Fundamentals of Web Design

Third New Edition • Unit 3

BY LUIS POZA

ART280 • Fundamentals of Web Design • Lakeland University

CLASS TEXTBOOK **UNIT 3**, Chapters 9 - 12

Chapter 9: Moving Boxes

TECHNICAL

9a. Publishing a Web Site

It is one thing to create a web site on your computer, and view it locally on a browser. It is very different to put it up on the World Wide Web on the Internet. Let's see how that process works, and learn a little bit about the Internet as well.

HOW THE INTERNET WORKS, LIMITED VERSION

The Internet is, essentially, billions of computer devices connected together in a huge network. Each computer has an address, called an IP address (for example, 201.47.116.28).

Although many are different from each other, all the computers communicate using the exact same set of rules, called **protocols**. The main protocol set for the Internet is TCP/IP, which sets the rules for how data is transferred between computers on the network.

Every activity on the Internet has a protocol or a set of protocols which describes how computers will carry out that activity. Email has SMTP, POP3, and IMAP. If you want to exchange files, you may use FTP. For the World Wide Web, the protocol is HTTP. Web pages and web sites are governed by the HTTP protocol.

CLIENTS AND SERVERS

Most activity on the web is between clients and servers. These are the same words used in a restaurant. A customer is a client, the people who bring their food are called servers. A client requests food; the server receives the request, gets the food, and serves it to them.

This is very similar to how things work on the Internet. You open your computer, and you start up a browser. Browsers are "client" apps.

In the browser, you type an address, like **luj.lakeland.edu**. This is a **request**. You are requesting to see the web page documents at that address. Your request is sent to the **server**, more specifically the **web server**. A "server" is a program on a computer on the network which delivers information that is requested by clients. The web server receives your request for a web page, collects the files you want, and sends them to you.

THE IP ADDRESS

Every server on the Internet has a fixed IP address, such as 216.58.197.206 (which is the address for Google's servers).

However, people cannot be expected to remember or type in such numbers. Therefore, a different address is used: the domain name.

A domain name is a placeholder for an IP address. When you type in the domain name, the IP address is retrieved and then used to communicate with the server.

How a Web Page Is Served (Simplified Version)

You open a client browser and type in a domain name.

The domain name is sent to a **DNS server**. DNS stands for Domain Name System. The DNS server has a list of domain names, and what IP addresses are assigned to them.

The DNS server receives your domain request, finds the associated IP address, and sends it back to your computer.

Your computer then sends the request directly to the web server at that IP address.

The web server receives your request and sends the file you asked for.

Your computer receives the file, which is opened by the browser.

The browser's **engine** begins to **parse** (read and execute) the web page code.

When the code requires other files, such as images, the browser sends additional requests to the server for that data. Some data (videos, ads, etc.) may even be kept on different servers, which will get the requests from your browser.

When all the data is received, the browser then begins to **render** (draw) the page. You see the page appear in the browser.

Web Servers

A web server has to meet several requirements in order to work.

Since people might be requesting data at any time, servers have to be operational and connected 24/7.

Because the address needs to be stable, a **static** (unchanging) **IP address** is required. Most people using Internet at home get IP addresses which change from time to time.

Depending on how many people will be visiting the site and how much data is exchanged, a web server might need to have a high-speed connection, especially for uploading (sending) data.

Depending on how many people visit and how much programming is used (e.g., web apps), the computer with the web server might need to be fairly powerful and fast.

The server computer, of course, requires the server software to be installed, set up, and regularly maintained by someone who understands how it works and how to fix it if problems occur.

All of this can be fairly difficult and expensive. Just the computer for the server costs quite a bit! Most people also do not have the training to maintain a server in any case.

WEB HOSTS

As a result, most people who have web sites use a **web host**. A web host is a company which has all the equipment, connections, and services to run web servers. They can offer these services much more cheaply than it would cost you to run it yourself, because the web hosts have dozens or hundreds of servers and thousands of accounts. Staff operate the site 24/7, maintaining the equipment and software, and deal with problems that come up.

There are various types of accounts a customer can choose from. If you pay a low price per month, it means you will get **shared hosting**. This is when your web site shares a single server computer with up to hundreds of other customers. The same one computer which hosts and serves your site might also be hosting 500 or so other web sites. Decent shared hosting can cost less than $10 a month for a small site.

If your web site is serious—if it is supposed to be for a business or organization that will receive a large number of visitors every day—you probably want **dedicated hosting**, which means that your web site has its own private server computer. This can be much more expensive—often more than $100 a month—but it means you have a much greater guarantee that the web site will be strong and stable. Businesses often want to have dedicated servers.

When you are looking for a web host, you want to look at various offerings:

Bandwidth — a certain amount of data uploaded/downloaded every month
Storage Space — a certain amount of HDD or SSD space for your files
Uptime — a guarantee that the server will not go offline too much
Email Accounts — the ability to create and use email accounts in your domain
Add-on Domains — the ability to have more than one domain for one account
Software — automated software to install blogs, forums, and other special features

You want to find a web host that has the features that you need, at a cost you can afford.

You will want to get information and recommendations for a good web host (many web hosts are bad). If you do not know anyone who could recommend one, you could go to a discussion site like webhostingtalk.com, do some searches, and ask around.

The web host used by this class is called BigScoots (http://bigscoots.com). For $10 a month (if you pay for a whole year at a time), you can get:

Bandwidth — 200 GB / month
Storage Space — 10 GB (SSD, which is faster than HDD)
Uptime — 99.99% (essentially, no more than 1 minute per week offline)
Email Accounts — Unlimited (which means "a lot, but not infinite")
Domains — Unlimited (you can have as many different domains as you like)
Software — cPanel front end controls, Softalicious Web App Installer

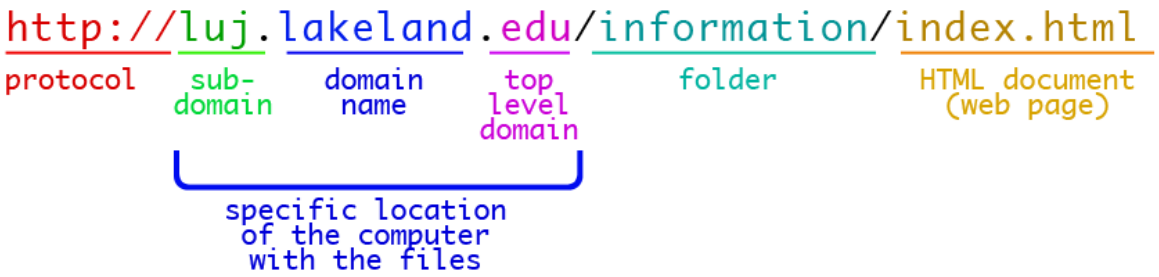
The host has other features as well: online databases, protection from some DDoS attacks, daily data backups, full-time technical support, and other services as well.

Once you have paid for your hosting, you can control your domain using a control interface, usually **cpanel**, which you can find by going to **yourdomain.com/cpanel** and using your account user name and password.

The Domain

When you want to see a web site, you have to enter a URL (Uniform Resource Locator). This is usually a **domain name**, and possibly a more specific address within the domain name.

A URL can look like this:



A URL begins by stating which **protocol** is being used. When manually typing a URL in a browser, this is not required, because it is automatically expected that the HTTP protocol will be used. *However, it is absolutely required when writing HTML code.*

Typing "www" (or including it in the HTML code) is usually optional these days. In some cases, including *www* will even cause problems.

Next, there may be a **subdomain**. A subdomain is a way to divide a domain into different parts, like Lakeland's web site is divided into *japan.lakeland.edu*, *luj.lakeland.edu*, and *my.lakeland.edu*.

Next, there must be a **domain name**. A domain name is a specific name chosen by the owner of the servers.

The **top level domain** (**TLD**) identifies the type, or category of web site—.com for commercial, .edu for education, etc.

If the desired web page is in a **folder** (also called a **directory**), the folder name is then typed, followed by a slash. The single slash means that the previous text is a folder name.

Finally, we get the full **filename** for the HTML document. This includes the filename extension (usually *.html*). If the filename is *index.html*, then it is not really necessary to include that; if no filename is given, the server automatically gives the index file.

Getting a Domain Name

A domain name can be acquired from a **registrar**, or a company that handles such sales. Popular registrars include GoDaddy and NameCheap.

When you find a domain name, you have to decide your **domain name** and the **TLD**.

TLDs determine the price. Any domain in the .com TLD will cost between $10 and $16 per year, depending on which registrar you use (prices vary). Other TLDs could cost as little as $3 or as much as $100 per year.

While **.com** is the most desired TLD, most of the really good names are taken.

What is a good domain name? The best domain name is one that is short, related to your web site's topic or purpose, easy to remember, and not easy to misspell or mistake. Domain names can have any letter, number, and a hyphen, but no other characters are allowed. Hyphens are not usually desired.

In the **.com** TLD, it is extremely difficult to find domain names which are short or use common words, terms, or phrases. Most have been claimed; most others are sold at especially high process by cyber squatters (people who buy many domain names so they can resell them at higher prices).

Fortunately, many, many other TLDs are available. For example, I recently got **luj.tokyo** with the ".tokyo" TLD. It works quite well for me!

Some TLDs work well with specific names. For example, if you want to make a small news-type site called ***The Lakeland Times,*** but the domain **lakelandtimes.com** is taken, then perhaps you could use the Spanish **.es** TLD, and get **lakelandtim.es** instead.

Once you have decided upon a domain name, you need to buy it from the registrar. You can pay for as many years in advance as you like, or have the registrar charge you once per year.

Once you have paid for the domain name, you can then access and control the domain using a special administrator page.

Make the Connection

Once you have both a **web host** and a **domain name**, you have to make sure the two are connected via the **Domain Name System**.

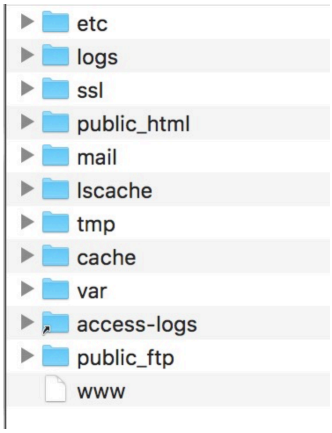
When you sign up for a web hosting account, you will receive an email with your account information. Included in the email should be two **Nameserver** addresses.

You can then log in to your registrar administrator account, and go to the area that allows you to manage your domains. Among the settings should be the Nameserver address settings. That will automatically be set to the nameservers of the registrar. You must change them to the two custom nameserver addresses you got from your web host.

Once you enter the addresses, the changes must **propagate** through the system—that is, the IP address indicated by the web host's nameserver must be recorded on all DNS servers around the world. This could take anywhere from a few minutes to a few days.

You can test whether the domain name has propagated by uploading an index page to the main directory. If the page appears when you type your domain name into the browser, then the DNS information has gone active. However, there are different DNS servers for different regions of the world, and some update their data earlier or later than others.

Uploading Files Using FTP



To upload files to your web site, you should use an FTP program like FileZilla. These instructions were already given to you in Part III in "Getting Ready" in the Unit 1 handout.

FTP means "File Transfer Protocol," and is the set of rules that allow computers to exchange files remotely.

When you access your account using FTP, you will find that there are several directories (folders), as seen in the image at right.

Most of these are folders you do not need to access; they contain the email accounts, visitor logs, and other necessary files. ***Do not change anything at this level!***

There is **one folder** that you need to access: **public_html**. That is the folder which contains your web site.

When you open it up, it may already contain a few things, such as a **cgi-bin** folder, or an error log file. Just leave these files alone. Do not put anything into the cgi-bin or other pre-existing folders.

The **index.html** file in the public_html folder will be the main web page for the domain.

If you put or create a folder in the public html folder, that will be a folder in the main site. For example, if I have an account for **lcjcamp.us** and I go into the public folder, and create a new folder named **info**, then the address for that folder would be **lcjcamp.us/info/**

Add-on and Sub Domains

A web host account is based upon a single domain name. What if you want to create multiple web sites, however? For luj.tokyo, I have **cps.luj.tokyo** and **gen.luj.tokyo** and a few others, so that I can put materials for each class into each site. These are called **subdomains**. The cpanel controls allow you to create subdomains with a special tool. The tool will create a folder in your public_html folder (or a sub-folder), and it will treat *that* folder as the main folder for the subdomain.

To create **cps**.luj.tokyo, I opened the cpanel for my luj.tokyo account, and created a subdomain in a folder with the name **cps**. Everything I put into that would appear if someone asked for cps.luj.tokyo.

Add-on domains are similar. You will choose a domain name for your project in this class. You will add the domain to the lcjcamp.us account.

Again, you can do this in the cpanel; go to "Add-on Domains" and input the information. A folder will be created which will hold the web site for that domain. Anyone who types your domain name into a browser will be directed to that folder.

CODE

9b. Columns

Most web sites have **columns**. Up until now, you have been creating web sites with only one column, all of it in the wrapper: header, nav, main, and footer. Now that we know about floats and clears, we're ready to make things a little more complex.

First, we'll need more semantic structural elements. Here are three new ones:

`<article>`	*Used for individual articles, as in a blog, magazine, or newspaper*
`<section>`	*Used for sections of a single document, such as chapters or units*
`<aside>`	*Used for content which is less closely related to the main content*

Of the above new tags, `<article>` and `<section>` could replace `<main>` as you have used it until now.

