Simply a Better Way to Learn!

HTML and CSS Essentials

Student Guide

Academy of Learning
CAREER COLLEGE

HTML and CSS Essentials

Course Code: 5229CC
Version: v2.0 Mar18

DISCLAIMER

While Academy of Learning College takes great care to ensure the accuracy and quality of these materials, all material is provided without any warranties or representations of any kind or with respect to use or performance thereof, whether expressed or implied, statutory or arising from otherwise in law or from a source of dealing or usage in trade, including but not limited to implied warranties or conditions of merchantable quality or fitness for the particular purpose of the User.

Academy of Learning
CAREER COLLEGE

HTML and CSS Essentials

A Course Information

Overview

Academy of Learning College offers the HTML and CSS Essentials course in partnership with **Sessions.edu Online School of Design**. Founded in 1997 by renowned designers and educators from three continents, Sessions.edu is an accredited online school of design and new media.

In **HTML and CSS Essentials** course, you'll learn just how fun and creative hand coding and designing Web sites can be. Working with a professional Web designer, you'll discover how to create Web pages with HTML and control page design and layout with CSS. Working with a pro Web designer, you'll get up to speed with current Web design practices including HTML5, CSS3, semantic coding, Web fonts, and responsive design. By the end of the course, you'll have designed several Web sites, including your first responsive Web site.

Lesson One begins with an overview of today's Web coding environment. You'll learn how HTML works in combination with CSS and JavaScript, and you'll establish a foundation for writing clean, standard-compliant code. Using a text editor, you will create and structure an HTML page and add images and hyperlinks, using important HTML tags used to structure content. To prepare for your assignments, you'll also learn how to organize Web site files and FTP them to the Web.

In **Lesson Two**, you'll learn the benefits of using CSS to control the design of a Web site. Through practical, step by step examples, you'll learn the three primary ways of applying CSS to an HTML page, explore how to correctly write CSS rules, and then apply your design styles using CSS selectors: classes, IDs, and spans. You'll also get an overview of how to use color and typography styles from a design perspective.

Lesson Three, begins by showing you how to apply CSS using the box model to style elements using borders, padding, and margins. This design approach will help your content breathe. You'll explore the ins and outs of CSS positioning, learning how relative positioning, absolute positioning, and floats are used to lay out page sections and create fixed and fluid one and two-column layouts. The lesson wraps up with some essential troubleshooting tips for CSS positioning and floats.

Lesson Four explores this new world with a look at how to use HTML5 uses semantic elements to mark up Web pages to improve layout control and accessibility. Since HTML5 has set new standards for multimedia, you'll learn how to audio and video content to your Web pages and learn principles for testing sites on multiple platforms and browsers.

Lesson Five explores various techniques that will improve your site designs. First, you'll examine how to create and style horizontal navigation and how to use CSS height and width properties to standardize layout proportions. Then you'll gain an introduction to the many new typographical options offered by Web fonts and Google fonts. Finally, you'll explore some simple but awesome CSS3 effects that will take your Web sites out of the ordinary.

Lesson Six you'll learn how responsive sites work, and explore the concept of media queries and how to include them in your style sheets. This will equip you to actually build your own responsive site by working for a framework or boilerplate.

Prerequisites

There are no prerequisites for this course.

The Online Environment

Accessing HTML and CSS Essentials

To access the online environment log in to MyAOLCC learning hub and follow the instructions in course information page. Note that this course is delivered through Sessions.edu Canvas learning management system.

What Is Expected of You

Read the course material carefully, and make your own notes as required. At the end of each lecture is an exercise. Once you have read a lecture, complete the associated exercise. Completed exercises **must be submitted** for evaluation via the **Dropbox** folder located within each lesson. All exercises presented within the course must be successfully completed before writing the Academy of Learning final exam.

Finding Assistance

Any general course administration questions or questions about your computer, software programs, or using the Internet should be directed to your school facilitator.

Any course-content related questions should be addressed to the Academy of Learning College online instructor. Use email within **MyAOLCC** learning hub to send an e-mail query to your online instructor. Academy of Learning College online instructor will respond to your query within 24 hours (business days).

Please check for additional reading and help provided in the **Student Library**, found by clicking the **Learn** button, were you can access relevant learning content of numerous digitized books.

Behavior and Conduct

You are expected to behave with respect towards your instructors and fellow students. Actions that demonstrate failure to respect instructors and fellow students include:

- Plagiarism
- Posting obscene material to discussion groups or to one's instructor
- Verbal hazing and/or derogatory remarks degrading an individual's gender, race, religion, national origin, sexual orientation, or disabled status
- Email flaming (sending hostile email)
- Privacy infringements

Committing any of these actions will result in disciplinary action.

Tips for Successful Online Training

In order to successfully complete the course, it is recommended that you:

- Read the course material carefully and thoroughly.
- Do all your exercises/assignments to the best of your ability.
- Accept the guidance and critiques offered by your instructors.
- Participate in the online discussions.

Miscellaneous Issues

Screenshots

You may notice that the screenshots of an application are different from what you see on your computer. This is because Sessions.edu uses screenshots from a Macintosh computer. The differences are usually minimal.

Evaluation Guidelines

1. All exercises presented within the course as well as the Academy of Learning final exam, must be successfully completed in order for Academy of Learning College to consider the course complete.

2. Your final grade is a percentage grade based solely on the Academy of Learning final exam. The Sessions.edu grade does not reflect on your academic record. However, you may only write the Academy of Learning final exam once you have received the notification of successful completion from Sessions.edu instructor. The notification of successful completion is sent to your personal email once all exercises/projects presented within the course are successfully completed and graded. You must present this proof of successful course completion to your school facilitator before attempting to write the Academy of Learning final exam.

3. Each **exercise** must be posted via the **Dropbox** located within each lesson for evaluation. The Sessions.edu instructor will evaluate each exercise based on a set marking scheme that corresponds to stated performance objectives, and will send a critique and feedback together with a grade to you. You should expect a response from Sessions.edu instructor to your assignment within 2 business days.

B Publishing Your Files

In this course, you will need a Web host to post course projects online for the purpose of evaluation. You should use a Web host that does not place banner ads or pop-up ads on your site as these will interfere with your pages and graphics and should be avoided.

If you do not have Web space to publish your coursework to, three options exist for acquiring Web space:

1. Free Web Hosting Service Providers

Free Web hosting services provide free Web space usually with minimal services (e.g. bandwidth and disk storage space). The free Web space provided to you is generally more than adequate for the purposes of completing your course exercises and assignments. However, if some of your Web sites and files are very large, you may have to remove previous Web sites from your Web space or publish over them as you progress through your courses. Some free Web hosting service providers are:

www.awardspace.com
www.biz.nf
www.agilityhoster.com
www.x10hosting.com

Note that we don't endorse these Web hosting providers, but they should provide ad-free Web space to allow you to upload your assignments without interference from forced ads like pop-ups or banners.

You do not need to purchase a **domain name** (e.g. www.yourname.com) if you use a free Web hosting service. To sign up for free Web space, go to the free Web hosting provider's Web site and follow the step-by-step instructions. The process should only take a few minutes. Please review your hosting provider's policy on inactivity. If you have not made any changes or nobody has accessed the site in a specified timeframe, your Host may delete your site.

2. Internet Service Providers (ISPs)

If you have a personal subscription to a high-speed internet service, such as those provided by Bell, Rogers, or Cogeco, you may already have free Web space available to you. The advantage of using these services is that they do not display advertisements on your Web site. However, your Web site will still be located in a subfolder or folder of your ISP's Web site. Check your ISP's Web site for details.

3. Commercial Web Hosting Service Providers

If you wish to host your Web sites and files using your own **domain name** (e.g. yourname.com) you must purchase the domain name and a **hosting plan** from a commercial Web hosting service provider.

> **Note**: The purchase of a domain name and hosting plan is *not required* by Academy of Learning College and Sessions.edu in order to fulfill the requirements of this course.

You may wish to shop around before purchasing a domain name and hosting plan as the fees Web hosting service providers charge may vary considerably. Entering "Web hosting" in a search engine like Google (www.google.com) will yield links to additional Web hosting service providers.

Additional Considerations

You may be charged a fee if you go over your allotted disk space or bandwidth. These fees can be expensive if you choose a plan that has far less than what you need. Talking to a representative from a hosting company should help you decide on a plan that suits your current needs, and that is scalable if you intend to expand your Web presence.

When selecting a Web hosting plan, the amount of technical support available should also be considered. Some Web hosting service providers provide support by Email only, while others provide 24/7 telephone support.

C Glossary

In the table below are definitions of terms you will often encounter in the fields of Web and graphic design.

Term	Description
Absolute positioning	The positioning of a Web page element at an exact location in a Web page. Web page elements that use absolute positioning are not inline with text, and therefore do not move with text.
Adobe Authorware	A visual authoring tool for creating rich-media e-learning applications for delivery on corporate networks, CD/DVD, and the Web. Develop accessible applications that comply with learning management system (LMS) standards.
Adobe Coldfusion	An application server that enables developers to rapidly build, deploys, and maintains robust Internet applications for the enterprise. ColdFusion, available in version 9.0 and below, allows developers to condense complex business logic into fewer lines of code.
Adobe Director	A software that helps users create and publish compelling interactive games, demos, prototypes, simulations, and eLearning courses for the web, Mac and Windows® desktops, DVDs, and CDs. This software allows users to integrate virtually any major file format, including FLV and native 3D content, for the greatest return on your creativity.
Adobe Dreamweaver	A WYSIWYG Web authoring program.
Adobe Fireworks	A Web graphics creation and editing program that has both bitmap and vector tools.
Adobe Flash	A vector graphics-based animation program that enables users to create interactive animated sites and elements. Web site visitors can view content created with Flash if they have the Adobe Flash Player plug-in.
Adobe Illustrator	A vector graphics creation and editing program produced by Adobe Systems Inc.
Adobe Photoshop	A bitmap image and vector graphics creation and editing program produced by Adobe Systems Inc.

Term	Description
ARPA	Advanced Research Projects Agency. An agency of the US Department of Defense that developed the ARPANET, which eventually became the Internet. ARPA also developed the Berkeley version of Unix and TCP/IP protocol.
ASCII	American Standard Code for Information Interchange. The American standard code numbers used by computers to represent the English characters in text.
ASP/.asp/.aspx	Active Server Page. A Web page containing one or more scripts written in either VBScript or JavaScript that are processed on a Microsoft Internet Information Server before the Web page is sent to the user's browser. Active server pages have the extension .asp or .aspx.
AVI/.avi	Audio Video Interleave. An audio/video file format developed by Microsoft. AVI files have the extension .avi.
B2B	Business to Business. Electronic commerce between businesses.
B2C	Business to Consumer. Electronic commerce between businesses and consumers.
Bandwidth	The difference or width between the highest and lowest frequencies of an electronic signal across a given transmission medium.
Berners-Lee, Tim	The creator of the World Wide Web.
Bitmap image	An image that is composed of pixels. Bitmap images are also called "raster images" or "raster graphics". Bitmap images are ideal for displaying subtle color gradations on the Web.
BMP/.bmp	A Microsoft Windows graphic format for bitmap images.
Body	The section of a Web page that contains text and graphics.
Bricks-and-mortar	Traditional businesses with retail outlets that customers can visit, as opposed to virtual or online businesses.
Broadband	A communications network that is capable of supporting a range of frequencies, such as audio, data, and video.

Term	Description
Button	In a graphical user interface (GUI) system, a button is an area within an interface that results in an action when it is clicked with a mouse button.
Cache	A computer memory that holds data that has been recently accessed, and therefore speeds up access to that same data. It is also called "cache memory".
CGI	Common Gateway Interface. A standard that specifies how programs on Web servers can be run from Web pages. Many CGI scripts on the Web are used to process information that users enter in forms.
Clicks-and-mortar	Businesses that have retail outlets customers can visit as well as Web sites from which customers can order.
Client-side	Refers to a program or script that runs on a client (desktop computer), as opposed to a server.
CMYK color model	A color model that describes colors according to the quantities of cyan, magenta, yellow, and black present in each color. The CMYK color model is mostly used in printing.
Compression	The process by which the space needed to store or transmit data is reduced.
CSS	Cascading Style Sheets. An extension to HTML that is used to apply consistent style information (e.g. color, font, font size) across multiple pages of a Web site. Style information can be included in an HTML file or in a separate CSS file that can be attached to one or more HTML files.
Cyberspace	A term coined by William Gibson in 1984 to refer to a technological interface that lets humans communicate with each other, even though they may not be physically near each other.
Data capacity	The amount of data that can be sent via a communications channel in a given period of time. Data capacity is often expressed as bits per second.

Term	Description
DHTML	Dynamic HTML (DHTML) is a set of innovative features originally introduced in Microsoft Internet Explorer 4.0. DHTML enables authors to create visually compelling Web sites by enabling authors to dynamically change the rendering and content of a Web page as the user interacts with it. An example of DHTML includes having the color of text change when a user positions the mouse over it.
DNS	Domain Name System. The way in which Internet domains are located and translated into Internet Protocol (IP) addresses.
Domain name	A name that identifies a site on the Internet. For example, academyoflearning.com.
DPI	Dots per Inch. An indicator of resolution for printers, scanners, and monitors.
Dynamic	Refers to items that do change or interactive elements of a Web site. In Web development, a dynamic Web page is one that is assembled on the fly from information in a database when a user performs an action.
E-business	The buying and selling of goods and services, servicing of customers, and the collaboration of business partners on the Internet.
E-commerce	The buying and selling of goods and services across the Internet, the transfer of funds using the Internet, the use of Electronic Funds Transfer (EFT), smart cards, digital cash, and all other ways of doing business over digital networks.
Email	Messages that are transferred from one computer user to another across the Internet.
File compression	The compression of data within a file in order to reduce the size of the file and increase the speed at which it is downloaded.
File extension	A two to four letter suffix that follows a filename and is separated from the filename by a period. A file extension indicates the type of data that is stored in the file.

Term	Description
File format	The layout of a file with respect to how the data within the file is organized or encoded. The file format indicates whether the file is a binary or ASCII file and specifies how the information is organized.
Font	A set of text characters in a particular size and typeface.
Footer	Specified lines of text that appear in the bottom margin of a page and are usually repeated throughout a document.
Frames	A frames page is a Web page that is divided into two or more independently controlled sections that contain other Web pages.
GIF animation/Animated GIF	A series of static GIF files that are displayed one after the other, or, on top of each other, to give the effect of motion or animation.
GIF/.gif	Graphics Interchange Format. A graphics file format originally developed by Compuserve and based on algorithms developed by Unisys. GIF supports color, different resolutions, and data compression. GIF files contain up to 256 colors, are relatively small, and download quickly. The GIF format is best used for logos, buttons, and icons.
Global navigation	A set of hyperlinks that are used for navigating the main pages of a Web site, such as the Home page, the About Us page, and the Contact Us page, and are displayed on every page in a Web site.
Google	A web search engine designed to search for information on the World Wide Web, owned by Google Inc.
Gradient	A progressive blend of two or more colors that is often used to show depth and/or perspective in images.
Graphic design	The creation and modification of a picture or still image on a computer.
GUI	Graphical User Interface. The use of pictures in a program interface, instead of words or commands to represent the input and output of a program.

Term	Description
Hand-code	The process of creating Web pages or a Web site by writing lines of HTML, JavaScript, or VBScript code without using the automatic code generating capabilities of a Web authoring tool.
Header	In a document, a header is one or more lines of texts or images that appear in the top margin of a page. In an Email message, the header is the part of the message that lists the author, the recipient or recipients, the message priority level, the date the message was sent, and the subject line.
Hexadecimal system	A base-16 number system that consists of 16 unique symbols: the numbers 0 to 9 and the letters A to F. In the field of computing, the hexadecimal system can represent every byte as two consecutive hexadecimal digits. A hexadecimal system is used to specify colors that are used in Web pages.
Home page/Homepage	The first page that a Web user arrives at when entering a Web site's URL in a Web browser.
Host	A host is a computer that acts as a Web server for one or more Web sites, or a company that provides such computers. A host can also be defined as a computer that has TCP/IP access to a network.
HTML	Hypertext Markup Language. The language that is used to create hypertext on the World Wide Web. In HTML, tags are used to tell a Web browser how to display the words and images on a Web page.
HTML editor	A program that lets you create and edit HTML files.
HTML tag	A piece of HTML code that tells a Web browser how to display a document or a part of the document. HTML tags are enclosed in angle brackets. For example, <font>. HTML tags can be either opening or closing tags.
Hyperlink	Text, or a graphic, in a hypertext document that when clicked will take a user to another place in the same document, or, a new document.
Hypertext	A system in which documents or objects contain links that cross-reference other documents or object. The term was coined by Ted Nelson in the mid-1960s.

Term	Description
Icon	An image that represents an object or a program. When an icon is clicked, an action is performed.
Image optimization	Also called image compression, image optimization is the process of reducing the number of bytes in a graphics without degrading the quality of the image too much.
Information architecture	The management or distribution of information blocks (objects) within a Web site's visual design and navigation system.
Internet	A worldwide network of computer networks connected to each other via the TCP/IP protocol.
Internet Explorer	A World Wide Web browser produced by Microsoft Corporation. Internet Explorer lets you view Web pages.
JavaScript	This client-side scripting language is one of the easiest and most powerful ways to customize PDF files. Based on JavaScript version 1.5 of ISO-16262 (formerly known as ECMAScript), JavaScript in Adobe Acrobat software implements objects, methods, and properties that enable you to manipulate PDF files, produce database-driven PDF files, modify the appearance of PDF files, and much more. Beginning with Acrobat 7, there is now support for multimedia, improved printing control, controlling layers, 3D support, and more.
JPEG/JPG/.jpg	Any still image file produced according to the image compression standards set by the Joint Photographic Experts Group. The JPEG format is one of the three most common image formats in the Web. The other common image formats are GIF and PNG. The JPEG format is used for displaying photorealistic content.
Kilobyte	A unit of measurement equal to 1024 bytes and abbreviated as K or KB. In order to distinguish between a decimal K (1,000) and a binary K (1,024), the Institute of Electrical and Electronics Engineers recommends using a lowercase k for a decimal kilo and a capital K for a binary kilo.
Local navigation	A set of hyperlinks that are used for navigating within a specific section of a Web site, and are not displayed on every page in the Web site.

Term	Description
Look-and-feel	The general appearance of a program's user interface. The look and feel of a Web site can include the types of icons and buttons used as well as the shortcut keys required to initiate commands.
Lossless compression	A data compression technique that does not reduce the data in a file. Lossless compression is a feature of GIF files.
Lossy compression	A data compression technique used with the JPEG and MPEG file formats to reduce the size of a file by excluding unnecessary data.
Megabyte	When describing data storage, a megabyte is 1,048,576 bytes or 1024 Kilobytes. When describing data transfer rates, a megabyte is 1,000,000 bytes.
Microsoft FrontPage	A WYSIWYG Web authoring and Web site management program produced by Microsoft Corporation.
Modem	A device or program that converts digital signals from a computer to analog signals that can be sent via a telephone line, and converts analog signals to a digital signal for a computer.
Monitor resolution	The maximum number of pixels that can be displayed on a computer monitor. Monitor resolution is expressed as the number of horizontal pixels by the number of vertical pixels. For example, 1024 × 768.
Monitor size	Also called the screen size, the monitor size is the distance from one corner of a screen to the opposing corner of the screen as measured in inches.
Mosaic	The first graphical Web browser. Mosaic was developed by Marc Andreessen, Eric Bina, and others at the National Center for Supercomputing Applications in Illinois in 1993.
MOV/.mov	An audio/video file format developed by Apple Computer and playable on Windows operating systems through the QuickTime Movie Player plug-in.
Mozilla Firefox	A free and open source web browser descended from the Mozilla Application Suite and managed by Mozilla Corporation.

Term	Description
MPEG/.mpg/.mp3	A file format and audio/video compression standards developed by the Moving Pictures Experts Group. The file format MP3 was derived from MPEG-1. MPEG-1 is intended for CD-ROMs. MPEG-2 is intended for broadcast quality video. MPEG-4 is a graphics and video compression standard.
Multimedia	The combination of text, sound, and/or motion video as presented by computers.
Navigation bar	A set of hyperlinks or directional tools used for navigating a Web site.
Netscape	Netscape Communications Corporation, a company founded in 1994 by James H. Clark and Marc Andreessen, in 1994. Netscape Communications Corporation engages in the development, marketing, sale, and support of enterprise software solutions. In 1998, Netscape became part of America Online.
Opera	A Web browser produced by the Norwegian company Telenor.
Page element	An object that is included in the content of a Web page, such as text, an image, a plug-in, or a table.
PERL	PERL is a highly capable, feature-rich programming language designed for processing text by Larry Wall in 1987. PERL is often used to write CGI scripts and can be used on the Windows, Unix, and Mac platforms. PERL, specifically version 5, is suitable for both rapid prototyping as well as large scale development projects.
PHP	PHP is a widely-used general-purpose scripting language that is especially suited for Web development, used mostly on Linux Web servers, and can be embedded into HTML.
Pixel	Originally called “picture element”, a pixel is the basic unit of programmable color in a computer image or on a computer screen. On color monitors, each pixel is composed of a red dot, a blue dot, and a green dot.
PNG/.png	Portable Network Graphics, a bitmapped graphics lossless file format. PNG files are supported in Internet Explorer and Mozilla Firefox.

