

FUNDAMENTALS OF WEB DESIGN

THIRD NEW EDITION • UNIT 2

BY LUIS POZA

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <title>Ponta's Page</title>
5     <meta charset="UTF-8">
6     <meta name="author" content="Your Name">
7     <link rel="stylesheet" href="styles.css">
8   </head>
9   <body>
10    <div id="wrapper">
11      <header>
12        <h1>Ponta!</h1>
13        <h2>A Fuzzy Shiba Inu!</h2>
14      </header>
15    </div>
16  </body>
17 </html>
```

ART280 • Fundamentals of Web Design • Lakeland University

By Luis Poza; copyright 2023. This text is not to be distributed outside the LUJ design class.

CLASS TEXTBOOK **UNIT 2**, Chapters 5 ~ 8

Chapter 5: Beginning with CSS

5a. Multilingual Pages	98
5b. CSS Basics	99
5c. Inline CSS	102
5d. Embedded CSS	103
5e. Stylesheets	105
5f. Background Images	109
5g. CSS RGB Color	113
5h. Gradients	116
5i. Color and Web Sites	117
EX. Exercise 5-1	120

Chapter 6: CSS Selectors

6. Setting Up	125
6a. Selectors: Class and Id	126
6b. The Wrapper Div	130
6c. More CSS Selectors	132
6d. CSS Shorthand	134
6e. Some Basic CSS Effects	136
6f. Making Different Page Styles in One Stylesheet.....	140
6g. Centering Inline and Block.....	143
6h. CSS Ordering	146
6i. Using Google Fonts	148
6j. Using @font-face	151

Chapter 7: The Nav Bar Layout

7a. The Cascade.....	154
7b. Site Structure	156
7c. Creating a Simple Menu for a 2-Level Site	158
7d. Pseudo-Classes & Pseudo-Elements	167
7e. The Transition Animation.....	171

Chapter 8: Boxes and Backgrounds

8a. Project #1.....	174
8b. Div and Span.....	180
8c. Using Boxes.....	182
8d. Floats & Clears	183
8e. More Fun with Effects	187
8f. Site Planning: Site Maps & Wireframes	189
EX. Exercise 8-1	194

Chapter 5: Beginning with CSS

TECHNICAL

5a. Multilingual Pages

Web pages can consist of several different coding languages existing side by side within the same document. For example:

```
<main style="padding: 10px 30px; font-size: 12pt;">
  <div class="button" onClick="alert('Hello World!')">
  </div>
  <? include "intro.html"; ?>
</main>
```

In the above code sample, there are four languages:

```
blue: HTML
red: CSS
green: Javascript
orange: PHP
```

The main code is HTML.

CSS is embedded using the `style=""` attribute; everything inside the attribute's value quotation marks must be CSS, and must follow CSS rules.

Javascript is embedded here using an event attribute, in this case, the `onClick` event—meaning that when the element is clicked, the Javascript will be executed. Everything within the event attribute's value quotation marks must follow the rules of Javascript.

PHP is embedded using the `<? ~ ?>` element, which can also be expressed as `<?php ~ ?>` if you want.

In each of these cases, the new languages have their own areas within the attributes or tags which are used to introduce them. Think of them as being like embassies. Embassies are within foreign countries, but once you enter the embassy property, it becomes a new country where that country's laws are in force.

CSS can also be introduced to a page using the `<style>` tag; in that case, everything inside the beginning and ending style tags must follow CSS rules. For example, in HTML, a comment is added by using the declarative tag `<!-- comment -->`, in that manner. However, inside the style tags, you must use `/* comment */` instead—the CSS format for a comment.

Most often, these additional languages are not added inline (completely within the HTML as shown above), but instead **have their own documents** which are linked to in HTML. In the HTML, the `<link>` tag is used to connect the HTML file to the CSS file, and the `<script>` tag is used to connect Javascript and PHP code files to the HTML file.

CODE

5b. CSS Basics

HTML is used to create *structure and semantic meaning*. CSS is used for *styling*.

When the Web was new in 1991, HTML was used for everything, including styling. However, the styling ability of HTML was weak.

Most old HTML tags and attributes used for styling are now **deprecated** (no longer in use) and are no longer used. Examples of deprecated HTML (**do not use these!**) :

<code><center></code>	used to center something in the middle of a space
<code></code>	used to set fonts , font sizes, and font colors
<code><u></code>	used to create <u>underlined</u> text
<code><s></code> or <code><strike></code>	used to create striketrough text

These tags still mostly work, but anyone who sees them will believe that the web coder is ancient and does not know modern code. The reason why is that these tags are also very limited and weak for styling. Therefore, instead, we now use CSS.

How CSS Works

CSS was designed for styling. CSS is incredibly powerful, and can be used to make precisely styled pages. It was introduced in 1996, but was not widely used until about 2000. You can sometimes still find pages made in the 90's and if you check the code, it will be HTML only.

CSS is a different language than HTML. It has different rules. It can be added to an HTML file, or it can appear in separate documents called *stylesheets*.

CSS creates **styles**. In fact, the word "style" is the signal word for CSS; when you see the word "style" in code, it probably refers to CSS code.

A "style" is when some object on a page, such as an `<h3>` heading, can be changed in color, font, size, or other visual properties. You can make an `<h3>` heading so that the font is Palatino 17pt, the text color is dark blue, and that the background of the heading is light blue.

A central part of CSS styles is the **declaration**. A declaration is a single style point, like a width, height, color, or size.

Each declaration has two parts: the **property** and the **value**.

The property describes *what will be changed*, such as color, font-size, or width.

The value describes *how it is changed*, such as by specifying a color, a size, or a state.

The **property** is *always* followed by a **:** colon, and a **value** is *always* followed by a **;** semicolon.

It looks like this:

```
color: red;
```

CHAPTER 5

Each property and value is always expressed as a single word *with no spaces*. For example, the property "font size" is expressed as `font-size`, with a hyphen, so there is never a space within the property name. The same is true for each value, such as the `small-caps` value:

```
font-variant: small-caps;
```

No property or single value is allowed to have a space. A space signals a *new* value.

Sometimes, instead of a hyphen, words are joined without any space or punctuation between them, as they are with HTML color names:

```
color: darkblue;
```

Sometimes, there are **multiple values** within one property. For example, in the "font family" property, you must specify both a font family name and a category. These are divided by spaces, even while each value may be hyphenated:

```
font-family: 'Palatino', serif;
```

Declarations like these do not exist by themselves. They are always applied to some HTML tag, in one way or another. Everything inside the tag is affected by the style.

DEFAULT STYLES

Most HTML tags *already* have CSS styles applied to them. For example, the `<h1>` tag has this CSS styling automatically:

```
display: block;
font-size: 2em;
margin-top: 0.67em;
margin-bottom: 0.67em;
margin-left: 0;
margin-right: 0;
font-weight: bold;
```

The CSS says that an `<h1>` tag is a block tag; it has a font size double normal text; it has top and bottom margins 2/3rds the height of the text inside of it; it has not left or right margin; and the font weight is bold. You may have noticed this is true of `<h1>` tags.

UNITS

CSS can use a variety of units:

px	pixels
pt	point (used for fonts)
mm	millimeters
cm	centimeters
in	inches
em	the standard size; for line-height, 2em is double spacing.
%	a percentage of the total or standard size

The above units are used at various times to get the best possible effect.

INTEGRATING CSS INTO HTML

CSS is a completely different language than HTML. It uses different rules and different syntax. As a result, you cannot just mix the two together any way you like.

CSS is added to HTML in three different ways:

- **Inline CSS:** CSS code is added within a single HTML tag using the `style` attribute;
- **Embedded CSS:** CSS code is added in the `<head>` of a single web page using the `<style>` tag; and
- **CSS Stylesheets:** CSS code is added using a separate document which is linked to all or many of the web pages in your site.

In all three cases above, CSS has its own "territory," its own area where CSS rules are used.

In the next three chapter parts, I will introduce each one.

However, **keep in mind that CSS is usually only added by using the last method, stylesheets (external CSS), and you are expected to use only stylesheets in my class.** I need to teach you inline and embedded CSS so that you know when and can use them. I will sometimes use one of those types for class examples and/or for convenience. However, in your projects, I expect you to use only stylesheets.

5c. Inline CSS

In **Inline CSS**, the CSS code appears within each tag. The **style** attribute is used. Everything inside the "value" for the style attribute is CSS code.

This type of CSS is simple—you only use the **property** ("font-size") and the **value** ("17pt").

```
<h3 style="font-size: 17pt;">
```

Anything inside the quotation marks is CSS; anything outside the quotes is HTML or text.

You can have more than one declaration in any one tag:

```
<p style="font-size: 13pt; line-height: 1.3em; text-indent: 0.5in;">
```

In the above example, for that one paragraph, the font size will be 13pt, the line spacing will be 1.3 times the normal single spacing, and there will be a first-line indent of 0.5 inches.

TRY IT

Below is an example of Inline CSS in code. Create a web page in your code editor and try this code for yourself.

```

12 <body style="background-color: pink;">
13 <header>
14 <h1 style="font-size: 32pt; color: darkred;">
15     The Site Title
16 </h1>
17 <h2>
18     The Site Subtitle
19 </h2>
20 </header>
21 <main>
22 <h3 style="color: red;">
23     The Page Title
24 </h3>
25 <p style="font-size: 13pt; line-height: 1.3em; text-indent: 0.5in;">
26     Lorem ipsum dolor sit amet, pri debet mucius erroribus in, nec alterum scripserit ea, at sit eruditi
27     labores. His fugit semper admodum id, at est clita reprimique. Ne antiopam definiebas mei.
28     Mediocrempertinacia ut eam, ea illud homero principes sit.
29 </p>
30 </main>

```

Note that the HTML lines 12, 14, 22, and 25 all have inline CSS. Copy the code above into a complete HTML file. Save the page and view it in a browser. Try making changes to the values—the color names or numbers used in the CSS, and see how the changes look.

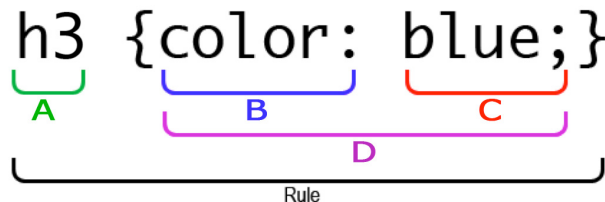
Keep in mind that Inline CSS will *only* affect the *one* tag in which it is used. As a result, inline CSS is very wasteful—it must be repeated endlessly, inside every single tag that is affected by it. It is rarely used, only in special situations.

The other forms of CSS are much more efficient.

5d. Embedded CSS

The two other forms of CSS, *Embedded CSS* and *Stylesheets*, use **Rules**. Each rule affects certain *groups* of HTML tags. Which tag is affected is shown in the **selector**. The selector is followed by { curly braces } which contain CSS styles.

In the example below, the selector is h3, so **all** h3 tags which this rule affects will be blue. If this rule affects four <h3> tags, all will be changed. The parts of the rule are:



- A. **Selector**
- B. **Property**
- C. **Value**
- D. **Declaration**

The CSS styles are expressed in **declarations**. Each declaration sets a very specific style, like color, font size, width, etc. Each declaration has two parts: the **property** which defines what will be changed, and the **value** which defines how it is changed.

Inside the curly braces, there is a declaration. The declaration has the "color" property, which sets the color of text within the h1 tag. The value is "blue," so the text will be blue.

In the example above, the code is all on one line. I did that so it would be easier to label the parts. However, in actual use, you would not do this because it would make the CSS very hard to read. For example:

```
p {margin: 1.4em 0px; font-family: 'Garamond', serif; font-size: 13pt;
color: rgb(40,40,50); font-style: normal; text-indent: 0.5in; line-height:
1.3em;}
```

That style of code would make the CSS extremely difficult to read and to figure out. As a result, when we make a CSS rule, we put each declaration on a separate line:

```
p {
margin: 1.4em 0px;
font-family: 'Garamond', serif;
font-size: 13pt;
color: rgb(40,40,50);
font-style: normal;
text-indent: 0.5in;
line-height: 1.3em;
}
```

That code is *much* easier to read and to edit. Each style point is clear, and the punctuation is much easier to check.

EMBEDDED CSS

In **Embedded CSS**, the CSS code is inside the `<style>` tag of a single web page; anything between the start and end tag must be CSS, using CSS rules:

```
<style>
  h3 {
    font-size: 17pt
    color: blue;
    text-align: center;
  }
</style>
```

You can have as many rules as you like inside the embedded `<style>` tag.

In the above example, a rule is created which says that all `<h3>` tags will be 17pt and blue. This type of CSS has the advantage of changing multiple tag's styles with just one bit of code. That is better than inline CSS, which must repeat for every tag that will be affected.

However, only the `<h3>` tags *on the page with the embedded CSS* will be affected. The rest of the pages on the site will not be affected. Embedded CSS changes one page only, and does not affect other pages.

Because web pages in a site are expected to have mostly similar styles on all the pages, this makes Embedded CSS weaker and less desirable than the most powerful type of CSS, which is Stylesheets.

TRY IT

Below is an example of Inline CSS in code. Create a web page in your code editor, make sure you use `h1`, `h3`, and a few `p` tags, and try this code for yourself.

```
3  <head>
4  <title>
5    Title
6  </title>
7  <meta charset="UTF-8">
8  <meta name="author" content="yourname">
9  <style>
10     body {
11       background-color: pink;
12     }
13     h1 {
14       font-size: 32pt;
15       color: darkred;
16     }
17     h3 {
18       color: red;
19     }
20     p {
21       font-size: 13pt;
22       line-height: 1.3em;
23       text-indent: 0.5in;
24     }
25   </style>
26 </head>
27 <body>
```

As before, type this code into a complete file, and then see what it looks like in a browser. Then change the values (e.g., colors, numbers) and see what they look like.

5e. Stylesheets

CSS Stylesheets are the best way to use CSS in a web site.

In **CSS Stylesheets**, all the CSS is in one file, such as **styles.css**, where everything in the document is in CSS code, a list of CSS rules—there is no HTML. The stylesheet is then linked to from each page, which makes each page affected by the stylesheet's styles.

To create a CSS stylesheet, open your code editor and make a new page. When you choose a language, this time **choose CSS** as the language, and save the page immediately. When you name it, use "styles.css"; normally, you can choose any filename you want, so long as it ends with **.css**, but in most people use "styles.css".

The entire document is CSS rules:

```
body {
    margin: 0px;
    background-color: rgb(150,200,255);
    font-family: 'Garamond', serif;
}
header {
    font-family: 'Arial', sans-serif;
}
h1 {
    margin: 20px;
    font-size: 34pt;
}
main {
    min-height: 400px;
    background-color: lightblue;
}
p {
    margin: 10px;
    font-size: 13pt;
    line-height: 1.3em;
}
```

Creating a CSS document by itself will do nothing, however, unless the HTML pages are **linked** to the stylesheet. You would do this with the <link> tag, which must be located **inside the <head> tag**. The <link> tag has two attributes:

```
<link rel="stylesheet" href="styles.css">
```

In the first attribute, you define the **relationship** between the HTML and CSS files; in this case, it is a stylesheet relationship. In the second attribute, you give the address (filename) of the CSS file.

Once you make this link tag, you can change the style of HTML tags in the HTML document. No CSS is needed in the HTML document at all.

Important: you should understand that the <link> tag is different from the <a> tag for making links in text. Both involve the word "link," and so it might be confusing. However, the <link> tag is used to link the whole page to external documents, and the <a> tag is used to create text hyperlinks which visitors can click on to travel to other HTML pages.

TRY IT

First, create a complete HTML document with a header with h1 and h2 tags, and a main area with at least a few p tags. Add the <link> tag to the <head> with the correct attributes. Make sure that the name of the actual stylesheet is the same as the name is the "href."

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  <title>
5      Title
6  </title>
7  <meta charset="UTF-8">
8  <meta name="author" content="yourname">
9  <link rel="stylesheet" href="styles.css">
10 </head>

```

Open the page in a browser; it should be boring.

Then create a stylesheet with the same filename as given in the "href" of the <link> tag, and add CSS. Remember, **select the language for the file before typing the CSS**, or else your code editor will not give assistance with things like automatic code completion.

```

1  body {
2      margin: 20px;
3      background-color: lightgreen;
4      font-family: 'Garamond', serif;
5  }
6  header {
7      font-family: 'Arial', sans-serif;
8  }
9  h1 {
10     margin: 20px 0px 0px;
11     font-size: 34pt;
12     text-align: center;
13 }
14 main {
15     min-height: 400px;
16     padding: 20px;
17     background-color: lightblue;
18 }
19 p {
20     margin: 10px;
21     font-size: 13pt;
22     line-height: 1.3em;
23 }

```

Once you have saved the file, go back to the browser and reload the HTML page in the browser. If you have done everything correctly, then you should see a big change in the page's appearance.

Then try changing the values again, and check to see the result every time you make a change.

WHICH CSS SHOULD WE USE?

We just learned that you can add CSS to an HTML file in three ways: Inline, Embedded, and Stylesheets.

It is possible to use any combination of these methods; you could use all three at the same time, for example.

However, **CSS Stylesheets** are the best way to use CSS. Professional web sites use only stylesheet CSS except in very specific situations where a special fix is needed.

Why are stylesheets the best way? The answer is that stylesheets are concise, convenient, consistent, organized, and powerful. Specifically, stylesheets (1) save space by not repeating the same styles on multiple pages; (2) it allows you to make all changes in one location so there the exact same style can be applied to an entire site; and (3) that a single stylesheet makes it possible to change dozens or hundreds of pages in the exact same way with just one edit to one document.

If you have 100 pages in a site, one stylesheet can set the styles for all of the pages. If you used embedded CSS, for example, then you would have to copy the CSS a hundred times! You save a lot of space and time with stylesheets, and you avoid errors.

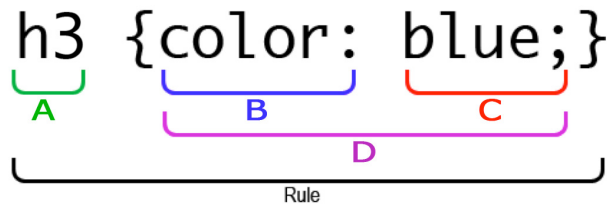
In addition, web sites should be consistent—the basic styling should be the same on all pages. With one stylesheet, you can be sure that the styles are consistent between all pages.

In this class, I will show you all three styles in the lecture, and I will sometimes use embedded CSS in class practices because it can be easier to use when you want to test something specific.

However, in your projects, I want you to use Stylesheets. Only use Embedded CSS with permission, and avoid using Inline CSS altogether.

CSS SYNTAX

Again, here is the basic form of a **rule**:



Here is an examples of such a rule:

```
main {
  margin: 10px;
  font-size: 13pt;
  line-height: 1.3em;
}
```

A CSS rule begins with the **selector**. After the selector, insert a space. This may not be necessary for the code to work, but it is a standard style for code.

After you type the selector and a space, type the left curly brace (a shift character, next to the "p" on your keyboard). Your code editor will automatically add the other brace.

As noted before, each **declaration** is on its own line. A declaration has a **property** and a **value**.

Notice that the **property** is followed by a **colon**, and the **value** is followed by a **semicolon**. You must use this punctuation correctly in every declaration. If you don't, it could easily break your site.

Do **not** put a space before the colon or semicolon. This could break your code.

```
h1 {
  margin: 0px;
  font-size: 30pt;
  font-family: 'Raleway', sans-serif;
}
```

Note the default styles listed on page 100; most tags have several default declarations. You should not make declarations that repeat those, but you can make some that change them.

Here are some basic CSS properties you can use; more will be introduced later in this chapter:

color:	Changes the text color
background-color:	Changes the background, highlight, or fill color
font-size:	Changes the font size
text-align:	Changes alignment, esp. to center
line-height:	Changes the line spacing; use the "em" unit
width:	Changes the width of a block element or an image
height:	Changes the height of a block element or an image
margin:	Creates a space around an element or image.

5f. Background Images

Background images can really make your site come alive in a way that solid colors cannot.

TECHNICAL POINTS

A background image can be applied to any block element. Usually, they are applied to the body, but can be applied to structural and normal block elements as well.

For example, let's say that you want to show an image with some text over it. Instead of using the `` tag, you could create a `<div>` with a normal-sized image as the background, and then put text into the div. Background images can also be used in `<header>` or even `<h1>` tags, or even to place small images within any block tag.

Normally, background images are for the whole page, and are applied to the `<body>` tag.

When you use a background image, the default is for the image to appear as its native size (simply do not use any width or height attributes or styles); **the background image will repeat**, starting at the top left corner; it will repeat from left to right and from top to bottom. The background image will be "behind" everything and won't directly affect content.

TYPES

There are two types of background images:

- Tiled (repeating) images
- Single images

Tiled background images are often small in size (resolution and file size). They are a single image, but when they repeat they create the appearance of a much larger image. They must be carefully created so that the top and bottom match exactly, as well as the left and right sides. This creates a **seamless** tiled background image. This means that a viewer cannot easily tell where the actual edge of the image is if it is repeated.



Above: the actual image (left), will appear as a much larger image when repeated (right).

Normal images look terrible when they are used as a background:



You should never do something like what I show above; if a background image is tiled, it should be seamlessly tiled. A repeated image like above looks horrible; use a single image.

Single, large background images must be large if they fill the entire body background. You can break the "400 KB rule" for this image (because it is only one image per page), but try not to go over 800 KB. A single image for a full-page background might be 1600 x 1000 pixels in resolution at most. You can reduce file size in Photoshop: save for web, and choose a quality somewhere around 40% ~ 50%. The background image does not need to be as high quality as most images. However, the quality should be *decent*; some people use extremely low-quality images, or they take a tiny image and blow it up, also creating poor quality.

CLASS RULE: *background images should not exceed 1600 px width or 800 KB.*

Selecting the right background image is important. Remember, the page content will be taking up most of the screen. In a Jello layout, the center of the background image will be blocked; in a Fixed layout, the left side will be blocked. You must choose a background image which does not have anything important in these places.



In the example above, see how the dog is blocked out, and the background loses its appeal. Later in this chapter you will learn the CSS to make an image always cover the whole page.

Next, when you choose a background image, it shouldn't be too different from the site's color scheme, as it will be a large part of the site's display. It should also not be too distracting, as the content of the page is more important. It should be decorative, not commanding.

Finally, remember that the background does not *have* to be an image. You can use a solid color if you like.

STYLE CONSIDERATIONS

When you choose background images, you should always consider the following:

- Does the background go well with my site style and color scheme?
- Is the background distracting?
- Can I read text clearly over this background?
- Is this background image really necessary?

You might be surprised at how often beginners ignore that last point! Indeed, many people ignore *all* of these points.

BACKGROUND-IMAGE

The code is simple—just use one declaration in CSS:

```
background-image: url(myimage.jpg);
```

BACKGROUND-REPEAT

Normally, a background image tiles (repeats) to fill the page. It repeats left-to-right, and top-to-bottom. You can limit this repeating in three ways:

```
background-repeat: repeat-x;           repeats left and right only
background-repeat: repeat-y;           repeats up and down only
background-repeat: no-repeat;          does not repeat
```

The repeat-x option is when you have a background that you want to repeat along the top of a page, such as an image of a rooftop pattern, or sky with clouds.

The repeat-y option is if you have a background image which you want to repeat along the left side, such as spiral binding, or a colorful or patterned margin.