An **article** is an entry in a series. If you created a blog, each blog post would be in an **article**. Whenever you visit news or other periodical web sites, each entry by an author on a different topic or on a different date would be within an article. Articles can also be information which is independent of other information on the page, but which is still central content.

A **section** is a unit of a larger writing, such as a chapter of a book, or a section of a report. For example, if this document were on a web page, each of the Sections (this is Section #16) would be in its own `<section>` tag.

Both article and section usually have a heading tag (h1 ~ h6) at the top, but this is not absolutely required.

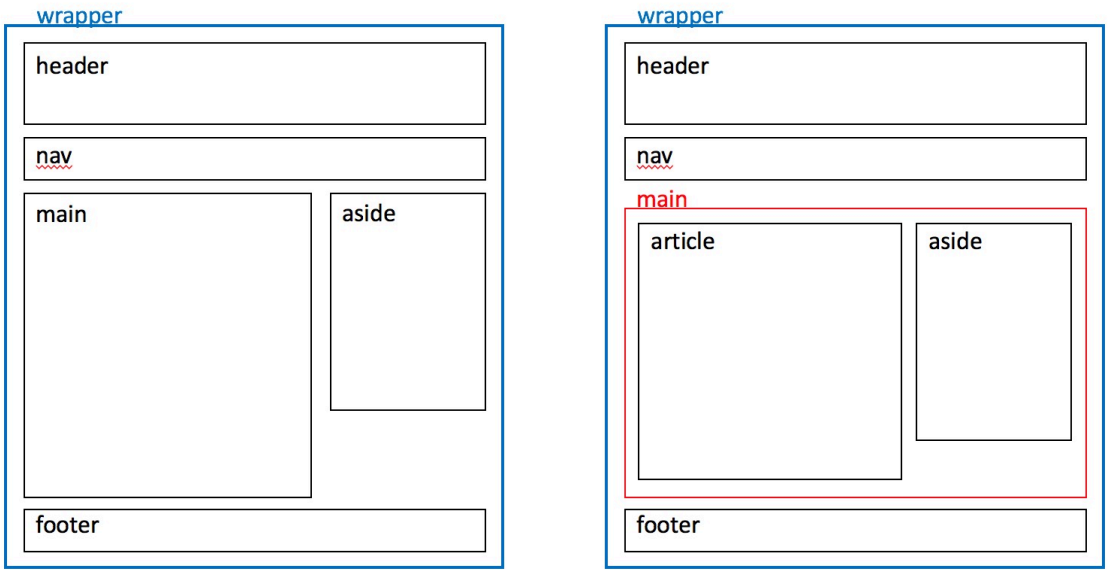
There is much disagreement amongst web designers as to how these can and cannot be used. Many designers use main, section, and article interchangeably, depending on what they decide they should be. Others are much more strict about how they may be employed. In this class, I will allow a more casual use of these tags, mostly because we will not be creating the longer and more complex kind of sites that would perhaps require stricter use.

An **aside** is typically used as a container that does not contain core content. An aside can have elements such as pull quotes (quotes taken from the main text but displayed in larger font size). However, it is typically used as a column, in particular a column that has information that is related to the main content, but not directly so: links to related articles, links to outside web sites, a search bar, or special features such as news feeds, Twitter feeds, calendars, jump menus, etc.

You can still use the **main** tag for the main content of the page. You could also use the main tag as a container for two or more columns.

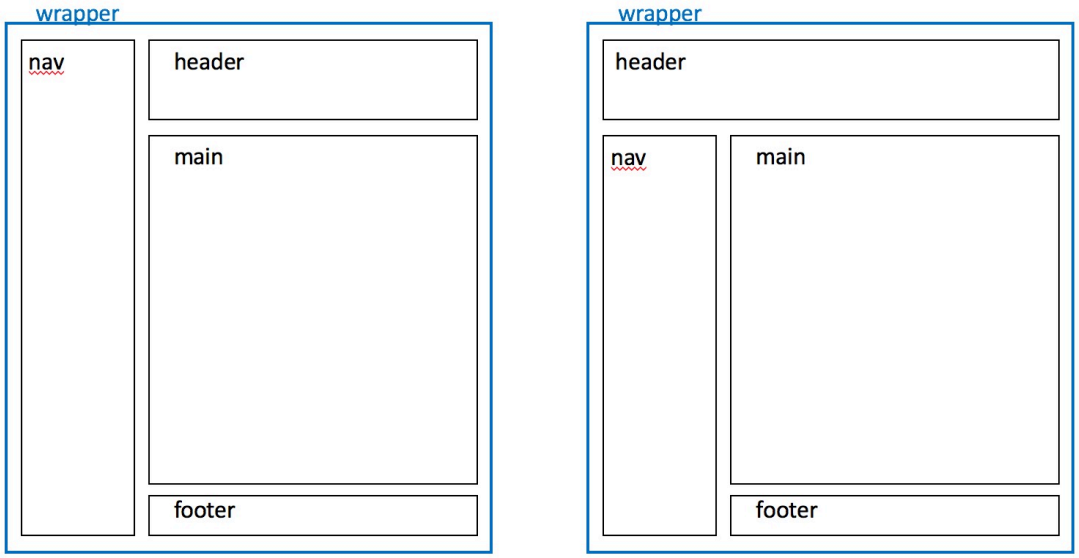
ARRANGEMENTS OF COLUMNS

Columns can be used in a variety of ways. Below is a very standard layout:



You can see that the `<main>` can be used to hold the two columns, or could be one of the columns, depending on which works better for the design you might want. A `<div>` could also hold the columns. Having a container for multiple columns can help with problems such as smooth control of background colors or images, or varying column lengths.

In many cases, the nav bar could be the second column—but it is better to use this only if the nav controls are more than just a few small links, as most of the column would be empty, leaving a large blank area that might look awkward.

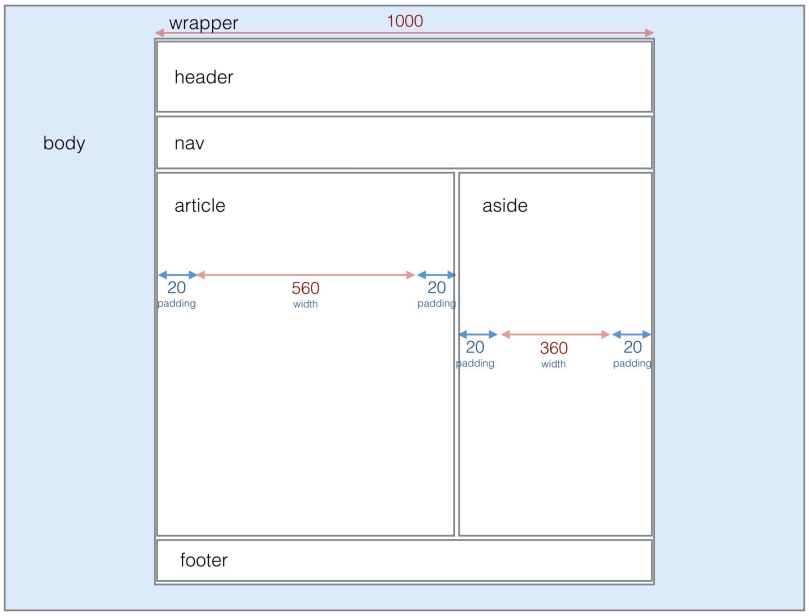


Note that the header and/or the footer can be above, below, or to the side of the column. Any group of block which form a rectangle can also be grouped together within a `<div>`.

CODING COLUMNS

The first thing you have to do is plan. This is where **wireframes** come in. Remember back in Section 1b when I mentioned wireframes as being an important part of planning; before now, we have only used single columns, so wireframes were less important. However, when using floats and columns, counting every pixel exactly becomes very important.

Here is an example of a wireframe with basic width planning for two columns:



Note that the **widths** of the article and aside do not include the padding; you must calculate the space for both the width *and* the padding together. The above planning assumes that there is no border between the columns: 600px for the article (20+560+20) and 400px for the aside:

planning with no border

article			aside		
padding	width	padding	padding	width	padding
20	560	20	20	360	20

That adds up to 1000px, which fits the container (the wrapper) exactly. If you also add a **one-pixel border** between the article and the aside (for example, `border-right: 1px solid rgb(150,150,150);`), that *also* adds to the width, making the total width 1001px—too much for the wrapper. Therefore, one pixel must be subtracted from somewhere else—below, I subtracted 1px from the article width. Anywhere is OK, not just the width of that column.

*planning **with** border*

article				aside		
padding	width	padding	border	padding	width	padding
20	559	20	1	20	360	20

All of the numbers must add up to the **width** of the container (in this case, the wrapper). You should not count the border or padding of the container in this case—just the width. In some browsers, if you zoom out on a page, (the content looks smaller), the pixel sizes mismatch, and even 1000px of content gets ruined. Leaving a few pixels extra space fixes that.

Once you have calculated sizes, you are ready to code. You know how to create the basic structure, such as the header, nav, and footer; however, how do you create the columns?

The answer is, follow normal flow. Remember, normal flow is left to right, top to bottom. If we create the page shown in the wireframe above, the structural tags should look like this:

```
<div id="wrapper">
    <header>

    </header>
    <nav>

    </nav>
    <article>

    </article>
    <aside>

    </aside>
    <footer>

    </footer>
</div>
```

Notice that the article comes before the aside; in my design, the article will be on the left, and the aside will be on the right. You could still have them appear the same way even if the `<aside>` comes first in the code, but it is best to place objects in the code in the order they appear on the page.

So now the CSS:

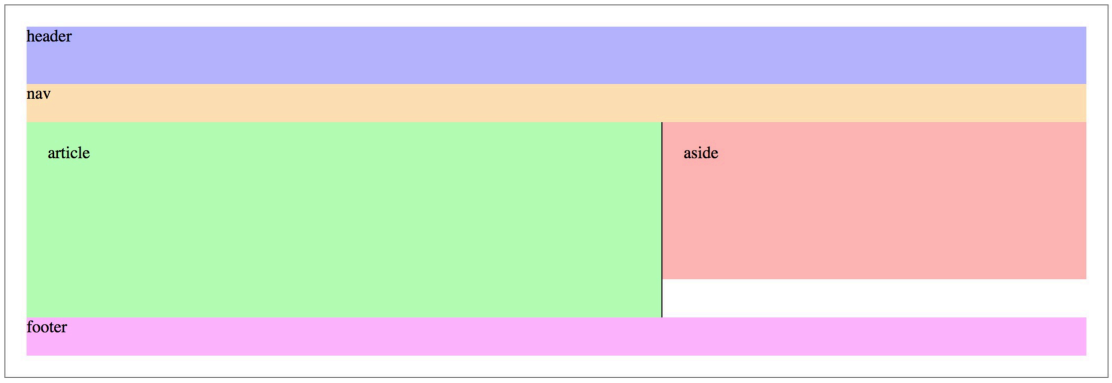
```
article {
    float: left;
    width: 559px;
    padding: 20px;
    border-right: 1px solid black;
}
aside {
    float: left;
    width: 360px;
    padding: 20px;
}
footer {
    clear: both;
}
```

Notice that I used the measurements that I calculated with the wireframe.

Notice that both the article *and* the aside are floated left; this makes them act similar to the `inline-block` that we used with the nav bar buttons. **You could float the aside** (not the article) **to the right, and it would still work**.

Notice that I used the clear: both; in the footer; otherwise, the article and aside would go outside the wrapper and stick out on the bottom.

The code above will create the page shown in this illustration, with the different structural elements emphasized with background colors; note the border between the article and aside:



Notice that the wrapper has a border and padding (the grey box and the white space between that and the structural elements); these are separate from the width, which is 1000 pixels. The structural parts all fit within the 1000-pixel width.

Notice that the aside is shorter than the article; this causes a white gap below the aside to appear. If you have different backgrounds, this can create awkward problems with the design. How can you fix this? There are several possible approaches:

1. Don't use backgrounds for the different structural parts, and let the wrapper background show through;
2. Use the same background for the wrapper and the shorter of the two structural parts;
3. Create a container around the article and aside; give the container the same background as the article; then create a left and bottom border for the aside.

It might be hard to visualize that last one, so let me show you the code:

```
<div id="wrapper">
    <header>

    </header>
    <nav>

    </nav>
    <main>
        <article>

        </article>
        <aside>

        </aside>
        <br style="clear: both;">
    </main>
    <footer>

    </footer>
</div>
```

Note that I used a special tag at the bottom of the main: `<br style="clear: both;">`. Why?

The answer is that I am using floats for the <main> content. Remember, blocks don't "see" floats. If I did not have the
 tag there, there would be nothing to clear the floats.