Term	Description
Pop-up window	A window that appears in a Web browser when an option is selected, a function key is pressed, or a Web page is opened.
Pure-play Internet companies	A company that sells only over the Internet, as opposed to bricks-and-mortar and clicks-and-mortar companies.
Relative positioning	The positioning of a Web page element at a location that is inline with text. Relative positioning is used with animation and DHTML effects.
RGB Color Model	A color model that describes colors according to the quantities of Red, Green, and Blue that are present in each color. The RGB color model is mostly used for images that are to be displayed on computer screens or displays.
Rollover	Also called a "mouseover", a rollover is a JavaScript script that swaps a page element (text, or an image) with another page element when a user rolls a mouse over a page element.
Server	A program or computer that provides services to itself or other computers or programs. With respect to the World Wide Web, a server stores Web sites and responds to requests for Web pages and page elements from clients (Web browsers) that are connected to it via a network.
Server-side	Refers to a program or script that runs on a server in a client/server system.
Server-side include	A variable value that a server can include in an HTML file before it sends it to the client.
SMTP	Simple Mail Transfer Protocol. The standard protocol for sending Email messages.
SQL	Structured Query Language. The standard query language originally developed by IBM and used for requesting information from and updating databases. SQL is often embedded in VBScript.
Static	Refers to something that does not change. In Web development, a static Web page is one that cannot return information.
Streaming	The process of transmitting audio or video in real-time as it is downloaded via the internet.

Term	Description
Tag	In markup language such as HTML, a tag is a piece of code that describes the layout and formatting of text and page elements to a Web browser. Tags can be inserted immediately before text or a page element (opening) or immediately after text or a page element (closing).
TCP/IP Protocol	Transmission Control Protocol over Internet Protocol. The basic communication language or protocols that make the Internet possible.
Template	In Web design, a template is a page that can contain specified page settings, formatting, and page elements.
Text editor	A program that lets you create, edit and save text files.
TIFF/TIF/.tif	Tagged Image File Format. A file format used for bitmap images. TIFF files have the extension .tiff or .tif and are supported on both PC and Mac platforms.
Typography	The art of composing and printing type.
URL	Uniform Resource Locator. The form of Internet addresses typically used on the World Wide Web. The first part of a URL describes the access method or protocol (e.g. http) and the last part describes the location of the file (e.g. the IP address or the domain name).
Usability	A product's ability to accomplish the goals of its users.
VBScript	Visual Basic Script. A Web scripting language loosely based on Visual Basic and developed by Microsoft. VBScript is supported by Internet Explorer.
Vector graphic	Also called object-oriented graphics, vector graphics are images created as mathematically-defined lines and shapes. Vector graphics downloaded quickly and can be scaled and transformed without losing any resolution or quality.
WAV/.wav	An audio format developed by Microsoft and IBM. WAV files have a .wav extension and can be played by nearly all Windows applications that support sound.
Web authoring tool	An HTML editor.
Web browser	A program that lets users locate and view Web pages.

Term	Description
Web site	A site on the World Wide Web that contains one or more related Web pages and is accessible via a URL.
Web-safe color	A color that will be accurately displayed on PC and Mac platforms and in different Web browsers.
World Wide Web	A hypertext system of sites and services that resides on the Internet.
WYSIWYG	What You See Is What You Get. A user interface or application that lets a user view and print a document exactly as it appears on screen.
XML/XHTML	Extensible Markup Language. A programming language based on SGML that lets users interchange data between different applications.

Simply a Better Way to Learn!

HTML and CSS Essentials

Lecture Content

Academy of Learning
CAREER COLLEGE

HTML and CSS | An Intro to HTML

sessionscollege FOR PROFESSIONAL DESIGN

Course Developer: Hannah Shaffer
Instructors: Hannah Shaffer, Piper Nilsson
Layout: Patricio Sarzosa
Editors: Joshua Trimmer, Gordon Drummond

Course Developer

Hannah Shaffer

I've been designing Web sites for more than ten years, and it's still exciting to me to see a site design come to life. I'm looking forward to sharing with you the fundamentals of coding for today's Web in this course.

Lecture 1

An Intro to HTML

Ready to journey down the deep, dark rabbit hole of web design? That's great, but it's easy to get lost down there!

In this lesson, we'll begin our adventure by learning how HTML is used to create and structure web pages.

To make sure we know where we're going, it's important to know where we've been. Let's prepare for the descent with a brief introduction to the history of HTML.

With such an adorable guide, it's hard to feel intimidated by code!

Learning Objectives

In this lecture, you can expect to:

- Learn the basics of how today's web coding environment evolved.
- Learn the benefits of writing clean, standard-compliant code.
- Learn how to create an HTML document and structure page content using common tags.
- Learn how to add images and hyperlinks to web pages.
- Learn how to reserve web hosting and FTP files to the web.

A Brief History of HTML

To really grasp the way modern HTML works with other web languages, it's important to understand a bit about the history of HTML. HTML, which stands for **Hyper Text Markup Language**, is a language for describing documents on the web.

Notice that I'm not calling HTML a programming language. HTML simply defines the structure and basic presentation of the information on a web page.

Pretty much every website you visit, from Etsy to Reddit to Taco.com, is made up of HTML. Large sites like Etsy and Reddit utilize a number of different languages to display web content. Taco.com, on the other hand, is a relic of an earlier time—a time when few tools existed to beautify a web page.

note

HTML describes the structure of a web page to a browser like Chrome or Firefox.

note

CSS describes how an HTML document should look in a browser.

'Twas a kinder, simpler time; a time when a site called Taco.com didn't actually have anything to do with tacos.

A quick glance at Taco.com reveals that, while functional, it isn't very pretty. The site features a plain white background, a vertical taco-based navigation bar, a bit of paragraph text on each page, and bulleted lists with links to Taco.com's services.

Taco.com illustrates some of the early challenges designers faced on the web. The intended purpose of HTML was to describe the basic structure of a web page to a web browser; the purpose was never to make websites look beautiful.

For early web designers, creating aesthetically pleasing websites wasn't just a design challenge—it was error-prone and tedious. On a twenty page website, if you wanted to specify a font style or background color, you had to repeat the process over and over again, for every element on every single page. If you ever had a change of heart and reconsidered the use of hot pink text throughout your site, you'd have to go through the site and change these styles paragraph by paragraph, page by page. As you can imagine, this led to lots of mistakes, bloated code, and many computers thrown from second story windows in frustration.

important!

Your goal in this course is to create clean, standards-compliant code.

note

The World Wide Web Consortium (the W3C) sets the standards for coding on the web.

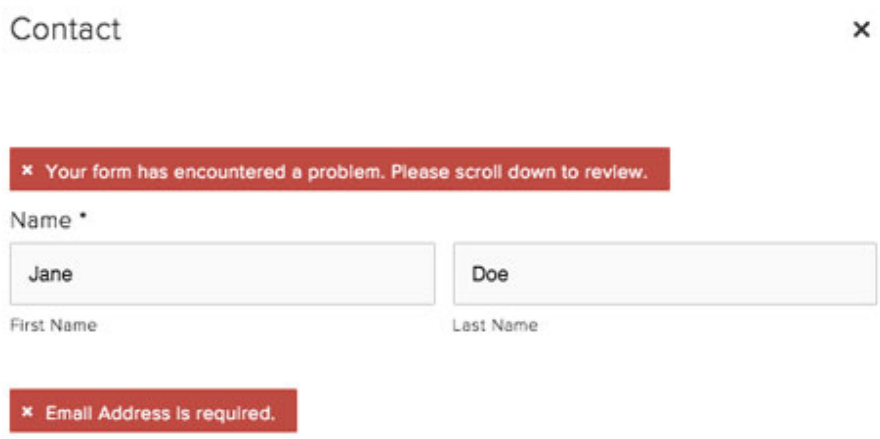
Be gone with ye, infuriating technology!

Fortunately, in the early 2000s, developers came up with new languages that would work together to vastly improve the design and the interactivity of web pages. CSS, which stands for Cascading Style Sheets, came along to simplify and extend layout and design, and JavaScript (and later, JQuery) arrived to give us increasingly simple ways to make pages respond to web users. HTML itself evolved into a more sophisicated language called HTML5.

In a nutshell, HTML and CSS are known as markup languages, since they do not take direct action on their own but require JavaScript or PHP to make decisions. So I can use HTML and CSS to create a neat-looking contact form, but I'll need JavaScript to tell me when I've skipped a required field!

note

All HTML and CSS documents for this course should be created in a text editor like Sublime.



This form is structured using HTML, styled with CSS, and programmed in JavaScript.

Course Objectives

Writing Clean, Standards-Compliant Code

important!

Avoid using word processors, templates, or HTML editors like Dreamweaver for this course.

In this course, we're going to learn how to code websites using HTML and CSS. We're not just going to learn code, though—we're going to learn clean, standards-compliant code.

Clean code is code that's written with proper spacing and indentations, making it easier for you (and future web designers) to read and troubleshoot. **Standards-compliant** code is code that meets the web standards and "best practices" put forth by the World Wide Web Consortium (the W3C).

W3C standards are constantly evolving, and include things like:

- defining the standards for building and rendering web pages
- keeping structure (HTML) separate from presentation (CSS)
- making web pages more accessible for people with disabilities
- helping web pages function the same across all web browsers and devices

As a web designer, learning to write standards-compliant code and familiarizing yourself with the best practices in coding will help you create all-around better web sites. Standards-compliant sites are more usable and accessible, they're easier to maintain and update, they're search-engine friendly, and they follow the basic principles of good design. The first step in creating Web sites that look good on the surface is writing clean code that looks good "behind the scenes."

In this course, we'll begin by learning how tostructure a page using HTML and attach CSS style sheets to design color and typography. We'll look at different ways that HTML5 code is used to provide semantic meaning and structure to content and examine how CSS can be used to customize both page layout and design.

Finally, so you can begin thinking about designing for mobile devices, we'll wrap up with a look at responsive design, in which we will learn how to use a responsive framework to create a fully responsive site.

note

Professional web designers know how to hand code as well as use HTML editing programs.

Using a Text Editor

For this course, we'll create all of our HTML and CSS documents using a text editor. A **text editor** is a software program that allows you to write text without any formatting (like Notepad for Windows or TextEdit for Mac). One of my favorite text editors, and one that I highly recommend for this course, is a cross-platform editor called Sublime.

All of my coding examples are created in Sublime, so if you use it, it will easier to follow along. Sublime also has several features including indenting and color-coding that make code easier to read.

tip!

Double click any HTML file on your computer to open it in a web browser.

Sublime's color-coded interface

Sublime may be downloaded and evaluated for free, however, the hardworking developers would like you to purchase a license if you plan on using the program in perpetuity. There is currently no enforced time limit for exploring and evaluating Sublime, though you can expect a friendly popup reminder every once in awhile.

To download Sublime, visit www.sublimetext.com now and click the large Download button on the home page. Unzip the program after downloading, then follow instructions for your operating system to complete the download. Please do this now before you continue!

Notes on Using Other Text Editors

After you try out Sublime Text, if you would really prefer to use your computer's built in text editor (such as Notepad for Windows or TextEdit for Mac) for editing pages, that's OK. Just bear in mind you might run into some trouble with the editor stripping out your HTML.

In Apple's TextEdit, you can work around that. Go to TextEdit's preferences, and in the New Document section, check "Plain text" under the Format subheading. Then, in the Open and Save section of preferences, select "Ignore rich text commands in HTML files." Hopefully that should get you coding!

Definitely make sure you're not using a word processing program like Microsoft Word or Apple Pages in this course. These programs will add formatting to your text that will mess up your code.

note

The .html extension tells a browser that a file is an HTML document.

Equally important, you may not use Dreamweaver or pre-built HTML/CSS design templates for this course. As you develop proficiency as a designer, you can use more visually-oriented design software in conjunction with hand-coding. For now, we'll focus on learning valid HTML5 and CSS through hand-coding alone.

In this first lecture, we'll start by going over the basics of HTML. We'll cover how to construct an HTML document, how to use HTML tags to mark up your document, and how to launch your pages on the web.

Your web pages won't look great at this stage, but you'll be building a firm foundation for more visual designs in the lectures that follow.

Start off by downloading and expanding lesson1.zip found in the downloads section of this course. This folder contains a file named alice.txt and an image folder with a file inside named alice.gif that you will use throughout this lecture. It's important that you keep these files in one place so that all of the files work together properly. Let's go!

note

The angle brackets in HTML code are called tags.

Constructing an HTML document

When you first open Sublime, you'll be greeted by a pristine editing window. Nothing but a plain black background and a humble little number 1 in the top left corner of the editor, indicating where you'll begin typing your first line of code. Take a moment to appreciate the beauty of this empty document—it won't stay that way for long!

Before we start coding, let's take a sneak peek at the finished version of the HTML document we'll be marking up:

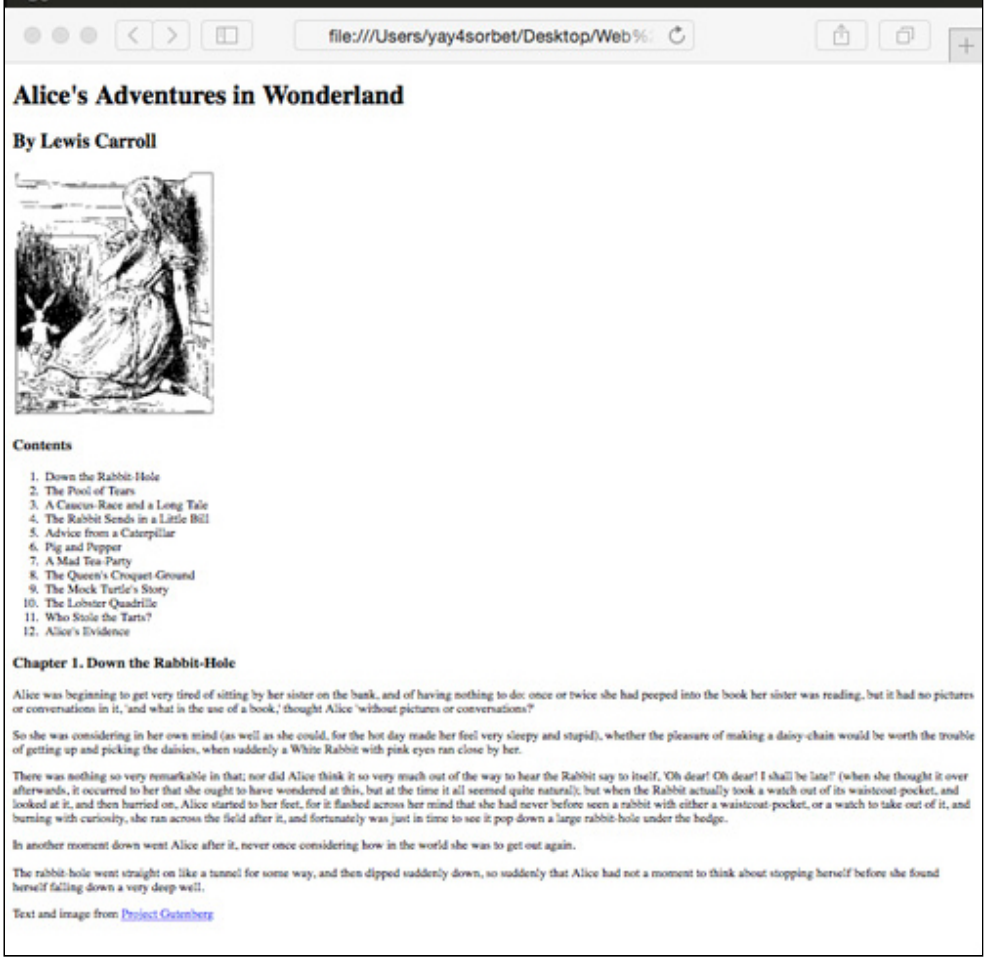
note

Most HTML elements have opening and closing tags, like and .

note

Void elements in HTML such as <img> and
 do not need a closing tag.

```
<h1>Alice's Adventures In Wonderland</h1>

<h2>By Lewis Carroll</h2>

<p>Image of large Alice and small rabbit in hallway</p>

<h2>Contents</h2>

<ul>
    <li>Down the Rabbit-Hole</li>
    <li>The Pool of Tears</li>
    <li>A Caucus-Race and a Long Tale</li>
    <li>The Rabbit Sends in a Little Bill</li>
    <li>Advice from a Caterpillar</li>
    <li>Pig and Pepper</li>
    <li>A Mad Tea-Party</li>
    <li>The Queen's Croquet-Ground</li>
    <li>The Mock Turtle's Story</li>
    <li>The Lobster Quadrille</li>
    <li>Who Stole the Tarts?</li>
    <li>Alice's Evidence</li>
</ul>

<h3>Chapter 1. Down the Rabbit-Hole</h3>

<p>Alice was beginning to get very tired of sitting by her sister on the
bank, and of having nothing to do: once or twice she had peeped into the
book her sister was reading, but it had no pictures or conversations in
it, 'and what is the use of a book,' thought Alice 'without pictures or
conversations?'</p>
```



The HTML code above creates the web page below when viewed in a web browser.

note

The <head> and <body> are the two main parts of an HTML document.

You might recognize the text and image above from the classic children's book *Alice in Wonderland*. *Alice in Wonderland* is in the public domain, which means the work is available to the public and can be reused without copyright restrictions. The *Alice and Wonderland* text and image we'll be using in this

note

All content that will appears in the browser window should be in the <body> section.

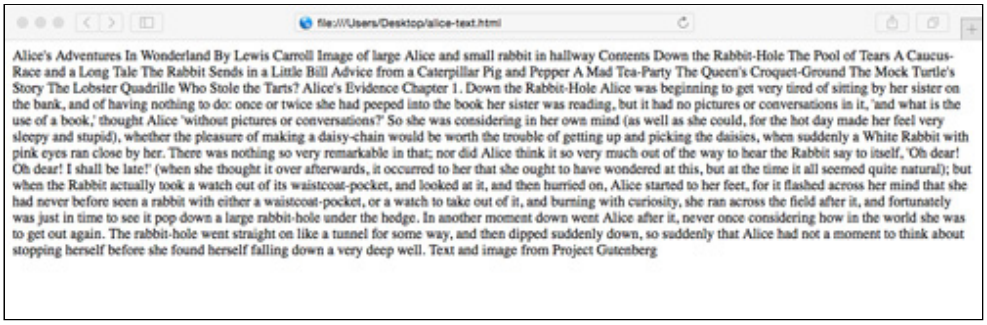
lesson come from Project Gutenberg (gutenberg.org) a wonderful resource for finding free eBooks on the web.

The HTML code in the Alice page might look overwhelming, but fear not! We'll go through the document piece by piece, constructing the markup as we go. We'll start by declaring the HTML document type, then we'll add some basic HTML tags until the finished product looks just like the page pictured above.

Open the Alice folder inside the lesson1 folder you just downloaded. In that folder, open the alice.txt file in Sublime, or your text editor of choice. Go to **File > Save As...** and save the file as "alice.html".

When I view this file in a text editor, it doesn't look too bad. There are clear spaces between the lines of text and I can tell where one line ends and another begins. But without any markup, what will this HTML file look like on the web?

To preview an HTML document live in your web browser, simply double-click the file icon, or go to **File > Open** in your browser of choice. Here, I'm previewing the document in Safari.



The page without any HTML or CSS

Yuck! Without a proper doctype declared and without any HTML tags, the page looks like one long, unreadable block of text. Where are the page headings? The spaces between paragraphs? The bulleted lists and links? Your web browser can't interpret that information without your help, so let's get coding!

Writing HTML Tags

All HTML documents are made up of elements, and elements are made up of tags. HTML **elements** are short abbreviations—like p for paragraph—that make up the structural parts of a Web page. All elements are surrounded by two angle brackets called **tags**, commonly recognized as the "less-than" and "greater-than" symbols. Here's an example:

```
<p>I'm a bit of paragraph text!</p>
```

The majority of HTML tags start with an opening tag and end with a closing tag, as in the example above. Opening tags and closing tags look similar, but closing tags are written with a forward slash (/) before the tag name. There are generally no spaces between an element and a bracket.

note

Adding structural tags such as <h1> and <p> provides basic line breaks, spaces, and font sizes.

note

Lists can be ordered <ol> or unordered <ul>.

Here's a list of common HTML tags, many of which we'll be covering in this lesson:

Element	Tags
HTML document	`<html></html>`
Page head	`<head></head>`
Page title	`<title></title>`
Page body	`<body></body>`
Header	`<h1></h1>` and `<h2></h2>`
Paragraph	`<p></p>`
Ordered list	`<ol></ol>`
Unordered list	`<ul></ul>`
List item	`<li></li>`
Emphasized text	`<em></em>`
Important text	`<strong></strong>`
Anchor	`<a></a>`
Image	`<img>`

While most tags follow the open-and-close format, there are a handful of elements in HTML that don't require an end tag. These are called void elements. Examples of **void elements** include the `<br>` tag used for line breaks, the `<hr>` tag used for horizontal rules, and the `<img>` tag used for—you guessed it—images.

Creating an HTML Page

Declaring the HTML5 document type

Now that we're veritable experts in HTML tags, it's time to mark up some text! Direct your attention to the **alice.html** document in your text editor. All HTML documents begin with a document type (or **DOCTYPE** for short). The HTML5 document type looks like this:

```
<!DOCTYPE html>
```

Pretty simple, huh? It used to look a lot more complicated.

