ways to add an image:

Option 1 (best practice) — Host your image locally by putting it in the **images** subfolder inside your \*\* unit\_3\*\* folder, and use a \*\* relative file path\*\* to reference the image.

Option 2 — Find an image online. Right click on the image (or press **CTRL** + \*\* Click\*\* ) and select “Copy Image URL.” Then, use an \*\* absolute file path\*\* to reference the image.

A common technique for slightly more advanced users is to add an image using the \*\* background\*\* property in CSS. This will allow you to have text on top of the image. Try this out if you’re feeling ambitious!

Underneath this new image, create another paragraph with a few sentences telling us about your business. Maybe talk about your history or your mission. Beneath this paragraph, add an unordered list that summarizes a few of your business’s offerings.

Finally, at the bottom of your HTML, add the phrase “Contact me:” with your email address following. Turn your email address into a link. To go above and beyond, see if you can make your email link automatically open an email that’s addressed to you. It’s easier than you think!

## Step 3: Make it Personal

The above content is primarily educational — we want to make sure you can apply the lessons in this unit; however, we understand that you probably have some cool content ideas of your own, so please take this step to add any additional content to your site that you’d like.

The more content you add now, the more interesting style and layout opportunities you'll have later. If you don't have ideas, check out competitor pages and see what content they include on their home page and how it's organized.

Also, add a line or two of space between chunks of content that you imagine will eventually be broken into different sections (as per your design mockup). Don't worry, those lines of space won't affect the layout of your content when it's rendered on the screen. The purpose for such space is simply to make your code more legible and to facilitate future organization of your code.

Additionally, we won’t be adjusting layouts until Unit 6; however, to make the layout process easier, for the time being, list your content as you expect it to eventually appear, top to bottom, left to right.

Once you’re finished adding content, save it. Now, it’s time to style your HTML.

## Step 4: Styling Your Content with CSS

Create a new file in Sublime Text and save it as style.css in the css subfolder of your unit\_3 folder. Make sure to add a link to your CSS in the section of your HTML. Save your HTML again once you’ve added it, then return to your stylesheet.

Consider the selectors you have available:

body

h1

p

ul

li

img

#slogan

Now, let’s give these elements some style! First, give your page’s <body> and <h1> elements unique colors using either the RGB or hexadecimal methods. If you’re having trouble picking the proper hues, check out a site like [Adobe Color CC](https://color.adobe.com/create/color-wheel/?base=2&rule=Custom&selected=4&name=My%20Color%20Theme&mode=rgb&rgbvalues=1,0.13279019591095675,0,0.5,0,0,1,0,0,1,0,0,1,0,0.37792984243083083&swatchOrder=0,1,2,3,4).

Next, make your slogan italicized.

After that, it’s up to you! Play around with the CSS properties you’ve learned thus far and adjust your site however you see fit.

background

background-color

color

font-size

font-style

font-weight

text-align

text-decoration

text-transform

## Step 5: Review the checklist below to make sure your project is ready to submit.

1. I’ve added all content to my page, even if it doesn’t look like my unit 2 sketch
2. I’ve used semantic tags correctly (I learned in the lecture!)
3. I’m using the simple reset (I learned in the lecture!)
4. I made sure to include alt attributes on all of my image tags
5. I only used classes and id's when absolutely necessary

**Thought this unit was the most challenging so far? It was!**  
Book a 1:1 with your mentor by clicking My Mentors.

## Step 6: Host your project for submission

Make sure you’ve saved your finished HTML and CSS files. Then, commit and sync your changes using the GitHub app. The link for your site should be:

<http://your_repository_name.github.io/unit_3/index.html>

If that doesn’t work, go to github.com, open your repository, and copy and paste the URL. Then, discuss whatever issues you had with your mentor.

## Advanced Study:

Try taking your page to the next level with some new CSS properties affecting layout. We're not covering these properties until a later unit, but try doing some quick research and see if you can figure them out!

margin

padding

border

width

text-align

NOTE: If you try to center your list with text-align: center, you'll notice that the list-items and their bullets behave in different ways. There's a reason for that, but it's not one you need to worry about yet.

A good workaround for right now—and one you'll find super-helpful very frequently—is to make the bullets invisible by including the following CSS rule in your stylesheet:

ul {

text-align: center;

list-style-type: none;

}

* [9 of 11 Review Guide: Web Design Basics](https://circuits.generalassemb.ly/student/209/pages/806) 
  + [Video: Typography Fundamentals](https://circuits.generalassemb.ly/student/209/pages/801)
  + [Video: Typography and Readability](https://circuits.generalassemb.ly/student/209/pages/802)
  + [Code Challenge: Line Height](https://circuits.generalassemb.ly/student/209/code_challenge/challenges/144)
  + [Video: Typographic Pairings: Best Practices](https://circuits.generalassemb.ly/student/209/pages/803)
  + [Code Challenge: Fonts](https://circuits.generalassemb.ly/student/209/code_challenge/challenges/145)
  + [Video: Advanced Typographic Considerations](https://circuits.generalassemb.ly/student/209/pages/804)
  + [Code Challenge: Using Web Fonts](https://circuits.generalassemb.ly/student/209/code_challenge/challenges/146)
  + [Video: Color — Typography and Beyond](https://circuits.generalassemb.ly/student/209/pages/805)
  + [Review Guide: Web Design Basics](https://circuits.generalassemb.ly/student/209/pages/806)
  + [Quiz: Web Design Basics](https://circuits.generalassemb.ly/student/209/quizzes/235)
  + [Set the Font to Set the Tone](https://circuits.generalassemb.ly/student/209/assignments/158)
* [My Mentors](https://circuits.generalassemb.ly/student/209/mentors)
* [Lectures](https://circuits.generalassemb.ly/student/209/lectures)
  + [My Profile](https://circuits.generalassemb.ly/me)
  + [Edit my Profile](https://circuits.generalassemb.ly/me/edit)
  + [My Account](https://accounts.generalassemb.ly/account)
  + [Sign Out](https://circuits.generalassemb.ly/sign_out)
* [HTML, CSS & Web Design Circuit](https://circuits.generalassemb.ly/)
* [My Mentors](https://circuits.generalassemb.ly/student/209/mentors)
* [Lectures](https://circuits.generalassemb.ly/student/209/lectures)
* [My Profile](https://circuits.generalassemb.ly/me)
* [Edit my Profile](https://circuits.generalassemb.ly/me/edit)
* [My Account](https://accounts.generalassemb.ly/account)
* [Sign Out](https://circuits.generalassemb.ly/sign_out)

# Review Guide: Web Design Basics

Let’s review some of the key ideas introduced in this unit. You can also download all this content for future reference by clicking on the attached file.

## Key Definitions

* **Glyph**: A single symbol of a given typeface. Individual letters, numbers, and punctuation are the most commonly used glyphs.
* **Typeface**: An entire family of glyphs defined by shared visual characteristics, e.g., Helvetica.
* **Font**: A specific set of characters of a given typeface, e.g., 12pt Helvetica Bold.
* **Serif**: A bracket-like extension from a stroke of a glyph that is usually not part of its essential structure. Common serif typefaces used in web design: Hoefler Text, Garamond, Times New Roman, and Perpetua.
* **Sans Serif**: Typefaces lacking serifs. Colloquially shortened to \_sans—\_French for “without.” Common sans serif typefaces used in web design: Helvetica, Futura, Gill Sans, and Arial.
* **Point**: Common size measure of a typeface from the apex of the tallest ascender, e.g., the top of a lowercase “f,” to the base of the longest descender, e.g., the lowest point of a lowercase “y.”
* **Font Stack**: A prioritized listing of typefaces based upon a shared classification, as a means to standardize viewing experiences across browsers.
* **X-height**: The measure from the top of the lowercase “x” of a given typeface to the baseline, the optical plane on which a line of type rests.
* **Baseline**: The optical plane on which a line of type rests.
* **Cap Height**: The height of a capital letter above the baseline.
* **Header**: Often referred to as headlines and subheads. These are usually a focal point on a page and are visually reinforced by size and whitespace. In HTML, features of this type are normally created with the h1 through h6 elements.
* **Body**: Also referred to as “body copy” and not to be confused with the HTML body element, this term refers to visually smaller, long-form paragraph text. In HTML, features of this type are normally created with the p element.
* **Line Length**: Frequently measured in the number of characters per line.
* **W3C**: An international consortium focused on developing web standards.
* **Type Foundry**: A business that builds and sells typefaces.
* **EM (CSS)**: The value of this element ties an element’s size relative to a default value.
* **Ligatures**: Common glyph pairs joined as a single unit.
* **Alternates**: Additional versions of standard characters such as swashes.
* **Small Caps**: Capitals designed specifically to pair with lowercase letters.
* **Rivers**: An excess of vertical negative space that runs through blocks of copy and interferes with readability.
* **En Dash**: Used to indicate a range of values, e.g., “3–4 times per week.”
* **Em Dash**: Used to indicate a break in thought, e.g., “if you have questions regarding grammatical style—and the corresponding typographic design conventions—a resource will be provided.”
* **Smart Quotes**: AKA typographer’s quotes. These are used for apostrophes and quotation marks. Smart quotes look like this: (“ ”).
* **Prime Marks**: Used for measurement: (' ").

## Web-Based Type: Readability And Legibility

Legibility and readability of type are two factors that mediate text-based communication.

* **Legibility**: the degree to which the details of a typeface can be discerned, rooted in the physical characteristics of the typeface itself. Poor legibility example: small, heavily bitmapped text.
* **Readability**: the degree to which text can be read and ultimately understood, a function of the arrangement of text. Poor readability example: a paragraph of otherwise legible text set with too little space between lines.

## Typographic Pairings

Let’s review a few key ideas around pairing typefaces:

**I. Typeface selection is largely based upon its function.**

* Outline the text-based needs of the page and, ultimately, the site.
* Select your typefaces accordingly.

**II. Initial selection of family pairs frequently begins with one sans and one serif.**

* This will give the designer/coder a high degree of contrast between type—contrast between these classifications is generally greater than between two sans or two serif faces.

**III. Select typefaces with ample degrees of weight.**

* Common practice dictates leaving at least one degree between weights of a typeface when creating visual contrast.

## Typographic Sins and Virtues

**Use only a single space following periods or any other punctuation at the end of a sentence.**

The usage of a double space between sentences is a relic of the days when typewriter usage was the norm. More importantly, usage of the double space throughout a page creates “rivers,” an excess of vertical negative space that runs through blocks of copy and interferes with readability.

**Understand the functional and physical differences between hyphens (-), en dashes (–), and em dashes (—).**

An intro to rules of usage: hyphens join compound words, e.g., “a blue-green sea;” en-dashes are used to indicate a range of values, e.g., “3–4 times per week;” and em dashes indicate a break in thought, e.g., “if you have questions regarding grammatical style—and the corresponding typographic design conventions—a resource will be provided.”

**Use typographer’s quotes (“ ”), also know as “smart quotes” for apostrophes and quotation marks; use prime marks (") for measurement.**

Typographer’s quotes — Matthew Carter remarked, “type is a beautiful group of letters, not a group of beautiful letters.”

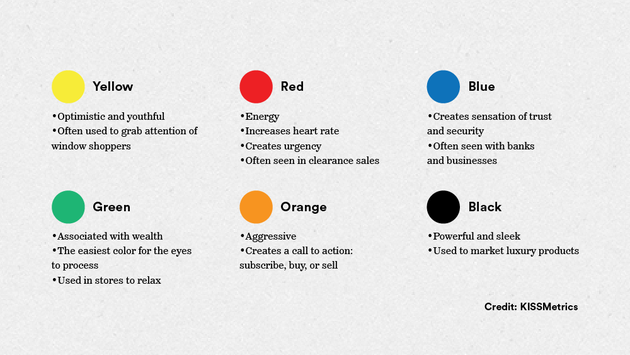
Prime marks — Stefan Sagmeister, the Johnny Depp of the design world, is 6\*'5"\* tall.

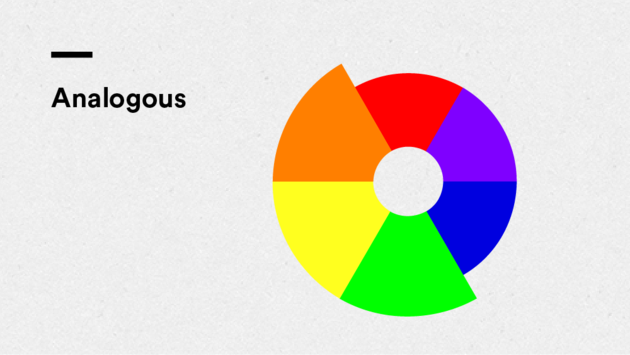
## Typography and Color

The use of color on the web can create, entice, or repel users. It can establish a focal point or create distraction, and it can assist in way-finding or add confusion for the user.