If you use no-repeat, then the image will simply appear as it is, and will never repeat.

Where the background does not appear, background-color will fill in. Both can exist at the same time. The background image will always show above the background color.

BACKGROUND-SIZE

You can control how big background images can appear. You can make them display larger or smaller than their actual size. You can do this with tiled images or single images.

```
background-size: 200px 150px;          sets image to 200px wide, 150px tall
background-size: 200px;                 sets image to 200px wide, height is auto
background-size: 50%;                    resizes image to 50% of native size
```

If no background size is given, the background image will appear at its native size.

FULL-SIZE SINGLE-IMAGE BACKGROUNDS

Before, you learned that it is possible to use a single image as the page background. However, there is a problem. Different people have different display sizes, and the browser window may be even smaller. How can you make the whole image fit, so it will not repeat on a large screen, or be mostly cut out on a small display? In other words, how do you make the background image adjust—grow and shrink depending on the area it covers?

There is a way to do that:

```
background-attachment: fixed;           prevents background from scrolling
background-size: cover;                resizes image to fit the whole page
```

Both of these must be used together in order for the effect to work.

These two declarations will not only make the background fit the window, the image will also be static—that is, the background will not scroll, even though the content does scroll. This is desirable, because you do not want the image to repeat.

Using the "cover" for background size is not perfect. If the window is too narrow or too short to show the whole image, a little bit will be cut off the bottom or the right side. However, it is the best possible solution.

CITING BACKGROUNDS

You must give a citation / attribution for each background image you use. For a normal image, you can use the `title=""` attribute to give credit, and make a link out of the image itself. However, this is not possible for a background image.

The best way to **cite background images** is in the **footer**. Close to the copyright information, add a line of text which reads, "*Background image by ...*" and make the name into a link to the source page for the image.

DESIGN

5g. CSS RGB Color

When you were a kid, you probably learned about colors in school. From using paints, you perhaps discovered that there were three "primary" colors—red, yellow, and blue—which, with black and white paint, could create any other color. For example, red and yellow mix to make orange.

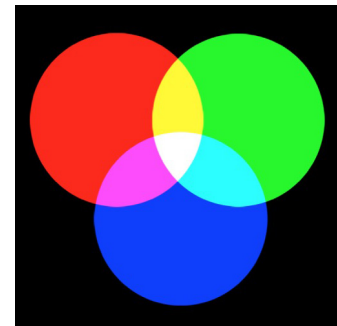
Colors used on web sites are based upon a different principle, however: the colors of light. In light, colors are additive—in other words, mixing two colors creates a brighter color, instead of a mix between the two.

Think of it like being on a dark stage with three spotlights: one red, one green, one blue. If you shine two spotlights on the same location, it looks brighter; shine all three, and it is brightest.

In addition, the primary colors of light are different from those of paint. For light, the primary colors are **red**, **green**, and **blue**. This is shortened as "RGB."

These three colors can be mixed to make any color that the human eye can see. To make different colors, one or all of the three basic colors can be brightened or darkened to create new colors.

As you can see in the illustration to the right, mixing all three colors gives you white (where they intersect in the middle). Otherwise:



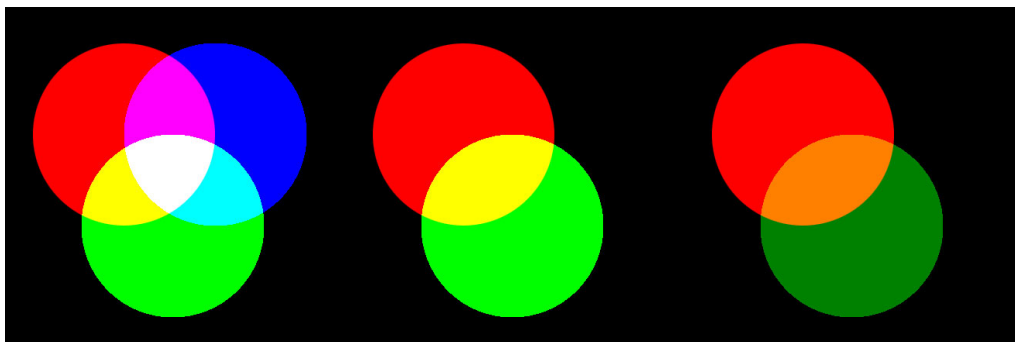
- Mixing red and blue makes a light purple color, called **magenta**
- Mixing blue and green makes a light blue color called **cyan**
- Mixing red and green together, oddly, makes **yellow**!

More colors can be made by varying the strength of any of the three colors.

Take the three images below. We start (image at left) with all three colors at 100%.

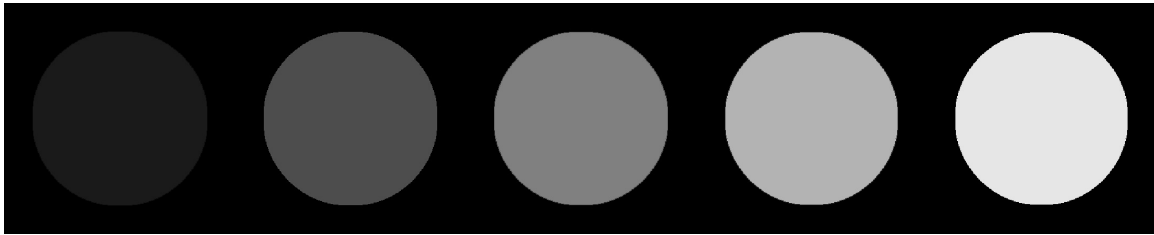
Then we turn off blue (image in center), and red & green meet to make yellow.

But then, we turn green down by 50% (image at right). and the mix turns to orange—because the red is stronger, and red + yellow = orange.



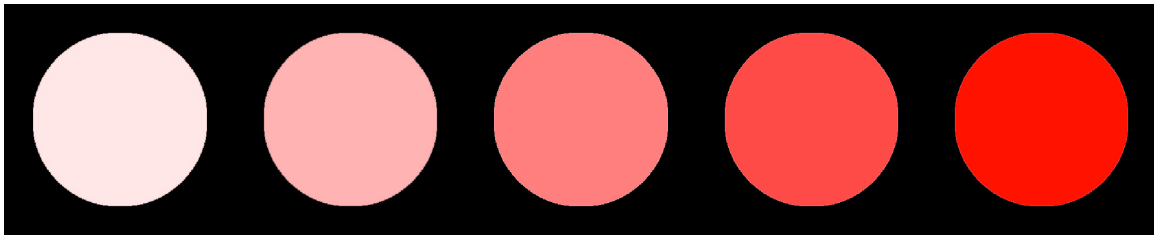
CHAPTER 5

Making all three colors the exact same percentage creates shades of black, gray, and white. 100% of all colors, makes white; 0% of all colors makes black. Below are the between:



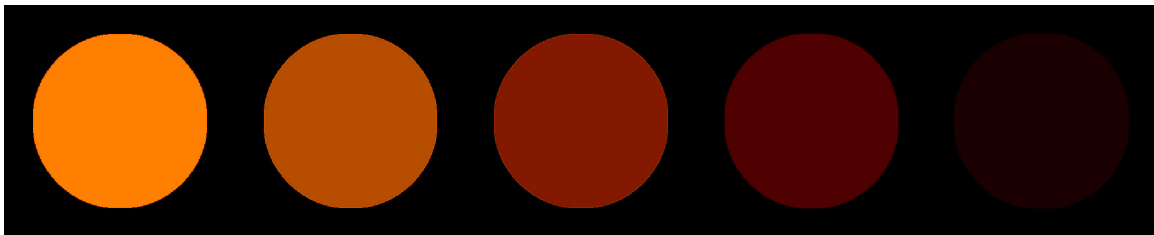
In order, by R-G-B percentage: 90-90-90, 70-70-70, 50-50-50, 30-30-30, and 10-10-10.

In contrast, to make a light or dark shade of one color, turn one color all the way up, and the other colors all the way up or down:



In order, by R-G-B percentage: 100-90-90, 100-70-70, 100-50-50, 100-30-30, and 100-10-10.

In the next example, we start from orange (100-50-0) to very dark red (10-0-0):



In order, by R-G-B percentage: 100-50-0, 70-30-0, 50-10-0, 30-0-0, and 10-0-0.

There are actually several ways to represent colors in CSS. The most simple is a scale from 0 to 255 for each of the three colors. In RGB color, each primary color (red, green blue) has 256 levels of brightness. "0" is the darkest, and "255" is the brightest.

In addition, the colors are always in the same order: Red, Green, Blue, or RGB.

Therefore, in RGB color code, "255, 0, 0" is bright red (red is brightest, green and blue are dark). Yellow would be "255, 255, 0"; orange would be "255, 128, 0"; and so on.

Because each color can be any of the 256 settings (from 0 to 255), the total number of combinations is 256 x 256 x 256, or a total of 16,777,216 colors! This is enough to represent images for people, because most humans can only see 10 million different colors (though some can reportedly see up to 100 million!). It is certainly enough colors for most purposes.

CHAPTER 5

Basic Principles of RGB color codes:

1. The three numbers are always in the order: **red , green , blue**
2. All numbers must be 0 or 255 or between; 127 is halfway
3. Higher numbers are brighter, lower numbers are darker
4. If all three numbers are the same, the color is white, black, or a shade of gray
5. To make one color stronger, raise its number or lower the others
6. To make **lighter** shades, raise all numbers, but the color you want should be *highest*
7. To make **darker** shades, lower all numbers, but the color you want should be *highest*
8. An alternate numbering can be in percent: For example, 60%, 80%, 100%

Example colors in 256³-color RGB:

	RED	GREEN	BLUE
Red	255	0	0
Green	0	255	0
Blue	0	0	255
Yellow	255	255	0
Cyan	0	255	255
Magenta	255	0	255
Black	0	0	0
Medium Gray	127	127	127
White	255	255	255
Orange	255	127	0
Medium Brown	150	100	0
Dark Red	75	0	0
Pink	255	200	200
Dark Green	0	75	0
Light Green	200	255	220
Dark Blue	0	0	75
Light Blue	200	220	255

To make these colors in CSS, use the `rgb()` value:

`color: rgb(255,0,0);` Changes text color (to red)
`color: rgb(100%,0%,0%);` Changes text color (to red)

`background-color: rgb(200,230,255);` Changes background color (to light blue)
`background-color: rgb(80%,90%,100%);` Changes background color (to light blue)

I prefer using numbers 0 ~ 255 instead of percentages, but that's a personal choice.

In CSS, you can use names of colors (black, red, aqua, etc.), but outside of a dozen or so very basic colors, the names are impossible to remember. Better to use RGB so you have have millions more colors!

You can practice rgb colors at lujweb.com/rgb/.

5h. Gradients

In CSS, you can plan gradients—the slow changing of one color to another. This is done using the "background" property, and the "linear-gradient" value. Add the "background-attachment: fixed;" declaration to help fill the whole page:

```
background: linear-gradient(white, gray);
background: linear-gradient(rgb(255,255,255), rgb(128,128,128));
background-attachment: fixed;
```

If that fails to cover the whole page, add this CSS rule:

```
html {
  height: 100%;
}
```

GRADIENT ANGLE

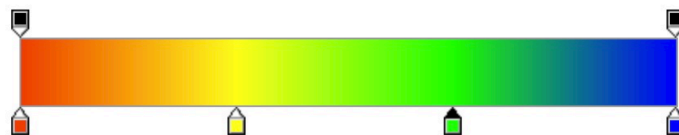
You can set the angle that the gradient runs as either a direction or a degree:

```
background: linear-gradient(to right, white, gray);
background: linear-gradient(to top, white, gray);
background: linear-gradient(to bottom right, white, gray);
background: linear-gradient(to top left, white, gray);

background: linear-gradient(45deg, white, gray);
background: linear-gradient(-45deg, white, gray);
background: linear-gradient(270deg, white, gray);
```

COLOR STOPS

The colors that you list are called "color stops." Here is an illustration of a gradient with four color stops:



Each stop, from left to right, has the **stop position** set as a percent. 0% starts at the far left, and 100% is at the far right. If you only have 2 color stops and no position information, then the stops are automatically set to 1% and 100%.

To set the color stops at specific places, add the percent value after each color stop:

```
background: linear-gradient(red 0%, yellow 33%, green 66%, blue 100%);
```

The above code will set the gradient to be what is seen in the illustration above.

To use rgb colors, be careful to use the parentheses correctly:

```
background: linear-gradient(
  rgb(255,0,0)    0%,
  rgb(255,255,0) 33%,
  rgb(0,255,0)   66%,
  rgb(0,0,255)   100% );
```

Experiment! Play around with angles, colors, and stop positions.

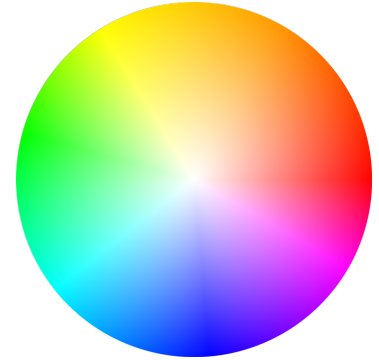
5i. Color and Web Sites

Colors are even more important than fonts when considering the design of a web site. Colors convey meaning and feeling. Colors can highlight the importance of a topic, they can relax visitors and make them open to ideas, and they can appeal to different groups of people in different situations. Colors can set the tone of your web site, and they can use associations people have with the colors to great effect.

THE COLOR WHEEL

The color wheel is based upon a circular layout of colors. This is useful because it also demonstrates the relationship of colors to other colors.

For example, colors (hues) have opposites: red and green, blue and orange, yellow and purple. If you stare at a blue shape for a minute and then look up at a white area, you will see an afterimage of the color in orange—the color's opposite.



Go ahead: stare at the very strange-looking flag below for a while, and then look at a white space somewhere (maybe blink a few times). Surprise! Same with the photo of the woman.



What you see is an *afterimage*, which appears because our eyes have become accustomed to seeing certain colors, and when those colors are removed, or eyes take a few moments to adjust—but in the meantime, we see the *lack* of that color, which is the opposite of it.

We can use the color wheel to help choose colors for our color schemes. Certain combinations of colors look good together.

For example, one way to choose color is by deciding on a single hue (for example, blue), and using various *shades* of that color—light blue, medium, blue, dark blue, and other shades in between.

Opposites also sometimes work. For example, the colors of Christmas (green and red) are opposites. Blue and orange, popular colors for businesses, are also opposites.

CHAPTER 5

These are a few of the basic color patterns, which include:

1. **Monochromatic:** choose a single color, and use light and dark versions of that color (for example, light to dark blue).
2. **Analogous:** choose a color, then use colors based on that and colors very close on the color wheel (for example, red, orange, and yellow).
3. **Complementary:** use two opposite colors, such as red and green (Christmas colors), or blue and orange (Amazon.com)
4. **Split complementary:** use two opposite colors, and the colors next to one of those. For example, blue & orange (opposites) along with red and yellow (close to orange).
5. **Triadic:** use three colors equally distant from each other, such as red, green, and blue (this site!)
6. **Tetradic:** use four colors, two pairs of opposites (e.g., red & green, blue & orange). Google and Microsoft use such schemes.

Two very good sites for exploring these colors schemes are Adobe's Color CC page (color.adobe.com), or Paletton's Color Scheme Designer (paletton.com).

COLOR SCHEMES

Color will be an important part of any web site you create. In order to create a good web site, you must understand how colors are used.

Every web site has a **color scheme**. These colors are carefully considered and chosen for specific reasons. The scheme is a specific set of colors which is used to convey a certain theme, feeling, or concept to a viewer.

We have strong *associations* with colors. Seasons, for example, have their colors; you might match orange with autumn, white with winter, green with spring, and yellow with summer. Green, brown, and blue are associated with nature; blue with the sea and sky, green with life and the environment in general. Colors can be associated with gender or sexuality.

Colors can have associations with feelings. Red is a color of anger, stress, and passion; green with relaxation and health; blue can signal trust and peace.

Colors can indicate properties. Red is danger, yellow (amber) is caution, green is safe. Purple or blue can be royalty. Red is war.

Some colors are thought to have psychological effects on people. Blue is a common favorite color, and expresses comfort or trust. Orange is considered aggressive, active, and optimistic, and is often used in business. Green can be relaxing, red can be sexy, purple can be luxurious.

Colors are commonly associated with temperatures; red is the hottest, blue is the coolest.

Colors can be associated with events; Christmas is red and green, Halloween orange and black. Black is usually associated with funerals, white with weddings.

Colors can be institutional; most institutions choose a set of colors to represent them. Countries have their national flag color schemes, while individual regions and cities have their own as well. Universities most often have two colors as their own, though blue and gold are a common pairing. Corporations and other organizations have their own colors.

Some meanings of colors can be culture-specific. In American culture, for example, yellow is cowardice, green is envy, and blue is sadness. In Japan, the sun is red and the moon is yellow, but in America, the colors are gold and silver. Green is associated with money in America, for obvious reasons.

Any organization will take a great amount of time, thought, and effort to decide which colors represent themselves, and will have specific reasons why they chose those colors. It is considered a very important decision.

CHOOSING A COLOR SCHEME

So, when you choose which colors to use on your web site, you will want to think about it carefully. Do not just use any and every color whenever you feel like it. Choose your colors based upon your site's associations, meaning, theme, purpose, or feeling.

Once you have chosen a color scheme, stick to it. Make sure the colors run through most or all of the pages on your entire web site. Perhaps even select your photos based on your color scheme.

Here are some of the common ways to choose your color scheme:

1. Colors may be chosen because they are **the colors already chosen for an organization**. For example, Lakeland College uses the colors dark blue and gold; thus, you will see these colors as the main ones used in its web sites and logos. Countries have colors; in America, it is red, white, and blue; in Japan, it is red and white. In Spain, red and yellow; and so on. These colors are often used to create a patriotic feeling.
2. You can also choose colors **based upon an organization's purpose**. For example, an organization which deals with nature may use browns, greens, and blues (soil, plants, and water & sky). A soccer organization may choose grass green, white, and black (a playing field and the colors of a soccer ball).
3. Another way to choose a color scheme is **based upon an image**. If you feature one image as representing your organization, the colors in that image may give you your color scheme set. For example, if you have a Scuba club, and you use an image of someone at a resort beach, the colors of the water and features of the sea floor may become your color scheme.
4. You can also **use colors based upon their accepted meanings**. For example, red is considered important, energetic—or alarming—but is also associated with both love and war; blue is a favorite color, considered peaceful and trusting; green is the color of nature and growth, and can be soothing; and so on. You might select your colors based upon a variety of interpretations.
5. One more way is to use colors based upon the **color wheel**. In this technique, you choose a main color, and then choose others based on that color.

So, when you create your site, **consider all of these influences**, and **choose your color scheme wisely**. Make sure it is not just cool or attractive, make sure that it is relevant, make sure it has appeal to your audience, and make sure that it is consistent with your theme and used consistently throughout your web site.

Exercise 5-1

Make a web site with four pages. The pages must have the exact same styling.

There should be no CSS on any of the HTML pages. Instead, make a separate CSS stylesheet, and have all the CSS rules in the stylesheet. Add the CSS to the pages using a <link> tag.

The pages must be named:

- index.html
- style.html
- styled.html
- styling.html

Every page should have four links, one to each page. Make sure that every page has a link in the <h1> tag which leads to the index file, and then a and list in the <nav> with links to the three other HTML pages. The header and nav must be exactly the same on all four pages. **Do NOT copy any previous homework, start from blank pages!**

Create a CSS stylesheet with many rules, and try to use as many of the CSS properties and values as you can. Many are listed on the following pages.

Save the pages in a folder named **yourname-5-1**, and email it to me as an attachment by the due date that I give in class.

Chapter 5 Code

There is one new tag for this unit:

Beginning	Purpose
<code><link rel="stylesheet" href="styles.css"></code>	Begins and ends the head

This is where it should go in the code:

```

3  <head>
4      <title>
5          Title
6      </title>
7      <meta charset="UTF-8">
8      <meta name="author" content="yourname">
9      <link rel="stylesheet" href="styles.css">
10 </head>

```

Here is the CSS you can use in this unit:

Beginning	Purpose
<code>color: darkblue;</code>	Sets the text color using color names
<code>color: rgb(40,40,50);</code>	Sets the text color using RGB values
<code>background-color: lightblue;</code>	Sets the background color using color names
<code>background-color: rgb(150,200,255);</code>	Sets the background color using RGB values
<code>height: 300px;</code>	Sets the height of an object
<code>width: 400px;</code>	Sets the width of an object
<code>margin: 10px;</code>	Sets a margin space outside the border
<code>min-height: 400px;</code>	Sets a minimum height to an object
<code>font-family: 'Arial', sans-serif;</code>	Sets a font family to text
<code>font-family: 'Garamond', serif;</code>	
<code>font-size: 12pt;</code>	Sets a size to the font (can be decimal)
<code>font-style: italic;</code>	Sets a style to the font (italic or oblique)
<code>font-weight: bold;</code>	Sets a weight (boldness) to the font
<code>font-weight: 700;</code>	(100s values used with some Google fonts)
<code>font-variant: small-caps;</code>	Sets a SMALL CAPS style
<code>text-align: center;</code>	Sets the alignment of text (center, right)
<code>text-decoration: none;</code>	Sets an underline or removes one
<code>text-indent: 0.5in;</code>	Sets a first-line indent to a paragraph
<code>text-shadow: 1px 1px 2px rgb(0,0,0);</code>	Sets a text shadow (rgb colors are better)
<code>line-height: 1.2em;</code>	Sets line spacing
<code>background-image: url(myimage.jpg);</code>	Sets the background image
<code>background-attachment: fixed;</code>	Stops the background image from scrolling
<code>background-size: cover;</code>	Sets the image to fill the browser window
<code>background-repeat: repeat-x;</code>	Makes the background image repeat left-right
<code>background-repeat: repeat-y;</code>	Makes the background image repeat up-down
<code>background-repeat: no-repeat;</code>	Stops the background image from repeating
<code>background-size: 200px 150px;</code>	Sets the size of the background image
<code>background-size: 50%;</code>	

CHAPTER 5

Used to make a web page, the HTML tags may look like this:

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <title>
5       Title
6     </title>
7     <meta charset="UTF-8">
8     <meta name="author" content="yourname">
9     <link rel="stylesheet" href="styles.css">
10  </head>
11  <body>
12    <div id="wrapper">
13      <header>
14        <h1>
15          Title
16        </h1>
17        <h2>
18          Subtitle
19        </h2>
20      </header>
21      <main>
22        <h3>
23          This is the Page Title
24        </h3>
25        <figure>
26          <a href="https://flic.kr/p/6b4vvR" target="_blank">
27            
28          </a>
29          <figcaption>
30            Shiba Inu puppies are the best!
31          </figcaption>
32        </figure>
33        <p>
34          Lorem ipsum dolor sit amet, pri debet mucius erroribus in, nec alterum scripserit ea, at sit
35          eruditi labores. His fugit semper admodum id, at est clita reprimique. Ne antiopam
36          definiebas mei. Mediocrempertinacia ut eam, ea illud homero principes sit. In sit viderer
37          delenit, in aperiam repudiare nam, habeo labitur eu vis. Illud dicant ex mel, inani legimus
38          ea ius.
39        </p>
40        <p>
41          Utamur oblique dissentiet cu vim, his an duis possit accusata. Quem labores ad qui, aliquam
42          mentitum sit ei. No ius solum scaevola voluptaria, sed liber dicit ea. Per idque
43          theophrastus an, etiam democritum cu sea. Oblique ancillae accusata sit ea. Numquam ponderum
44          invidunt cu qui, eam homero accusam quaerendum eu. Alterum antiopam id nam, in usu vivendum
45          postulant.
46        </p>
47        <p>
48          Cibo pericula molestiae quo an. Ex sea dico delenit atomorum, ei vocibus patrioque mel. Invidunt
49          accommodare eam ad. An vim agam conclusionemque, intellegat temporibus pro in. Sea aequae
50          assentior voluptatum an, cu pertinax comprehensam per, quo error quando expetenda cu. Et duo
51          aperiam legendos, ea nullam malorum eripuit vix, diceret gloriatur vis ad.
52        </p>
53      </main>
54      <footer>
55        <p>
56          &copy; 2019 Yourname
57        </p>
58      </footer>
59    </div>
60  </body>
61 </html>
```

CHAPTER 5

Used to make a stylesheet, the CSS rules may look like this:

```
1 body {
2   margin: 0px;
3   background-color: rgb(150,200,255);
4   background-image: url(images/fuji.jpg);
5   background-attachment: fixed;
6   background-size: cover;
7   font-family: 'Raleway', sans-serif;
8   color: rgb(50,50,50);
9 }
10 header {
11   width: 800px;
12   height: 150px;
13   padding: 1px 50px;
14   background-color: rgb(250,230,190);
15   background-image: url(images/ponta.jpg);
16   background-repeat: no-repeat;
17   background-size: 800px 150px;
18 }
19 h1 {
20   margin: 20px;
21   font-family: 'Abril Fatface', cursive;
22   font-size: 34pt;
23   text-align: center;
24   text-shadow: 1px 1px 2px rgb(0,0,0);
25 }
26 h2 {
27   margin: 10px;
28   font-family: 'Exo', sans-serif;
29 }
30 main {
31   min-height: 400px;
32 }
33 p {
34   font-family: 'Raleway', sans-serif;
35   font-size: 12.5pt;
36   line-height: 1.3em;
37   text-indent: 0.3in;
38 }
39 a {
40   text-decoration: none;
41   color: rgb(200,0,0);
42   font-weight: 500;
43 }
44 figure {
45   width: 300px;
46 }
47 figcaption {
48   font-style: italic;
49 }
50 footer {
51   clear: both;
52   border-top: 1px solid rgb(210,150,50);
53   font-variant: small-caps;
54 }
```

Chapter 5 Checklist

- What are three examples of languages that can be inserted into HTML?
- What are *deprecated* tags? Should you use them?
- What is the key or "signal" word which indicates that CSS will be used?
- What is a CSS *property*?
- What is a CSS *value*?
- What is a CSS *declaration*? (It is different from an HTML declaration!)
- What is the syntax (order, punctuation, and spacing) for a CSS declaration?
- What happens if you leave out a semicolon?
- What are the three ways to insert CSS into HTML code?
- Where is Inline CSS added to HTML code?
- Where is Embedded CSS added to HTML code?
- What tag is required to connect a web page to a CSS stylesheet?
- Why is Embedded CSS better than Inline CSS?
- Why are stylesheets better than Inline or Embedded CSS?
- What are the names of all the parts of a rule?
- Without looking, how many **unit** types are used in CSS? (Try to name at least 5)
- What are the three primary colors of light?
- How can you change the colors to create new colors?
- What do you get when you combine green and blue? What about red and green?
- How many different colors can be represented on a display?
- Can you figure out (not look up) the RGB codes for orange, medium gray, and pink?
- What are the CSS rules and properties necessary to make a gradient background?
- What are color stops, and how do they work?
- How do you change the direction of a gradient?

Chapter 6: CSS Selectors

SETTING UP

Here is the CSS introduced in this unit:

Beginning	Purpose
<code>color: darkblue;</code>	Sets the text color using color names
<code>color: rgb(40,40,50);</code>	Sets the text color using RGB values
<code>background-color: lightblue;</code>	Sets the background color using color names
<code>background-color: rgb(150,200,255);</code>	Sets the background color using RGB values
<code>height: 300px;</code>	Sets the height of an object
<code>min-height: 400px;</code>	Sets a minimum height to an object
<code>max-height: 400px;</code>	Sets a maximum height to an object
<code>width: 400px;</code>	Sets the width of an object
<code>padding: 6px;</code>	Sets spacing between content and border
<code>padding-top: 2px;</code>	sets spacing at top only
<code>padding-right: 10px;</code>	sets spacing at right only
<code>padding-bottom: 4px;</code>	sets spacing at bottom only
<code>padding-left: 0px;</code>	sets spacing at left only
<code>border: 1px solid gray;</code>	Sets 4-sided border for shape
<code>border-width: 1px;</code>	Sets border width
<code>border-style: solid;</code>	Sets border style; also dashed, etc.
<code>border-color: rgb(50,50,50);</code>	Sets border color
<code>margin: 10px;</code>	Sets spacing outside content
<code>margin-top: 20px;</code>	sets top margin only
<code>margin-right: 10px;</code>	sets right margin only
<code>margin-bottom: 5px;</code>	sets bottom margin only
<code>margin-left: 0px;</code>	sets left margin only
<code>box-shadow: 6px 6px 12px black;</code>	Sets a drop shadow for a rectangular object
<code>border-radius: 6px;</code>	Rounds the corners of a rectangular object
<code>font-family: 'Arial', sans-serif;</code>	Sets a font family to text
<code>font-family: 'Garamond', serif;</code>	
<code>font-size: 12pt;</code>	Sets a size to the font (can be decimal)
<code>font-style: italic;</code>	Sets a style to the font (italic or oblique)
<code>font-weight: bold;</code>	Sets a weight (boldness) to the font
<code>font-weight: 700;</code>	(100s values used with some Google fonts)
<code>font-variant: small-caps;</code>	Sets a SMALL CAPS style
<code>text-align: center;</code>	Sets the alignment of text (center, right)
<code>text-decoration: none;</code>	Sets an underline or removes one
<code>text-indent: 0.5in;</code>	Sets a first-line indent to a paragraph
<code>text-shadow: 1px 1px 2px rgb(0,0,0);</code>	Sets a text shadow (rgb colors are better)
<code>line-height: 1.2em;</code>	Sets line spacing
<code>background-image: url(myimage.jpg);</code>	Sets the background image
<code>background-attachment: fixed;</code>	Stops the background image from scrolling
<code>background-size: cover;</code>	Sets the image to fill the browser window
<code>background-repeat: repeat-x;</code>	Makes the background image repeat left-right
<code>background-repeat: repeat-y;</code>	Makes the background image repeat up-down
<code>background-repeat: no-repeat;</code>	Stops the background image from repeating
<code>background-size: 200px 150px;</code>	Sets the size of the background image
<code>background-size: 50%;</code>	

CODE

6a. Selectors: Class and Id

In Chapter 5, we learned how to apply CSS rules to **tags**.

Inline CSS could be applied to individual tags, but requires far too much repetition and work, and leads to too many problems.

Embedded and Stylesheet CSS is far better, but using an HTML tag as the selector means that only one style can be applied to all instances of that tag. That can be a problem: what do you do if you want the same tag to be styled differently in different places?

For example, when you make nav menu buttons, the the `` and `` tags are used. These tags are normally for making bullet lists. In a nav menu, they are styles to look like buttons in a row. However, in the main text, they need to look like simple bullet lists. How can you style the nav menus and bullet lists to look different?

A more common example of this is using the `<div>` tag. The `<div>` tag is commonly used for a variety of purposes; it might be used for several different purposes on a single page, and most times, each `<div>` tag requires its own unique style. How can you style the CSS for these tags?

The answer is: use **classes** and **ids**.

The type of CSS rules we learned in Chapter 5 is definitely helpful, and is very often used. However, sometimes we want to create:

- styles which are *sometimes* used on a tag, but not always
- styles which are used on *various* tags at various times, but not the same tags always
- styles which might be used only once on the page

In such cases, you would need styles which are not directly associated with specific HTML tags. Remember, the **selector** decides which tag gets the style changes. Before now, we only learned how to use HTML tags as the selector. For example:

```
p {
  margin: 10px;
  font-size: 13pt;
  line-height: 1.3em;
}
```

In the rule above, *all* `<p>` tags will have that exact same style. So, how do we make it so only *some* `<p>` tags are styled like that, but not all of them?

The answer is to use a class or an id, depending on the case.

A **class** is a rule which can be applied to any tag any number of times per page.

An **id** is a rule which can only be applied once per page. You could have several different ids on one page, but each id could only be used once on that one page.

Other than that, class and id work almost exactly the same. Their difference is **semantic**.

CLASS

A class in CSS has a different selector—a **class selector begins with a period**:

```
.ponta {
  color: rgb(255,0,0);
  font-weight: bold;
}
```

Notice the **selector**. It is called **.ponta** instead of having an HTML tag name. A class must always start with a period, but the name can be almost anything you want. The name must not contain spaces, and it must not be the name of an existing HTML tag or CSS property. (For example, don't name the class **.p** or **.color**, as these are already used in HTML and CSS.)

After you have created the class, it will not affect anything until you **assign the class** to specific tags in the HTML. To do that, add the `class=""` attribute:

```
<p class="ponta">
```

Notice that the class value in HTML is the class name *without the period*.

Both class and id can be applied to different tags—for example, to `<p>`, `<i>`, ``, or `<blockquote>` tags.

The difference is that *a class can also be applied to as many tags as you want on a page*. You can apply it to 20 of 25 `<p>` tags, or you can apply it to three different tags in the page.

This option adds a lot of flexibility to your ability to add styles within your page.

MULTIPLE CLASSES

By the way, there is a neat trick with classes: **you can use more than one in one tag**. You can have multiple class names within the attribute for one tag. For example:

```
<p class="ponta doggie">
```

In the above case, the tag would use styles from the "ponta" class *and* the "doggie" class. So if the "ponta" class sets the color and boldness, and the "doggie" tag sets the font size and family, using both class names separated by a space will apply both styles to the same tag.

Technically, there is no limit to the number of classes you can join together, but if you are using more than two or three, you are probably arranging the classes poorly.

The same multiple-class trick can *not* be done with an id; in id attribute can only have one id assigned to it. Never more than one.

Id

An in CSS also has a different selector—an **id selector begins with a hashtag (#)**:

```
#ponta {
  color: rgb(255,0,0);
  font-weight: bold;
}
```

When used in HTML code, it uses the `id=""` attribute:

```
<p id="ponta">
```

Like the class, it can be applied to any tag—but the id is different, in that **one id rule should be used only once in a whole page**. Otherwise, it acts in exactly the same way as a class.

Why the difference? Some ids should never be used more than once—for example, if the ID places an object on a specific location on the page; you don't want two objects in the exact same location.

USING ID TO LINK / JUMP TO A PART OF A PAGE

The id can be used for other purposes: the most common alternate use is to identify a specific location on a web page.

You may have seen pages which are very long and have a number of sections, each with its own header. There may be an index with links which leads you to that part of the page. The most common example of this is in a Wikipedia page.

You may also have seen pages with links to the "top" of the page. These links lead to the start of the page after you have scrolled down a long ways.

These are links which do not lead to another page, but to another part of the same page.

To do this, you must:

- mark the location you want to jump to by adding an id to a tag


```
<h4 id="conclusion">
```
- use an `` tag with an address for the same page plus the #id name, or even just the id name:

```
<a href="index.html#conclusion">
<a href="#conclusion">
```

To make a "Top" link, just add an id to the `<body>` tag and link to that.

Other times an id might be used would be to identify a specific input in a form, allowing an outside app, such as a Javascript program, to identify that tag or location. For now, however, we will be using the id for styling.

CLASS VS. ID: SEMANTIC USE

As we just established, a **class** can be used as many times as you want on one page, but an **id** can only be used once per page.

Here's the thing that confuses most people: you can use an id rule five times on a page, and it will still work.

The difference between class and id is not in how they work; it is purely *semantic*.

For example, the tags <header>, <main>, <article>, <aside>, and <div> will all appear *exactly* the same way on a web page. You could use the <main> tag instead of the header and a <div> tag instead of the <main> tag. It would work. They would all look exactly the same.

However, we don't do that. We created all of those tags for specific purposes. Using the <main> tag to contain the <header> would work, but it would confuse people. It would be like putting a sign on the door to your bedroom which says "kitchen." You could call your kitchen a "bedroom," and the room would still *work*—if you put a bed in it, you can sleep there—but it would be confusing for anyone else who came to your house!

Often the distinction is very important. Remember the example I gave from Chapter 3, in which you might have two jars on your table at home, a *sugar* jar and a *salt* jar, each with the name printed on the side. Of course, you could put sugar in the salt jar and salt in the sugar jar. Both are containers, both would "work." However, visitors would be very unhappy with the taste of their coffee!



When someone looks at your code and they see a **class**, they will accept that it can be used many times on the page. In fact, they will *expect* to see it used many times on a page. They will certainly expect that the rule is written so that it *can* be used many times on a page.

However, if they look at your code and see an **id**, they expect that you are making a rule: *don't use this more than once on this page!* They expect that there is a *reason* why it should not be used more than once on a page. Usually that reason is that an id can only be applied to one tag or else the page breaks, or perhaps using it more than once would be strange.

For example, take the **wrapper**. It is always an id. Why? Because it is designed to hold the header, nav, main, and footer structures, to be the container for all content on the page. You would never need it for anything else on the page. (If you have some strange setup where you need two wrappers with the same styling, you would simply make it into a class, not an id.)

In Unit 3, we will learn how to *exactly* set the position of an object on a page—for example, 50 pixels from the top and 100 pixels from the right. It would look bad if two objects tried to be in that space. It *should* be used only once—so an id would be used to define the rule.

6b. The Wrapper Div

Before we go on, let's establish something that I have hinted at before: the Wrapper.

Actually, let's go a bit further back... into HTML4. In HTML4, there were no structural tags aside from `<div>`. There were no special tags for the header, nav, main, footer, or any other structural tags. Instead, web designers used the `<div>` tag. *A lot*.

They would use it like this:

```
<div id="header">
</div>
<div id="nav">
</div>
<div id="main">
</div>
<div id="footer">
</div>
```

All the structural tags were just `<div>` tags, and each structural part was created with an id. This worked, but there were a lot of problems with it.

One problem came at the end of the code, when you would often see something like this:

```

                </div>
            </div>
        </div>
    </div>
```

This is called "div soup," as it is confusing—you cannot easily tell which end tag is ending which div, and it is easy to mistakenly have too few or too many end tags.

To help solve this problem and make web page structuring easier, the structural tags were created and implemented in HTML5, which is what we are learning now.

However, one structural part was overlooked: the **wrapper**.

The wrapper is a tag that surrounds and contains all the structural elements, such as header, nav, main, and footer.

Why do we need a wrapper? Well, several reasons. A big one is so that you can hold everything together. Imagine trying to carry ten boxes stacked on top of each other. They would always be sliding out and falling apart. However, wrap them in plastic, and now they stay together, and it's much easier to carry them.

Web pages are the same way. In the first chapter, we learned the layouts: liquid, fixed, and jello. These layouts require all the structural elements to act in the same way. While we *could* just style each element to act in that same way, it would require a lot of extra code. It is much easier to just wrap them together.

Another important reason is that you will want to apply one style which will affect the structural tags as one big unit. For example, if you want a border to outline the whole page content, it would be hard to style each section so that only the outside edges (and not the inside edges) would have a visible border. It is much easier to just apply one plain border to an enveloping wrapper.

CHAPTER 6

However, no wrapper tag was included in HTML5. As a result, we still have to use the `div` tag with an `id` for that purpose (HTML6, which may not begin widespread use for at least a few more years, will allow you to create your own custom tags, including a wrapper tag.)

In fact, many advanced web designers use a variety of wrappers and containers. We won't be getting into that complexity in this course, however.

I will require you to use the wrapper `div` in all of your documents from this time forward. That usage will look like this:

```
<body>
  <div id="wrapper">
    <header>
      page titles are here
    </header>
    <nav>
      links to other pages in site are here
    </nav>
    <main>
      main content is here
    </main>
    <footer>
      footer content is here
    </footer>
  </div>
</body>
```

What's more, using the wrapper allows you to easily set any one of the three general layouts: liquid, fixed, or jello.

If your wrapper has no formatting at all, then your page layout will be **liquid**. The `div` will serve to hold things together, and because it is a block tag, it will expand to 100% of the width of the browser area, and will expand or contract with the size of the window.

If your wrapper has a width setting:

```
#wrapper {
  width: 1000px;
}
```

Then it will create the **fixed** layout: it will naturally align to the left, and will be a set width.

Finally, if you have both width and `margin-left: auto;` settings:

```
#wrapper {
  width: 1000px;
  margin-left: auto;
  margin-right: auto;
}
```

Then it will create the **liquid** layout: it will have a set width but will be center-aligned.

Again, the reasons for using a wrapper:

1. Allows you to keep all the structural areas bound together
2. Allows you to apply page effects to all parts as a whole (borders, shadows, etc.)
3. Allows for layout formatting (liquid, fixed, jello)

6c. More CSS Selectors

There's one more thing to know about CSS rules. You can type the selectors in a variety of ways that change the way they work.

DESCENDANT SELECTORS

Let's say that you want paragraph `<p>` tags in the `<main>` section to be ordinary paragraphs, but you want the `<p>` tags in the `<footer>` to look differently. The `<p>` tags in the `<main>` section can be Garamond 13 pt. with a dark font color. However, the footer maybe has a dark color background, so you want the text there to be Arial 10pt with a light blue color.

How do you style that?

You could use classes, but it turns out there is an easier way: define the selectors by their area. You can have a selector which says, "All `<p>` tags in the `<main>` tag." It looks like:

```
main p { declarations go here }
```

That rule will only affect `<p>` tags which are children of the `<main>` tag. Similarly:

```
footer p { declarations go here }
```

That rule will only affect `<p>` tags which are children of the `<footer>` tag.

Alternately, you can just make a plain "p" rule for most of the page, and then create a special rule only for `<p>` tags in the footer:

```
p {
  font-family: 'Garamond', serif;
  font-size: 13pt;
  color: rgb(20,50,150);
  line-height: 1.2em;
}
footer p {
  font-family: 'Arial', sans-serif;
  font-size: 10pt;
  text-align: right;
  color: rgb(200,220,255);
}
```

The above two rules should get the desired effect.

Why not use `main p` for the first rule? Notice that the first "p" rule also includes a line-height. You want the line height to be in all `<p>` tags, including the footer. In you make the first rule `main p`, then the line height will only apply to the paragraphs in the main tag. By making the rule selector a plain "p" only, it will affect *all* the paragraphs, everywhere in the document.

In short, the general `p{ }` rule applies to *all* paragraph tags in the whole document, unless a different descendant rule, such as `footer p { }`, has different values for the same properties.

For example, in the above rules, I have two paragraph styles each for font, font size, and font color. Won't they clash? The answer is no: the more specific selector "wins." Even though I defined a font color for all `<p>` tags, I made a more specific font color style for `<p>` tags in the `<footer>`. That more specific style will "win," but only in the footer.

ADDITIVE SELECTORS

Let's say that you want to apply the exact same style to multiple tags. For example, you want the header, nav, main, and footer tags all to have 20px of padding and 10px margin. How can you do this?

You could simply add the two declarations to the rules for each tag, but that would be a total of 8 lines of code. You could do the same thing with less typing:

```
header, nav, main, footer {  
    margin: 10px;  
    padding: 20px;  
}
```

Now we only have three lines of code, plus the end bracket. Much better!

Notice that the selectors are separated by **commas**. This is necessary! Commas, in this case, mean "and." If you do not use commas, then it becomes a descendant selector—note that **the following example is an example of an error**:

```
header nav main footer { }
```

In this case, the rule applies to a footer which is inside a main tag, which is inside a nav tag, which is inside a header tag! Of course, that would never really happen. this rule would not change anything.

There are other types of selectors, but for now, this is enough.

6d. CSS Shorthand

You already know how to use a single value in CSS which creates a margin which is 10px on all four sides:

```
margin: 10px;
```

However, each of the four sides of a box can be styled independently in CSS.

4-NUMBER SHORTHAND

For example, let's say that you have an image which is floated to the right side of the web page. You need it to have a big left margin to separate it from the text to the left, and a smaller margin at the bottom, and no margin at the top or right. In CSS, this would be:

```
margin-top: 0px;
margin-right: 0px;
margin-bottom: 10px;
margin-left: 25px;
```



That code takes up a lot of space—four lines! Fortunately, there is a shortcut: use multiple values in a single declaration. Each value can represent a different side of the box:

```
margin: 0px 0px 10px 25px;
```

The order of the numbers is specific: starting at the top, moving clockwise. Perhaps an easier way of remembering this is with the word **trouble**, or **T-R-B-L**, for Top, Right, Bottom, and Left. The first number is for the top of the box, then the right, then the bottom, then the left.

The same can be used for padding:

```
padding: 0px 25px 10px 0px;
```

3-NUMBER SHORTHAND

Sometimes you have the exact same values for the left and the right, but different values for the top and the bottom. For example, this is a common margin setting for wrappers:

```
margin: 20px auto 50px auto;
```

Note that you can sometimes use word values (such as "auto") instead of numbers.

In order to center the wrapper, we need to use auto at the right and the left. When the right and the left are exactly the same, they can be shortened into only one value:

```
margin: 20px auto 50px;
```

In the above example, the auto value represents both the right and the left; this distorts the "trouble" a little bit: **T-RL-B**. The top and bottom are defined separately, but the left and the right are combined into one.

2-NUMBER SHORTHAND

There are other times when you want the top and bottom to be the same, and the left and right to be the same, but different from top and bottom.

```
margin: 20px auto;
```

In the above example, the top and bottom margins would be 20px, and the left and right would be auto (for centering). This would be **TB-RL**.

And, of course, as explained at the start, one value will represent all four sides.

CORNER SHORTHAND FOR BORDER-RADIUS

The border-radius property can make the corners of a box rounded. Just like with sides of a box, you can give each *corner* a different value. In fact, this can *only* be done with shorthand—there is no CSS property for each corner (for example, "border-radius-top-left" does not work).

```
border-radius: 0px 10px 80px 30px;
```

In the case of border-radius, the cycle begins at *top-left* and also goes around clockwise: top-left, top-right, bottom-right, bottom-left (**TL**, **TR**, **BR**, **BL**).

This can be used to great effect sometimes. See the second illustration to the right: four square images, 200px wide each, each one with a single border corner cut to a radius of 200px (all other corners are 0px).

Note: for another nice border-radius CSS trick, if all four corners of a square object (image or block tag) have a border-radius equal to the total width of the object, then the object will become a circle.

Here are some shortcuts to use for now:

20px;	TRBL	All sides are the same value
10px 20px;	TB RL	*Top & Bottom are 10px; Right & Left are 20px
10px 20px 15px;	T RL B	*Top is 10px; Right & Left are 20px; Bottom is 15px
10px 20px 15px 5px;	T R B L	*Top is 10px; Right is 20px; Bottom is 15px; Bottom is 5px

For border-radius:

20px;	All sides are the same value
10px 20px;	Top-Left & Bottom-Right are 10px; Top-Right & Bottom-Left are 20px
10px 20px 15px;	Top-Left is 10px; Top-Right & Bottom-Left are 20px; Bottom-Right is 15px
10px 20px 15px 5px;	Top-Left = 10px; Top-Right = 20px; Bottom-Right = 15px; Bottom-Left = 5px

For borders:

1px solid red;	border width - border style - border color
----------------	--

For shadows:

3px 3px 5px gray;	*left-right offset - top-bottom offset - blur value - color
-------------------	---

There are more, but these are the more common ones.



6e. Some Basic CSS Effects

Let's look at how CSS can be used, and some basic rules and parameters for the class.

DEFAULTS

An important point to remember are **defaults**. These are automatically used for the body tag:

```
body {
  color: black;
  background-color: white;
  font-family: 'Times New Roman', serif;
  font-size: 12pt;
}
```

If you make any style rules on a new page like the ones above, they will not change anything, because these styles are already set. Remember the pre-set default styles so you do not waste code space by repeating them. For example, all of the headings (h1, h2, h3, etc.) are already bold, so you should not try to make them bold. You might, in fact, want to make them *not* bold, if you are using a font that looks best in regular weight.

Here's what might be used as a basic paragraph rule set. Note that I am using the descendant selector `main p`, so paragraphs in the footer will not be included in this styling.