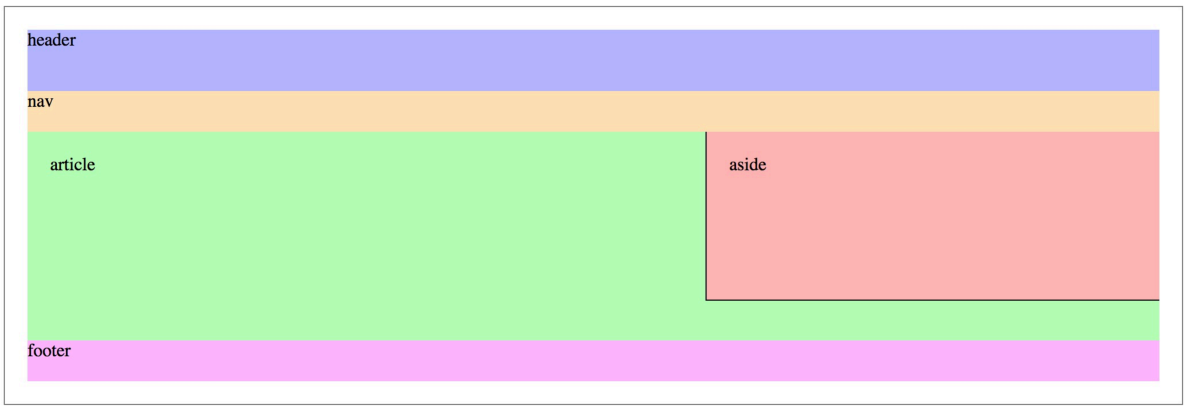
The result would be that the <main> area would collapse into 0 pixels height, and we would not see the background color we are trying to create. I cannot expect the footer to help; if the footer has `clear: both;` then that just means the footer will appear below the floats, and would help the wrapper also go below the floats. However, the footer would not help to expand the main.

I call this the break with `clear: both;` the "magic tag." It can be used to clear floats in this manner.

Back to work—here is the CSS code for our new page:

```
main {
    background-color: rgb(180,255,180);          light green bg
    clear: both;
}
article {
    float: left;
    width: 559px;
    padding: 20px;
    background-color: rgb(180,255,180);          light green bg
}
aside {
    float: right;
    width: 360px;
    padding: 20px;
    border-left: 1px solid black;
    border-bottom: 1px solid black;
    background-color: rgb(255,180,180);          light red bg
}
```

Notice the main and article both have the same background color. Notice the borders have moved to the aside, but since we have not added extra borders, the widths remain the same. Here is the final effect:



This solution works no matter whether the article is longer than the aside, or the aside is longer than the article. The only potential problem is if the footer has a top border, and the aside is longer than the article; in that case, the bottom border of the aside would hit the top border of the footer, creating a double-border that would look too thick.

9c. Positioning

Sometimes you need more control over the position of an object. The `float` property is nice, but it is limited to only shoving objects to the far left or the far right.

The `position` property gives you *exact* control.

Positioning in Normal Flow

In a web page, an object's position is determined by the top-left corner of its container. That position is set by normal flow.

In normal flow, you can only move something by creating margins, padding, or indents, or by putting something before something else. These do not actually move the object; these methods only create extra spaces around the object.

Imagine if you could only move if somebody pushed you, or by expanding a balloon around yourself. That's similar to what we're talking about

Sometimes you need to just move an object by itself.

This can be done in CSS with the **position** property.

The CSS

First, you must decide *how* the object will be moved:

`position: static;`	*this is the default*
`position: relative;`	*moves the object relative to the current position*
`position: absolute;`	*places the object relative to the edge of the page*
`position: fixed;`	*places it relative to the edge, without scrolling*

Next, you must decide *by how much* the object is moved or placed:

`left: 100px;`	*moves the object rightwards from a set position—"from the left"*
`right: 100px;`	*moves the object leftwards from a set position—"from the right"*
`top: 100px;`	*moves the object down from a set position—"from the top"*
`bottom: 100px;`	*moves the object up from a set position—"from the bottom"*

Note that static and relative maintain the normal flow—the object's location in the normal flow is preserved. However, if an element is set to have either absolute and fixed position, *they are taken out of the normal flow*, and so they are "ignored" by the rest of the objects on the page.

9d. The Positions

STATIC POSITIONING

Static is the default position setting. This is the same as normal flow. You should not set anything as "static" unless it is somehow set to be in some other position.

An object which is "static" will not be affected by the top, bottom, left, and right properties.

RELATIVE POSITIONING

Relative will "nudge" an object in some direction, starting with its position in the normal flow. For example, if you have a box and give it the style:

```
position: relative;
left: 10px;
```

Then it will move 10 pixels to the right, from the left starting position in normal flow.

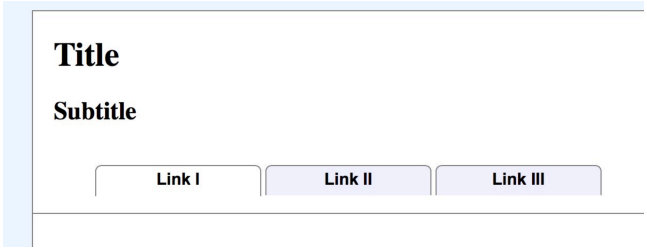
An important point about the relative position is that **the object stays in the normal flow**. Even if the object appears to move, it leaves a "ghost" of itself in its original position.

No matter where the object moves because of relative position movement, *other objects act as if it is still in the normal plow position*. Normally, if you move an object downwards, for example, by adding to its height, or adding a margin-top, then everything below it moves by the same amount. Not so with relative positioning: other objects ignore the movement.

One problem is that relative positioning can cause overlap, where two objects occupy the same area on a page. In this case, one object will always appear "above" or "below" the other object. Normally, the object which comes later in the code appears on top.

The relative position is good for making an object move a few pixels this way or that way without disturbing anything else on the page.

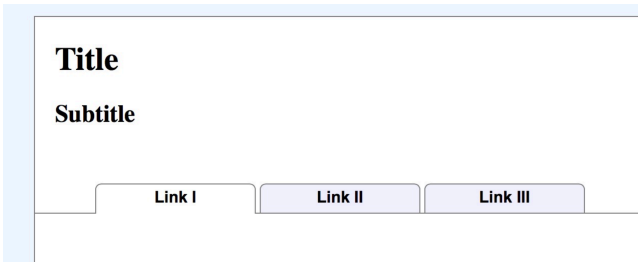
For example: let's say you want to make your nav buttons look like folder tabs. You could begin by rounding only the tops, and removing the bottom border. You then add a top border to the <main> area.



However, the tabs are still too high up. They don't reach the top of the <main> area. How do you make them reach down? You use position: relative; of course.

```
position: relative;
top: 17px;
```

That code will make them appear down far enough that they will look like they are sitting right on top of the <main> area.



This is certainly not the only use, but a good example of a popular trick using relative positioning.

The next two positions, **absolute** and **fixed**, work in very different and important ways:

1. Objects with these positions will be removed from normal flow
2. All other objects, inline and block, will "ignore" the positioned objects
3. The objects will appear in their "normal flow" locations until the top, bottom, left, and/or right properties are introduced.
4. Using the top, bottom, left, and right properties, the objects will be positioned relative to the edge of the **window** (not the edge of the page)
5. The objects will appear "above" everything in normal flow

ABSOLUTE POSITIONING

Absolute positioning allows you to set the position of any object *relative to the top, right, bottom, or left edges of the browser window.* For example, if you set an object's position to:

```
position: absolute;
right: 10px;
```

That object will move to a position 10 pixels from the right edge of the browser window. It is important to note that the object will move *with the browser window*, meaning that if the browser window becomes smaller or larger, the object will move with that—it will *not* move with the page content, unless the page content is *also* locked to the right edge of the window.

Anything with an absolute position will be removed from the normal flow, appearing "above" everything in the normal flow. Everything else in the normal flow will react as if the "absolute" object has disappeared.

If you give an object the style `position: absolute;` but you do not add top/right/bottom/left values, then it will remain in its position in the normal flow, "floating above" everything else, which now ignores it. Only when you add a top, bottom, left, or right property will the object then take up a position relative to an edge of the browser window.

Any object which has absolute position will move with the edge of the browser window; if you make the browser window smaller, the absolute object will move relative to that, no matter what is happening in the normal flow.

Absolute positioning is very good for exact placement of graphics or boxes, but you must be careful that they do not appear in front of text or other objects, thereby obscuring them.

"Capturing" the Absolute

Sometimes you will want to use absolute positioning, but instead of making it relative to the browser window, you will want to make it relative to the edges of a structural area, such as the <header>.

How do you do that? give the CSS style `position: relative;` *to the container* of the object you want to position inside of it.

For example:

```
header {
	position: relative;
}
```

```
header img {
	position: absolute;
	left: 30px;
	top: -10px;
}
```

This allows the `position: absolute;` to define a location within the header! You can see the effect of the code above in the image shown below: Ponta's image is now relative to the header.

FIXED POSITIONING

The `position: fixed;` property works almost exactly the same as the absolute property, except this version will not scroll with the rest of the page.

The position in which the object stays is relative to the edge of the browser window, not the edge of the page. Therefore, if you give an object this style:

```
position: fixed;
bottom: 0px;
right: 0px;
```

Then the object will appear glued to the bottom-right edge of the browser window, no matter how much you scroll.

Fixed objects are good for placing banners or boxes that you want to always be visible, no matter where the user scrolls. However, if you use a Jello layout, different browser window widths could create extremely different effects, so be careful.

9e. Arranging in 3D Space

A basic rule of code is that whatever comes later is “above” or on top of anything that came before, if their locations overlap.

Giving an object a `position` allows it to appear “above” other images that came below it. Floats have a similar effect.

For example, when you make a drop-down menu, you use `position: relative;` to put it “above” other close-by objects. However, if you have columns or images beneath it which are floated, their floating status puts them on the same “level” as a positioned object, and again the later code appears to be on top.

Because positioning and floating causes objects to overlap in ways you may not expect, there is another property in CSS which allows you to control the order of layering in a web site.

```
z-index: 10;
```

Notice that there are no units for the value. Quite simply, if you have two objects, the highest number puts it on top, even if lower-numbered object came first in the code. The z-index number can be any number you like. It is best *not* to arrange the "lowest" object with the number "1" because you may want to leave room to put other objects below that one later.

If you find that just adding “`z-index`” does not do the trick, add `position: relative;` to the same rule; that may complete the effect.

DESIGN

9f. Image Filters

CSS includes a set of tools that allows you to adjust the appearance of images; these are called **filters**. You can make images have different color values, different brightness and contrast, or even cause the image to blur or turn into a negative image.

CSS FILTERS

Note: these will **not** work with Internet Explorer. To make them work with Internet Explorer, you must use special code which works only with IE. You can use just these filters and ignore Internet Explorer, so long as the page still works otherwise in IE.

You can use the **filter** property to adjust how images look:

```
filter: blur(5px);
filter: brightness(50%);
filter: hue-rotate(90deg);
filter: contrast(200%);
filter: invert(75%);
filter: saturate(30%);
filter: sepia(60%);
```

Blur uses a pixel setting;
Brightness can use a 0 ~ 1 setting in addition to percentages; Hue-rotate uses a degree setting (imagine a color wheel); All others use percentages.

Blur will add a blur effect to the entire object (even 1px is pretty blurred);

Brightness will make the object be dark (0%), normal (100%), or brighter (100%+);

Hue-rotate will add a false hue (color) to the object;

Contrast will make an object be gray (0%), normal (100%), or deep-color (100%+);

Invert will go from normal (0%), gray (50%), or negative (100%);

Saturate will go from black & white (0%), normal (100%), or extreme colors (100%+);

Sepia will make an image change from normal (0%) to sepia coloring (100%).

Example:

```
img {
        filter: blur(2px) saturate(200%);
}
```

Note that you can use more than one filter at a time, by adding a space and then a second filter value, as shown above. Just remember to make two of each: one normal, and the other for webkit. Also note that this works for most objects, not just images!

In addition to simply using the filter effects, you can animate them as well, with the transition property. You can make a normal image change to a filter by placing the filter code in the hover rule, or you can change from one filter setting to a different setting by placing different filter styles in the normal and the hover rules.

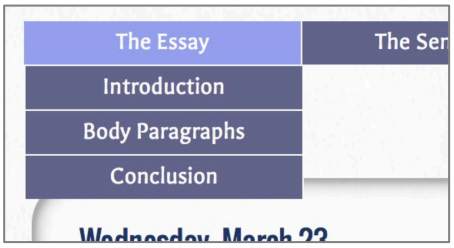
Chapter 9 Checklist

- ❑ What is FTP and why do we use it?
- ❑ What is a Web Host? What services do they provide? Why are they useful?
- ❑ What is a Web Server? How does it work, basically?
- ❑ What is a domain name? What are the parts of a IRL?
- ❑ What are the semantic purposes for the `article`, `aside`, and `section` structural tags?
- ❑ What does the position property do?
- ❑ Which two position values are not in the normal flow?
- ❑ Which position is the default (all objects have this unless a different one is set)?
- ❑ Which position allows you to "nudge" an object while keeping its place?
- ❑ What is the advantage of "nudging" an object using the relative position?
- ❑ If you set a relative-positioned object to be `right: 20px;` which direction will the object appear to move? To the right, or to the left?
- ❑ Once you make an object have absolute positioning, what happens to the content which used to be below it? Will the absolute object appear above or below the normal flow content?
- ❑ Without setting a top, right, bottom, or left location for an absolute object, where will the object appear on the page?
- ❑ When you set an absolute object's position to top, right, bottom, or left, what is the origin point for these locations? If you set an absolute object to be `left: 20px;`, then where will the 20px be measured from?
- ❑ How two steps do you need to take in order to make an absolute object set its location within a block tag?
- ❑ If you want to make one part of the page stay in the same place even when the rest of the page scrolls, which position do you use?
- ❑ Using position can make two objects use the same space on the page. What property is used to control which one is on top?
- ❑ Without using that special property, which object is "above" the other, by default?
- ❑ Can you use all the different filters? Did you try?