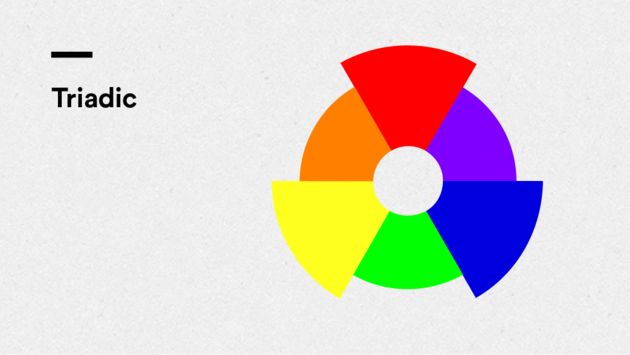
**Color and Navigation**:  
Color is often applied to text as a significant navigational aid. For example, moving the pointer over navigational text can trigger a “hover state,” changing the appearance of text and providing a cue to the user.

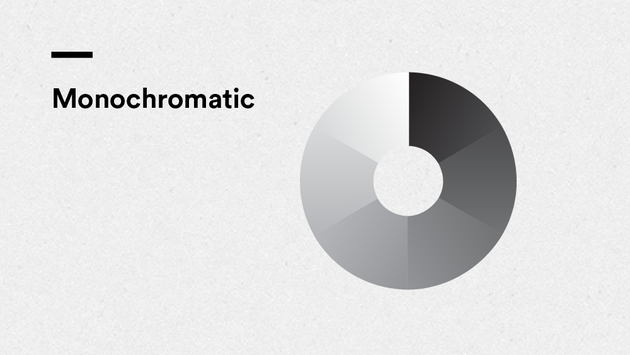
**Color Selection — Emotional:**  
Color is used to evoke an emotional response by drawing upon familiar associations. For example, red hues are evocative of energetic states and can imply activity. Blue hues, the color of the skies and oceans, can inspire feelings of serenity and security.

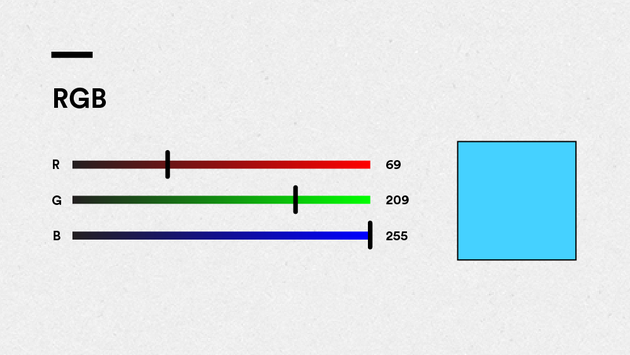
  
**Color Selection — Technical**:  
Analogous Colors: Colors that are adjacent along the color wheel, e.g., green, yellow, and orange.

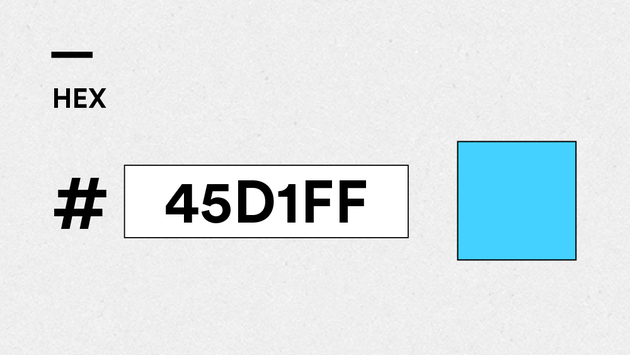
  
Complementary Colors: Colors that are opposite one another on the color wheel, e.g., blue and orange.

  
Triadic Colors: Colors that are equally spaced apart along the color wheel. The primary colors—red, blue, and yellow—are an example of triadic colors.

  
Monochromatic Colors: A color scheme built upon a single hue, i.e., tints and shades of one color.

  
**Color Models — RGB vs. Hex:**  
RGB: A model that expresses the additive color process as values of red, green, and blue ranging from 0 to 255.

  
HEX: Color on the web, represented by its hexadecimal value..



## Questions to Review With Your Mentor

What do I need to know in order to become conversant with the basics of typography?

Where can I find inspirational sites that effectively use color?

What are good resources I can reference for additional readability and accessibility insights?

Are there standard typographic conventions I should adhere to?

What are the color models that are most relevant to the web designer/coder?

## Further Reading

Looking for more help with typography and color? Here are a few places you can check out:

A further discussion of typographic basics can found at FontShop, a highly-regarded type design company:  
<http://www.fontshop.com/education/>

Strong typographical resource site:  
<http://ilovetypography.com/2008/03/21/extreme-type-terminology/>

Articles such as How People Read on the Web: The Eyetracking Evidence demonstrate the importance of readability, layout, and hierarchy in web design:  
<http://www.nngroup.com/reports/how-people-read-web-eyetracking-evidence/>

Hoefler & Co. is a highly regarded type foundry, a company that designs and distributes typefaces, and offers a Combing Fonts page on their site that is both informative and visually inspiring:  
<http://www.typography.com/techniques/index.php>

For more on advanced typesetting concepts see Type on Screen: A Critical Guide for Designers, Writers, Developers, and Students by Ellen Lupton, a designer and educator well known for her writings on typography and design:  
<http://www.papress.com/html/book.details.page.tpl?isbn=9781616891701>

Strunk and White’s The Elements of Style is an English writing and style reference frequently referred to by writers, readers, and designers alike:  
[http://www.amazon.com/Elements-Style-Illustrated-William-Strunk/dp/0143112724/ref=sr11?s=books&ie=UTF8&qid=1296660224&sr=1-1](http://www.amazon.com/Elements-Style-Illustrated-William-Strunk/dp/0143112724/ref=sr11?s=books&amp;ie=UTF8&amp;qid=1296660224&amp;sr=1-1)

Companies such as the Color Marketing Group monitor trends and forecast colors:  
[http://www.colormarketing.org](http://www.colormarketing.org/)

Maintain your awareness of current design trends and bolster your technological prowess by continuously learning through sites such as A List Apart and Communication Arts:

* [http://alistapart.com/topic/typography-web-fonts](http://alistapart.com/topic/typography-web-fontshttp:/alistapart.com/topic/typography-web-fonts)
* [http://www.commarts.com](http://www.commarts.com/)

# Set the Font to Set the Tone

Web Fonts might seem easy, but type design can make a huge impact on your design. Check out the lecture this week to learn ways to make your site look even better.

For your Unit 4 project, we’re going to take your business website to the next level by upgrading the typography.

## Step 1: Setup

Pull up your HTML and CSS from the Unit 3 Project. In this unit you're going to set the fonts for your webpage. This is done in the CSS by using the property "font-family" on the selectors you wish to adjust.

While there are a number of fonts that are automatically recognized by browsers — classics like Times New Roman and Arial — we’re going to make it more interesting by adding fonts from the web using Google Fonts.

**Note:** If you have multiple <p> elements or headlines and you want to give them different fonts, remember to use a class or ID to give yourself more granular control.

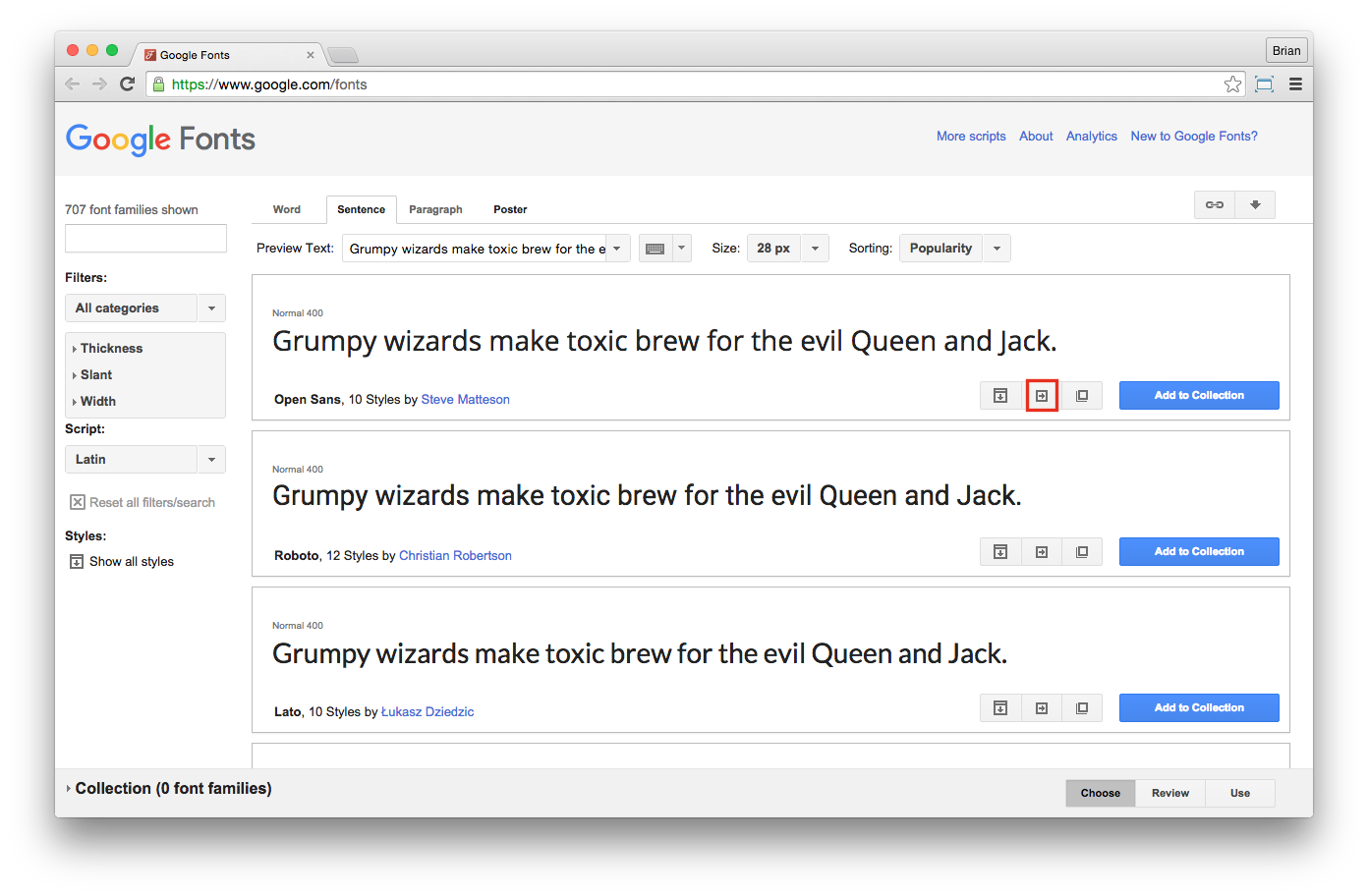
## Step 2: Exploring Google Fonts

Once you’ve chosen two elements you’d like to give new fonts, go to [www.google.com/fonts](http://www.google.com/fonts) and think about what kind of typography will best communicate your business's identity to your target customers.

Use the search bar or the filters in the left-hand column to narrow down fonts by type; consider weight, width, and other typographic elements covered in this unit. Do you want text that’s sleek? fun? old-fashioned? futuristic? For practice, try experimenting with a few different typefaces on your page.

Generally speaking, you should be looking for one typeface that is for the majority of your text and one typeface that is for your headers. These can be either be two different font-families or it can be two different styles of the same typeface. The important thing to remember is that your base font styles (paragraphs, list items, links, etc) must be a font that is easy to read for the best user experience.

