Fundamentals of Web Design

Third New Edition

by Luis Poza

GDN103 • Fundamentals of Web Design • Lakeland University

CLASS TEXTBOOK **UNIT 1**, Chapters 1 ~ 4

Getting Ready

I. Software: Browsers

Browsers are software we use to view the World Wide Web. Web pages are written using code, and each browser reads and then **interprets** or **parses** the code to display it on a page.

However, there are many differences between browsers, and the same browser's different versions on Mac and Windows can differ in how a page is displayed.

As a result, you must install all possible browsers on your computer, so that **when you make a web site, you can check it on various browsers** to check for differences so you can fix them.

Macs come with Safari pre-installed; Windows comes with Edge installed. Viewing your work on browsers on both operating systems is best. You should add these browsers:

- **Chrome**: https://www.google.com/chrome/
- **Firefox**: https://firefox.com
- **Opera**: https://www.opera.com
- **Edge**: https://www.microsoft.com/edge
- **Brave**: https://brave.com

Make sure that each one is updated so it is the latest version.

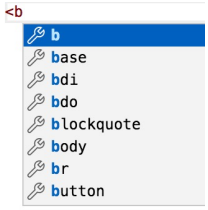
There are many more browsers available; just search for them if you would like even more.

II. Software: Visual Studio Code

You need a program which lets your write code. Almost any text editor *can* be used, but in this class, you must use **Visual Studio Code**. This is the best software for our purposes. It is available for Mac and Windows; it is free; and it has excellent tools to help the beginner.

There are many code editors out there, and many, like Visual Studio Code, are free. I will allow the use of a different editor only if (1) there are no "WYSIWYG" or automatic code-writing features, and (2) you get my direct approval *before* you begin using it.

Visual Studio Code has several important features, the most important being its auto-complete feature. This means that when you start writing commands, VSC will provide you with a list of commands and other code elements which you may choose from. After typing just a few characters, VSC can often auto-complete a whole command for you.

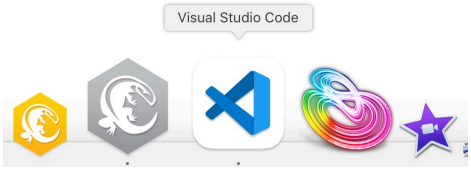


1. You can download VSC at this location: `https://code.visualstudio.com`

2. Once downloaded, you can unzip it (if it is zipped) and install it.

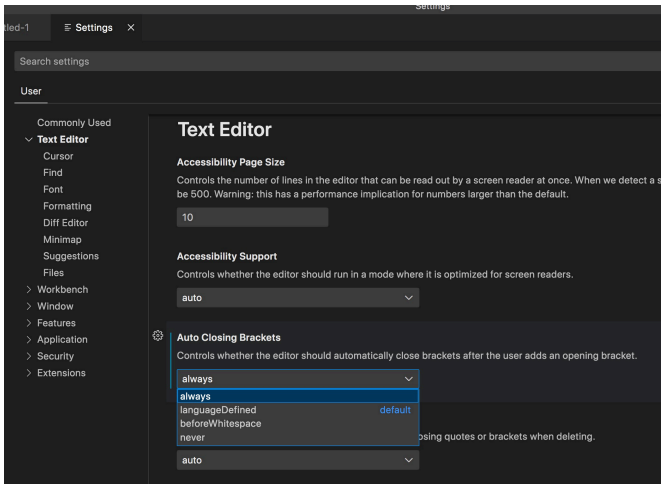
   On the Mac, just move the unzipped app from the Downloads folder to the Applications folder. On Windows, run the installer app.

   On both Mac and Windows, I would suggest pinning the app to the Dock / Taskbar for easy access.



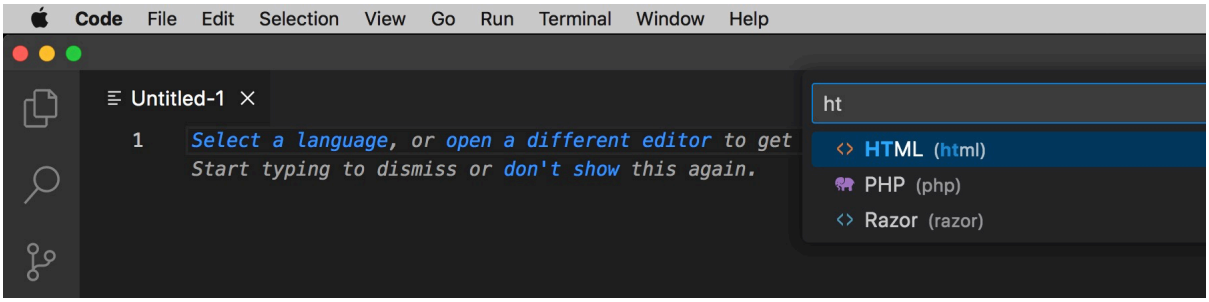
3. Open the app. If any tabs are open, you should close them to start with a blank window.

4. Go to the Preferences / Settings. Click on **Text Editor** on the left (do not click on the words below it). In the main pane area, go down to **Auto Closing Brackets**. Select **always**. That is the most important one; see other settings at the end of this section.

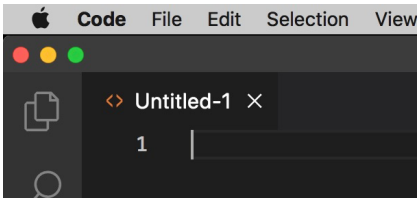


5. In the blank window, make a New Text File (Command+N for Mac, Control+N for Windows).

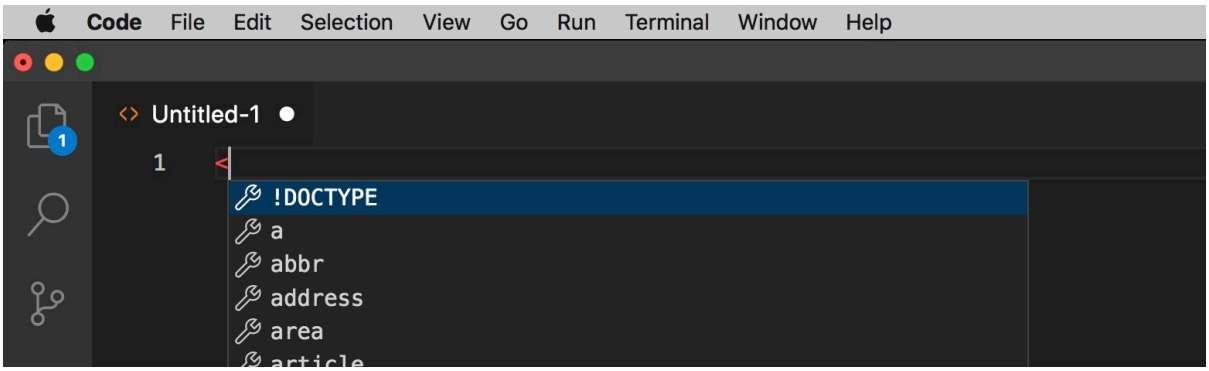
6. Click on the link to **Select a Language**. In the text box that pops up, type "**ht**" and hit the Enter key to select HTML.



7. The file will open and you will see the HTML brackets < > in the tab, and the file should be titled **Untitled1**.



8. Start typing! You will notice that the auto-suggest feature helps you with HTML commands and attributes.



9. When you want to save, just use **Save** or **Save As**, usually Command+S on a Mac and Control+S on Windows.

Whenever you begin working on VSC on **any new computer**, always check the Preferences for these settings. You can find them by going to the preferences or settings and then do a search for any specific setting.

A. In **Editor: Word Wrap**, change the setting to "on."

B. In **HTML: Auto Closing Tags**, check the box to enable.

I may add other settings as the class goes on.

III. Software: Filezilla

You will learn to create and test web pages on your computer. However, you will soon need to **upload** your web pages to a location on the Internet. Web pages are stored on computers running **web server** software. You will need to contact the web server, log in, and then see the folders / directories where web pages are kept.

To do this, you will need FTP (File Transfer Protocol) software. The software we use in this class is Filezilla.

Download Filezilla from: `filezilla-project.org/download.php`

The current version of Filezilla as of this writing is **3.64.0**.

If you download at home, make sure you get the **client** version of Filezilla. Open and run the latest version of the installer, using all of the default settings. You will be able to use the software exactly as it is installed.

Also, be sure to download the regular version of FileZilla; do **not** download the "Pro" version, which costs money. The free regular version will work fine for this class.

Open Filezilla. Dismiss the "Welcome" window if it appears. In class, if a window alerts you of an update, dismiss it. After testing Filezilla, you may close it and wait for the first use in class.

Later, we will log in; you do not need to do so now.

IV. Computer Settings

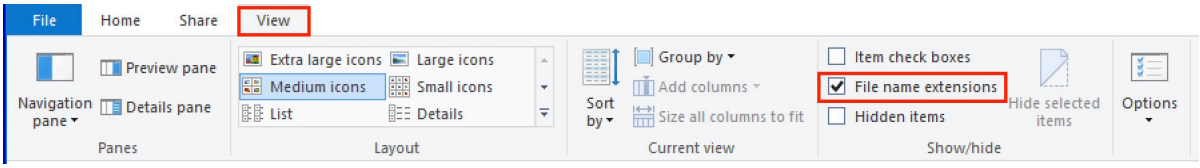
Please keep in mind that for this class, you must use a **laptop** or **desktop** computer. A tablet or phone is not acceptable. The needed software must run on Windows/Mac desktop systems.

FILENAME EXTENSIONS

A filename extension is the three- or four-letter add-on at the end of any filename. For example, a Microsoft Word document is *filename.**docx***; a web page is *filename.**html***; or an image might be *filename.**jpg***. When you create web pages, you **must** see and often type in the correct filename, complete with the filename extension.

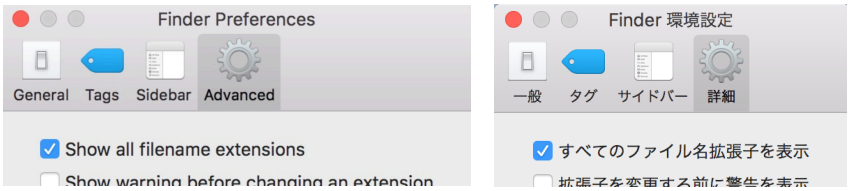
Normally, filename extensions are invisible, because changing them can lead to problems. Therefore, you will need to change the settings to make them visible. **You should always make sure that every computer you use be set this way**. If you do not, then you will probably have many problems with your web site!

How to make the filename extensions visible is simple. In Windows, open up any folder or file window on the Desktop, and in the **View** tab, you'll see the **File name extensions** option:



In Japanese, the ribbon may look like this:

On a Mac, go to the **Finder**; in the Finder Menu, choose **Preferences**. In the Advanced tab, check the box marked **Show all filename extensions**. In English and Japanese:



If the filename extensions do not show up immediately, turn the option off and on again, and see if that makes them visible.

Once again, **it is very important that you be able to see these at all times when you are doing web design!** Therefore, you must confirm that extensions are visible on **any** computer that you begin to use!

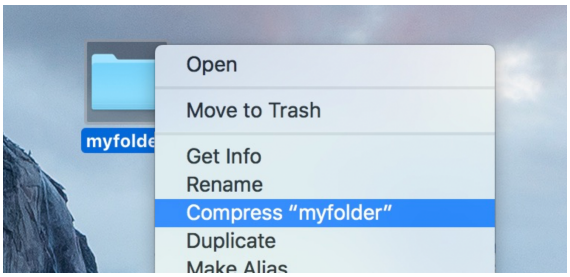
V. Sending Files

When you send files to your teacher, you might have to do it in a specific way.

If you have just one or two files, **simply attach them as normal files**. If there are more than three files, don't attach them all separately—that's too many and is difficult to handle.

If you send a whole folder, it is usually best to "zip" it, or in other words, **compress** it. This can be done on Mac and Windows without installing any special software. In fact, it works better if you *don't* install any special software!

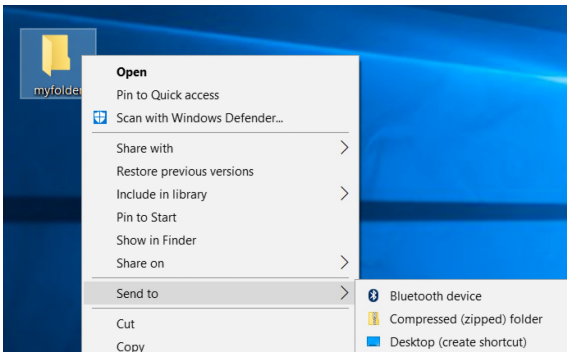
On a Mac, find the folder you want to send. Right-click on the folder. Select "Compress…" with the name of the folder.



A second later, a new file will appear on your desktop with the name of the folder and the filename extension **.zip** (the extension for zipped files)

You may attach the zip file and send it to me.

Using Windows, find the folder you want to send. Right-click on the folder. Select "Send to…" and then, in the submenu, select " Compressed (Zipped) folder."

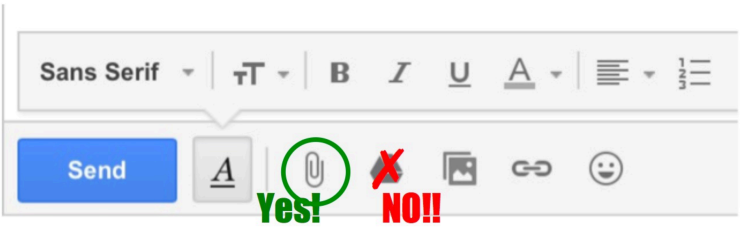


A second later, a new file will appear on your desktop with the name of the folder and the filename extension **.zip** (the extension for zipped files)

You may attach the zip file and send it to me.

Make sure to follow my rules about naming files / archives, as well as what you must put in the Subject line of emails.

If you use LUJ Mail or Gmail, and Google Drive, do **not** use the "Insert Files Using Drive" button! **Only** use the **Attach Files** option. You should always upload from your computer, if possible, without using drive. Do not upload the files into Drive and then share them—this causes problems.



VI. Organizing Files

During this semester, you will be sending and receiving a great number of files. If you do not keep them organized, then things can get difficult and messy.

First, in a place you often visit, like your Desktop, create a folder called **GDN103**. This will contain all the files for your class.

Inside, I recommend that you make the following folders as a second level of organization:

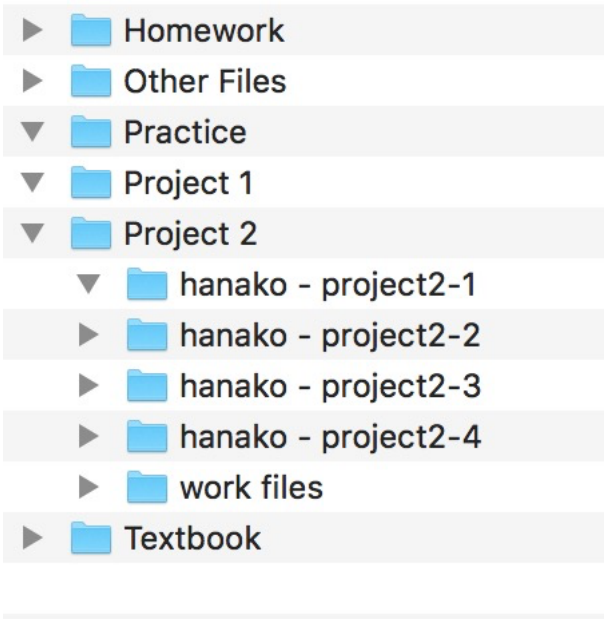
- **Textbook** — *I will give you PDF versions of the textbook and other information files which you should put here*
- **Practice** — *I will often send you folders with partially-completed files so that you can work on them as practice*
- **Homework** — *If I give you homework which is not a project, use this folder*
- **Project 1**
- **Project 2**
- **Other Files** — *Any other files and folders go here*

You can organize within these folders as you like, or add other folders.

Keep in mind that all of your projects will go through many versions. You may want to save each version in a separate folder within your project folder.

Doing things this way also allows you to search the GDN103 folder separately from the rest of your computer.

Below is an image showing how these folders might look. Keep your files organized, and the class will be a lot easier to do!



VII. Projects

You will create two web site projects for this class:

Project 1: A simple, 4-page web site with two levels (one main page and three sub-pages). This will exemplify your ability to use HTML and basic CSS. The topic will be local (LUJ, Ryogoku, or something close to your home), and *you can only use photographs that you have taken yourself*. This project will be uploaded to a **subdomain** after it is submitted

Project 2: a more complex web site with:

- Minimum of 10 pages (can be more)
- Three Levels (a main page, a category level, a specific-examples level)
- A drop-down nav menu
- One level with a sidebar (second column)
- At least one table
- At least one form which will send an email to you when submitted
- At least one JavaScript image gallery

This project will be uploaded to a specific domain name which you will receive for free. The domain name will have a **.me** top-level suffix (e.g., **lakelanduand.me**).

Both projects must have different general topics. For example, do not make Project #1 about Ramen Restaurants in your neighborhood, and then Project #2 about Sushi Restaurants in your neighborhood.

For project #2 you can use images from any source so long as:

- The images were marked as having a license which allows free use; or
- The images are promotional in nature; or
- You own the images; and
- Any images you did not take are marked with a credit to the author with a link to the web site you found the image at.

You are allowed to use text from other web sites so long as:

- The text makes sense, on each page and as part of the site;
- The text fits the topic of your site and page;
- You remove any extra marks like the superscript [4] from Wikipedia text; and
- You add a note at the end of the text which links to the [Source].

All projects must have:

- A color scheme;
- A set of fonts, mostly Google Fonts;
- At least two images per page;
- A fixed header & menu bar that do not change in form or position from page to page
- A menu bar that allows any page to be accessed from any other page

Fonts and font sizes must never differ between the pages, but you may change colors, image placement, and other layout styles such as columns.

Project Process

In each project, you must take these steps. Each step is a draft:

1. **Plan the site**. Choose your color scheme, fonts, and exact layout.
   Create a Site Map showing the pages, the links between them, and their main topics.
   Create a Wireframe which shows the layout for each level.

2. **Create a "template" page**. Although each level will have slightly different styles, most of the pages in your site will be similar to each other. The easiest way to start is to create one page which has all the basic design points for all pages in the site. The base page might be your main page, or a page elsewhere in your site. Use "dummy text," such as *Lorem Ipsum*, to understand how text looks on your site.

3. **Create "Levels."** After you create the base page, make one or two copies of it, and then make changes to each copy. These pages will represent different "levels" on your site, and will have small but distinct changes from the original. After you finish this, you should have one page showing each variation in layout and style for your site.

4. **Copy level template to create all the pages, and then add content**. Create as many copies of each level page as are required for your site. Add images and final text, which will complete the pages.

Each of the above four steps must be completed and turned in as soon as you can make it.

I will often make due dates for various steps of each project, **but I expect you to turn in revisions much more often**.

Each time you send me a new version of your work, I will make comments, point out errors, give you suggestions, and I might even provide you with resources. In essence, the version comments I make **are a part of the grading and review**. At the end of a project, I will give you a grade *but probably no comments* because I already will have made many comments on the various drafts that you send me.

Every time you send me a new version, do not just say "Please check this." I expect that you will **ask questions about how to do things** and **point out parts you cannot fix and ask for specific help**. The less you ask, the less I will help.

I might give four due dates per project, for the steps I outlined above, *but I expect that you will send me at least 4~5 versions of each project*, possibly more. The more you send, the more I will help. **You are not allowed to send me nothing and then give me a completed project near the final due date**.

Also, do not just make the exact changes that I suggest and nothing more—if you do that, then I am simply doing your project for you! Make sure that each new version *also* has all kinds of new work—your own corrections, style fixes, and additions.

It is also important that if I give instructions on how to fix something, then you send me a new version *which does not fix many of the problems I mentioned before*, then I will not offer new help. I will be very busy and will not have time to repeat instructions.

I will discuss each step and fill in the details which are not yet explained as the class goes on.

VIII. Plagiarism

Text & Images

As stated above, you are allowed to use borrowed text (both projects) and borrowed images (Project #2 only).

If you use borrowed text, it must be followed by a [Source] link which leads to the site where you found it. The text must be readable and must make sense in the context of your site.

If you use borrowed images, they must either be promotional or must carry a visible license (such as a Creative Commons license) which allows you to freely use the image. When you find each image, make notes of (1) the name of the person who owns the image, and (2) the URL of the web page where the image was found. You will be required to make a note somewhere giving credit and linking to the site. We will study various ways to do that later in the text. I allow copying text and images because this is not a course on writing or doing photography. It is a course on web design. So long as you cite, you're fine.

Code

This is very important, so make sure that you read and understand this.

I will not permit any code that was not original for any particular assignment or project.

Every project must start as a blank. You may now "borrow" the code or style from a previous project, a practice or exercise page, or from any other pre-existing page or site.

This may be a little confusing, as I teach you standard code, and give you some segments of code which I ask you to copy and use. This is like, in a writing class, when the teacher tells you to use an introduction, body paragraphs, and a conclusion; to use specific types of introduction techniques, or a thesis statement which is shaped in a certain way, or to use topic sentences, and so on. You must follow basic instruction—**but after that, you are expected to be original and creative**.

Many students think it is OK to copy and paste code from their previous work and use it in a new project. **It is not.** Many students believe they can take previous work—by themselves or others—and simply change the code to make a "new" page. **It is not.**

If I see you using recycled code, or copying and pasting any part of your code, I will have to cite you for plagiarism, and give you a 0% grade for the current assignment. I will furthermore require you to delete all work for that project and start over again.

If you have any doubt or are not sure if something is plagiarism, **ask me**.

I also **strongly** recommend starting your assignments early and avoid any delay in finishing each step. The most common plagiarism comes in the last week or two of class when a student has projects due, is too far behind, so they decide to take shortcuts.

I would remind you: it is actually pretty easy for me to see when you copied or stole code. For me, it is like listening to a recording which suddenly changes voice in mid-sentence. **Do not plagiarize!**

All work you do in this class must be your work, and it must be original.

In general, you should know the rules of plagiarism. Here are some of the specific issues that arise in this class. This is not a complete list of plagiarism offenses.

- Do not copy code from any site or page. **Learn how the code works**, and use it **in an original fashion on your page**. Do **not copy and paste from another source, ever.**
- Do not use code generated by any AI page. You must understand how the code works, not just copy what you see**.**
- Only use code we learn in class. If you have learned some interesting code from some other resource and you want to use it, **you must see me first to discuss it**. Never use advanced code without my approval. In addition, you must truly **understand** any code you use; you must **not** simply copy and paste it. If you use code, I will quiz you about what it means, and ask you to use it in a new way.
- Do not plagiarize your own work. Many people finish a project, and when they begin the next project, they simply copy files or parts of code from the previous project, and make minor changes. **This is not allowed. It is plagiarism.** Every single assignment or project **must be a separate work beginning with a blank page**.

  For example, if you are taking a Composition class, and you write a comparison essay between Tokyo and Osaka, can you then write another essay about the differences between Nagoya and Kyoto, using the previous paper but changing some of the words? That would never be allowed. Same for this class.
- Any project which uses borrowed material will be rejected, and you will be required to start all over again, with **no material** carried over from the previous version.
- You may not "help" or "get help from" someone else by copying their code or having them write it for you. It is not better if you copy code but then change some of the numbers. **Any** copying is plagiarism.
- Never use another person's code as a "template" or a "foundation." You must always begin from a blank page.

AGAIN, MOST IMPORTANTLY:

> **You MUST NEVER copy your own work. When you begin a new project, you MUST begin with a BLANK document with NO code. You MUST NEVER copy and paste text from anywhere else. Each project must be ORIGINAL; if I find that you have the same code from one project to another, I will treat that as PLAGIARISM.**
>
> *Within* a project, copying may be done; that will be explained in the project instructions. Just remember: each new project must begin with a blank document. You must write ALL of the code from nothing, at the start of the project.

The above rules are not all of the examples of plagiarism, but they are common ones I have encountered. Use your common sense, and when in doubt, ask.

I know that I am repeating myself sometimes o these two pages about plagiarism, but I want to stress how important this all is!

COPYING CODE

You might be a little confused about copying code: throughout this text, I give example code, and **it will be necessary to use much of it on your page**. For example, I will show you a technique to make drop-down menus on a web site. There are a few dozen lines of code that must be mostly copied into your sites.

You might ask: when is it OK to copy, and when is it not?

The answer is: (1) copy **only the absolutely necessary** parts that I show you, (2) make sure that **you understand any code** that you reproduce, and (3) you must **make original styling**.