```
main p {
  font-family: 'Garamond', serif;
  font-size: 14pt;
  line-height: 1.3em;
  color: rgb(40,40,40);
}
```

Here I chose Garamond as my font. This is one of the **web-safe** fonts (explained later in this chapter) that should be available on almost all computers. It is good for paragraph text because it is a serif font; sans-serif is also acceptable (serif and sans serif are also explained later). Garamond is more elegant than Times New Roman.

However, Garamond is also a naturally small font; at 12pt in size, it looks quite a bit smaller than Times. To compensate, I changed the font size to 14pt.

CLASS RULE: *main paragraph font sizes shall not be more than 14pt.*

I set the "line height" (line spacing) to 1.4em. An "em" is the natural height of normal text. The default line height is 1.2em, or 20% more than standard text height.

CLASS RULE: *main paragraph line height should not be more than 1.3em.*

The reason for the above two rules is that people often want to fill space but don't want to write more text, so they set big font sizes and line spacings. In addition, such large size & spacing generally does not look very good!

Note that I set the font color as a very dark gray, not black. This is often done to reduce contrast, making it less uncomfortable for many people's eyes. You would not notice the color change, but it *is* easier to read.

Note that I did *not* set a **text-align**. Paragraph text should *always* be left-aligned, the default.

STANDARD WRAPPER

Here is an example of how the wrapper might be styled:

```
#wrapper {
  width: 1000px;
  margin: 30px auto 80px;
  border: 1px solid black;
  box-shadow: 6px 6px 12px rgba(0,0,0,0.4);
  border-radius: 12px;
  background-color: white;
}
```

Notice that I set the width at 1000px. There is a range where the width of the wrapper looks best on the greatest number of screen sizes. Usually that range is between 900 ~ 1100px.

CLASS RULE: *the wrapper must be between 900 and 1100px wide.*

Notice that I gave a 3-value margin rule. As explained in Chapter 6d a few pages ago, the first number is the top margin, the second is the left & right, and the last is the bottom. The left and right are set to **auto** so the layout will be jello, or centered. I added 30px to the top and 80px to the bottom so there would be comfortable space there. No margin space at all will put the page content too close to the top or bottom, and may seem crowded.

BORDER

The **border** property has three values: (1) border width (thickness), in pixels; (2) border style, which includes solid, dotted, dashed, and double; and (3) the border color. You can use a kind of shorthand, like this:

```
selector {
  border: 1px solid black;
}
```

However, you can also break it down into parts:

```
selector {
  border-width: 1px;
  border-style: solid;
  border-color: black;
}
```

The reason to break it down into parts is if you want to use a different shorthand for one of the parts:

```
selector {
  border: solid black;
  border-width: 5px 1px;
}
```

That could not be expressed with all the parts together; if you want different values for different sides of the box, it needs to be broken down like that. Notice that you can still use the simple “border” property for just two of the three parts.

BOX SHADOW

Notice that in the standard wrapper above, I used a **box shadow** to give the wrapper a 3D quality. It is certainly not required (most of these styles are variable and optional). The box shadow can give your objects a sense of “realness” to make them appear like they are floating off the page. This is used a great deal in “skeuomorphism,” an artistic style which makes objects on a web page appear like they do in real life. This was used heavily in the early 2000s, but many people today don’t use it as much. Currently, shadows are used in a subtle manner. This is what a box-shadow declaration looks like:

```
box-shadow: 3px 3px 6px rgba(0,0,0,0.4);
```

To create a box shadow, you need four values:

1. **left-offset** (how far to the right the shadow will be; use negatives for leftward)
2. **top-offset** (how far down the shadow will be; use negatives for upward)
3. **blur** (0px will be a sharp shadow, higher amounts will make a fuzzier shadow)
4. **color** (an **rgba** color usually works best for shadows)

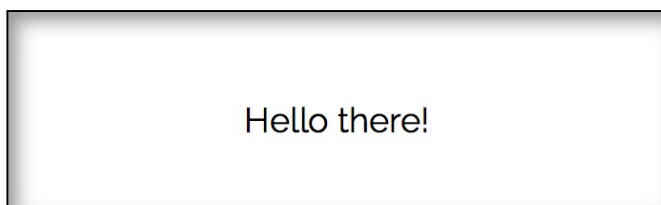
OK, what is “rgba”? The **rgba** value adds one more definition to red-green-blue: alpha. The alpha is a **transparency** setting. Instead of going from 0~255 or 0% to 100%, the alpha goes from 0 to 1. Zero is invisible, One is fully visible. Anything in-between will be semi-transparent. In between, use decimal values, such as my favorite, **0.4**. When using an alpha after an rgb, be careful to separate it with a comma but use a period for the decimal.

The reason why rgba looks better is because in the real world, shadows are not a color; instead, they cast a color shading—usually black—on objects. A shadow cast on a white object typically makes it a shade of gray, not completely black. A shadow cast on a blue object makes it darker blue. The “alpha” transparency setting in **rgba** will make the shadow color partially transparent, which is what a real shadow does.

One effect you can use is to make the shadow an inset:

```
box-shadow: inset 4px 4px 8px rgba(0,0,0,0.4);
```

The inset produces a shadow inside the shape:



Multiple value sets can be used for shadows to make more complex effects; we will study this in a later chapter.

TEXT SHADOW

Text shadows are the same as box shadows, except they apply to text:

```
text-shadow: 3px 3px 6px rgba(0,0,0,0.4);
```

Note that an inset cannot be used for a text shadow.

BORDER RADIUS

The **border radius** will round the corners of a rectangle. The higher the pixel value, the more it will be rounded. The maximum value that will be visible is half the size of the shortest side. If you make a square (for example, width: 200px; height: 200px;) then a border-radius of 100 will make it into a circle.

The following examples have a border radius of 0px, 10px, 15px, and 20px:

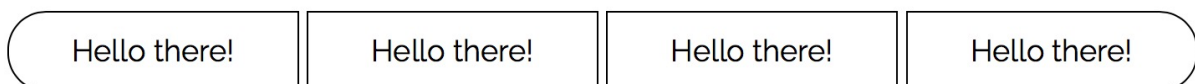


You can use shorthand for border radius values, but they will not be counted as top-right-bottom-left. Instead, they will be used as top-left-top-right-bottom-right-bottom-left (counter-clockwise from top left).

```
border-radius: 0px 20px 20px 0px;
```



The above effect can be used to make the end buttons in a nav menu rounded. Just apply a special class to the first and last buttons with special border radius for each:



I will introduce more special effects for this property later on.

THE BEYONCÉ

Finally, I would like to teach you a **diagnostic** or debugging style, which I call the "Beyoncé"—as in, "put a ring on it." Because so much of the structure of a web page is invisible in the browser, it helps to be able to look at the structure from time to time.

For example, we will be doing nav menus soon, and there is a lot of possibility for problems with margins and padding. In order to see what is usually hidden, I "put a ring on it," or in other words, place a red border around everything to make it show up visibly:

```
nav * {  
  border: 1px solid red;  
}
```

The * asterisk is a "wild card," which basically means "everything." By putting it after the nav selector, it is defining the selector as "everything inside the nav." This will make everything—the unordered list, the list items, and the links—all show up with red outlines. This often reveals the problem you might be having, and shows you what is going on with your code.

You should note that the added borders might add to the width of elements that you outlined, and so might distort things a bit. You should understand that in case the borders themselves create problems.

Of course, you want to remove the "Beyoncé" as soon as you have finished diagnosing.

6f. Making Different Page Styles in One Stylesheet

In your projects, I will ask you to make different styles and layouts for each level of the web site. For example, the index page will have one style, but the second level will have a different style. There may be a different background for each level, and maybe different colors. (Note: do not use different fonts or font sizes on different pages; these must always be the same on all pages.)

Many students feel it is easier to make a different stylesheet document for each level. **Do not do that!** Using separate stylesheets creates a lot of problems. First of all, different levels have only a few small differences in style—maybe image size and layout, and various background colors and images. The differences are usually much less than 5% of the code—meaning that if you make separate stylesheets, more than 95% of the code is repeated—a great waste of space!

Additionally, the biggest advantage of a stylesheet is that it applies to every page in your site, so that if you want to change all the pages, you only need to adjust one stylesheet. However, if you make multiple stylesheets, then you will have to change all of them to make a single change—a waste of effort!

Therefore, confine all of your CSS to a single stylesheet. In order to do that, and yet still have several different styles (one for each level, for example), you can do what I show below.

Let's say that you make a stylesheet with a background color:

```
body {
    background-color: rgb(200,220,255);
}
```

However, for a different level, you want to use a background image. If you change the body rule to have the background-image property, then the image will appear on the first level also, covering up the background color:

```
body {
    background-color: rgb(200,220,255);
    background-image: url(images/bgimage.jpg);
    background-attachment: fixed;
    background-size: cover;
}
```

In the above example, the background image will appear on all pages. So, how do you make the background image appear on only *one* level, and not all of them?

The answer is to use an id.

In the HTML for the index page, just leave the body tag as it normally is. However, on the second-level pages, add a special id to the body tag:

```
<body id="second">
```

Next, create a new id in the CSS:

```
body {
    background-color: rgb(200,220,255);
}
#second {
    background-image: url(images/bgimage.jpg);
    background-attachment: fixed;
    background-size: cover;
}
```

The #second id applies *only* to the pages where the body tag has the special id. It will *not* show up on the index page, where the body tag has no id applied to it.

You can make as many different level ids as you like—e.g., #third, etc.

In addition, you can affect *every* style in the #second and other id pages by using descendant selectors:

```
body {
    background-color: rgb(200,220,255);
}
#second {
    background-image: url(images/bgimage.jpg);
    background-attachment: fixed;
    background-size: cover;
}
#second header {
    background-color: rgb(230,240,255);
}
#second footer {
    background-color: rgb(30,40,155);
    color: white;
}
```

Notice that the rule to change the body tag is *not* "body #second," but is just the id alone. The id for the body is equal to the body rule.

Important Note: The primary rule for the base page may have many styles, such as:

```
wrapper {
    width: 1000px;
    margin: 40px auto 60px;
    border-radius: 40px;
    box-shadow: 0px 10px 16px rgba(0,0,0,0.4);
    background-color: rgb(200,220,255);
}
```

When you make the rule for another level, **only include the changed styles, do not repeat the ones that are the same.** In the below example, I do not include width, margin, border radius, or box shadow, as those stay the same. I only change the background color:

```
#second wrapper {
    background-color: rgb(255,180,235);
}
```

6g. Centering Inline and Block

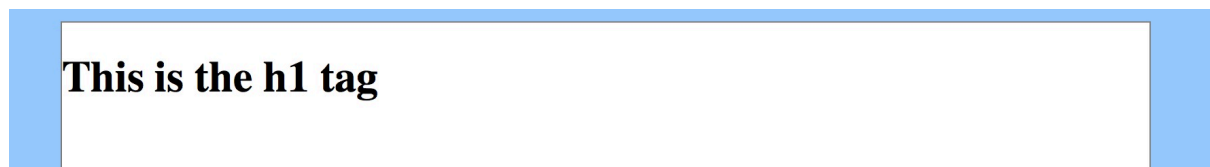
There are two ways to center content in a page; it depends on whether the content is **inline** or **block**. Each has a different technique.

INLINE CENTERING

It should be remembered that block tags automatically fill up all available space horizontally (from left to right). If your wrapper's width is 1000px, and you create an `<h1>` tag, it will automatically be 1000px wide—filling the available space in the wrapper.

The `<h1>` tag is a block, but the text inside of it is inline. In this example, I have created a wrapper (the white area against the blue body background):

```
<div id="wrapper">
  <h1>
    This is the h1 tag
  </h1>
</div>
```



The outlines of the `h1` tag are not visible. In cases such as this, I will create a "Beyoncé" so I can see what is happening to invisible tags:

```
h1 {
  border: 1px solid red;
}
```

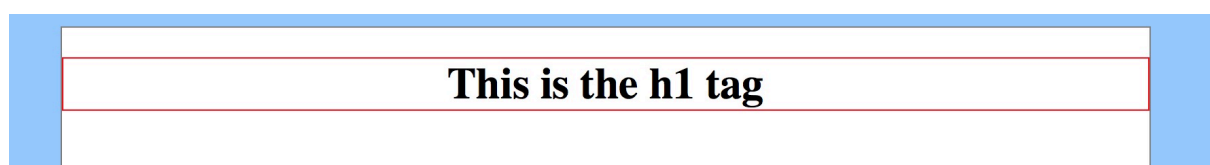


The `<h1>` tag has no alignment, as it fills up all available space left and right.

The text inside is **inline**; by default, inline content is aligned to the left.

To center the text inside the block use the CSS property `text-align` on the `h1` block:

```
h1 {
  border: 1px solid red;
  text-align: center;
}
```



You can see that the text inside the h1 block is now aligned to the center.

That is how you center inline content: use the `text-align: center;` CSS declaration, applying the style to the block element which contains the inline content.

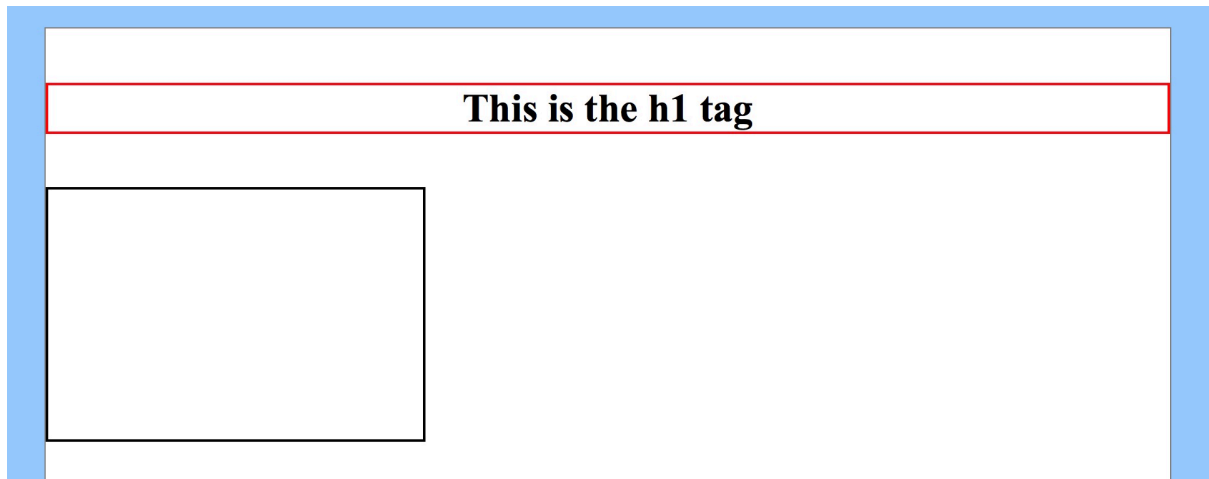
BLOCK CENTERING

To center a block element, a different technique is used.

First, you have to remember that a block element automatically fills up all available space from left to right. You have to realize that "centering" something means that the object is equally distant from both sides, left and right.

However, if a block fills up all of the available space already, *then it cannot be centered*—there is no free space to the left or right to center the object within.

Therefore, normal block tags at 100% width cannot be centered—they are, in a way, already centered. As a result, you can only center block tags if they are less than 100% wide, as in the case of a `<div>` tag you will use in the middle of your page:



Here is the code for that:

```
.box {
  width: 300px;
  height: 200px;
  border: 2px solid black;
}

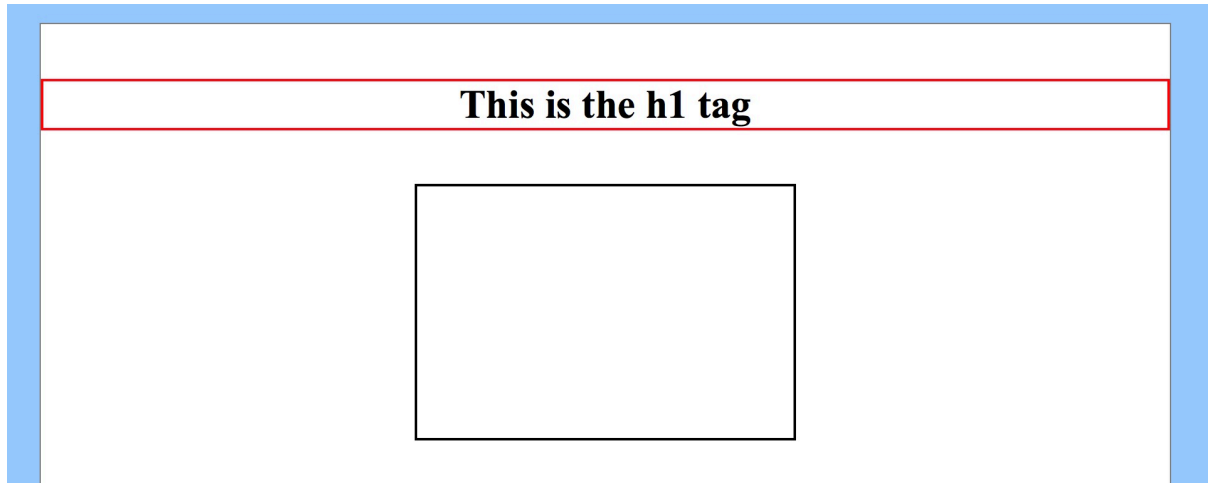
<div class="box"></div>
```

CHAPTER 6

We have actually learned how to center block tags already: use the auto value for the left and right margin:

```
.box {  
  width: 300px;  
  height: 200px;  
  margin: 0px auto;  
  border: 2px solid black;  
}
```

Note that I made the top & bottom 0px. The left & right auto will center the block:



Note: *You may not center the block tag with text-align.* That simply does not work. Text centering is for inline content—text and images—and has no effect on block tags.

In a future chapter, we will see how to center more than one block if they are all on the same line. That requires knowing things that we have not studied yet, so we will wait on that one.

6h. CSS Ordering

It is important that you *do not have multiple rules for the same selector*. Some students create main p rule near the top of their stylesheet, and then create another main p rule later in the same stylesheet. Even if the two rules have completely different properties, this should not be done! The two rules should be merged. If they remain separate, you might later make changes to one of the rules, and forget that the other rule exists; this could create conflicts.

In fact, there is a format that you must follow in this class from this time forward: you must **order your CSS rules**. That is, your CSS rules should be in a specific order, from the top of the list downward.

```

body                Always start with the body
#wrapper            Wrapper comes next, if you have one

/* ---- Structural tags and their common block tags ---- */

header
h1
h2
nav
nav ul
nav ul li
nav ul li a
main
footer
/* ---- Other block tags ---- */
h3
h4
h5
h6
p
blockquote
ul
ol
li
/* ---- Inline tags ---- */
img
b
strong
i
em