Chapter 10: Drop-Down Menus

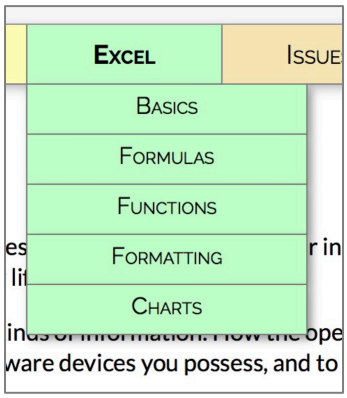
FUNDAMENTALS

10a. What Are Drop-Down Menus, and Why?



Many web sites use dynamic menus which expand when you hover over a given button. Usually, the visible menu bar shows the names of categories; when you hover over a specific category button, a list of specific page buttons drops down from the original.

This is done partially for economy of space, so that with a web site that has dozens or even hundreds of pages, each page is not filled with buttons.



More specifically, the menus help express **an extra level of pages**.

LEVELING

Web sites can have multiple levels of pages.

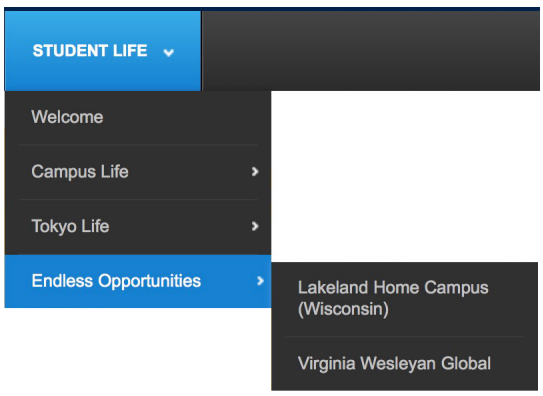
The first level is often just one page: the main, top, or "home" page for the web site. This is the index.html document, the one that automatically opens when a directory is viewed.

The second level is often sections or categories. Large web sites are organized into sections. For example, a news web site might be divided into Domestic, International, Business, Sports, Entertainment, and Living. A supermarket web site could be divided into Produce, Meat & Fish, Dairy, Bakery, Canned Goods, Frozen Goods, and so on. Some sites may only have two or three sections/categories, but often they have more.

In a smaller web site, the second level might be the "end" of the site, with no other pages below it. A site with only two levels does not require a drop-down menu.

The third level is usually the last level, and has individual pages with very specific content.

It is possible for a web site to have **a fourth level**, if any third-level topic can be further divided into separate smaller topics which each deserve a separate web page.



That fourth level would be a **submenu**. In the example at right from our school home page, the index page is the first level; "Student Life" is the second level; "Endless Opportunities" is the third level; and "Lakeland Home Campus" is the fourth level. We will not learn submenus in this class.

Level Content

Assuming that a site has three levels, what content would each level have?

The first level, the main page, would be an overall introduction to the subject of the web site. This could be equated to the introduction paragraph of an essay. The subject is introduced, perhaps explained, and major themes are expressed. *By looking at the main page, a visitor should have a strong understanding of what the content of the entire web site will be about*. If the visitor is surprised by the kind of content they find in other pages, then the main page did not do a very good job.

The second level, the section pages, have a slightly more difficult task. These pages must introduce the specific categories, and lead readers to the more specific pages. These pages represent something similar to paragraph topic sentences in an essay.

The problem is that these pages are stuck between the explanatory main page and the specifically detailed third-level pages. The main page already explained all about the general topic, and the third-level pages will handle all of the specifics. What, then, is left for the second-level pages?

This is a problem you have to deal with when you design your web site. The second-level pages should be more specific than the main page, and should have more information. They should give an overview of the section or category.

These pages can explain all of the facts and attributes which the more detailed pages have in common.

For example, in the case of a web site about birds, the sections of the web site would be the different categories of bird species: waterfowl, predators, songbirds, flightless birds, etc. The second-level "category" web page introducing, say, predatory birds would explain the sub-groups of birds (hawks, eagles, ospreys, owls, etc.), show the common physical features (bills, talons, wing types), and types of prey (mice & rats, fish, other birds). There should be more than enough details which would not fit into the first or third levels to create interesting and fully-detailed second-level web pages.

Alternately, the page could have abstracts, or *short, one- or two-paragraph summaries of the information expressed on the third-level pages*. For example, if you go to a newspaper web site, and visit any one of the sections, it may look like a collection of various stories; each story would have a headline and a short paragraph or two, just enough to let you know what the story is, probably designed to make you interested in the story, and want to know more.

The third level would be filled with the specific examples and details of the topic it covers.

CODE

10b. How to Make the Drop-Down Menus

There are many ways to make drop-down menus. I will show you the simplest way I can find.

Drop-down menus depend on the extra buttons to be hidden until a user hovers over the visible button to which they are related. They require the use of several CSS tricks that we learned in the previous three or four sections:

- *float* is required to make the different columns of the menu to line up; for this effect, inline-block will not work.
- *position* is needed to allow the drop-down menu to appear above text below it.
- *display* or *opacity* is needed to hide the extra buttons until they need to appear.
- *hover* is needed to make the extra buttons appear when the visitor moves to the menu.

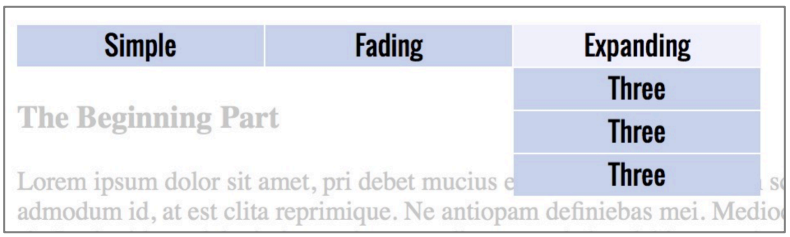
When creating the menu in HTML, the `<ul>` list is still used, but this time, all the buttons that form a drop-down menu must be grouped together; the best way to do that is with a `<div>`.

Here is an example of a normal, static menu, like we learned in Chapter 7f:

```
nav ul li {
    list-style-type: none;
    width: 150px;
    display: inline-block;
    background-color: rgb(200,210,235);
}
nav ul li a {
    display: block;
    text-align: center;
    padding: 3px 0px;
    text-decoration: none;
}
```

The menus we will make use much the same code, but with an important difference: the `display: inline-block;` show in red above is not used because it does not work in this type of code.

Here is what a drop-down menu may look like, when finished:



what the finished menu will look like

This is the necessary HTML; note that each menu column is inside a `<div>` with a class, and that only menu options which *don't* appear at first have another class:

```
<nav>
     <ul>
          <div class="group">
               <li><a href="dropmenu.html">Simple</a></li>
          </div>
          <div class="group">
               <li><a href="drop2.html">Fading</a></li>
               <li class="drop"><a href="#">Two</a></li>
               <li class="drop"><a href="#">Two</a></li>
               <li class="drop"><a href="#">Two</a></li>
               <li class="drop"><a href="#">Two</a></li>
          </div>
          <div class="group">
               <li><a href="drop3.html">Expanding</a></li>
               <li class="drop"><a href="#">Three</a></li>
               <li class="drop"><a href="#">Three</a></li>
               <li class="drop"><a href="#">Three</a></li>
          </div>
     </ul>
     <br style="clear: both;">
</nav>
```

The following points are my plan for how the menu will act:

- menu buttons with no drop-down buttons appear normally (the “Simple” link)
- any menu button with drop-down buttons is contained in a div tag called "group"
- any menu button which is hidden but will appear and drop down, I gave a class which I call "drop"
- the top (always visible) button of each drop-down group has no class assigned

Using these basic rules, you can add as many visible buttons as you like; you can make as many of them drop-downs as you like; and you can have as many drop-down buttons as you like in each group. Each group can have any number of buttons, there is no need to have a specific number, or for the menus to have the same number of buttons.

Now the CSS. To begin with, let's have the CSS which simply creates the buttons, with height:

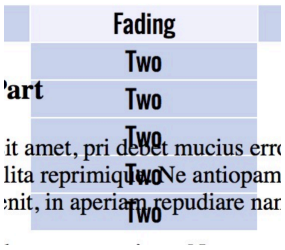
```
nav ul li {
       list-style-type: none;
       width: 150px;
       font-family: 'Arial', sans-serif;
       background-color: rgb(200,210,235);
       height: 27px;
}
```

Note that I am making simple color buttons, with no borders, rounded corners, or shadows.

Note that the `height` *will not be necessary* once we create the **nav ul li a** rule. It can be deleted then.

Next, let's add the CSS that helps make these buttons drop-downs:

```
nav ul li {
	width: 150px;
	list-style-type: none;
	font-family: 'Arial', sans-serif;
	background-color: rgb(200,210,235);
	height: 27px;
	position: relative;
}
```



The `position` property makes the buttons appear above any text that exists below the buttons. If you don't have this declaration, then the text in the article below the buttons will later appear to remain above the buttons, making a mess of them (see illustration above at right).

Don't forget the CSS that makes the links cover the whole buttons:

```
nav ul li a {
	display: block;
	text-align: center;
	padding: 3px 0px;
	text-decoration: none;
}
```

Next, we must define the two classes we created: `group` and `drop`.

Remember, the `group` class defines the div which holds the visible button at the top of the drop down, as well as all of the temporarily invisible buttons below it.

```
.group {
	width: 152px;
	height: 30px;
	float: left;
}
```

- The width prevents the buttons from spreading out to the right instead of down.
- The height prevents the drop-down buttons from pushing the rest of the page downwards.
- The float makes the buttons sit to the right of the previous button.
- The height can be between 1px and 19px in order to work; 20px and over will push the rest of the page down. Margins can be added to space the buttons apart.

Next, we must hide the buttons that will drop down, or else they will be visible all the time:

```
.drop {
	display: none;
}
```

Using `display: none;` makes the buttons invisible. We define the drop class as being a descendant of the group class because of the next rule. The descendant selector may not be completely necessary, but it makes the code more clear.

Finally, we have the hover rule. This is a little tricky:

```
.group:hover .drop {
	display: block;
}
```

The `display: block;` declaration makes the buttons visible. The border-top separates the buttons in a way that is specific to the style I am using for this menu.

Notice that **the hover applies to the `group` class, and not to the `drop` class**. That's the most important trick in the whole design! The `.group:hover .drop` rule *defines* the drop class, but it only *appears* when the group class is hovered over.

You might think, "Why not just make it `.drop:hover`?" That won't work, because the drop buttons are hidden with `display: none;` so they cannot be hovered over. The way we have done this, we are applying a rule to the invisible buttons, but triggering it only when the cursor hovers over a different button.

Here is all of the CSS in one bunch to make it more clear. Hopefully, now that I have explained all the parts, you can read this and see what everything is doing and why it is shaped like this. **Note** that you can now delete the `height: 30px;` in the **nav ul li**.

You should try deleting the `position: relative;` in the `nav ul li`, and see what happens!

```
nav ul li {
	list-style-type: none;
	width: 150px;
	font-family: 'Arial', sans-serif;
	background-color: rgb(200,210,235);
	height: 30px;
	position: relative;
}
nav ul li a {
	display: block;
	text-align: center;
	padding: 5px 0px;
	text-decoration: none;
}
.group {
	width: 152px;
	height: 30px;
	float: left;
}
.drop {
	display: none;
}
.group:hover .drop {
	display: block;
}
```

To better see how the links respond to a hover, add this:

```
nav ul li:hover {
	background-color: white;
}
```

That will show the menu buttons behave when you hover over them. Add a transition to the main rule to get a smooth animation effect!

10c. How to Make Animated Drop-Down Menus

The drop-down menu described in the previous section is a very basic one, the simplest I could devise. When you use it, the drop-down menus appear suddenly in their full form. That presentation works, but it is abrupt and not very attractive.

A much more elegant design has the drop-down buttons appearing more gently, either by fading in, descending, or both.

Here is how to change the code I expressed before so the menus appear in a much more pleasing fashion.

FADING MENUS

In this version, the menus will fade in and out. This uses the `transition` property we learned earlier.

However, we cannot use the `display: none;` declaration to hide the drop-down buttons with an animation, because `display` does not work with `transition`.

Therefore, we need to use two new properties: `visibility` and `opacity`.

Visibility and opacity both make an object on a page disappear. However, for drop-down menus, each one by itself has a weakness.

Like `display`, `visibility` does not work with the `transition` property. Both display and visibility are on/off properties; since you cannot have anything in-between, there is no possibility for animation.

`display: none;`	completely invisible
`display: block;`	completely visible
`visibility: hidden;`	completely invisible
`visibility: visible;`	completely visible

The opacity property does work with transition, because opacity works in grades:

`opacity: 0;`	completely invisible
`opacity: 0.1;`	slightly visible
`opacity: 0.5;`	half visible
`opacity: 1;`	completely visible

Therefore, opacity must be our choice for fading menus. However, opacity has a weakness: it remains "visible" to the cursor in regards to a hover. If you use opacity by itself, then whenever your cursor appears above a hidden button, the button appears. This means that moving the button *below* the drop menu makes the menus appear, which is strange!

As it turns out, using both visibility and opacity works. Both make the buttons invisible; opacity makes the animation possible; and visibility hides the buttons from the hover.

Here is the new CSS for the drop class (the other CSS remains the same):

```
.drop {
	visibility: hidden;
	opacity: 0;
	transition: 0.4s;
}
.group:hover .drop {
	visibility: visible;
	opacity: 1;
}
```

If you have the visibility without the opacity, the menus will not animate, and will delay disappearing when the hover ends. If you use opacity but not visibility, the menus will look good, but they will expand when the cursor is *under* the menu, but not hovering.