When I give you the code for the menus, for example, I will give you the most basic code possible. If you remove a line from the code for the menus, the menus may not work correctly. You will have to use it. However, don't just copy and paste the whole thing exactly as it is in the textbook. Read the code, **understand** the code, and then write the code on your own in your site. Copy by hand, not by clipboard, and make sure you know what you are doing. If there are names or identifiers which can be changed, make them original.

Most importantly, **style the site** with your styling. Don't use the same heights, widths, colors, borders, padding, fonts, and so on. This will significantly change the code, and make it your own. For example, if I have a background color using the color code **235,240,210** then don't use that color. Choose your own colors based on how you want your site to look. Change all the variable sizes, fonts, and colors to get the exact look that *you* want, that matches your site best.

Getting Ready Checklist

- ❑ Did you install at least Chrome, Firefox, Opera, and perhaps other browsers?
- ❑ Did you successfully install Visual Studio Code and set the preferences?
- ❑ Did you successfully install Filezilla? (Note: installing, connecting or changing settings is not required at this time; we will use after midterms)
- ❑ Did you make filename extensions visible on your computer?
- ❑ Do you understand how to compress files into a ZIP archive?
- ❑ Do you understand how to attach files to an email?
- ❑ Do you understand the rules of plagiarism in this class? If you are unsure even a little, it is your responsibility to ask before doing any work!
- ❑ Do you understand the basics about the three projects?
- ❑ Do you understand the importance of keeping due dates?
- ❑ Do you understand why I ask for drafts of each project?
- ❑ Do you understand that with each reading assignment, you should open Visual Studio Code and practice making pages with the new code you see in the chapters?

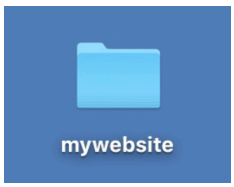
Chapter 1: Web Basics

TECHNICAL

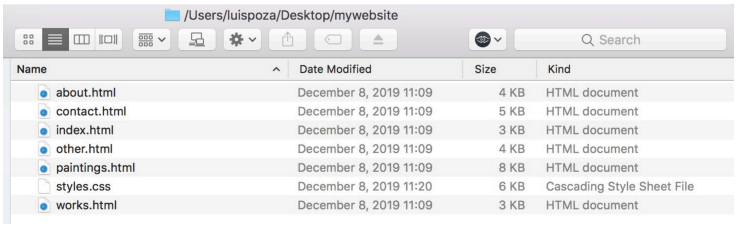
1a. Web Sites

Web pages are pretty simple:

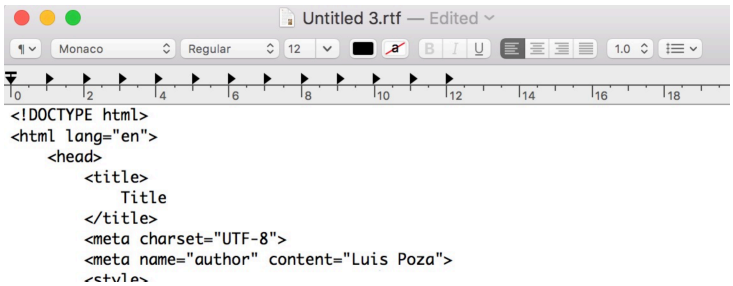
Fact #1: Web sites are just folders



Fact #2: Web pages are just text files with different filename extensions



Fact #3: You can make web pages with any word processor / text editor



Any web site must begin with a folder.

You will create **HTML files** in the folder; these will be web pages.

You will create web pages using a **code editor**, in this case, Visual Studio Code. (Although any text editor can make web pages, most are not suited for the job; a code editor like VSC makes it far easier to create your pages, which is why we use it in the class.)

When you make the web page, you will use HTML, or **HyperText Markup Language.** "Hypertext" refers to links. "**Markup**" means notes which indicate size, type, styles, etc.

WEB SITES

In computer terms, a **web site** is simply a folder on a computer. The folder contains web pages, images, and other resources for the site. A web *site* is a collection of web *pages*.

When we talk about a "web site," we are talking about a set of **web pages** which work together to give information based on a single thesis. For example, *Wikipedia* is a web site which holds an encyclopedia of general information. *Ars Technica* is a web site which provides news about technology. A blog is a web site with posts by one or more authors on a specific topic. However, all web sites center on one **topic** or **thesis**, no matter how broad or specific.

Web sites can have one page or thousands of pages.

Basic web sites can work on a normal computer; you can have a folder (web site) with files (web pages) and you can open and view them in a **browser**. Web designers commonly create web sites on their own computers before putting them up on the Internet.

However, it is unworkable for every web site to be on your computer. Instead, they are placed on computers connected to the Internet. When you want to access one, you type in its address and you are sent a copy. This is then viewed in your browser.

HOW WEB SITES WORK ON THE INTERNET

The **Internet** is a network of computers across the globe, all connected together. They are connected by wired and wireless means of transferring data, but more importantly, they are connected with the use of **protocols**.

A **protocol** is a set of rules used by participants so that communication and actions will be coordinated and understood. Think of each protocol as being the rules of a game, like a card game. If you do not fully understand the rules, you will not be able to play the game.

A protocol, in short, allows you to do a specific activity on the Internet. The **World Wide Web** is one of those activities that has a protocol. Email is another activity on the Internet, with its own protocols. Every online game has its own protocols. Every online chat and video has protocols. Every protocol represents an activity. All of these, including the World Wide Web, are parts of the Internet. The Internet is bigger than all of them.

Computers on the Internet are connected by the **TCP/IP** protocol set. TCP (Transmission Control Protocol) is a set of rules about how data is sent over a network. IP (Internet Protocol) is a set of rules about how each computer has an **IP address** so that it can be contacted.

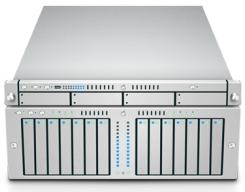
An example of an IP address would be 119.245.180.222. This is what is called an **IPv4** (Internet Protocol version 4) address. There are sometimes longer IPv6 addresses. These are the addresses that computers use to contact each other.

Another protocol is the one used specially for the World Wide Web. This is called **HTTP**, or **HyperText Transfer Protocol**. This protocol is used to make sure that all computers and browsers will work in the same way so web pages can be transmitted and seen the same way on all computers. You have probably typed "http://" thousands of times. That's the protocol.

Clients & Servers

These terms are used very commonly on the Internet. The names are modeled after how people use restaurants. When you use a restaurant, you are a customer, or a "client." When you order your meal, you ask a "server" (formerly a "waiter" or "waitress") to get your food. The same thing happens on a computer network: clients request data from servers.

A **client** is a program on your computer that requests data, receives it, and shows it to you, the user. A web browser, such as Chrome or Firefox, is a **web client**. An email program is a mail client, an FTP program is an FTP client, and so on.



A **server** is a program on a special computer on the Internet that has the data, receives the request from the client, and then delivers or "serves" the data to the client. You interact with such servers everyday, but they stay behind the scenes, so you would not be familiar with them.

When you type in a web page URL or if you click on a link in a browser, a **request** for that web page is sent to the address.

The address leads to a **web server**, a program which is on a computer (usually also called a "server"). The web server uses the HTTP protocol. Once it receives your request, the web server program collects the necessary files and sends ("serves") them back to your computer, the client.

MINI-REVIEW

So far, I have introduced several concepts. In summary:

Web pages are simple text documents which are written with HTML code. They exist in **web sites**, folders which contain a collection of related web pages. Web sites can be created and viewed on a personal computer which is not connected to the **World Wide Web**. However, web sites are typically stored on the Web. The Web is just one of many parts of the **Internet**.

The **Internet** is all the computers connected together in a worldwide computer network, which uses sets of rules called **protocols**. Any activity on the Internet has a protocol. One protocol, the Internet Protocol (**IP**), is used to provide addresses to every computer. Since these numerical addresses are hard for people to remember, we use **URL**s instead. A URL is based on a **domain name**. If you type in the domain name, using the **HTTP** protocol, you can be connected to the computer which has the desired web site.

Web sites are stored on the Web using a **web server**. A web server is a program which will serve web page data to anyone who requests it over a network, like the Internet. To request a web page, you use a **web client** (a **browser**, such as Chrome or Firefox) to type in the address. The client then contacts the web server, which serves the web page files; your client then receives and displays the files.

CODE

1b. HTML

HTML is not a programming language. A programming language is able to do a great many things, including accepting input and processing data.

HTML, instead, is a **Markup Language** (the "ML" in "HTML").

A Markup Language is much simpler than a programming language, and is intended only to create the shape of a web page. That markup language is used in a **web page**.

A web page is a single document within a web site.

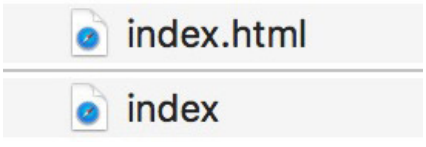
A web page is a **text file**, containing code and text. A web page can be made with almost any text editor, so long as it is saved in the correct format. The correct format is a **plain text file**, which is a text file without any special formatting. In short, the file has text only—nothing extra at all, just the text.

A plain text file normally has the **filename extension** `.txt`, which decides which program will open it. Filename extensions are usually added by the programs that save the document.

In your computer's operating system, there is a list of filename extensions; each extension is connected to a program. If you open a file with that extension in the filename, the assigned program will try to open the file. You can change the extension names. For example, if you have a Microsoft Word file with the extension `.docx`, you can change the extension to `.pptx`, and when you open it, the computer will try to open the Word file using PowerPoint. It will not work, however, because the file is for Word, and not PowerPoint.

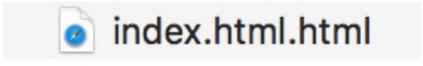
This is why filename extensions are usually hidden from users; most people do not know what extensions are, so they often delete or mistype them, causing files to open incorrectly.

However, when creating web pages, you need to know the filename extensions; you need to see them, so you can type the filenames fully and correctly, and sometimes you need to be able to change the extensions. That's why we turned them on in Part IV of "Getting Ready."



In the image at left, you can see the same file with the filename extension visible (above), and the extension hidden (below). Notice that they otherwise look the same. When the extension is hidden, *it is still there*, and any text added to the end of the filename will be placed before the extension.

If you don't have the settings done right, you might accidentally add ".html" when the extension is present but invisible, which would result in this filename:



And, no, that is not good. The first ".html" is not the filename extension. The second one is.

Once again, on *all* computers that you use for web design, make sure that filename extensions are made visible (see "Getting Started," Part IV, "Computer Settings").

TAGS / ELEMENTS

A web page is a text file. The text includes code and text.

In HTML, the code is in objects which are called **tags**. Tags are also called **elements**, and sometimes **commands**. (Tags, elements, and commands are all the same thing in HTML.)

Tags are always within angled brackets like this: `<header>`. In the following code examples, all the red text is code, and the black text will appear as words on the page:

```
Normal text. <b>Bold text.</b> Normal text.
```

In an HTML file, everything is either **HTML code** inside `<angled brackets>`, or else it is text (words) which appears on the page.

```
<body>
    <div id="wrapper">
        <header>
            <h1>
                This Is the Page Title
            </h1>
        </header>
        <main>
            <p>
                This is paragraph text.
            </p>
        </main>
    </div>
</body>
```

That code will look like this on a browser:

Notice that the different tags make the text look different. In this case, the `<h1>` tag makes the text larger and bold, and the `<p>` tag makes the text look like ordinary paragraph text.

NORMAL, VOID, AND DECLARATIVE TAGS

There are three basic types of tags: Normal, Void, and Declarative.

Normal

Most tags are Normal tags: they have a **beginning** `<p>` and an **ending** `</p>` tag. The end tag begins with a slash.

Everything between the beginning and ending tags is affected by them. One example of a tag is `<b>`, which means that text becomes bold. The beginning `<b>` tag shows where the bold text begins, and the end tag `</b>` shows where the bold text ends.

For example, if you have this code in a web page:

```
It is <b>really important</b> to lock the door.
```

It will look like this when seen in a browser:

> It is **really important** to lock the door.

This is actually what word processing used to be like in the early days. Some of the very first computers could not show things like font size, bold, or italic on the screen. To add these styles to text, writers had to add markup. This is very similar to what HTML is.

Void

Many other tags are Void tags: they do not have ending tags. These tags are for cases where nothing is contained within them. For example, a `<br>` tag is simply a command to go to a new line; nothing is contained within, so an ending tag is not required to define the contained text.

```
<br>                <!-- Creates a line break -->
<img>               <!-- Inserts an image -->
```

Tags for images, line breaks, page metadata, and more are void tags.

Declarative

There are only a few declarative tags. These tags begin with an exclamation mark and two hyphens. They also have no end tag. The only two we will use in this class are:

```
<!DOCTYPE html>     This declares the version of HTML (HTML5 in this case)
<!-- This is a comment and it will be ignored by the computer -->
```

The DOCTYPE is used at the start of every HTML file, to announce what version of HTML is being used.

The comments are a way to insert notes into HTML code; anything inside a comment will not be read by the browser. You can leave notes to yourself or others with comments. In my class, you can use comments to leave questions or explanations that you wish me to see.

Normal elements have a beginning and an end tag. The end tag begins with a forward slash. *Everything between the two tags is affected by them.*

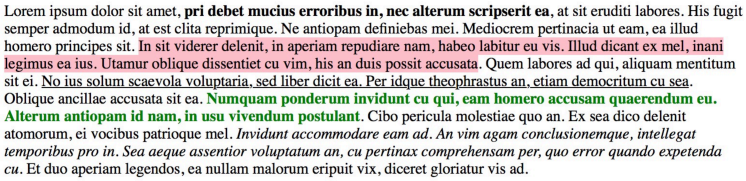
BLOCK AND INLINE TAGS

When you use tags in the body, they may not always be visible, but when they are visible, they fall into two basic types: **inline** and **block**.

Inline tags:

- include text formatting and images (text and images are referred to as **inline content**)
- appear on the same line as items (text, images) before and after them

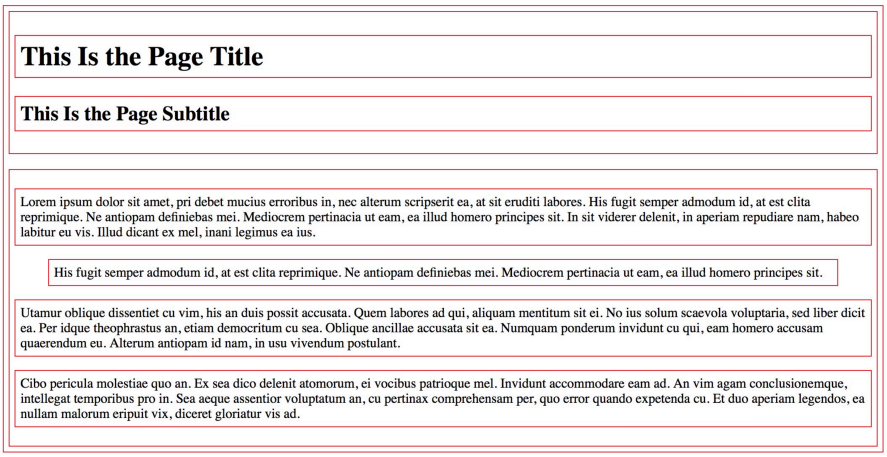
Below is an illustration of inline text with inline tags. Notice that the text simply goes from left to right, from top to bottom, and when formatting begins and ends (e.g., bold, coloring, underline, italics), it does not break the text and start on a new line:


Lorem ipsum dolor sit amet, **pri debet mucius erroribus in, nec alterum scripserit ea**, at sit eruditi labores. His fugit semper admodum id, at est clita reprimique. Ne antiopam definiebas mei. Mediocrem pertinacia ut eam, ea illud homero principes sit. In sit viderer delenit, in aperiam repudiare nam, habeo labitur eu vis. Illud dicant ex mel, inani legimus ea ius. Utamur oblique dissentiet cu vim, his an duis possit accusata. Quem labores ad qui, aliquam mentitum sit ei. No ius solum scaevola voluptaria, sed liber dicit ea. Per idque theophrastus an, etiam democritum cu sea. Oblique ancillae accusata sit ea. **Numquam ponderum invidunt cu qui, eam homero accusam quaerendum eu. Alterum antiopam id nam, in usu vivendum postulant**. Cibo pericula molestiae quo an. Ex sea dico delenit atomorum, ei vocibus patrioque mel. *Invidunt accommodare eam ad. An vim agam conclusionemque, intellegat temporibus pro in. Sea aeque assentior voluptatum an, cu pertinax comprehensam per, quo error quando expetenda cu*. Et duo aperiam legendos, ea nullam malorum eripuit vix, diceret gloriatur vis ad.

Block tags:

- create a rectangular box
- usually are on a line by themselves; there are line "breaks" above and below
- the box has no height, but has a width which is 100% of the available space; height is created when content is added

Below is an illustration of block tags, which I outlined in red to emphasize them (normally the outlines are not visible). The inline text is inside the block tags. Notice how all of them are rectangles, all on their own horizontal line, above & below each other, not next to each other:



Generally, **block tags should not go inside block tags** unless they were designed to do so. Do **not**:

- Put any block tags into header (`<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`) tags;
- Put any block tags into paragraph or blockquote tags;
- Put any block tags into list (`<ol>`, `<ul>`, `<li>`) tags.

Web Page Parts

Web page code has two main parts: the **head** and the **body**.

The *head* contains **information about the page**, and is mostly not shown in the browser.

The *body* contains the page **content** (text, images, etc.)—i.e., what you see in the browser.

The body has what people think of as "the web page," the visible web site. The body has certain **structures**; put these structures together in different ways, and you can create **layouts**— much the same way you could re-arrange furniture to get a different room layout.


```
<!DOCTYPE html>
<html lang="en">

    <head>
        <title>
            Title
        </title>
        <meta charset="UTF-8">
        <meta name="author" content="Luis Poza">
        <link rel="stylesheet" href="styles.css">
    </head>

    <body>
        <div id="wrapper">
            <header>
                <h1>
                    <a href="index.html">Site Title</a>
                </h1>
                <h2>
                    Site Subtitle
                </h2>
            </header>
            <nav>
                <ul>
                    <li><a href="life.html">Life</a></li>
                    <li><a href="universe.html">The Universe</a></li>
                    <li><a href="everything.html">And Everything</a></li>
                </ul>
            </nav>
            <main>
                <h3>
                    Page Title
                </h3>
                <p>
                    Lorem ipsum dolor sit amet, pri debet mucius erroribus in, nec alterum scripserit ea, at sit eruditi labores.
                        His fugit semper admodum id, at est clita reprimique. Ne antiopam definiebas mei. Mediocrem pertinacia ut
                        eam, ea illud homero principes sit. In sit viderer delenit, in aperiam repudiare nam, habeo labitur eu
                        vis. Illud dicant ex mel, inani legimus ea ius.
                </p>
                <p>
                    Utamur oblique dissentiet cu vim, his an duis possit accusata. Quem labores ad qui, aliquam mentitum sit ei.
                        No ius solum scaevola voluptaria, sed liber dicit ea. Per idque theophrastus an, etiam democritum cu sea.
                        Oblique ancillae accusata sit ea. Numquam ponderum invidunt cu qui, eam homero accusam quaerendum eu.
                        Alterum antiopam id nam, in usu vivendum postulant.
                </p>
                <p>
                    Cibo pericula molestiae quo an. Ex sea dico delenit atomorum, ei vocibus patrioque mel. Invidunt accommodare
                        eam ad. An vim agam conclusionemque, intellegat temporibus pro in. Sea aeque assentior voluptatum an, cu
                        pertinax comprehensam per, quo error quando expetenda cu. Et duo aperiam legendos, ea nullam malorum
                        eripuit vix, diceret gloriatur vis ad.
                </p>


            </main>
            <footer>
                <p>
                    &copy; 2020 Yourname
                </p>
            </footer>
        </div>
    </body>

</html>

```

Structural Tags

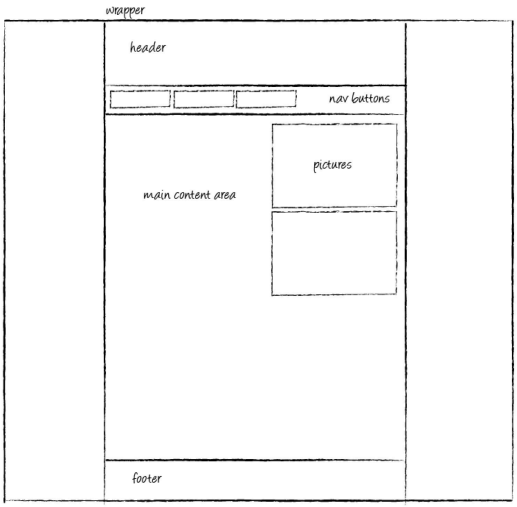
There are several structures, or areas, in which you can put the content on a page:

- **header** (contains the main title)
- **nav** (contains links to other pages in the site)
- **main** (contains the main information, text and images, of the page)
- **footer** (contains the copyright and other information about the page)

These are really block tags, but are used differently than most block tags. There are more parts which we will learn later, but these are the first **structural** parts you should know for now.

The body HTML on a page with structural tags may look like this:

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



The structural tags are inside the body.

All other tags fit into the structural elements.

Structural tags can go inside other structural tags.

Nothing should go between the structural tags, only inside them.

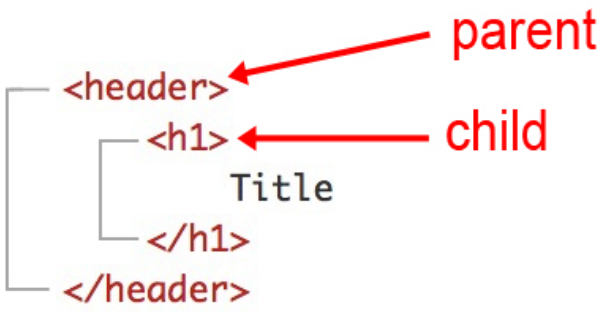
Important Note: As a general rule:

- **Inline** content (text and inline tags) must be inside **block** tags (images may be different);
- **Block** tags should be inside **structural** tags;
- **Structural** tags must be inside the **body** tag.

Parent and Child

In HTML code, a very important concept is one of *inheritance*. Inheritance, in common English, is where children receives the features or property of their parents. In this case, it is better to focus on the inheritance of physical features. You can *inherit* hair color, eye color, height, facial structure, and much more from your parents.

In HTML, a tag is a **parent** if it encloses another tag. If a tag exists within another tag, it is a **child** of that tag. For example:



In the case above, the `header` encloses the `h1` tag, and the `h1` tag is within the `header`. Therefore, the `header` is the parent, and the `h1` is the child.

Any tag that begins in a parent must end in the parent.

Children are represented by **indents**. If code is indented from the left, it means that it is the child of some enclosing tag. With indents, it is visually very easy to see these relationships.

The parent and child relationship is very important for a few reasons. First, it organizes the code very neatly by structure, making the code much, *much* easier to read. It shows which code is grouped together, in which parts of the page.

Second, the styling of parent tags very often controls the styling of the children. For example, if I make the `header` tag so it has a text color of red, the `h1` tag inside of it will have the same text color. This will become even more important when we get to CSS later on.

MINI-REVIEW

Here are the concepts on code which have been introduced so far:

- HTML files are plain-text files with the .html filename extension
- HTML code consists of **tags** (also called elements or commands)
- HTML tags are inside <angled brackets>
- There are three types of tags:
    - **Normal** tags have a beginning and ending tag; the ending tag starts with a slash; everything between the beginning and ending tags is affected by the tag
    - **Void** tags have no ending tag because they contain nothing
    - **Declarative** tags include the Doctype and comment tags
- Inside the body, there are inline and block tags:
    - **Inline** content refers to text and images; inline tags define inline content
    - **Block** tags are rectangles which break above and below, expand to 100% width of the available space, and have only as much height as they have content
- The two main parts of a web page are the head and the body:
    - The **head** contains *metadata*, information about and resources for the site
    - The **body** contains the actual web site content, what appears in the browser
- Inside the body, parts of the page are defined by **structural tags**:
    - The **header** contains the web site headings, the name of the site
    - The **nav** contains the nav menu buttons which link to all pages in the site
    - The **main** contains the main context of the site
    - The **footer** contains additional information about the site, including copyrights
- An important relationship in HTML is called **parent** and **child**:
    - A *parent* tag is a tag which contains other tags inside it
    - Tags inside another tag are considered *children*
    - This relationship is marked by indents, which neatly organize the code
    - Child tags often inherit the styles or characteristics of parent tags

1c. Editing & Viewing Code

I would like to introduction an important technique in web design: **side-by-side authoring & viewing**.