After you find a typeface that you like, click on the button surrounded the red box below:

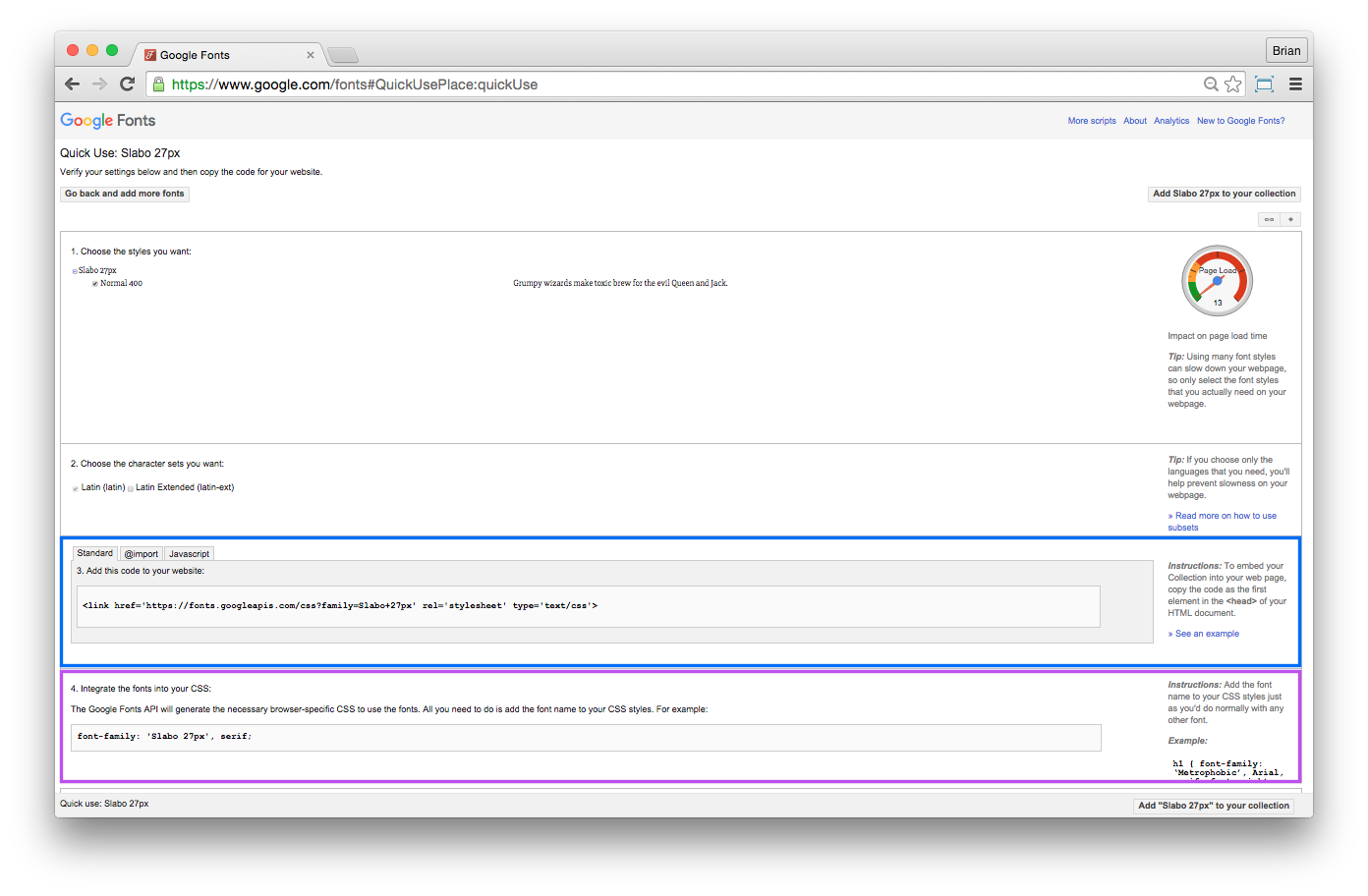


## Step 3: Adding Google Fonts

That button will take you to a new page with more information about the typeface you selected. Sections 1 and 2 on this page allow you to fine tune your selection.

To add this new typeface to your website, scroll down the page to section 3 (blue box in the image below), click on the tab that says **@import**, and copy the code in the box to the very top of your external style sheet.

After you’ve completed that step, copy and paste the code from section 4 (purple box in the image below) into the CSS rule for which you want to use this new typeface.



**Advanced:** If you’d like to go above and beyond, try adjusting your fonts even further by adjusting their letter-spacing, word-spacing, and line-height.

Be careful though, a little word and letter spacing goes a long way. keep it subtle for maximum readability!

## Step 4: Double-check!

Review the checklist below to make sure your project is ready to submit. By this unit, all of your content should be on the page, and your HTML should be complete. You can check your HTML for errors using The Validator. <https://validator.w3.org/>

1. I’ve validated my HTML and asked my mentor questions about any errors
2. All child elements are correctly nested inside their parent elements
3. I haven’t used too many different fonts on my page (2 or fewer is recommended)
4. I’ve overwritten all default font styles with my own css
5. My CSS in the correct order (fonts, element selectors, classes, id's)
6. I understand that I haven’t learned layout yet, but soon my page will start taking shape

**Want feedback on your project? Of course you do!**  
Book a 1:1 with your mentor.

## Step 5: Save, Sync, and Submit

After you’ve applied type treatment to your site with Google, save your page in the unit\_4 subfolder of your local GitHub repository. Then, commit and sync your changes.

Finally, submit your project on Circuits by providing the link to your updated page using the URL for your online GitHub repository. Your URL should look something like this:

<http://your_repository_name.github.io/unit_4/index.html>

And you’re all set. Congrats on another great week!

# Set the Font to Set the Tone

Web Fonts might seem easy, but type design can make a huge impact on your design. Check out the lecture this week to learn ways to make your site look even better.

For your Unit 4 project, we’re going to take your business website to the next level by upgrading the typography.

## Step 1: Setup

Pull up your HTML and CSS from the Unit 3 Project. In this unit you're going to set the fonts for your webpage. This is done in the CSS by using the property "font-family" on the selectors you wish to adjust.

While there are a number of fonts that are automatically recognized by browsers — classics like Times New Roman and Arial — we’re going to make it more interesting by adding fonts from the web using Google Fonts.

**Note:** If you have multiple <p> elements or headlines and you want to give them different fonts, remember to use a class or ID to give yourself more granular control.

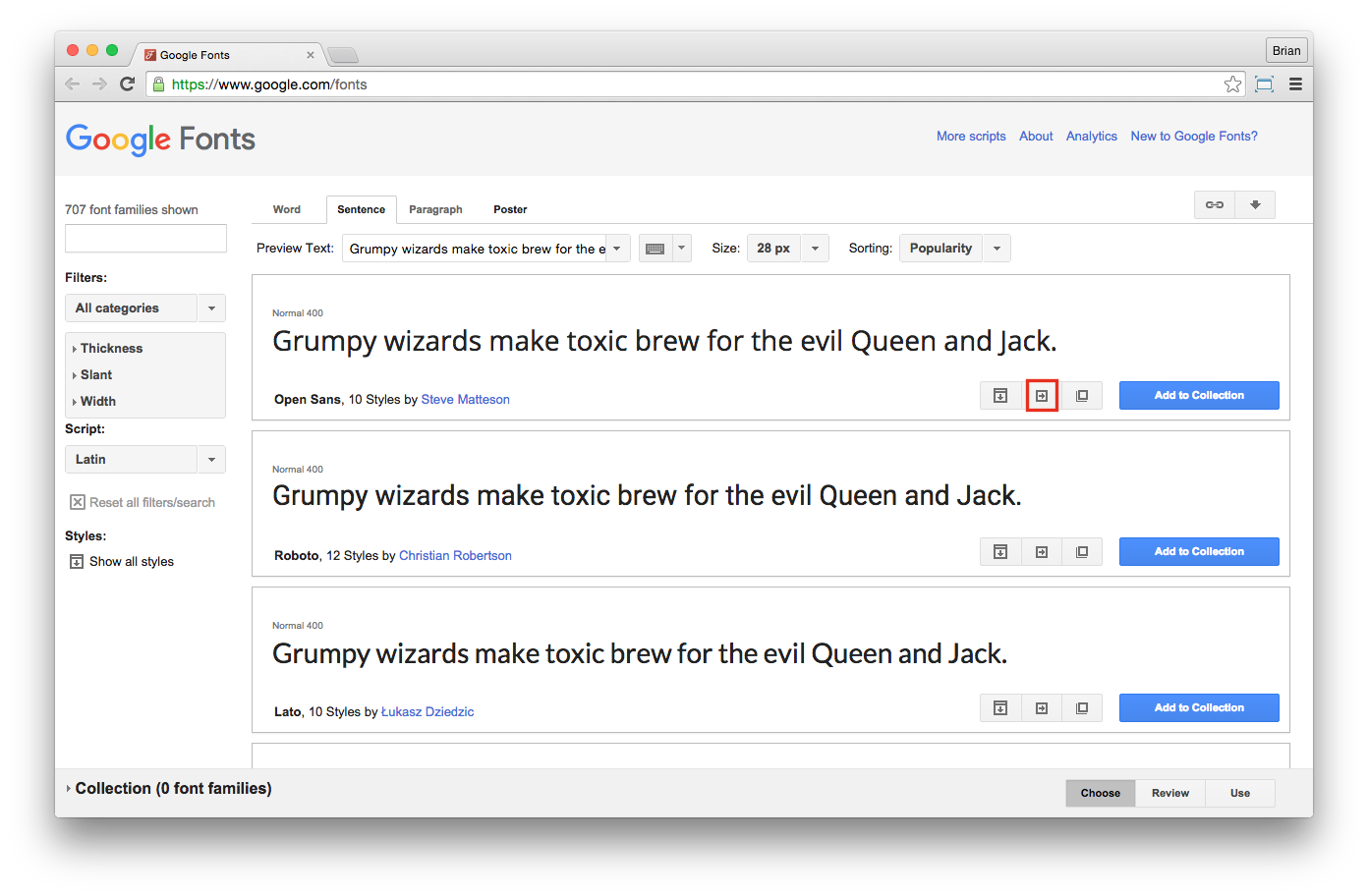
## Step 2: Exploring Google Fonts

Once you’ve chosen two elements you’d like to give new fonts, go to [www.google.com/fonts](http://www.google.com/fonts) and think about what kind of typography will best communicate your business's identity to your target customers.

Use the search bar or the filters in the left-hand column to narrow down fonts by type; consider weight, width, and other typographic elements covered in this unit. Do you want text that’s sleek? fun? old-fashioned? futuristic? For practice, try experimenting with a few different typefaces on your page.

Generally speaking, you should be looking for one typeface that is for the majority of your text and one typeface that is for your headers. These can be either be two different font-families or it can be two different styles of the same typeface. The important thing to remember is that your base font styles (paragraphs, list items, links, etc) must be a font that is easy to read for the best user experience.

After you find a typeface that you like, click on the button surrounded the red box below:

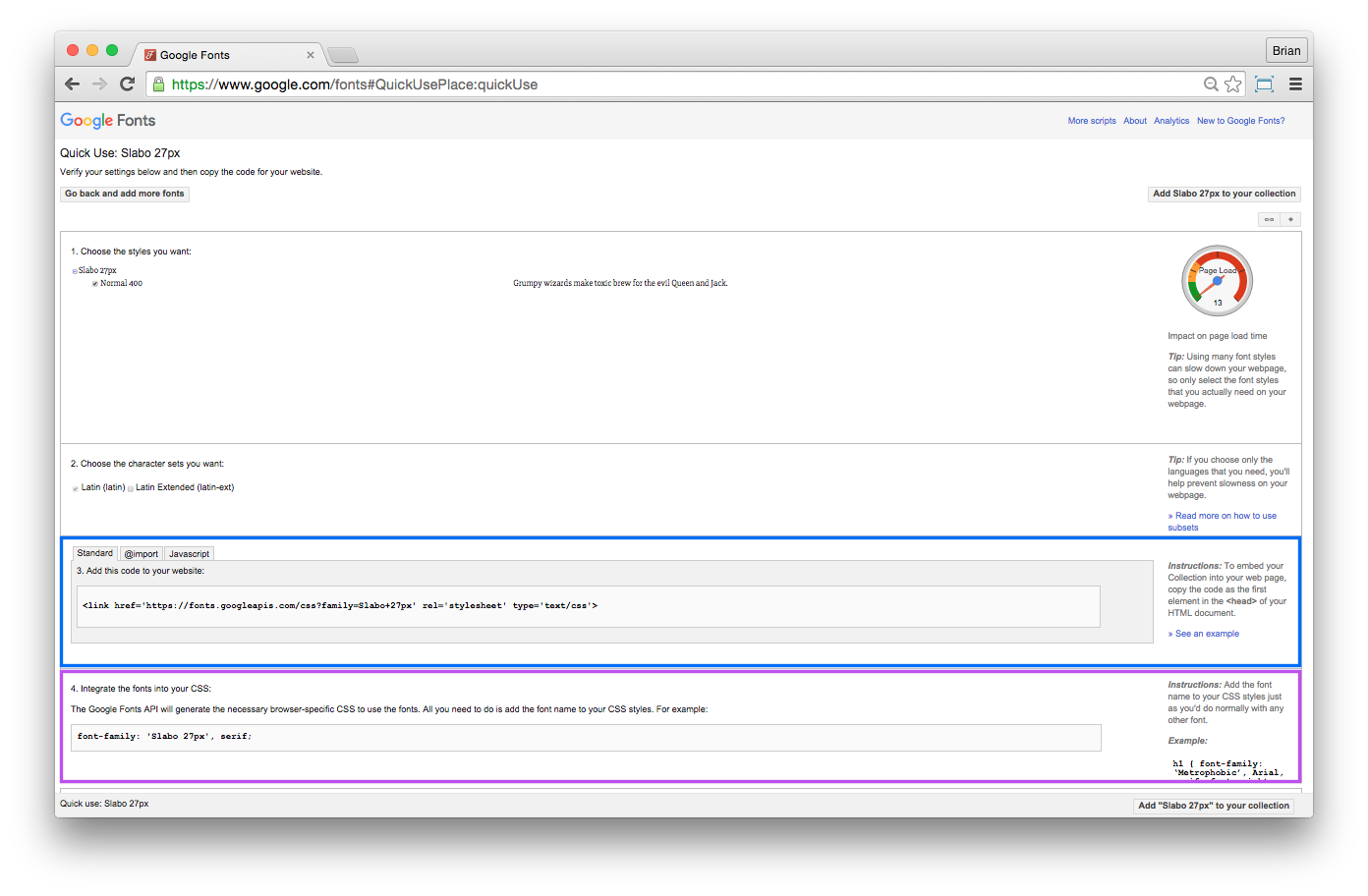


## Step 3: Adding Google Fonts

That button will take you to a new page with more information about the typeface you selected. Sections 1 and 2 on this page allow you to fine tune your selection.