Dropping Menus

In this version, the menus will literally drop down, moving from top to bottom.

As before, we will use `opacity` instead of `display`, but this time, we will not need `visibility`, because we will instead remove `height` from the hidden buttons. They will begin with no height, therefore allowing them to grow and expand, pushing from the top to the bottom.

The CSS is exactly the same, but `height` is added—none in the drop, full height in the hover:

```
.drop {
	height: 0px;
	visibility: hidden;
	opacity: 0;
	transition: 0.2s;
}
.group:hover .drop {
	height: 27px;
	visibility: visible;
	opacity: 1;
	transition: 0.4s;
}
```

Notice that we have `height` going from 0px to 27px. That's all, add that and you will get a dropping menu animation! Also note that I changed the transition duration; one second is too long for this animation. I found that anywhere from 0.2s to 0.5s is acceptable, and you can vary the drop time and the collapse time however you like—like I did, with the menu dropping in 0.4s and snapping back up in 0.2s.

From there, you can change the styles as you like. If you change menu height, borders, padding, button shape, or any other variable, it will probably require adjustments in the code. You will have to work that out by yourself, depending on the button design you want to use. The border-top is used for this color & style of menu, it might not be used elsewhere.

DESIGN

10d. Style Point: Font Choices

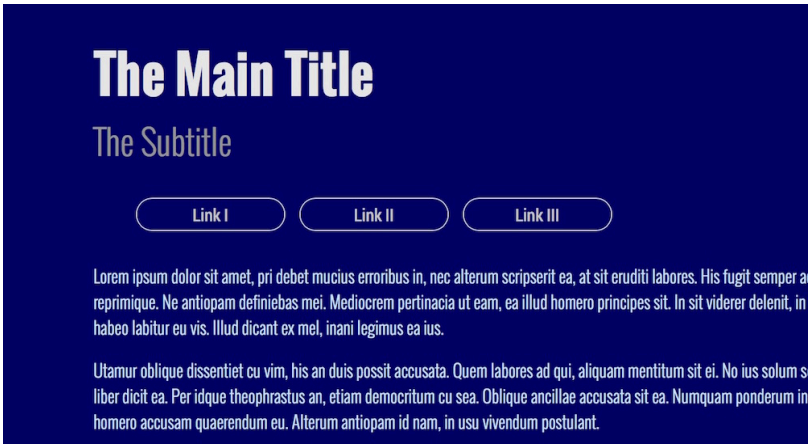
When you create a web site, you should pay close attention to which fonts you will use. Do not use fonts without thinking about topics like what your site is about, what impression you want to make, and what message you want to send.

As with most artistic decisions, there is no clear set of rules. The important thing is *how you use what you use*. Every decision you make should have a *reason* behind it; you should not make artistic choices randomly or without consideration.

HOW MANY FONTS

It is possible to use only one font on a page and make it work. It is also possible to use seven different fonts and make them work.

Lower numbers of fonts can be given variety and contrast by using different sizes, colors, and styles (bold, italic). In the example below, one font—Oswald—is used for everything. The main title is bold and a very light gray; the subtitle is thin (300 weight) and medium gray; the body text is also thin, and a bright light blue. Everything is Oswald only—one font!



Two fonts is a common minimum: one font for titles (h1, h2, etc.) and another font for paragraph text. More important titles can be larger and possibly a deeper color; blockquote text can be italic and slightly larger than paragraph text. The example below uses Raleway and Crimson Text fonts.

Three fonts could work, especially if one is used for the main title, another for the subtitle, and a third for the text. Below we have Abril Fatface for the `h1` title, Dancing Script for the `h2` subtitle and Cormorant Garamond for the text.

Things quickly become more difficult as you add more fonts. Much of this depends upon what is on your page. Do you have special elements? Do you have a medium-sized shape somewhere with special information? Do you have multiple columns? Do you have several images of very different styles?

Contrast

One basic rule is **contrast**: avoid using fonts which are similar to each other side by side. For example, if you use Garamond and Georgia right next to each other, they will look too similar. They may appear to be the same font, but there will be a subtle difference that may confuse the style of the page. Mixing serif and sans-serif, or slab serif and sans-serif often provides for a good contrast.

Too *much* contrast can be a bad thing, however. If you have six different fonts on the page and they are all different sizes, weights, styles, and colors, the page becomes a jumble. Using multiple fonts is often helped by having only one or two colors and sizes, which helps to unify the page design.

Very often, a very **subtle** (barely noticeable) contrast helps a lot. Instead of black text, use very dark gray (rgb value 30,30,30); instead of a white background, use very light gray (rgb 240,240,240). The difference is small, but has a big impact—even if people don't notice! Great contrasts can sometimes be a strain on the eyes; toning it down even just a little helps.

Consistency

Fonts should be used the same way throughout the web site. If you choose Garamond as the paragraph text, use it for paragraph text everywhere. If you choose Josefin Slab for the main title, use it for the main title on every page.

The same is true for all elements on a page: font sizes, colors, and weights, as well as background and other colors, image sizes, text and image alignment, and so on. Unless there is a very important reason, the same styles should be used the same way throughout the entire web site.

Which Style of Font

Display ("fantasy") fonts can be used only in titles; they look horrible in paragraph text, or sometimes even just in small sizes. Script ("cursive") text is often harder to read at 12 pt. size, especially in a paragraph with not much line spacing.

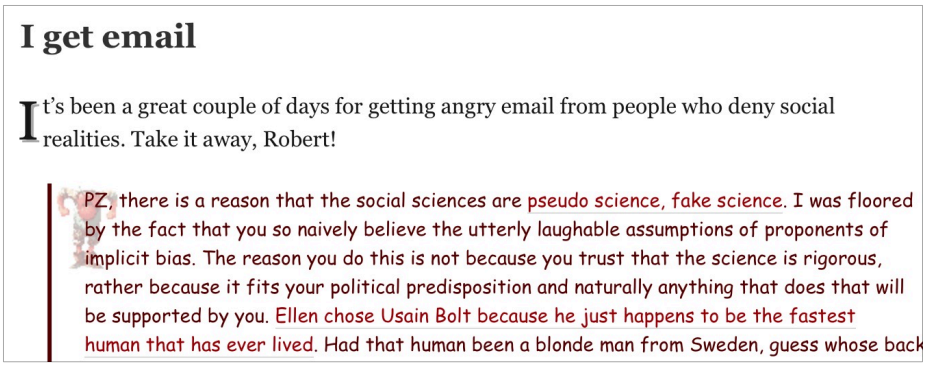
Serif and Sans-serif fonts usually work well almost anywhere. Classic serif fonts convey a sense of elegance; slab serif or sans serif fonts might give a sense of more strength or importance.

Historically, web sites have commonly used sans serif fonts for paragraph text, based on the idea that they are easier to read. This was perhaps more important many years ago, when screen sizes were smaller, and screen resolution was much worse than today. However, screen size and clarity is now so good that either type is fine for paragraph text; the choice depends greatly upon your sense of style and what the web site is about.

Remember also that many people have different *associations* with certain styles. For example, I have been involved in hiring people for many years. People send their resumes, and I must review them. I began to notice something interesting: the best teachers often used Garamond for their resumes. Now, this is not because using Garamond makes you a good teacher; however, the choice was significant. Garamond is an especially elegant font, but not too many people know about it, or about fonts in general. Choosing Garamond for the resume is like choosing a stylish outfit to wear to an interview. It shows that the person is aware that presentation is important, and they take the extra time to make the best choices, even in small ways that most people don't notice. Since I realized this, I pay closer attention to fonts used in such documents. Even if you are not fully aware, it still makes a difference.

The impressions made are not always good, however. In the United States, for example, there is a font called Comic Sans MS. It is a handwriting font, made to look similar to writing seen in comics. For a long time it was used commonly. However, it began to be used excessively in the wrong situations. Comic Sans is a playful, casual, child-like font—using it for professional purposes is considered a poor style choice. Additionally, the font was associated with people who loudly proclaimed extreme political or religious beliefs.

One popular science blogger, in fact, uses Comic Sans for a very specific purpose: he uses it in blockquotes whenever he thinks the person who wrote the text was not very intelligent:

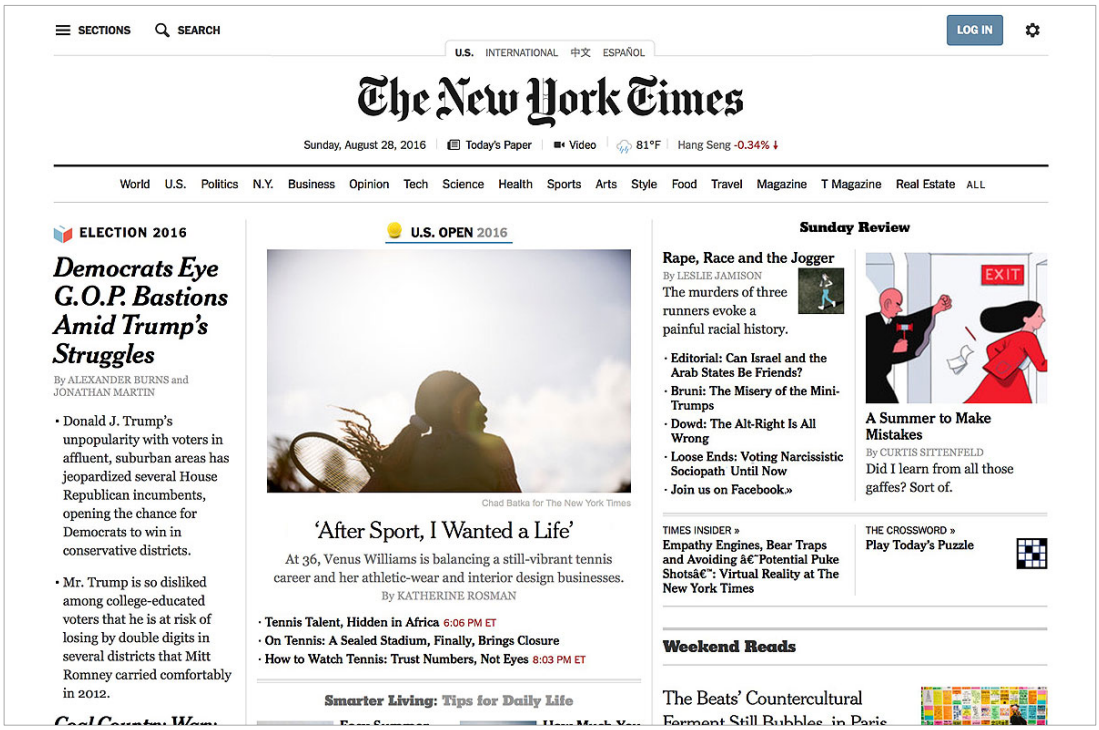


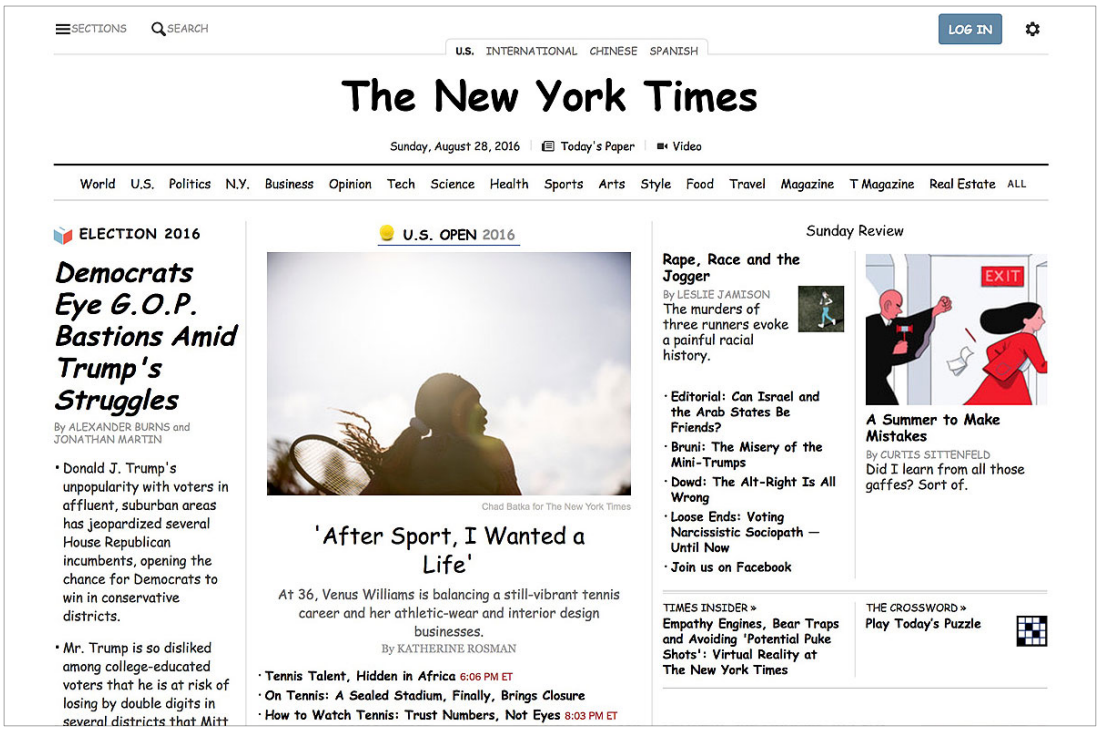
I get email

It's been a great couple of days for getting angry email from people who deny social realities. Take it away, Robert!