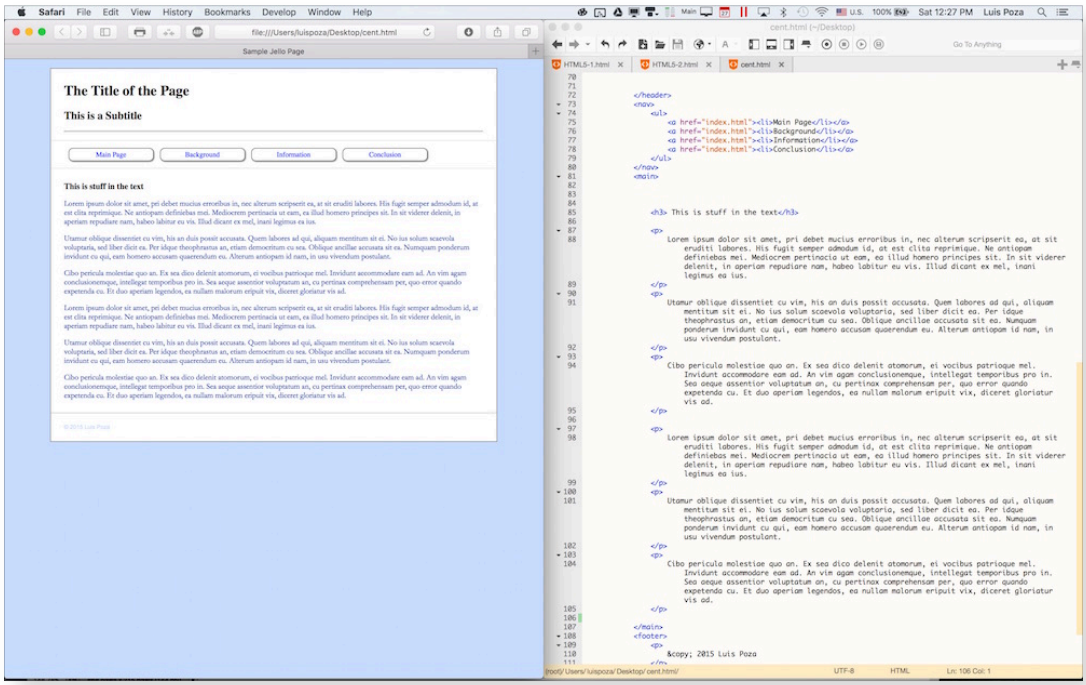
When you are creating web pages, you are only looking at the HTML code. However, this is not very clear sometimes; when you see the code, it might be hard—especially for a beginner—to imagine what the finished web page actually looks like. As a beginner, you don't know how the browser will render code. You also may make mistakes without knowing it.

Because of this, it is a good idea to constantly check the appearance of your web page on a browser. This means that you have to constantly switch back and forth between programs.

However, if your HTML editing app (in our case, Visual Studio Code) is taking up the whole screen, then you have to click on a small icon in the taskbar (Windows) or the dock (Mac) to change programs, or else use a different method of window-switching.

Also, when looking at the browser, you have to remember in your head what the code was, which can also be difficult.

It is *much* better to have both application windows open and visible at the same time, as shown here:



In order to do this, open both programs. Make sure the window for each is not full-screen. Then resize each window (browser and VSC) so it takes up half the screen.

On Windows, this can be done by dragging the window all the way to the right or left side; in something called *Aero Snap*, the windows will automatically resize to take up half the screen.

On a Mac, this must be done manually, by placing and then resizing each window.

Once you have done this, you can see the web page *and* the browser side by side. Now, just make sure that the same page is open in both applications, and you are ready.

Caution: sometimes, beginners make a mistake in which they open up different documents in each app, and are therefore confused when changes in VSC never appear in the browser. Make certain that you have the exact same file, in the exact same location, open in both apps!

Switching between the apps is incredibly easy: just click on the window you want.

Now, when you type new code in VSC, all you have to do is save it (I prefer to use the keyboard shortcut, CTRL+S (Windows) or Command+S (Mac). Then click on your browser and **reload** (also called **refresh**) the web page. To reload in Windows, it is easiest to simply press the F5 key at the top of the keyboard. On a Mac, use the Command-R shortcut.

When you refresh the page, the changes you made in VSC should appear. If they do not, then (1) you might not be looking at the same file in both apps; (2) you might have made an error in your code; or (3) you might have typed some code which made no visible difference.

Using this technique is valuable in understanding how code works, and in discovering and repairing errors (**debugging**).

DESIGN

1d. Layouts

There are three basic layouts for web pages:

- **Liquid**: uses up to 100% of the page width, text and images shift when the browser window is resized.
- **Fixed**: the page content is in a structural block with a set width, aligned to the **left side** of the page / browser window. The right side of the page is a blank margin which shrinks when the browser window is made smaller.
- **Jello**: the page content is in a structural block with a set width, aligned to the **center** of the page / browser window. The right *and* left sides of the page are blank margins which shrink when the browser window is made smaller.

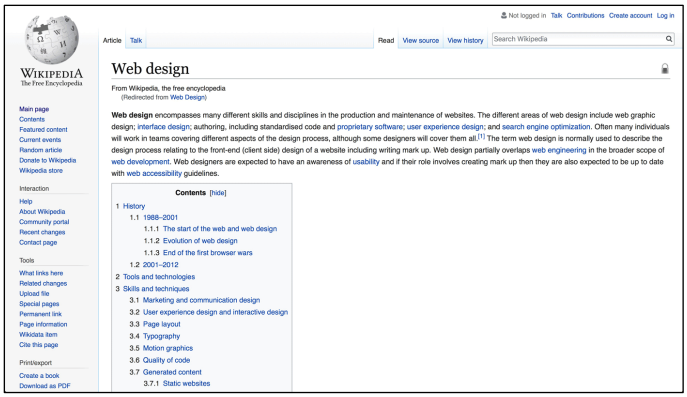
One of the major reasons for different layouts is to accommodate visitors with different screen sizes. Let's say that you have a content area set to a width of 1400 pixels. If someone visits your page and their screen is only 1200 pixels wide, then part of your content will be cut off. On the other hand, someone visiting the same page on a computer with a display 2800 pixels wide will see mostly blank space.

The two visitors may have a dramatically different view of your page, and content can even be obscured or hidden from view.

By using various layouts, you can make your page appear more similar between computers, with an improved general style that looks best on a wide variety of machines.

Examples of the layouts:

Liquid



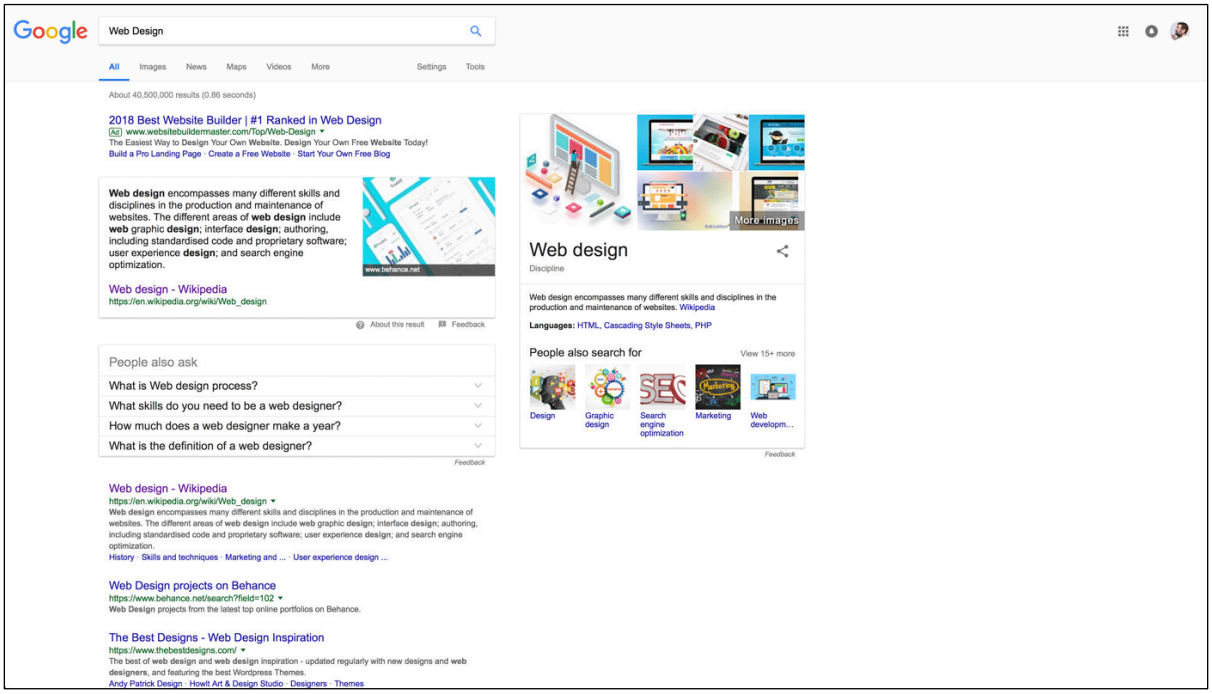
A good example of a liquid layout is **Wikipedia**. There is a sidebar at left with a fixed width, but the rest of the page is fluid, with no noticeable blank margins anywhere. If you make the browser window less wide (as seen in the image to the right), the text and other parts of the page change to fit the new space.

The advantage of a liquid layout is that it will fill up the screen with content, no matter what size the visitor's monitor is. A liquid layout may also appear better on smaller mobile devices.

There are two major disadvantages of a liquid layout, however. The first is that the inline content (images and graphics) will appear differently on different-sized windows, and the formatting could easily be distorted if you do not lay it out carefully.

The second disadvantage is that lines of text can appear far too wide on a larger monitor, making the text much more difficult to read. *Lines of text are harder to read when the width of the page is too long*.

Fixed



A good example of a fixed layout is **Google**, or really most search engines. They have the main content in a fixed-width area aligned to the left side of the screen (though technically, it could be on the right side as well, but it rarely is), creating a blank margin on the right side which appears smaller in smaller displays, and larger in larger displays. Note that in the above example, this leaves ample room for extra content to the right of the main content for a second column.

The advantage of a fixed layout is that your page content will appear the same way on any sized monitor, with the only difference being the size of the margin on the right side. Content (text and images) will always be the same in proportion to each other.

The disadvantage of this layout is that the right side could become a vast, empty white space on a larger monitor.

Jello

One example of a fixed layout is **The New York Times**, though most major sites tend to have jello layouts. They have the main content in a fixed-width area in the center of the screen, creating blank margins on the left and right sides, which appear smaller in smaller displays, and larger in larger displays. Doing it this way appears more balanced, as people like to have things centered. It also distributes the margin space to both sides, meaning each margin is less wide. This gives the feeling that the space is being utilized better.

The advantage of a jello layout is that your page content will appear more similar on any sized monitor. As the window space decreases all that disappears is blank margin space. This is more consistent between window sizes because the content is always in the center, and both margins always appear the same width. This layout is more natural because we tend to use equal left and right margins for most printed matter. Content (text and images) will always be the same in proportion to each other.

The disadvantage of this layout is that the margins on the sides could become larger on a larger monitor. However, because the space is distributed evenly between two margins, this disadvantage is less than it is with the Fixed layout.

The main danger is making the content area too narrow or too wide. If the content area is too narrow (say, 800 pixels wide or less), then your content could look extremely compressed on larger monitors. If the content area is too wide (say, 1200 pixels wide or greater), then the content area may be too wide to fully appear on smaller monitors.

Most people use displays 1280 pixels wide or greater, but most windows take up some horizontal space for scroll bars, and some people do not expand widows to the full width of their displays. **I recommend content width of 1000 pixels (between 900 ~ 1100 pixels)**. Advanced designers will use something called **responsive web design**, which is code that allows the same page to automatically adjust to a wide variety of screen sizes.

Exercise 1-1

As explained in "Getting Ready" Part VII, you will have two projects to complete. You must decide what the topics will be. Go back to page 8 and re-read the requirements for both projects.

For **Project #1**, I want you to create a completely self-made site. You will not be allowed to use any photos taken by anyone else.

Because you will be using only photos taken by you, the topic must be something close to you. If you don't leave your home town, make it something about that. If you go to a different place to work, perhaps make the site about something in that area. LUJ and Ryogoku are possible areas for a topic.

For **Project #2**, you must make a larger, more complete site. It must be a completely different topic from Project #1. It must be a very different style, and cannot use any of the same styles. The site will have its own domain name, and will be the main project for this class.

Below is a brief review of the main elements of both projects.

Go to the link that your teacher gives you to fill out the ideas for both projects.

PROJECT #1

Topic: LUJ, Ryogoku, or your home/work neighborhood

Minimum number of web pages: 4 pages

Levels: 2 levels

Requirements:

- There must be one main page, and then three 2nd-level pages below it
- The main page must have at least one photo, all other pages must have 2+
- All photos must be taken by you

PROJECT #2

Topic: Anything so long as it is aimed at an audience, not just for yourself

Minimum number of web pages: 10 pages

Levels: 3 levels

Requirements:

- There must be a drop-down menu that leads to all pages
- The site must include an image gallery, email form, and column layout
- All photos not taken by you must have citations and links

Chapter 1 Checklist

- ❑ What is a folder?
- ❑ What is a web page?
- ❑ Do you understand the difference between a web page and a web site?
- ❑ What is the Internet?
- ❑ What is a protocol?
- ❑ Do you understand the difference between a the Internet and the World Wide Web?
- ❑ What is the difference between a client and a server?
- ❑ What is a web client?
- ❑ What filename extension do web pages usually have?
- ❑ What are two alternate names for an HTML command?
- ❑ An HTML tag is put inside which punctuation marks?
- ❑ What is located outside the tags?
- ❑ If you see something like `</b>`, what is that?
- ❑ Do all tags have beginning *and* ending tags?
- ❑ What is a "void" tag?
- ❑ What happens to content (e.g., text) which is between a starting and ending tag?
- ❑ What do you call most text and images in a web page?
- ❑ Can you name the three basic features of a block tag?
- ❑ What are the differences between a block and inline tag?
- ❑ What are the two main areas of a web page?
- ❑ What does the “head” contain?
- ❑ In the body, what are the four basic structural tags that we learned?
- ❑ Can you explain the “parent / child” relationship?
- ❑ How is a child indicated in the code?
- ❑ Can you explain the differences between liquid, fixed, and jello layouts?
- ❑ What are the advantages and disadvantages of each layout?

Chapter 2: HTML Basics

To review: HTML is made up of **commands** called **tags** or **elements**. A tag is a command for the browser to show or do something. All tags are inside `<angled brackets>` ; anything outside of angled brackets will be presented as text on the page.

TECHNICAL

2a. Browsers & Rendering

Browsers are web clients. They request data (web pages and associated files) from web servers, and then they **render** the pages to create the visual representation that you know as a "web site."

When your browser receives the file, it begins to organize the tags. It recognizes all of the parent-child relationships. For example, it sees that the `<html>` tag is the overall container, and it contains the `<head>` and the `<body>`; it recognizes that the `<body>` is the parent of the `<header>` and `<main>` tags, and so on.

As the browser goes through the code, it figures out how everything should look—what margins, font sizes, widths, heights, background colors, and so on must be applied.

Initially, the browser only receives the plain HTML file; as the browser encounters additional files (such as an image reference), it will send another request to the server for these files. The server will then send these files to your browser, which adds them to the image it is building of the web page.

All of this work is done by the **browser engine**, which is software designed to render web pages for the browser. Different browsers use different engines, and as a result, some web pages will render differently—and will appear differently—on different browsers. Usually those differences are very small, but at times, they can make or break a page. This is why it is important to have as many different browsers as you can install on your computer, so you can see how your site looks on each different browser.

It also helps explain why some web sites will not work well or at all on some browsers; if a page is not loading on your browser, try the same page in a different browser.

- **Gecko** is the engine used by Firefox
- **Webkit** is the engine used by Safari
- **Blink** is a version of Webkit, and is used by Chrome, Opera, and Edge

Browsers have changed engines a number of times over the years. For example, Opera used to use its own "Presto" engine, but abandoned that and used Blink along with Chrome in 2013. Edge began with its own "EdgeHTML" engine, but in 2018, also switched to Blink.

If you develop web sites as time goes on, you will probably want to keep track of these changes.

CODE

2b. Attributes

HTML tags sometimes need extra information. For example, <img> is the tag to insert an image. However, the computer needs more information than that. You must at least say **which** picture will be inserted. You do this by giving the name/location of the image file.

This is done with an **attribute**. An attribute describes what is being added. The attribute includes a **value**. The value gives the information. The style is always like this:

```
<tag attribute="value">
```

You must never forget to (1) add a space before the attribute, (2) use the equal sign between the attribute and the value, or (3) put the value in quotation marks.

The attribute appears after the tag. It only appears in the beginning tag of a **normal** element, or in a **void** tag. The attribute never appears in an end tag. For example:

```
<img src="ponta.jpg">

<span class="myclass">This text will have unique styling.</b>
```

2c. Your First HTML Tags

Here are several of the basic tags used in the first web page you will create. Notice that two tags here require attributes.

Beginning	Ending	Purpose
<!DOCTYPE html>		Declares that HTML5 is used
<html lang="en">	</html>	Begins and ends the page
<head>	</head>	Begins and ends the head
<title>	</title>	Sets the document title
<meta charset="UTF-8">		Allows all languages to be used
<meta name="author" content="yourname">		Notes who is the author of the page
<body>	</body>	Begins and ends the content area
<header>	</header>	This will hold the site title/subtitle
<nav>	</nav>	This will hold the nav menu
<main>	</main>	This will have the main content
<footer>	</footer>	This will show info about the page
<h1>	</h1>	This is the main page title
<h2>	</h2>	This is the main page subtitle
		Everything inside this is a paragraph

What are these tags?

<!DOCTYPE html> sets HTML version 5 as the document code type. The DOCTYPE declaration is only at the start of the page, with no ending tag. It must be included on every page.

<html> is the tag which encloses the whole web page. It essentially says, "This is a web page." The attribute (lang="en") sets the reading language of the page. While the page will work without this, you must still use it. HTML is very forgiving, and will still show a web page even if you make some mistakes. However, you must still use all of these parts and create the web page according to the rules for the best performance.

`<head>` is the tag which encloses the "head" section of the web page; the head contains much information about the page, including the title, text encoding, author identification, and links to outside resources.

`<title>` is the text that will appear in the browser tab or title bar when this page is displayed.

`<meta>` is a void tag which can be used for various purposes. Here, we use it for deciding the **charset** (character set). The character set decides which text language will be used on the page. If you do not set this, then non-English languages may not display properly. Currently, most web pages use **UTF-8**, the Unicode character set which includes 146 scripts systems from all languages.

The meta tag can also be used to note the name of the author of the web page; it can provide a summary of the page that search engines can display; and it can help set the appropriate display sizes for images and text when a page is viewed on different screen sizes. More on these uses will be explained later in the course.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>
            Title
        </title>
        <meta charset="UTF-8">
        <meta name="author" content="Taro Lakeland">
    </head>
```

a typical beginning for web page code.

`<body>` is the tag which contains the actual web page information which is displayed in the browser window.

`<h1>` ~ `<h6>` are "heading" tags, which serve as titles, subtitles, section titles, etc. for the parts of the page. All headings are bold and have top & bottom margins. The `h1` tag is the largest, the `h6` tag is the smallest.

`<p>` is the paragraph tag. Every paragraph of text should be in a paragraph tag.

```
    <body>
        <h1>
            <a href="index.html">Site Title</a>
        </h1>
        <h2>
            Site Subtitle
        </h2>
        <h3>
            Page Title
        </h3>
        <p>
            Lorem ipsum dolor sit amet, pri debet mucius erroribus in, nec alterum
                scripserit ea, at sit eruditi labores. His fugit semper admodum id, at est
                clita reprimique. Ne antiopam definiebas mei. Mediocrem pertinacia ut eam,
                ea illud homero principes sit. In sit viderer delenit, in aperiam repudiare
                nam, habeo labitur eu vis. Illud dicant ex mel, inani legimus ea ius.
        </p>
    </body>
</html>
```

Headings and paragraphs in the body of a web site.

EXAMPLE CODE

Here is how that code is used in a web page:

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <title>
               This is the title
          </title>
          <meta charset="UTF-8">
          <meta name="author" content="Luis Poza">
     </head>
     <body>
          <h1>
               This is the main heading (title) for the web site
          </h1>
          <h2>
               This is a heading subtitle
          </h2>
          <p>
               This is a paragraph of text
          </p>
     </body>
</html>
```

The DOCTYPE establishes the use of HTML5;

The **html** tag begins the web page, setting the general language as English;

The **head** tag contains information about the page:

> The **title** will be seen in the browser tab or title bar, and is expected to be the name of the web page (though could be the name of the whole web site);
>
> The **meta** with **charset** tells the browser which set of characters to use. UTF-8 will allow the page to display all necessary languages;
>
> The **meta** with the name **author** will announce who made the page—when you make pages, your name should appear here. Be careful to write this correctly, as it is confusing; the **name** attribute should have a value of "author," not your name!

The **body** tag establishes the area which will display the actual web page content;

The **h1** and **h2** tags are *headings***,** which should not be confused with *head* or *header* tags. The h1 should be the name of the entire web site (not this page only). The h2 is usually the subtitle for the site's name.

The **p** tag contains a single paragraph of text.

Every page in a site will have this code. (More, actually—we'll learn that soon.)

2d. Indents and Code Formatting

In HTML, **it is very important if a tag is contained inside another tag**. This is the **parent-child** relationship I mentioned earlier. Tags containing other tags are called **parents**. The tags inside are **children**. Without formatting, it is hard to tell:

```
<html><body><header><h1>This is the title</h1><h2>This is a
subtitle</h2></header><main><p>This is a paragraph of
text</p></main></body></html>
```

As you can see, that looks more like a line of junk than anything else. If we add paragraph breaks, then it becomes a *little* more clear:

```
<html>
<body>
<header>
<h1>
This is the title
</h1>
<h2>
This is a subtitle
</h2>
</header>
<main>
<p>
This is a paragraph of text
</p>
</main>
</body>
</html>
```

However, that is still not very clear; you have to look carefully to see what is a parent and what is a child.

Therefore, we must also add **indenting**, as shown here:

```
<html>
    <body>
        <header>
            <h1>
                This is the title
            </h1>
            <h2>
                This is a subtitle
            </h2>
        </header>
        <main>
            <p>
                This is a paragraph of text
            </p>
        </main>
    </body>
</html>
```

This is *much* more clear! We can see that the text is inside the heading and paragraph tags, which is a child of the main tag, which is a child of the body tag, and so forth.

If you do not use indents correctly, your code is not clear, and you will lose points!

2e. Debugging, Part I

Before too long, you will be making mistakes in your HTML code. Everyone does.

Browsers, the programs that display web pages, are designed to work as well as possible, *even if you make mistakes in the code*. In other words, you may make errors in your HTML but the web page displays OK even though there are errors. This is both good and bad. It is good because it means your web page is working even though you made mistakes. However, it is bad, because you probably will not *notice* that you made mistakes—and therefore, you will continue to make the same mistakes!

Still it is possible to make mistakes that even the browser cannot save. Your web page will suddenly look wrong. It is up to you to find the errors and fix them. How do you do that? Below are three common types of errors you may have problems with:

1. Punctuation.

This is one of the most common types of errors. See if you can find the mistakes:

```
He said that he was <b>really<b> hungry.
<meta charset="UTF8">
<html lang=en>
```

In the first error, you would notice something is wrong because the text does not stop being bold. In the second error, you would notice if you used foreign-language or special characters and they did not appear correctly on your page. In the final error, you would probably not see anything wrong with the page in a browser—but it *is* an error, and *must* be corrected.

2. Spelling.

This is another of the most common types of errors. See if you can find the mistakes:

```
<titl>My Cool Page!</titl>
<meta charset="UFT-8">
<html lang="eng">
```

In the first error, the title for the web site would not appear in the browser tab; instead, the title text would appear on the actual web page. In the second error, again special characters would not display correctly. And a spelling error in the third example again might not have an immediate, easy-to-see effect on the page, but again, it is an error and must be fixed.

3. Missing Parts.

Sometimes you just forget to end a normal tag that must have an end:

```
<p>This is a paragraph of text.
```

This kind of error is more common when the start and end tags are located far apart from each other. Always check to see if you are missing the end tag!