To add this new typeface to your website, scroll down the page to section 3 (blue box in the image below), click on the tab that says **@import**, and copy the code in the box to the very top of your external style sheet.

After you’ve completed that step, copy and paste the code from section 4 (purple box in the image below) into the CSS rule for which you want to use this new typeface.



**Advanced:** If you’d like to go above and beyond, try adjusting your fonts even further by adjusting their letter-spacing, word-spacing, and line-height.

Be careful though, a little word and letter spacing goes a long way. keep it subtle for maximum readability!

## Step 4: Double-check!

Review the checklist below to make sure your project is ready to submit. By this unit, all of your content should be on the page, and your HTML should be complete. You can check your HTML for errors using The Validator. <https://validator.w3.org/>

1. I’ve validated my HTML and asked my mentor questions about any errors
2. All child elements are correctly nested inside their parent elements
3. I haven’t used too many different fonts on my page (2 or fewer is recommended)
4. I’ve overwritten all default font styles with my own css
5. My CSS in the correct order (fonts, element selectors, classes, id's)
6. I understand that I haven’t learned layout yet, but soon my page will start taking shape

**Want feedback on your project? Of course you do!**  
Book a 1:1 with your mentor.

## Step 5: Save, Sync, and Submit

After you’ve applied type treatment to your site with Google, save your page in the unit\_4 subfolder of your local GitHub repository. Then, commit and sync your changes.

Finally, submit your project on Circuits by providing the link to your updated page using the URL for your online GitHub repository. Your URL should look something like this:

<http://your_repository_name.github.io/unit_4/index.html>

And you’re all set. Congrats on another great week!

# Review Guide: Basic Webpage Layout

Let’s review some of the key ideas introduced in this unit. You can also download all this content by clicking on the attached file for future reference.

## Key Definitions

**Span**: a generic wrapper for any inline content. Spans are generally used to group small sections of content for styling purposes.

**Div**: a generic wrapper for any block content. Divs are used to group other elements together or to provide style to a specific area.

**Header**: These elements kick off your page with introductory content like logos, headlines, titles, and links.

**Footer**: This element wraps the content at the bottom of your web page. This element often contains copyright information, links to career pages, contact information, terms of use, etc.

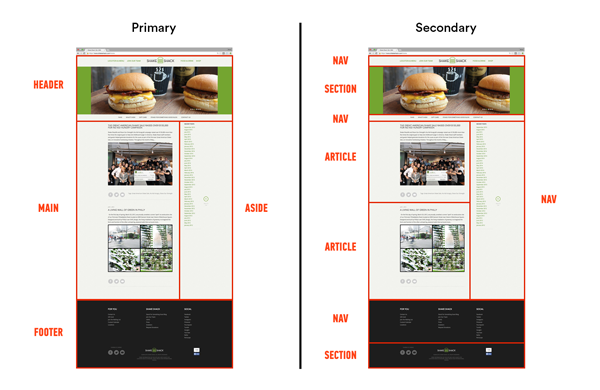
**Main**: These elements contain all the primary content — articles, blog posts, images, videos, etc. — between the <header> and the <footer>.

**Aside**: These elements contain all the secondary content between the <header> and <footer>, supplemental information like recommended stories or archived blog links.

**Nav**: Any navigational links that help users get around your page should be wrapped in <nav> tags.

**Section**: These elements are used to group the content of a page into related chunks.

**Article**: These elements are used to contain standalone blocks of text, such as news articles, blog posts, etc.



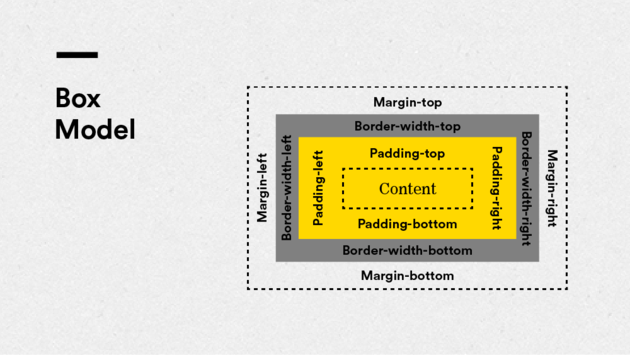
**Box Model**: Every html element on the page is a box, and the box model dictates what the element will look like. The model consists of the content, padding, border, and margin.

**Content**: The text and images that are included within any element's opening and closing tags.

**Padding**: The area outside of the content, but within the border. For example, padding will extend the background-color and contribute to the overall width of the object.

**Border**: A line of variable thickness, color, and style that goes around the padding and content.

**Margin**: The area outside of the border that separates the object from other objects around it. Margin has a transparent background-color.



## Inline vs. Block

**What are Inline Elements?**  
One of two categories of HTML elements, they take up the width of their contents and do not begin with new line. They cannot have height or width assigned.

**What are Block Elements?**  
One of two categories of HTML elements, they break to a new line before and after the element and they take up the width of their containers by default (this is often the browser window itself). They can have padding, margin, height, and width assigned.

## Span and Div Tags

To understand the difference between Span and Div, we need to think back to the last lesson on inline and block elements.

**Span** is inline and is used to apply CSS to inline HTML, while **Div** is a block element and is used to apply CSS to—you guessed it—blocks of HTML. Divs typically wrap big sections of content on a page.

**NOTE:** use semantic elements over divs whenever possible.

## Advanced Sublime Text Shortcuts

Bookmark [this guide](https://docs.google.com/document/d/1zf4VmfLqAoP_oDcgr9GYBkTvReH9cWT4u24PphjB5eo/edit?usp=sharing) or download it for review.

## Questions to Ask Your Mentor

Can you show me some great examples of well-styled inline and block elements? How should I be using these in my pages?

What are some good examples of the box model in action?

How should I be thinking about prioritizing content based on its placement on the page?

When would it be appropriate to create a page based on a single-column layout? When will we learn about multi-column layouts?

## Further Reading

Border  
<http://www.w3schools.com/css/cssborder.asp>

Shorthand Properties  
<https://developer.mozilla.org/en-US/docs/web/CSS/shorthandproperties>  
<http://www.dustindiaz.com/css-shorthand/>  
[http://www.456bereastreet.com/archive/200502/efficient\_css\_with\_shorthand\_properties/](http://www.456bereastreet.com/archive/200502/efficientcsswithshorthandproperties/)

Block and Inline  
<http://webdesignfromscratch.com/html-css/css-block-and-inline/>

Span and Div  
<http://www.htmldog.com/guides/html/intermediate/spandiv/>  
<http://www.w3schools.com/html/htmlblocks.asp>

Box Model  
<http://www.htmldog.com/guides/css/beginner/margins/>  
<http://www.htmldog.com/guides/css/beginner/borders/>  
<http://css-tricks.com/the-css-box-model/>

Header, Footer, Nav  
<http://html5doctor.com/the-header-element/>  
<http://www.html-5-tutorial.com/footer-element.htm>  
<http://html5doctor.com/the-footer-element-update/>  
<http://html5doctor.com/nav-element/>  
<http://www.html-5-tutorial.com/nav-element.htm>

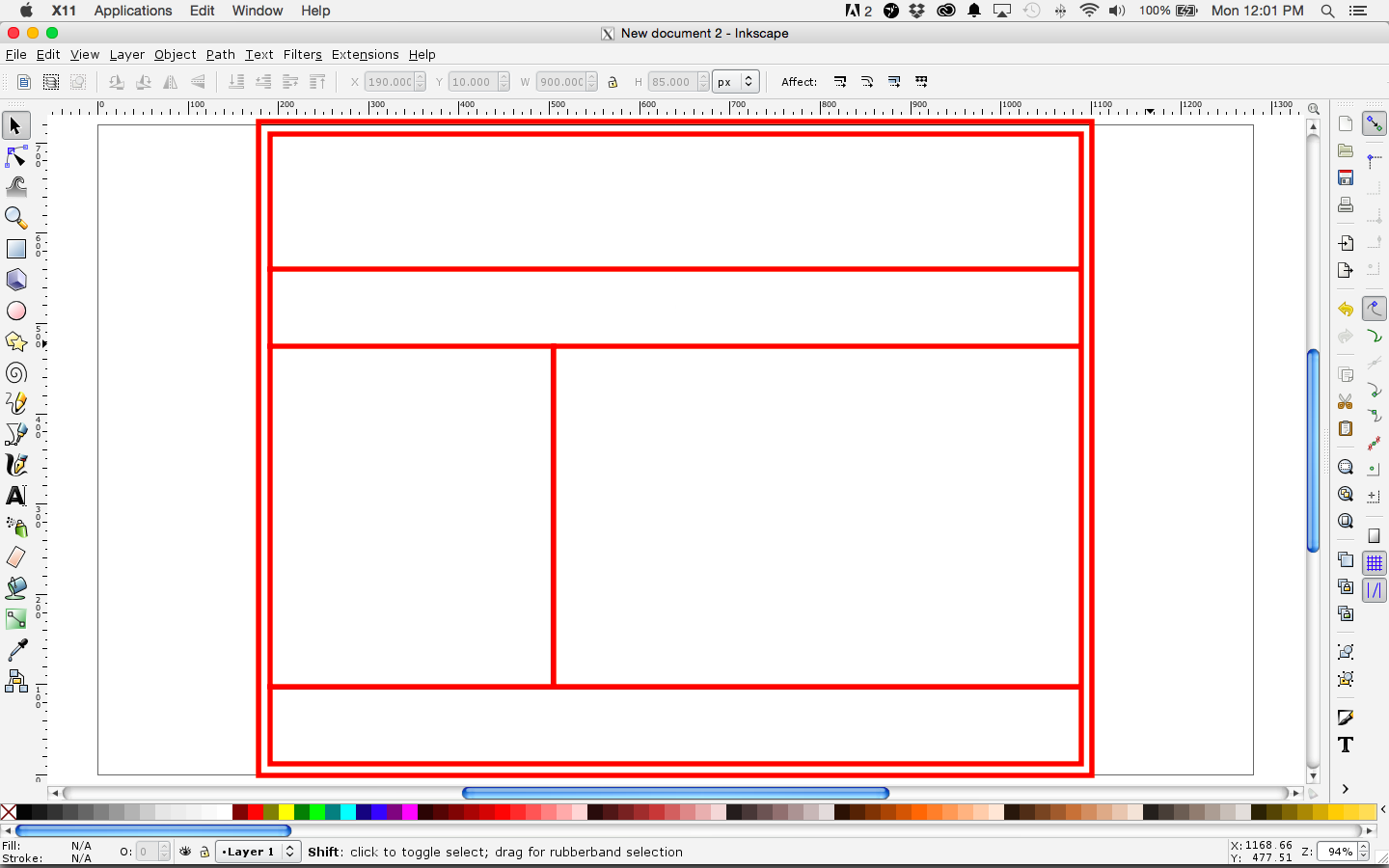
# Organizing and Visualizing Your Page with Semantic Elements

In this unit, you learned about inline versus block elements, divs and spans, semantic elements, and the box model. These concepts exist to help you organize the content of your page. Want to get a head start on making you layout more complex? The lecture this week has you covered.