Place your cursor at the beginning of the first line of text, which reads, "Alice's Adventures in Wonderland." Press return once, and notice that "Alice's Adventures In Wonderland" is now on line two. With your cursor back on line one, type `<!DOCTYPE html>` at the very top of the page.

```
1  <!DOCTYPE html>
2  Alice's Adventures in Wonderland
3
4  By Lewis Carroll
5
```

This communicates to the browser that this document should be interpreted as HTML5, and not an older version of HTML.

The HTML Head and Title

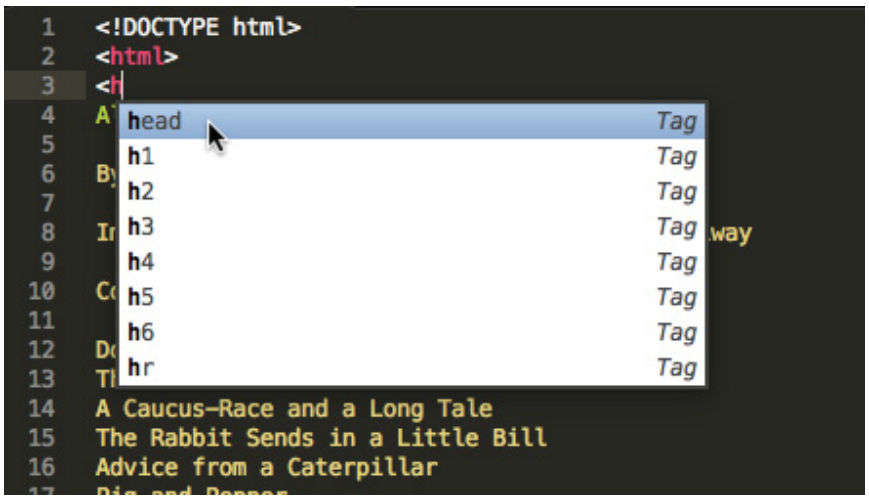
Next, wrap the text of your document in an opening and closing html tag. Place the opening tag below the DOCTYPE declaration and the closing tag at the end of the document, after the last line of text.

```
<html></html>
```

Now you'll add tags to divide your document into two parts: the head and the body. The **head** element contains some important "behind the scenes" info, like the document title, keywords that improve search engine results, CSS styles, and the script tag (for JavaScript and other client-side code). Place opening and closing `<head>` tags at the top, below the opening `<html>` tag and before the Alice in Wonderland text begins.

```
<head></head>
```

Notice anything cool as you begin typing the opening `<head>` tag? Because we've told Sublime that this is an HTML document, Sublime will now start suggesting code. After typing an open bracket and the letter "h," Sublime guesses (correctly) that I'm trying to add a head section to my HTML document.



Code suggestions help speed up the coding process by predicting what you will type.

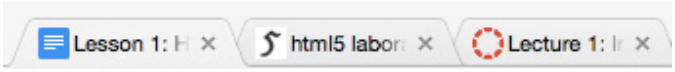
When Sublime is in suggesting mode, I need only press **Return/Enter** once for Sublime to add an opening and closing head tag to my document. Pretty neat, huh?

Inside the head tag it is important to add a `<meta>` tag with the charset attribute to specify the character encoding for the HTML document.

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

Without this piece of code, some characters in your text could take on an odd appearance when displayed in the browser. Also, you will run into issues with validating your code.

There's lots of optional information that can go in the head of an HTML document, but every HTML doc has to have a **title**. The document title gets displayed in the page tab at the top of a web browser and helps orient website visitors like me who keep too many tabs open at a time.



Browser tabs with titles displayed for easy browsing

Unlike website keyword information or JavaScript code, which stay hidden behind the scenes, the title is the only part of the document head that the site visitor can see. Enter your title tag like so:

```
<title>Alice in Wonderland</title>
```

You can use the handy code suggesting trick we learned above, but remember to type the title text between the two `<title>` tags. Pressing **Return/Enter** in suggesting mode will automatically close tags, so you'll have to go back and add text content between the opening and closing tags.

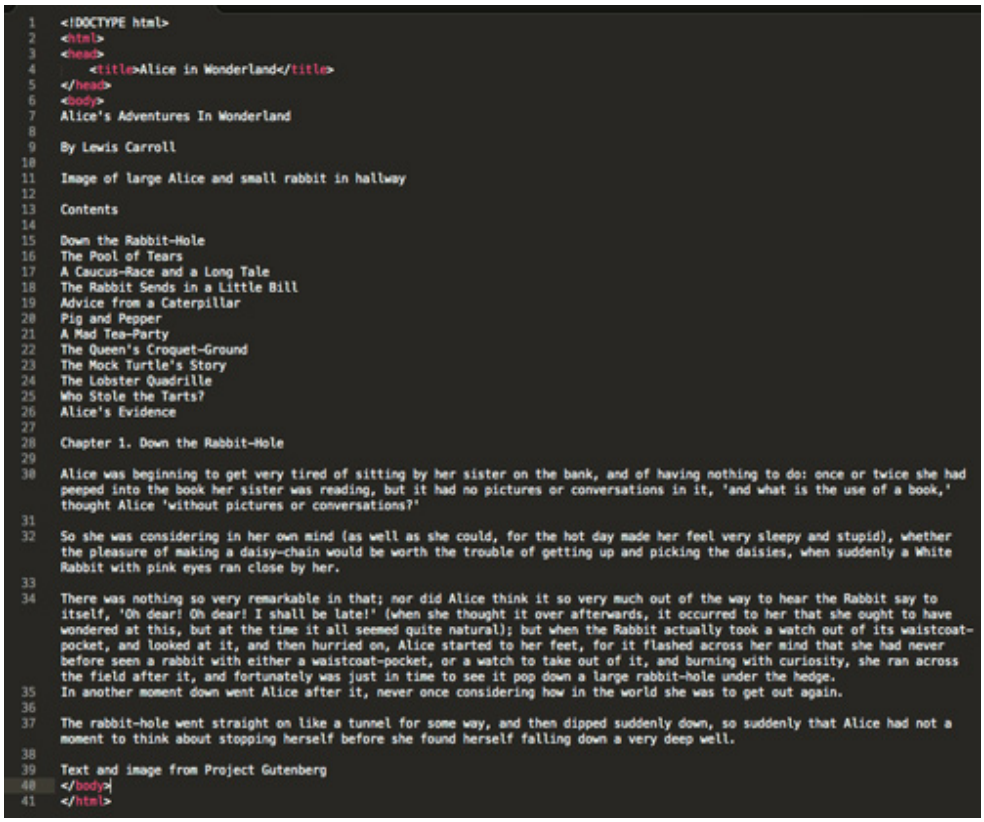
As you add new tags to your HTML document, Sublime may start automatically indenting your code. As our HTML documents get more complex, clear spacing and indentations will be a big help down the road. Let Sublime do its thing, and feel free to add your own indentations along the way!

The Body Section

With the document head complete, we're ready to start marking up the rest of the page content, and we'll start by adding `<body>` tags. The **body** of the page contains the meat of the document: the content that actually gets displayed in the window of your web browser. Wrap the page content in `<body>` tags, starting after the head section and ending right before the closing `<html>` tag :

```
<body></body>
```

Your document should now look like this:



```
<!DOCTYPE html>
<html>
<head>
    <title>Alice in Wonderland</title>
</head>
<body>
Alice's Adventures In Wonderland

By Lewis Carroll

Image of large Alice and small rabbit in hallway

Contents

Down the Rabbit-Hole
The Pool of Tears
A Caucus-Race and a Long Tale
The Rabbit Sends in a Little Bill
Advice from a Caterpillar
Pig and Pepper
A Mad Tea-Party
The Queen's Croquet-Ground
The Mock Turtle's Story
The Lobster Quadrille
Who Stole the Tarts?
Alice's Evidence

Chapter 1. Down the Rabbit-Hole

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had
peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,'
thought Alice 'without pictures or conversations?'

So she was considering in her own mind (as well as she could, for the hot day made her feel very sleepy and stupid), whether
the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White
Rabbit with pink eyes ran close by her.

There was nothing so very remarkable in that; nor did Alice think it so very much out of the way to hear the Rabbit say to
itself, 'Oh dear! Oh dear! I shall be late!' (when she thought it over afterwards, it occurred to her that she ought to have
wondered at this, but at the time it all seemed quite natural); but when the Rabbit actually took a watch out of its waistcoat-
pocket, and looked at it, and then hurried on, Alice started to her feet, for it flashed across her mind that she had never
before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and burning with curiosity, she ran across
the field after it, and fortunately was just in time to see it pop down a large rabbit-hole under the hedge.
In another moment down went Alice after it, never once considering how in the world she was to get out again.

The rabbit-hole went straight on like a tunnel for some way, and then dipped suddenly down, so suddenly that Alice had not a
moment to think about stopping herself before she found herself falling down a very deep well.

Text and image from Project Gutenberg
</body>
</html>
```

Notice that we placed our `<title>` tags within the `<head>` tags of the HTML document, and the title, head, and body tags all got placed inside the `<html>` tags? This is called tag nesting! **Tag nesting** means that certain HTML tags must be enclosed within each other. We'll learn more about tag nesting along the way, but here are the rules we just covered:

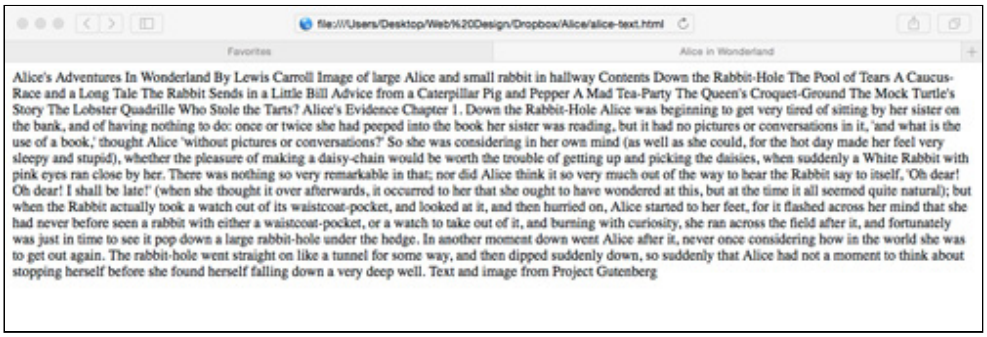
1. Head and body tags are always nested inside the html tags.
2. Title tags are always nested inside the head tags.
3. The page content that gets displayed in the web browser is always nested inside the body tags.

Following these rules, the skeleton of any HTML document should look something like this:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>My page</title>
</head>
<body>
This is my web page, and here's the web page content.
</body>
</html>
```

Save and View the Page

We've given our HTML document a basic skeleton, but our work isn't done yet! After saving your document in Sublime (it's good practice to save often), navigate to the file on your computer and either double-click the icon to view it in your web browser, or go to **File > Open** in your browser of choice:



The text will still display as one large paragraph without the proper formatting.

Well, that's disappointing! Fortunately, because of the work we've already done, the browser knows that this is an official HTML page. Unfortunately, none of the text is formatted on the page! Without any headings, lists, or paragraph tags, the web browser still reads this as one long chunk of text.

As you can see, the browser ignores any extra line breaks or spaces included in the original HTML document. If we want line breaks, extra spaces, or paragraphs, we have to specify those elements by adding markup tags. Let's make our web page a bit more presentable by providing some structure.

Structuring the Page Content

Header Tags <h1> <h2> <h3>...

We'll start by adding heading tags to the page. Heading tags range from <h1>, the most important page heading, down to <h6>, the least important page heading. Because it's the most important heading on the page, we'll wrap the first line of text, "Alice's Adventures in Wonderland," in <h1> tags.

Wrap the words "By Lewis Carroll" in <h2> tags, and the words "Contents" and "Chapter 1. Down the Rabbit-Hole" in <h3> tags. Remember to close your tags as you go!

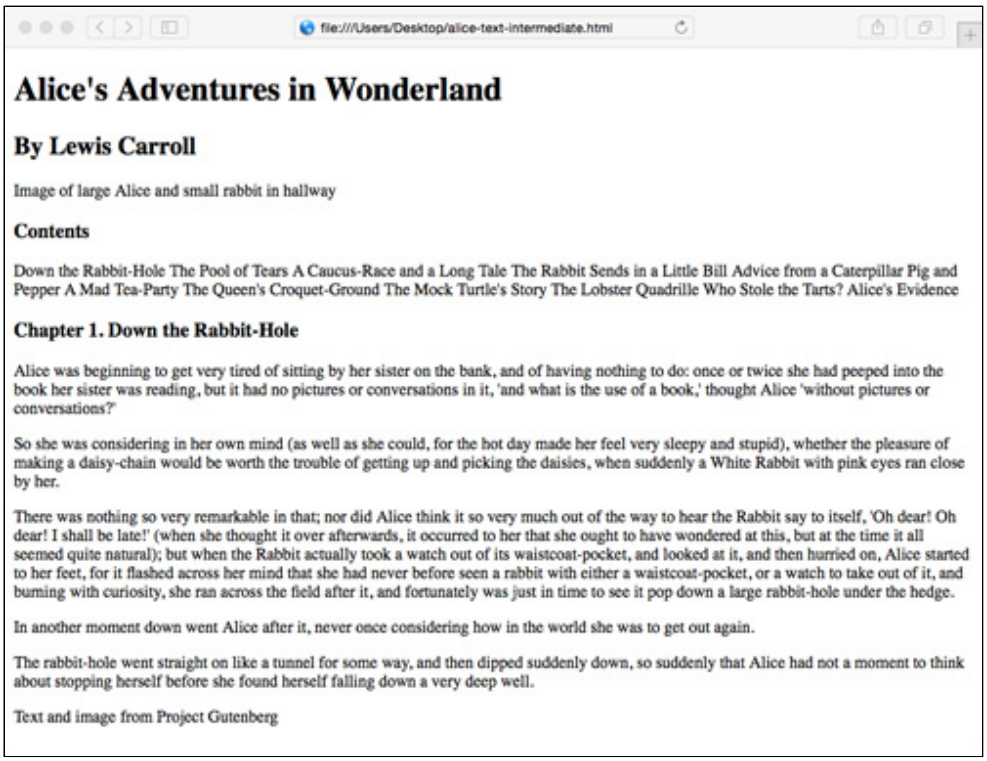
Behind the scenes, headings help search engines determine the structure of your HTML document. To a site visitor, headings help break up page content, making your text easier to read. It's important to use heading tags in the right order, beginning with <h1> tags for the main heading, followed by <h2> tags, then <h3> tags, and so on. You don't need to use every type of heading on a page—you just have to use them in the right order.

Note: Headings should be used to indicate the importance of text, but not to style the size or weight of text. At this point, don't worry too much about what the headings look like in your browser. Later, we'll be using CSS to customize our text styles.

Paragraph Tags <p>

Next, we'll add paragraph tags to space out the paragraph text on our page. Add <p> tags to the sections of text that appear after the <h3> heading that reads, "Chapter 1. Down the Rabbit-Hole."

After wrapping text in paragraph tags, the Web browser will automatically add an empty line before and after each paragraph. This is a great time to save and view the page in your web browser.



Now that proper formatting has been added, the text can be easily read.

Check out those headings and paragraph spaces. Now we're getting somewhere!

List Items <ul>

Below the heading that reads "Contents," we'll want to add a proper bulleted list to the Alice in Wonderland table of contents. There are two types of HTML lists we'll cover in this course: unordered lists and ordered lists.

Unordered lists are used when the order of list items isn't very important, like a grocery list. Unordered list items appear with bullet symbols in a web browser:

```
<ol>
        <li>Chocolate</li>
        <li>Cake</li>
        <li>Cupcakes</li>
        <li>Pie</li>
</ol>
```

Unordered lists begin and end with `<ul>` tags, with individual `<li>` tags nested inside the opening and close `<ul>` tags.

Ordered lists are used for numbered lists, like chapters in a book. Ordered list items appear as ordered numbers in a Web browser:

```
<ol>
        <li>The Beginning</li>
        <li>The Middle</li>
        <li>The End</li>
```

```
        <li>The Epilogue</li>
</ol>
```

Ordered lists begin and end with `<ol>` tags, with individual `<li>` tags nested inside the opening and closing `<ol>` tags.

Because the chapter order matters in our Contents section, we'll want to wrap the table of contents in an ordered list. Wrap the entire list in `<ol>` tags, and wrap the individual list items in opening and closing `<li>` tags, as shown in the example above. Here's a side-by-side image of what the code looks like in Sublime, and what it looks like in my Web browser:

In web browsers, properly coded <ol> tags appear as a numbered list.

Note: Notice that there are no `<p>` tags around my ordered list. Ordered lists, unordered lists, and heading tags should never be nested inside paragraph tags. Doing so will lead to validation errors.

Adding Hyperlinks

note

Hyperlinks are created using the anchor tag <a>.

Without hyperlinks, it would be pretty hard to get around on the web. HTML links allow a user to navigate from page to page on a website, or to jump to an external site on the web. Hyperlinks can be added to text or to an image, and are defined using the **anchor tag** `<a>`.

Anchor tags look a bit different from the standard heading, paragraph, and list tags you've learned about in this lesson. Hyperlinks are comprised of an `<a>` tag followed by an **attribute**. An attribute is a short abbreviation placed inside an opening tag that provides additional information about an HTML element. Attributes always follow the same format—the name of the attribute and an equal sign, followed by the value of the attribute in quotations:

Adding External Links

There are a few different types of links we'll be covering in this course, but we'll cover external links first. **External links** are hyperlinks that take you from your current location to a different site on the web. For the hyperlink to work, the code must contain a fully qualified URL (starting with http:// and including a domain extension such as .com or .org, like these links):

note

An absolute link contains a complete URL specifying a location on the web.

http://www.gutenberg.org
http://www.gutenberg.org/books

Near the bottom of the page in your Alice document, navigate to the line of text that reads "Text and image from Project Gutenberg." Wrap the words "Project Gutenberg" in the link shown above, like so:

```
<a href="http://www.gutenberg.org/">Project Gutenberg</a>
```

When you preview the document in your browser, notice that the text between the `<a>` tags turns into a clickable blue link. Later in the course, we'll use CSS to add color and pizazz to text and jazz up the default link appearance.

Inserting Images

The look of our document has improved, but we can't publish an Alice in Wonderland web page without including one of Lewis Carroll's amazing original illustrations! In your Alice document, navigate to the line of text between the author heading and contents heading that reads "Image of Alice and rabbit in hallway". Delete this placeholder text so we can place our image on that line.

The HTML `<img>` tag is another example of a void element, meaning it contains no closing tag. The `<img>` tag is always paired with the src attribute and alt attribute, like so:

```
<img src="images/alice.gif" alt="Alice and rabbit in hallway">
```

The **src** attribute tells us where the image is located. You can pull an image address from the Internet, or target one locally, as we did here. In the example above, the alice.gif image file is in a local folder called "images". (When I use the term local folder, I'm referring to a folder that's located on your computer, and not online.)

The **alt** attribute tells the browser what text to display if a user can't view the image. This is particularly useful for:

note

A relative link specifies the location of a file on your site relative to your file or the root folder.

- Helping visually impaired users navigate the web
- Text-only browsers or email clients that disable images for security reasons
- Errors in the src attribute that lead to broken image links
- Images that time out on a slow Internet connection

Compare two broken image links of a frog below. Which link is more helpful to the web user?

important!

Make sure you use an ad-free web host for this course.

A broken image with no alt text (left) and the same image with alt text (right).

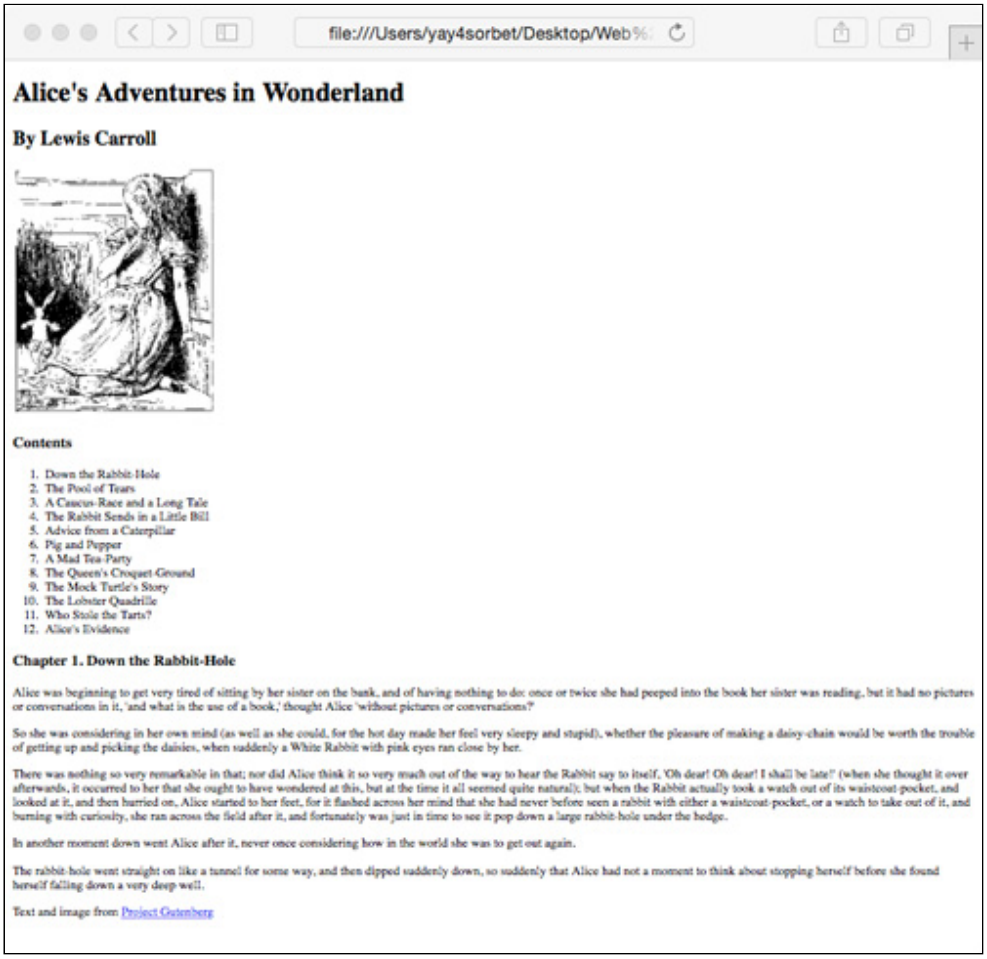
Even though the user can't see the original image, they'll be given some context that improves their experience on the web.

Inserting Image Links

The final thing we'll do in our Alice document is link the alice.gif image to the Project Gutenberg website. Like text, images can be transformed into clickable links by wrapping them in anchor tags:

```
<a href="https://www.gutenberg.org/">
<img src="images/alice.gif" alt="Alice and rabbit in hallway">
</a>
```

Preview the document in your browser and click to test the link. Voila! The image should take you directly to the Project Gutenberg home page.



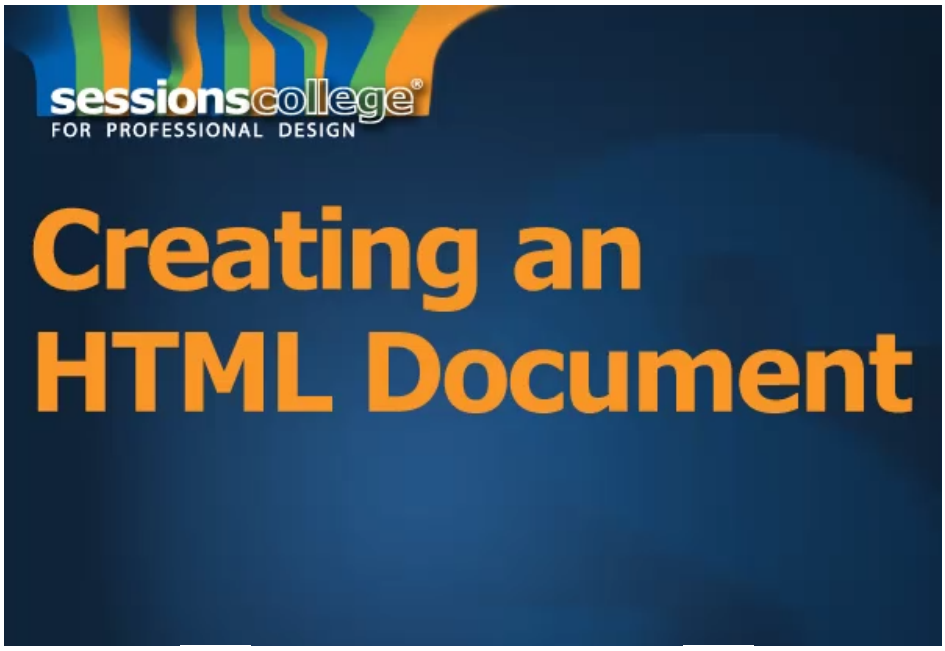
The final web page viewed in a browser

Congratulations! You've now created your first HTML web page. It may not look very fashionable, but it's clear, readable, and semantically sound.

Before moving on to the final sections, let's review what you've learned about creating an HTML page, structuring the page content with basic tags, and adding links and images. This video tutorial will covers the essentials:

(**Note:** In some of the videos you will see the DOCTYPE declaration for HTML 4.0. Be aware that the current HTML5 DOCTYPE declaration is `<!DOCTYPE html>`.)

VIDEO TUTORIAL: Creating an HTML Document



0:00 / 5:11

Posting to the Web

To become a proficient web designer, you need to become comfortable with posting your web designs to the web. For that reason, all assignments in this course will need to be posted to the web and tested for validation issues, broken links, and other errors before you submit them or post them for grading.

This means:

- **You will need a domain name and web hosting.** You will need have an account with a company that provides a Domain and designated web space for your files. If you don't already have a web hosting account, free accounts exist, and we'll walk you through how use one below.
- **You will need to use an FTP program.** FTP (File Transfer Protocol) is the common standard for uploading files to the web. If you don't already have an FTP program, you can use a free program. Details on how to use an FTP program are below.

Reserving an Ad-Free Web Hosting Space

Before you can post your site to the web you need a domain and a web hosting account. There are a range of inexpensive, quality web hosts with budget pricing for entry level space. Alternatively, there are numerous sites that provide free hosting with a domain.

It's essential that you get a host with ad-free hosting. Some free hosts will provide free space with popup banners or ads that make it impossible to properly view. You wouldn't want to launch a client's website with pop-up ads and banners, would you? Assignments turned in with ad banners will be marked as incomplete.

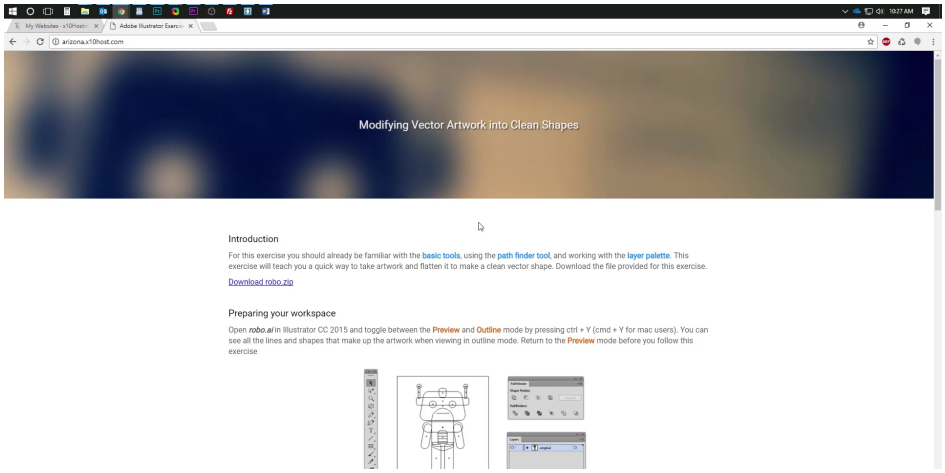
- The following companies offer ad-free hosting:
  - awardspace.com
  - agilityhoster.com
  - x10hosting.com
- The following sites cost money:
  - nearlyfreecpeech.net: is very trustworthy, though may be on the techie side of things.
  - asmallorange.com: is also good. Check their website for current prices.

Please note: We are not affiliated with nor do we endorse these web hosts. At time of writing, these hosts provide web hosting at no cost to allow you to host your assignments without forced ads.

Every web host functions a bit differently. Once you've registered, make sure you save the email providing the information you'll need to post your files.

- Domain name
- Username
- Password
- FTP address

VIDEO: Getting Started with x10hosting.com



0:00 / 2:57

If you're not sure what your domain name is, you'll have to contact your hosting provider for that information. If you need help understanding the terms involved in web hosting (like host, server, and domain), please see our guide to web hosting.

Once you have somewhere to host your files, you're nearly ready to publish to the web.

Downloading and Installing an FTP Program

The simplest way to move files from your computer onto your website (so people can see your pages on the web) is through FTP or **file transfer protocol**. You'll use an **FTP client**—a program specifically designed to make this transfer.

First, you need to decide which program you are going to use. There are many free ones available for download for Windows and Mac. I'm going to demonstrate how they work using FileZilla, which is a free program available for Mac and PC, but Cyberduck is also a good alternative.

- FileZilla (Windows or Mac): FileZilla started out as a class project in 2001. Filezilla is an free open source FTP client that can be used on both Mac and PC. The interface may seem busy which could turn some inexperienced users away but the user can make adjustments to the layout.
- Cyberduck (Windows or Mac): This software is an open source client for FTP. It is free to use but you can make a donation to become a registered user. It has a drag and drop interface and includes a bookmark manager.

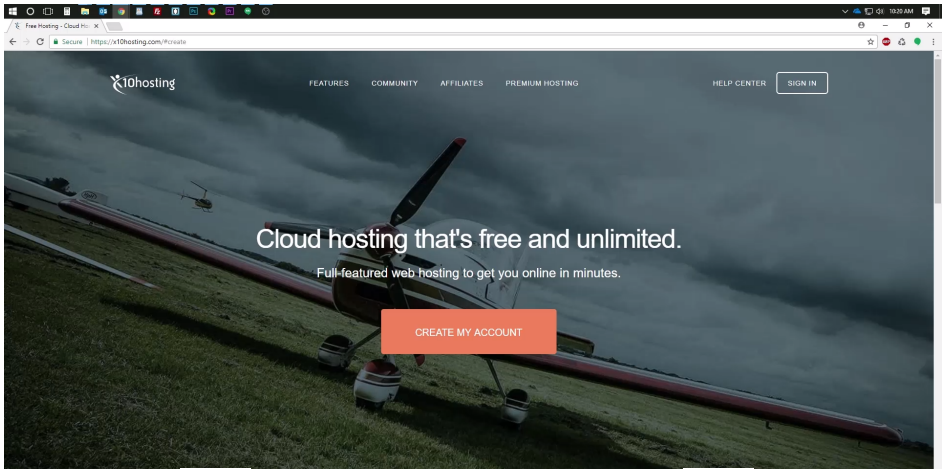
Once you've selected your program of choice, we can begin setting up the FTP client to move files from your computer to your website. Most FTP clients work in a similar fashion.

Retrieve Your FTP Hosting Information

Once you've signed up for hosting, be sure to check your email. An email from your host is usually where your server name, username, and password will be. Guard it closely!

This is the information you need to metaphorically walk into your website storage space on your web host and deposit your files. If you are still unsure of your server name, username, and password after checking your email, call or email your web host to find out this information. Otherwise, it's like having a physical storage space without knowing the address or how to get there.

VIDEO: FTP Information with x10hosting.com



0:00 / 1:31

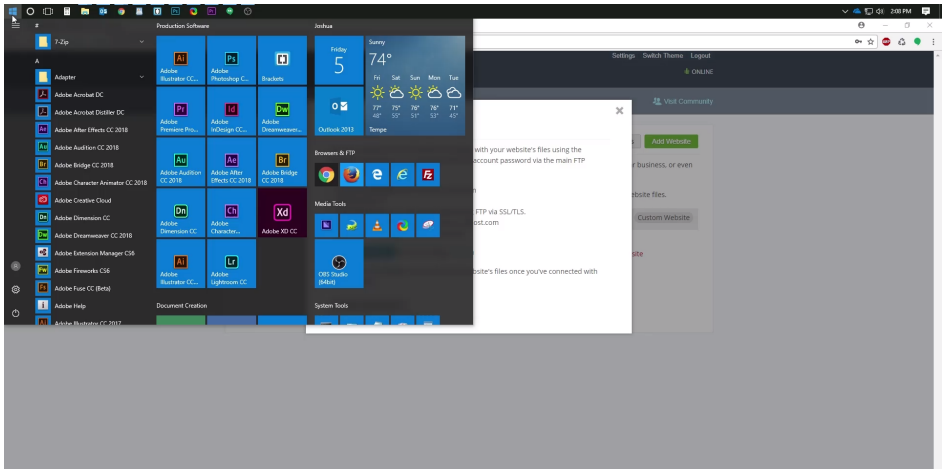
Connecting to Your Web Server

Once you have your FTP information, you'll need to connect to your web server in order to begin transferring files.

Open your FTP program. For the purposes of this example, we are using Cyberduck. While most FTP programs function similarly, the user interface may differ, so you may need to adjust these steps depending on the FTP application you're using.

First, I will need to create a new site in FileZilla. Watch the following video to see how I set up my FTP with x10hosting.com.

VIDEO: Setting Up FileZilla



0:00 / 1:55

When you signed up for web hosting, you may have received a welcome email from your web host with FTP login information. If you didn't, contact your host to get that information now, and keep the information saved in a secure place. You'll be using those login credentials throughout the course to upload your files to the web.

There are three pieces of information you'll need to establish an FTP connection:

- **Server address (or host name):** This is usually the domain name you chose or were assigned when you signed up for web hosting.
- **Username:** The username you created when you signed up for web hosting.
- **Password:** The password you created, or the password your web host assigned, when you signed up for web hosting.

In the Server field, enter or copy/paste in the server address provided by your web hosting service. This is usually your website domain name (for example, mygreatwebsite.net), though depending on your web host, it may be a string of numbers separated by dots (such as 182.20.3.30).

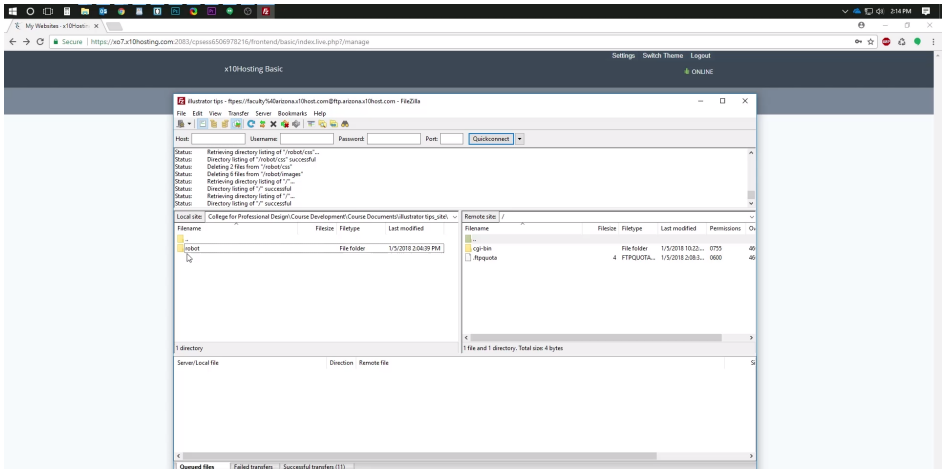
Once this is entered, you will see a Host information URL which is a URL that usually begins with "ftp", appear in the field below. Enter the Username and Password that you created for the web hosting service and click Connect. Some web hosts may require you to set a specific encryption.

If all goes well, you will connect with your web server. These three pieces of information (the host, username, and password) are usually sufficient to connect you. If you cannot connect, check everything you entered against the login information provided by your host.

Transferring Files to Your Web Server

In your FileZilla window, you may see some default files and folders provided by your web host. In general, there may be a directory called "public_html" or "www" that corresponds to the root of your site. Navigate to your site folder on your computer. Click and drag your file or files into the appropriate directory in the FileZilla window. A transfer window should open up and you can view the progress of your upload. In the video below, I will show how I uploaded my files to my site.

VIDEO: Uploading Files



0:00 / 0:52

Once your files are uploaded to the web, launch your browser (Chrome, Firefox, et al.) and check to see that they are live.

Enter the URL of your web page by typing in or copy/pasting the location of your file on the site. Here are two examples:

- http://www.mysitedemo.com/alice.html
  "My alice.html document is at the site root"

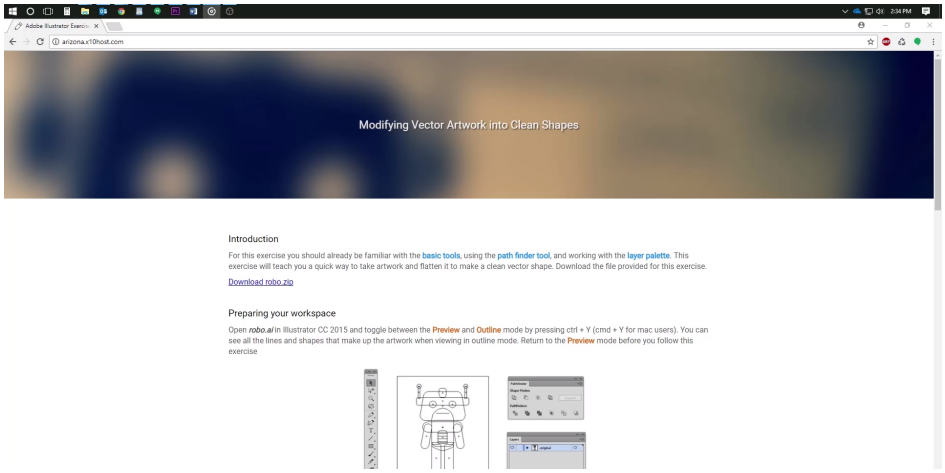
- http://www.mysitedemo.com/alice/alice.html
  "My alice.html document is inside an a folder called 'alice' at the site root"

Bear in mind, the spelling and capitalization must exactly match your files. If you cannot connect, check everything you entered against the login information provided by your host.

Once this process is complete, your web server will mirror your local website files, and if you have an index file at the root (index.html), you will be able to see your website if you navigate to your site's address on your browser. (If you don't have an index file, you'll probably just see a list of your files when you pull up the site URL. I'll explain more on index files shortly.)

Every time you want to update the files on your server, you will need follow the process you've just learned: connect using your FTP, and copy files from your local drive to your remote site.

VIDEO: Updating my Site



0:00 / 2:19

When you upload a file that already exists on the server, the file transfer window will pop up and warn you that you are about to overwrite the remote file. Check the dates provided on the files to make sure you're uploading the correct version!

To download files to your hard drive, simply copy the files from your the remote site displayed in your FTP program to the corresponding location on your hard drive.

Folders and Organization

Last but not least, let's talk about some principles for organizing files as you build websites for this course.

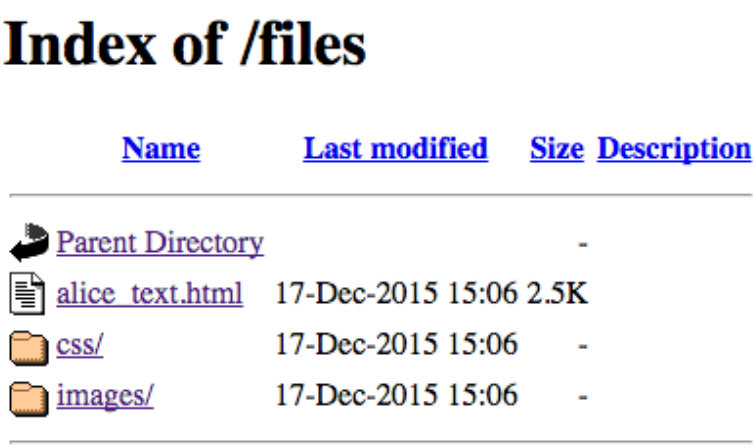
Organizing the files of a website is much like organizing any other files. Even a small website of only a handful of pages can require dozens of image files, CSS files, and script files. If you throw all of these files into one big folder, you'll end up with a mess of a site that's difficult to update and maintain.

If you structure and organize your site from the get-go, you'll save yourself some headaches along the way. Here are some tips:

1. Make sure there is an index file in every navigable folder

The file name index.html is the universally recognized name for a site's home page. In fact, all website home pages are named index.html. A web browser cannot display a website without an index.html page. This is the only page of your website where the naming is not optional. All other pages can and should be given unique names that identify them.

When a browser navigates to some directory on the Internet, it immediately tries to display the index file (named "index.html" or "index.htm"). If it can't find an index file, it will just display a list of all the files in that folder. At best, this is a bit ugly and, at worst, is a security risk! So, make sure that you put an index file in every folder that a user can navigate to on your website.



2. Use all lowercase names when naming your files

Some Unix systems have trouble with uppercase filenames (and many web servers are Unix-based), so make sure to use all lowercase letters when naming your documents. It's a good habit to write HTML tags in lowercase letters as well!

3. Reserve separate folders for image, script, and CSS files

It's a good idea to keep all of your images in a single folder (named "images", for example), and so the same for any script and CSS files. If you have only a few image files, you can keep them all in one folder. As your website gets bigger, and you are working with more and more images, you'll want to make sub-folders within your image folder, splitting them up by category.

4. Categorize your web pages into folders

If you have a small website (ten pages or less), you may be able to get away with putting all of your HTML files in the main (root) folder. But if your website is larger than that, you'll want to split up your files into separate folders. Organize them in a way that makes sense to you by grouping similar pages together in their own folders. For example, all of your product pages can go into one folder, your artwork pages can go into another, and so on.

In the next lesson

- Learn the benefits of using CSS to control the design of a website.
- Learn the three primary ways of applying CSS to an HTML page.
- Learn how to write CSS rules.
- Learn how to use selectors: classes, IDs, and spans.
- Learn how to use CSS to control color and typography.

Coming Up Next:

Discussion
Share your thoughts and opinions on HTML coding in the Discussion area.

Exercise
Use HTML code to structure and mark up a coffee shop site.

HTML and CSS Essentials | The Bare Beans

sessionscollege
FOR PROFESSIONAL DESIGN

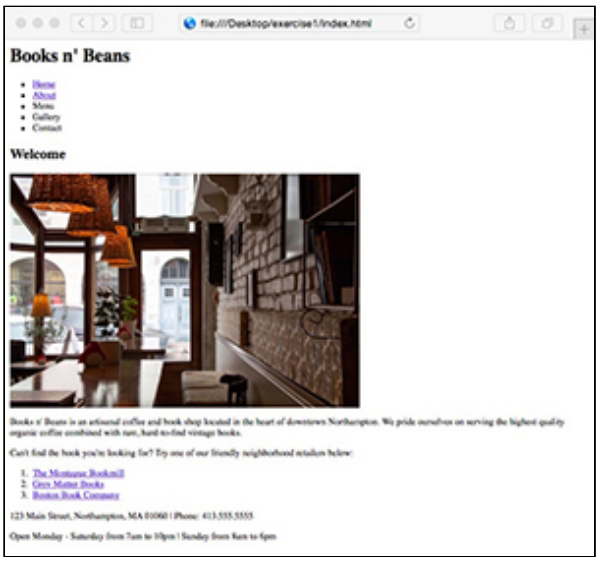
Exercise 1

The Bare Beans

In Lecture One, we learned how to create a basic HTML document and structure its content.

By doing all that, you've certainly earned a coffee break.

You'll put your knowledge into practice by building a some pages for a coffee shop Web site, structuring the content by adding heading tags like <h1> and <h2>, paragraph tags <p>, and so on. (Later in the course you'll use CSS to layout and design the site).



In this exercise, we'll begin to create this coffee shop Web site in HTML and post it to the Web.

In part two of the exercise, I'll ask you to do a bit of sleuthing to find a page on the Web with HTML markup similar to this site!

Performance Objectives

- Use HTML to create the first two pages of a coffee shop site.
- Use HTML tags to structure the content and add links and images.
- FTP your files to the Web, and test and validate your site.
- Analyze the use of basic HTML on a similar Web site.

note

Assignments are evaluated for creativity, technical proficiency, and understanding of concepts covered in the lecture.

Part 1: Books n' Beans

The quirky downtown coffee and book shop Books n' Beans is desperately in need of a Web site. After years of insisting that they didn't need a Web presence, they've caved to customer demands, and they're turning to us to help with the transition to the Web.

The staff at Books n' Beans has provided a plain text document with text copy and images for their Home and About page. Before we can start incorporating stylistic elements in the design, it's our job to provide them with a clean semantic document that passes HTML5 validation.

Download and expand exercise1.zip from the downloads section of this course to get the text and other files for this exercise. The following directions review how to put together your HTML documents.

Mark Up the Home Page

1. Create an HTML Document