DESIGN

2f. Web Page Standards

When you make a web page, you should choose elements of design carefully, and not at random. For example, colors and fonts should be decided very carefully. Don't just choose the first ones that come to mind, or the first you encounter that seems okay. Look at all possibilities, and think about why the element matches and contributes to your site. Choose page elements that work best with your concept.

One way to learn about this is to examine web sites that you see. Find a variety of web sites that seem nicely designed to you, as well as sites that might have a similar topic or theme as your idea for a site. Based on the following design points, examine those pages and find out how they do things.

Layout

First, should you choose liquid, fixed, or jello? Each one has its good an bad points. Sometimes you might choose based on how much information you want to be visible (liquid might be best), or you might want to choose a layout which displays your page similarly on different-sized screens (fixed or jello). See what professionals use, and how often.

Structure

Most web pages should have a header (with the main heading and sub-heading for the site); there will be a nav bar somewhere, with link buttons that allow you to visit the main pages of the site (possibly all pages); and at the bottom there will be a footer, which has at least copyright information, but also often a list of staff, contact information, a site map, or other information about the site. These three areas should be **identical** on all pages in the site.

<h1> Link

On almost every web site you find, the site title, logo, or main graphic should be a link to the home page of that site. Normally, there is no "Home" button in the nav menu; instead, the main heading for the site (h1) should be that link. This is a strong standard, and people will expect the link to be there.

What Kind of Font Is Used for Paragraphs?

Two categories of fonts are acceptable for character text in paragraphs: serif and sans-serif. Serif is the kind of type you are reading right now; it has little lines and bulges at the end of each letter's main stems. It is considered more elegant and beautiful (often used in fiction books), but can be harder to read. Times New Roman, Garamond, and Georgia are examples.

Sans-serif is a text design without serifs—just plain lines to make each letter. This category looks less interesting, but is easier to read, and thus is used to express important information, like instructions or warnings. Arial, Helvetica, and Verdana are examples.

In print, serif is the standard choice. Books, newspapers, and magazines mostly use serif fonts.

For a long time, however, sans-serif was considered the standard choice for web pages because, in the early days of the web, computer monitors were very low-quality, and serif lettering was especially hard to read on them. However, in recent years, the quality of monitors has increased to the point where there is no problem reading either type, so serif fonts are now acceptable—but sans-serif has been such a tradition by this time that many web sites use it simply because that's how it's been done for so long.

Check the web sites you use, and see if they use serif or sans-serif fonts.

Do you see rounded corners? Box Shadows? Hovers?

These are all special effects which can add nice styling to your page.

For example, all shapes on a page start is rectangles; however, it is possible to round corners in various ways, even to the extent of making circles. You can also add shadows to text and shapes. And finally, it is possible to add an effect where, if you hover your cursor over an object (like a button), it may change in color or in some other way.

There is actually a word for the overall look involved: **Skeuomorphism**. That means that a design is made to look like a physical object with height, shading, texture, or some other effect that makes it look like a "real" object. However, skeuomorphism has gone out of style recently, and more pages tend to deal in solid colors and flat designs. As always, it remains a matter of style and personal choice.

Do sites that you visit use these effects?

Visit at least ten web sites, and take notes on how these design points are handled on those sites. Avoid web sites with extreme designs.

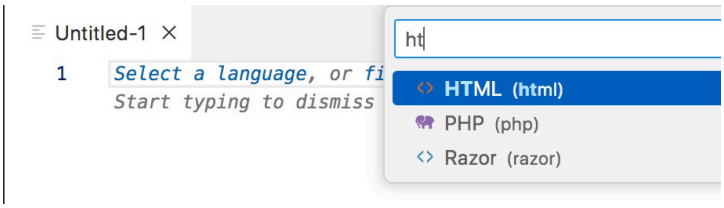
Exercise 2-1

For your homework over the first weekend, I would like you to **create two web pages**. I will give you detailed instructions for both. You should have followed all the steps in *Getting Ready* before you do this, especially the first two steps. You will need a browser, and you will need Visual Studio Code.

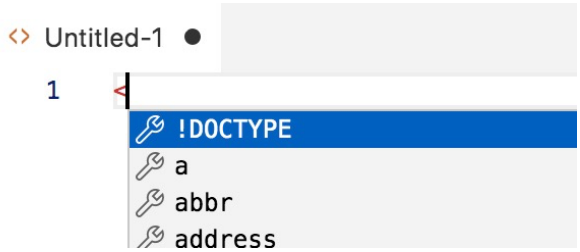
First, make sure that VSC is set up properly. If you are using the computer you are on for the first time, make sure that the steps to set up VSC Preferences has been completed.

DOCUMENT #1

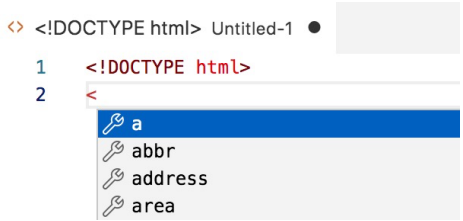
1. On a computer which has Visual Studio Code installed and set up according to the instructions, **create a folder**. The folder must have your name, all lower case, with an underscore _ between your first and last name. In my case, it would be `luis_poza`
2. Open **Visual Studio Code**.
3. Make a **New File**.
4. In the new document, click "Select a language" and type "**ht**" into the search bar.



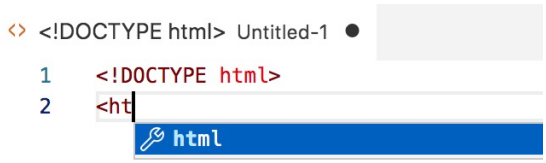
5. Click on "**HTML**".
6. On line 1, begin typing, starting with a **left angled bracket** <



7. You will notice that a little menu appears. This shows the possible tags you can use. Click on !DOCTYPE.
8. VSC will complete the whole declarative tag for you, including the "html."
9. Hit the Enter key to go to the next line. Notice that the new line is not indented. Indents should happen only after typing a **normal** tag (with a start and end tag) when you hit the "Enter" key between the two tags.
10. On the new line, again begin typing, starting with a **left angled bracket** <



11. Continue by typing the letters "ht", like: `<ht`



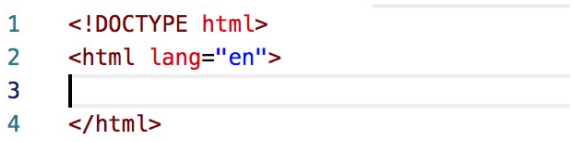
12. Type the closing angle bracket `>`

13. Notice that VSC will automatically complete the end tag for you! Note that the cursor is between the start and end tags.

14. Hit the **Enter** key again. *VSC will automatically indent for you!* The end tag will be pushed down two lines; a new, blank line will appear between the start and end tags, with an indent for the child tags you will type.

15. In the beginning `<html>` tag, click before the right bracket.
Add the attribute `<html lang="en">`. Notice that the assist menu only comes up after you type the first letter of the attribute. Complete the attribute.

16. Notice that if you click away from the middle indent and click back to it, the indent may disappear. That is an odd feature of VSC. If you hit the tab key in the space where the indents used to be, the correct number of indents will reappear.



17. Inside the `<html>` tags, add the `<head>` and `<body>` tags. In both cases:

When finished, the code so far should look like this:

18. Click inside the `<head>` tag at the indent and add a `<title>` tag. Make any title you want.
19. Add the tag & attribute `<meta name="author" content="Your Name">` after the title, inside the head. Use your name, not the text `"Your Name."`
20. Add the tag & attribute `<meta charset="UTF-8">` after the title, inside the head. This tag correctly displays text in multiple languages on the web page. Some browsers will correctly show multiple languages without this, but always use it anyway.

```
2   <html lang= en >
3       <head>
4           <title>
5               My Kewl Page
6           </title>
7           <meta name="author" content="Luis Poza">
8       </head>
9       <body>
```

21. Next, click inside the `<body>` tag.
22. Add the four structural tags seen on page 22 in Chapter 1. They are the `<header>`, `<nav>`, `<main>`, and `<footer>` tags. In this exercise, the nav and footer will be empty.
23. In the header, add an `<h1>` and an `<h2>`; the `<h1>` will contain the web site title, and the `<h2>` tag will have the web site subtitle. Add title text to each of them.
24. In the main tag, add an `<h3>` and put the page title in that.
25. After the h3 tag in the main, add three `<p>` tags; type a few lines of paragraph text inside each.
26. Click on the Save button in the toolbar, or go to the **File** Menu and click on **Save**. In Windows, you can see the File menu by pressing the ALT key.
27. Navigate to the folder you made in Step #1.
28. Change the file name to **index.html**
29. Click "Save." You can open the page in a browser if you want to see it!

This exercise is continued on the next page!

DOCUMENT #2

30. Open **Visual Studio Code**, or return to the app already open.
31. Click on the **New File** button again.
32. This time, the file should open up as **Untitled-2**.
33. This time, save the document right from the start; save it in the same folder as before. Give this second file a different filename: call it `creative.html`
34. Type the DOCTYPE, html, head, and body the same as you did last time.
35. In the `head`, create a `title` and an author `meta` tag.
36. Also create a charset meta tag: `<meta charset="UTF-8">`
37. In the `body`, add the four structural tags again (`header`, `nav`, `main`, and `footer`).
38. In the `header`, add only an `<h1>` tag, and add text to it. The text inside should be the site title. Do not add an `<h2>` tag in the header.
39. In the `main`, add an `<h3>` tag, and add a title for the page inside the tag.
40. After the end of the `<h3>`, add two paragraph tags, and fill them with several lines each of English text.
41. Add a third paragraph **with Japanese text**. You can type the text directly, or copy and paste it from somewhere else.
42. Save the page (Control + S, or click the Save button).
43. Check your exercise folder. Make sure both assignment files (index and creative) are in the folder.
44. Compress your folder into a ZIP file (see "Getting Ready: Part V" for instructions on how to compress the files. Do *not* use .rar or other compression types, *only* use ZIP.
45. Send me the assignment by email **by the time I give in class as a deadline**. Attach the work as a folder or a ZIP file (See page 7 for ZIP / compression instructions). **Important:** do *not* attach the file using Google Drive!! Doing so usually fails, as extra permission is needed before I can open the file.

That's it!

One extra note about UTF-8: even after all browsers have upgraded and can read any language without the `<meta charset="UTF-8">` tag, you should still include that meta tag in all pages. The reason is simple: many people use old browsers, sometimes many years old, and non-English text will fail on their browsers if you don't use the meta tag.

Chapter 2 Code

Once again, the tags are:

Beginning	Ending	Purpose
<!DOCTYPE html>		Declares that HTML5 is used
<html lang="en">	</html>	Begins and ends the page
<head>	</head>	Begins and ends the head
<title>	</title>	Sets the document title
<meta charset="UTF-8">		Allows all languages to be used
<meta name="author" content="yourname">		Notes who is the author of the page
<body>	</body>	Begins and ends the content area
<header>	</header>	This will hold the site title/subtitle
<nav>	</nav>	This will hold the nav menu
<main>	</main>	This will have the main content
<footer>	</footer>	This will show info about the page
<h1>	</h1>	This is the main page title
<h2>	</h2>	This is the main page subtitle
		Everything inside this is a paragraph

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

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>
            My Kewl Page
        </title>
        <meta charset="UTF-8">
        <meta name="author" content="yourname">
    </head>
    <body>
        <header>
            <h1>
                <a href="index.html">Site Title</a>
            </h1>
            <h2>
                Site Subtitle
            </h2>
        </header>
        <nav>

        </nav>
        <main>
            <h3>
                Page Title
            </h3>
            <p>
                Lorem ipsum dolor sit amet, pri debet mucius erroribus in, nec alterum scripserit ea, at
                    sit eruditi labores. His fugit semper admodum id, at est clita reprimique. Ne antiopam
                    definiebas mei.
            </p>
            <p>
                Utamur oblique dissentiet cu vim, his an duis possit accusata. Quem labores ad qui, aliquam
                    mentitum sit ei. No ius solum scaevola voluptaria, sed liber dicit ea. Per idque
                    theophrastus an, etiam democritum cu sea.
            </p>
            <p>
                Cibo pericula molestiae quo an. Ex sea dico delenit atomorum, ei vocibus patrioque mel.
                    Invidunt accommodare eam ad. An vim agam conclusionemque, intellegat temporibus pro in.
                    Sea aeque assentior voluptatum an, cu pertinax comprehensam per.
            </p>
        </main>
        <footer>
            <p>
                &copy; 2020 Yourname
            </p>
        </footer>
    </body>
</html>
```

Chapter 2 Checklist

- ❑ Do you understand what a browser engine is?
- ❑ Do you understand what it means to "render" a web page?
- ❑ What is a tag attribute?
- ❑ How is an attribute typed? What are the required punctuation marks and spacing?
- ❑ What is the `DOCTYPE` tag for?
- ❑ What does the `html` tag do?
- ❑ What attribute is usually used with the `html` tag?
- ❑ What tags are used within the `head` tag?
- ❑ Why is the `meta` tag with the `charset` attribute needed?
- ❑ What are heading tags? What are they used for?
- ❑ Why are indents important in HTML code?
- ❑ What is “debugging”?
- ❑ What are the three debugging points I list in this chapter?
- ❑ Do you fully understand all the tags that were introduced in this chapter?
- ❑ What are some of the design you should consider when creating a web site?
- ❑ Did you visit ten web sites and make notes on their design choices?
- ❑ Do you understand why they made these design choices?

Chapter 3: More about HTML

TECHNICAL

3a. Pathnames and Filenames

When you save a file, it should have a filename that follows certain rules. We already covered the idea of *filename extensions*, but that's just part of it. There are a few more things about how computers handle things that we need to go over. All of them have to do with **filesystems**.

A filesystem is a set of rules the computer uses for keeping files. The filesystem will decide how files are saved, how they are deleted, and many other things—including how files are *named*. That's what our points are about: naming files.

PATHNAMES

A pathname is like an extended address for a file. It will locate a file based on which folders and volumes it is inside.

An important element of a pathname is a **slash** /, which comes after a folder name. If a pathname is **documents/images/photo.jpg**, it means that a file named "photo.jpg" is located in a folder called "images," which is inside another folder called "documents."

There are two kinds of addresses: global and relative.

A **global** pathway begins at a root, or basic location. Starting from a well-known landmark, how do you find a file?

A **relative** pathway begins at your current location. Starting from where you are, how do you find a file?

Imagine your friend calls you up and asks how to get to your house. What's the first thing you do? *You ask their current location*. If you don't know where they are, you can't tell them where to go.

If they say they are lost and don't know where they are, you will point them to a well-known landmark, like a train station, and then they will give you **global** directions from the train station.

If they are able to tell you their location, the you will give them **relative** directions: from their current location, go this way and that way.

Two common **global** addresses are for your computer and for the internet:

Your computer:	file:///C:/Users/luis/Desktop/mysite/images/photo.jpg
On the Internet:	http://mysite.com/images/photo.jpg

Each one begins at the base point of the system—your computer hard drive, or the base location for all web pages on the Internet. It is a full set of directions from the base to the specific file.

An example of a **local** address would be:

images/photo.jpg

That's it. It works for your computer and for the web. It assumes that your starting point (your web page) is in a folder which also contains a folder called "images," and your photograph is inside that folder.

If the path is simple "photo.jpg" with no folder, that means the image is in the same folder as your web page.

What if you want to find a file *outside* your web page's folder? That is, you web site folder is inside another folder, and you want to go there. What then? You use this:

../photo.jpg

The "../" means "go up one folder." No folder name is needed. You could do this:

../images/photo.jpg

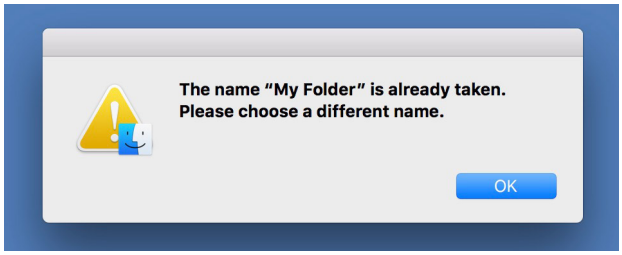
That would mean that your desired file is outside you current folder, and then inside a folder called "images."

In all these cases, the address begins with the current location of the web page file.

CASE

The second point is about **case sensitivity**. Try this out: on your Desktop, make a folder, and call it **my folder**.

Then make another folder, and call it **My Folder**. The same name, but with a few capital letters.



On a standard computer desktop, this is not allowed. When you try to make the second folder, you will get an error message about how the name is already taken.

This is because your computer is **case-insensitive**, which means that it sees no difference between uppercase and lowercase letters. So if you ask for **myphoto.jpg**, the computer will give you **MYPHOTO.JPG** if it exists.

However, the Internet is **case-*sensitive***, meaning that those two files are considered different. Asking for "myphoto.jpg" when the only file available is "MYPHOTO.JPG" will result in an error. That error will *not* happen on your computer.

As you can see, that can cause problems. You will design and test your web site on your computer, before you upload it to the web. If you accidentally have uppercase letters in an image filename in the web site you are making, that would not create any error in your site. However, once you upload it to the Internet, the image link would break, leaving an empty image rectangle on your site.

For this reason, I am making a rule: **all folder and filenames must use all-lowercase letters**. This rule is generally followed by web designers, but not 100%. It is possible to use uppercase letters; it is just a little harder to control without errors popping up. So only use lowercase letters in filenames, at all times.

Filename Rules: "Legal" Characters

When choosing a filename, you should follow these basic guidelines:

- Filenames should be short; longer names are harder and create errors
- Filenames should be descriptive; **mountains.jpg** is clearer than **IMG_8297.JPG**
- Filenames should use only legal characters

Okay, so what do I mean by "legal"?

"Legal" characters include lowercase English letters (*a~z*), numerals (*0~9*), as well as a hyphen – or an underscore _ character.

Everything else is not allowed. That includes symbols (*@&%=¥, etc.*), spaces, or punctuation marks (*?!/:;", etc.*). Most of these are reserved. For example, the slash is reserved for indicating a folder name.

Another character type to avoid: non-English lettering. Like capital letters, these may sometimes work, but they cause too many problems. Don't use them.

Legal filenames:

```
greenpark.jpg
green_park02.jpg
green-park02_2022.jpg
```

Illegal filenames:

```
green park.JPG
green+park"02".jpg
green:park02_2022?.jpg
```

Keep his in mind, it will be a rule set you must remember and use for the entire course!

CODE

More HTML

In this chapter, we will learn more HTML elements (the first 6 are *inline*):

Beginning	Ending	Purpose
		Makes text bold
		Makes text bold
		Makes text italic
		Makes text italic
		Creates a link
		Allows for specific styling of text
		Creates an all-purpose block / box
<header>	</header>	Contains the page main title & subtitle
<nav>	</nav>	Contains the links for the site
<main>	</main>	Contains the main content
<footer>	</footer>	Contains copyright & other site info
<h1> ~ <h6>	</h1> ~ </h6>	Creates titles for various page sections
<blockquote>	</blockquote>	Creates an indented paragraph for quotes
		Contains a numbered (ordered) list
		Contains a bullet (unordered) list
		A list item in an or

In the above list, the first part is *inline* tags, and the second part is *block* tags.

3b. Semantic Code

This is an important web design concept: **Semantic code**. "Semantic" means that the tag identifies what type of content should go inside the tag. Only paragraph text should go inside a paragraph tag, only quotes should in a blockquote tag, and only list items in a list tag, etc.

However, this can be confusing. For example, there are two tags for italic text: and . On a browser, both look exactly the same: like italic text. However, one is intended for text which appears italic but is not pronounced with stress (for example, foreign loan words, like *sashimi*), while the other is for words that are spoken with emphasis: "Look at *that*!"

However, why two tags, and why must we follow the rules about it?

For example, take the sugar and salt jars in the image at right. Aside from the text on the labels, the jars are identical. Either jar could hold sugar or salt. However, you *expect* sugar in the sugar jar, and salt in the salt jar.

You *could* put salt in the "sugar" jar and sugar in the "salt" jar. It is physically possible. *You* would know what is in them. However, *other people would not*, and it would create problems.

Other people will see your code; you want them to understand it without any problems. Therefore, put things in places where others will expect them. If you use semantic code, everyone can easily understand how your code and your page works.

The major structural tags, <header> <nav> <main> and <footer>, are simply boxes, and they all look exactly the same. You *could* use them interchangeably, like the salt and sugar jars. However, each one is *supposed* to have only a certain kind of content. The <header> must only have things that appear in the main page title area. It will have <h1> and <h2> tags, for example, but it will *never* have <p> or paragraph text inside of it.

Most tags are semantic, and must have a specific kind of content. and are the only normal tags in the body which are not semantic at all.

Declaration and **void** tags *cannot* be semantic, because they contain no content which affects the page.

Here are a few semantic tags:

BLOCKQUOTE

The <blockquote> tag is exactly the same as a paragraph, except (1) it is indented by 0.5 inches, and (2) **it must only contain sentence-or-longer quotes** (text from another source).

BOLD & ITALIC TEXT

There are two tags each for bold and italic.

<b> and will create bold text.
<i> and will create italic text.

So, how are they different? The differences are **semantic**.

 and are for **stress**. Any text inside these tags has special meaning.

The tag is for any text of special **importance**. If something is *really important*, and would be *pronounced more strongly* than other words, then it would use the tag.

The tag is for any text of special **emphasis**. If you write the sentences, "I *want* that" and "I want *that*," there is different meaning. The first emphasizes the fact that you want something; the second emphasizes what you want.

In contrast, <b> and <i> are purely for **style**. They are simply written in bold and italic because of a style rule.

In textbooks, key words are bold; they are *not pronounced differently*. This uses the <b> tag.

When you write a foreign word in English, like *Izakaya*, it must be in italics. Book titles and technical words are often shown in italics. There is no emphasis. These call for the <i> tag.

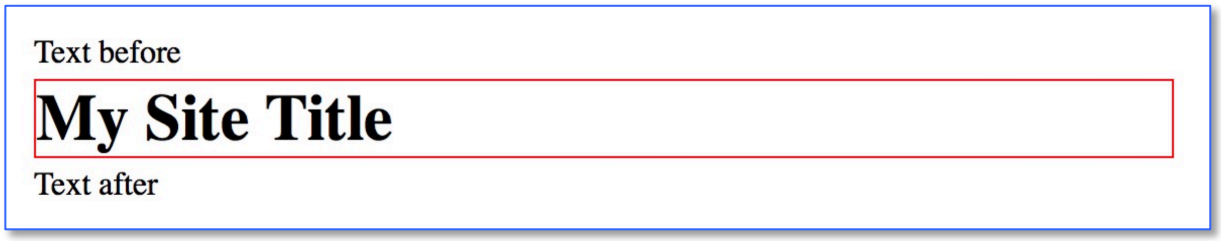
3c. Block & Inline

There are two basic categories of tags: **block** and **inline**.

Block tags:

- usually act as containers or structure for something
- have a rectangle shape, like a box
- create a new line above and below themselves; the block is alone on a line
- automatically fill up all the space from left to right, but only increase vertically when content is added (an empty block has no height)

For example: Text before `<h1>`My Site Title`</h1>` Text after



Inline tags:

- do not create line breaks; content before and after inline tags stay on the same line
- usually are used to style text, but images and line breaks are considered inline also

For example: Text before `<b>`My Site Title`</b>` Text after

Block elements include structural tags (`header`, `nav`, `main`, `footer`), paragraphs `<p>`, blockquotes `<blockquote>`, and header titles (`<h1>`, `<h2>`, etc.).

Inline elements include bold `<b>` or `<strong>`, italic `<i>` or `<em>`, links, and images.

As I mentioned on a previous page, `<div>` *and* `<span>` *are the only normal tags in the body which are not semantic at all.*

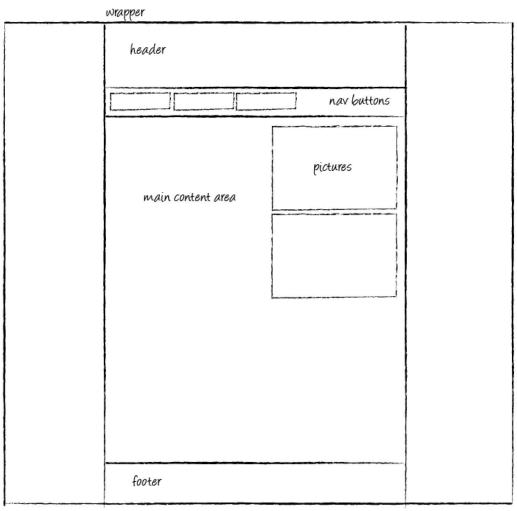
The `<div>` tag is used to create a block rectangle box **for any purpose**. If you need a box for a purpose which does not have a special tag already, you would use a `<div>` tag. The `<div>` tag can be used to create new structural elements such as the **wrapper** container.

The `<span>` tag is used to style text in ways that HTML cannot do. In HTML, for example, you can change text to bold and italic with HTML only. However, you cannot give text a color or shadow in HTML. That requires CSS. Therefore, you would use the `<span>` tag, and use CSS to add the color or shadow. We will learn CSS in a later chapter.

The `<div>` and `<span>` tags are essentially "blank" or general-purpose tags which can be styled with CSS. They are used when there is no HTML tag which serves your needs.

3d. HTML Structural Elements

Structural tags create areas on your page. They are block tags, but they are a special group of tags.



As you can see in the **wireframe** shown at right, there are four areas of the page:

- `header` (contains titles)
- `nav` (contains links)
- `main` (contains main content)
- `footer` (contains info about page)

These are **structural** elements. There are more of them, but for now, these are the ones we are learning.

Structural elements have several purposes:

- They separate different types of information
- They help organize the page
- They allow for easier styling (colors, fonts, etc.)

If you look at almost any page on the Web, you will probably find these areas.

As I have noted before:

Important Rule: everything in the body should be inside a structural tag. Text, images, and non-structural block tags, should not be "free-floating" outside a structural box.

Important Rule: block tags may be placed in structural tags, but block tags should usually not be in other block tags. Exceptions include block tags *intended* to reside in each other, like using the list item `<li>` tag inside the list tags `<ol>` and `<ul>`.

The body HTML on a page with structural tags may look like this:

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

All other tags fit into the structural elements. **Nothing should go between the structural tags** (though as you become more skilled, you can break this rule).

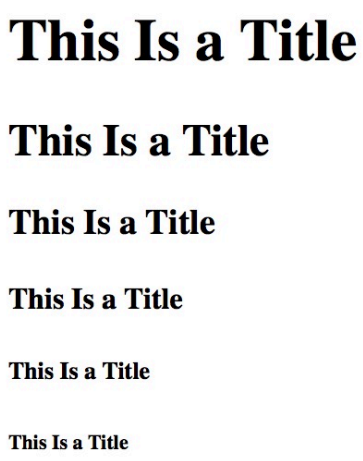
Don't forget: As a general rule:

- **Inline** content (text and inline tags) must be inside **block** tags (images not always);
- **Block** tags should be inside **structural** tags, except in special cases;
- **Structural** tags must be inside the **body** tag.
- Nothing should be outside the structural tags, except in special cases.

3e. Headings

First, let's go over the heading tags:

```
<h1>This Is a Title</h1>
<h2>This Is a Title</h2>
<h3>This Is a Title</h3>
<h4>This Is a Title</h4>
<h5>This Is a Title</h5>
<h6>This Is a Title</h6>
```



These tags look like they do in the illustration at right:

Each of these creates a title for a document or section of text. All have bold text with large top and bottom margins. Otherwise, they each have a different font size. The `<h1>` tag has a font size double the normal size; the `<h6>` is quite small.

Because these are semantic tags:

- The `<h1>` tag should only be used for the **main title of the whole web site** (not the web page) in the `<header>` tag;
- The `<h2>` tag is usually used for the **subtitle of the whole web site** in the `<header>` tag;
- The `<h3>` tag is often used as the **main title of the web page** (not the site). If the `<h2>` tag was not used in the header as the site subtitle, then it can be used for the web page title. The title for the web *page* is usually located at the top of the `<main>` tag.

It is important to note that the heading tags should be used **consistently** on all pages; if an `<h2>` tag is used as the web site subtitle on any page, it should be used that way on *all* pages, and should not be used as, say, the web *page* title on any other page.

You should also know that the headings used in the `<header>` tag are for the *whole web site*, and therefore should remain the same on all pages in your site.

The heading tags should be used *in order*, from the most important to the least important, and often from the top of the page to them bottom. It is possible to break the top-to-bottom order; for example, you can have an `<h4>` tag used to begin a section of the page, then several `<h5>` tags to mark parts of the section, and then use an `<h4>` tag again to begin a new section.

Their styles can be changed (we will learn about CSS styling later).

Do not confuse **head**, **header**, and **heading** tags. They are all very different!

3f. Lists

You can create two types of lists with HTML: ordered (numbered) or unordered (bullet). The two basic tags are:

`<ol> </ol>`	*ordered list (numbered)*
`<ul> </ul>`	*unordered list (bullet)*

Each of these creates a standard block, but with one difference: there is 40 pixels of padding on the left—a ½-inch indent. Anything inside these list blocks will be indented by that much.

To create the list, just add **list items**, inside `<li>` tags:

```
<ol>
	<li>This is the first item</li>
	<li>This is the second item</li>
	<li>This is the third item</li>
</ol>
```

If you create list items inside an `<ol>`, the list will be numbered; if you create list items inside a `<ul>`, the list will be a bullet list (big black dots next to each item).

```
<ul>
	<li>This is the first item</li>
	<li>This is the second item</li>
	<li>This is the third item</li>
</ul>
```

The two lists shown above should display like this:

1. This is the first item
2. This is the second item
3. This is the third item

- This is the first item
- This is the second item
- This is the third item

You can create an ordered list with letters or roman numerals by using the **type** attribute:

```
<ol type="A">
	<li>This is the first item</li>
	<li>This is the second item</li>
</ol>
```

By using "A", you get a capital-letter list (A, B, C, etc.); using "a", you get a lowercase-letter list (a, b, c, etc.); using "I", you get a capital-roman-numeral list (I, II, III, IV, etc.); and using "i", you get a lowercase-roman-numeral list (i, ii, iii, iv, etc.).

Note that `<ol>`, `<ul>`, and `<li>` are all **block** elements. Since the `<li>` tags are all next to each other, they will each occupy a row by themselves, taking up all available space left to right.

Also note that the `<ol>` and `<ul>` (not `<li>`) have a built-in style with 40px of padding on the left side. This is what gives them the apparent indent on the left. If you want to make them flush with the left margin, you will have to get rid of the padding, which we will learn later.

3g. Links

The <a> tag is used to create links. It requires the href="" attribute:

```
<a href="about.html">Visit the About Page</a>.
```

All content (text or images) between the start and end tags becomes a link. The href identifies the destination.

There are two kinds of addresses you can put inside the href.

Absolute (global) address: http://lujweb.com/gdn103/about.html

Relative (local) address: about.html

As you can see, the **absolute** address is a full URL. This is mostly used when linking to a page on a different web site.

The **relative** address is used to link to a page in the same web site. In this case, "relative" means "relative to the location of the link." If the linked page is in the same folder, then you only have to put the exact name of the web page.

Where to Put a Link

Let's say you have a sentence of text with a link, such as:

> You should visit the newspaper's web page with the story about links.

In such a sentence, if you have a link to that web page, what part of the text should be the link? Assuming that the link would be blue and underlined, which of these is correct?

> You should visit the newspaper's web page with the story about links.
> You should visit the newspaper's web page with the story about links.
> You should visit the newspaper's web page with the story about links.
> You should visit the newspaper's web page with the story about links.
> You should visit the newspaper's web page with the story about links.
> You should visit the newspaper's web page with the story about links.
> You should visit the newspaper's web page with the story about links.

The correct answer is that there is no "correct" answer. I would probably use the second one, or the second one from last. Other people might have other preferences. Few people would use the whole-sentence link.

Keeping Visitors

Normally, when someone clicks on a link, the new page will replace the old one in the same browser tab. When a visitor goes from one page in your site to another, this is the way it should be. However, when someone clicks on an outside link, you don't want them to completely leave your site! Adding the target attribute will stop that; the new page will open in a different tab, leaving your web site still open in the original tab:

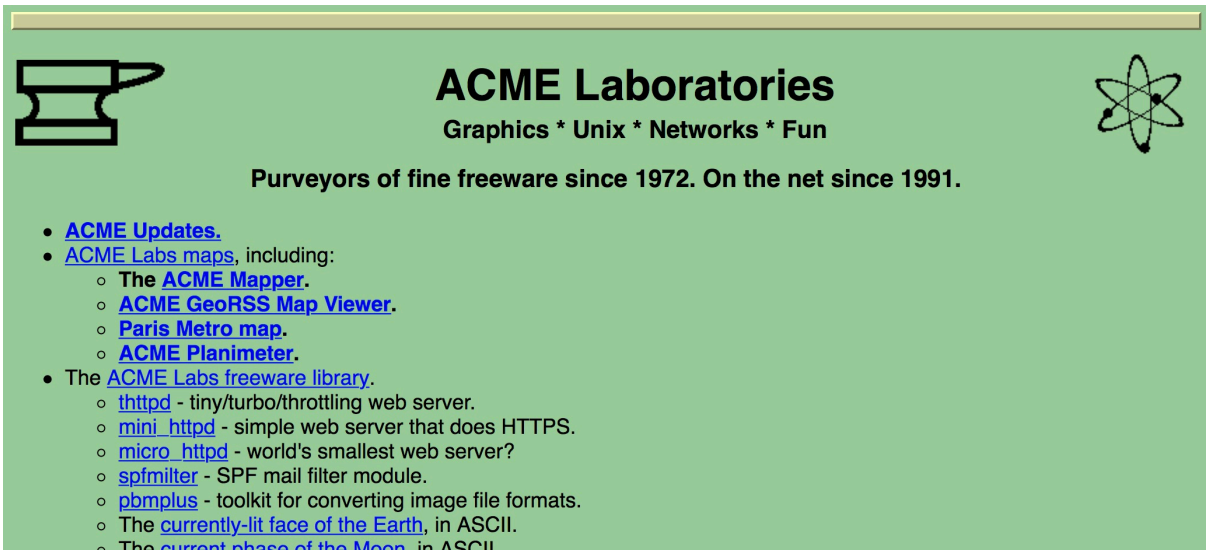
```
<a href="about.html" target="_blank">Visit the About Page</a>.
```

Never use the “target” attribute in a link to another page in your own site!

3h. Unordered Lists and the Nav Menu

Nav Menus

Typically, nav menus are built using the unordered list, featuring the `<ul>` tag with `<li>` tags inside. This comes from the 1990's tradition of presenting links as items on a non-numbered list; see an example of an actual 1990's web site below. While this practice eventually died out, the use of the bullet lists for menus remained.



As a result, when you make a nav menu (menu bar) for a web site today, it is expected that you will use the unordered list tags—even advanced menus. It is possible to use other tags to create menus, but the use of the `<ul>` and `<li>` tags is expected. We will therefore use the `<ul>` list for nav menus in all projects in this class.

- *An important note: the `<ul>` and `<li>` tags are **only** used for links in the nav menu.* Do **not** use these tags for any links outside the `<nav>`.

A very basic, simple nav menu, as a result, will look like this:

```
<ul>
	<li>Link</li>
	<li>Link</li>
	<li>Link</li>
</ul>
```

In the example above, I just placed text inside the list items (`<li>`). I did this to show you the basic code. When you add links, the links go **inside** the list items (not outside).

```
<ul>
	<li><a href="ponta.html">Ponta</a></li>
	<li><a href="tricks.html">Tricks</a></li>
	<li><a href="adventure.html">Adventures</a></li>
</ul>
```

Note that the **index.html** is not included; that link goes inside the `<h1>`, not the `<ul>`.

Later, when we learn CSS, we will learn how to make these bullet points change into buttons!

3i. HTML Whitespace

In learning HTML code basics, we must review how HTML handles something called **whitespace**. Whitespace refers to **spaces**, **tabs**, and **line breaks**. In your experience, a line break is when you hit the "Enter" key on the keyboard. Whitespace refers to any character you display or any key you press which does not leave any visible mark on the page.

Computers mostly ignore whitespace. When an HTML document is rendered, it ignores tabs and Enter-paragraphs. It will pay attention to *single* spaces, but any consecutive spaces after that are ignored. Whitespace is mostly for people, allowing us to format the code in a way that makes it much easier to read and understand.

I would like you to understand how whitespace works in HTML, and why.

Set up Visual Studio Code and your preferred browser as I described earlier. Next, open VSC, and create a simple web page with the following code:

```
<html>
     <head>
          <title>
               My Page
          </title>
     </head>
     <body>
          <p>
               Add text to the paragraph tag.
          </p>
     </body>
</html>
```

Once you have finished, save the file on your desktop as "`test.html`" and then open that file in the browser. It will appear very simple: just the text will appear in the browser window, along with the document title in the browser tab.

Now you are ready to test whitespaces. Add many extra spaces between words in the paragraph tag:

```
<p>
    Add text to              the paragraph tag.
</p>
```

Save in VSC, refresh in the browser. What do you see?

You should see *nothing different* than before. The first space between the words will appear as normal, but the second, third, and all other spaces after that do not appear—they are not rendered.

The same thing happens with tabs. Go ahead—in VSC, put several tabs between words in that sentence. Then, save and refresh; again, you will see no change in the rendered web page in the browser.

Finally, try adding paragraph spaces with the Enter key.

```
<p>
    Add text to
```

```
    the paragraph tag.
</p>
```

(Keep your applications open, we're not finished yet.)

Once again, you will find that the paragraph spaces in the code do not appear in the rendered web page. Why not?

Remember what I just taught you a few pages ago in Chapter 3e, about indents and code formatting. With your web page code, it is important to type the code neatly, using new lines (Enter key) and indents (tabs & spaces). Using this style of code formatting, your code remains clear and easy to see, with parent-child relationships very clearly shown.

However, in order to do this, we must use whitespaces in the code which are not intended for rendering in the browser. If all of our spaces, tabs, and new lines showed up in the browser, the formatting in the code would destroy the formatting in the rendered web page!

As a result, the browser will not show most of these whitespace characters. It will render a single space, but nothing more. Two or more spaces in a row, tabs, and Enter line breaks do not show up.

Therefore, if you want extra spaces or line breaks to appear, **you must use special types of code to make them appear.**

The Line Break

There are two ways to create extra vertical space between objects. One way is with CSS code, which you have not learned yet. The other way is with `<br>`, the line break tag.

Try this in the code you've been working on:

```
<p>
    Add text to

   <br>

   the paragraph tag.
</p>
```

Save and refresh. You'll notice that the text now appears on two lines instead of one. Try adding more break tags:

```
<p>
    Add text to

   <br><br><br>

   the paragraph tag.
</p>
```

You'll see that each new tag increases the line breaks by one line.

The `<br>` tag is a handy way to make inline content begin on a new line, or to add spacing between objects. *However, keep in mind that this can also be done with CSS, and using CSS is usually preferred.* In other words, only use the `<br>` tag when necessary; do not over-use it.

The Space Entity

That title, "The Space Entity," might sound like a science fiction movie title, but in HTML, it makes perfect sense.

An **entity** is a special code which allows you to create any character in any language, including whitespaces, symbol, and even emoji.

These are all based upon something called **Unicode**. Do you remember trying to look at Asian-language web pages, and seeing gibberish instead? That gibberish is called *mojibake* in Japanese. *Mojibake* happens when the computer does not understand the correct text encoding being used.

This is because, many years ago, each language had various special code sets. If your browser was set to look for English, but found a Japanese web page, it would show *mojibake*.

This problem was solved by using Unicode. You have already used Unicode, in fact—when you wrote your first web page in Chapter 3, you added a line of code:

```
<meta charset="UTF-8">
```

This tag tells the browser to use the UTF-8 character set, which is based on Unicode. Unicode includes more than 128,000 different characters from 135 different writing systems. Virtually every character from every language can be expressed in Unicode. Each one has its own code number.

Each character on a computer is represented by a number. For example, the lowercase letter "a" is 97. That is the Unicode number for the letter "a": 97.

You can express this in HTML by using the entity code. Each entity begins with &# and ends with a semicolon (;). So, if you want to have the lowercase letter "a" appear, just type a in your HTML document. Go ahead—type that, then save and refresh. You should see an "a" appear where you typed the code.

Normally, if you want to use "a," you just type it normally with a keyboard. However, there are some characters which are not easily typed. For example, let's say that you want to type a heart shape. How do you do that? It turns out that the heart symbol's code number is 9829. So type this in the HTML document:

```
&#9829;
```

Save & refresh; the heart symbol will appear on your page! See Appendix C for a list of common entity symbols and their codes.

There is an alternate way of expressing entities. Many of the commonly-used symbols have special codes, using words instead of numbers. For example, the yen sign is `&yen;` (notice there is no # in the code this time).

So, finally we come back to the space. In Unicode, the code for a space is 160, so you could type   and you will see a space. However, it might be easier to remember ` ` ("nbsp" stands for "non-breaking space").

And that's how you can get repeating spaces in an HTML document: just use ` `.

Once again, however, there is a caution: the ` ` entity is seen as sloppy and undesirable, especially when used repeatedly. Normally, if you need any kind of spacing, using CSS is preferred. However, there are some times when the ` ` is more convenient to use.

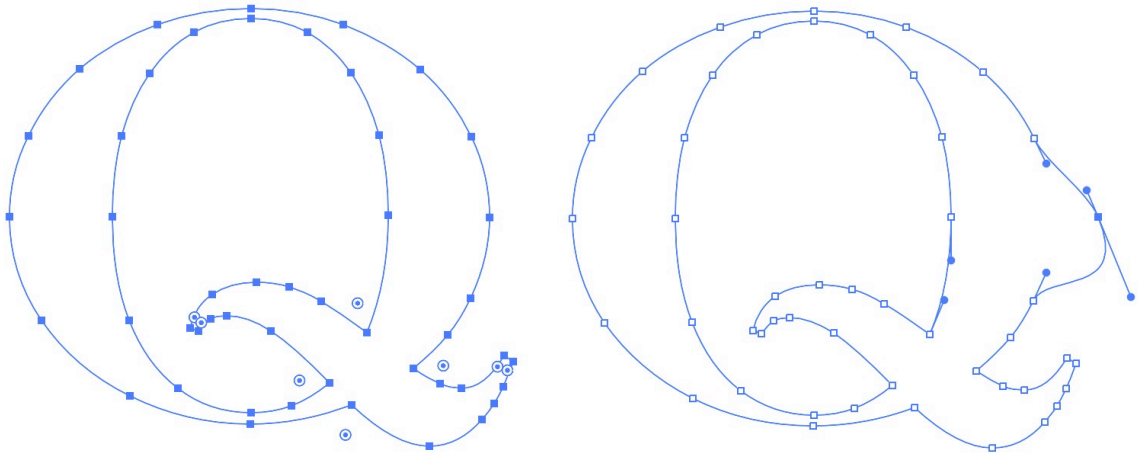
DESIGN

3j. Font Basics

WHAT ARE FONTS?

A font is a variation of the shape of characters according to a theme. Most fonts have a complete set of numbers, lowercase and uppercase letters, punctuation, and a variety of symbols.

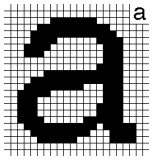
Each character can be changed to any size, because they are made up of a number of "Bézier" curves. These are created by points representing mathematical coordinates, as well as values representing the details of the curve or corner.



By clicking on any of these points, you can reposition them and change the angle of the curve, creating whatever shape you want. Because all of these points are mathematical representations, they can be resized to any size, making the font appear perfectly smooth whether it is tiny or huge. This size is called **point size**. The standard point size is 12pt.

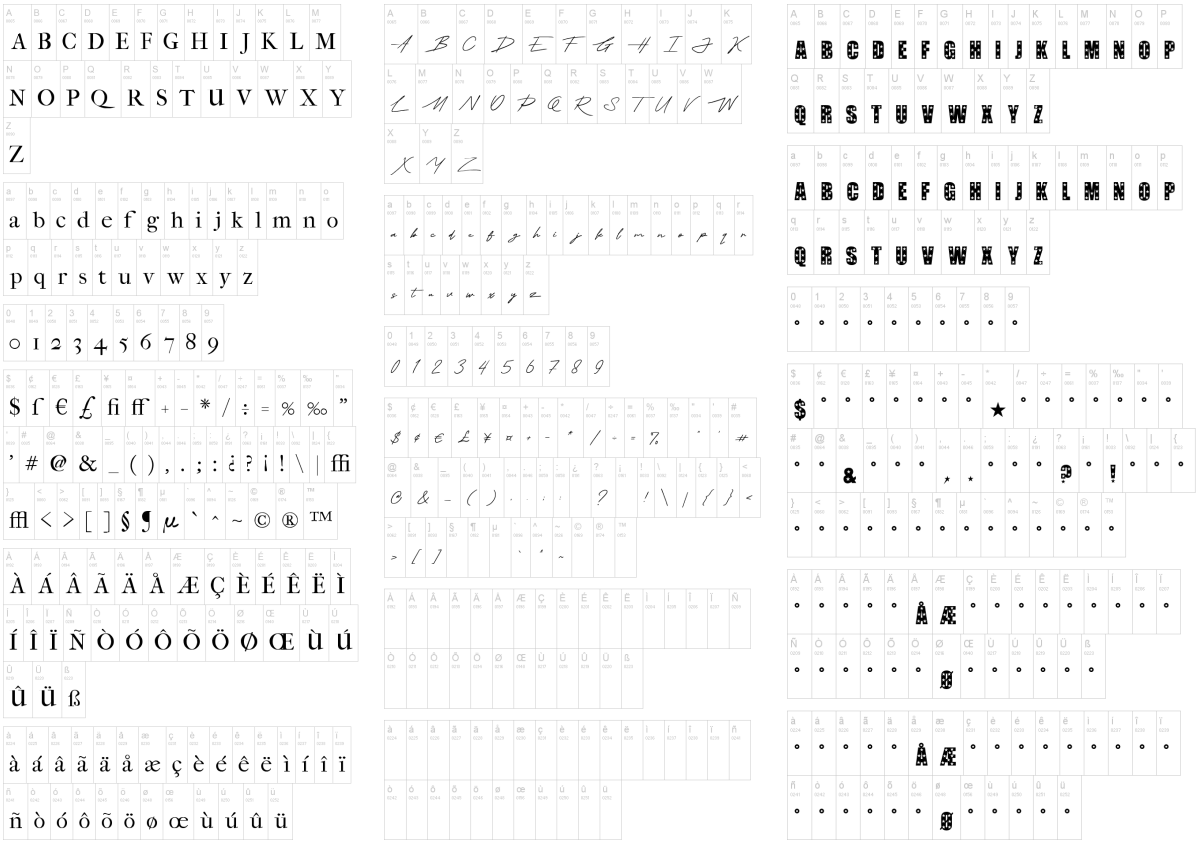
> ***Class Rule:*** *font sizes for paragraph text should not be greater than 14pt, except when very small fonts are used.*

There are various font formats, including **TTF** (TrueType format), **OTF** (OpenType Format), and **WOFF** (Web Open Font Format). You may also find **SVG** and **EOT**, but you can ignore those. The best to use at any time is TrueType format. TrueType was created by Apple in 1991, and was later adopted by Microsoft, and became an industry standard.



The above formats are called **outline fonts** because the Bézier curves are used to create outlines of the characters. Previous to that, computers used **bitmap fonts**, which created each character by "painting" each pixel in the glyph. These were not scalable (you could not change the font size smoothly), and to make the best-quality fonts, you had to have separate files for specific point sizes. Even with that, the pixels were often visible, and created an effect known as "the jaggies," for the block-like shapes they created.

Any one font may have a full set of characters, including symbols and diacriticals (accents, umlauts, and more, such as é, ü, and î). The three glyph charts below give examples of fonts with different collections of characters.



Some fonts you may find for download are “Demo” fonts, which have a very limited glyph set, like the “Hollywood” font above at the right. Notice that it does not have numbers or most of the punctuation marks.

FONT CATEGORIES

There are various categories of fonts, notably the following:

Serif	*An elegant style of font for reading, with marks at the end of lines*	My Little Table
Sans-Serif	*A plain style with no decorative features, very plain lines*	My Little Table
Script	*Personal handwriting or calligraphy style; possible for paragraph text but not usually*	My Little Table
Display	*Highly decorative fonts that can never be used as paragraph text*	My Little Table
Monospace	*All characters are the same width (as opposed to “proportional” fonts)*	My Little Table

You will note that only serif and sans-serif fonts are suitable for paragraph text. Display and cursive fonts are best for headings and titles, or other special text. Monospace is often used to exhibit samples of code (I use the “Monaco” monospace font to represent code in this text).

There are many other subcategories as well:

Slab Serif	*With serifs that are large blocks; used in many of the Trump campaign banners*	My Little Table
Modern	*Font with very wide vertical and very thin horizontal contrasting lines*	My Little Table
Blackletter	*Very old calligraphic style associated with Germany (often called Germanic)*	My Little Table
Geometric	*A sans-serif font basic on geometric forms like perfect circles*	My Little Table
Dingbat	*All characters are drawings*	

Font Associations

There are some basic font associations that may be known worldwide. For example, **serif** fonts are seen as elegant, used in novels and essays to make the text look more attractive. **Sans-serif** fonts are more plain, but are much easier to read clearly, and so are often used with instructions and warnings. Sans-serif fonts are therefore often considered more important and attention-grabbing, and so are used when someone wants to project a stronger image.

These are common **associations**, a kind of "personality" for a font which is used in specific situations. For example:

Have You Seen This Font Before?

This font, Western, is recognizable as a 19th-century American "Wild West" cowboy-style font, perhaps used on Saloon signs or "Wanted" posters. Maybe you have seen:

Have You Seen This Font Before?

This font, Bank Gothic, is often used in titles and credits for TV shows and movies. Maybe:

HAVE YOU SEEN THIS FONT BEFORE?

Many fonts you will perhaps not recognize specifically, but you will get a "feeling" or a "sense" from it, like Engravers MT above, which strongly resembles the font used on U.S. paper currency. There is also:

Have You Seen This Font Before?

The font above, Magneto, is often used when the name of a machine or device is made into a metal attachment, like cars or refrigerators. One more example would be:

HAVE YOU SEEN THIS FONT BEFORE!

That would be Neuland Inline, which looks like a woodcut style, often used for any entertainment show with a jungle or forest theme, like Jurassic Park, The Lion King, or Jumanji.

By seeing these fonts commonly in very specific contexts, you begin to see how people's feelings and moods might be changed by which fonts you use.

FONT USAGE

In order for someone to see a font on their computer, it usually must be installed on that specific computer. Computers come with a range of fonts pre-installed, and when you add software like Microsoft Office, a number of other fonts may be added. Additionally, you can download and add fonts yourself, often free ones, like from the site **dafont.com** or elsewhere. These fonts live in the operating system of your computer.

This is often a problem: fonts which are on your computer may not be on other people's computers. For example, if you make a document on your computer using an unusual font that you installed, people who open that document on different computers will probably not see that font. Instead, the computer will substitute a different font often one that is very different.

There are some ways to show fonts which are not on other computers. For example, when you save a document as a PDF, the font is embedded into the file, and will appear exactly the same on any other computer.

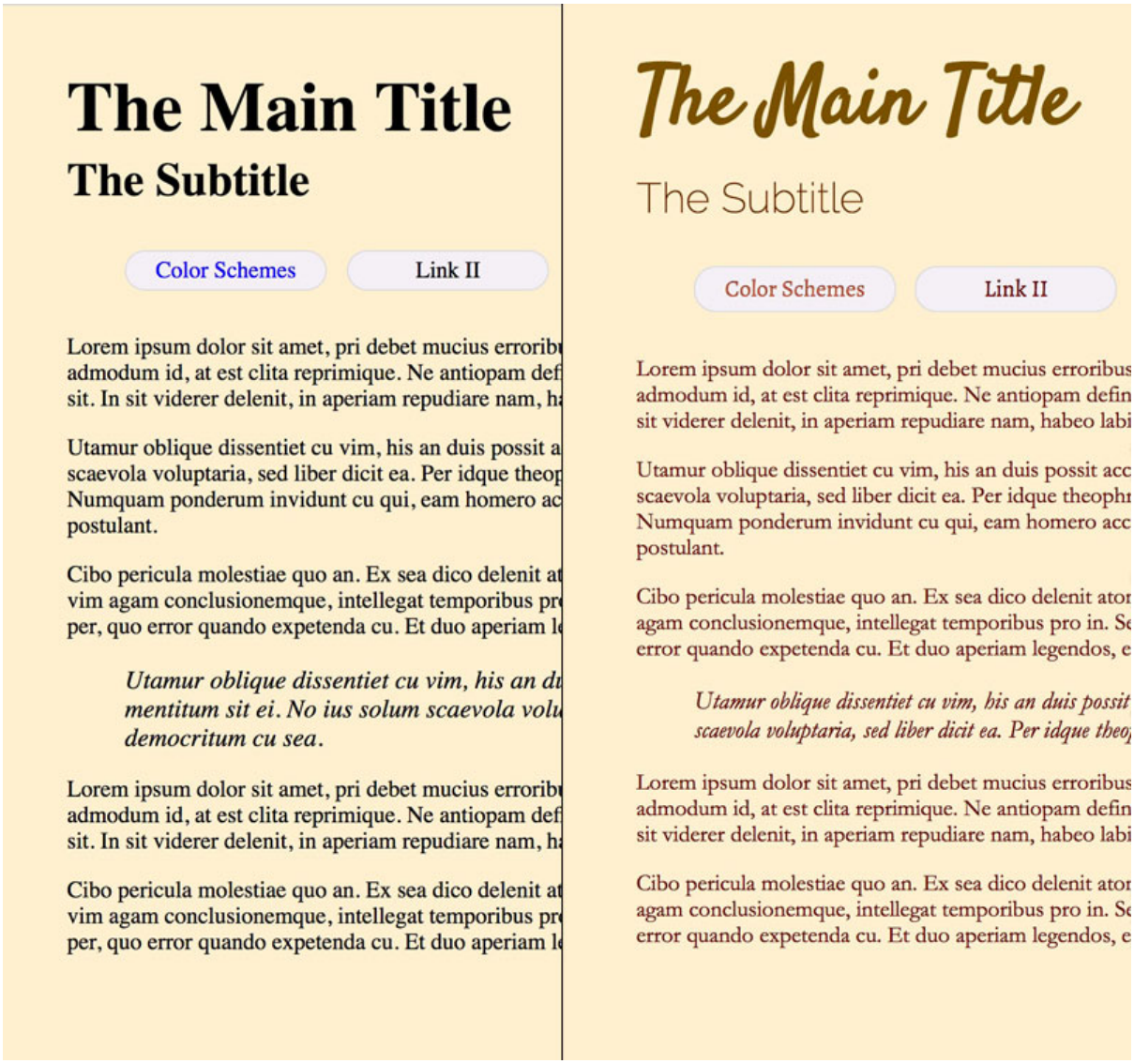
Still, most computers have at least a dozen or so fonts in common. These are called **web-safe** fonts:

Times New Roman, Times	*serif*
Georgia	*serif*
Palatino Linotype, Palatino	*serif*
Garamond	*serif*
Arial, Helvetica	*sans-serif*
Tahoma	*sans-serif*
Verdana	*sans-serif*
Lucida Sans	*sans-serif*
Trebuchet MS	*sans-serif*
Impact	*sans-serif / display*
Courier	*monospace*
Comic Sans MS	*script*

These fonts, however, are not very good. Most are very similar to each other, and are boring. Courier is too specific a style (old-style typewriter), Impact is too dense and plain—and Comic Sans, perhaps the most interesting web-safe font, is so over-used that many people consider it a "stupid" font, used only by people with not very much style sense.

For web sites, the same may be achieved by using **web fonts**, or fonts that "live" on the Internet, and will be visible to anyone connected to the Internet. While the most commonly used is Google Fonts (https://fonts.google.com/), there are other web font repositories.

Choosing the right fonts can be very important for your site; professionals put a great deal of thought and effort into finding the right fonts and the right combinations. For example, using Times New Roman, while possible, will be recognized by many people as plain and boring. See the visual example on the next page to see how different a site can appear when using different typefaces.



The version on the right looks much better, don't you think? You should absolutely consider the tone and appearance of the page, and ask yourself: which fonts look best on my site? Which match my content the best? Which are most attractive? Which fonts work the best together?

For example, a more casual sire might use more handwriting fonts; a site about sports might use heavier block or slab fonts; a site about history, more elegant, classical fonts.

When considering fonts for your sites, just browse Google's fonts, select the ones you want, and apply them in CSS.

Chapter 3 Code

All the tags we have learned so far are:

Beginning	Ending	Purpose
Page Frame		
<!DOCTYPE html>		Declares that HTML5 is used
<html lang="en">	</html>	Begins and ends the page
<head>	</head>	Begins and ends the head
<title>	</title>	Sets the document title
<meta charset="UTF-8">		Allows all languages to be used
<meta name="author" content="yourname">		Notes who is the author of the page
<body>	</body>	Begins and ends the content area
Structural		
<header>	</header>	Contains the page main title & subtitle
<nav>	</nav>	Contains the links for the site
<main>	</main>	Contains the main content
<footer>	</footer>	Contains copyright & other site info
Block		
<h1>	</h1>	This is the main page title
<h2>	</h2>	This is the main page subtitle
		Everything inside this is a paragraph
<blockquote>	</blockquote>	Creates an indented paragraph for quotes
		Creates an all-purpose block / box
<h1> ~ <h6>	</h1> ~ </h6>	Creates titles for various page sections
		Contains a numbered (ordered) list
		Contains a bullet (unordered) list
		A list item in an or
Inline		
		Makes text bold
		Makes text bold
		Makes text italic
		Makes text italic
		Allows for specific styling of text in CSS
		Creates a link
 		Breaks inline content; jumps to new line

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

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>
            Title
        </title>
        <meta charset="UTF-8">
        <meta name="author" content="yourname">
    </head>
    <body>
        <header>
            <h1>
                Web Site Title
            </h1>
            <h2>
                Web Site Subtitle
            </h2>
        </header>
        <nav>
            <ul>
                <li><a href="index.html">Link I</a></li>
                <li><a href="index.html">Link II</a></li>
                <li><a href="index.html">Link III</a></li>
                <li><a href="index.html">Link IV</a></li>
            </ul>
        </nav>
        <main>
            <h3>
                Wep Page Title
            </h3>
            <p>
                Lorem ipsum dolor sit amet, pri debet mucius erroribus in, nec alterum scripserit ea, at sit eruditi
                    labores. His fugit semper admodum id, at est clita reprimique. Ne antiopam definiebas mei.
                    Mediocrem pertinacia ut eam, ea illud homero principes sit. In sit viderer delenit, in aperiam
                    repudiare nam, habeo labitur eu vis. Illud dicant ex mel, inani legimus ea ius.
            </p>
            <blockquote>
                "This is a quote by some other person." &mdash;Luis Poza
            </blockquote>
            <p>
                Utamur oblique dissentiet cu vim, his an duis possit accusata. Quem labores ad qui, aliquam mentitum
                    sit ei. No ius solum scaevola voluptaria, sed liber dicit ea. Per idque theophrastus an, etiam
                    democritum cu sea. Oblique ancillae accusata sit ea. Numquam ponderum invidunt cu qui, eam
                    homero accusam quaerendum eu. Alterum antiopam id nam, in usu vivendum postulant.
            </p>

            <img src="images/pontaleaf.jpg">

            <figure>
                <img src="images/pontaleaf.jpg">
                <figcaption>
                    This is an image caption
                </figcaption>
            </figure>

        </main>
        <footer>
            <p>
                &copy; 2019 Yourname
            </p>
        </footer>
    </body>
</html>
```

Note: the figure & figcaption tags will be introduced in the next chapter.

Chapter 3 Checklist

- ❑ What is a "pathname"?
- ❑ What does the slash / mean in a pathname?
- ❑ What is the difference between a global and a local address?
- ❑ How do you make a local pathname for an item inside or outside a folder?
- ❑ Do you understand the nature and importance of case-sensitive names?
- ❑ Do you understand the rules of naming files and folders?
- ❑ What is “semantic” code? What does “semantic” mean in this case?
- ❑ What is the only kind of text that should be put into a blockquote tag?
- ❑ What is the difference between the `<b>` and `<strong>` tags?
- ❑ What is the difference between the `<i>` and `<em>` tags?
- ❑ Which two “normal” tags are not at all semantic?
- ❑ What are “structural” tags? What are their purpose?
- ❑ What are the four structural tags we learn here, and what do they contain?
- ❑ Is it possible to put a structural tag inside a normal block tag? Why?
- ❑ Is it always OK to put a block tag within another block tag?
- ❑ How many heading tags are there? When and why are they used?
- ❑ What styles are the same in all heading tags?
- ❑ What are the two list types, and how are they different?
- ❑ What tag is used for each item in either kind of list?
- ❑ What is the tag used for links? What is the one necessary attribute for that tag?
- ❑ What is the difference between absolute and relative links?
- ❑ What tags contain a list of links in the nav menu bar?
- ❑ What attribute is used to make links open in a separate tab or window?
- ❑ When should you not use the “target” attribute?
- ❑ What is “whitespace”?
- ❑ When writing code, what is whitespace used for?
- ❑ If you add an “Enter” break, a tab, or more than one space in the code, how does that display in a browser? Why does it work that way?
- ❑ How do you add a line break in HTML?
- ❑ Is it OK to use the `<br>` tag to separate paragraphs? Why not?
- ❑ What is an “HTML Entity” and how is it used?
- ❑ What characters are used to make entities? What does each entity start & end with?

Chapter 4: Beginning with Images

TECHNICAL

4a. Image Basics

People love using images on web pages. This is great, but there are some basic rules you **must** follow with images:

1. Images must be as small in file size as possible while still having good quality
2. Images should be the correct format (JPG, PNG, or GIF)
3. Images should be a similar pixel size (resolution) as they will appear on the page
4. Image size on the page should be reasonable and consistent
5. Image files should be named correctly and descriptively, not randomly
6. Image rights must be respected

IMAGE FILE SIZE (SIZE IN BYTES)

All files have a **file size**. The file size is expressed in **bytes**. In simple terms a "byte" is a single letter you type on a keyboard. Type a thousand letters, and you have a thousand bytes, or a "kilobyte" (KB).

A plain text file (text without images or formatting) is very small, usually no more than a few kilobytes. A formatted text file (with fonts, font sizes, indents, margins, etc.) can be hundreds of kilobytes.

Images are larger. A photo taken with you cell phone camera can be millions of bytes, or several **megabytes** (MB). Video is even larger.

All of this is important when it comes to designing a web site because web sites are **downloaded**—that is, the data, including images, is sent over an Internet connection. Sometimes that connection can be slow. Plain text files—which is what web pages are—will be sent fairly quickly, but images, which are much larger, can take a much longer time, especially if your visitor has a slow connection. If you page has dozens of images, it could take a great deal of time for them to load.

Slow loading times are bad. Visitors become impatient, and will leave a site if it takes too long to fully appear in their browser. Therefore, images with large file sizes are bad for your site. If they are too big, they must be made smaller.

An image taken by your cell phone is going to be perhaps ten times wider and ten times taller than what you will need. Therefore, you should **edit** the image before using it on your web site. You should do three things: Crop, Resize, and Save for Web (see Chapter 4c on how to do this).

When you **crop** a picture, you take away parts from the edges that are not needed, to better center on the image's subject. When you **resize** the picture, you are reducing the number of pixels (see "Resolution" below). Both of these reduce the image's file size.

The last step is **saving** the image. When you save the image, you have a choice of three standard image file types: JPG, PNG, and GIF. These are explained in the next section.

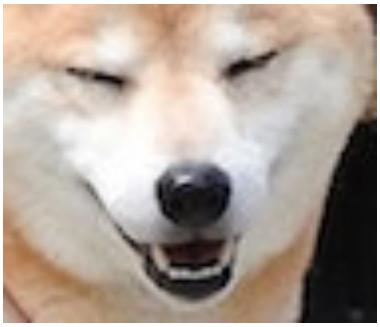
The most common choice for file type is JPG, which produces normal, rectangular images, but allows you to control the file size through **compression**.

Compression is a way to reduce the file size by getting rid of excess information and using various other tricks. There are two types of compression:

- **Lossy compression**: image quality can worsen, but file size becomes small
- **Lossless compression**: image quality remains perfect, but file size stays large

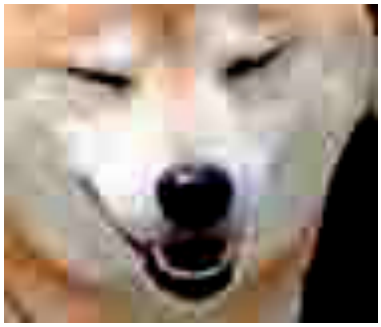
JPG uses lossy compression, while PNG and GIFs use lossless. This makes the JPG type the best for general images.

Lossy compression allows you to compress images to a very small size while still retaining fairly good quality. You can cut more than half the file size by using 60% quality and the image still looks very good. 50% is acceptable, but 40% and below begin to distort the image. If you compress the image all the way, down to 0% quality, the file size is greatly reduced, but the image quality becomes pretty bad.



100% quality, 850 KB

60% quality, 400 KB



0% quality, 100 KB

After you crop and resize almost any image to a size acceptable for a web page, compressing it to 60% using JPG compression should get the image to a very acceptable size.

If you use images taken by your cell phone or other digital camera without cropping, resizing, and saving for web, even one photo that large can make the page load much more slowly, with the image slowly building over the course of up to a minute or more.

When you handle the image in an editor, you will reduce the file size. **Most images on web pages should be 200 Kilobytes or less**. 400 KB can acceptable, but if you have many images, that can still slow things down. Larger files will slow down page loading, and will make visitors impatient and unhappy.

An exception is **background images**. These must be larger (usually 1400 ~ 2000 pixels wide). These can be larger because there is only one of these per page. However, the image quality can also be made less; the lower quality will not be as apparent. Of course, you want it to be the best quality you can get it, but saving it at 2000 pixels wide at 50% quality might bring it down to 600 KB, which is acceptable.

IMAGE FILE FORMAT

There are many image formats you should know about. The three common image formats for the web are **JPG**, **PNG**, and **GIF**.

JPG (also JPEG) is the most popular image type used on the web, because it gives the best quality image at the smallest file size. JPG files use strong **lossy compression** to reduce the file size. JPG images have 24-bit color, which means that millions of colors can be expressed in the image for excellent quality. JPG files have no other special features. Almost all standard images on web pages are JPG files.

PNG also uses **lossless compression**. This does not reduce the file size so much, but it maintains excellent quality. PNG images can be in 24-bit color, but such an image will have a much greater file size than an acceptably compressed JPG file. For example, a 1000-pixel wide image saved at 60% quality as a JPG might be 200 KB in size; the same image, saved as a 24-bit PNG, might be 1.2 MB in size—six times larger!

PNG images can also be saved in 8-bit color, which only allows 256 different colors to be used. This can reduce the size of the image, but it will still be at least twice as large as the JPG image.

PNG images are only used in special situations. One case is when **transparency** is required. Let's say that you have a logo which is circle-shaped. You want the part of the image outside the circle to be transparent, so the web page background will show through. PNG files can do this. A 24-bit PNG has 256 levels of transparency, so the edges of the object will be very smooth, and even drop shadows can be included in the image. An 8-bit PNG, however, only has 2 levels of transparency (off and on), and so the edges look jagged, and no drop shadows are possible. (JPG images *cannot* use transparency at all.)

The other case where a PNG could be used is with an image with very few colors. For example, if you have a logo with just three or four solid colors and not much detail, a PNG image can be smaller than a JPG.

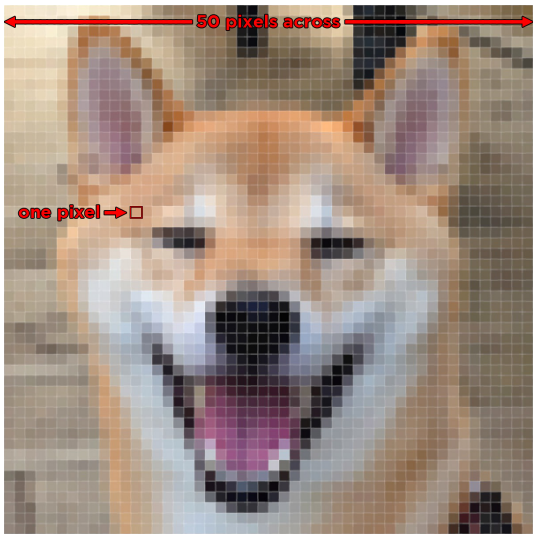
GIF images are all 8-bit color with transparency and lossless compression—very much like 8-bit PNG images. The main difference is that a GIF image can be **animated**, as you have no doubt seen on the web. However, animated images are often not desirable in web design, as many visitors will greatly dislike constant movement of the page.

Like PNGs, GIF images can be smaller than JPG when the number of colors is very limited, but with normal photographs, even with fewer colors, GIF images are larger in file size.

WHICH TO USE?

Use the JPG format at 60% quality for most images, and 40 or 50% for background images (if the result looks OK). Use PNG for images that require transparencies and that have low numbers of colors (e.g., logos). GIFs are rarely used.

Resolution (size in pixels)



Everything on your page must be designed to fit just right. In order to do that, you need to know the size of the content on your page.

A **pixel** is one "dot" of color on your screen.

Display Size is about how big a person's screen is. It can be measured in pixels.

Resolution is the total number of pixels on a screen or image (left & right and up & down).

The best screen resolution to design for is 1366 x 768—in other words, there may be 1366 pixels from left to right, and 768 pixels from top to bottom. The vast majority of people on the web have monitors that size or larger; 95% have a screen which is at least 1000 pixels across. This will probably be the width of your content, or else it will appear too large on browsers for many people. People with larger screens will have blank margins.

You should plan your image sizes so that there will be enough room for text and other layout elements. In addition, the images should be **consistent**—that is, they should have the same width (preferable the same height also), and they should be positioned similarly.

Image Size (pixels)

When you use an image on a web page, the image file you use should have the same (or similar) resolution as the image that appears on the web page. Since most images you take or find will probably be too large, **you usually will have to resize the image**.

For example, when you take an image with the back camera on an iPhone 6, the image size is 3264 x 2448 pixels. This is **much** too big for a web page! The image will not only be too large in resolution (pixel width and size), but it will also have too big a file size. **You must use Photoshop or another image editing program (like Pixlr) to resize the image**. The new size should be what you want to appear on the page.

Image Rights

Images are **intellectual property**, meaning that they are information which is owned by someone, and that person has legal rights which say the image cannot be used without their permission. This is too often ignored on the Internet; it will not be ignored in this class! Every image you use must be accounted for. If you did not take the image yourself, it **must** be cited. You must show who owns the image. This includes images by friends.

There are images with no owner: **public domain** images. These are images which have been given up by their owners, or are so old that copyright no longer applies. An example of public domain would be images issued by the United States government, for instance images from NASA. However, you should cite even public domain images as well.

CODE

4b. Image HTML

The `<img>` tag is a void element—there is no end tag.

There are two essential attributes: `src` and `alt`. **You must always use both.**

```
<img src="ponta.jpg" alt="Ponta in the Park">
```

The `src` attribute shows the location of the image. If the image is in the same folder as the HTML file, then just the image name is OK. If the image is in a folder, you need to add the folder name followed by a slash / .

```
<img src="images/ponta.jpg" alt="Ponta in the park">
```

Images should be within your own web site. You should collect your images in a folder named "images." This makes the site more organized and easy to use. The address you use should be a **local** address—no "http://" and no "file:///"—just the folder and filename.

The `alt` attribute should give a simple, short, plain-text description of the image. The alt text will usually show if the image is broken and does not appear. The alt text is also important for web page reading devices used by people who are blind or have vision problems.

Images are **inline**, meaning they do not break the flow of text. If you place an image alongside text, the image will appear in the line of text on the web page, making the single line of text as tall as the image.

For now, it would be a good idea to place an image or images **between two block elements**. That way, the image will be on a line all by itself. Later, we will learn how to make images stand within text, making the text wrap around them.

Older HTML code included more attributes in the `<img>` tag code. You could set width, height, alignment, borders, and padding. **Avoid using these in your HTML code**. When we learn CSS soon, we will learn how to set those using CSS. Many people still set height and width in attributes, but CSS is easier when you know it, so let's not start bad habits!

HOTLINKING

When typing the `src` attribute, it is possible to use a global address. For example:

```
<img src="http://shiba.me/images/ponta.jpg" alt="Ponta in the Park">
```

This is almost always a bad idea. There are two possible ways to do this. First, there is an image on your site which you want to use. In such a case, you should not use the global address because you do not have to; using a local address is simpler and more direct.

The second situation is far worse: the image is *on somebody else's web site*, and you want it to appear on yours. For example, your web site is, perhaps, on the domain `web.luj.tokyo`; someone else owns and operates `shiba.me`. There is an image on their site, so you use the above image tag. Presto! Their image appears on your site!

This is a really big "netiquette" error, for two reasons. First of all, the image belongs to them, and you are using it without their permission. (If you ask for and gain permission, that is different—but in most cases, people don't ask.)