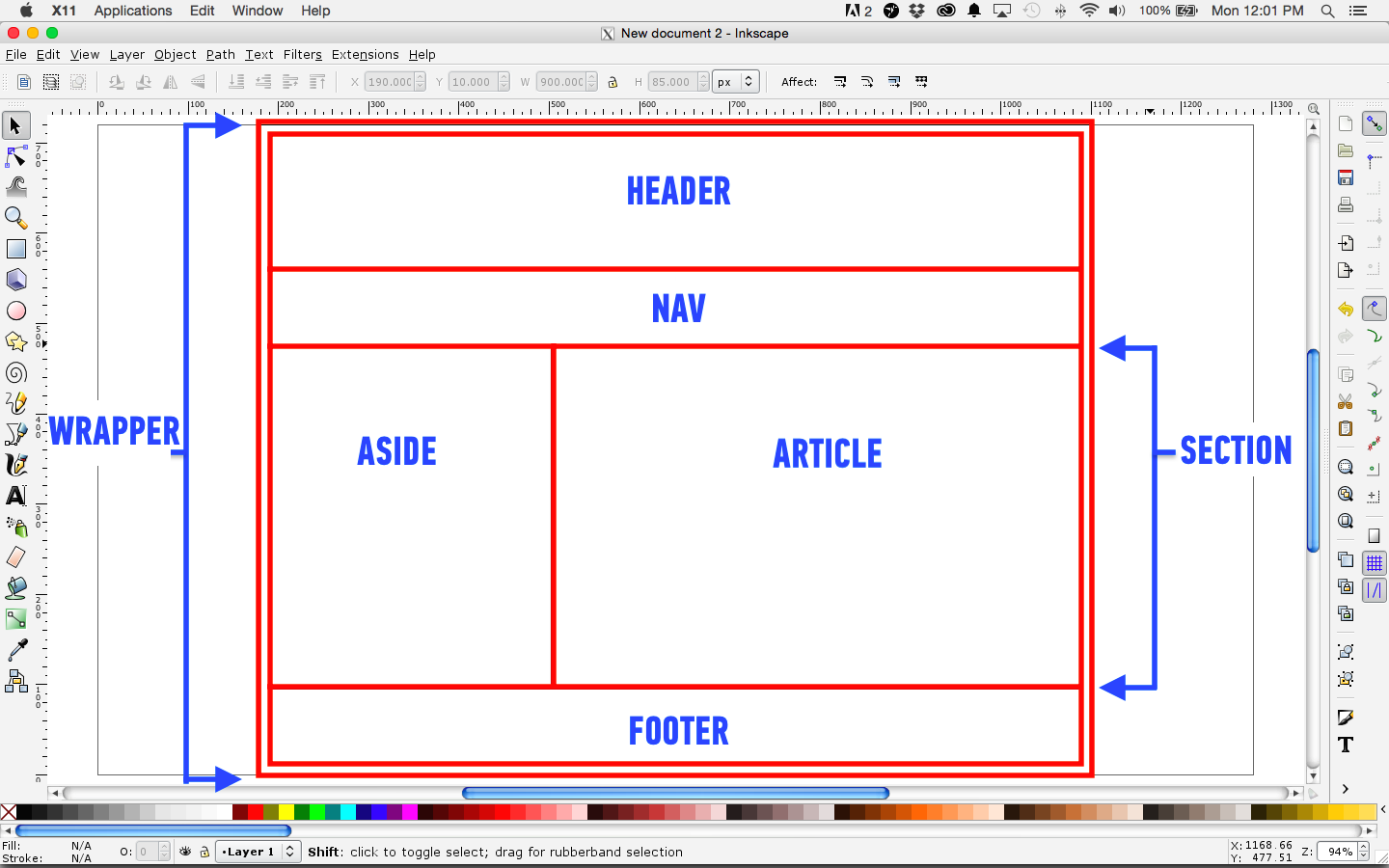
In this week’s project, we’re going to apply some of these concepts to help you visualize your page as blocks of content. This exercise will help prepare you for next week’s activities, which focus on creating various types of layouts with HTML and CSS.

## Step 1: Design Blocks to HTML Blocks

Think back to the Unit 2 project, when you created design mockups for your page. Upon completing that project, your mockups should have looked like a series of blocks, like this:



Using the design program of your choice, open your design mockups and review the blocks into which you divided your page. Now, using divs and semantic elements, label those blocks, like this:



Once you’re done labelling the blocks making up your design mockup, save the file as a .png and put it in the images folder of your the **unit\_5** project folder of your local GitHub repository folder. Make sure to commit, sync, and publish this update so that your online repository reflects this addition.

## Step 2: Incorporating Semantic Elements in HTML

Now, you’re going to apply these labels to the HTML for your website.

Copy your most recent index.html file and save it as a new file in your **unit\_5** subfolder. Then, open this page with Sublime Text, find each block of content, and organize it using the divs and semantic elements you just applied to your design mockup.

Right now your blocks of content will all be stacked on top of each other. We’ll teach you how to create multiple-column layouts in the next unit. For the time being, if you have any blocks that you want to have laid out next to each other — for example an and an — the one that will eventually be on the left should be on top of the one that will eventually be on the right.

For example, the code for our above design mockup would look like this:

<body>

<div class="wrapper">

<header>

</header>

<nav>

</nav>

<section id="one">

<aside class="picture">

</aside>

<article class="description">

</article>

</section>

<footer>

</footer>

</div>

</body>

Notice how we’ve given our elements classes and IDs. If any of your generic or semantic content wrappers are repeated — namely, if you have several divs, sections, or articles — make sure to make them unique with classes or IDs.

For example, if you have an outer <div> containing all of your content, it’s common to give that div the class name “container” or “wrapper.” If you have several articles, asides, or sections, you could label them by category or you could just number them, like article class=“news” or section id=“one”, etc.

## Step 3: Finish Adding Content

If your design mockups contain any blocks for which you haven’t yet created HTML content, go ahead and add that content now and wrap it with appropriate semantic or generic elements. The next several units will focus on finishing your page’s layout and style, which will be much easier if you have all of your HTML content finished.

That said, if you have an area of your page for which you simply don’t yet know what you’re going to write, you can use temporary images, titles, and lorem ipsum to fill out your page for the time being; however, we strongly insist you start giving your page content as soon as you can.

When you’re done, make sure to save your changes; however, you can wait to commit and sync them until the end of the next step.

## Making Content Blocks Visible

Now that you’ve organized your index.html page with generic and semantic elements, and you’ve added content, let’s use background colors, padding, and borders to make of your new blocks clearly visible.

**Don’t worry if this step contradicts your style plans. The changes here are only temporary and intended to help with layout. You’ll be able to remove them at the end of the next unit.**

Open your style.css file. Add selectors for each of your new semantic elements and give them the following declarations:

header {

background-color: [enter color here];

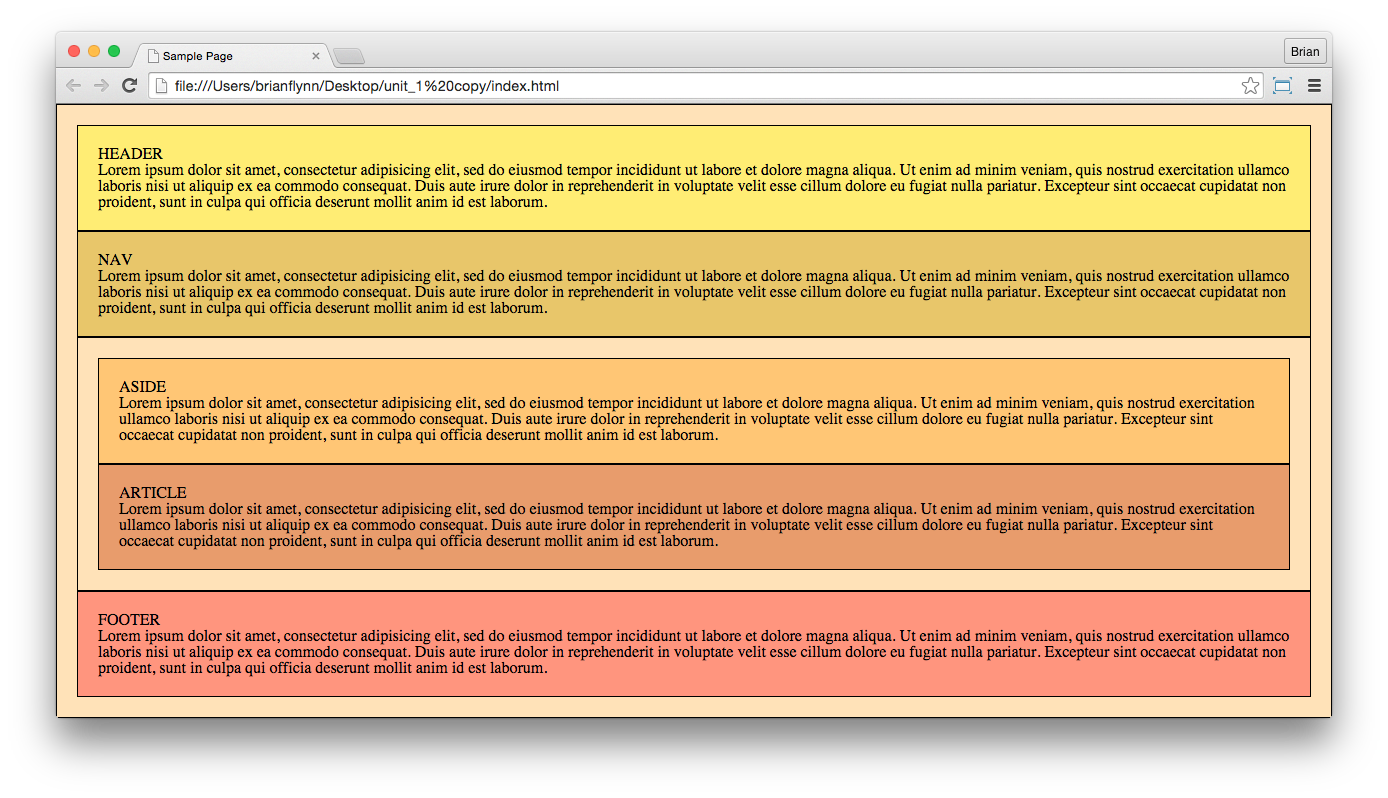
border: 1px black solid;

padding: 20px;

}

For the background colors, use the following colors:   
- header = #FFED74  
- nav = #E8C66A  
- aside = #FFC675  
- article = #E89C6C  
- footer = #FF957E  
- div / section = #FFE2B8

Once you’re done applying this code, save your **style.css** file. Then, use the GitHub Desktop app to commit, sync, and publish your changes with your online GitHub repository. When you open your page now, the blocks making up your site should be labelled and color coded, kind of like this:



## Step 5: Double-check!

Review the checklist below to make sure your project is ready to submit.

1. I’m not using divs where semantic tags should be
2. I don’t have too many divs on my page
3. I’ve set widths on elements where needed
4. All text blocks on my page fall within 45-75 characters per line
5. I’ve read this article to prep for the next unit: <https://css-tricks.com/all-about-floats/ghcv>

**Need help fine-tuning?**  
Book a 1:1 with your mentor.

## Step 6: Submitting Your Project

To submit this week’s project, copy the URL for the **unit\_5** folder from your online GitHub repository, much like you did in the Unit 2 project, and enter it as the project URL.

That way, your mentor can take a look at both your labelled design mockups and your color-coded website, both of which will help them assist you in achieving your layout goals for your site — something which you should definitely discuss with them during your weekly one on one session this week.

And that’s it. Congrats on another great week!

# Review Guide: Navigation & Page Layout

Let’s review some of the key ideas introduced in this unit. You can also download all this content by clicking on the attached file for future reference.

## Key Definitions

**Navigation Bar**: a section on web page with links to other pages or parts of a site. It can appear as a vertical, horizontal, or alternatively-styled block.

**Pseudo-class**: a keyword added to selectors that indicates a particular condition of the element in which to apply CSS to the selected element. For example, [:hover](https://developer.mozilla.org/en-US/docs/Web/CSS/:hover) will apply a style when the user hovers over the element specified by the selector.

**Nav**: A semantic element used to define a set of navigation links.

<nav></nav>

**Clearfix**: A technique in CSS to automatically clear after an element. It's generally used in column layouts where elements are floated. It involves giving the floated elements the class "clearfix" and creating a corresponding CSS rule for that class with the pseudo-class :after.

.clearfix:after {

visibility: hidden;

display: block;

font-size: 0;

content: " ";

clear: both;

height: 0;

}

## CSS Reset

There is no true standard for how un-styled HTML elements get rendered in various browsers. In other words, each browser assigns its own padding, margin, line-height, font-size, etc. to every element.

To combat this, we reset the CSS, essentially wiping out all of the default styles and starting with a clean slate.

We’ve provided an example of a reset.css in the downloadable file attached to this page. Download it and start adding it to your CSS folders.

Always link to reset.css in the head section of your HTML, above the link to your main CSS file, as follows:

<!DOCTYPE html>

<html>

<head>

<title>CSS Reset Links</title>

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

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

</head>

<body>

</body>

</html>

## Floats and Clears

By default, block elements in HTML stack on top of each other. Floats let us change this behavior, allowing elements to sit horizontally on the page.

The CSS property is called float and common values are left, right, and none (the default is none).

Clears are used when we want to “turn off” floating—essentially preventing elements from flowing around the floating element.

A common method is to use the CSS property clear with the value both. When this method is ineffective, the clearfix method—as seen above—should resolve any issues.

## Column Layout

Column layout refers to the design of pages with multiple different block sections at the same vertical height on the page.

Columns are created by wrapping a block of content with a div element, applying a float, and modifying the padding, margin, and border dimensions to adjust their size.

## Questions to Ask Your Mentor

What are some strategies for sketching out my page elements that will make it easy for me to position them on my page using float and clear?

When is it best practice to add an additional column to my page layout? Are there standard practices around column layouts?

How does the inclusion of special effects (i.e. hover state) affect the overall UX (user experience) of my site?