```
<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
```

```
<title>My Web Page</title>
</head>
<body>
My Web Page
Lorem ipsum dolor sit amet, consectetuer adipiscing elit...
</body>
</html>
```

Take some time to get familiar with this document structure. This is the basic framework for all HTML documents, and you'll see it many times in this course.

There's currently a placeholder title between the <title> tags, so you'll need to customize the title for this page. Replace the placeholder title with the title "Books n' Beans," but without any quotation marks surrounding the text.

Next, replace any placeholder text in the document body with the Books n' Beans Home page copy. All of the text copy should be placed inside the opening and closing <body> tags.

In Lecture One, you were provided with a plain text document that had already been saved with a .html extension. For this exercise, we're starting from scratch, so we'll have to save our text document with the proper .html extension before moving on to the next section.

Go to **File > Save As** in Sublime and save the file with the .html extension, with the file name "index.html" (again, no quotation marks). Remember to use lowercase letters in file names with no spaces or special characters.

(Note: If you're using TextEdit for Mac OS X, make sure to select "Make Plain Text" from the "Format" window, and save it under Unicode (UTF-8) plain text encoding. On a PC, make sure All Files and Unicode (UTF-8) are selected.)

Congratulations. You've just created a Web page! Double click the document on your computer to view it in your default browser, or navigate to **File > Open File...** in your browser of choice. You should see a big block of text that sorely needs some semantic markup. On to the next step.

2. Mark Up Headings and Paragraphs

With the basic document structure in place, we can start working on the text copy.

Start by marking up the Home page with the appropriate headings and paragraph tags. Wrap the "Books n' Beans" page title in <h1> tags and wrap the friendly "Welcome" text in <h2> tags.

Next, wrap each of the following blocks of text in paragraph tags:

"Books n' Beans is…"

"Can't find the book…"

"123 Main Street…"

"Open Monday…"

With some basic tags in place, things should be looking much better. Save and preview the page in your Web browser. And as a friendly reminder, you can never save too often!

Tip: To optimize your Web site's rankings in Google or other search engines, it's generally recommended to make your title tags and your H1 tags unique—to create a different, descriptive title and header for each page. For more

information on using these tags for SEO (search engine optimization), here's a good article at Search Engine Land.

3. Format the Lists

Now let's format the lists so they're easier to read. For the first list, our future site navigation, we'll be using an unordered list. Wrap the navigation list in `<ul>` tags. Remember, the entire list should be enclosed in opening and closing `<ul>` tags, with individual list items wrapped in `<li>` tags.

Wrap the second list, featuring three friendly neighborhood retailers, in `<ol>` tags, and enclose individual list item in `<li>` tags here as well.

To confirm that your code works (and to build good workflow habits), save and preview the page again.

4. Add Anchor Links

Next, let's insert anchor links to add some interactivity to the page.

Add anchor links to the ordered list featuring the Montague Bookmill, Grey Matter Books, and Boston Book Company. This will be a good opportunity to practice proper tag nesting, so follow this format closely:

```
<li><a href="http://www.montaguebookmill.com/">The Montague
 Bookmill</a></li>
```

The URLs are as follows:

- http://www.montaguebookmill.com/
- http://www.greymatterbookstore.com/
- http://www.rarebook.com/

Save your file and preview the page. Are the links in the right place with tags nested properly? Does each link connect to the right page in your browser?

When a site visitor clicks on an external link from your Web site, it’s important that the visitor doesn’t lose track of your site in the process! When formatting external links, you can keep a visitor from having to press the “back” button to return to your site by using the handy HTML **target="_blank"** attribute.

```
<li><a href="http://www.montaguebookmill.com/" target="_blank">
The Montague Bookmill</a></li>
```

The **target="_blank"** attribute conveniently opens a Web page in a new window, rather than taking a visitor away from your Web site. Use this attribute when adding external links to your page, but not on local navigation links within your own site!

5. Add Images

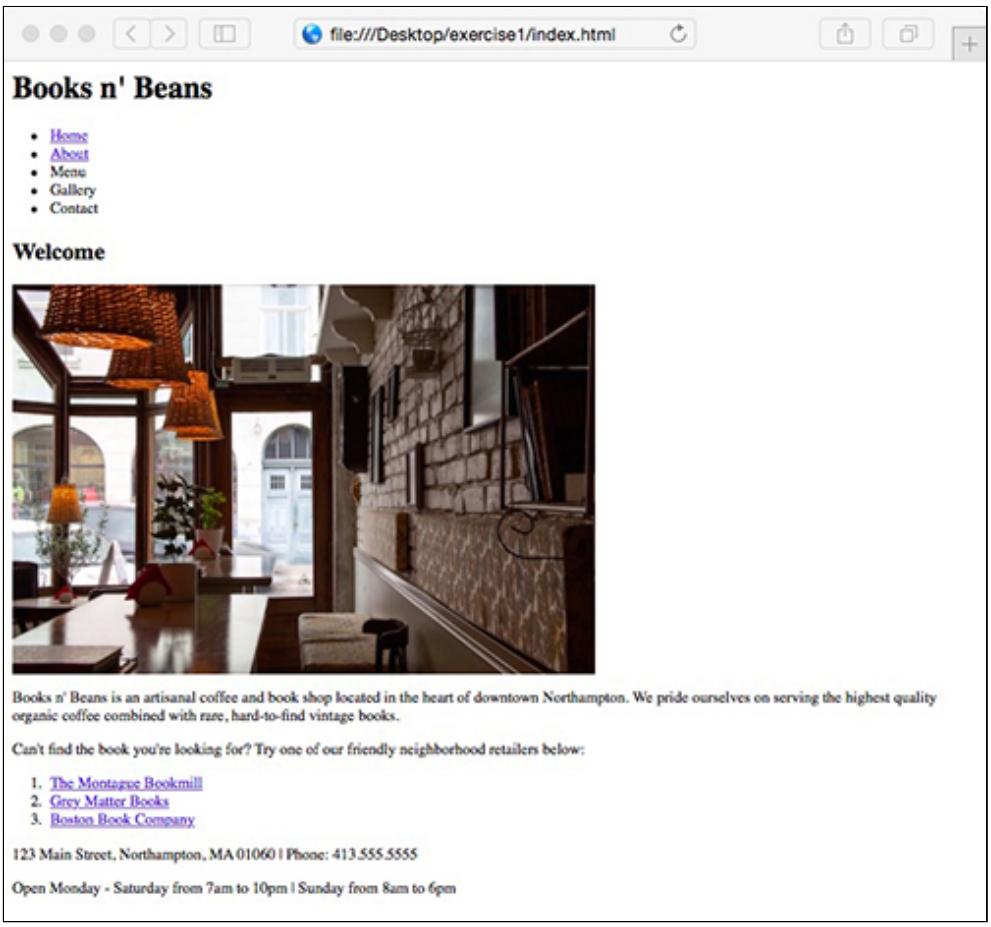
Next, create a folder called "images" within your root folder—the folder where you saved your index.html file. Choose one of the coffee images from the exercise downloads and insert an image link near the top of the page, under the "Welcome" heading.

The image link format should look similar to this, with the file name and alt text customized to match your chosen image:

```
<img src="images/coffee.jpg" alt="coffee">
```

Remember to add an **alt** attribute describing the content of the image. The alternative text should match whichever image you end up choosing for the Home page. Save your file and preview your page. Is the image displaying as intended? If not, check that your file path is spelled correctly and the image is located in the correct place.

Save and preview the document to make sure the image displays properly. If you've coded your page correctly, you should see something like this:



If the images are broken, try using an absolute file path. An image link with an absolute file path looks like this:

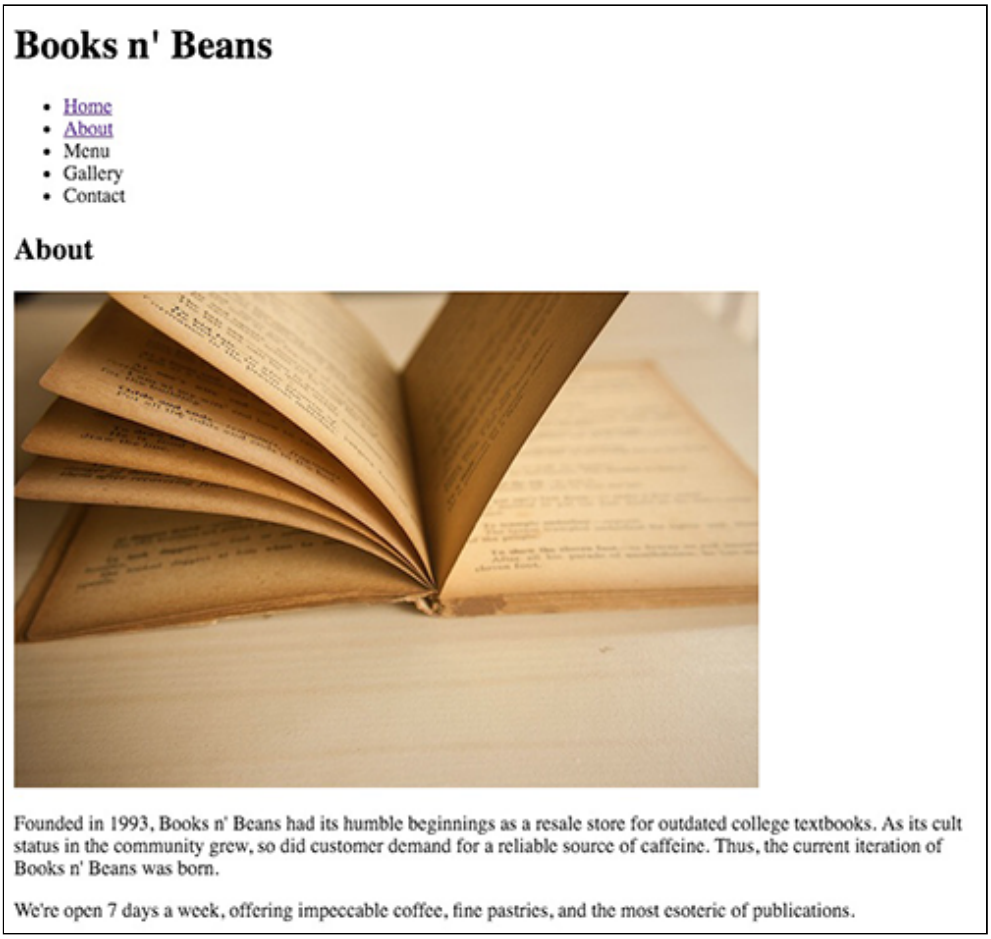
```
<img src="http://yourdomain.com/images/coffee.jpg" alt="coffee">
```

Mark Up the About Page

Are you feeling like an HTML expert? That's great! Let's give this markup stuff another go, this time on the About page.

Open a new document in Sublime and save the document as about.html. Mark up the basic document structure for the About page, just like you did with the Home page. After adding a head section to the document, remember to customize the page title.

Add the correct page <h1, h2... > headings to "Books n' Beans" and "About," and wrap the remaining blocks of text in paragraph tags <p>. This page doesn't contain any links or ordered lists, but remember to wrap the site navigation items in `<ul>` tags. Browse through the images folder and place an image for the About page directly below the About heading. There are lots of great images to choose from here!



Before uploading our site to the Web, we've got to make sure that the navigation menu links take us to all of the right pages. We can do this using **local navigation links**—links that lead to a page on the same Web site. Local navigation links can be formatted as relative urls (urls that refer to the same directory as your Web site url) or absolute urls (urls that link to a complete file path anywhere on the Web).

Relative url examples:

```
<a href="index.html">Home Page</a>
```

```
<a href="exercise1/index.html">Home Page</a>
```

Absolute url examples:

```
<a href="http://mygreatwebsite.com/index.html">Home Page</a>
```

```
<a href="http://mygreatwebsite.com/exercise1/index.html"> Home Page</a>
```

You can use relative urls any time we're referring to a link within our own Web server. When referring to a link elsewhere on the Web, you must use an absolute url.

When you're finished formatting the urls, both the Home and About links should lead to the correct pages. After adding local navigation links to your pages, remember to preview your work in the browser to make sure all links are functioning as expected.

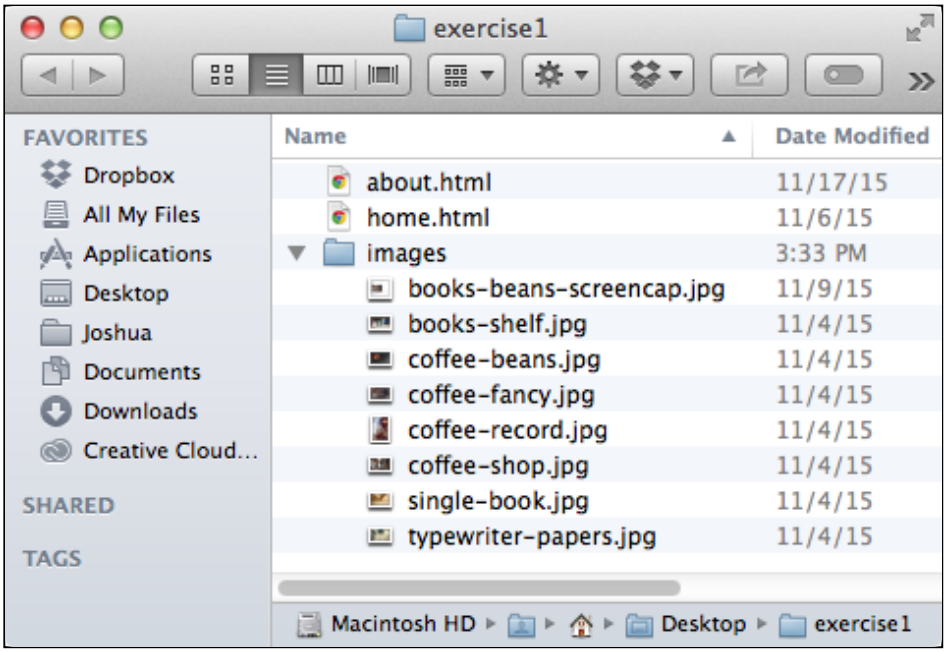
```
<ul>
<li><a href="index.html">Home</a></li>
<li><a href="about.html">About</a></li>
<li>Menu</li>
<li>Contact</li>
</ul>
```

We'll add links to the Menu and Contact pages after building out these pages in a future exercise! In the meantime, you can leave those menu items as unlinked list items.

FTPing Your Work

As you learned in Lecture One, you need to post your finished work on your Web server via FTP.

The local folder on your computer with your Books n' Beans files should look like this:

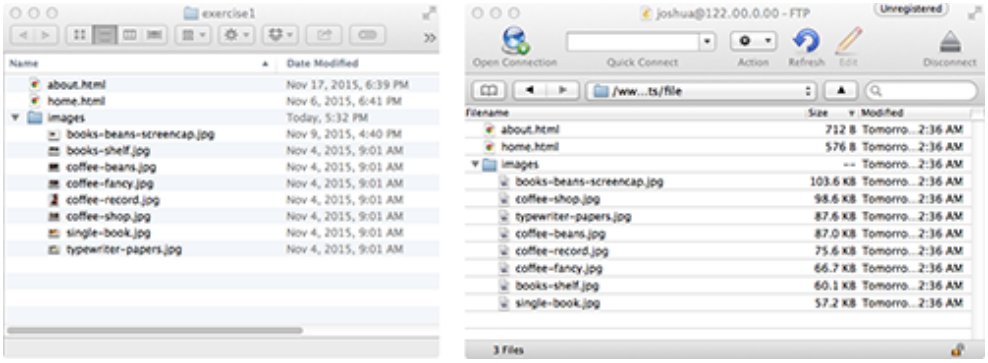


Log in to your server using the information provided by your host. You'll need the host URL to which you are posting (this may or may not be different from your site's domain name—your hosting company will tell you), your username, and your password. If you can't log in, check with your host to make sure you're using the correct information.

Once logged in, you should see two panes in the FTP program, one for your local (hard drive) files and one for your server files. Navigate in the local pane

to the files you wish to upload.

For this exercise, you'll most likely want to FTP your files and image folder to the main area of your server. To do so, simply click on the files and folders you want to FTP from the local pane and either click the upload arrow button or drag them to the server pane.



This is a side-by-side view of your local folder (left) and Cyberduck (right). Click the image to view a larger version.

Part 2: Semantic Scavenger Hunt

Now that you've got a good sense of the HTML document flow, it's time to do a little scavenger hunt! Do a Google search for coffee shops or cafés in your local area. If you can't find anything nearby, it's okay to expand your search results.

Find a nice-looking coffee shop Web site and describe which parts of the page correspond to the tags we've learned about in this lesson.

For this part of the assignment, you'll need to experiment with using your Web browser's "View Page Source" feature, which allows you to take a behind the scenes look at a Web site's code. You can access this feature in most Web browsers by right-clicking on an empty area of the page (or ctrl-click on a Mac) and selecting "View Page Source" from the list of options.

This is a small sample of code from Peixoto Coffee's website viewed using the page source option in a web browser. Click the image to view a larger version.

Try to find a site that contains at least one example of:

- heading tags

- paragraph tags
- an ordered or unordered list (a navigation menu counts!)
- images
- anchor links

Then, submit the site URL with answers to the following questions:

1. What's one example of heading text on this Web site? (Hint: heading text refers to heading tags like <h1> <h2> <h6>, not to the <head> section of the document)
2. What's one example of paragraph text on this Web site?
3. What's one example of an ordered or unordered list?
4. What are three examples of anchors link on this Web site?
5. Are any images on the page also clickable links?

If you're not able to find any clear examples of headings, paragraph tags, etc. in the site's source code, that's OK. You can try to use the "View Page Source" option to answer the exercise questions, or you can make educated guesses about which parts of the page best represent headings, paragraphs, lists, images, and anchor links.

Grading Criteria:

What your instructor expects you to do:

☐ Demonstrate the ability to use HTML to create the first two pages of a site and structure provided content.

☐ Demonstrate the ability to create clean, accurate, and standards compliant code in your HTML documents.

☐ Demonstrate the ability to FTP a site to the Web, testing and troubleshooting each page and fixing any issues.

☐ Show your understanding of common HTML tags are used by finding a small business cafe site and posting an analysis of how it is structured.

How to Post:

Once you're done, go to the Dropbox for this exercise and post both parts of the assignment in one submission. For Part 1, post the URL of your Books n' Beans home page along with brief comments on your experience. For Part 2, post the URL of your "Scavenger Hunt" coffee shop site along with your analysis of its HTML use.

If you have a question before sending your completed exercise for grading, be sure to contact your instructor via the People area.

I look forward to seeing your work!

sessionscollege
FOR PROFESSIONAL DESIGN

Lecture 2

CSS Color and Typography

Now that we've built up a solid foundation in HTML markup, it's time to begin exploring the limitless potential of CSS.

I like to imagine that HTML represents the plain black outlines in a coloring book, and CSS represents a giant box of crayons. With HTML, we can build the structure of our Web site. Then, with CSS, we can turn it into a work of art.

We've got the basic structure. Now it's time to color in the lines! Photos from Pixabay

In this lesson, we'll explore how CSS works and how to link it to your Web pages. You'll learn to write CSS rules and apply them to HTML elements and selectors: classes, IDs, and spans. Finally, and most excitingly, you'll learn how to use CSS to customize color and typography.

Learning Objectives

In this lecture, you can expect to:

- Learn the benefits of using CSS to control the design of a Web site.
- Learn the three primary ways of applying CSS to an HTML page.
- Learn how to write CSS rules.
- Learn how to use selectors: classes, IDs, and spans.
- Learn how to use CSS to control color and typography.

What is CSS?

In Lecture One, we learned that HTML and CSS are languages that work together to describe the structure and presentation of a Web page. We also learned that, while HTML markup is essential to the structure of any Web document, it falls short in the presentation department. That's where CSS swoops in to save the day.

It's a bird! It's a plane! It's CSS, here to make your Web pages more functional and aesthetically pleasing!

note

CSS stands for Cascading Style Sheets.

note

The CSS Zen Garden site includes 200 different CSS designs of the same page.

CSS, which is short for **Cascading Style Sheets**, describes how the HTML markup we've added to our Web page should be displayed onscreen. In the past, HTML font and color styles had to be added tag by tag and page by page. Today, a single CSS style sheet can be used to control the colors, fonts, and layout for an entire Web site. CSS doesn't just make the Web more beautiful, it also makes Web sites easier to troubleshoot, edit, and update over time.

To illustrate the power of CSS, let's take a wander in the garden—a CSS garden.

The CSS Zen Garden

Because CSS is such a visual language, one way to demonstrate its power and flexibility is to show, rather than tell. Below are four Web page designs published at the CSS Zen Garden site. Click on the graphic to launch a large version in a new window.

CSS Zen Garden Web page designs

note

You can attach CSS to an HTML document via external style sheets, internal style sheets, or inline styles.

Can you guess what this smiling robot, candlelit kiss, geometric pop of color, and noir city all have in common?

The answer? Every one of these designs is using the exact same HTML markup, but with a *different* CSS style sheet. Pretty cool, huh?

These designs are from a project called the CSS Zen Garden, which was launched in 2003 as a way to showcase the power of CSS. Today, the CSS Zen Garden site features more than 200 different designs. That's over 200 unique-looking Web sites, united by a single HTML file. Take some time to look through the Zen Garden gallery, noting all of the different font styles, colors, images, and layouts the designers have used on these pages. Feel free to return to CSS Zen Garden any time you need a little inspiration!

Now that you've witnessed the flexibility of CSS, let's learn how to apply some of these styles to our HTML documents.

Applying CSS to an HTML document

There are three different ways to add CSS to an HTML document: external style sheets, internal style sheets, and inline styles. We'll be focusing primarily on external style sheets in this course, but it's important to familiarize yourself with the ins and outs of each method.

Inline Styles

External style sheets allow you to control styles for an entire site in one file.

Using inline styles, you can put CSS rules right inside the body section of your HTML page, like so:

```
<html>
<head>
<title>My Web Page</title>
</head>
<body>
<p style="text-align: right">Lorem ipsum dolor sit
amet.</p>
</body>
</html>
```

The thing is, this method isn't any different from applying presentational formatting to individual HTML tags (the old "tag by tag, page by page" method I mentioned earlier). You may see inline styles used on some Web pages, but I'd recommend avoiding them when you can.

Internal Style Sheets

A second method is to put a CSS style sheet inside the head of your HTML document by enclosing the CSS rules in a `<style>` tag:

```
<html>
<head>
<title>My Web Page</title>
<style type="text/css">
p {
    text-align: justify;
}
</style>
</head>
<body>
<p>Lorem ipsum dolor sit amet, consectetuer
adipiscing elit.</p>
</body>
</html>
```

The basic structure of CSS code includes a selector, a property, and a value.

The internal style sheet method is a step above using inline styles, as it allows you to style elements across an entire page. This works fine if you're creating and styling just one Web page. But if you're working with an entire Web site, it's not the most practical method. Presumably, you will want to keep the look of all of your pages pretty consistent, and updating a site can become a nightmare if each page has its own style sheet.

If you want to, say, change the color of the links in your whole site to green, you would have to change the color of the anchor selector in the head of each and every page. So the internal style sheet approach doesn't really take advantage of the power of CSS. You don't experience that until you work with external style sheets.

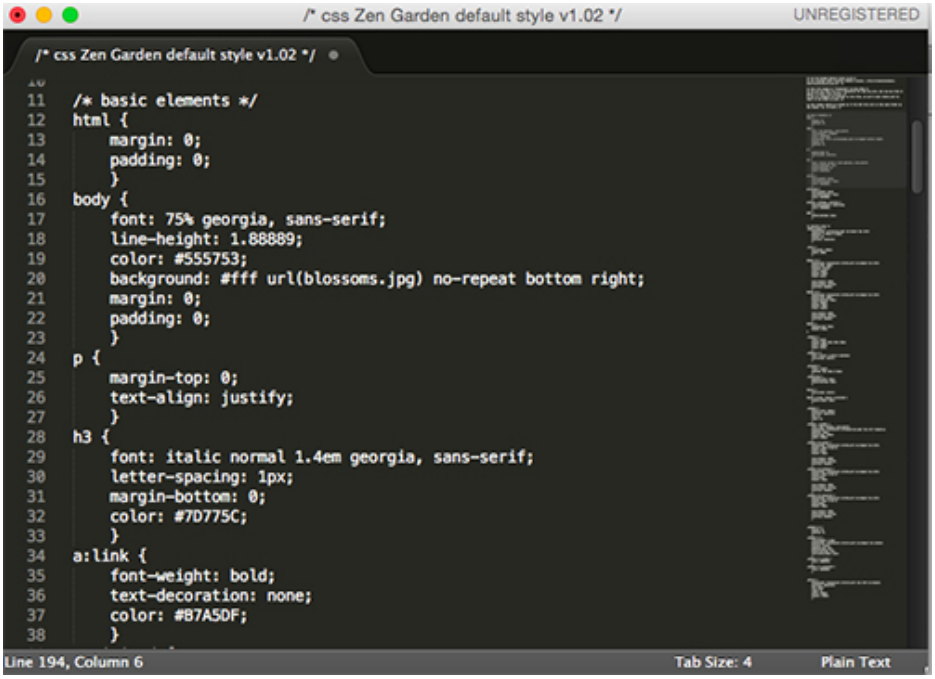
External Style Sheets

External style sheets give you the ability to control the look and feel of your entire Web site with a single file.

Here, instead of including the CSS rules within the HTML source code, you include them in a separate document that you link in the head section of your HTML document. The style sheet itself is just a text document (created with any text editor) with a list of all of the relevant CSS rules. The text document is saved with the .css extension.

note

Any HTML element or tag can act as a selector in your CSS code.



A CSS Zen Garden stylesheet downloaded and viewed in Sublime

In the following example, we are linking to a style sheet named "style.css" located in the "css" folder of a local site.

```
<html>
<head>
<title>My Web Page</title>
<link rel="stylesheet" type="text/css"
href="css/style.css" />
</head>
<body>
<p>Lorem ipsum dolor sit amet, consectetuer
adipiscing elit.</p>
</body>
</html>
```

note

CSS styles cascade down from parent to child elements by default. You can style a child element to override inherited values.

Attaching an external style sheet is the preferred method of working with CSS, as you can link one style sheet to multiple pages. You can have a single style sheet that controls all of the pages in your Web site. If you need to change the colors of all the links within a Web site, instead of making changes tag by tag, or page by page, you simply change one line in the global CSS style sheet.

The Basic Format of CSS

The structure of CSS code is fairly straightforward. Here's some sample code:

```
body { background-color: #eee; }
```

The code pictured above is used to apply a background color to the body element of the HTML document. The basic format for all CSS is as follows:

Keep your CSS file in a standard location at the root of your site.

The **selector** indicates which HTML tag the CSS rule gets applied to. In the case above, it's the body tag.

The **property** defines which aspect of the HTML element is being styled. In the example above, a background color is being applied to the body of the site. Properties can include anything from color and font style, to size, position, line height, and more.

The **value**, just as it sounds, indicates the value of the CSS property. For example, what's the exact color of the property? What size is the text? Where should the selected element appear on the page?

In the example above, the hexadecimal value #eee defines the background color of the site as a very light gray. (The color of any HTML text or page element can be styled in CSS using hexadecimal values or default descriptions such as "white" and "red".)

So at this point we know that this is the basic structure of a CSS rule:

```
selector { property: value; }
```

One neat thing about CSS is that any HTML element (or "tag") can act as a selector. You can apply CSS styles to the entire body element (as we demonstrated above), to the paragraph element (p tag), to list elements (li tags), and so on. Every HTML element can be styled using CSS—though it's worth noting that not all properties will work with every element.

A single CSS selector can contain a number of different properties. The important thing to remember is that every property must be separated by a semicolon (;). For example, if you wanted to define the font family, font size, and font color for the paragraphs on your site, your CSS code might look something like this:

```
p {
    font-family: Helvetica;
    font-size: 1.2em;
        color: #669900;
}
```

It's important to include a semicolon at the end of every property declaration, even if the selector only contains a single property. It's also good practice to include only one property per line, as pictured above. It's technically OK to write out multiple properties on a single line, each separated by a semicolon, but the code quickly gets difficult to read!

CSS Inheritance

Now, before we start adding CSS styles willy nilly, it's important to understand the basic order of operations in CSS (the *cascading* effect in Cascading Style Sheets). While not all properties follow this rule, there are a number of CSS properties that are bound by the rules of **CSS inheritance**. CSS inheritance refers to *children* elements that inherit the properties of their *parents*.

The paragraph element, for example, is the *child* of the body element, meaning that the paragraph is contained within the body of the site. Because of this, the paragraph element will *inherit* certain CSS rules that we define for the body.

Here's an example:

```
body {
        font-size: 13px;
        color: #555;
}

p {
```

important!

Remember to close each CSS rule with a semi-colon (;).

```
        color: #fff;
}
```

By default, the paragraph element inherits both the font and color settings of its parent: the body element. This means that any text within a paragraph tag will be 13px; there's no need to duplicate that size rule by adding it to the paragraph selector.

If we want to style the text inside of our paragraphs differently than the parent element, we can *override* the inherited values. So, above, while we colored the text of the body element gray, we've *overridden* that color in the paragraph element by specifying that its text should be colored white.

OK, before we get *too* far down the CSS rabbit hole (and end up in a pool of tears!) let's put some of these concepts into practice.

Writing CSS

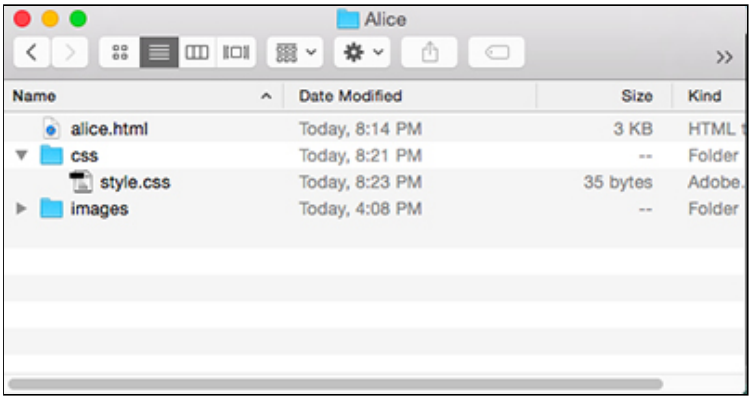
Let's write some CSS rules. We'll use the Alice page we created in Lesson One.

1. Create a CSS document

Locate the local folder where you saved your alice.html document from Lesson One. In the root folder, create a new folder named "css". Open your text editor and save a document named "style.css" into that folder.

note

CSS rules can style colors, fonts, and even link states.



2. Link to it from your alice.html document

Open alice.html in your text editor and insert a link to your CSS document in the head section, under the title tag:

```
<head>
<title>Alice's Adventures in Wonderland</title>
<link rel="stylesheet" type="text/css" href="css/style.css" />
</head>
```

3. Write Some CSS Rules

Now let's practice writing some CSS rules to add some style to your page.

Suppose I want to change all body text to Helvetica. I would select the page body, make the property "font-family", and choose the value "helvetica".

The formula is as follows:

```
selector {  property: value; }
```

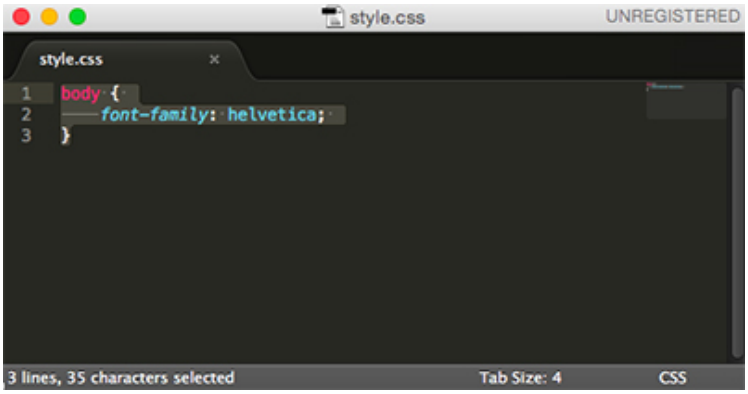
note

Selectors can be combined to make your code more compact.

The code would be written like so:

```
body {
font-family: helvetica;
}
```

My CSS document would look like this:



Now it's your turn. We've learned the most important HTML tags, and each of them are present in the Alice.html document.

Write a CSS rule for each style below, saving your CSS document as you go, and previewing the HTML document. (In each case, the property and value are provided.)

1. Display all headings as sans-serif

```
font-family: sans-serif;
```

2. Display all paragraph text as verdana

```
font-family: verdana;
```

important!

Make sure you comment your CSS rules like so: /*comment*/.

3. Display the ordered list text as monospace

```
font-family: monospace;
```

4. Display the main header at 28 pixels

```
font-size: 28px;
```

5. Make the main header white

```
color: #FFFFFF;
```

6. Display the second and third header at 18 pixels

```
font-size: 18px;
```

7. Make the second and third header a light blue

```
color: #1ABC9C;
```

8. Display the ordered list at 14 pixels

```
font-size: 14px;
```

9. Make the ordered list text grey

```
color: #bdc3c7;
```

note

Class, ID, and span selectors help you apply styles to specific parts of your HTML documents.

10. Display the paragraph text at 14px

```
font-size: 14px;
```

11. Make all paragraph text a light grey

```
color: #ECF0F1;
```

12. Make the background color of the page a dark blue

```
background-color: #34495E;
```

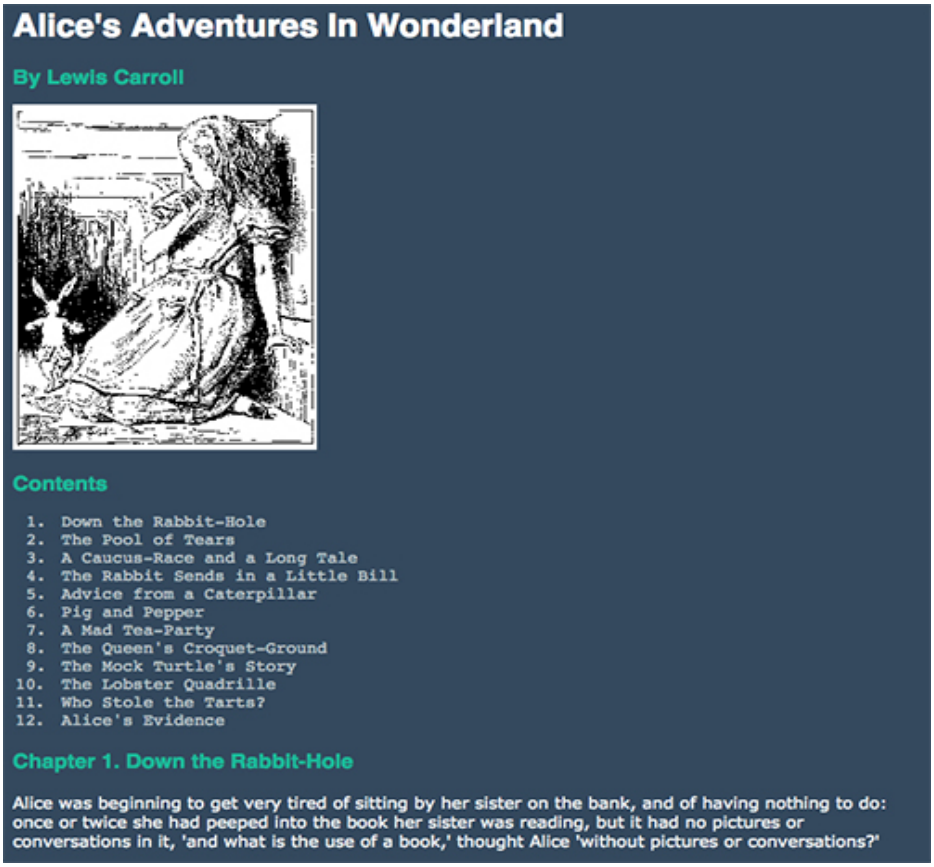
13. Make the link text color a dark orange

```
color: #F39C12;
```

14. Make the link hover color a light orange

```
background-color: #F1C40F;
```

Done? Your document previewed in the browser should look like this:



We're going to cover how to style typography and color thoroughly later in this lesson. For now, we've learned that CSS rules can be applied any common, visible HTML tag, that they obey rules of inheritance, and that they must be typed precisely!

Here are three more CSS concepts that will help you write clean CSS code:

1. Combining Selectors

Do you have a style you wish to apply to several selectors? You can combine more than one selector, allowing you to apply styling to a bunch of selectors at once. Just separate them by commas:

note

ID selectors or divs are used to break your page into sections.

```
h1, h2, h3 {
    color: #00ffff;
}
```

2. Adding More Than One Value

For some CSS properties, you can set more than one value. To do so, just separate the values with a comma (and note that values with more than one word are enclosed in quotes):

```
h1, h2, h3 {
    color: #00ffff;
    font-family: Verdana, "Trebuchet MS", sans-serif;
}
```

3. Adding Comments

Want to remember your code when you revisit it weeks or months later? Be a conscientious coder and add comments to your CSS style sheet. Comments begin with a forward slash and an asterisk `/*`, and end with an asterisk and forward slash `*/`, like this:

```
/*This is a comment, which is used to clarify snippets
of CSS code*/
```

Comments are ignored by Web browsers, so your comments won't be displayed on your Web page!

Coffee Break: In this next section we go further down the CSS rabbit hole with the topic of CSS selectors. If you need a break, take one now and return to this spot.

Using Selector Types

Class, ID, and Span Selectors

We've just learned that we can use CSS to style element selectors like the site body, paragraphs, lists, anchor links, and more.

But what happens when we're looking for more fine-grained control—like styling a single paragraph on a page, or styling a single world in a sentence? And what about applying styles to parts of the page that don't have traditional element selectors—like a special donation button, or a site navigation bar?

That's where class, ID, and span selectors come in. When a p or h1 tag just won't do, these selectors are used to help define and style different parts of an HTML document. Unlike p tags and h1 tags, classes, IDs and spans don't have any meaning on their own. It's the CSS that gets applied to these elements that adds significance.

Classes

If you add a CSS rule to a paragraph selector, that rule gets applied to *all* paragraphs in the relevant HTML document(s). But what if you wanted to apply certain styles to *some* paragraphs, but not others? One way to do that is to use the class attribute.

What is a class attribute? The **class attribute** simply lets you categorize your elements into various kinds, and any number of elements on a page can have the same class.

For example, suppose you want to apply a yellow background to all news items on your page (and there may be more than one paragraph that is a news item), but not

note

ID selectors are styled with a hash sign (#) in your CSS.

all paragraphs of your page. You can apply a specific class to those paragraphs within the HTML document that are news items, and then style that class in your CSS.

Start a new HTML document and add the following code to the body. Notice the class attribute here and the name I have given it, "news".

So, first add the paragraph class to the HTML:

```
<p class="news">See the beautiful blue columbines flowering
right now in Mrs. Morgan's garden!</p>
<p>Old-fashioned and easy to grow, columbines are a
long-time garden favorite. Delicate green or blue-green
leaves are usually divided and fan shaped. Bell-shaped
flowers appear on leafy stems.</p>
```

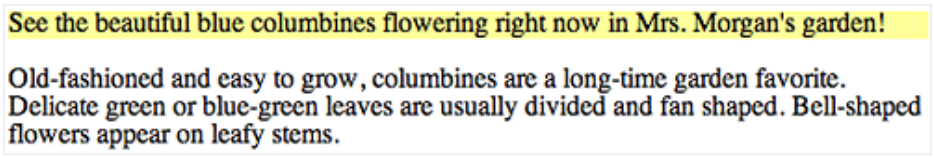
If you preview the page, nothing will happen yet, because we have not assigned any CSS rules to the class. For the purposes of a quick demonstration, let's do that using an internal style sheet.

Move up to your document's <head> area. Within the <head></head> tags, insert the tags that define our internal style sheet. Inside those tags, we can style our news class:

```
<head>
<style type="text/css">
.news {
    background-color: #ffff99;
}
</style>
</head>
```

Important note: Be sure to put a period (.) in front of all class names—that indicates that we are styling a class rather than a regular HTML element.

Now, the resulting page looks something like this:



You can also restrict the class styling to certain elements. If you want to only apply a yellow background color to *paragraphs* with the "news" class (perhaps you want to style images with that class name differently), just put the paragraph selector in front of the class name:

```
p.news {
    background-color: #ffff99;
}
```

note

Spans help you style inline sections of text.

IDs

Another way to select elements for styling is by assigning an ID attribute. **IDs** are special, because you can only assign one element on a page a particular ID. This is useful for breaking up your page into unique sections. For example, most Web pages have some sort of header, a navigation menu, and a main content area. You can assign custom IDs to these different sections, and apply styles to them. The CSS for IDs looks like a hash mark, followed by a unique element name:

```
#menu {
        text-align: center;
        color: #444;
}

#sidebar2 {
        width: 250px;
        font-size: 12px;
}
```

To try this out, open Sublime and start with a fresh HTML document. Insert the following code in the body. You should recognize that this will create a heading, a subheading, an unordered (bulleted) list, and a paragraph of text.