Much worse, you are using *their* resources to build *your* web site. The image not only belongs to them, it resides on their web server, which they pay to rent. Every time a visitor to your site looks at your page, their server is contacted and uses bandwidth (Internet traffic) to send the image, which is often limited, and again, costs money. In addition, you are effectively stealing the image, which they also own.

Imagine that an organization is holding an marathon race event, and the race runs by your house. Not only do they use your lawn as a staging area without asking, but they use your garden hose to provide them with water, and your outdoor electrical socket to run their equipment. Would you feel okay with that? Probably not! Hotlinking is like that—using another person's property and resources without asking.

You might think that the resources used are small, but often they are not. Once, a web site with thousands of visitors hotlinked to an image on my site—an image I took of a sunset, which was almost a megabyte in size. I noticed when my web host informed me that gigabytes of data was suddenly being accessed on my domain!

There is danger for you, in fact: since the image is hosted on their site, *they can change the image, and it will change on your web site!* All they have to do is upload a different image with the exact same name. That new image will display on your web site, even though you made no changes. I once did this to a hotlinker—I replaced the original image with a much smaller image which simply had the text, "To see this image, go to my site at…" Very soon after I did that, they removed the hotlink—and stopped stealing my resources.

4c. Figure and Figcaption

Images are **inline** objects. While you can float them and treat them like block elements all by themselves, it is sometimes a good idea to use a **block container** for images.

As I have noted several times already, **inline elements should be inside block containers**, and block elements should be inside structural containers. Since images are inline, they should be inside a block.

However, the block element you would try to put it into is the paragraph tag—and in that case, it would be mixed with text. Generally, it is not a good idea to mix images and text on the same line. Therefore, try not to put images inside paragraph tags.

There is one HTML tag created exactly for images: the `<figure>` tag. The word "figure" is often used in academic papers to refer to any image, chart, or illustration included with the text. The tag looks like this:

```
<figure>
     <img src="ponta.jpg" alt="Ponta winking">
</figure>
```

The figure tag has a companion tag: `<figcaption>`. It is not necessary to use this, but it will be useful in several ways as we learn more. The figcaption tag was specifically made to hold the descriptive text for an image. Here is an example of what the HTML could look like:

```
<figure>
     <img src="hanazono.jpg" alt="Ponta winking">
     <figcaption>
          My father visiting Hanazono Shrine
     </figcaption>
</figure>
```

This creates the following on a web page:

My father visiting Hanazono Shrine

In a following chapter, we will learn how to add styles to the figure and figcaption tags so the image can appear any way you wish on the web page.

4d. Finding and Preparing Images

Whenever possible, use your own images.

If you must use someone else's images, then find images which make their usage rights clear. Promotional images for a movie or product are usually fair to use, so long as the owner is given credit, and the image source is made clear.

Otherwise, you should only use images with clearly labeled usage rights. the best place to find images like this is from a page called **Creative Commons Search** (I strongly recommend the "oldsearch" version of the page, the new one is not as good for our uses):

```
https://oldsearch.creativecommons.org
```

From that page, you can search many sites which label their photos by license. A "Creative Commons" license usually means that you are allowed to use an image for free (even if you use it for a business), but you are required to give credit (a citation) to the owner.

In addition, always obtain the actual image file and host it on your web site. Do not hotlink!

Preparing an Image

You rarely find images which are "ready to use." As explained in the design point for this chapter, images on a site should have similar dimensions and ratios. You will not get this result by randomly pulling images from different sources.

Before you begin creating a web site, you should decide on what the shapes and sizes of your images will be. See Chapter 4d for further details!

It is best, in fact, to find images which are larger than you need. That gives you the freedom to crop the images the way you like, and then to reduce the size of the image to fit the image size you have chosen for your page. Photoshop is a great program to use for this, but it is expensive and most people do not have it. It is installed on the lab computers, but that is not helpful if you cannot use them.

Other programs you can use:

Preview (Mac only): Preview comes installed on all Macs. It is a small but powerful app that will handle all your image editing needs.

Paint.NET (Windows only): a long-time Windows app, at getpaint.net.

GIMP (Mac, Windows, Linux): A free alternative to Photoshop. It is a bit more difficult to use, but can be pretty powerful. Find the latest version at gimp.org

Fotor (browser-based): A simple image editor, available online at fotor.com.

Pixler (browser-based): Another simple image editor, available online at pixlr.com/x/

Note: the browser-based web apps usually work best with Chrome, and may ask you to install Adobe Flash.

There are many other image editors that you can find and use, many free for download. You may already have a favorite.

It is very important that your images be both small in file size, and high quality. This is a balance which the JPG image format is best suited for. JPG images can be reduced to the smallest file size with the best image quality through the use of compression.

You must **never** just use images from your camera as they are. They are usually several megabytes in size, and thousands of pixels wide. Your photos will vary from a few hundred pixels at least, to as much as 1000 or 1100 pixels wide. Even background images should usually not be more than 1600 pixels wide.

While your images can be larger than the intended size, they must never be smaller. Also, they should never be more than 2x the size they appear on the page.

As I have mentioned, you will want to decide on exact sizes for the images used on your site—which means that almost all images have to be processed. You will lose points on your projects if you do not do this.

You must adjust your images as explained here.

I strongly suggest the **Crop – Resize – Crop Again – Save** steps:

- **Crop** your image. This is not required in every image, but it should be considered every time. You may want to focus better on the part of the image that is important for your topic and theme.

- **Resize** your image. You will have decided on an image size you would like to use. You will want to resize your image to that size. However, the ratio will probably not be correct. For example, your cropped image may be 2200 x 1300 pixels, but you want the image to be 600 x 400. If you change the width from 2200 to 600, the height will change from 1300 to 355 pixels. Since 355 is too small, that's won't be usable. Instead, **set the height or width so that one of them is exact, and the other one is larger than desired**. In this case, setting the height to 400 will create 677 x 400.

- **Crop Again**. So now you have an image where one dimension is too big—for example, the 677 x 400 image I spoke of. In this case, just crop the dimension which is too large—make the 677 into 600, for example—and now you have the right size.

- **Save** your image for the web. If you require transparency or animation, then you would want PNG or GIF formats—however, such effects require special software and practice. Most or perhaps all of your images should be saved in the JPG format. After cropping and resizing, save your images so you can set the file format, the file name, and the **compression**.

  70% is usually the best balance for size and quality. If your image is 600 x 400 pixels, it will probably be under 100 KB, which is good. Even if your image is 1200 x 800 pixels, it will still be around 200 KB. If the file size gets too big, lowering the image quality to 60% or even 50% may be acceptable. As I noted earlier, 200 KB or under is best, but a little over 200 KB should be fine, so long as it does not go over 400 KB.

Image Editing

In this class, I suggest using Photoshop or Pixlr. Photoshop is installed in our lab, but it is a commercial app and is very expensive to own. There are cheap and free alternatives, but the best downloadable apps very by platform (Mac, Windows, Linux). The online web app Pixlr is a good free alternative because it can be used on any computer.

There are three basic editing skills that everyone in web design should know about: cropping, resizing, and saving. These skills allow you to take a digital image, frame it exactly as you need, and reduce both the pixel and file size to acceptable amounts.

Cropping

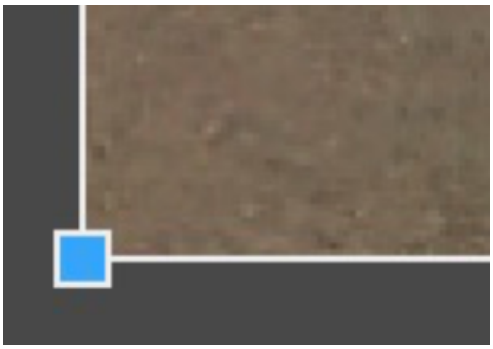
The crop tool allows you to cut away parts of the image you do not want. It also will allow you to create the correct ratio of width to height.

For example, let's say that you have decided that most of your images will be 400 x 300 pixels. That's a 4:3 ratio for width-to-height. If your original image is 1000 x 800 pixels, that is a 5:4 ratio, and when you scale down, the dimensions of the image will not be correct. Cropping can help you fix that.

Most of all, the crop tool can get the image to focus on exactly what you want. For example:

In the image above, we want to focus on the dog and the cherry blossoms. We don't want to see the baby and its mother. We can use crop to correct this.

In most image editing apps, the crop buttons, shown at right, look like corners overlapping with a thin line coming out diagonally to the upper right. When you click on this button in Photoshop, the entire image will be surrounded by a dashed line, and the corners & edges will have **crop marks**. In many online web apps, there is simply a button label "Crop" or "Cropping."



You can then grab one of the crop marks at the edge and drag it to the correct location. In the case of the photo I am using, the ratio is already 4:3, which is the ratio I want to use. If I crop the image "freehand" (sizing each side independently), I will probably lose that ratio.

After you drag the crop marks, you will see the frame of the crop. If the new frame is what you desire, just hit the "Enter" key on your keyboard, and the new crop will be set.

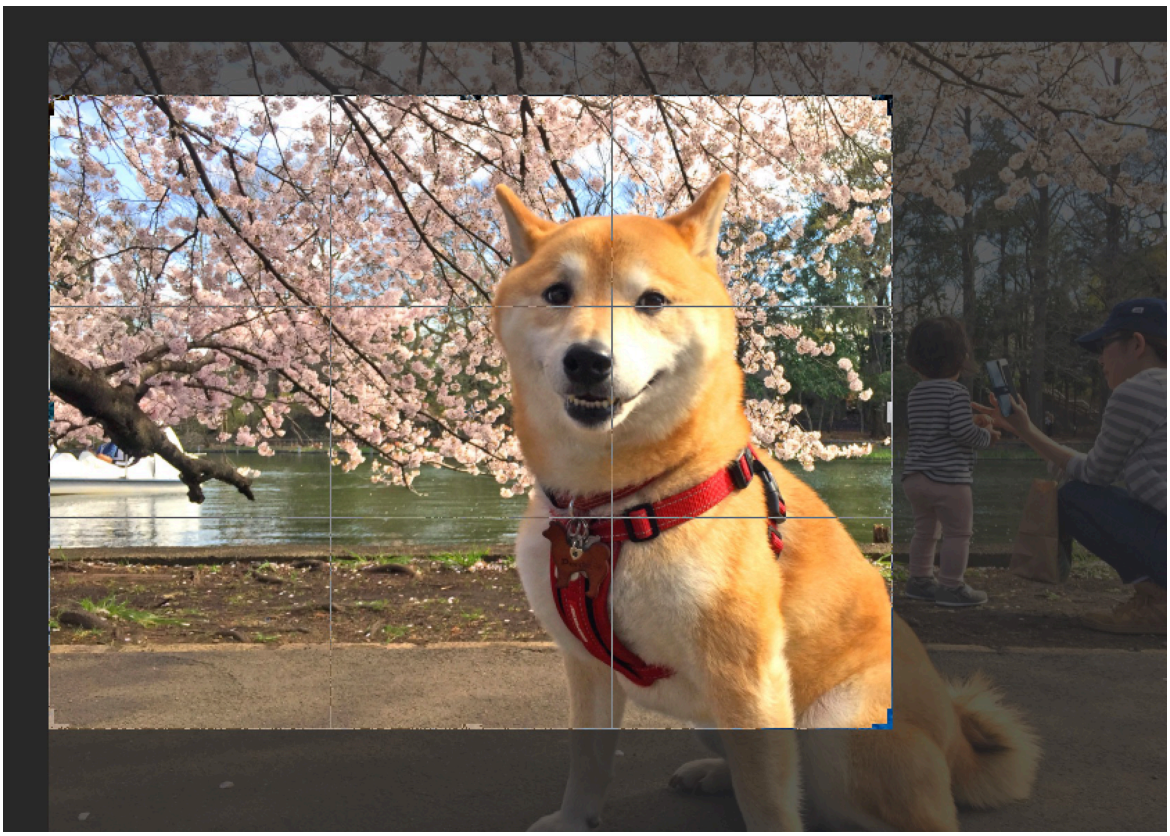


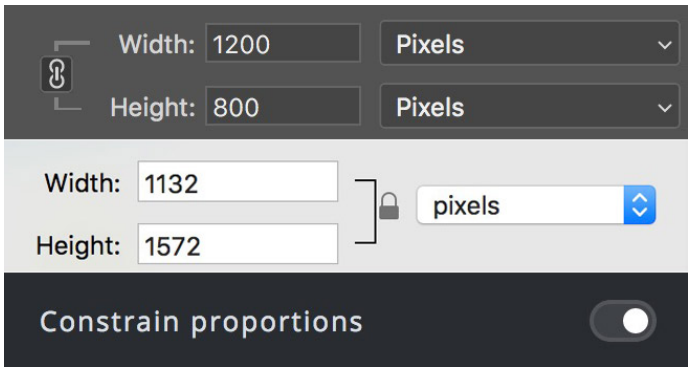
Image Size

The next step is to set the image size in pixels. By cropping, you were able to cut out parts of the image that you did not want, which made the image smaller. However, the remaining image is still probably far too large. In my case, the original image was 3264 x 2448 pixels; after the crop, it is still 2437 x 1828 pixels. This will be two and a half times wider than my entire content area!

True, in the code, I could change the **display size** of the image to, say, 400 x 300 pixels. However, the page would still be using the 2437 x 1828 image, which would also have a file size of 1.4 MB, even with compression.

Since I want to display the image as 400 pixels wide, having an image 2437 pixels wide is excessive—it is more than I need. Therefore, we will use **Image Size** to resize the photo.

In many apps, go to the "Image" or "Tools" menu, and select "Image Size" or "Adjust Size." You will see the dimensions of the image in pixels. Both apps have an option to **constrain** the resize; it often looks like a small lock or chain, or might be a marked switch.
You want to make sure that this is on, otherwise the image will be distorted—the image will appear flattened or stretched in one dimension.



Type in the desired width. The height should change at the same time.

Let's say, for example, that you type "400" pixels as the width. If the height is 300 pixels, and you want a 4:3 ratio, then things are good, and you can click "OK." This exactness is not common, however.

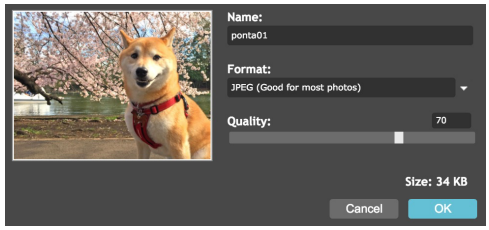
However, if you type "400" and the height becomes 250 pixels, then you have a problem: the height needs to be greater. You want a 400 x 300 image, but if the height is 250, you can't add to it. So instead, make the height 300. This will make the width too large—480 pixels, in this case—but then you can crop down the width and make the whole image 400 x 300 again.

Crop the Image Again

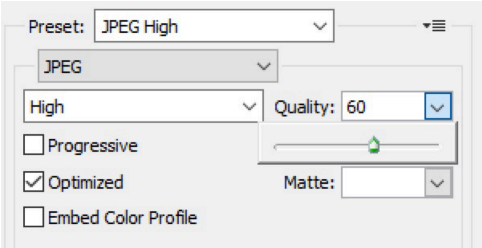
It is likely that your image is exactly right one way (width or height), but too big in the other way. Go back to cropping as described above, but this time reduce the dimension which is too big until it is exactly right.

Saving the Image

Finally, you need to save the image. The three image file types used on the Web are JPG, PNG, and GIF. If you save the image I am using (at 400 x 300 pixels) as a PNG, it would be 286 KB. If you save it as a GIF, it would be 108 KB (though the color quality would be poor). If you save it as a JPG, however, at a decent quality it could be as low as 60 KB.

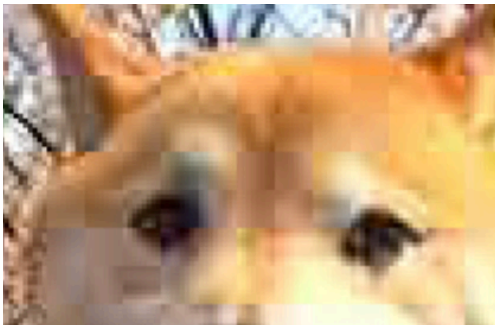


Why use PNG and GIF, then? PNG images are needed if you want to create a **transparency**. GIF files are for images with simple animations. Both PNG and GIF will create smaller file sizes for images which have very simple colors, like an image of a Simpsons character.



A JPG, however, allows for **compression**. Compression means that the file can be decreased in size but not lose much quality. If you Go to the File menu in Pixlr and select "Save," you can choose the JPEG Format, and then use the Quality slider below to choose the amount of compression. 60 is usually the lowest you can go and still have an acceptable quality for your image.

In Photoshop, go to the File menu and choose "Save for Web…." You will see a preview of the image quality I a dialog box, and in the top right corner, you will see the saving options. Ignore the Preset, and choose "JPEG" from the menu below the preset. The quality is to the right, and again has a slider that ranges from 0 to 100.



A setting of 0 on the slider (full compression) will make an image with a tiny file size (just 16 KB in my case), but the image quality becomes terrible (see example at right).

A setting of 100 on the slider (no compression) makes the highest quality image, but the file size becomes large (181 KB).

Typically, the best compromise is a quality of about 60 or 70. This allows for the smallest file size, while keeping a quality which is close to the original.

When finished, save the image using an acceptable filename.

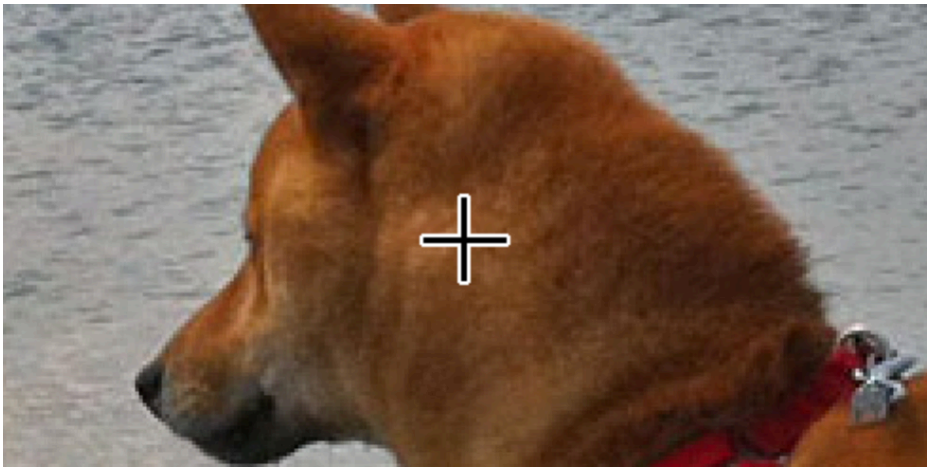
Now, after cropping, resizing, and saving with JPG compression, you have an image which (1) shows exactly the content you want, (2) has the correct ratio, (3) has the correct resolution (size in pixels) that you will use on the page, and (4) is small enough in file size so that your page and all of the images should load quickly, for the best visitor experience.

There is much more that can be done with these image editing apps. For now, these basic skills will allow you to do the most important tasks that you will need.

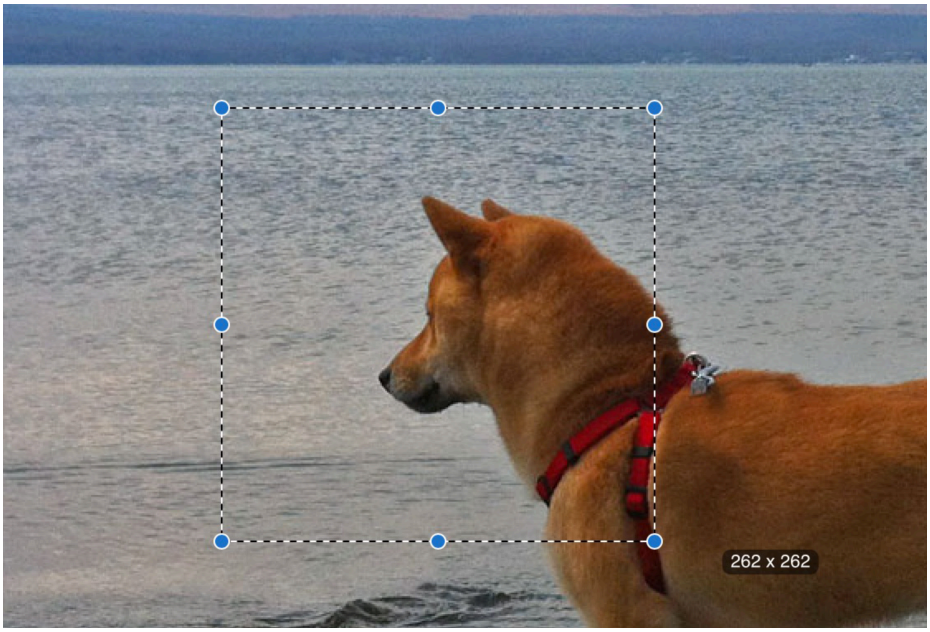
USING PREVIEW ON A MAC

A free program on Macintosh for editing images is called **Preview**. It is already installed on any Mac you use. It is a good, powerful, but simple image editing program (which also edits PDF files). Here is how to use it to crop, resize, and save.

When you open an image in Preview, the cursor is a "crosshairs" (+):

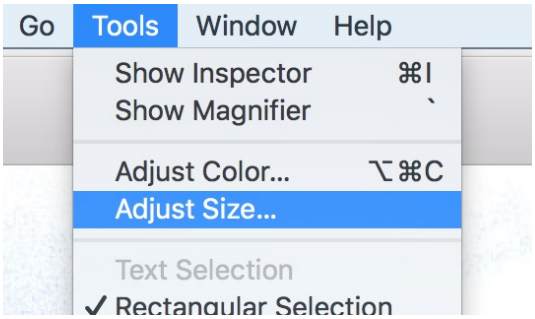


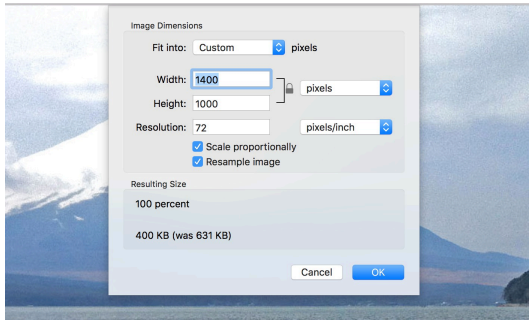
If you click and drag the crosshairs, you can make a box:



If you then choose "Crop" from the "Tools" menu, or else do the keyboard shortcut Command + K, the image will be cropped to the shape of the box.

To resize an image, go to the "Tools" menu and choose "Adjust Size...". You will get a dialog box allowing you to choose the size of the image in pixels.



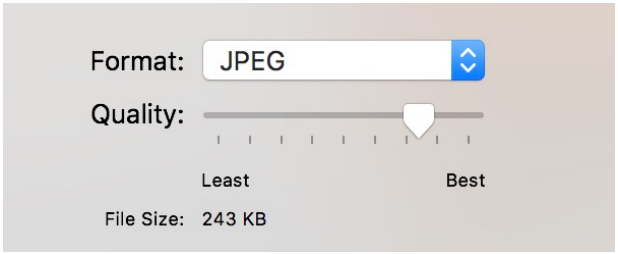


You should always choose to "Scale proportionally," otherwise your images will seem squeezed and distorted.

At the bottom of the dialog, it estimates the resulting size in bytes.

Finally, to save the image, go to the "File" menu. You can save the changed image, and it will overwrite (erase) the original image. You will see that there is no "Save As" command; if you wish to keep the original, then choose "Export...":

In the "Export" menu, you can choose the file type (JPEG) and the quality (somewhere between 60~80% is often best):



This will save the image as a new file. However, remember that Preview automatically saves the changes you make, even if you did not actively save it. If you close it, then it may simply close with all the changes you made. Otherwise, you will get a dialog box asking if you want to revert the image (but not always).

DESIGN

4e. Image Size & Placement

It is not just important to understand images and do what is necessary to display them. You must also **select**, **prepare**, and **lay out the images** in a way that is attractive and appealing. Images will be one of the most important visual aspects of your web site; do not just use them randomly!

I. IMAGE SELECTION

If possible, **find images which are larger than you need**, and then crop and resize them.

Find images which fit the theme of your web site and relate to your web site's topic.

Use images which support the text on the page in that location.