> PZ, there is a reason that the social sciences are pseudo science, fake science. I was floored by the fact that you so naively believe the utterly laughable assumptions of proponents of implicit bias. The reason you do this is not because you trust that the science is rigorous, rather because it fits your political predisposition and naturally anything that does that will be supported by you. Ellen chose Usain Bolt because he just happens to be the fastest human that has ever lived. Had that human been a blonde man from Sweden, guess whose back

Look at the web site for the New York Times pictured below. Then look at the next image, with the page's fonts changed to Comic Sans. The difference is hard to miss!





The normal version looks like a serious newspaper. The Comic Sans version looks like children's newsletter. You can see how using the wrong font at the wrong time can cause people to react badly.

Font Sizes

Like everything else, this is a matter of judgment. However, it is particularly important in paragraph text. The normal size is 12pt. However, it should be noted that some fonts are larger or smaller than others at the same point size. Bookman is a large font; Garamond is a small font.

For most fonts, 10pt is perhaps the smallest permissible size before the text becomes too small to read comfortably. You should probably not go below 11pt for most fonts, especially smaller ones. Bookman would definitely be a candidate for 11pt.

Similarly, you should not make the font size too big. 14pt is as big as you should get for a font like Times New Roman. At 16pt and larger, it begins to look like you are making the text oversized so it is easier for senior citizens to read it. This can create the wrong impression. If you use Garamond, a small font, 14pt may look normal.

What size and line spacing will depend on which font you choose. Do your best to keep sizes reasonable. To find what is reasonable, look at mainstream web sites and see what they do.

Chapter 10 Checklist

- ❑ Why is it useful to use drop-down menus?
- ❑ Why is "float" required for drop-down menus?
- ❑ Can you name three different ways to make something invisible on a web page? How are the three ways different from each other?
- ❑ Why is it necessary to give the drop-down buttons a relative position?
- ❑ CSS does not allow one object to affect another if the two have separate locations. What is the trick that allows you to get around that limitation?
- ❑ Did you make pages with all three different types of drop-down menus? Please try.
- ❑ What is the lowest number of fonts needed for one page?
- ❑ What is the highest number of fonts you can use on one page and not look ridiculous?
- ❑ How can you make one font look like many fonts?
- ❑ What is the best font size for paragraph text?
- ❑ What do I recommend as the smallest and largest font sizes for paragraph text?

Chapters 9 & 10 CSS Properties and Values

Property	Value	Purpose
list-style-type:	none;	Sets the style for list bullet/number
position:	static;	Object is in normal flow
position:	relative;	Object in normal flow can be moved
position:	absolute;	Object can be placed anywhere on page
position:	fixed;	Object placed anywhere & doesn't scroll
top:	50px;	Sets distance of object from top
bottom:	10px;	Sets distance of object from bottom
right:	2px;	Sets distance of object from right
left:	20px;	Sets distance of object from left
z-index:	100;	Sets "3D" layer of object
transition:	1s;	Sets duration of hover/active changes
opacity:	0.5;	Sets transparency of objects
visibility:	hidden;	Makes objects visible or not

Chapter 11: Programming for the Web

CODE

11b. A Little Fun with JavaScript

In addition to HTML and CSS, two other languages are commonly used in web pages: Javascript and PHP. These are programming languages, meaning that they do more than just create appearances and design—they interact with the user, taking input, processing the data, and giving output. Simply put, they don't just *show* things, they *do* things.

Both languages act in a similar way to CSS: they can be added to HTML **inline** (within the HTML code), or they can exist as separate files (like a CSS stylesheet) and be **linked** to the HTML file.

An important difference between the two is that Javascript is a **client-side** language, meaning that it is processed by the user's computer; PHP, in contrast, is a **server-side** language, meaning that the language is processed by the remote web server before the files are sent to the client browser.

In this class, we will not learn either language, as they are far too difficult for the time we have. Instead, we will simply use very small, limited bits of the language, so that you can get an idea for how they work.

One last note: **Javascript** is not related to another programming language called "**Java**." Their names are similar because the same company (Sun) was involved in both. However, the two are very different from each other.

THE DOM

An important part of Javascript is the "DOM," or "Document Object Model." This takes a web page *document* and divides it into **objects**. One part is the document itself; other objects include **tags**, **attributes**, **ids**, **classes**, and **text**. Even the browser window is treated like an object. Because they are treated like "objects," Javascript can change them—opening and closing windows, changing classes and styles, replacing text, and so on.

In simpler terms, Javascript has the ability to "grab" any part of the page and change it in various ways you might want.

Javascript can also react to certain **events**—things that happen on the page, and things that users do, including **mouse clicks** or **hovers**.

And important point here is that Javascript can make a button on one part of the page change an element somewhere else in the page—something that CSS cannot really do.

EVENTS

While using your code editor, you may have noticed some strange attributes. When you begin to type an attribute, editors offers a list of possible choices. Some of those choices may seem strange: onClick, onDrag, onDrop, onMouseOver, and so on. These are **events**. They indicate things that can happen on the page, to which Javascript can react.

Here are a few events which happen when using a **mouse** ("mouse events"):

`onClick`	when you click the mouse
`onDblclick`	when you double-click the mouse
`onContextMenu`	when you right-click to get a pop-up (contextual) menu
`onMouseDown`	when you press a button down
`onMouseUp`	when you release a button
`onMouseOver`	when you hover the cursor over something
`onMouseOut`	when the cursor stops hovering over something

There are other events as well:

Keyboard events	such as `onKeyPress`, when a key is pressed
Form events	on a form, such as `onSubmit`, when a form is submitted
Drag events	such as `onDragOver`, when something is dragged over an area
Clipboard events	such as `onCopy`, when you copy something to the clipboard
Print events	such as `onBeforePrint`, just before something is printed
Media events	for audio/video/images, such as `onPlay`, when a movie starts
Animation events	for CSS animations, such as `animationEnd`

There are more—events for touch screens, scroll wheel use, and much more. Remember, these event handlers are HTML element **attributes**. They can be added after a tag, and the **value** of the attribute can be Javascript.

CHANGING STUFF

Now let's actually try using these things. We'll learn two tricks: **alerts** and **changing the class of an object**.

THE ALERT

The alert has a simple structure:

```
alert('Well, hello there!');
```

Notice that the word "alert" is followed by something in (parentheses). This is a very common programming language structure, called a **function** with an **parameter**, which has the **argument**.

```
function(parameter);
```

The **function** is a pre-set mini-program; it carries out a task or set of tasks. For example, `rand()` is a function that will return a random number.

The **parameter** is in parentheses. It is an input for the function. For example, `rand(1,75)` will give you a random number between 1 and 75. If there is no parameter, a default is used.

The **argument** is the actual data (e.g., the numbers 1 and 75) sent to the function to be used.

So, in this case:

```
alert('Well, hello there!');
```

The `alert()` is a **function** which creates a dialog box, and the text inside is the **parameter**, which has the **argument** "Well, hello there!"

All of this can be the **value** of an HTML tag **attribute** such as `onClick`. Such as:

```
<div onClick="alert('Well, hello there!');"> Click me for a message! </div>
```

Important Note: notice that the parameter inside the function has **single quote marks**. This is necessary, because the attribute value is in **double** quote marks. If you use a double-quote inside the function, it will be seen as the end of the attribute value. Therefore, single quotes are used instead.

Another note: the Javascript Alert which you just learned is fun for beginners, but be warned that most people are incredibly *annoyed* when a pop-up dialog is presented. The user must suddenly stop what they are doing and dismiss the dialog box. Therefore, only use the dialog when there is a **reasonable purpose** to it—for example, if you want to give the user some text which they can copy and then paste somewhere. The web site visitor must feel that the alert box was useful, or else they will be unhappy with the extra trouble.

CHANGING A CLASS

One of the difficulties of CSS is that it doesn't *do* much. One of the only ways CSS is *interactive* with the user is with things like the **hover**. However, even the hover has a problem: it only works on itself. You cannot hover over one element and have a completely separate element change in some way.

We do tricks to get around this—for example, with the drop-down menus and the image galleries we have created. However, these are limited and a bit complex.

Javascript allows us to do a lot more, in a much more free way.

Here is one small example of that:

```
<div onClick="changeme.setAttribute('class','sky')">
```

In the above case, we are using the `onClick` event attribute. Therefore, when you click on the div in the web page, the "`setAttribute`" function will be used.

The `setAttribute` function will find an HTML tag by an id name, and then give that tag a class which you can define in CSS.

For example, on the next page is code which should work:

```
.sky {
	color: blue;
}

<h1 id="changeme">
	This text will turn blue.
</h1>
<div onClick="changeme.setAttribute('class','sky')">
	Click on me to make the Heading blue!
</div>
```

There are three different parts to this:

1. The style created in CSS
2. The tag to be changed, which is identified by the id attribute
3. The tag with the Javascript

The Javascript has this structure:

```
<div onClick="changeme.setAttribute('class','blue')">
```

The `changeme` is the id of the tag to be changed;

The `setAttribute` is the function;

The parameters are ([the attribute to be added] , [the value of the attribute])

The arguments are `'class'` and `'blue'`, meaning that class="blue" will be added to the tag.

All of this happens when you click on the div (`div onClick`).

The Result

The result is very nice: by clicking on one part of a page, you can affect another part of the page. This is much more powerful than plain CSS. And this is just the *beginning* of what you can do with Javascript.

In Conclusion...

I should point out that web designers don't usually use JavaScript as I show above. Generally, they avoid **inline** JavaScript (using JavaScript directly inside the HTML document), and instead link to JavaScript files. Nevertheless, this lesson has given you a very small peek at what JavaScript can do if you take the time and effort to learn it! Hopefully, after reading this and trying out some of the stuff, you will have a better idea of what JavaScript is—the DOM, events, functions, and so forth.

That said, here's one useful inline JavaScript nugget that will act as a universal "back" button:

```
<a href="javascript:history.back()">Go Back</a>
```

Using that as your link will send any visitor to the last page they came from, no matter where it was!

11c. PHP

PHP is a programming language that can work within HTML files (web pages). If you are familiar with the LUJ Bingo game, then you know what I am talking about.

> **An important note:** PHP is a *server-side* language, which means that it will *not* work on your computer, unless you have installed web server software. (This can be done on a Mac, but you need to follow some specific technical instructions.)

To insert PHP into a web page is easy—just do this:

```
<?

?>
```

That's it! You can also do it a little more specifically:

```
<?php

?>
```

Both do the same thing.

Anything that goes inside those two lines will be PHP programming.

However, the web page file must be named filename**.php**—and the file must be run on a web server—or the PHP will not work.

You can "type" things with PHP, using the command **echo** or **print**. Both do about the same thing.

```
<?
    echo "Hello, World!<br>";
?>
```

The above line will print out the words "Hello, World!" in the web page.

Three points:

1. Notice the semicolon at the end. Every command line in PHP must end with a semicolon.
2. Note that PHP can "write" HTML commands as well as text.
3. After PHP code is executed, it disappears from the HTML file. No PHP code is sent from the web server. (Of course, it stay in the original file.)

Remember:

1. PHP works on the web server. It finishes its work before it is sent to the visitor.
2. HTML tags are processed by your browser after they are sent by the server.

As a result, you can "write" HTML by using PHP:

```
<?
    echo "<p>This is put into the web page before it is sent!</p>";
?>
```

For example, let's say you put this into a PHP web page:

```
<p>Here is my first paragraph.</p>
<?
    print "<p>Here is a paragraph in between!</p>";
?>
<p>Here is my Second paragraph.</p>
```

When the file is sent, the HTML will look like this:

```
<p>Here is my first paragraph.</p>
<p>Here is a paragraph in between!</p>
<p>Here is my Second paragraph.</p>
```

After the PHP code is used, it is not sent with the HTML file. Only the HTML, CSS, and the text remain. Just be careful not to use any "double quotes" in the text you are entering, or else it will end the PHP string!

Programming languages are able to perform math operations. You can add those into your page:

```
<?
    print 3*5;
?>
```

If you do this, the number 15 will appear in your page.

You can also use **variables**. Variables being with a $, and should not have spaces.

```
<?
    $a = 3*5;
?>
```

Variable names are case-sensitive, so $a and $A are two different values.

Notice that in the above example, I did not use "echo." It still works—only the result will not be printed, at least not yet. If you want, you can do this:

```
<?
    $a = 3*5;
    print $a;
?>
```

That will do the math first, and then print it in the page.

Just like Excel, PHP has pre-set **functions**. These are algorithms which have been decided and prepared; all you need to do is use them.

For example, here is the function to get a random number:

```
<?
    print rand(0,100);
?>
```

That will print a random number between 0 and 100. You can set any two numbers you like.

There are some handy functions for math using binary, decimal, and hex:

```
decbin(255);        changes base 10 into binary      (1111111)
dechex(255);        changes base 10 into hex code    (ff)
hexdec(ff);         changes hex code into base 10    (255)
bindec(1111);       changes binary into base 10      (15)
```

There are several dozen math functions alone!

LOOPS & CONDITIONALS

Two of the most useful commands are **loops** and **conditionals**.