When is the appropriate time in the web design process to introduce floats and clears to my code?

What are some good places to go for feedback or inspiration on page layout?

How do I know when to use clear:both versus the clearfix method? Are there other methods for clearing?

## Further Reading

**List Navigation**  
<http://css-tricks.com/navigation-in-lists-to-be-or-not-to-be/>

**CSS Reset**  
<http://www.sitepoint.com/css-resets-useful-or-useless/>  
<http://meyerweb.com/eric/thoughts/2007/04/18/reset-reasoning/>  
<http://www.cssreset.com/which-css-reset-should-i-use/>

**Pseudoclass**  
[http://www.w3schools.com/css/css\_pseudo\_classes.asp](http://www.w3schools.com/css/csspseudoclasses.asp)  
<http://css-tricks.com/pseudo-class-selectors/>

**Hover**  
<https://developer.mozilla.org/en-US/docs/Web/CSS/:hover>

**Float and Clear**  
<http://learnlayout.com/float.html>  
<http://learnlayout.com/clear.html>  
<http://learnlayout.com/clearfix.html>  
<http://learnlayout.com/float-layout.html>  
<http://css.maxdesign.com.au/floatutorial/>  
<http://css-tricks.com/all-about-floats/>  
<http://www.smashingmagazine.com/2007/05/01/css-float-theory-things-you-should-know/>  
<http://css-tricks.com/the-how-and-why-of-clearing-floats/>

**Column Layout**  
<http://www.thesitewizard.com/css/design-2-column-layout.shtml>  
<http://www.cssreset.com/creating-a-two-column-fixed-width-css-layout/>

# Implementing Multi-column Layout

In our last unit, you organized your code by adding generic and semantic elements to your HTML, and you gave them content and applied background-colors, padding, and borders to those elements to make them visible.

The purpose of that exercise was to emphasize that websites, at their most fundamental design, are just a bunch of blocks — blocks on top of blocks, blocks nested within other blocks, and blocks next to other blocks. The reason that we apply this emphasis is because thinking of websites as a series of blocks will make executing layouts much easier.

This week is an upgrade! You learned the skills to make your layout take shape. Get more practice and a deeper understanding of floats by watching the Unit 6 lecture.

In this week’s project, we’ll put that thought into action. You’ll upgrade your code from last week by using floats and clears to introduce a multi-column layout to your page.

## Step 1:

Open up your **index.html** file in Sublime Text.

Choose two elements that you want to have next to each other. Perhaps you want to have a couple <section> elements next to each other inside your <footer>. Or perhaps you want an <article> and an <aside>.

If you don’t plan on having multiple columns anywhere on your site, just add an <aside> for this project and use a <p> tag with lorem ipsum to give it content.

Give the first element the ID “left” and the second element the ID “right.”

If you haven’t already, nest those two elements inside a parent element. You should do this by wrapping them with an appropriate semantic element. If you’re unsure which one to use, just wrap them with a element for now. Give that parent element the ID “two\_column” and the class “clearfix”.

This additional HTML should look something like this:

<section id=“two\_column” class=“clearfix”>

<aside id="left">

<p>ASIDE<p>

<p>Lorem Ipsum…</p>

</aside>

<article id="right">

<p>ARTICLE</p>

<p>Lorem Ipsum…</p>

<p>Lorem Ipsum…</p>

</article>

</section>

Click “Save As” and save this update in the unit\_6 folder of your local GitHub repository clone. You can wait to commit, sync, and publish until you’re done with the project.

## Step 2:

Open up your style.css file. You’re going to use CSS to give your new elements some style and turn them into a two column layout.

Give the elements you’re putting into columns the following CSS so that you can easily see them when you render your code in the browser:

#two\_column {

background-color: #FFB7CF;

border: 1px black solid;

padding: 20px;

}

#left {

background-color: #E582A2;

border: 1px black solid;

padding: 20px;

}

#right {

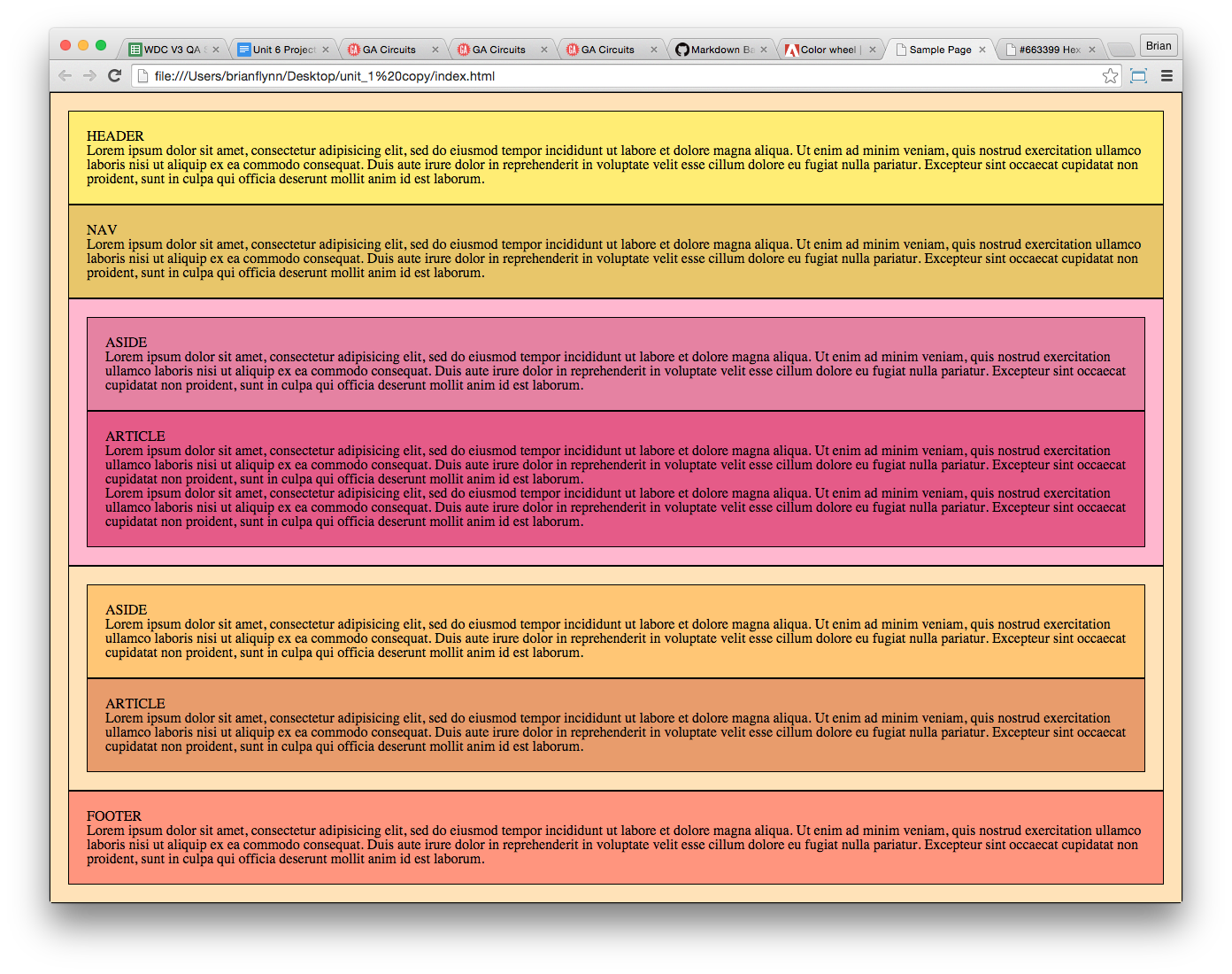
background-color: #E55B88;

border: 1px black solid;

padding: 20px;

}

If you render your code in the browser at this time, it should look something like this:



Now, let’s turn those boxes into a two column layout!

## Step 3:

Go back to your style.css file and and add the following code to your two elements:

#left {

background-color: violet;

border: 1px black solid;

padding: 20px;

float: left;

width: 30%;

}

#right {

background-color: pink;

border: 1px black solid;

padding: 20px;

float: right;

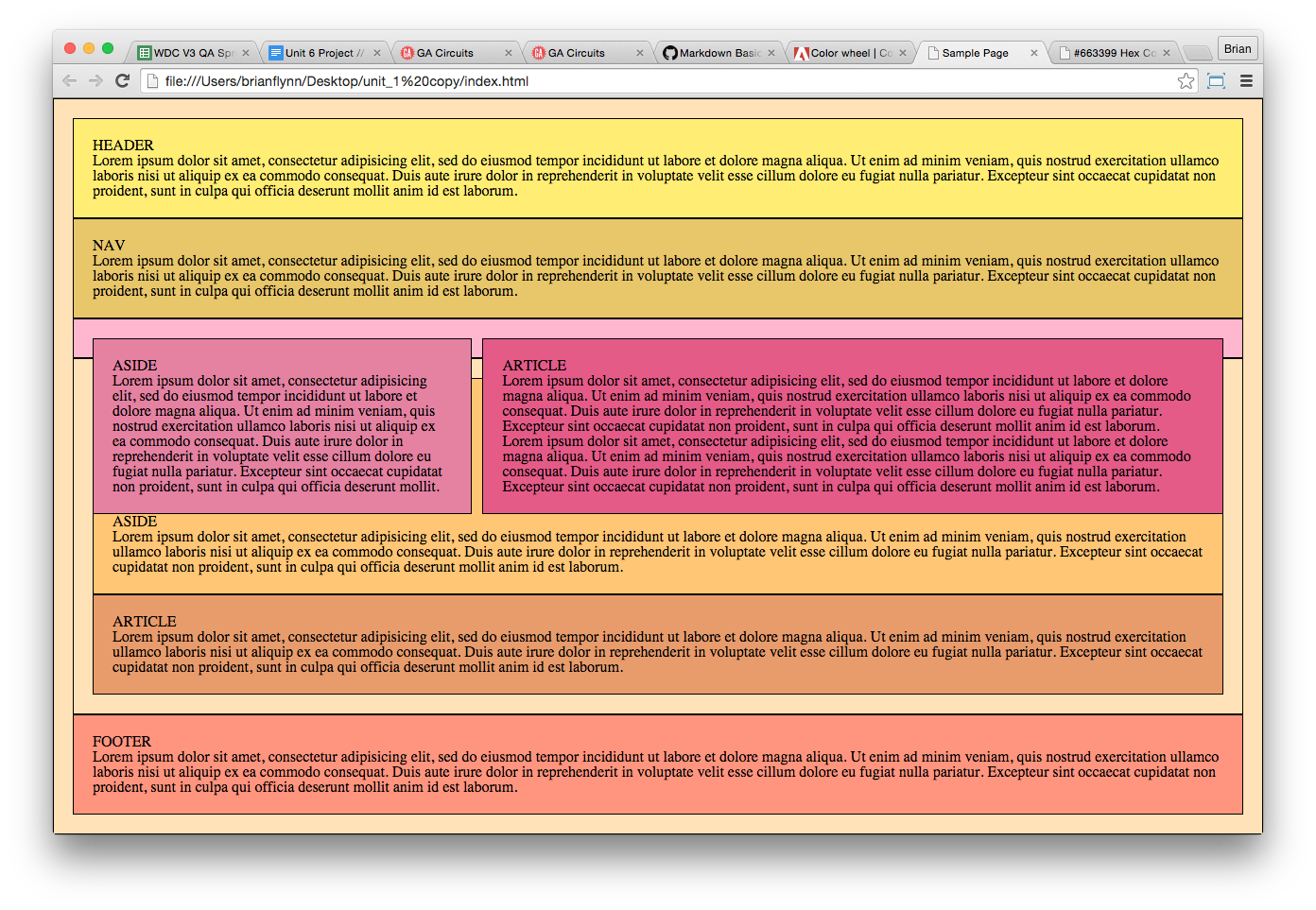
width: 62%;

}

You’ll notice we did something new with our widths — instead of pixels, we entered percentages. This technique is called “fluid layouts” and we’ll cover it in greater detail in future lessons. For now, just note that it works exactly as you might expect: the #left element’s width will be 30% of the available space and the #right element’s width will be 62%.

You can adjust these number if you want your objects to be different sizes. Just keep in mind that the total amount of space — including borders, paddings, and margins — is 100%, and you'll probably want a little extra space for padding, margin, or just room for error (we left 8% of the space open).

If you save your CSS and render your code in the browser, you’ll notice these two elements are now sitting next to each other:



Congrats! You have columns!

Unfortunately, you’ll also notice that the wrapper within which they’re nested has collapsed. This is because, once floated, elements that are nested inside a parent element no longer add to the height of that parent element.

Let’s fix this with the .clearfix method!

## Step 4:

In step 1, you should have given your wrapper element the class “clearfix” in your HTML. If you haven’t done that yet, do it now, and make sure to save the change.