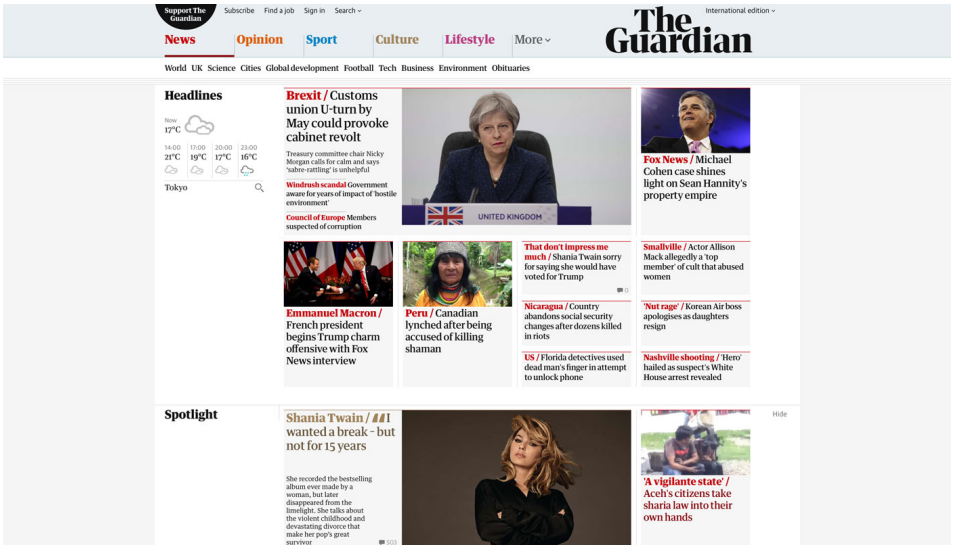
Use images which will not look too different from everything else on the page. If you have nine images with red and orange hues, and one image with bright blue colors, it may look strange (depending on how you use it).

Also, remember that your visitors do not see images the same way that you do; you may want to use an image you took of Florence when you visited there because you know it is of Florence, and it reminds you of your visit. However, the same photograph may not mean anything special to your visitors; they may not understand where it was taken or why you are using it on the page. Make sure the image will be **understood** by anyone.

Find images you are allowed to use. Remember, they should be images with licenses that allow you to use them. Too many people just take and use *any* image. You should be more careful. Starting with the Creative Commons Search page is a good start.

II. DIMENSIONS

Before you begin to create your web site, you should decide on what size images you will use. Images on a web site almost always use specific set sizes. There may be a few different sizes, but all images will fit one of those sizes exactly. Look at the images on these web sites:

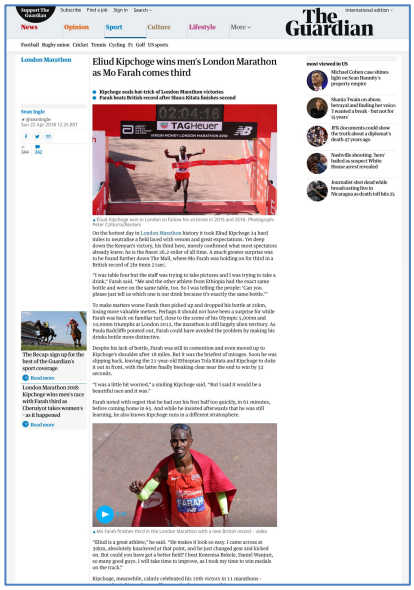


Notice that all the images on the example above fit into exactly 2 sizes. The larger images are 460 x 275 pixels, and the smaller images are all 220 x 130 pixels. Farther down the page, we see images with a different size—140 x 85 pixels—but again, they all fit the same standard. All images are a 5:3 ratio.

This example is of a main page, a site which is displaying only images and titles with minimal text. Let's look at a web page that mostly has text, but also has images to supplement that text. At right is a page to show a few different solutions.

Notice that the page's main content uses two image sizes. One (the coffee beans) goes completely across the column of text, while another (the goats) is smaller than the width of the text, and the text wraps around it.

So, what should you use? A lot depends upon the dimensions of your page. For example, if your text is in a space 900 pixels wide, you have room for 400px with an image, and 500px with text (with some of the text space taken up by the margin space between them). 400 pixels is a fair size for an image, and it should work well.



However, many web sites have multiple columns; in a 1000-pixel-wide space, if the main text only has 650 pixels, then there is not enough space for a sizable image plus text.

The Guardian article at right shows the solution for that: use images that have a width equal to the whole column of text.

This works best in pages that have text columns less than 700px or so. If you have only one column at 1000 pixels in width, a 1000-pixel-wide image will dominate the page, and may be too much.

Therefore, in a wider text column, smaller images on the side would be better, while in a more narrow text column, images with the same with of the column would work better.

Ratios

Another point to consider in using images is the **ratio** of width to height in your images.

For example, images which are 700 x 525, 600 x 450, and 500 x 375 all have the same proportion of width to height—in this case, a 4:3 ratio.

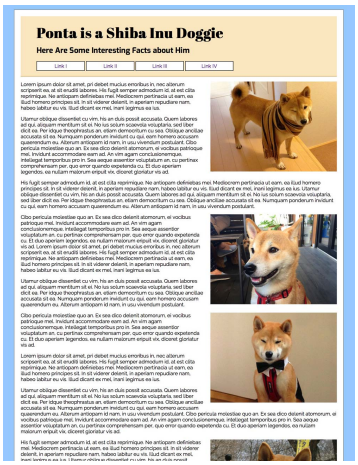
While using images of different ratios is possible, it is best to use images of the same ratio, especially images which are the same size.

If there is a difference in size, it is better that the images are of different *heights*, and not different *widths*. Images which have differing widths look uneven, and break the symmetry and lines of the page.

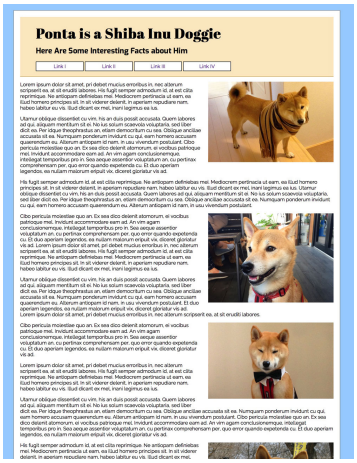
If you do have images with different heights, make sure the differences are not that great.

Below are three pages:

- Layout #1 has images all with the exact same height and width;
- Layout #2 has images with the same width, but different heights;
- Layout #3 has images with the same height, but different widths.



Layout #1



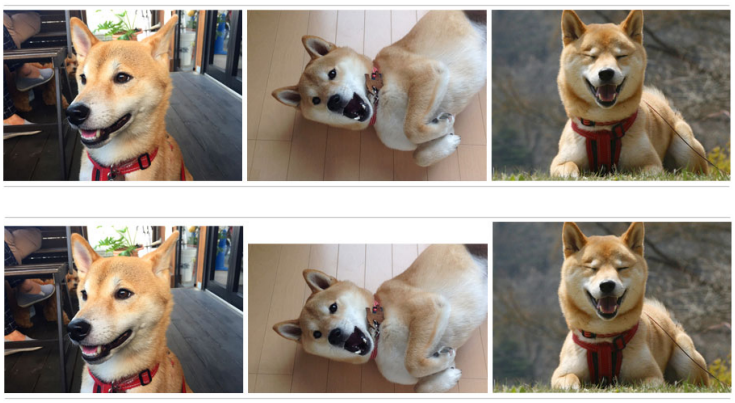
Layout #2

Layout #3

You will note that while the first two styles look clean and very similar, the last one—the one with different widths to the photos—looks the most uneven.

Layout #1, with images of the same width & height, is best. Layout #2 is fine, but Layout #3 is not very good.

Avoid different image sizes, but if you must, then make the height different, not the width. The exception is if you have several images next to each other in a row. In that case, it is important that they all have the same height. Having the same width is also better, especially because it makes it easier to calculate the needed width to fill up the space from left to right.

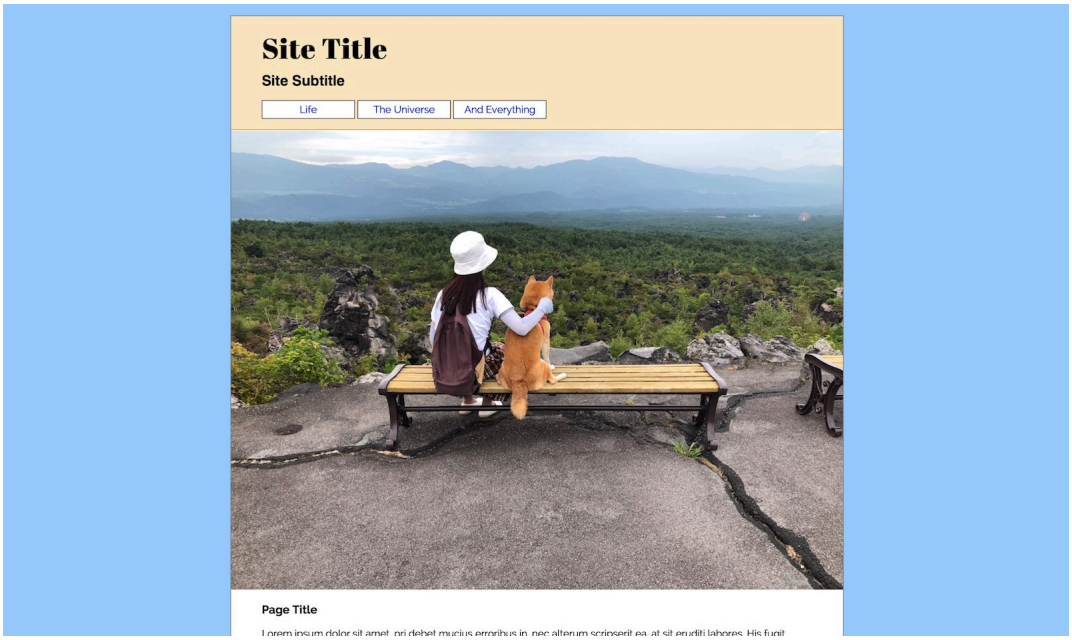


III. Banners

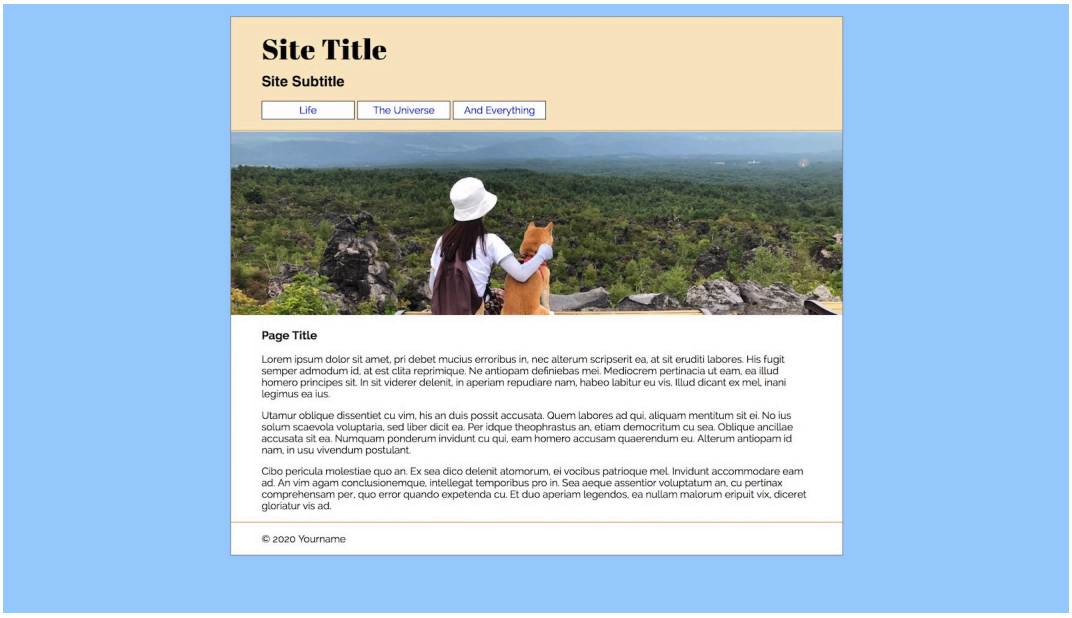
Many pages use what I would call "Banner" style images: large images that are very wide, and not very tall.

A banner image is especially common on main (top or index) pages, as they catch people's attention and can be colorful and interesting.

However, large images have a problem: they are also tall. Too tall. Look at the page example below; even with a small header and nav, the image takes up most of the page.



Instead, you want an image that is wide but not too tall:



This allows for a beautiful, wide photograph with scenic or broad detail while not taking up almost all of the window.

However, taking banner pictures is more difficult than you may think. The images you take with your camera are wide *and* tall. To create a banner image, you will have to crop out much of the image at the top and/or bottom:

Therefore, when you take a banner image, you must frame the photo carefully so the part you want is fully visible, and make sure that you can easily get rid of extra space at the top and bottom.

Here are a few images which are **bad** for banners:

Can you see why they are not good? Most of the interesting detail of the images is shared between the top and bottom of each photograph. It would be impossible to get a good banner image from either of these photos. You would have to cut out the the most interesting parts, and what remained would be so disconnected and partial that it would look very strange.

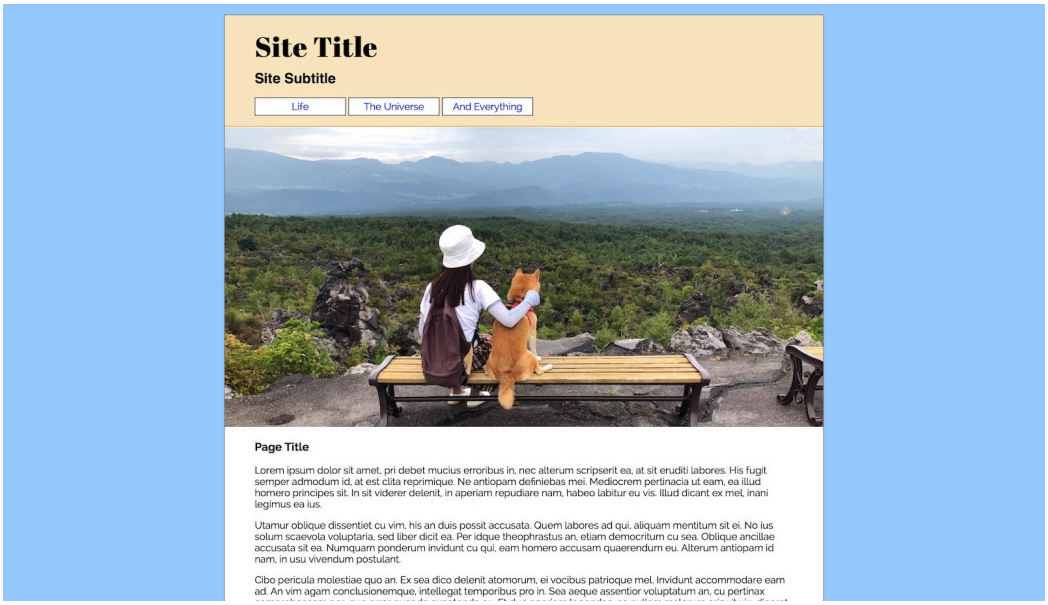
When taking images for banners, always consider these points. Similarly, when you see a view that looks like you could make a good banner, make sure you step back away far enough to get all the width you can. It is better to make the width too great than too small!

Keep in mind that most cameras today, even smartphone cameras, create very high-resolution photos; a width of 4000 pixels is quite common. If your banner image is 1000 pixels wide, you could crop out 75% of the photo on the left and right and still have a clear image!

In this example, the content area of my page is 1000 pixels wide. That's a good width in any case—just about right for most monitors / displays.

There is no strict rule about the ratios of banner images, but the one I used in the first example a few pages ago is 10:3 (1000 x 300 pixels). Any less height and the image starts to look like a ribbon or strip.

However, you can do taller; 10:4 (5:2) is fine, and 10:5 (2:1) is usable, but maybe I would not use something taller than that. A 10:5 / 2:1 banner is exemplified below:

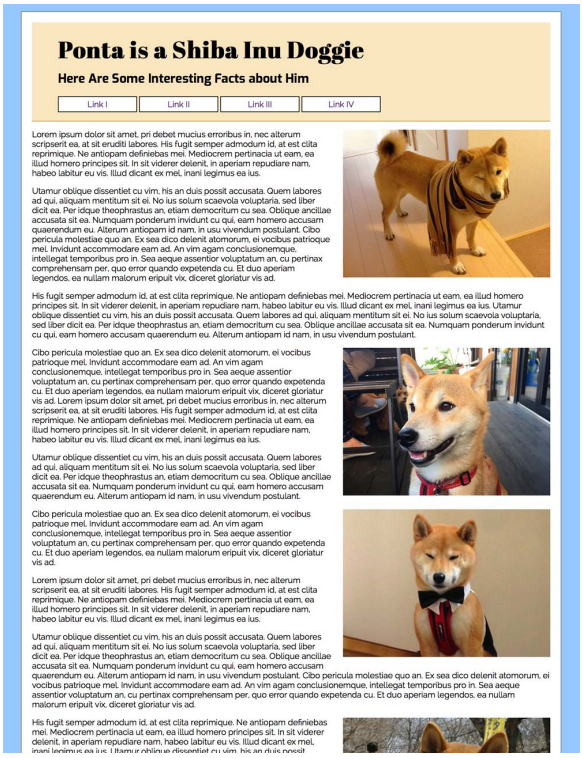


Below is one more example of a good banner image and the result:

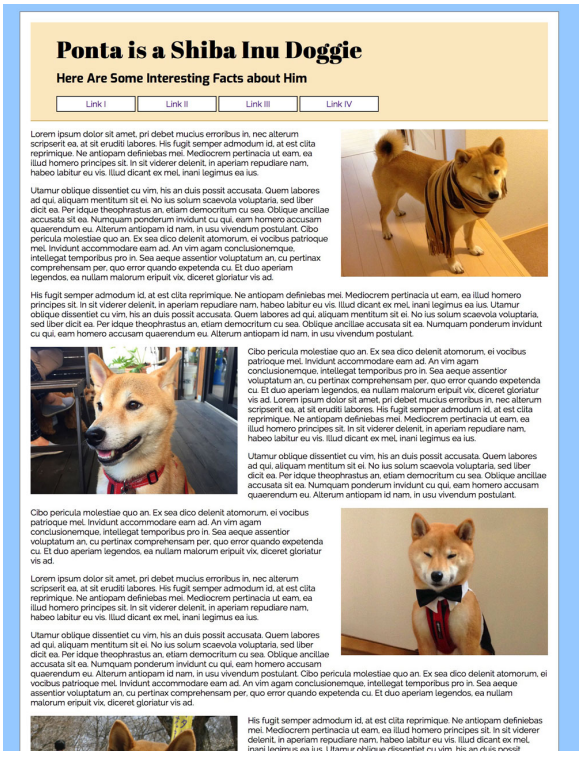
IV. ALIGNMENT

There is a common temptation to stagger images between the left and right. While this could work on the main page of a web site where images dominate and there is little text, on pages with mostly text, this layout does not work.

In Layout #1 below, the images are all aligned to one side. In Layout #2, the images alternate between left and right alignment:



Layout #1



Layout #2

Alternating sides for images on a page is generally considered bad style. While it is possible to do in some cases, it is usually not a good idea, and you should avoid it.

The first reason is that a good layout is usually well-aligned. Go ahead and check out professional web sites where there is a lot of text on the page; you will find them following the standard style of aligning images—as well as having images of at least the same width, if not the same height.

Chapter 4 Code

All the tags we have learned so far are:

Beginning	Ending	Purpose
Page Frame		
<!DOCTYPE html>		Declares that HTML5 is used
<html lang="en">	</html>	Begins and ends the page
<head>	</head>	Begins and ends the head
<title>	</title>	Sets the document title
<meta charset="UTF-8">		Allows all languages to be used
<meta name="author" content="yourname">		Notes who is the author of the page
<body>	</body>	Begins and ends the content area
Structural		
<header>	</header>	Contains the page main title & subtitle
<nav>	</nav>	Contains the links for the site
<main>	</main>	Contains the main content
<footer>	</footer>	Contains copyright & other site info
Block		
<figure>	</figure>	This is a block to contain images
<figcaption>	</figcaption>	This creates a caption below the image
<h1>	</h1>	This is the main page title
<h2>	</h2>	This is the main page subtitle
		Everything inside this is a paragraph
<blockquote>	</blockquote>	Creates an indented paragraph for quotes
		Creates an all-purpose block / box
<h1> ~ <h6>	</h1> ~ </h6>	Creates titles for various page sections
		Contains a numbered (ordered) list
		Contains a bullet (unordered) list
		A list item in an or
Inline		
<img src="" alt="" title="">		Places an image on a web page (void tag)
		Makes text bold
		Makes text bold
		Makes text italic
		Makes text italic
		Allows for specific styling of text in CSS
		Creates a link
 		Breaks inline content; jumps to new line

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

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>
            Title
        </title>
        <meta charset="UTF-8">
        <meta name="author" content="yourname">
    </head>
    <body>
        <div id="wrapper">
            <header>
                <h1>
                    Title
                </h1>
                <h2>
                    Subtitle
                </h2>
            </header>
            <main>
                <h3>
                    This is the Page Title
                </h3>
                <figure>
                    <a href="https://flic.kr/p/6b4vvR" target="_blank">
                        <img src="images/shiba.jpg" alt="A Shiba Inu puppy" title="Image by derfian">
                    </a>
                    <figcaption>
                        Shiba Inu puppies are the best!
                    </figcaption>
                </figure>
                <p>
                    Lorem ipsum dolor sit amet, pri debet mucius erroribus in, nec alterum scripserit ea, at sit
                        eruditi labores. His fugit semper admodum id, at est clita reprimique. Ne antiopam
                        definiebas mei. Mediocrem pertinacia ut eam, ea illud homero principes sit. In sit viderer
                        delenit, in aperiam repudiare nam, habeo labitur eu vis. Illud dicant ex mel, inani legimus
                        ea ius.
                </p>
                <p>
                    Utamur oblique dissentiet cu vim, his an duis possit accusata. Quem labores ad qui, aliquam
                        mentitum sit ei. No ius solum scaevola voluptaria, sed liber dicit ea. Per idque
                        theophrastus an, etiam democritum cu sea. Oblique ancillae accusata sit ea. Numquam ponderum
                        invidunt cu qui, eam homero accusam quaerendum eu. Alterum antiopam id nam, in usu vivendum
                        postulant.
                </p>
                <p>
                    Cibo pericula molestiae quo an. Ex sea dico delenit atomorum, ei vocibus patrioque mel. Invidunt
                        accommodare eam ad. An vim agam conclusionemque, intellegat temporibus pro in. Sea aeque
                        assentior voluptatum an, cu pertinax comprehensam per, quo error quando expetenda cu. Et duo
                        aperiam legendos, ea nullam malorum eripuit vix, diceret gloriatur vis ad.
                </p>
            </main>
            <footer>
                <p>
                    &copy; 2019 Yourname
                </p>
            </footer>
        </div>
    </body>
</html>
```

Chapter 4 Checklist

- ❑ What is “file size”?
- ❑ Why is it bad to have an image with a large file size on a web page?
- ❑ What is the upper limit for a good file size for a web page image?
- ❑ What type of image is an exception, and can be bigger?
- ❑ What file **format** is best for most web images? Why?
- ❑ When is it best to use a PNG image? What is this format's special feature?
- ❑ When is it best to use a GIF image? What is this format's special feature?
- ❑ What is a pixel?
- ❑ What is “resolution”? Why is it important?
- ❑ What are two reasons it would be a bad idea to use an image with 3264 x 2448 pixels?
- ❑ What should an image (or any other file) **not** have in the filename?
- ❑ If you have an image in your site named `doggie.JPG`, what is wrong with it?
- ❑ If you have an image in your site named `f85mbc352.jpg`, what is wrong with it?
- ❑ What is the tag for placing images in a web page?
- ❑ What two attributes are always necessary for each image?
- ❑ What is “hotlinking,” how do you do it, and why is it bad?
- ❑ What are the figure and figcaption tags for?
- ❑ What is a good site to find shared images for free usage?
- ❑ Using an image editing program, what four steps should you take for each image?
- ❑ What is cropping good for?
- ❑ What is resizing good for?
- ❑ What is compression good for?
- ❑ Do you know how to use at least one image editing app for cropping, resizing, and saving with compression?
- ❑ Can you make images for your site all different sizes and ratios? Why not?
- ❑ How should photos be aligned? Is it a good idea alternate between left and right?
- ❑ Why is a banner used? How should you take a picture that you will make into a banner?