A loop will repeat an action a certain number of times, and then stop when a condition is met. For example:

```
<?
    for ($x = 0; $x <= 10; $x++) {
        echo “The number is $x!<br>;
    }
?>
```

In that code, the loop is set up in the first line:

```
$x = 0;             Sets $x as a number, in this case 0
$x <= 10;           Sets the condition: if $x is smaller than or equal to
                    10, then continue the loop
$x++                Add one to $x every time the loop runs
```

In this particular loop, the instruction is to print **The number is 0
** and then repeat. With every repeat, the number will grow by one, until the set limit is reached. In this case, the loop keeps going until the variable gets to 11.

A conditional will test if a situation is true or false:

```
<?
    $a = 5;
    if ($x == 5) {
        echo “True!”;
    }
?>
```

The above will result in the word “True” being printed. If you set $a to any other value, nothing will be printed.

Notice that in the conditional, there are two equal signs: == The two equal signs mean “is equal to.” There is a difference:

```
$a = 5       $a becomes 5 ($a changes)
$a == 5      $a is the same as 5 ($a does not change)
```

You can also say what happens if the condition is not true, using **else**:

```
<?
$a = 4;
if ($x == 5) {
	echo “True!”;
}
else {
	echo “False!”;
}
?>
```

The above would print only “False!”

You could also set multiple tests using **elseif**:

```
<?
    $a = 4;
    if ($x < 5) {
        echo “Less than 5!”;
    }
    elseif ($x == 5) {
        echo “Five!”;
    }
    else {
        echo “False!”;
    }
?>
```

Conditionals like these are commonly used in programs.

Okay, you may be wondering, “That’s nice, but until I can do *real* programming, how will any of this help me?”

Well, there are a few useful tricks you can do, with the **include** command and **date()** function.

Let’s say that there is some code you want to add to many files. For example, maybe there is a long and complex <nav> bar which is 40 lines of code, and it is exactly the same on 20 different pages. If you type it out on all pages, you will have to type (or copy and paste) 800 lines of code! That not only takes time to type, but it also takes up space.

Instead, you could just put the code into a file, and then use **include**.

For example, let’s say this is the repeating code:

```
<div>
	<p>
		Here is some code!
	</p>
</div>
```

Type ONLY that—no other HTML, nothing—and save it as an html file, for example, **extra.html**.

Then, in your PHP web page, type this:

```
<?
    include ‘extra.html’;
?>
```

This will find the file **extra.html** and insert that code into your page at that location! However, any page that uses the php code must be renamed with the .php extension and will only run on a web server.

One more nice feature is to put the date and time which someone visits onto your web page.

```
<?
    date_default_timezone_set("Asia/Tokyo");
    echo date('l, F j');
    echo "<br>";
    echo date('g:i:s a');
?>
```

The above will print the date and time.

It is required to first set the time zone, or else an error will happen. As a result, you may wish to specify what time zone is being used. For example:

```
echo "The time in Tokyo is ".date('g:i:s a');
```

In the above line, the period joins two echo statements.

TIME UNITS

When printing the date and time, the following letters have the following meanings:

l	lowercase “L”	Day of the week (full name)
D		Day of the week (shortened, e.g. “Mon”)
F		Month (full name)
M		Month (shortened, e.g. “Dec”)
d		Day of the month (as two digits, 01 to 31)
j		Day of the month (as one or two digits, 1 to 31)
Y		Year (4 digits)
g		Hour (12-hour format)
g		Hour (24-hour format)
i		Minutes (00 to 59)
s		seconds (00 to 59)
a		am or pm
A		AM or PM

You may insert punctuation and spacing as you like.

There is no checklist for this chapter, as it will not be on the tests.

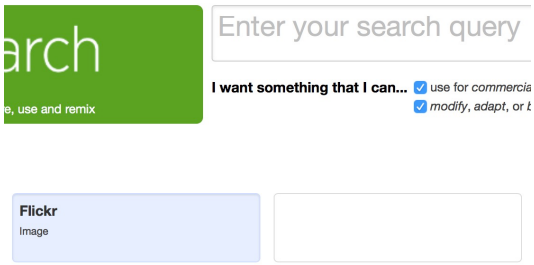
DESIGN

11d. Making Citations

Most of the images that you will be using will be borrowed—photos taken by other people, and made available with a Creative Commons license. As noted in Chapter 4, the best place to get images for your site is the Creative Commons image search engine, which will only give you images that have been shared in this way.

My preference is for the old search engine, found at this address:

`http://oldsearch.creativecommons.org`



When searching, I strongly advising using **Flickr**; it has a huge database of almost any kind of image, and you can be confident that they are legally shared.

Another source of free images would be **public domain**, which means there is no owner. Photos provided by the US government, for example, are usually public domain. NASA photos are, for instance, except when they are made in cooperation with the European Space Agency (which claims copyright).

Another source is **promotional** images, photo released by companies to promote their products. Even if the license is not clearly provided, I allow you to use them as they are intended to be spread around the web.

Careful, though: whether it is an open license, public domain, or promotional, **you must always give a full citation**.

When getting images from the web, you should make a note of four things:

1. The filename of the image
2. What the image is of
3. The name of the author
4. A URL link (hopefully short) which connects to the page where you found the image

All of these are available of Flickr, for example. Look for the arrow button at the lower right of the image display; it will give you a short URL. Short URLs can also be created using a service like **Bitly.com**.

When you download images, try your best to get larger images, with a resolution at least 1000px wide or more, so you can crop it as you like, and easily resize it to the ration and sizes you have chosen for your site.

Collect all your images **in a folder outside your project folder**. You do *not* want to send me a folder full of giant images every time you submit a project.

And again, be sure to keep a complete list of the images you get so you can make your citations.

Each image must have a note showing who the author is, and a link back to the original source. I will give you three basic methods for doing this; you can use any of these, or variations.

METHOD #1: CLUMSY BUT EFFECTIVE

The first method is fairly easy, but not very pretty. Just put a link around the image tag.

```
<a href="https://flic.kr/p/9dN1Dv" target="_blank">
    <img src="images/shiba02.jpg" alt="Shiba Inus" title="Image by Taro the Shiba Inu">
</a>
```

This will make the entire image into a link. In addition, notice the `title=""` attribute; that creates a small, pop-up "tooltip" label when the image is hovered over. The result looks like this:

This is a simple way to cite an image, but it is not the best; the image is too easy to click on, and the tooltip label can be distracting.

METHOD #2: FIGURE & FIGCAPTION

The second method is about as easy, using the figure & figcaption tags to add a caption, which can then include the link. The code may look like this:

```
<figure>
    <img src="images/shiba02.jpg" alt="Shiba Inus" title="Image by Taro the Shiba Inu">
    <figcaption>
        Image by <a href="https://flic.kr/p/9dN1Dv" target="_blank">Taro the Shiba Inu</a>.
    </figcaption>
</figure>
```

In this case, the link is in the text, so the image does not activate the link. That text also serves as a caption for the image, clearly presents as a link, and does not use a pop-up label.

With good CSS styling, this method has a cleaner look to it:

Image by Taro the Shiba Inu.

METHOD #3: IMAGE OVERLAY

The third method is similar to the second, but you use CSS to make the credit and the link appear inside the image. The HTML code is the same as Method #2, but the CSS moves the figcaption over the image itself:

```
figcaption {
    color: rgb(100,100,100);
    font-size: 11pt;
    font-family: 'Raleway', sans-serif;
    position: relative;
    bottom: 35px;
    left: 15px;
}
figcaption a {
    color: darkred;
}
```

This technique is the best for both style and layout, saving space on the page. You may not like having the text over the image, though; if that's the case, method #2 might be better for you.

Keep in mind that because you used the relative position, the figcaption still uses space below the image. You may need to include a negative bottom margin to compensate, such as:

```
figure {
    margin-bottom: -30px;
}
```

This technique gives the following look:

Notice that the font color must be balanced so that it looks good over the part of the image where it is placed. You may want to two or three class rules instead of the figcaption rule, so you can apply each style exactly where you need it.

Here are some alternate looks, which include text shadows:

One other alternate method would be to create a div tag, set the image as the background image for the div, size the div tag to be exactly the same size as the image, and then just have text inside the div, which will cover the image. The `position: relative;` declaration can be used to place the text anywhere inside the div.

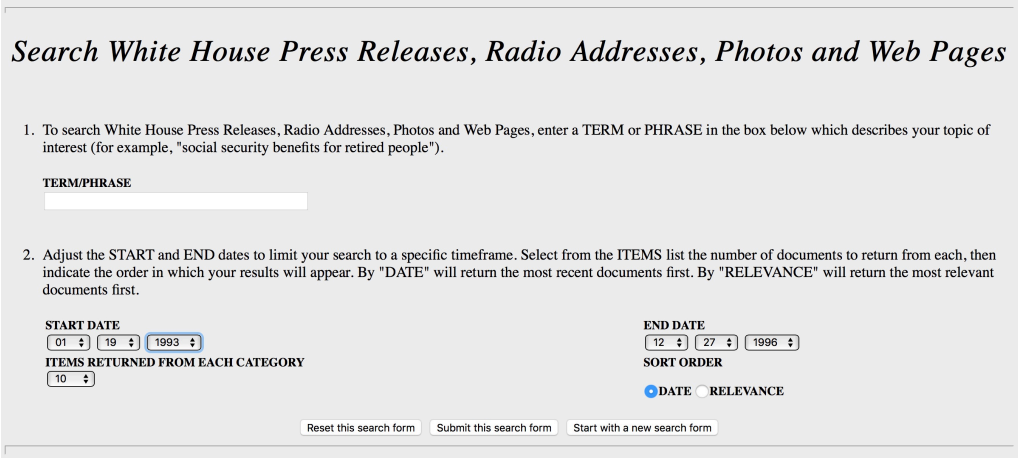
If you can create other methods, that may also be good. There are many possibilities.

Chapter 12: Tables

TECHNICAL

12a. Layouts

Before CSS became common, web page layouts had to rely on simpler forms. Many web sites from the 1990s used no real layout at all, which is to say they had a "fluid" page design.



Search White House Press Releases, Radio Addresses, Photos and Web Pages

1. To search White House Press Releases, Radio Addresses, Photos and Web Pages, enter a TERM or PHRASE in the box below which describes your topic of interest (for example, "social security benefits for retired people").

TERM/PHRASE

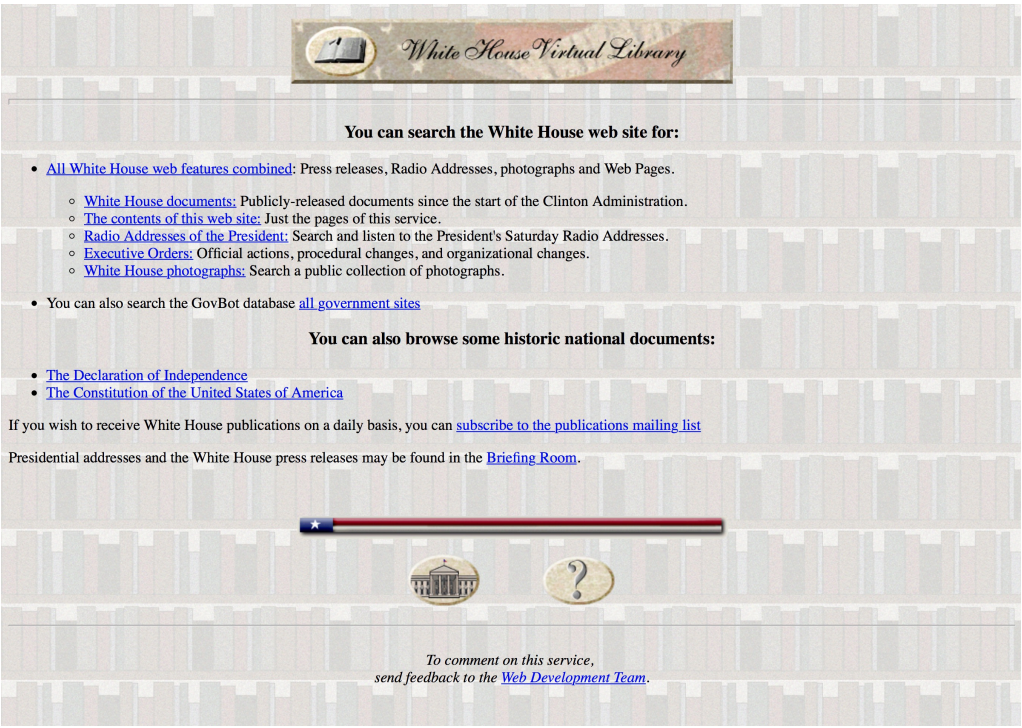
2. Adjust the START and END dates to limit your search to a specific timeframe. Select from the ITEMS list the number of documents to return from each, then indicate the order in which your results will appear. By "DATE" will return the most recent documents first. By "RELEVANCE" will return the most relevant documents first.

START DATE 01 19 1993 **END DATE** 12 27 1996

ITEMS RETURNED FROM EACH CATEGORY 10 **SORT ORDER** DATE RELEVANCE

Reset this search form | Submit this search form | Start with a new search form

While this style evolved to look more presentable, it was still the same basic layout—no headers, nav bars, or real columns.