Next, in your CSS file, add the .clearfix method:

.clearfix:after {

visibility: hidden;

display: block;

font-size: 0;

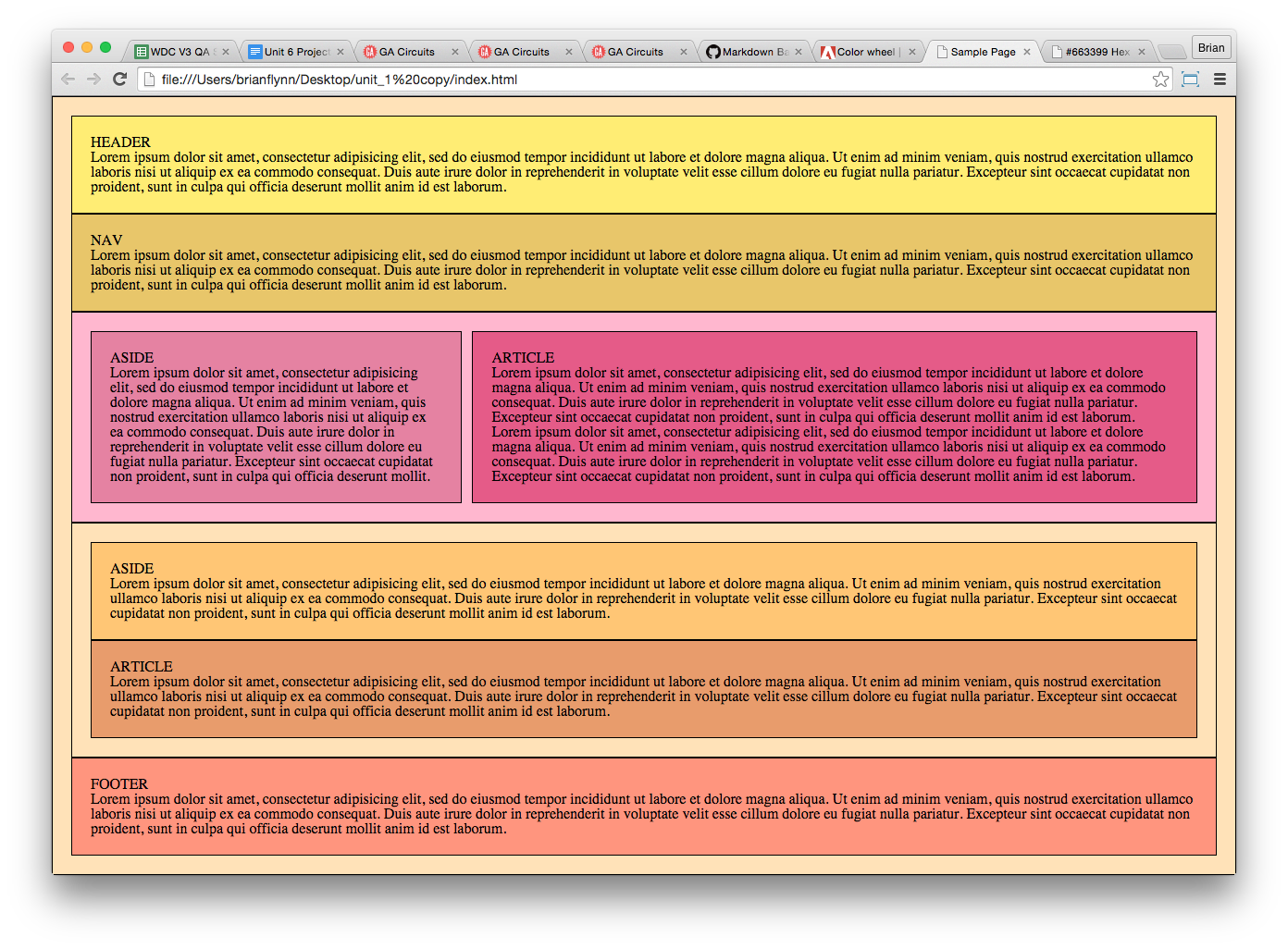
content: " ";

clear: both;

height: 0;

}

Now, when you save and render your code in the browser, it should look something like this:



Perfect! A two-column layout!

## Step 5:

Review the checklist below to make sure your project is ready to submit.

1. I’ve added the clearfix class to my CSS
2. I’ve implemented floats to put objects next to each other on the page
3. I’ve applied the clearfix class to the parent element of floated children
4. I booked a 1:1 this week because I’m more than halfway through the course!

**Want feedback on your progress so far?**  
Book a 1:1 with your mentor.

## Step 6:

To submit your project, save your HTML and CSS files to the **unit\_6** folder in your local GitHub . Then, using the GitHub Desktop app, commit and sync these changes with your online repository.

Once synced, enter the URL for your site as the project URL. If you’ll recall, your URL should look something like this:

repository\_name.github.io/unit\_6/index.html

If you have any issues, let your mentor know during your one-on-one session, and they’ll help you out!

## Note:

Once you've reviewed this project with your mentor, feel free to replace or remove the borders and background colors from your generic and semantic elements to fit whatever style you have in mind. Let us know if you found having background colors helpful while learning these concepts.

[←](https://circuits.generalassemb.ly/student/209/quizzes/237?direction=back) [→](https://circuits.generalassemb.ly/student/209/assignments/160)

### Projects

[Submit my project](https://circuits.generalassemb.ly/student/209/assignments/160/projects/new)

* [[Nr7ivyac5kisv369mxfsTiny img 80151](https://circuits.generalassemb.ly/student/profiles/4393-christopher-lopez/projects/20511)Unit\_6](https://circuits.generalassemb.ly/student/profiles/4393-christopher-lopez/projects/20511)
* [[C3ueoovcswygcoughqoqTiny 0bc4952](https://circuits.generalassemb.ly/student/profiles/4398-sam-scherma/projects/19345)Tribute Portraits Floats, Clears and Columns.](https://circuits.generalassemb.ly/student/profiles/4398-sam-scherma/projects/19345)
* [[Q6mvdbhtxjgebbwaoncx](https://circuits.generalassemb.ly/student/profiles/4435-vanessa-meza/projects/20420)Unit 6](https://circuits.generalassemb.ly/student/profiles/4435-vanessa-meza/projects/20420)
* [[Lehzoqbcjiha4xu2upstTiny darrell o profile](https://circuits.generalassemb.ly/student/profiles/4488-darrell-harris/projects/19366)japan combines homepage](https://circuits.generalassemb.ly/student/profiles/4488-darrell-harris/projects/19366)

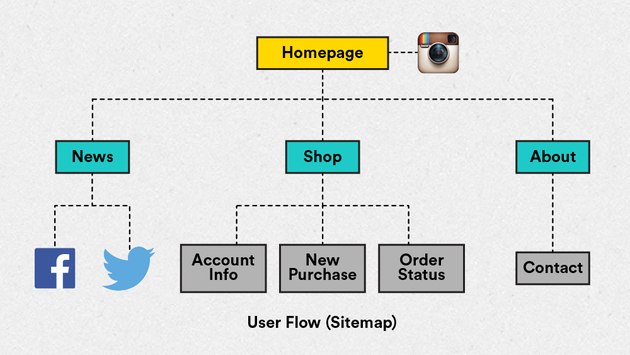
© 2016 General Assembly. All rights reserved. [Regulatory Information](https://generalassemb.ly/regulatory-information) [Terms](https://generalassemb.ly/terms_and_conditions) [Privacy](https://generalassemb.ly/privacy_policy) [Assembled in NYC](http://nytm.org/made-in-nyc)

# Review Guide: Designing Across Platforms

Let’s review some of the key ideas introduced in this unit. You can also download all this content by clicking on the attached file for future reference.

## Key Definitions

* **Device Agnostic**: This suggests designing for communication to the user, not for any specific device.
* **UX Design**: A process to make each step in your page navigation as simple and clear as possible for users.
* **Wireframe**: Outline or sketch of the content areas of a page with an eye to focal point and hierarchy.
* **Prototype**: An early site model, either digital or analog, that can serve as the basis for research and design iterations.
* **User Flow**: Illustrates the primary and alternate paths you intend your user to take through the site, based upon user goals.

  
- Media Query: A technique used in CSS to detect the display size and subsequently activate different page styles.  
- Hamburger Icon: A commonly used page element, consisting of three lines bounded by a rectangular block. Typically used as a link to the page menu or a vertical list. Here are two enlarged images showing what this symbol normally looks like:



## Mobile First

The practice of mobile first design suggests first designing for a smaller screen size (i.e. smartphones) and then expanding the design to larger screen browsers.

The term "mobile first" was coined by Luke Wroblewski. Check out a presentation on the topic\*[*here*](http://www.lukew.com/ff/entry.asp?1137).\*

## Design Considerations for Mobile

**Physical characteristics:**  
Different devices have different physical characteristics, i.e., screen size, screen resolution, and different guidelines for best practices.

Consider: Designing to suit the devices most frequently used by your audience.

**User ergonomics:**  
Devices (tablet and phone) are held by hand and navigated by hand. Usage may be single, or in some cases double-handed, with finger(s) as the primary input device.

Consider: The context in which your site might be viewed (e.g., a crowded train vs a quiet bedroom). Is your mobile site easy to navigate / use?

**User navigation via touch:**  
On mobile, the finger acts as a mouse hitting buttons and links. Often, users will “drive with one thumb and one eye.”

Consider: Designing controls that are within reach and with enough contrast to surrounding elements.

**Gestures:**  
Familiarity with standard gestures is key to designing interactions already familiar to users (ease of use) and creating engaging experiences.

Consider: Gestures may or may not be of concern for basic websites. While gestures are dominant in app design, basic sites may just require tap to click. Still, an awareness of common gestures helps designers understand their users' mindsets.

## Mobile Websites vs. Responsive Websites

**Mobile websites:**  
Sites that are specifically coded for mobile devices. Generally speaking, such sites are made in addition to a standard desktop site and are often used on sites where the mobile experience needs to be incredibly different from the standard desktop experience.

**Responsive websites:**  
Sites that “respond” to changes in the size of the browser on which they’re being viewed. This allows creators to only build and maintain one site and provide a fully cohesive experience to their users.

## Image Optimization

In order to shrink an image’s file size, you have to reduce its width and height prior to uploading it to the web using an image editing program like Microsoft Paint, Apple Preview, or Adobe Photoshop.

There are a couple issues you need to be aware of when reducing an image's size:

Most graphics — especially photographs — cannot be expanded once shrunk without losing resolution, so never shrink an image smaller than the largest size at which it will appear on your site.

When adjusting image sizes, always do so proportionally. If you need to change the shape of an image, crop it. Don't disproportionally adjust its size.

Whenever possible, save an image as an screen vector graphic or SVG. This is only possible for simple, illustrated artwork, like logos or icons. SVGs can be scaled to ANY size on your web page that you want without increasing the file size.

## Mobile Limitations

UX for mobile can be especially difficult for a variety of reasons:

* Limited screen space
* Less stable internet connections
* Users are often in motion
* Users are often more hurried

Keep these facts in mind when designing for mobile. Think about how your content is prioritized, ensuring that the most important content is easily accessible, even for users on the go or with slower internet connections.

## Questions to Ask Your Mentor

* Is Mobile First an approach I should always use?
* What are the differences between mobile design and app design that I should be aware of?
* Where can I find guidelines for designing to mobile interface standards?
* What are the steps I should take in order to plan a mobile site?
* How can I possibly design for every mobile device?
* Where can I find guidelines for designing to mobile interface standards?
* What is UX design and where does it fit in the process?

## Further Reading

Presentation on Mobile First  
<http://www.lukew.com/resources/mobilefirst.asp>

Apple Human Interface Guidelines  
<https://developer.apple.com/library/iOS/documentation/userexperience/conceptual/mobilehig/>

Planning and Production Tools  
<http://shop.oreilly.com/product/0636920001133.do>  
<http://www.adobe.com/products/illustrator.html>  
<http://www.adobe.com/products/indesign.html>  
[http://www.axure.com](http://www.axure.com/)  
<http://bohemiancoding.com/sketch/>  
[http://balsamiq.com](http://balsamiq.com/)

Adobe Digital Publishing Report  
h[ttp://www.canerel.com.tr/v2/images/publication/201301AdobeDPSshoppingSURVEY.pdf](http://www.canerel.com.tr/v2/images/publication/201301AdobeDPSshoppingSURVEY.pdf)

# Responsive Mock-ups

This week had a lot of concepts and not a lot of code. Don’t let your skills grow stale! Review the unit 7 lecture to stay ahead of the curve.

So far, you've designed and built a functional desktop website, complete with an external style sheet. That’s a huge accomplishment, and we hope you feel great about your experience so far.

Before you’re done, there’s one last big step you need to take: you need to make your page responsive so you can show it off on screens of all sizes!

We’ll walk through this process in two steps. First, in this week’s project, you’ll design mockups for how you want your site to appear on tablet and mobile sizes. Next week, you’ll implement the code to bring those designs to life!

Ready? Let’s get started!