```
<h1>My Web Business</h1>
<h2>Menu</h2>
<ul>
<li><a href="index.html">Home</a></li>
<li><a href="about">About</a></li>
<li><a href="projects">Projects</a></li>
<li><a href="contact">Contact</a></li>
</ul>
<p> This portfolio represents some of the biggest,
baddest projects we've completed this year. Browse,
interact, and enjoy, then consider whether our design
capabilities can meet your branding needs. We'd be
delighted to hear from you. </p>
```

An unstyled page displayed in the browser

Let's divide this page into logical sections. I've added some extra spacing here so you can see the hierarchy of our sections:

```
"<div id="header">
  <h1>My Web Business</h1>
</div>
<div id="menu">
  <h2>Menu</h2>
    <ul>
      <li><a href="index.html">Home</a></li>
      ...
    </ul>
```

```
</div>
<div id="main">
  <p>This portfolio represents some of the biggest, baddest
projects...
  </p>
</div>
```

You may be wondering at this point: "What are div tags?"

The tag defines a block-level section or division within a Web page. (When an element is a **block-level element**, it has a line break before and after it. This is contrasted with **inline elements**, which do not contain line breaks.)

The name block-level element connotes its function in a Web page: They can be stacked, moved around, and sized in various ways. Div tags are very useful for dividing up a page into logical sections.

Once we mark up the page with ID-labeled divs, we can then style those sections. And until the divs are styled, you won't see any visual difference on your page compared to the old version without divs.

To style these divs, let's try an external style sheet. First, create a new, blank text document and save it as business.css. Save it in the same directory as your HTML page. In the CSS document, the only code belongs to the styling—no HTML tags here!

In your new, blank file, enter the following. Don't worry if you don't know what all of these properties do yet—all of that comes later!

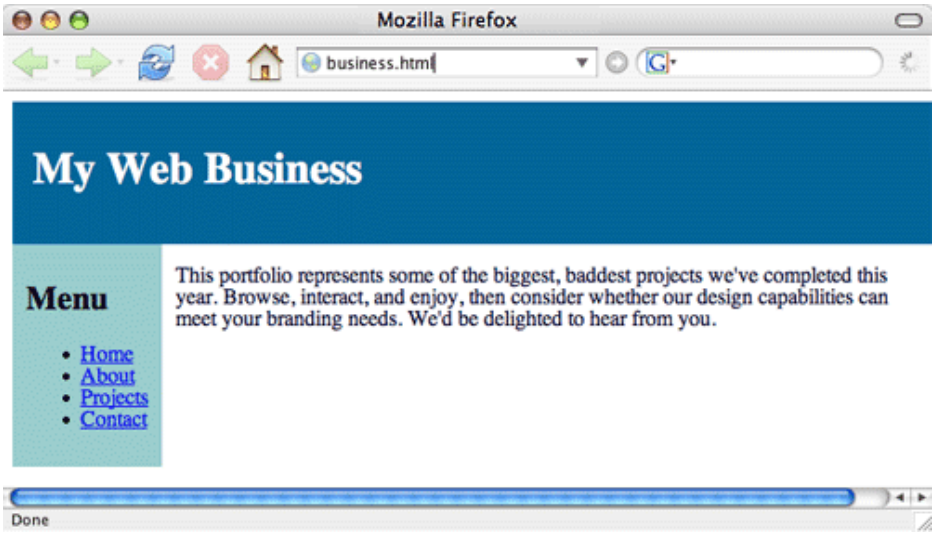
```
#header {
    background-color: #006699;
    color: white;
    width: 100%;
    padding: 15px;
}
#menu {
    float: left;
    background-color: #99cccc;
    padding: 10px;
    margin-right: 10px;
}
#main {
    color: #000033;
}
```

Note the hash sign (#) in front of the ID names—that just indicates that we are styling a specific ID attribute.

Now we need to get our HTML page and our CSS document talking to one another. Go back to your HTML page and add the following code between your <head> </head> tags. This code tells the HTML page where the CSS rules are:

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

Now save your page and check out the results!



The page with div tags used to control layout

Also note that you can style particular elements inside of divs. Say we wanted to color the anchor elements in the "menu" div differently than the links in the main content div. We would add the anchor selector after the ID name to target just the links inside the menu:

```
#menu a {
    color: #000033;
}
```

We can do this with any selector (not just ID selectors). For example, suppose we wanted to color the link items inside of ordered lists differently than the rest of our links. We could target those elements like this:

```
ol a {
    color: #cc0000;
}
```

This tells the browser: "When anchor elements are inside of ordered list tags, color them red."

If we wanted to change the color of our header, or put the menu on the right instead of the left, doing so would be as simple as changing a line or two in our CSS. And since we used an external style sheet file, we could apply the very same styles to as many pages as we like just by adding the appropriately identified `<div>` and `<style>` tags to each HTML page.

Spans

What if you want to apply specific formatting within a paragraph? The `<span>` tag is also useful for marking up content to be styled with CSS.

Used in conjunction with the class and ID attributes, the `<span>` tag helps you assign class attributes to inline sections of text. Unlike the `<div>` tag, which creates a block-level element, `<span>` is an inline element, meaning that you can wrap a word or sentence right within a paragraph, then color or otherwise manipulate it with CSS.

The page with span tags used to control layout

In the above example, I simply wrapped a few words with the tags, like this.

```
<span class="highlight">Browse, interact, and enjoy</span>
```

What I was doing was using a class attribute called "highlight" to bold and color the text.

Let's summarize the general purpose of CSS selectors before we continue.

1. Class selectors are added to HTML tags (such as p or h) to style them in a specific way.
2. ID tags or divs are used to markup areas of a page (such as header, footer, menu, and so on) so that they can be styled.
3. Span tags are used to isolate smaller areas of text for specific styles.

You'll get a chance to apply CSS selectors in Exercise Two. For now, let's take a closer look at how to implement typography and color in CSS.

Typography in CSS

note

CSS enables you to apply a wide range of styles to fonts.

Superior typography is one of the main benefits of using CSS, so let's look at the main parameters for designing type on a Web page.

Selecting a Font-Family

To specify the font in a certain element, you use the **font-family** property. It's a good idea to begin any project by exploring basic options for paragraph and header styling. In your lesson2 downloads folder, you'll find a folder called Emigre

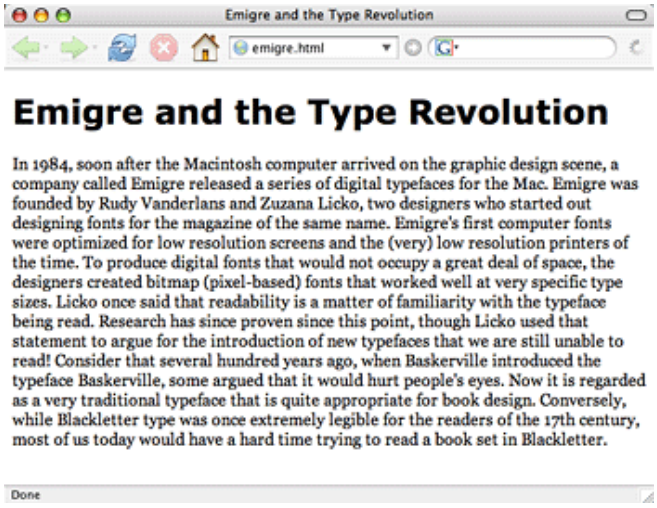
note

The fonts displayed on a Web page are actually pulled from a user's operating system.

containing an HTML file called emigre.html. Download it and structure it as an HTML document.

Create the rules below in your text editor, then save the document with the name emigre.css in your emigre folder, and then view the HTML page in your Web browser (the CSS file is already linked in the HTML page for you.)

```
h1 {
    font-family: Verdana, Arial, sans-serif;
}
p {
    font-family: Georgia, "Times New Roman", serif;
}
```



Goodbye, default browser fonts! Hello, hours of trying to choose between Trebuchet and Tahoma!

Notice anything funny about values specified for `h1` and `p` above? While Verdana and Georgia are the first values listed, a list of font names appears after them. What's the deal with that? When I declare Georgia I expect to see Georgia, not this "Times New Roman" nonsense!

Welcome to the exciting and unpredictable world of typography on the Web! When you declare a specific font family using CSS, the font that gets displayed in your Web browser is actually pulling that font information from your computer's operating system. So, if you don't have Georgia installed on your computer, the browser will continue on down the line until it finds the next best similar font.

How can you ensure that your font choices will look similar across the Web? To accomplish this, you'll want to use **Web safe fonts** in your designs. Web safe fonts are fonts that come pre-installed on many different operating systems. I use a site called CSS Font Stack to copy/paste Web safe font stacks directly into my CSS style sheet. Give it a test run, and make sure to bookmark this invaluable resource for use throughout this course!

Regardless of which font you choose, remember to include a generic font-family at the end of the value list. Specifying a generic font-family should display a font somewhat similar to the font you really want. There are five different generic font-family types:

important!

Use Web-safe fonts in addition to a generic font-family to make sure your

- **Serif:** Serif fonts have small finishing strokes (called "serifs"), and are generally associated with more traditional typefaces. Times New Roman is a prime example. Georgia is a font made specifically for screen legibility, making it a great choice for Web use.
- **Sans-serif:** Sans-serif fonts are fonts that lack serifs. Arial is a typical sans-serif font, and Verdana is a sans-serif font

text displays satisfactorily.

made specifically for the screen.

- **Cursive:** Cursive fonts have strokes joining the letters together, like cursive writing. Zapfino is an example of a cursive font.
- **Fantasy:** Fantasy fonts are typefaces used primarily for decorative purposes. These can work well for display purposes but it's generally not a good idea to use them for long blocks of text. Papyrus is an example of a fantasy font.
- **Monospace:** Monospace fonts are fonts whose characters all have the same width, like a typewriter typeface. Courier and Monaco are examples of monotype fonts. They are often used to represent samples of computer code.

note

Font-family names with more than one word, like "Times New Roman," are enclosed in quotes.

Experiment with different values for the font-family property, viewing the changes in your browser, before you continue—and feel free to use CSS Font Stack for easy access to lots of great, Web safe fonts!

Size Properties and Values

You can set the size of a font using the **font-size property**.

```
h1 {
    font-family: Verdana, Arial, sans-serif;
    font-size: 1.8em;
}
p {
    font-family: Georgia, "Times New Roman", serif;
    font-size: .9em;
}
```

There are various units of measurement available to set the size of your font. You can set the size using **absolute** units, which represent a fixed dimension. These include:

- inches (in)
- centimeters (cm)
- millimeters (mm)
- points (pt) (1/72nd of an inch)
- picas (pc) (12 points)

You can also set the size using one of the following keywords:

- xx-small
- x-small
- small
- medium

note

The font-size property can use absolute or relative measures.

- large
- x-large
- xx-large
- smaller
- larger

Finally, you can use relative units, which represent measurements that vary in relation to the user's default font size. These include:

- pixels (px) (which vary according to the computer's pixel size)
- em (1 em is the default font size)
- x-height (ex) (the x-height is equal to the font's lowercase "x")
- percentages (%)

So, what measurement should you use? Most designers tend to use either pixels or ems.

Pixels let you have more precise control over the design of the page (particularly when you measure other elements in pixels). However, some Internet Explorer users can't resize the text when it is set in pixels. This decreases the usability of the page, as users won't be able to alter the text size to their comfort level. Ems, which are a relative unit, give you less control over the look of the page, but are a bit more friendly to the end user.

Weights, Styles, Variants, and Decorations

If you like typography, you'll be excited to know that further options exist beyond selecting font-family and size. Weights, styles, variants, and decorations can each be modified to give you more precision.

Try out each of the following on the Emigre page.

1. Using Font-Weight

The **font-weight** property controls the weight (or thickness) of a font.

```
p {
    font-weight: bold;
}
```

Possible values include:

note

Most browsers currently render itaic and oblique fonts the same.

- normal
- bold
- bolder
- lighter
- 100, 200, and so on through 900 (where 400 is normal thickness and 700 is bold)

important!

Remember, use CSS to specify how something looks.

2. Using Font-Style

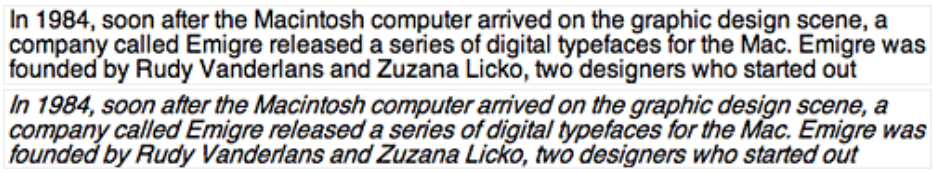
You can set the style of a font using the **font-style** property:

```
p {
    font-style: oblique;
}
```

There are three possible values:

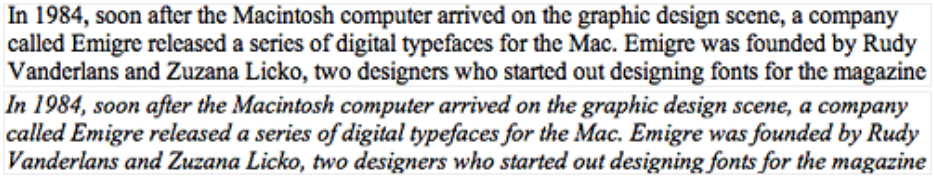
- normal
- italic
- oblique

Oblique styled fonts are just like the normal font, but just tilted a bit forward:



Helvetica normal vs. Helvetica oblique

Italic styled fonts are also tilted like oblique fonts, but they are actually a separate typeface from their normal counterparts (notice the difference between how the i's, m's and n's are formed):



Times New Roman normal vs. Times New Roman italicized

Unfortunately, most modern browsers render italic and oblique fonts the same, so at this point it doesn't much matter that you specify.

At this point we should recall the HTML tags that appear to perform the same function as the font-weight and font-style tags:

- `<em>`: The `<em>` tag indicates emphasized text. By default, it is rendered as italicized text.
- `<strong>`: The `<strong>` tag indicates more strongly emphasized text, and, by default, it is rendered as bolded text.

It *looks* like these tags perform the same function as the font-style and font-weight CSS properties. However, unlike the CSS properties, these tags are not really meant to control the presentation of the text (or specify how it *looks*). Rather, they are meant to point out the *function* of certain text—to point out that that "this text is

important!

Be careful with the underline style, as users will expect a link.

emphasized," rather than "this text is italicized." This information is important to screen readers, for example.

With that said, you may decide that you'd rather emphasize your emphatic text by coloring it red. I could easily do that by styling the tag:

```
em {
    font-style: normal;
    color: red;
}
```

Then, this HTML:

```
<p>I was <em>very, very</em> upset by the
disheartening news.</p>
```

would look like this:

I was very, very upset by the disheartening news.

Furthermore, remember that not all *italicized* text is really *emphasized* text. Take a book title, for example. Conventionally, it is represented by italicized text, but you wouldn't want to say that we are trying to *put emphasis* on text titles by italicizing them. In fact, it would be semantically inappropriate to wrap them in tags. Rather, we would probably want to mark them up with their own class:

```
<span class="book" >The Lion, the Witch, and the
Wardrobe</span>by C.S. Lewis
```

and then style that class as italicized:

```
.book {
    font-style: italic;
}
```

That way, we don't rely on HTML for presentation, but leave that up to CSS!

3. Using Font-Variant

You can also turn the text into small caps by using the **font-variant property:**

```
p {
    font-variant: small-caps;
}
```

This takes one of two values: small-caps or normal.

The grass is always greener.

THE GRASS IS ALWAYS GREENER.

Normal font-variant on the top. Small-caps on the bottom.

4. Using Text-Transform

Similar to font-variant, the **text-transform** property controls different types of text capitalization.

note

Spacing and line height can improve the readability of the page.

```
p {
    text-transform: lowercase;
}
```

It supports the following values:

- none
- lowercase (transforms are all text to lowercase)
- uppercase (transforms all text to uppercase)
- capitalize (capitalizes the first letter of every word)

text-transform: lowercase;

appears as all lowercase text.

---

text-transform: uppercase;

APPEARS AS ALL UPPERCASE TEXT.

---

text-transform: capitalize;

Appears With First Letter Of Every Word Capitalized.

Normal font-variant on the top. Small-caps on the bottom.

5. Using Text-Decoration

You can add various effects to text by using the **text-decoration** property.

```
p {
    text-decoration: line-through;
}
```

It takes the following values:

note

List styles provide many opportunities to improve on default styles.

- none
- underline (puts a line under the text)
- overline (puts a line over the text)
- line-through (puts a line through the text)

Line-through is of course a classic style to use for the omnipresent online price discount, and "none" can be used to remove the underline that appears by default below most anchor links. Remember, Web users depend on some type of visual signifier to indicate a clickable link. If you remove it, you'll want to change the text color or alter the text style to indicate the presence of an anchor link.

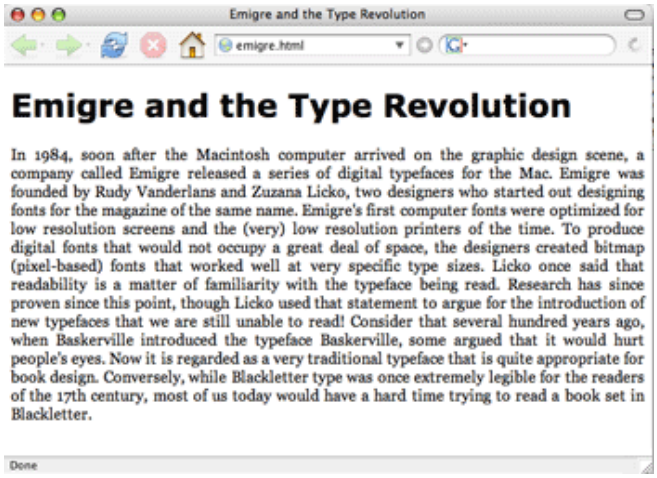
6. Using Alignment

Here's a technique you can apply to a section of a page. You can align text in a block-level element using the **text-align** property.

```
body {
    text-align: justify;
}
```

It takes one of four values:

- left (aligns the text to the left)
- right (aligns the text to the right)
- center (centers the text)
- justify (justifies the text)



Notice the subtle difference between justification and left alignment.

7. Using Letter-Spacing and Line-Height

In print, there are two ways to adjust the spacing in your text: **kerning** is the process of adjusting the spacing between the letters, while **leading** is the spacing between the lines.

Using CSS, you can adjust the horizontal spacing between the letters using the **letter-spacing** property. It takes as its value some length:

```
h1 {
    letter-spacing: 2px;
}
```

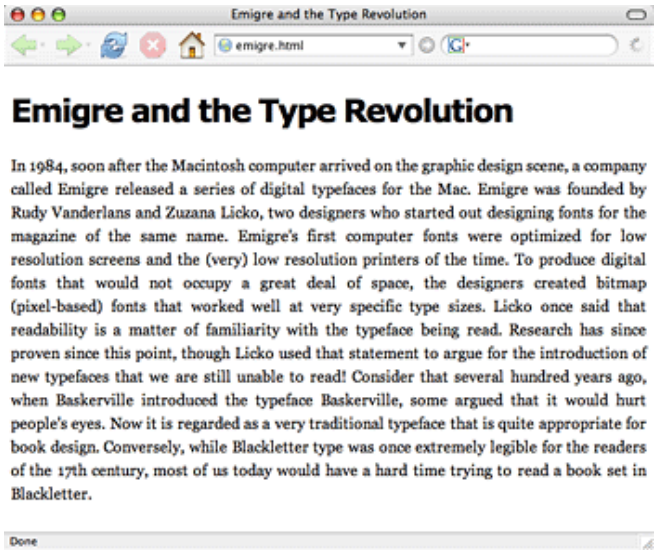
You can assign a negative length (such as, **-2px**) to squeeze the letters more tightly together. Use a positive number to spread them farther apart. This level of control is particularly helpful for headings.

A heading with 2-pixel letter-spacing (top) and a heading with -2-pixel letter-spacing (below).

You can adjust the space between the lines of text using the line-height property. Try applying this to the page body, then to the paragraph.