White House Virtual Library

You can search the White House web site for:

- All White House web features combined: Press releases, Radio Addresses, photographs and Web Pages.
  - White House documents: Publicly-released documents since the start of the Clinton Administration.
  - The contents of this web site: Just the pages of this service.
  - Radio Addresses of the President: Search and listen to the President's Saturday Radio Addresses.
  - Executive Orders: Official actions, procedural changes, and organizational changes.
  - White House photographs: Search a public collection of photographs.
- You can also search the GovBot database all government sites

You can also browse some historic national documents:

- The Declaration of Independence
- The Constitution of the United States of America

If you wish to receive White House publications on a daily basis, you can subscribe to the publications mailing list

Presidential addresses and the White House press releases may be found in the Briefing Room.

To comment on this service,
send feedback to the Web Development Team.

The best way to format in the days before the `<div>`, structural tags, or CSS was to use **tables**. The entire page would be a table, and the parts of the page would be table cells.

Instead of a header, there would be a table with one wide cell at the top; instead of columns, there would be a row of cells of various heights. More detailed formatting would be created by using tables within tables.

In a table, it is possible to merge cells—the can be merged horizontally, to make a wider cell, or they can be merged vertically, to make a taller cell. In HTML, these are called "spans." Below is an example of a layout using tables in Microsoft Word:

<table>
<tr><td colspan="4">header</td></tr>
<tr><td rowspan="3">column</td><td rowspan="2">small column</td><td rowspan="2">small column</td><td>individual boxes</td></tr>
<tr><td></td></tr>
<tr><td colspan="2" rowspan="2">wide column</td><td></td></tr>
<tr><td></td></tr>
</table>

Sites like cnn.com would use these table-based layouts. Below is an example of a page from the year 2000, and the code used to make it. Notice the extensive use of tables, including the `<tr>` tag (to make a table row), and the `<td>` tag ("table data," to make a table cell):


```
<!-- static on cnn -->
<table bgcolor="#cccccc" width="340" border="0" cellpadding="0" cellspacing="0">
<tr><td colspan="2">
<table width="100%" border="0" cellpadding="2" cellspacing="0">
<tr><td width="5" bgcolor="#ff6600"> </td>
<td class="BoxTitle" bgcolor="#666666" nowrap><a href="/web/20000817204102/http://www2
class="BoxTitle"> CNN ON TV</span></a>: <a
href="/web/20000817204102/http://www2.cnn.com/ELECTION/2000/conventions/democratic/"><
Coverage</span></a></td>
</tr>
</table>
</td></tr>

<tr><td width="275" valign="top">
<table width="275" border="0" cellpadding="2" cellspacing="0">

<tr>
  <td class="BoxTxt" nowrap> <b>Watch Live on CNN today (all times EDT)</b></td>
</tr>
<tr>
  <td class="BoxTxt" nowrap>  &#149; 5 p.m.:  Inside Politics</td>
</tr>

<tr>
  <td class="BoxTxt" nowrap>  &#149; 6 p.m.: Gore Campaign Manager Bill
</tr>

<tr>
  <td class="BoxTxt" nowrap>  &#149; 9 p.m.: Tommy Lee Jones on <a
href="/web/20000817204102/http://www2.cnn.com/CNN/Programs/larry.king.live/">Larry Kin
</tr>


</table>
</td>

<td>
<table cellpadding="0" cellspacing="0">
<tr>
        <td valign="top"><a href="/web/20000817204102/http://www2.cnn.com/ELECTION/200
```

Tables, now an aged part of the HTML code, are no longer used in this fashion, but they still have utility. They are commonly used to present data in formatted table form, or, even more commonly, they hold labels and inputs used in **forms**.

In this chapter, we learn how to make tables, and in the next chapter, forms.

CODE

12b. Table Basics

Tables are blocks which have columns and rows—a boxes within boxes. Many years ago, tables were used to make the structure of web pages. When CSS advanced and HTML provided structural tags, we stopped using tables that way; now, tables are just tables.

THE STRUCTURE

To create a table, you must first make a table container. After that, you create table rows. After that, you can create table cells to contain data. The table cells are like columns, but they are created one cell at a time. A special tag can create bold and centered "header" cells.

There can be three sections of a table: the head, body, and foot. These sections allow different styles to be applied in CSS.

THE CODE

The table container is simply a "table" tag:

```
<table>
	|
</table>
```

Inside the table, you can create table rows, or <tr> tags:

```
<table>
	<tr>
	</tr>
	<tr>
	</tr>
</table>
```

Inside each row, you can add cells with the table data <td> tag. Table data tags create columns, one row at a time. In this case, we are creating a table with two rows and three columns. The result of the code is on the right (the blue highlighting comes from selecting the table).

```
<table>
	<tr>
		<td> cell content </td>
		<td> cell content </td>
		<td> cell content </td>
	</tr>
	<tr>
		<td> cell content </td>
		<td> cell content </td>
		<td> cell content </td>
	</tr>
</table>
```

cell content	cell content	cell content
cell content	cell content	cell content

Note that without any set width value, the width of cells and the whole table are set by the contents. The width of each column is set by the column with the widest content.

If you add a row with `<th>` (table header) cells instead of `<td>` cells, that row will be centered and bold:

```
<table>
	<tr>
		<th> head </th>
		<th> head </th>
		<th> head </th>
	</tr>
	<tr>
		<td> cell content </td>
		<td> cell content </td>
		<td> cell content </td>
	</tr>
	<tr>
		<td> cell content </td>
		<td> cell content </td>
		<td> cell content </td>
	</tr>
</table>
```

head	head	head
cell content	cell content	cell content
cell content	cell content	cell content

12c. Styling a Table

You can add various CSS styles to a table to make it more presentable. The first two you might want to create are table width and cell borders. Note that the table width is applied to the `table` tag, and the borders are applied to the `td` and `th` tags.

```
table {
    width: 500px;
}
td, th {
    border: 1px solid rgb(120,120,120);
}
```

You can also add a border to the table, creating a separate border surrounding it.

You can add background colors in three ways:

1. `table { background-color: yellow; }`
2. `tr { background-color: yellow; }`
3. `td { background-color: yellow; }`

If you color in the table, everything, including the space between cells, will be colored. If you color in the rows or cells, then the space between the cells will remain the page background color. If you give the table and the cells different background colors, then the cells will take on one color, and the space between the cells will be a different color.

head	head	head
cell content	cell content	cell content
cell content	cell content	cell content

You can increase the space between cells with the `border-spacing` property, which must be applied to the table tag; for example, `table { border-spacing: 6px; }` gets you the appearance seen at right.

head	head	head
cell content	cell content	cell content
cell content	cell content	cell content

Note that the increased border spacing does **not** increase the table width; instead, it subtracts from the cell (`td`) width.

Most web designers, however, prefer to have **no** spacing between table cells. To get this effect, you could use `border-spacing: 0px;`. However , if you do this, then all borders for all cells still remain, making double-borders between the cells. You probably want only 1px of borders between each cell. To get that effect, you need to use:

```
table { border-collapse: collapse; }
```

Notice that in this case, the table background color no longer appears, and all borders are the width of a single border (whatever you set the border width to). This is the preferred style.

head	head	head
cell content	cell content	cell content
cell content	cell content	cell content

Table Cell Sizing

You may wish to change the size of table cells. This can be done in various ways. Let's begin with height. If you set the `th` and `td` tags to have a certain height, this effect can be achieved:

head	head	head
cell content	cell content	cell content
cell content	cell content	cell content

Notice that if the cell content is smaller than the cell height, then the cell content is vertically centered. You can fix this by using the rule `td { vertical-align: top; }` (notice that I applied it only to `td`, not `th` cells). You can also add space between cell borders and text with the `padding` property.

You can also align text horizontally with `text-align: center;` or `text-align: right;` .

head	head	head
cell content	cell content	cell content
cell content	cell content	cell content

Up until now, I have shown you a table in which the content in all the cells is exactly the same. This is usually not the case. When each column has content of different widths, then the columns automatically re-size to fit the contents. Notice that each column will size itself based upon the cell with the largest contents.

head	head	head
cell	cell content	cell content is large
cell	cell	cell content is very much large

You can change this by setting the width of the `th` and `td` tags, or you can use the CSS style `table { table-layout: fixed; }` to make all cells of equal width.

head	head	head
cell	cell content	cell content is large
cell	cell	cell content is very much large

12d. Spanning Cells (HTML)

Sometimes you may wish to merge cells which are next to each other. This can be done with the HTML attributes `colspan` and `rowspan`. What they do is to make one cell larger, either horizontally or vertically. These can be a little difficult to understand, because in addition to making a cell larger, **you must delete the cells it replaces**. The example here has a `colspan` in the top row, and a `rowspan` on the left cells below.

head		
cell content	cell content	cell content
	cell content	cell content

A **colspan** makes a cell expand to the right. A **rowspan** makes a cell expand downwards.

To create the colspan which I show in the first ("head") cell above, you would need to replace three table cells with one table cell. This is the simpler task. The original code:

```
<tr>
        <th> head </th>
        <th> head </th>
        <th> head </th>
        </tr>
```

The new code:

```
<tr>
        <th colspan="3"> head </th>
        </tr>
```

This can be done with `th` or `td` tags. Notice that the value of the attribute is the **total number of cells** covered, and not just the number of cells *deleted*.

Creating the rowspan on the left side of the table shown above is a bit more tricky. Here, you will be *expanding* the cell in one row, but *deleting* a cell in the **next** row. The original code:

```
<tr>
        <td> cell content </td>
        <td> cell content </td>
        <td> cell content </td>
</tr>
<tr>
        <td> cell content </td>
        <td> cell content </td>
        <td> cell content </td>
</tr>
```

The new code:

```
<tr>
        <td rowspan="2"> cell content </td>
        <td> cell content </td>
        <td> cell content </td>
</tr>
<tr>
        <td> cell content </td>          ← cell deleted here
        <td> cell content </td>
</tr>
```

By increasing the rowspan value, you can take over more rows' cells, deleting the old ones.

12e. Table Sections (HTML)

You may wish to style parts of a table differently. To do this, you can add table sections, named `thead`, `tbody`, and `tfoot`. By themselves, these add no changes to the appearance of the table. However, they provide different tags which CSS can style, allowing for up to three different

```
<table>
<thead>
	<tr>
		<th> head </th>
		<th> head </th>
		<th> head </th>
	</tr>
</thead>
<tbody>
	<tr>
		<td> cell content </td>
		<td> cell content </td>
		<td> cell content </td>
	</tr>
	<tr>
		<td> cell content </td>
		<td> cell content </td>
		<td> cell content </td>
	</tr>
</tbody>
<tfoot>
	<tr>
		<td> cell content </td>
		<td> cell content </td>
		<td> cell content </td>
	</tr>
</tfoot>
</table>
```

By adding the following styling, you could change these sections separately.

```
thead th {
	background-color: rgb(255,230,200);
	border-bottom-width: 2px;
}
tbody td {
	font-family: 'Arial', sans-serif;
}
tfoot td {
	text-align: right;
	font-style: italic;
}
```

head	head	head
cell content	cell content	cell content
cell content	cell content	cell content
cell content	*cell content*	*cell content*

12f. Pseudo-Classes for Tables

These are similar to the `:hover` structure, but have a different purpose. They include:

`:first-of-type`	*apply change to first tag **of this type** in the parent*
`:first-child`	*apply change to first tag in the parent*
`:nth-of-type(2)`	*apply change to second tag of this type in the parent*
`:nth-child(2)`	*apply change to second tag in the parent*

The above will apply a style once only, at most.

First-of-type means that the first tag of that type within a parent block will get that style. So, if you have a table, and there are 5 `<tr>` rows, only the first one will get the style.

First-child means that the tag will receive the style *only if it is the first tag of any kind in the parent block.* For example, if you have a `<div>` with a `<blockquote>` first and then four `<p>` tags, that means that only the blockquote is a first child. In that case:

`p:first-of-type { color: red; }`	*the first `<p>` tag will turn red*
`p:first-child { color: red; }`	*this will not show any change*

The second one will not work, because the first `<p>` tag is a *second* child.

Nth-of-type(*x*n) will apply a style to every nth occurrence of a type of tag within a parent:

`p:nth-of-type(2n) { color: red; }`	*the 2nd, 4th, 6th etc. will turn red*
`p:nth-of-type(3n) { color: red; }`	*the 3rd, 6th, 9th etc. will turn red*

Nth-child(*x*n) will apply a style to every nth occurrence of any tag within a parent.

You can change which tag will come first, second or third:

`p:nth-of-type(3n) { color: red; }`	*changes begin on the third `<p>` tag*
`p:nth-of-type(3n-1) { color: red; }`	*changes begin on the second `<p>` tag*
`p:nth-of-type(3n-2) { color: red; }`	*changes begin on the first `<p>` tag*

Here is a table with `tr:nth-of-type(3n-2) { background-color: rgb(150,240,150) }`

One	Two	Three	Four
One	Two	Three	Four
One	Two	Three	Four
One	Two	Three	Four
One	Two	Three	Four
One	Two	Three	Four
One	Two	Three	Four
One	Two	Three	Four
One	Two	Three	Four

12g. Where Properties May Be Used

Here is a quick list of CSS properties which can be used with tables, arranged by the tags they can be applied to.

```
TABLE
     background-color
     background-image
     text-align
     border-spacing
     border ..........................  only creates border around whole table
                                        does not increase size of table
     border-collapse: collapse;
     table-layout: fixed;

TR
     background-color ................  fills all row cells, does not fill space between cells
     background-image
     text-align
     height
     vertical-align: top/bottom

TD, TH
     background-color
     background-image
     text-align
     height
     width
     border
     padding
     vertical-align: top/bottom
```

There is no checklist for this chapter, as it will not be on the tests.