## Step 1: Research

First, let’s get in the right mindset by checking out examples of responsive design on [http://mediaqueri.es](http://mediaqueri.es/). This site showcases a number of responsive websites and makes it easy to compare designs across desktop, tablet, and mobile.

As you browse mediaqueri.es and explore different responsive sites, observe their overall design direction, how mobile and desktop navigation is handled, and whether or not transitions, menus, and gestures are used effectively.

Once you’ve spent some time reviewing examples and have a general idea of responsive design solutions, revisit some of the sites you looked at in earlier units that have similar goals to the one you’re creating. How do they adjust their designs to respond to different screen and browser sizes?

As you check out other sites for research, remember to:   
1) Navigate the site on both a mobile device and a desktop browser.  
2) Resize the desktop browser window to reveal the site’s different states.  
3) Consider the different contexts and mindsets desktop and mobile users might have while visiting each site.

## Step 2: Designing Mock-ups

Now that you've found some inspiration, it's time to start imagining how you'd like your own site to look on mobile devices. Using [wireframe.cc](https://wireframe.cc/) (or the design software of your choice), create 2 new mockups: one for smartphones and one for tablets.

The process for creating mobile mockups will be the same as it was for creating desktop mockups in Unit 2. It's just a bunch of boxes. The only difference is you can select the tablet and mobile device sizes using the icons in the upper-lefthand corner.

Try to challenge yourself to consider the nuances of each device—the screen size, user control, content organization, and navigation. Additionally, rather than starting from scratch, try to consider ways in which your current design could be reformatted to adjust to different screen sizes.

Ask yourself:  
- How will my layout be affected by the smaller width of tablet and mobile screens?   
- Which objects will need to be resized to fit smaller screens?  
- Will the order of my content change based on the user's priorities while accessing my site on the go?  
- How will the navigation menu change with the new layout?  
- Where should I place the menu?  
- How do I want users to navigate my page? What will be the primary “Call-To-Action”?  
- Will the color, font, or image scheme change?  
- What order should the content “stack”?

The process for creating your mockups should be similar to the process we used in the Unit 2 project. If you have any difficulty remembering how to create designs on [wireframe.cc](https://wireframe.cc/), revisit the Creating Design Mockups slideshow in Unit 2.

## Step 3: Double-check!

Review the checklist below to make sure your project is ready to submit.

1. My design looks how I imagined it
2. There are no elements left un-styled or under-styled on my page
3. My layout is easy to understand
4. I’ve established proper hierarchy to convey importance and create flow
5. I’ve established a max-width wrapper to control your content on different sized monitors?\*

\* This is some advanced stuff! Help is just a click away.  
**Book a 1:1 with your mentor.**

## Step 4: Submitting Your Mock-ups

Once you’ve completed your mock-ups, take screenshots of them as you did in the Unit 2 project, then place files in the **unit\_7** subfolder of your local GitHub repository. Make sure you commit and sync your changes so that your finished designs are uploaded to your online repository. Then, just as you did in Unit 2 and Unit 5, copy and paste the link from your **unit\_7** folder to submit your project.

# Review Guide: Responsive Design

Let’s review some of the key ideas introduced in this unit. You can also download all this content for future reference by clicking on the attached file.

## Responsive Web Design

As more and more users are accessing the internet from smart phones and tablets, this means that your site needs to be able to perform well on much smaller screens. However, simply making everything smaller might make parts of your page difficult to read or use.

Responsive Design means designing a single site that adjusts to provide an optimal viewing experience for every device on which a site could be viewed.   
To master responsive design, there are three tools available: fluid layouts, media queries, and ems.

## Fluid Layouts

Fixed Width Layouts vs. Fluid Layouts

“Fixed-width layout" means that regardless of the browser or device screen size, the elements on the page will maintain the same fixed pixel width.

Fixed-width layouts are fine for sites that are only intended to be seen on one type of device; however, they quickly experience problems when launched on a screens of different sizes.

"Fluid layouts" (also known as “liquid layouts") offer a solution to this issue. While fixed-width layouts are defined with hard-coded pixel values, fluid layouts define space with percentages. By using percentages, we are able to create layouts that shrink and expand relative to the browser and device size.

Fluid Layout Example

div {

width: 100%;

}

Fixed Width Layout Example

div {

width: 1024px;

}

## Media Queries

Media queries allow us to retrieve information about the type of device that’s viewing our site and deliver CSS styles accordingly. This allows us to make changes that go beyond just shrinking the space of elements on the screen.

We can un-float and rearrange elements, alter our navigation items, and hide and show elements, all based on the width of the device that is rendering our page.

Media Query Example

In the following example, we are telling the browser that's loading our page, “If the maximum width of the browser is larger than 500 pixels, change the font-size of the body to 12 pixels.”

@media screen only and (min-width: 500px) {

body {

font-size: 12px;

}

## Ems

Ems are a unit of measurement, like pixels; however, unlike pixels, which are absolute, ems are relative to the font size of their parent element. If the parent element is not given a size, ems will default to 16px on most browsers.

The em allows us to define and measure elements in relative terms. In combination with our media queries, this allows us to set the typography sizing JUST BY setting the font size value on the body.

Em Example

body {

font-size: 10px;

}

.em-unit {

background-color: #FF6A5E;

width: 4em;

height: 2em;

}

Fixed-Width Example

.fixed-unit {

background-color: #FF6A5E;

width: 40px;

height: 20px;

}

## Target Sizes for Media Queries

The following guide is designed to give you a sense  
of the diversity in sizing on target devices. Generally speaking, when building your code, you should avoid targeting specific devices and instead use your media queries to specify general device widths and heights.

Only target specific devices if you expect that specific device to be your users' primary access device.

**HTC One**  
Device Width: 360  
Device Height: 640

**Samsung Galaxy S4/S5**  
Device Width: 320  
Device Height: 640

**LG Nexus 5**  
Device Width: 360  
Device Height: 592

**iPad Mini**  
Device Width: 768  
Device Height: 1024

**iPad 3 / iPad 4**  
Device Width: 768  
Device Height: 1024

**iPhone 4**  
Device Width: 320  
Device Height: 480

**iPhone 5**  
Device Width: 320  
Device Height: 568

## Questions to Ask Your Mentor

* Do I need to use fluid layouts, media queries, and ems in order to make my page responsive?
* When starting from scratch, what is the best process for building a responsive website?
* How can I test my page to ensure that it will be responsive across different browsers (without having to go to each different type of browser on every device)?
* Are there other approaches to building pages for various browsers and devices?
* How do I determine which elements of my page to highlight as the screen size “shrinks”?

## Further Reading

Looking for more help with Responsive Web? Here are a couple places you can check out:

Responsive Design  
<http://johnpolacek.github.io/scrolldeck.js/decks/responsive/>  
<http://designmodo.com/responsive-design-examples/>  
<http://responsivedesign.ca/>  
<http://www.smashingmagazine.com/responsive-web-design-guidelines-tutorials/>  
<http://blog.froont.com/9-basic-principles-of-responsive-web-design/>

Fluid Layout  
<http://www.smashingmagazine.com/2009/06/02/fixed-vs-fluid-vs-elastic-layout-whats-the-right-one-for-you/>  
<http://www.takeflyte.com/resources/newsletters/04/12-fluid-v-fixed-web-pages.php>  
<http://green-beast.com/blog/?p=199>  
[http://www.stuffandnonsense.co.uk/archives/fixed\_or\_fluid\_you\_decide.html](http://www.stuffandnonsense.co.uk/archives/fixedorfluidyoudecide.html)

Media Queries  
<https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Mediaqueries>  
<http://code-tricks.com/css-media-queries-for-common-devices/>

Ems  
<http://kevinperalta.com/playground/emsize.html>

Testing Responsive Design  
<http://www.jamus.co.uk/demos/rwd-demonstrations/>  
<https://chrome.google.com/webstore/detail/responsive-web-design-tes/objclahbaimlfnbjdeobicmmlnbhamkg>

Footer Anchor  
Description: <http://codepen.io/bradfrost/full/mlyvu>  
Code: <http://codepen.io/bradfrost/pen/mlyvu>

Expanding Nav  
Description: <http://codepen.io/micahgodbolt/full/mnLiF>  
Code: <http://codepen.io/micahgodbolt/pen/mnLiF>

Toggle Menu (Hamburger):  
Description: <http://blog.lavoie.sl/2013/11/responsive-menu-in-pure-css.html>  
Code: <http://codepen.io/lavoiesl/pen/KthjD>

Also, as usual, please feel free to reach out to your mentor.

# Build a Responsive Page!

This is the last week with a corresponding lecture. That means you’ll be off on your own soon! Check out that last lecture for final tips & tricks.

## Build a Responsive Page!

So, you’ve got a website. You’ve got mock-ups for how this page might appear on mobile devices. Time to make it fully responsive!

In this project, we’re going to put the three main aspects of responsive design into action by adding fluid layouts, media queries, and ems to your website code. After you finish this project, your website will be almost completely finished.

## Step 1: Getting Started

In the last project, you created mock-ups and received feedback during your 1 on 1 session with your mentor for what your responsive page will look like on mobile browsers. Now it’s time to put this design into action.

Open up your HTML and CSS files on Sublime Text and save them to the unit\_8 subfolder of your local GitHub repository.

Next, in your HTML, add the following meta-tag inside your element:

<meta name="viewport" content="width=device-width, initial-scale=1">

Adding this will ensure that your future media queries work properly. Save this update, then switch over to your CSS file.

## Step 2: fluid layout

Swap out your fixed-width layout for a fluid layout.

Find elements that you want to adjust with the browser size. Typically, this means block elements for which you’ve determined a specific width, like semantic elements, divs, and images. Switch their widths from pixels to percentages.

If you want an item to take up the entire width of the screen, set it to 100%. When adjusting nested elements, keep in mind that their percentages are proportional to the size of their parent element. Meaning, if you have a element that you’ve set to take up 80% of the page, and you want an image nested inside that to take up its full width, you would set the image at 100%.



Also, remember to adjust the padding and margin of your selected elements, too.

Once you’ve switched your selected elements from pixels to percentages, save your work and render it in the browser. Start with your browser set at a typical laptop width, like 1280px. Look for any areas where your layout doesn’t match your original design and see if you can adjust appropriately.

## Step 3: Media Queries

Once you’re satisfied with your fluid layout at a typical laptop width, shrink the browser to look for your first breakpoint (e.g., the navigation bar breaks into multiple lines, columns get too narrow, headlines look cramped, images become too small, etc.).

Identify the pixel width at this point using Chrome’s Inspect Element feature. Then, using this pixel width, create a media query at the bottom of your CSS to adjust your layout and fix the breakpoint.

Once it’s fixed, resume shrinking the browser to seek your second breakpoint, and repeat the fixing process. You should expect to need about 2 or 3 media queries to make your page work properly at all sizes.

Think of it this way—you’ve defined the CSS that desktop browsers will render at the top of your stylesheet; now you need to include the CSS that your site will need to display well on other types of screens.

## Step 4: Ems

Change your fonts from pixels to ems.

Remember, ems are based off of the font size of their parent element. Since the element is the ultimate parent element, give it a font size to set a base. We recommend using 16px as that base, which would mean that 1em = 16px, 2em = 32px, 0.5em = 8px, etc.

Once you’ve changed the fonts in your main CSS, you can adjust font sizes in your media queries more easily. Instead of changing each font one by one in your media queries, you can just adjust the font-size of the body element.

## Step 5: Test Your Page!

Once you’ve completed modifying your code, make sure to save your CSS and HTML to the unit\_8 subfolder of your local GitHub repository. Then, commit and sync your changes to your online repository.

Once your website is online, using your GitHub URL ( which should look something like this: <http://your_repository_name.github.io/unit_8/index.html>) open your site in the browser and shrink the window to see if all of your changes work. Then, using that same link, open your site on some mobile devices. Send it around to friends with different tablets and smart phones and see if it works.

If you find any issues for which you don’t know the solution, take note and bring it up with your mentor during your 1 on 1 session.

## Step 6: Double-check

Review the checklist below to make sure your project is ready to submit.

1. My design does not break anywhere on desktop
2. I’ve tested my design on a mobile and/or tablet device\*
3. I’ve tested my how my site looks in different browsers
4. My images are optimized for the web
5. I’ve addressed any usability concerns

\*Don’t have a smartphone or tablet? Your mentor does! Reach out to us and we’ll help you test.

**Have other questions?**  
Book a 1:1 with your mentor.

## Above and Beyond

Try using ems to adjust other text-related sizes, like line-height, letter-spacing, word-spacing, and padding and margins on heading and paragraph elements.

## Note:

The process of changing all this code can be overwhelming the first time you do it, so we’ve included the source code from the Elk Design code challenge as a reference for you.

Throughout the process, save and test your code frequently—this will help you to narrow in on the specific elements that you need to modify. Try resizing your browser to make sure your media queries are working properly.

As always, reach out to your mentor with any questions.