/* ---- Special ids and classes ---- */

#id
.class

/* The following is a set of rules to create an animation */
...
...
...
/* end animation rules */

```

CHAPTER 6

After the end of the normal tags, include any additional ids and classes. You may also wish to add groups of rules which are used for a specific effect or a special part of the page. Use the comment code to separate and label groups.

If you follow this ordering, then you can accomplish several positive goals:

- You can avoid having multiple rules for the same selector
- You can find rules quickly and easily
- You can make notes with comments for your future reference
- You can report problems or show special work to your teacher in comments
- Your teacher will be able to help you fix problems more easily
- You will make your teacher happy!

So remember, use ordering of your CSS rules!

From this point onward, I will expect all of your homework and projects to use stylesheets with rule ordering unless I give specific instructions to do otherwise. Using embedded or inline CSS should be avoided. If you think you have a special case, inform your teacher. Of course, after this class, if you continue web design, you can use whatever CSS in whatever location you wish. I require stylesheet-only and selector ordering because (like good code formatting) they are excellent habits to get into!

DESIGN

6i. Using Google Fonts

As discussed in Chapter 3j, it is best to use well-chosen, well-designed fonts which are best suited for your site's content and theme. Google Fonts has nearly 1500 font families which can be used for free on the web, making your site look far better.

Using Google Fonts is not too difficult, but it takes some practice before it becomes easy. Here are the instructions to use the service.

1. **Go to the web site**, fonts.google.com. You will see a page with several fonts visible. You can scroll down the page to see more. However, there are so many, it take a very long time to see them all.
2. **Narrow down your search.** On the right side, you can choose fonts by category and style. At the top right, you can search by name, if you know the same of the font you want.

Below that, you can choose fonts by category: serif, sans serif, display ("fantasy"), handwriting ("cursive"), and monospace. By un-checking categories you do not want or need, you can narrow down the number of choices shown.

Below that, you can choose how the fonts are sorted, and what languages the fonts are for (most are English, but a few are for other languages).

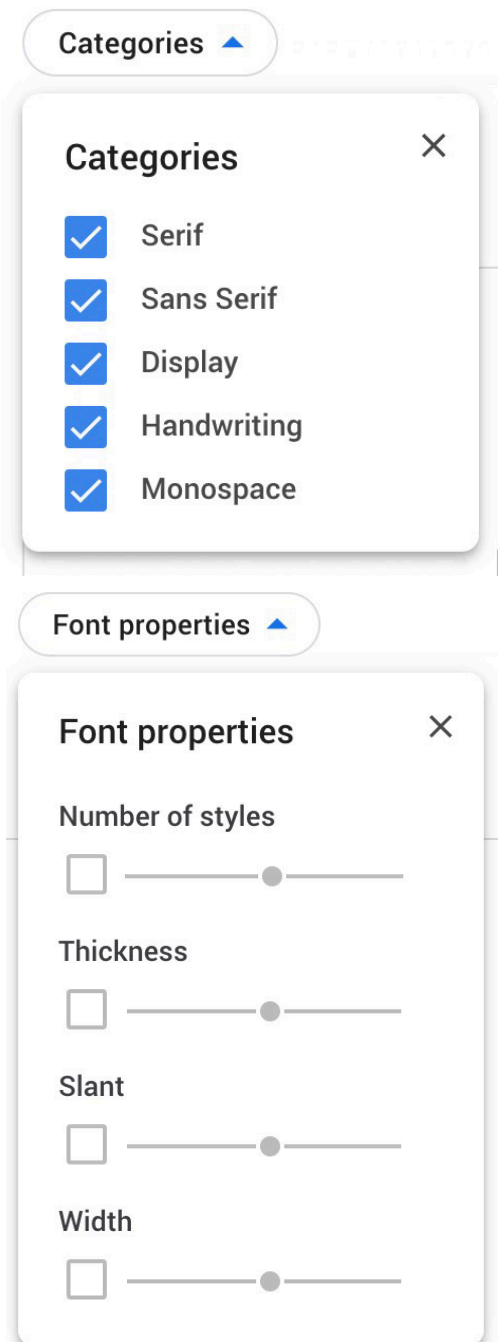
Finally, you can choose by the font style features:

Number of Styles: styles mean either weight or italic. Most fonts have at least a "normal" font weight, and many include a special italic version. Some fonts come in many different weights, however. You know weights as "normal" and "bold," but Google uses the professional scale of 100 ~ 900. 100 is "Thin," 400 is Normal, 700 is Bold, and 900 is Black. There are nine possible thicknesses, and each may have an italic version.

Thickness: this is different from weight. This means that the normal version of the font will be wider and darker than normal.

Slant: you can choose fonts by how much they lean to the right, or stand up straight. Many handwriting fonts have slants.

Width: this is equivalent to condensed or expanded fonts—fonts where the letters are very thin or fat.



3. **Choose fonts to use.** First, find a font that you want to choose. Note how many styles it has. Each different thickness and italics version of each font is counted as a style. There can be up to 18 styles for any one font—nine thicknesses, and an italic version of each thickness. When you hover your cursor over a font, there is a slight increase to its shadow. If you click on the font's panel, you will go to a page which shows all the possible styles.



Google Fonts Return to classic site Browse fonts Featured Articles About

Big Shoulders Text

Designed by [Patric King](#) [Download family](#)

[Select styles](#) [Glyphs](#) [About](#) [License](#) [Pairings](#)

Styles

Type here to preview text

Almost before we knew it, we had left the ground.

Size 30 px

Thin 100	Almost before we knew it, we had left the ground.	+ Select this style
Light 300	Almost before we knew it, we had left the ground.	+ Select this style
Regular 400	Almost before we knew it, we had left the ground.	— Remove this style
Medium 500	Almost before we knew it, we had left the ground.	— Remove this style
Semi-bold 600	Almost before we knew it, we had left the ground.	+ Select this style
Bold 700		

Note that to the right of every style, there is a "Select this style" button:

If you click the button, it changes to "Remove this style":

— Remove this style

You should select all the styles that you think you will use. At the very least, select the **Regular** and **Bold** styles, if both are there. You should also choose any thinner or thicker styles if you think you are going to use them, along with italics, if you plan to use them as well.

[+ Select this style](#)

4. **Review your choices.** On the right-hand side, a "Selected families" pane should have appeared, with tabs for "Review" and "Embed." The Review tab will show you all the fonts you selected so far.

By clicking on the small circle with a minus-sign in it (to the right of each style name), you can remove that one style.

By clicking on "Add more styles," you will be taken to that font's page where you can select other styles from the list.

By clicking on "Remove all," obviously you can get rid of all styles of that font, and the name will disappear from the list.

5. **Pare down the list.** It is common to choose too many fonts at first. Using these fonts adds download time to your page; do not choose many more than you know you will need. Get rid of any you will not be using.
6. **Get the Code.** Click on the **Embed** tab. There are two things that you will need to add these fonts to your stylesheets: the `@import` code, and the CSS font-family declarations.

@import rule: Under "Embed Font," you will see red buttons named "`<link>`" and "`@import`"; click on `@import`. Select only the code **inside** the `<style>` tags; do **not** select the style tags. Copy the selected code. **This code must be pasted in the stylesheet;** you must put it at the very top of the sheet.

CSS Declarations: below the `@import` is a list of CSS declarations for each font ("Specify in CSS"). Copy these and paste them in a text document somewhere so you can access them. You can also add them to the bottom of the stylesheet inside `/* comment markers */`. You will use these in your CSS stylesheet when you want to apply a style. Now that you have added the stylesheet `@import` link to the sheet, the fonts will be available.

Caution: many times, web designers will change their minds about fonts they want to use. After choosing their fonts earlier, they will go back to add more Google Fonts. A common mistake by beginners is to *add* multiple `@import` rules. **Do not do this.** Instead, make a list of all the fonts that you already use (delete any you discovered you will not use). Then, go to Google Fonts, **input all of the fonts you chose before**, and then add any new fonts you want.

After you get the new Embed `@import` code, **replace** the old one with the new one—do **not** add the new one after the old one. There should be only **one** `@import` rule for Google fonts!

Selected families ×

Review Embed

Big Shoulders Text

Regular 400 —

Medium 500 —

Bold 700 —

Add more styles Remove all

Dancing Script

Regular 400 —

Bold 700 —

Selected families ×

Review Embed

To embed a font, copy the code into the `<head>` of your html

`<link>` `@import`

```
<style>
@import url('https://fonts.goo
leapis.com/css2?family=Big+Shou
lders+Text:wght@400;500;700&fam
ily=Dancing+Script:wght@400;700
&family=Jost:wght@200;400;500;7
00&display=swap');
</style>
```

CSS rules to specify families

```
font-family: 'Jost', sans-serif
;
font-family: 'Dancing Script',
cursive;
font-family: 'Big Shoulders Tex
t', cursive;
```

6j. Using @font-face

Google Fonts has a great selection, but there are many font styles they do not have. You can complete the font choices for your site by using free fonts available elsewhere on the Internet. One site in particular has a wide selection of very interesting and useful fonts:



<https://dafont.com>

DaFont has thousands of fonts, for practically any style you can think of. If you want movie poster fonts, handwriting fonts, ethnic fonts—or even a font for [Walt Disney’s handwriting](#), this is the place to get them. You can search by category or by name.



The problem is, these are not “web fonts” as they cannot be used like Google Fonts. They are actual font files, ones that you can install in your operating system, like other normal fonts.

However, there is a CSS solution with which you can use these fonts perfectly on your web site: the **@font-face** rule. Here’s how it works:

1. Download the font(s) from DaFont, and isolate the **TTF** file. You may want other versions also, if they have **OTF**, **WOFF**, or **WOFF2** types. However, TTF is the best, and can be used alone if you wish.
2. Create a **fonts** folder in your site folder (like the “images” folder), and put the font files there. You do not have to add any accompanying files, just the fonts.

▼	fonts	Today, 9:42 PM	89 KB	Folder
	 waltograph42.otf	8/31/04, 9:18 PM	54 KB	OpenType® font
	 WEST___ 2.TTF	10/29/99, 8:04 AM	35 KB	TrueType® font

3. If the font names, like the ones above, are odd or use non-standard or unwanted characters, you may rename the font files, as shown below.

▼	fonts	Today, 9:44 PM	89 KB	Folder
	 walt.otf	8/31/04, 9:18 PM	54 KB	OpenType® font
	 western.ttf	10/29/99, 8:04 AM	35 KB	TrueType® font

4. Next, go to your CSS stylesheet and create this code, with one font-face rule per font:

```
@font-face {
  font-family: "West";
  src: url(fonts/western.ttf);
}
@font-face {
  font-family: "Disney";
  src: url(fonts/Walt.otf);
}
```

Notice that the font-family name can be whatever you want; it does not have to match the original font file name, or the name you gave to the font file.

5. In the CSS rules, use the fonts based on the font-family names that you used in the @font-face rule under “font-family”:

```
h1 {  
  font-family: "West", display;  
  font-size: 42pt;  
}  
h2 {  
  font-family: "Disney";  
  font-size: 28pt;  
}
```

That will give you the result below:



Chapter 6 Checklist

- What is a *class*? How is it marked in CSS? What HTML attribute is used for it?
- What is an *id*? How is it marked in CSS? What HTML attribute is used for it?
- How are the class and id different?
- How many times can you use each on a single HTML page?
- What is the div tag for? Why do we use it?
- What is the wrapper for? Why do we use it?
- What is a *descendant* selector? What does it look like? When is it used?
- What is an *additive* selector? What does it look like? When is it used?
- What is CSS Shorthand? If the property for a box has 1, 2, 3, or 4 values, what do each of the numbers mean?
- What is the maximum font size and line height for paragraphs?
- What is the best width range for the wrapper?
- How do you use box-shadow and border-radius?
- How do you center inline content and block tags differently?
- What is the reason for ordering CSS rules? What order should they be in?
- What are the five general categories of fonts?
- What are "web safe" fonts?
- How can you use more fonts than just the web-safe ones?
- What are advantages of Google Fonts?
- Did you successfully add Google Fonts to a test web page? If not, please try.
- What happens when you add a background image which is smaller than the page?
- What makes a good or bad tiled background image?
- What are the acceptable file size, resolution, and quality limits for background images?
- What CSS properties and values should you use for a full-sized background image??
- What are the four style considerations for choosing a good background image?
- What are the two CSS properties needed to best present a non-repeating background image which will resize with the page? Did you try it?
- How do you cite a background image?
- What are the acceptable font types?
- What is the code for doing `@font-face`?

Chapter 7: The Nav Bar Layout

TECHNICAL

7a. The Cascade

Normally, browsers use black text against a white background. This is called the **browser default style**. Let's say that you have a **stylesheet** where the font color is set to red. You have a web page that uses the stylesheet, but it also has **embedded** CSS, which sets the font color to green. Finally, you add **inline** CSS to the <body> tag which sets the font color to blue.

What color will your text be? Black, red, green, or blue?

The answer is *blue*. When there are conflicting styles, there is a fairly clear order of importance. That order, from weakest to strongest, is called the **cascade**:

4. **Browser defaults** (Times New Roman 12pt. black text, white background)
3. **External CSS Stylesheets** (red background)
2. **Internal (Embedded) CSS** (green background)
1. **Inline CSS** (blue background)

In other words, a stylesheet CSS rule will be used instead of browser defaults. However, if there is an internal (or embedded) style rule, that will be used instead of the stylesheet rule. Finally, if you use inline CSS, that will be the final winner.

IS THERE ALWAYS CONFLICT? No.

The cascade only has an effect when *the same CSS property* is defined in different locations for the same web page object.

For example, if you create a stylesheet rule which sets the font size of <h2> to 16pt, that will not affect the <h1> tag, the <p> tag, or any other tag, no matter where their rules may be located.

If you create a stylesheet rule which says the h2 tag has a font size of 16pt, and then you create an embedded/internal rule which says the h2 tag has a font color of green, then again there is no conflict.

However, if your stylesheet rule says that the h2 tag has a font-size of 16pt, but your embedded/internal CSS has a rule saying the h2 tag has a font-size of 20pt, then the stylesheet rule will be ignored, and the h2 tag will appear at 20pt size.

!IMPORTANT

If, for some reason, you want to override the cascade and make a stylesheet rule be followed instead of an embedded rule, you can add `!important` to any one declaration:

```
h1 {
  margin: 20px 0px 0px;
  font-family: 'Abril Fatface', cursive;
  font-size: 34pt !important;
}
```

The font-size value in the rule above will be followed, no matter what other font-size rules are given elsewhere for h1 tags.

SMALLER CASCADE CONFLICTS

Let's say you have an id called "ponta":

```
#ponta {
  color: red;
}
```

You also have a class called "doggie":

```
.doggie {
  color: blue;
}
```

You have a tag which contains both:

```
<p id="ponta" class="doggie">
```

Which color will the text be? The answer is *red*. An id rule will win over a class rule, no matter where they are located or what order they are in.

The order of declarations is also important. Sometimes, by mistake, beginning web designers will repeat the same CSS property:

```
h1 {
  font-size: 20pt;
  color: rgb(255,0,0);
  text-align: center;
  font-size: 16pt;
}
```

Notice that the font size is repeated. Which one will be used? The answer: the last one—in this case, 16pt.

The same is true if a rule is repeated:

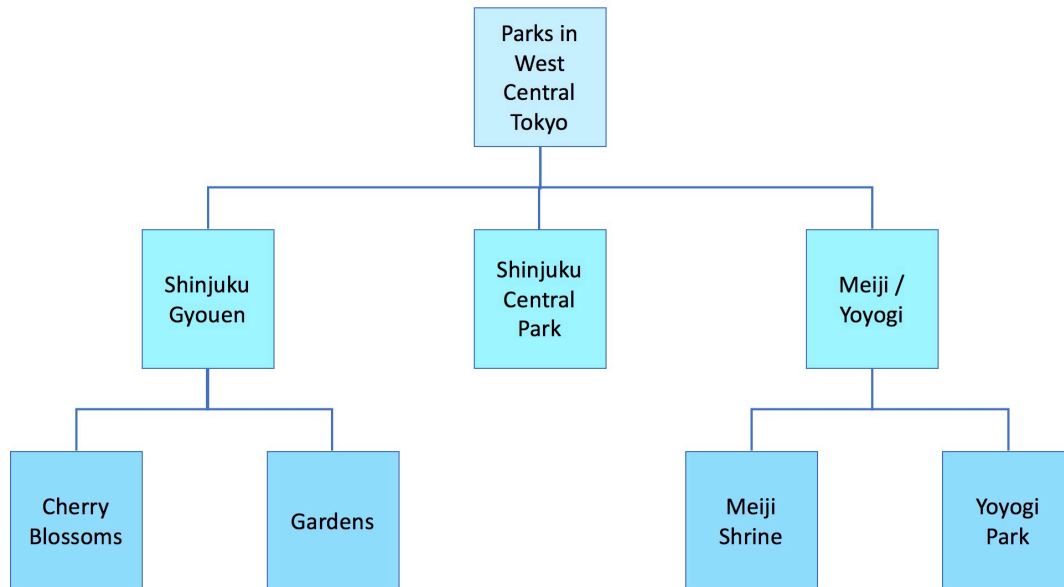
```
h1 {
  font-size: 20pt;
  color: rgb(255,0,0);
}
h1 {
  text-align: center;
  font-size: 16pt;
}
```

In the above case, the last font-size declaration (16pt) will be used. Both the color and the text-align will be recognized; the second rule does not erase the first.

7b. Site Structure

Before, we learned about Site Maps and how to plan a web site. I would like to cover more information about how that works.

Web sites are usually set up with a tree structure.



There is usually one **index page** at the top. This gives the main information about the site, generally describing the topic. The index page often is more colorful and interesting, and has larger photos, or even one large dominating photo that sets the theme for the whole site.

CLASS RULE: *the link for the index page must be in the <h1> tag on all pages.*

On the **second level**, there are category pages which split the topic into separate areas. Each page should be different from each other page on that level, but connected to the main site topic—very similar to how body paragraphs do in an academic essay. Each page on the second level will be *more specific* than the main page, taking a closer look at one part of the main subject.

CLASS RULE: *the link for the 2nd-level pages are visible buttons in the nav bar.*

Finally, the **third level** gets into detail about specific examples, or at least more-detailed information about the topic. Each third-level page branches from a second-level page, and gives even more detail based on the topic of the whole site and the category of the 2nd level.

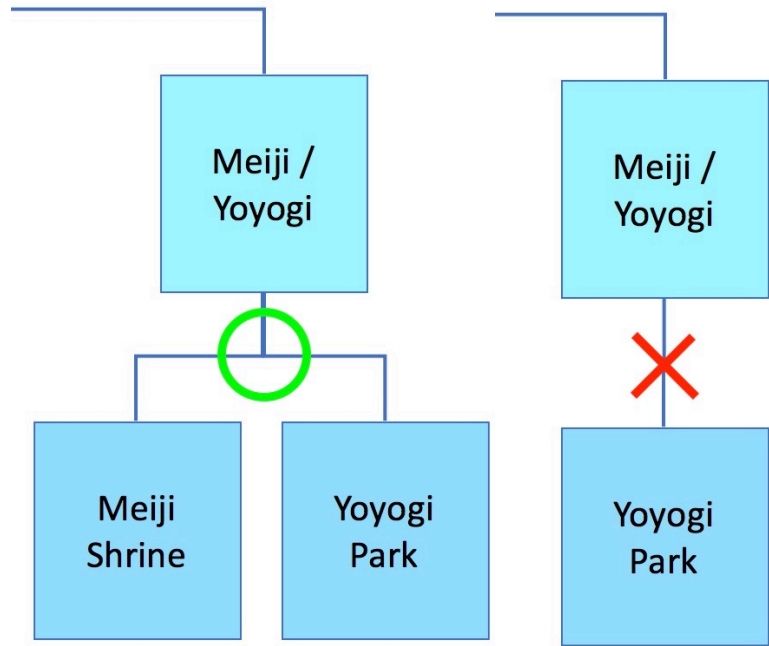
CLASS RULE: *the link for the 3rd-level pages are invisible drop-down buttons.*

In this chapter, we will only use the index and 2nd-level pages, required for Project #1.

Warning: When you create a new level of pages, each branch must have two or more new pages. You are not allowed to have only one page branch down from one page.

Think about having a house with a hallway, but there is only one door in the hallway. Why have the hallway at all? Why not just make the room bigger to include the hallway space?

The same idea is true here. If there is only one branch example going down one level, it is easier to simply put the next-level page content into the main page.



CODE

7c. Creating a Simple Menu for a 2-Level Site

In a later chapter, we will learn how to make more dramatic drop-down menus. However, for now, let's make just a simple multi-button nav bar.

Remember, the link to the index (main) page should be in the header, in the <h1> tag:

```
<h1>
  <a href="index.html">Site Title</a>
</h1>
```

If you make this code, the h1 site title heading will become blue and underlined. That does not look good. Therefore, in order to make the heading look appropriate, you must add a special rule in the CSS:

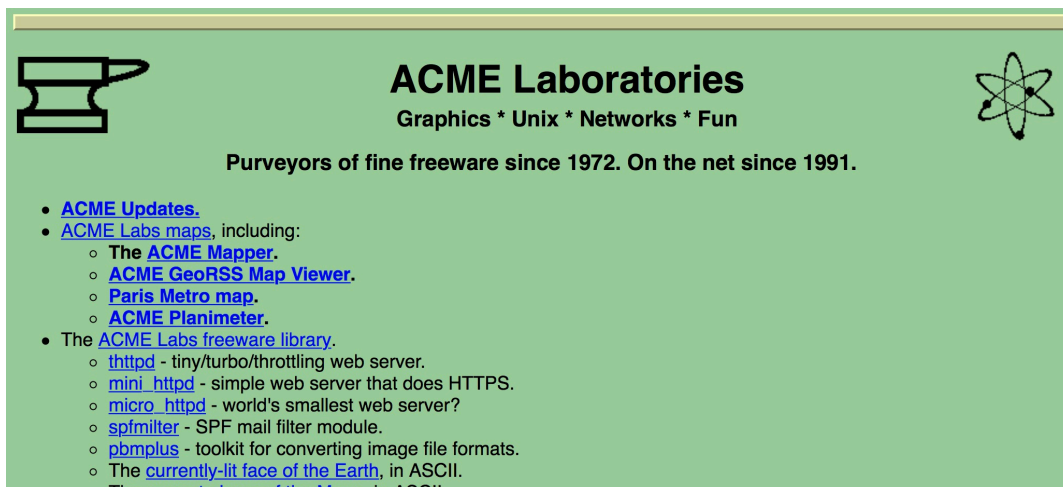
```
h1 a {
  color: black;
  text-decoration: none;
}
```

Link text must always be specified because **the default styles of the <a> tag will always overpower the inherited styles**. The font family, font size, and other text attributes will be inherited, and will be the same as whatever the h1 CSS rules declares for text. However, the <a> tag carries its own style for color (the usual default is blue) and decoration (underline).

If you want to override and replace these styles, you always have to create a special rule for the <a> tag which creates a different color and decoration.

BASIC CODE

If you remember back to Chapter 3h, the tags used very commonly to create a menu are and , the tags for making a bullet list. Why a bullet list? Because the nav section is essentially a list of links. You simply have to give them CSS styles to change their appearance. The oldest web pages presented links as bullet points:



This initial choice has remained until this day. However, with the addition of CSS, bullet points in a vertical list were transformed into buttons, a more human-like interface.

First, let's look at a very plain set of links in a nav bar:

- [Second Page](#)
- [Background](#)
- [Information](#)
- [Conclusion](#)

See? The same as the old web sites and their lists of links. The code for this is:

```
<nav>
  <ul>
    <li><a href="second.html">Second Page</a></li>
    <li><a href="back.html">Background</a></li>
    <li><a href="info.html">Information</a></li>
    <li><a href="conc.html">Conclusion</a></li>
  </ul>
</nav>
```

As you can see, it is very simple. Inside the `nav`, there is a `ul` tag.

Important: the `ul` tag is often forgotten because we do not see it. Its purpose is to contain a list; the `ul` tag by itself is not a button. Also, since it is inside another container—the `nav` tag—it seems unnecessary. But *it has a default value of padding-left: 40px; which is used to indent the bullet list. You may have to remove this.*

Inside the `ul` tag, there are four `li` tags.

Each `li` tag has an `a` tag, a link to the specific page.

Important: the `a` tag should be **inside** the `li` tag, and not the opposite. Remember, inline tags (like the `a` tag) should go inside block tags, and block tags should *not* go inside inline tags. This will require us to add extra formatting to the `a` tag later.

The list we have so far is actually very serviceable as a vertical or palette-shaped nav area, or, even better, a basic site map for the footer. You can use it in this form in special circumstances, if you wish. However, for the nav bar in most sites, we want to make these into buttons. Everyone *loves* buttons!

Side note: it is not absolutely necessary to use the `ul` and `li` tags to make nav menu buttons. You could use `div` tags, and in HTML6 there may be new ways of doing this. However, it is a standard in web design to use the `ul` and `li` tags. That's what we'll do.

LIST TYPES

```

<nav>
  <ul>
    <li><a href="second.html">Second Page</a></li>
    <li><a href="back.html">Background</a></li>
    <li><a href="info.html">Information</a></li>
    <li><a href="conc.html">Conclusion</a></li>
  </ul>
</nav>

```

The HTML is now fine as it is. The rest of the work can be done in the CSS rules.

First, we want to remove the bullets from the bullet list points. There is a CSS property which allows us to make a variety of list types:

list-style-type:

The values for this property include:

hiragana;	enables list with あ, い, う, え, お ordering
hiragana-iroha;	enables list with い, ろ, は, に, ほ ordering
katakana;	enables list with ア, イ, ウ, エ, オ ordering
katakana-iroha;	enables list with イ, ロ, ハ, ニ, ホ ordering
lower-roman;	enables list with i, ii, iii, iv, v ordering
upper-roman;	enables list with I, II, III, IV, V ordering
lower-latin;	enables list with a, b, c, d, e ordering
upper-latin;	enables list with A, B, C, D, E ordering
circle;	changes bullets to white circle
square;	changes bullets to squares
none;	<u>makes a list with no markers or bullets</u>

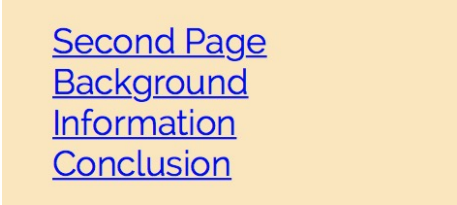
The last item, list-style-type: none; is what we want for the nav bar list. Since the tag is the one with the bullet marks, the style could look like this:

```

li {
  list-style-type: none;
}

```

This changes the list items so they look like this:



[Second Page](#)
[Background](#)
[Information](#)
[Conclusion](#)

USING DESCENDANT SELECTORS

Here we run into a problem: since we just styled the list items to have no bullets, we will find that all lists on the whole site have lost their bullets. We don't want that! We only want the list items in the nav section to lose the bullets; we want all other unordered lists to keep their bullets, lettering, and numbering.

How can we make the difference?

Remember that we learned about descendant selectors. You can make a selector which only affects tags which are children of other specific tags.

In this case, we can make our rule apply only to list items inside the nav area. It is usually better to include the whole list of parent and children tags, so the style rule would look like:

```
nav ul li {
  list-style-type: none;
}
```

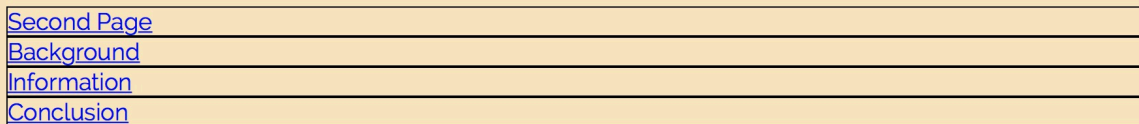
This affects only list items in an unordered list in the nav tag. No other list items are affected.

MAKING BUTTONS FROM A LIST

The next step is to make the list items look like buttons. We want our buttons to have borders, so let's begin there. Adding borders is actually a good way to see your layout; the borders show you where tags extend invisibly.

So, let's add a dark gray border to the tags:

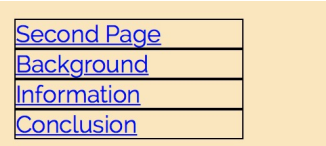
```
nav ul li {
  list-style-type: none;
  border: 1px solid rgb(100,100,100);
}
```



Now we can see that the list items extend all the way to the right—which is what we would expect, since that's what block tags do.

Naturally, if we want each of these to be buttons side by side, we need to make them smaller!

```
nav ul li {
  list-style-type: none;
  border: 1px solid rgb(100,100,100);
  width: 150px;
}
```



Now they look a little more like buttons! By the way, they don't need to be 150px wide—you decide what width is best for your page. However, we still have a problem: the buttons are still block tags (since they are `` tags), and therefore they all live on their own separate line, with a break above and below. How can we change that?

SPECIAL NOTE: BE ORIGINAL!!

Many students will create their web pages with buttons which use *exactly* the style show here. For example, many will always set the button width to 150px, because that's what the book says.

DO NOT DO THAT.

As per the instructions in this chapter, you have to use the (1) list-style-type, (2) display, (3) text-align, and (4) padding... *but that's all!* The border, border radius, width, colors, underlines, and even the amount of padding **are purely creative settings** which you must choose **independently**.

I do *not* want to see everybody with the same boring buttons!!

THE DISPLAY PROPERTY

It turns out that there is a CSS property for inline and block: `display`. By using `display`, you can change a block tag to be inline, or change an inline tag to be a block.

```
display: block;
display: inline;
```

Remember, **inline** means that objects go side by side, from left to right, in a line, just like text normally does. **Block** means that each object has its own line, and fills up all the available space in that line from left to right; there is a break above and a break below.

With the `display` property, you could turn images into a block tag:

```
img {
  display: block;
}
```

Similarly, you could make an `` tag into an inline tag:

```
li {
  display: inline;
}
```

What does this mean? Well, as you recall, **inline** acts like text—free-flowing and without a set shape. In contrast, **block** acts like a rectangle, breaking the lines above and below, and filling 100% of the available space.

Using the **display** property in CSS, you can make blocks into free-flowing shapeless entities, and you can turn inline ones into blocks.

However, without nav buttons, we need a compromise: we need something that acts like a block—a rectangle shape—but also can be side-by-side with other objects on a line.

As it happens, we have such a value: **inline-block**. This retains the rectangle shape, but gets rid of the breaks above and below. That's what we need here. So, for our list item style rule, we add:

```
nav ul li {
  list-style-type: none;
  border: 1px solid rgb(100,100,100);
  width: 150px;
  display: inline-block;
}
```

This has the effect of making our buttons all line up in a single row:



It is important to note that inline-block works well for plain buttons, but in a later chapter, we will learn how to make advanced drop-down menus for multi-level sites. In those menus, *you must not use inline-block* because the drop-down menus will not work with it. Again, inline-block is only for simple, stand-alone buttons!

THE LINKS

We're getting closer! However, there are still some things to be cleaned up. First of all, if you move the cursor inside the button area, you will note that *only* the link text is a link, and *not* the rest of the button! This can confuse people. Therefore, we need to alter the link style.

We can see this effect if we use my patented "Beyoncé" technique:



You can see that the red outline around the `<a>` tags shows the clickable area. We want the *whole button area* to be clickable.

In addition, we want the link color to be different, and to get rid of that pesky underline effect, just like we did in the link inside the `<h1>` tag. Since so much of this must be applied to the `<a>` tags specifically, we have to add a special rule just for them.

In addition, we must make this a descendant selector also, unless we want all links in the whole document to act like the nav bar links—which we don't.

First, create a link selector which *only* affects links inside `` lists inside the nav bar.

```
nav ul li a { }
```

Then we must make the links take on the same shape as the `` tag buttons. However, since link text is inline, that means we must also make these into **block**:

```
nav ul li a {
  display: block;
}
```

CHAPTER 7

We could use inline-block, and that would make the link into a rectangle... but only as wide as the text is; it will not fill out 100%. It would require another line of code to set the width, and would make things more complicated. That's why we should use **block** instead.

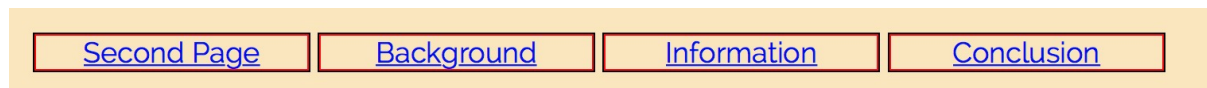
We also want the text to be centered:

```
nav ul li a {
  display: block;
  text-align: center;
}
```

Note that the width I set in this rule is *the same as* the width I set in the list item rule.

Also note that the text-align in the code here allows you to center the text. You could also set the text-align in the nav ul li rule; that could also work.

All of this not only centers the text, it makes the entire area clickable (link areas in red):

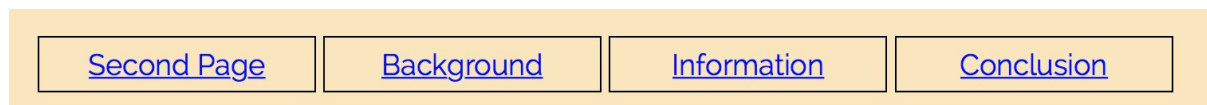


Next, you might notice that the buttons are not at all tall; the text almost touches the top and the bottom of the buttons. There should be more space in there.

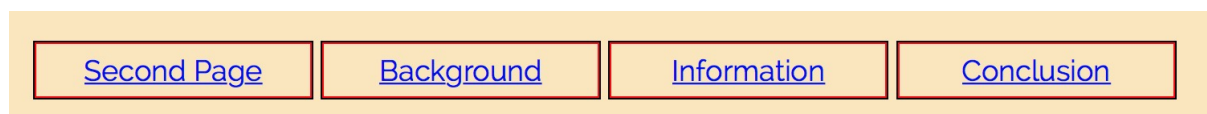
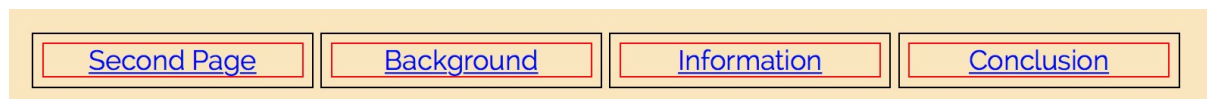
If you recall our previous discussion of the "box" in CSS, you may remember that to put space between the content (the button text) and the border, you need to add **padding**. In this case, just a little will do:

```
nav ul li a {
  display: block;
  text-align: center;
  padding: 5px;
}
```

That adds 5px of space inside the border. Notice that it adds to the left and right also, making our buttons 160px wide in this case. If you don't want that, just use `padding: 5px 0px;`.



The padding *could* go in the nav ul li rule, but if you do that, then the padding area will not be clickable; this could cause errors when a user tries to click on a link. If the padding is added in the link tag style, the area is clickable.



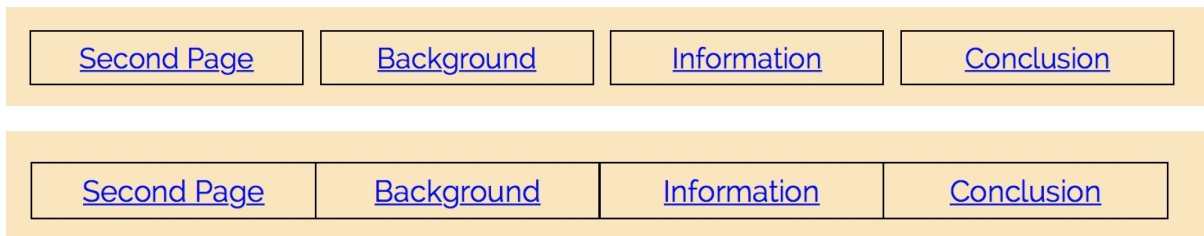
One more essential item: do you think the buttons look too close together? Let's add some space between them. This is added space outside the border, so that would be a **margin for the tag**. In fact, we only need a margin on the right and left, but not the top or bottom:

```
nav ul li {
  list-style-type: none;
  border: 1px solid rgb(100,100,100);
  width: 150px;
  display: inline-block;
  margin-left: 5px;
  margin-right: 5px;
}
```

The reason we do left and right is to keep the buttons exactly centered. We could also do this:

```
margin-left: -5px;
margin-right: -5px;
```

Here are the changes now, first for the extra margin, and then for taking it away:



Much better! For this exercise, let's go for the extra margins. Now for a few additions which make the buttons really stand out: background color, rounded corners, and drop shadows!

BOX AND TEXT SHADOWS

In Chapter 6e I introduced box shadows. Add one to get a shadow effect for the buttons:

```
nav ul li {
  list-style-type: none;
  border: 1px solid rgb(100,100,100);
  width: 150px;
  display: block;
  box-shadow: 2px 2px 4px rgba(0,0,0,0.5);
}
```

Shadows should naturally be a semi-transparent black. Using other colors might look interesting, but such shadows often strike people as strange and even nonsensical.

ADDING EFFECTS

The `background-color` property can be used to give a background to the buttons. Notice that they have the same color as the nav bar. I am adding: `background-color: rgb(255,255,255);`

The `border-radius` property can be used to round the corners. Try various pixel sizes and test each one out to see how much rounding looks best to you. I chose 10px.

CHAPTER 7

The shadow can be added with `box-shadow`. I am making a box shadow 2px to the right, 2px down, with a 4px blue, and a half-transparent black color:

```
nav ul li {
  list-style-type: none;
  border: 1px solid rgb(100,100,100);
  width: 150px;
  display: inline-block;
  margin-right: 10px;
  background-color: rgb(255,255,255);
  border-radius: 10px;
  box-shadow: 2px 2px 4px rgba(0,0,0,0.5);
}
```

One final adjustment must be made to the `link` rule: get rid of the underlines! Note that you cannot set the color or underlines of links with the `nav ul li` rule; it must be done in the `nav ul li a` rule.

```
nav ul li a {
  display: block;
  text-align: center;
  padding: 5px;
  text-decoration: none;
}
```

You can further style the links so that the font, font size, and font colors are different. For now, let's just keep it simple. This is the result:



Cool! We could continue, adding background colors, different fonts, etc., but I assume you get the idea. What we have gone over here will allow you to make nice buttons for a nav bar in your current web site!

One more point: you may notice that your buttons do not start at the left edge, and if you try to center them, they seem to go too far to the right. There is a reason for this: the default style for the `` tag includes 40px of padding on the left side as a way to indent the bullet list. To make the buttons align correctly, you should set the `padding-left` of the `nav ul` to 0px.

In order to center the buttons, find their total width, set the width of the `nav ul` to that width, then center the `nav ul` by setting the left and right margins to `auto`.

Note: if you want the buttons to be centered, you will need a rule for the “ul” container:

```
nav ul {
  padding: 0px;
  text-align: center;
}
```

The padding is set to 0px because the “ul” tag has a default style with 50px padding on the left side. Also note that this centering only works because the nav buttons (`nav ul li`) use `inline-block`. Drop-down menus, which we learn in Unit 3, don't have that.

DESIGN

7d. Pseudo-Classes & Pseudo-Elements

PSEUDO-CLASSES

At this point, there is a CSS feature which we haven't come to yet—the Pseudo-Class. A **pseudo-class** is a variation on a CSS selector which causes its behavior to change in a specific way. the pseudo-classes we will study here are mostly related to links.

Pseudo-classes are added after a selector, following a colon:

```
a {
    color: blue;
}
a:link {
    color: red;
}
a:visited {
    color: green;
}
a:hover {
    background-color: rgb(200,230,255);
}
a:active {
    background-color: yellow;
}
```

If you create a rule for the `<a>` tag and set a color, but there is no pseudo-class, then the color will be set for all links, including visited ones. (Normally, links are blue if they are not visited yet, and purple if they have been visited.)

If you create an `a {}` rule with the `:link` pseudo-class, then *only* the unvisited links will be set to that color. While the text color is usually the one property that is set for this pseudo-class, it is possible to add other effects as well.

If you create an `a {}` rule with the `:visited` pseudo-class, then *only* the visited links will be set to that color. While the text color is usually the one property that is set for this pseudo-class, it is possible to add other effects as well.

If you create an `a {}` rule with the `:hover` pseudo-class, then that style will be visible only when the cursor is over the target area, and will change back when the cursor leaves the area. Many various styles and effects can be used. In addition, **the hover pseudo-class can be used on almost any tag; no link is necessary for this one.**

If you create an `a {}` rule with the `:active` pseudo-class, then that style will be visible only when the link is actively being clicked—therefore, the effect will only last for a fraction of a second.

Important note: the pseudo-classes must be in that order (especially *hover* after *link* and *visited*, and then *active* after *hover*) in order for them to work correctly.

Using these pseudo-classes will allow you to style your links as you wish.

The hover is the most interesting of these effects. As stated above, it can be applied to almost any tag, and a variety of CSS styles can be used effectively with it. Undoubtedly, you have seen the hover effect used countless times on the web.

Please note that the hover usually requires **two** rules: a normal rule, and then a rule with the same selector and a hover.

Here is an example using a hover:

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

In the above example, the first rule is normal. The second rule has the hover pseudo-class.

What does this do? Simple: in the above example, if you move the cursor over the paragraph, the text color becomes red. Go ahead and try it!

The hover was intended for hyperlinks—you have probably noticed many links where hovering over the link changes the text color or perhaps makes underlines appear or disappear—but it works on almost any visible tag, block or inline.

Warning: avoid any hover effect that makes the tag or its content change size. If you increase padding, margins, borders, text size—sometimes even just making the text bold—it will increase the size of the content, which will make all content below it on the page jump down suddenly! That looks really bad. There are ways to make it work, but my recommendation is to avoid size changes altogether, at least until you become much more skilled at it.

The hover effect is great fun to use when you first learn it, but it must come with a warning: don't over-use it. I will repeat a design principle I use often: be **functional** and **subtle**.

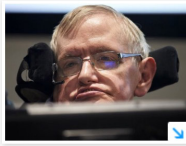
Functional means that when you use a design element that stands out, especially an animation of some sort, it should serve a purpose. It should not be used *just* to look cool. Looking cool is fine, but it is even better when there is a reason for it. For example, when you use a hover to change the color of a link, it is functional because it calls attention to the link. Furthermore, people recognize that as a common link feature. If you hover over text and it changes color, you expect it will be a link. That's a *functional* use of an effect.

Subtle means that an effect is minimal. For example, you might think that most web sites use black text on a white background. Usually, that's not true. For example, in articles on the USA Today web site, as shown below, the colors are *not* black and white:

Stephen Hawking will test his theory that humans must leave Earth. Let's hope he's wrong.

USA TODAY NETWORK Mary Bowerman, USA TODAY Published 7:23 a.m. ET May 4, 2017 | Updated 15 hours ago

Stephen Hawking is putting his "humans must leave Earth in 100 years" theory to the test. Josh King has the story (@abridgetoland). Buzz60



(Photo: Niklas Hallen/N, AFP/Getty Images)

6997 CONNECT 114 LINKEDIN 260 COMMENT EMAIL MORE

It's no secret that physicist Stephen Hawking thinks humans are running out of time on planet Earth.

In a new BBC documentary, Hawking will test his theory that humankind must colonize another planet or perish in the next 100 years. The documentary [Stephen Hawking: Expedition New Earth](#), will air this

POPULAR STORIES



Teachers are totally over fidget spinners

In the above example, the background is a very light gray, and the text is a very dark gray. The CSS values would be this:

```
body {
  color: rgb(50,50,50);
  background-color: rgb(250,250,250);
}
```

There is a reason they chose those colors: stark black on white can make people's eyes hurt, as the contrast is too high. Changing the colors to be slightly lighter than black and darker than white still has a black-on-white *appearance*, but is easier to look at. Few people notice it, however.

Animations should be the same way; they should be just enough to get the desired effect, but not so much that they distract people.

For example, when you scroll to the bottom of a page on some devices, and the scroll gets to the end, it may bounce back a little. That's an animation; in the past, the scroll just suddenly stopped. However, scrolls would sometimes stop in the middle of a page in some cases. So why make the bounce? Aside from adding an element of realism, the bounce animation tells you that it's indeed the end of the page. That's **functional**. But it is also so small and minor that most people don't even notice it when they see it. That's **subtle**.

The example on the previous page for the paragraph styling? That not functional *or* subtle! Avoid doing things like that.

Useful changes: text and background colors (subtle changes, please!) and shadows are a good start.

Another great use for the hover is in creating drop-down menus, which we will learn in another few chapters. Drop-down menus will be featured in Project #2.

PSEUDO-ELEMENTS

A similar structure is **pseudo-elements**, which are introduced in much the same way, except they typically use a double-colon before the element:

```
p::first-line
```

Pseudo-elements add effects in a variety of ways. Here are a few of them:

```
::before      Adds content (text, image) before the element, usually to the left of text
::after      Adds content (text, image) after the element, usually to the right of text
::first-letter Gives a separate style to only the first letter of text in an element
::first-line  Gives a separate style to only the first line of text in an element
::marker     Formats the bullet or other marker in an item list
::selection  sets the text & background colors when text is selected
```

Technically, you can only use one colon (:) instead of two (::) and it will probably work, but the standard is to use two colons to show the difference between a pseudo-class and a pseudo-element.

The `::before` and `::after` pseudo-elements are to add text or images before any kind of tag. For example, if you want to add a document icon before a link to

```
.doc::before {
    content: url(images/doc03.png);
}
```

The HTML would be:

```
<p class="intro">
  If you want information on LUJ facilities, please access <a class="doc" href="help.pdf">this document</a>
  for assistance. It may be able to help you with whatever problem you are having.
</p>
```

On the web page, it would look like:

If you want information on LUJ facilities, please access  [this document](#) for assistance. It may be able to help you with whatever problem you are having.

The `::first-letter` and `::first-line` do what they sound like. The `::first-letter`, for example, could create a Drop Cap:

```
p.dropcap::first-letter {
    font-family: 'Western', fantasy;
    font-size: 2.7em;
    font-style: bold;
    float: left;
    padding-right: 4px;
}
```

Yideo provides a powerful way to help you prove the embed code for the video you want to add, that best fits your document. To make your document footer, cover page, and text box designs that complete cover page, header, and sidebar.

7e. The Transition Animation

Okay, another cool effect! This is actually connected to the hover we learned about, and it allows you to make more subtle effects: the transition property.

If you apply the hover pseudo-class to the nav bar buttons we learned previously in this chapter, you can create a nice effect where the button text and background colors change if you hover the cursor over them. Here are the relevant parts:

```
nav ul li {
  background-color: white;
  box-shadow: 1px 1px 4px rgba(0,0,0,0.3);
}
nav ul li:hover {
  background-color: rgb(255,240,220);
  box-shadow: 2px 2px 4px rgba(0,0,0,0.5);
}
nav ul li a {
  color: rgb(50,30,0);
}
nav ul li a:hover {
  color: rgb(200,50,0);
}
```

The two hovers will change three styles: when you hover over the link button, it will get a light brown background color and a stronger text shadow, and the link text will turn red.



However, those changes come suddenly, flashing on and off as you hover over and then move away from the buttons. It's not really a *bad* effect—but it could be much better!

You can make it better with the transition property, which will animate the hover. More specifically, it will cause the hover effects to fade in and fade out:

```
nav ul li {
  background-color: white;
  box-shadow: 1px 1px 4px rgba(0,0,0,0.3);
  transition: 0.3s;
}
nav ul li:hover {
  background-color: rgb(255,240,220);
  box-shadow: 2px 2px 4px rgba(0,0,0,0.5);
}
nav ul li a {
  color: rgb(50,30,0);
  transition: 0.3s;
}
nav ul li a:hover {
  color: rgb(200,50,0);
}
```

The "0.3s" value means 0.3 seconds. That's rather quick, but works best with the button effect. Some hover transitions work best with longer fades or even shorter fades; it depends on what effect you are trying to create.

Important: if you want the start and end animation times to be the same, then the transition declaration goes in the *non-hover* rule. This allows the animation to fade in and fade out.

If you put the transition *only* in the hover rule, then the animation only happens when you hover—but when you move the cursor away, the change back happens suddenly.

You can add a transition declaration to *both* the normal and the hover, but this is only useful if you want to have different timings. For example make the start-hover longer, but the end-hover faster:

```
nav ul li {
  background-color: white;
  box-shadow: 1px 1px 4px rgba(0,0,0,0.3);
  transition: 0.3s;
}
nav ul li:hover {
  background-color: rgb(255,240,220);
  box-shadow: 2px 2px 4px rgba(0,0,0,0.5);
  transition: 1s;
}
```

In the above example, there will be different timings for the start- and end-hover animations.

Remember:

- A transition property in the main rule only will set the animation for both the start and end of the hover (hover-over and hover-out);
- A transition property in the hover rule only will result in animation only when you hover over, but there will be no animation when you hover out;
- Two transition properties, one each in the main rule and the hover rule will result in different times for over- and out-hovers. The hover-rule property will control the beginning (hover-over) animation, and the main-rule property will control the ending (hover-out) animation.

The transition effect can be fun and even elegant. However, it has a drawback: it would only animate when the user takes an action. There are times when you will want an animation to play by itself, on timing that you set, and without user intervention. We will learn about this in a later unit.

Chapter 7 Checklist

- If you set the color of text for the same selector in a stylesheet, embedded CSS, and inline CSS, which one will the web page display in the browser?
- How can you override the cascade order for any one declaration?
- Which has the stronger position: class or id?
- If you have two rules for the h1 font size, which one will "win"?
- What does the !important value do?
- What is the rule for links to the index page?
- What are the rules for links to the 2nd and 3rd level pages?
- What are *hiragana*, *lower-latin*, and *circle* examples of?
- What CSS property and value do you use to change an inline tag into a block tag?
- What does "inline-block" mean?
- How do you change tags from block to inline and inline to block?
- What are the different values for a border property?
- What are the four normal values for a box shadow?
- What property allows you to round the edges of a rectangle frame?
- What does the "hover" pseudo-class do?
- Can you make more than one property change in a single hover rule?
- What is a "functional" design element?
- What will the property transition do?
- How will the animation be different if the transition is in the main rule or the hover rule?

Chapter 8: Boxes and Backgrounds

PROJECTS

8a. Project #1

By now, you have learned everything that you will need to create your Project #1. I would like to describe that project in detail and tell you what the steps will be in creating it.

The basic elements of project #1 are:

1. Create a 2-level web site with a minimum of 4 pages with jello layout
2. There must be one index page and at least three 2nd-level pages
3. You must choose a topic which is close to you physically
4. You can only use photos which you take and edit yourself
5. There must be a link to the index page in the main heading
6. There must be a menu bar for all 2nd-level pages
7. The header and nav (probably also the footer) must be the same on all pages
8. You must choose a color scheme and specific Google Fonts

The steps, in order, are:

Prep: Create a site map and wireframe

1. Create a single, styled template HTML page with a CSS stylesheet.
2. Copy the template and make a second page with an alternate style
3. Fill in real-world text and images

Let's go over each step.

PREP: PLANNING: SITE MAP AND WIREFRAME

Your first step requires you to choose a topic. I want to limit your topic choice to something nearby you, because you will be using only photographs that you took yourself. I do not want to make you travel far and wide to take photos.

If you are staying at home all the time, the photos should be of something at home or close by. For example, a web site about a pet, your artwork, local parks, local restaurants or shopping, or anything else you can take photos of. If you already have photos (a vacation, family history) you can use those. If you travel to work or school, the topic could be about anything in that general area. For example, if you have a job in Shinjuku, then you can make it about that area.

Use PowerPoint or Keynote to create a **site map** showing the main topic in the top (index) box, and the examples or other specifics in the three 2nd-level boxes.

Then make **wireframes** for the 1st and 2nd levels.

The wireframes for Project #1 will be very simple: you will have a header, nav, main, and footer. All will be separate, from top to bottom, as exemplified on the previous page.

The changes between the first and second level will be in (1) colors and (2) image size and placement *only*. You must not change the header or nav, and probably not the footer.

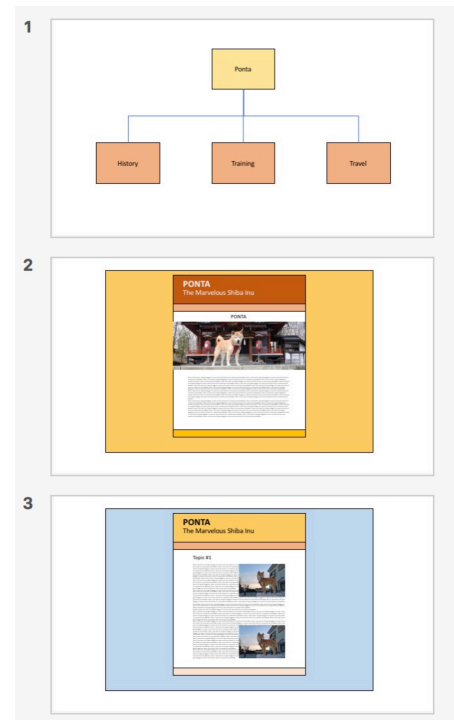
You *can* change the background color, background image (later in this chapter), and text color (especially for headings). Your color changes must be part of your overall site **color scheme** (see Chapter 5g). Do not use colors or images that go against your overall site theme.

You can also change image sizes and layout—have images of different sizes in different parts of the page. For example, the index page might have a larger **banner** image as I discussed in the chapter on images, while smaller images will appear on other pages. In the next chapter, we will learn how to make images go to the left or right side of a page and have the text flow around the image on the other side (this is called a "float"); you can include this in your Project because we will learn it while you are still working on your Project #1.

Choose your fonts carefully. You will not be allowed to use any "Web-Safe" fonts (see Chapter 6h), and instead must use only Google Fonts. You can use between one and four different fonts. Two is a normal minimum, three is a normal maximum. Only one font is possible, but could make the page boring. Four fonts is possible, but if there are too many fonts which are all too different, the design becomes rather confused.

Do not just choose the first fonts that you see which you think are cool. Find fonts that go well with your topic. If your topic is a vacation, a cursive font might be good for the main site heading, while a serif font might work better for the paragraph text. If your site is about technology, a futuristic or mechanical-looking display font might be better for the heading, with a sans-serif for paragraph text. And so on.

After deciding on a topic, you should immediately begin to get photos to use. You must also begin to write or collect text for the pages (each page should have several paragraphs at



minimum). The photos must be yours; for the text, you can write it yourself, or else get it from another source. If you get it from a source, you must give a citation for it.

Send me the file with the map and wireframes. Do not keep it in the project folder.

1. MAKING A TEMPLATE

Now you have your full plan. You have started to collect photos and text. You have looked through Google Fonts and have decided on your font set; you should have already gotten the **@import** code, as well as the declarations for the font families. You know what your pages will look like.

Now you must begin to build a page. The first step is to **create a project folder**, using the name `myname-project1-1`. Your first name only should be used; the final two numbers are the project number and the step number. The step number will change for each step.

Create a folder inside the project folder and call it `images`. All of your images must go in that folder. **Only** put in images that have been processed and are the correct size. Do not put unedited photos in that folder! Keep them in a work folder somewhere else.

Name your HTML file `template.html`.

Create that HTML code and create a CSS file with all the styling rules in it. Save these in the project folder, but not inside the `images` folder.

This will actually be somewhat simple. Project #1 is mostly just the code we've been learning and practicing. Make a DOCTYPE, an `<html>` tag, the head and the body, a wrapper div (use a jello layout), and the four structural parts (header, nav, main, and footer). We learned how to make the header and nav. The main will be an `<h3>` tag followed by text and images.

Note: when you create the nav menus, **you must choose the filenames of all the files that will be created later**. You must decide the **actual** filenames at this time, even though the files don't exist yet:

```
<nav>
  <ul>
    <li><a href="german.html">My Family from Germany</a></li>
    <li><a href="spanish.html"> My Family from Spain</a></li>
    <li><a href="american.html"> My Family in America</a></li>
  </ul>
</nav>
```

In the final step of your projects, you will have to use these filenames to name the HTML pages you will create, so make sure your names are correct!

When you make the page, you must have **full text of some kind** in the pages. If you do not have your final text ready, you may use **Lorem Ipsum** text. This is Latin text (written by Cicero) which is not intended to be read or understood. It is just sample text, so you can see how your paragraph font looks on the page as part of the full design. You can get Lorem Ipsum text on the Internet (such as <https://loremipsum.io/generator/?n=5&t=p>).

Any text which you copy from anywhere should have a link back to the origin. At the end of each segment of copied text, add [[Source](#)] after the end of the last sentence, and make the word "Source" into a link to the web page that you got it from.

CHAPTER 8

You should also have example images of the correct shapes and sizes that you decided. The template should have at least one image of each size. (you will probably have only two sizes: the main page image (perhaps a banner) and a smaller image that will go in the page text.

When you finish, you should have a single fully-designed web page **which has all the design elements for all the pages in your site**. Later, you will remove parts that specific pages don't need. **Do not make named copies of the page to test the links!** You will test the links in the final step. Adding extra files now will only confuse things. Just check the nav code carefully.

After you complete this page, you must send it to me. Zip the folder (see the "Getting Ready" section, Part V, on "Sending Files") and attach it to an email to me. **In the email text, make sure to point out any difficulties you had, any code you need help with, and any questions you might have about the page.** The more questions you ask, the more I can help you, and the better your grade will be.

I will send you back notes, comments, corrections, and suggestions. You will probably have to edit and make adjustments at least a few times, each time sending me the new version of your site. When I tell you that your code is OK, then you may start on the next step.

2. MAKING THE OTHER LEVEL TEMPLATE

After I review the project Step #1 and return it to you with notes, please do the following:

1. Fix all the errors I point out
2. Try to add new improvements, styles, special effects, and so on
3. If you want to make changes to the site's layout/styles, now is the best time
4. Look for any new or other errors that you might have, and fix them
5. Save all of your files and close them

Now you have a single web page and a stylesheet that are finished and corrected. Remember, your site has two levels. You must now start with your original design, and from it, create another page with a slightly changed design.

First, make a copy of your whole, finished first step of the project. Duplicate the folder. Change the name of the new folder to `myname-project-1.2`. Leave the first step's files alone, and only work on the new step.

Open the new Step 2 folder, take the `template.html` file, and make a copy of it so there are two of them in the folder.

Name one of the pages **`index.html`**, and the other one **`second.html`**.

Go into the HTML code for both pages and make changes so that one of them has the exact parts that it needs, and not any parts that should only be on the other page. One of them must be styled as the index (see your wireframes), and the other should be the second level page (again, it should match the wireframes you made).

If you have any completed text or images that you can add, add them now.

Again, link back to any site where you got text from. At the end of each segment of copied text, add [[Source](#)] after the end of the last sentence, and make the word "Source" into a link to the web page that you got it from. There should be a link after each borrowed segment on each page you have borrowed text.

Check the site once again for any possible errors.

Then, send me an email with the second step zipped and attached, and **again, add notes and questions about anything you need help with. Do not just write "Check my page please."** Be as specific as you can in your requests.

Again, I will review the project, and send notes, comments, corrections, and suggestions. You will probably have to edit and make adjustments at least a few times, each time sending me the new version of your site. When I tell you that your code is OK, then you may start on the final step.

3. FINALIZING THE PROJECT

After I review the project Step #2 and return it to you with notes, again do the following:

1. Fix all the errors I point out
2. Try to add new improvements, styles, special effects, and so on
3. Look for any new or other errors that you might have, and fix them
4. Save all of your files and close them

Now you have two different web pages, one for each for your project, that are finished and corrected. Now you will complete the site.

Again, make a copy of your whole, finished second step of the project. Duplicate the folder. Change the name of the new folder to myname-project-1.3. Leave the first step's files alone, and only work on the new step.

The index is fine and should be finished.

However, you only have one page for all the second level pages. **Make two extra copies of second.html**, so you now have three copies of that same page.

Give each copy a name that you decided in the nav menu links from Step #1 (in my example, I used the names "germany.html", "spanish.html", and "american.html").

Now open the index page in a browser, and check all the links. Make sure each of the four pages can successfully link to all three other pages. If there are any problems, fix them.

Make sure that **all the text** is real text, and not Lorem Ipsum or any other filler text. Any borrowed text must have the [[Source](#)] link that I described earlier.

Make sure that each page has at least two different images. Do not repeat images on different pages. If the index page has one large or banner image, then a second image is not necessary.

Do a final check to make sure the project works well.

Once again, send me an email with the second step zipped and attached, and **again, add notes and questions about anything you need help with. Do not just write "Check my page please."** Be as specific as you can in your requests.

If there are any remaining problems, I will send you an email asking you to fix them. Otherwise, I will accept the project as final.

Project #1 is due, **at latest**, the Friday before the midterm grades are due (check your calendar).

After I accept the final project, I will give it a grade and incorporate the grade into your Midterm Course Grade. I will not make any further notes or comments, having already done so many times.

That will be the end of the project.

CODE

8b. Div and Span

Most tags have a built-in **CSS style**. For example, the `<h1>` tag has these styles as default:

```
display: block;
font-size: 2em;
margin-top: 0.67em;
margin-bottom: 0.67em;
margin-left: 0;
margin-right: 0;
font-weight: bold;
```

In short, the `<h1>` tag is a block element; has a font size 2x the normal font size; has a top and bottom margin 67% of the `<h1>` font size; has no right or left margins; and has a bold weight. Most HTML tags are like this: they have styles built in.

Furthermore, many HTML tags are **semantic**: they are only used for very specific purposes. The `<header>` tag is only used for the site headings; the `<h1>` tag is used for the title; the `<nav>` tag is used only for the site's links; the `<p>` tag is only used for paragraph text; and so on.

So, Why DIV and SPAN?

The `<div>` and `` tags are **blanks**. That is, they have no styles and no semantic meanings at all. The only definition that `<div>` has is that it is a block element (`display: block;`). The `` tag has no definitions at all. Neither has any colors, sizes, margins, padding, borders or any other styles specified.

As a result, these tags, by themselves, do nothing. Make a `<div>`, and all you get is a featureless box. Add a `` tag around text, and nothing at all happens.

So, what are they for? They are used for anything that is not already specified.

For example, there is the `` tag to make text bold, and the `<i>` tag to make text italic, but there is no tag to make text a certain color. By using the `` tag with a CSS class rule, you can effectively make up your own tag.

For example, let's say that you want to have a *small amount* of text in a paragraph to be red. How do you do that? You cannot give the `red-text` style to the paragraph tag, or else all the text would be red. You need an inline tag—but which one should you use? The `` tag makes text bold and has semantic meaning which you don't need. The `` tag is the same, except for being italic instead of bold. But you don't want bold or italic—just red. So, what do you do?

Well, you create a class and use the `` tag:

```
.redtext {
  color: rgb(255,0,0);
}
```

```
<p>
  The expression <span class="redtext">red letter day</span> means a
  memorable day.
</p>
```


CHAPTER 8

Additionally, because a class can contain many styles, you can create a span that can affect many styles at once—for example, a class which makes text red, bold, a different size, a different font, etc.—all in the one class reference.

The same can be done with the `<div>` tag, if you want a box of some sort which has special styles. Take, for example, a "pull quote." A pull quote is when a quote from a magazine or web page article is displayed much larger than the normal text.

scaevola voluptaria, sed liber dicit ea. Per idque theophrastus an, etiam democritum cu sea. Oblique ancillae accusata sit ea. Numquam ponderum invidunt cu qui, eam homero accusam quaerendum eu. Alterum antiopam id nam, in usu vivendum postulant.

Cibo pericula molestiae quo an. Ex sea dico delenit atomorum, ei vocibus patrioque mel. Invidunt accommodare eam ad. An vim agam conclusionemque, intellegat temporibus pro in. Sea aequae assentior voluptatum an, cu pertinax comprehensam per, quo error quando expetenda cu. Et duo aperiam legendos, ea nullam malorum eripuit vix, diceret gloriatur vis ad.

Shinjuku Gyoen Park has cherry blossom displays in early April that you should not miss. Dozens of different types of trees create a dazzling display!

Lorem ipsum dolor sit amet, pri debet mucius erroribus in, nec alterum scripserit ea, at sit eruditi labores. His fugit semper admodum id, at est clita reprimique. Ne antiopam definiebas mei. Mediocre pertincia ut eam, ea illud homero principes sit. In sit viderer delenit, in aperiam repudiare nam, habeo labitur eu vis. Illud dicant ex mel, inani legimus ea ius.

Utamur oblique dissentiet cu vim, his an dui possit accusata. Quem labores ad qui, aliquam mentitum sit ei. No ius solum scaevola voluptaria, sed liber dicit ea. Per idque theophrastus an, etiam democritum cu sea. Oblique ancillae accusata sit ea. Numquam ponderum invidunt cu qui. eam homero accusam quaerendum eu. Alterum antiopam id nam. in usu vivendum

What tag would you use to create this? A pull quote is a quotation, but the `<blockquote>` tag only gives us a box with a set margin and padding. That's not enough for what we want. And although we could start with a blockquote and then change the styles, it is easier to begin with a box that has no styles set at all.

Therefore, to create a special box like a pull quote, we would simply use a `<div>` tag—a blank block tag to which we can easily apply the desired style:

```
.pullquote {
  color: rgb(150,20,0);
  font-size: 20pt;
  font-family: 'Garamond', 'Georgia', serif;
  font-style: italic;
  text-indent: 0.5in;
  border-top: 1px solid black;
  border-bottom: 1px solid black;
  padding: 24px 38px;
}

<div class="pullquote">
  Shinjuku Gyoen Park has cherry blossom displays in early April that
  you should not miss. Dozens of different types of trees create a
  dazzling display!
</div>
```

In short, use the `<div>` and `` tags when you need to define a text or block style when there is no existing tag which does that. In HTML6, these tags may be replaced by "express" tags where you can define new tags by yourself, but for now, use the `<div>` and `` tags with classes.

8c. Using Boxes

NORMAL FLOW

All content on a web page begins at the top left. Content moves first from left to right, and then from top to bottom. When content, moving from left to right, reaches the right edge, it moves to the next available space below. If you are near the end of a line and the next object is too big to fit, it jumps to the next available space below.

This is called **normal flow**. You see it all the time when you type something.

The left-to-right movement is within the **available space**. This is sometimes from the left edge of the browser window to the right edge, but usually it is within a smaller enclosed space. For example, if you use a Jell-O layout (with adjustable margins), then the available space is set by the wrapper. Your content may be in a smaller box, such as an unordered-list button in the <nav> area, and that might only be 200px wide, for example.

You can cause small **breaks** in normal flow with a
 tag, or a **block** element.

A
 tag will simply stop anywhere on a line and move down to the next line (the next available space below).

A block tag will stop the flow from left to right, jump down to the next available space below, create a rectangle, and then create another break below the rectangle.

However, it is sometimes necessary to break the normal flow in other ways. This is usually done with **floats** or **positioning**. Floats are described below. Positioning will be discussed later.

THE BOX MODEL

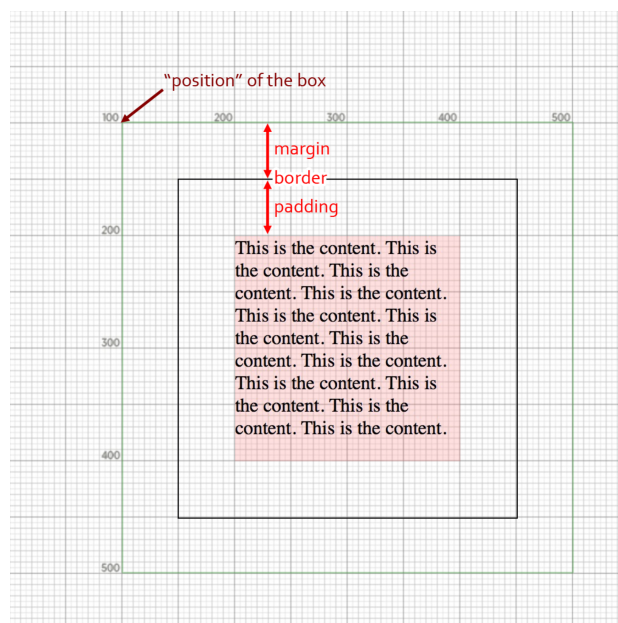
Web pages are made up of rectangles—call them "boxes." The window showing the page is a box.

The wrapper is a box. The structural elements are boxes. The block tags are boxes. As a result, the shape of a box is fundamentally basic for a web page, and therefore also for its design.

Each of these boxes has a **position**. The position can be defined as a single point, a certain number of pixels from the top and the left sides of the page. For example, in the image at right, the position of the <div> tag is 100 pixels from the top, and 100 pixels from the left.

In the example image at right, the pinkish area with text is the **content**. When you set the "width" of the <div> in the CSS, this is the area that is decided. If there is no margin, border, or padding, the content area would begin at the starting position (in this case, 100, 100).

```
#box1 {
  border: 1px solid black;
  margin: 50px;
  padding: 50px;
}
```



Adding a margin, border, and padding will **add** to the size of the area taken. The content box will remain the same size, and the extra margin, border, and padding space will increase the space taken up.

When you add a **margin**, the margin begins at the starting point (100, 100), and "pushes" the content to the right and down. In the example above, the margin is 50px. This pushes the content box 50px down, and 50px to the right. An additional margin appears on the other sides of the box, surrounding it.

When you add a **border**, it appears inside the margin, and outside the content box. It also increases the width, but usually just by a few pixels, so it is hardly noticeable. However, *it is there*, and it does add to the width and height, and this could cause problems if you calculate sizes exactly but forget that the border is adding to it.

When you add **padding**, it appears inside the border and margin, and again pushes the content box to the right and down.

So, with only the content box, if the width is set to 200px and it begins 100px from the left, the box will end sharply at the 300px mark. Only 200px will be taken up.

However, if you add a 50px margin, a 1px border, and 50px of padding, then you have added **101px** to every side of the box, and an additional **202px** to the entire width and height. Therefore, if the 200px-wide box starts at 100px from the left, it will end at the **502px** mark.

8d. Floats & Clears

The most common need to break normal flow is to add an image. Until now, our images have either been inline (with one line of text to the left and/or right), or they have been on their own line, with nothing to the left or right. By now, you probably really want to have them sitting on the left or right, and have the text flow around them.

This is done with floats. An inline example:

```

```

This will make the image jump to the right. Its position top-to-bottom will remain the same; it's position relative to the top of the page will be the same as it would be in normal flow. However, the left-to-right position will be disconnected from normal flow.

An important point: inline content, such as text, will flow *around* the floated object. However, **block elements will not!** This can be confusing and surprising, but it is necessary.

For example (see illustration on the next page), let's say that you have a paragraph of text, and you float an image to the right of the text. The paragraph is a block, and must always be in the shape of a rectangle. If the paragraph box "avoids" the image, then the text in the paragraph will stay to the left of the image, but it will never go *under* the image. Therefore, the paragraph box "slides under" the image, but the paragraph text wraps *around* the image.

CHAPTER 8

Here is how it works. Let's begin with simple paragraph text. Here is what four paragraphs look like with red borders on the <p> tag:

The red lines are <p> tags

One morning, when Gregor Samsa woke from troubled dreams, he found himself transformed in his bed into a horrible vermin. He lay on his armor-like back, and if he lifted his head a little he could see his brown belly, slightly domed and divided by arches into stiff sections. The bedding was hardly able to cover it and seemed ready to slide off any moment.

His many legs, pitifully thin compared with the size of the rest of him, waved about helplessly as he looked. "What's happened to me?" he thought. It wasn't a dream. His room, a proper human room although a little too small, lay peacefully between its four familiar walls.

A collection of textile samples lay spread out on the table - Samsa was a travelling salesman - and above it there hung a picture that he had recently cut out of an illustrated magazine and housed in a nice, gilded frame. It showed a lady fitted out with a fur hat and fur boa who sat upright, raising a heavy fur muff that covered the whole of her lower arm towards the viewer.

Gregor then turned to look out the window at the dull weather. Drops of rain could be heard hitting the pane, which made him feel quite sad. "How about if I sleep a little bit longer and forget all this nonsense", he thought, but that was something he was unable to do because he was used to sleeping on his right, and in his present state couldn't get into that position. However hard he threw himself onto his right, he always rolled back to where he was. He must have tried it a hundred times, shut his eyes so that he wouldn't have to look at the floundering legs, and only stopped when it was done.

Next, if there is an image between the 1st and 2nd paragraph, it looks like this:

One morning, when Gregor Samsa woke from troubled dreams, he found himself transformed in his bed into a horrible vermin. He lay on his armor-like back, and if he lifted his head a little he could see his brown belly, slightly domed and divided by arches into stiff sections. The bedding was hardly able to cover it and seemed ready to slide off any moment.



His many legs, pitifully thin compared with the size of the rest of him, waved about helplessly as he looked. "What's happened to me?" he thought. It wasn't a dream. His room, a proper human room although a little too small, lay peacefully between its four familiar walls.

A collection of textile samples lay spread out on the table - Samsa was a travelling salesman - and above it there hung a picture that he had recently cut out of an illustrated magazine and housed in a nice, gilded frame. It showed a lady fitted out with a fur hat and fur boa who sat upright, raising a heavy fur muff that covered the whole of her lower arm towards the viewer.

Gregor then turned to look out the window at the dull weather. Drops of rain could be heard hitting the pane, which made him feel quite sad. "How about if I sleep a little bit longer and forget all this nonsense", he thought, but that was something he was unable to do because he was used to sleeping on his right, and in his present state couldn't get into that position. However hard he threw himself onto his right, he always rolled back to where he was. He must have tried it a hundred times, shut his eyes so that he wouldn't have to look at the floundering legs, and only stopped when it was done.

Notice that the breaks below the 1st paragraph and above the 2nd paragraph put the image on its own line, between the block paragraphs.

Next, we will **float** the image:

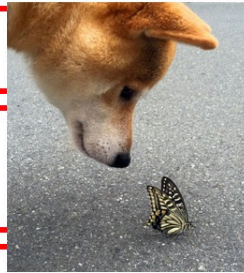
CHAPTER 8

One morning, when Gregor Samsa woke from troubled dreams, he found himself transformed in his bed into a horrible vermin. He lay on his armor-like back, and if he lifted his head a little he could see his brown belly, slightly domed and divided by arches into stiff sections. The bedding was hardly able to cover it and seemed ready to slide off any moment.

His many legs, pitifully thin compared with the size of the rest of him, waved about helplessly as he looked. "What's happened to me?" he thought. It wasn't a dream. His room, a proper human room although a little too small, lay peacefully between its four familiar walls.

A collection of textile samples lay spread out on the table - Samsa was a travelling salesman - and above it there hung a picture that he had recently cut out of an illustrated magazine and housed in a nice, gilded frame. It showed a lady fitted out with a fur hat and fur boa who sat upright, raising a heavy fur muff that covered the whole of her lower arm towards the viewer.

Gregor then turned to look out the window at the dull weather. Drops of rain could be heard hitting the pane, which made him feel quite sad. "How about if I sleep a little bit longer and forget all this nonsense", he thought, but that was something he was unable to do because he was used to sleeping on his right, and in his present state couldn't get into that position. However hard he threw himself onto his right, he always rolled back to where he was. He must have tried it a hundred times, shut his eyes so that he wouldn't have to look at the floundering legs, and only stopped when it was done.

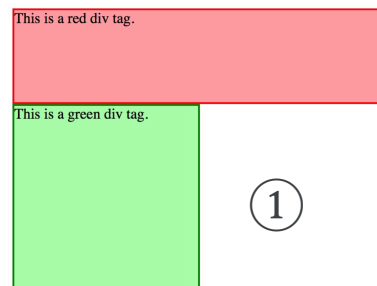


Notice that the block (p tag) and inline (text inside the p tag) act differently.

The block paragraph tags ignore the image, and go below it. However, the text *inside* the paragraphs avoids the image and wraps around it.

Therefore, you have to remember: a float will "push aside" text and images (inline content), but it will **not** push aside blocks! To illustrate this better, let's take a look at two <div> tags and see how floats affect them.

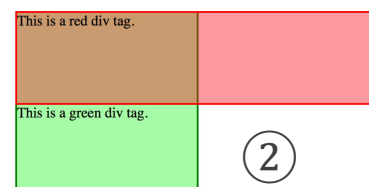
① You can see two <div> tags at right. Each has a border and background color (partially transparent). Each has a set width and height. Each has text inside.



As you see them here, they act as normal blocks: each has its own line, one above the other.

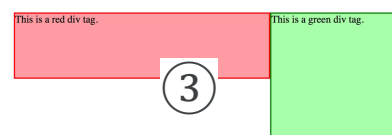
② In the next illustration, we will **left-float the red div, but not float the green div.**

Notice that the green div "slides under" the red div, but the text in the green div avoids the red div. This is the principle described above: a floated object will wrap text but it will be "invisible" to blocks.



Maybe you don't want that. Let's say that you want the green div to also avoid the red div. You want the green div to wrap around the red div, to the right.

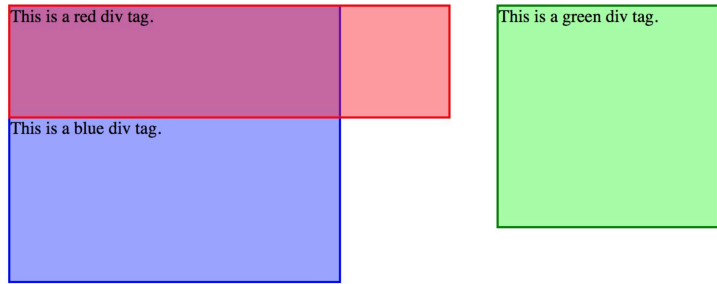
③ One way to do this is to **give the green div a left-float also.** *Two* floated blocks will always push each other away. They will not "slide under" each other. If there is enough space, the green div will sit just to the right of the red div. If there is not enough available space, the green div will jump down *below* the red div.



You can also right-float the green div, except that the green div will always align on the right side of the available space. This may create a blank space between the red and green divs.

CLEARs

Remaining blocks which do not have floats, however, will still "slide under" the floating objects. For example, here is an image of the red and green <div> tags, both floating, and then a blue div, which does **not** float.



The red div is **floating left**, and the green div is **floating right**.

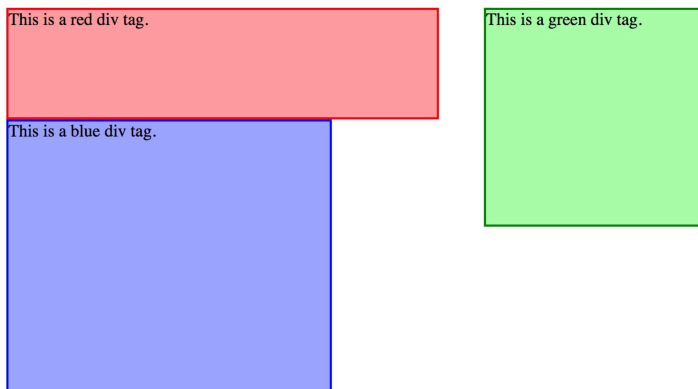
Notice that the blue div, which has no float, slides under the red. It would slide under the green also if it was wide enough.

So, how do you get the blue div to avoid the floating div? We learned that making the div float also could fix the problem.

A cleaner way, however, is to **clear** the float. In the style for the blue div, you would add this CSS declaration:

```
clear: left;
```

That will create what you see at right: the blue div will jump below the last object which floats left. Notice it is below the red div, but not below the green div.



If you added this declaration:

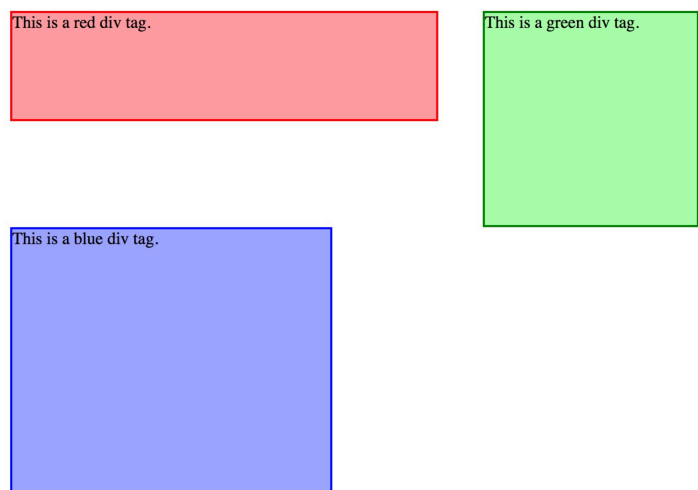
```
clear: right;
```

You would see what is show to the right: the blue div will jump below the last object which is right-floating—in this case, it is the green div.

You could also type:

```
clear: both;
```

And that would jump an object below all floating objects above it.



8e. More Fun with Effects

MULTIPLE SHADOWS

We have already covered box and text shadows, but there is an amazing wealth of special effects that you can do using these shadows in CSS. The trick lies in using **multiple shadows** in a single declaration. For example, here is a bit of text in *Raleway* font:

This Is Writing Plus Shadows

Here is the same sentence with the following shadow:

```
text-shadow: 2px 2px 0px rgb(150,150,150);
```

This Is Writing Plus Shadows

You have seen that before: the values are **offset-right**, **offset-down**, **blur**, and **color**.

Now, here's the secret: by adding a comma and a space before the semicolon, you can add another shadow *on top of that!* Actually, the first shadow is on top, and the last shadow is on the bottom. The code looks like this, followed by the shadow effect:

```
text-shadow: 1px 1px 0px rgb(255,255,255), 2px 2px 0px rgb(150,150,150);
```

This Is Writing Plus Shadows

Cool, right? To make the shadow code more clear, we can stack each shadow code on a new line:

```
text-shadow: 1px 1px 0px rgb(230,230,230),
            2px 2px 0px rgb(190,190,190),
            3px 3px 0px rgb(150,150,150),
            4px 4px 0px rgb(100,100,100);
```

This Is Writing Plus Shadows

You can add as many shadows as you have the patience to create. You can also use whatever colors you want:

```
text-shadow: 1px 1px 0px rgb(255,0,0),
            2px 2px 0px rgb(255,125,0),
            3px 3px 0px rgb(255,255,0),
            4px 4px 0px rgb(125,255,0),
            5px 5px 0px rgb(0,255,0),
            6px 6px 0px rgb(0,255,125),
            7px 7px 0px rgb(0,255,255),
            8px 8px 0px rgb(0,125,255),
            9px 9px 0px rgb(0,0,255);
```

This Is Writing Plus Shadows

You can also use color shadows to create a glowing effect:

```
text-shadow: 0px 0px 3px rgb(50,0,255),
             0px 0px 6px rgb(150,100,255),
             0px 0px 10px rgb(200,150,255),
             0px 0px 15px rgb(220,180,255),
             0px 0px 20px rgb(230,200,255),
             0px 0px 30px rgb(240,220,255),
             0px 0px 40px rgb(250,240,255),
             0px 0px 50px rgb(255,250,255),
             0px 0px 60px rgb(255,255,255);
```



Here are some cool shadows from web sites. These are from:

<https://designshack.net/articles/css/12-fun-css-text-shadows-you-can-copy-and-paste/>

```
text-shadow: 0px 3px 0px rgb(178,169,143),
             0px 14px 10px rgba(0,0,0,0.15),
             0px 24px 2px rgba(0,0,0,0.1),
             0px 34px 30px rgba(0,0,0,0.1);
```



```
text-shadow: 0 1px 0 rgb(204,204,204),
             0 2px 0 rgb(201,201,201),
             0 3px 0 rgb(187,187,187),
             0 4px 0 rgb(185,185,185),
             0 5px 0 rgb(170,170,170),
             0 6px 1px rgba(0,0,0,.1),
             0 0 5px rgba(0,0,0,.1),
             0 1px 3px rgba(0,0,0,.3),
             0 3px 5px rgba(0,0,0,.2),
             0 5px 10px rgba(0,0,0,.25),
             0 10px 10px rgba(0,0,0,.2),
             0 20px 20px rgba(0,0,0,.15);
```




You can have similar effects with box shadows; you can have multiple shadows with those as well, and create interesting effects. Just do some searches on the web for some cool text and box shadows, you'll find lots of examples.

However, I must **caution** you about shadows! Like so many other "cool" effects that you discover and want to use like crazy, *these are only for very special situations!* If you use shadows like this for no special reason, and especially if you use more than one, they will begin to stand out too much, and will distract the reader from your page!

Use these shadows **sparingly** (very seldom), and only when you have a good **design-based reason** for doing so.

FONT OUTLINES

It used to be that if you want a font to have an outline, you had to make four multiple shadows around the text, one for each side. However, the effect was not perfect. Fortunately, there is a new feature in HTML: **text-stroke**.

Because this is still an experimental feature, we have to add what is called a **vendor prefix** before the CSS property. In most cases, that vendor is **webkit**, with a hyphen before & after:

```
h1 {
  font-family: "West", display;
  font-size: 42pt;
  -webkit-text-stroke: 1px red;
}
h2 {
  font-family: "Disney";
  font-size: 28pt;
  color: rgb(100,200,255);
  -webkit-text-stroke: 1px blue;
}
```



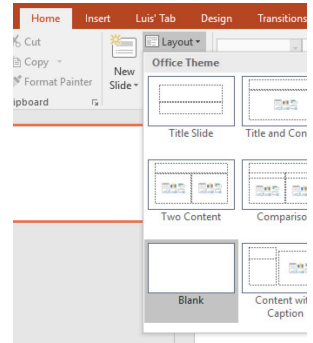
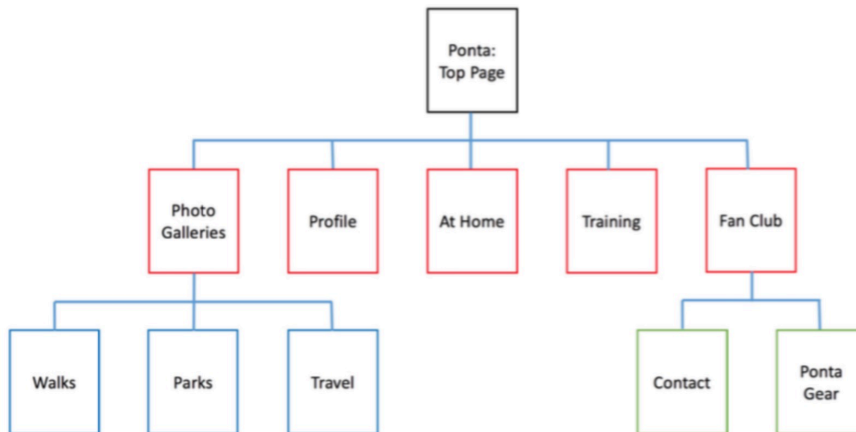
If the stroke (outline) is made with a larger width, the width expands both inward and outward. Unless the text is very thick, wider strokes will look strange.

DESIGN

8e. Site Planning: Site Maps & Wireframes

SITE MAPPING

When you create a larger web site, you should not begin with a wireframe. Instead, you should begin with a **site map** (an illustration which shows all pages in a site, and their organization), and then make wireframes. We will talk about maps later in the class, but you can see an example of a site map below:



If you want to create a site map, a good application to use is PowerPoint. Create a new, blank presentation. Then, in the Home tab, click on the Layout button, and choose a Blank layout.

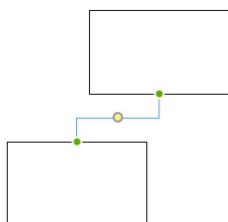
Also in the Home tab, or in the Insert tab, you can find a menu for Shapes. Choose a rectangle shape to create the boxes to show each page.

The tricky part is to connect the boxes together. Here you need to use a special shape called the "Elbow connector."

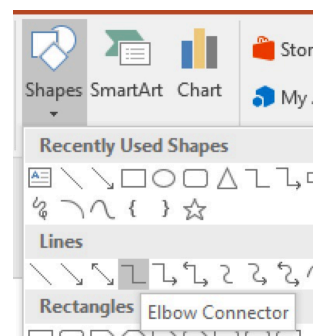
The Elbow Connector allows you to connect one side of a rectangle with one side of a different rectangle.

Select the Elbow Connector shape, then hold the cursor over one of the rectangles. You will see four dots appear on the rectangle, one for each side. Click the dot you want to connect first, then drag the cursor to the desired dot on the other rectangle.

This allows you to create the connecting lines.



Creating a site map is similar to writing an outline for an essay. You can better see the shape of the web site and how the information is organized. It also allows you to discover what information is and is not needed for the web site.



A site map is not a final plan, but rather is part of the creative process. Use the site map for planning, but allow yourself to change the plans.

CHAPTER 8

An **important point** is to think about what will go on **every** web page in your site. Each web page should be **full** of information, at least several paragraphs and probably some images as well. When you plan a page, you must think about what you will put on the page. If you can't think of good content to put there, then maybe the page has no meaning!

LEVELS

If you look at the site map on the previous page, you will notice that there are **three levels**.

1. the **first level** is the main page;
2. the **second level** is the category pages;
3. the **third level** is the detail pages.

It is similar to an essay. The main page is like the introduction; the second level pages are like the topic sentences and major details; and the third level is like the minor details, or examples and explanations.

In other words, the deeper you go into an essay, the more specific the details become. The same is true on a web site:

1. the **first level** has an overview of the whole site subject;
2. the **second level** introduces categories or divisions in the subject, and each page gives an overview of the category.
3. the **third level** has the most specific details and information.

It is not always necessary to have three levels. You can have one, two, four, or any number that works best for the content of your web site. However, in this class, project #1 must have two levels, and project #2 must have three levels.

For the next part, on wireframes, keep in mind that **each level must have a different design**. As I mentioned before, the size and content of the header, nav, and footer must remain the same. Additionally, you should not change fonts or font sizes from level to level

However, on each level you can have a different number of columns, different colors, backgrounds, image sizes, etc.

The changes should not be so drastic that your site looks completely different; often the changes are subtle—but the levels are all different. Again, go to your various web sites and go to the second and third levels and so on; you will notice different layouts.

WIREFRAMES

Web pages have structure. Sometimes the content (text, images, etc.) takes up the whole page. More often, the content is in the middle (inside a "wrapper"), with blank margins on the sides.

There are several parts to the content on a page:

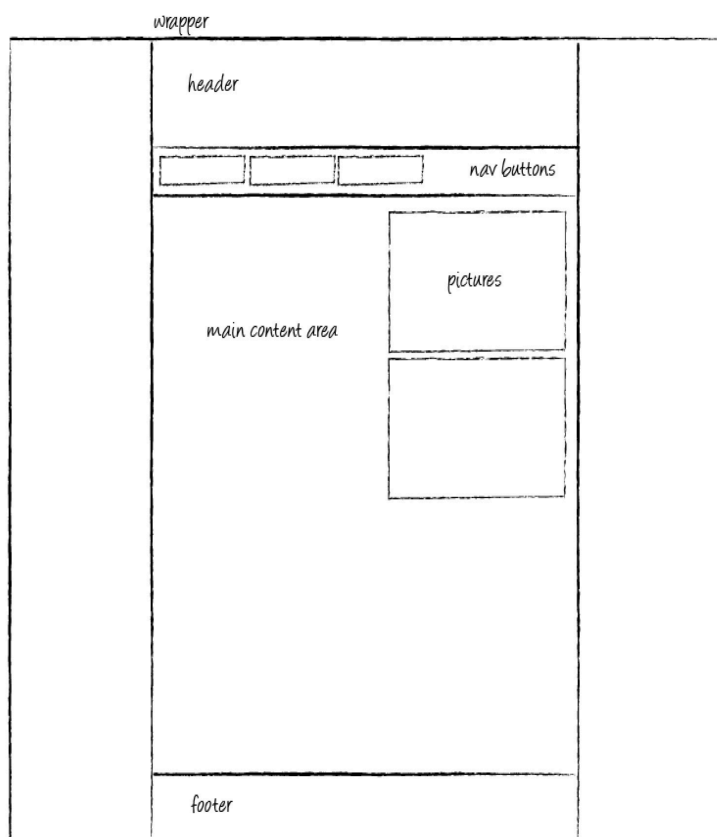
- **header** (main title)
- **nav** (links to other pages in the site)
- **main** (where the main text and images are)
- **footer** (the copyright and other descriptive information about a page)

When you plan a web site, you should always begin by drawing a diagram, or "**wireframe**" of what you want the page to look like. This helps you to write the code.

The word "wireframe" means that you only have to draw boxes to show the structure and content for the planned web page.

In the example at right, you can see a simple wireframe for a web page. A more complete wireframe would include notes about sizes, margins and padding, colors, and font styling. You could use PowerPoint to make your wireframes, but pencil on paper is also okay.

A hand-drawn wireframe might look like this:



CHAPTER 8

However, it might be easier to use PowerPoint again. At right is an example of three wireframes I made, each for a different level of my site.

Notice that I used only two images and repeated them; if you could see the text, you would see that it is meaningless "Lorem Ipsum" text.

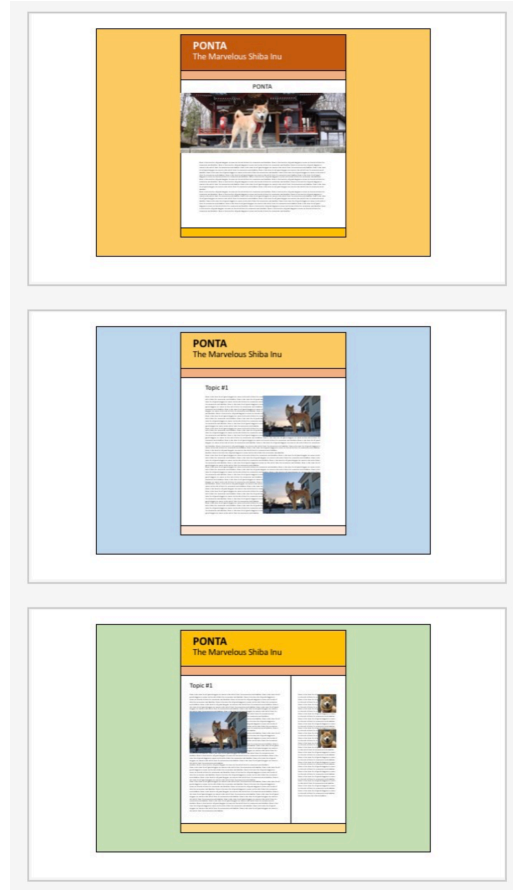
Lorem Ipsum is Latin text (from a writing by Cicero) which is used in the design industry when a designer wants to see how text will look on a page, without having to use the actual text. (Someone asking you to design a site rarely has the text ready at the beginning!).

In the example shown, you can see that I varied the background colors of the page, and even used different background and font colors on the top page.

The top page has a large banner image; larger images which establish the site's topic and style are very common on the main page that everyone visits first.

My other pages have medium-sized images which are located on the right side (on level 2) or on the left side (level 3). Level 3 has two columns, with smaller square-shaped images in it.

Using this plan, I can then begin preparing and then creating my web site.



Exercise 8-1

Your Project #1 will be very simple. It will have two levels, with one page at the top level, and three pages on the second level. By this time, you should have decided on a Project #1 topic.

Your homework will be to create a **site map** and **three wireframes** to plan your site.

You should use PowerPoint or a different type of presentation software.

1. Site Map

Make a slide with four boxes, in the style shown below. Put the main topic in the main box at the top. Then, decide what the three 2nd-level pages will be about, and mark those as well. For example, if your topic is "Excellent grocery shops near LUJ," the three bottom boxes may be "My Basket," "Maruetsu," and "Ito Yokado."

2. Wireframes

Make three wireframes, one for the main page, and one each for the second- and third-level pages. Decide colors, image shapes and sizes, and where the images will appear. By the time you make Project #1, you will be able to make images go to the left or right (not center) and have the text wrap around the other side.

1

2

LEVEL ONE

3

LEVEL TWO

4

LEVEL THREE

Chapter 8 Checklist

- Which corner of a box defines the "position" of the whole box?
- What are the four parts of the box model, in order from inside to outside?
- What separates the margin from the padding?
- When the size of the border, margin, or padding is increased, in which directions does the box grow?
- If you add a 1px border and 10px padding to a 100px box, how many pixels wide will it become?
- What is normal flow? Where does it begin, and in which directions does it go, in what order?
- What can break the normal flow?
- What is a float?
- Where can an object be floated to?
- What wraps around a float, and what does not wrap around? (What "slides below" it?)
- If you float two objects to the same direction (both right or both left), how will the two objects be affected?
- What is a clear?
- What is the difference between a right and left clear?
- What is a site map and why is it useful?
- What is a wireframe and why is it useful?