```
body {
    line-height: 1.5;
}
```

The line-height is the distance from one baseline (the invisible line on which the text rests) to the next. If you assign a line-height using no units of measurement, then the browser multiplies the value by the font-size to determine the line height. So, if you had a 10-pixel font, and you specified a line-height of 1.25, then the line height would be 12 pixels. For the Web, a good line-height to use is 1.5.



Text with the line-height set at 1.5.

I hope you'll agree that the text is much easier to read with these rules applied!

You can review the process of attaching an external style sheet, using divs to control the layout, and writing CSS rules to style text and color in the following video tutorial:

VIDEO TUTORIAL: Styling Web Text

0:00 / 3:41

Lists and Links

In this next section, we'll discuss how to style lists and links. These are two areas of a page where a Web designer can take their site to the next level. We're so used to seeing default options (such as blue links) for these items that a variation can be a nice design detail!

Lists

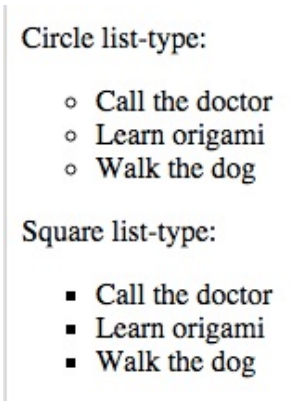
Let's make some modifications to a list, shall we? You can customize lists using the **list-style-type** property:

```
ul {
    list-style-type: square;
}
```

This changes how the list-item marker (whether it's ordered or unordered) will look. There are a bunch of different values for this one. Listed below are just some of the more common list-style types:

- none (No marker. This is an invaluable setting to turn off any list markers, so that you can customize the list to your liking.)
- disc (default bullet—just a little filled circle)
- circle
- square
- decimal (marker is a number)
- decimal-leading-zero (01, 02, 03, etc.)
- lower-roman (i, ii, iii, iv, etc.)

- upper-roman (I, II, III, IV, etc.)
- lower-alpha (a, b, c, etc.)
- upper-alpha (A, B, C, etc.)



Circle and square bullets

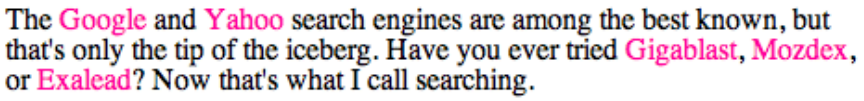
Links/Pseudo-Classes

You can style hyperlinks like any other text. You can, for example, remove the default underlining of links by setting their text-decoration property to none:

```
a {
    text-decoration: none;
}
```

You can also change the default blue color:

```
a {
    color: #ff0099;
}
```



Link color changed to pink.

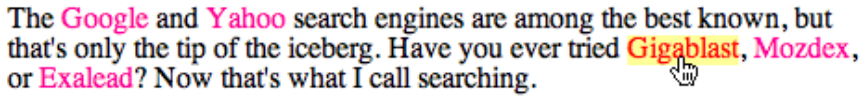
When styling hyperlinks, we often want to indicate when the link is being hovered over, clicked on, or that it has already been visited. We can target this specific state of a link by using **pseudo-classes**. There are four pseudo-classes for the anchor element:

- `a:link` (unvisited link)
- `a:visited` (visited link)
- `a:hover` (moused-over link)
- `a:active` (selected link)

Let's style some links by highlighting them and changing their color to red with a yellow background when the user hovers over them. Open any CSS file you've created so far to explore this concept.

```
a:hover {
    text-decoration: none;
    color: red;
    background-color: #ffff99;
}
```

Now, our links look like this when we hover over them:



It's important to note that link pseudo-classes must appear in a certain sequence in order to work properly. `a:hover` must be listed after `a:link` and `a:visited`, and `a:active` must be listed after `a:hover`.

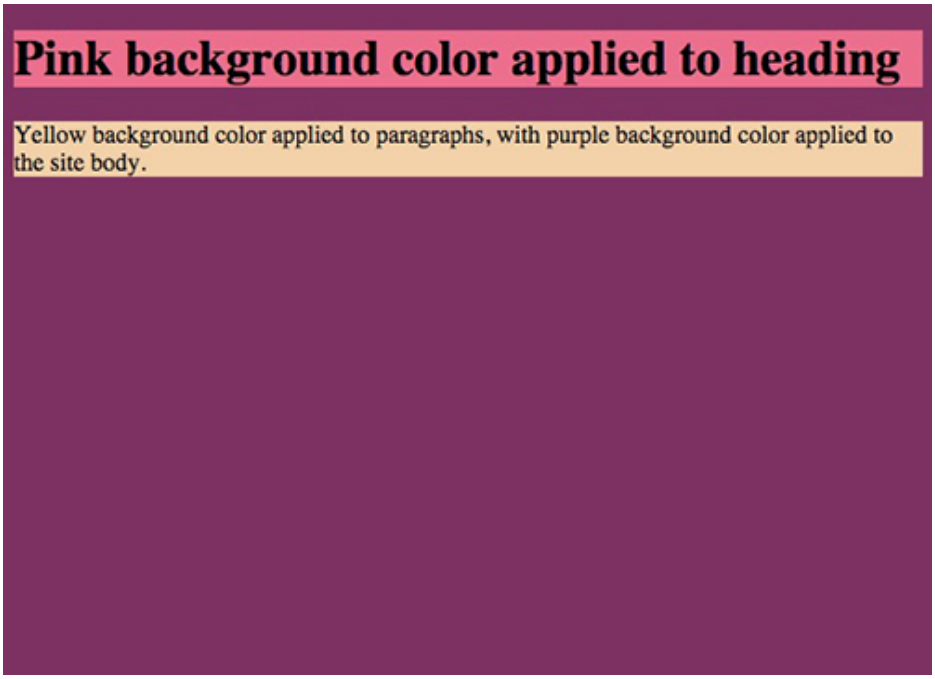
As you can see, link styling offers tremendous potential for creating links that are more interactive and support the overall graphic scheme of the page.

Color and Background Color

As we've discovered in this lesson, the color of any HTML text or page element can be styled in CSS using hexadecimal values. Thus far we have mostly explored how to modify text, so let's turn our attention to colors.

To add a background color to an element, just use the **background-color** property. It takes a color as its value.

In this example, different background colors have been applied to `h1`, `p`, and to the site body:



It's worth noting that the **background-color** property and **color** property sound the same, but they each do something different! Use **background-color** when styling the background color of an element. Use **color** to control the front color of an element.

Colors, Fonts, and Visual Accessibility

With all of the CSS typography options available to you, it may be easy to get a little carried away with using various fonts and font effects.

Use taste and discretion when styling your text and backgrounds. Don't include more than two or three typefaces on one page, and try to use fonts that are optimized for on-screen readability (like Georgia and Verdana).

Be judicious with your color choices. Here are some examples of common color pitfalls:

- Yellow text on a white background, or light-colored text on a light background.
- Brightly colored text on a bright background, which can create an optical illusion called a "vibrating color" effect.
- Red text on a green background (or vise versa), which may be indistinguishable to users' with color-blindness.

We'll discuss some tips on how to choose the *right* colors in the exercise that follows.

In the next lesson

- Learn how to apply the box model to style elements using borders, padding, and margins.
- Learn how to apply relative positioning, absolute positioning, and floats to layout page sections.
- Learn how to create fixed and fluid one and two-column layouts.
- Learn some basic troubleshooting tips for CSS positioning and floats.

Coming Up Next:

Discussion
Share your thoughts and opinions on CSS in the Discussion area.

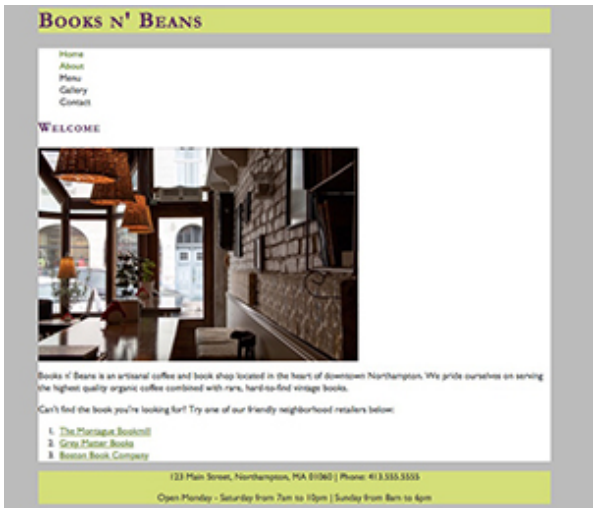
Exercise
Markup your coffee shop site with CSS selectors and style its color and typography.

HTML and CSS | Typographic Transformations

sessionscollege
FOR PROFESSIONAL DESIGN

Exercise 2

Typographic Transformations



In which we add CSS to our stylin' coffee shop site

In Lecture Two, we learned how to add CSS styles to HTML pages. We learned how use HTML elements and different CSS selector types (ID, class, and span) to assign styles to parts of a page. And we explored how to use CSS to format and style page text and color.

In this exercise, we'll add an external stylesheet to our unstyled Books n' Beans page. Then we'll transform the default serif type and boring blue links into something that's a better match for our quirky independent coffee shop/book store!

Important: Before beginning Exercise 2, make sure to duplicate your files so you don't overwrite your Exercise 1 website! The easiest way to do this is to copy the files and paste them into a folder called "exercise2". Remember to use lowercase letters for file and folder names. Short folder names without space marks will be easier to type into the URL after you post your folder online. Clearly label the exercise folders as you work through each exercise in this class.

Performance Objectives

- Attach an external stylesheet to the coffee shop site.
- Use CSS selectors to structure site pages (divs) and differentiate the style of specific areas (classes and spans).
- Develop a basic design for the color and typography of your site.
- Write CSS rules to consistently style site color, typography, and links.
- Upload, validate and troubleshoot your work.

Customizing Fonts and Colors

Using CSS

Ready to start styling? For this exercise, you'll be working on the Books' n Beans site you created in Exercise One.

Start by creating a new folder named "Exercise2". Then, copy and paste the Books n' Beans HTML documents and any associated image files into your Exercise2 folder and begin working from there.

Remember, you can only have one index.html file in each directory (folder) on your Web server, so it's good practice to separate your exercises into folders with distinct names.

note

Note: If you need to resubmit Exercise One but you overwrite the files when working on Exercise Two, you instructor won't be able to accept the

Assignments are evaluated for creativity, technical proficiency, and understanding of concepts covered in the lecture.

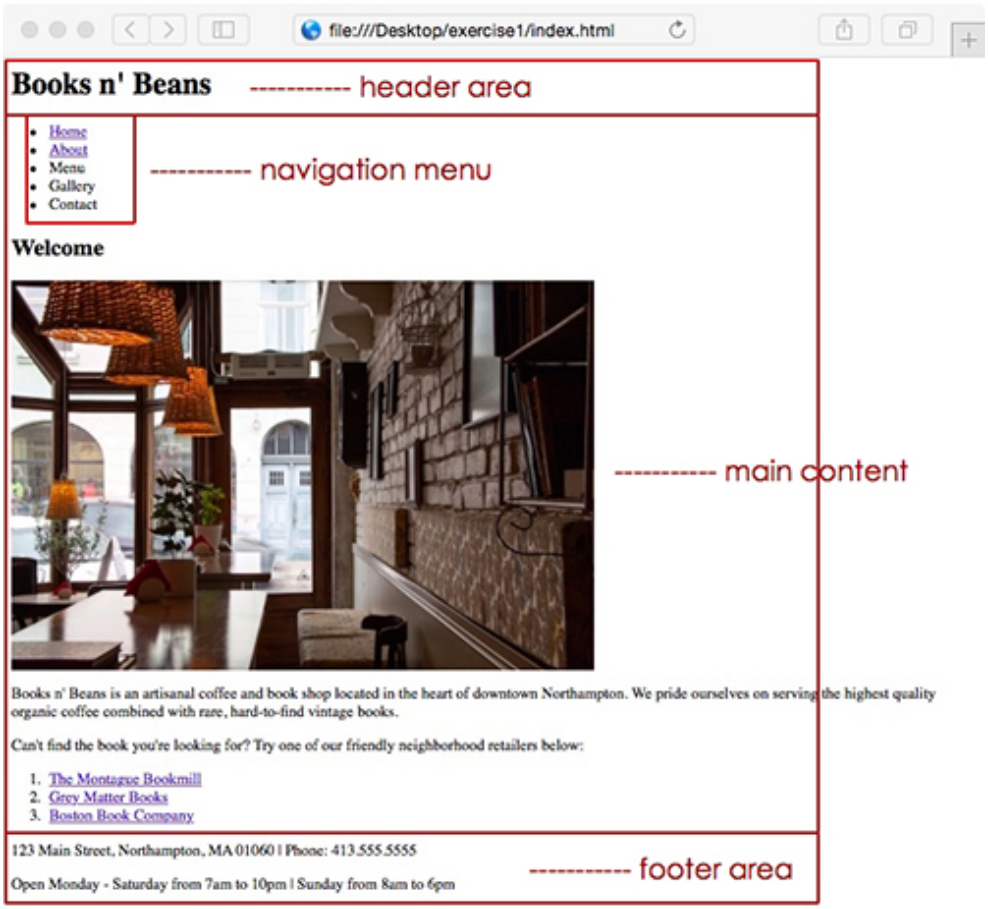
resubmission. Be especially careful with file naming and folder organization throughout this course.

1. Add CSS Selectors

Our Books n' Beans pages should already include clean, valid HTML5 markup. Now, it's time to add the appropriate CSS selectors so we can start styling the page.

If there were any errors in your Exercise 1 submission, now is a great time to correct them. Again, run your site through the markup validation service at https://html5.validator.nu/. You can enter the URL of each page in your site or upload a file, so feel free to check your work for errors along the way.

Before I start adding CSS selectors, I like to envision what the final product might look like. You don't have to have an exact image in mind, but it's important to think about where the header and footer will go, what the navigation bar might look like, and how the content is going to be laid out on the page. Here's an example:



I used Photoshop to make a few quick layout notes, but a notebook and pencil work just as well—sometimes even better—for sketching out your design!

So far, we're working with a pretty simple page layout: a header, a main content area, a navigation menu, and a footer. We can choose to add new selectors or style additional page elements as we go, but it's important to work from big to small when planning your page layout.

Ready for some style? Let's start by adding a header div and menu div to the Books n' Beans Home page.

```
<body>
<div id="header">
<h1>Books n' Beans</h1>
```

```
</div>

<div id="menu">
<ul>
<li>...</li>
<li>...</li>
</ul>
</div>
```

Here, I wrapped the Books n' Beans h1 heading in a div called "header" and wrapped the list-based navigation menu in a div called "menu."

Now let's designate the main content area of our site. I'd say the navigation menu should be part of this main content area, so begin the "main" div after the header div, but before the menu div.

```
<div id="header">
<h1>Books n' Beans</h1>
</div>

<div id="main">
<div id="menu">
<ul>
```

Close the main div right after the list of friendly neighborhood retailers, before the paragraph text containing the café address and phone number.

Now, wrap the address information and business hours at the bottom of the page in a div called "footer". The footer div should begin before the line of text that reads "123 Main Street…" and should end after the store hours, right before the closing body tag.

Finally, I'd like you to wrap all of the visible page content in a div called "container". Begin the container div right after the opening body tag, like so:

```
<body>

<div id="container">
<div id="header">
<h1>Books n' Beans</h1>
```

Close the container div right after the footer, right before the ending body tag. The order in which you open and close your divs is important, so keep track of the div nesting order. Before you go any further, now is a great time to make sure the Home page markup validates properly by visiting https://html5.validator.nu/.

Remember, we'll be using a single CSS style sheet to control the styles for every page on our site, but you'll have to add the proper HTML markup to each page so the styles display correctly. After adding CSS selectors to the Home page, you'll need to go through and add the proper markup to the About page as well. Just like the Home page, make sure the About page includes a:

- container div (#container)
- header div (#header)
- main div (#main)

- menu div (#menu)
- footer div (#footer)

The order of the markup on both pages should look something like this:

```
<body>
<div id="container">
<div id="header">
</div>

<div id="main">
<div id="menu">
</div>

<div id="footer">
</div>
</div>
```

Some of the text on the About page differs from the Home page text, but the div IDs should begin and end in the same order. We'll be carrying this format throughout our entire site, so if you're ever unsure, use the markup on the Home page as a guide.

2. Connect an External Style Sheet

All set? Create a link to an external style sheet in the head of each page, like so:

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>Books n' Beans</title>
<link rel="stylesheet" type="text/css" href="css/style.css"/>
</head>
```

Now create a style.css document in a local folder called "css" and make some basic styling decisions. (Remember to match the name of your CSS document with the name in the page head.)

3. Choose a Color Scheme

Take a moment to think about the overall visual design of the Books n' Beans Web site. The colors and font styles you use will communicate a certain feel to the site visitors, so consider which styles feel appropriate as you work on the design.

Colors have psychological associations, so it's important to choose them carefully. Below is a table of colors and positive and negative traits associated with them:

Color	Positive Traits	Negative Traits
Red	Power, hunger, impulsiveness	Anger, forcefulness, impatience,

		intimidation, conquest, violence
Yellow	Intelligence, joy, organization	Criticism, laziness, or cynicism
Blue	Tranquility, love, acceptance, patience, understanding	Fear, coldness, passivity, and depression
Orange	Steadfastness, courage, confidence, friendliness, cheerfulness	Ignorance, inferiority, sluggishness, and superiority
Green	Hope, growth, good health, freshness, soothing, sharing	Greed, constriction, guilt, jealousy and disorder

It's important to note that different cultures associate different meanings to colors. The chart above reflects more of a North American attitude towards color. When designing for different areas of the world or working on an international project, the cultural associations may vary.

If you're not comfortable developing a color scheme based on color harmony, you can "cheat" by visiting sites that are devoted to choosing color schemes such as colourlovers.com. The Colourlovers site provides millions of color schemes categorized in different ways, and you can grab the 6-digit hex values for each color.

Going for a cozy antique store feel? Consider muted colors combined with "bookish" serif type styles.

Cozy, vintage palette inspiration:

- colourlovers.com - 2 Kool for Skool
- colourlovers.com - Turismo Antique
- colourlovers.com - Writer
- colourlovers.com - Thumbelina

Looking for a slick, modern café feel? Consider a crisp, minimal color palette combined with san serif body text.

Clean, modern palette inspiration:

- colourlovers.com - Machu Picchu
- colourlovers.com - Walking Away
- colourlovers.com - w o r d l e s s .

- colourlovers.com - Gamebookers

Remember, your site won't look any different until you add CSS rules and selectors to your style sheet. So, now that we're already thinking about color, why don't we start there?

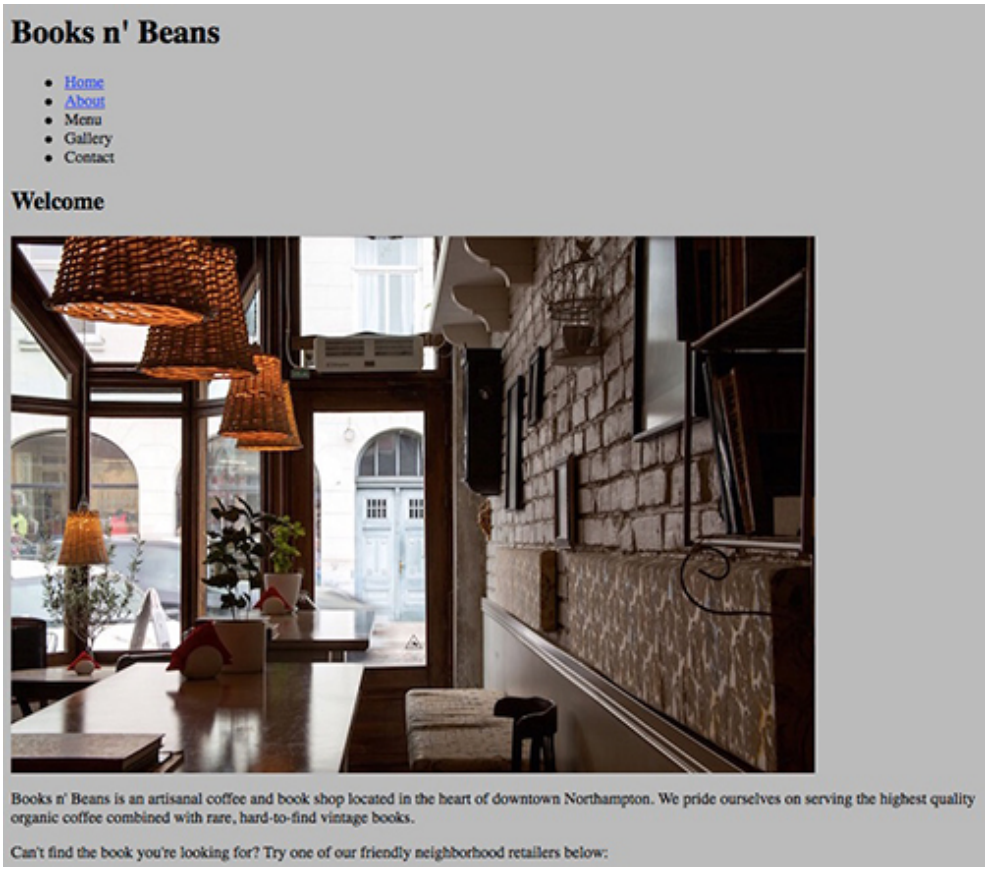
4. Choose Background Colors

After you've made some basic color decisions, we'll start by adding a background color to the body of the site. Copy/paste the code snippet below into the first line of your style.css file.

```
body {
    background-color: #bbb;
}
```

In the snippet above I've used the hex code value #bbb, a pale gray, for the background of the site body. I've included the color for example purposes only, so make sure to select your own custom background color before moving on.

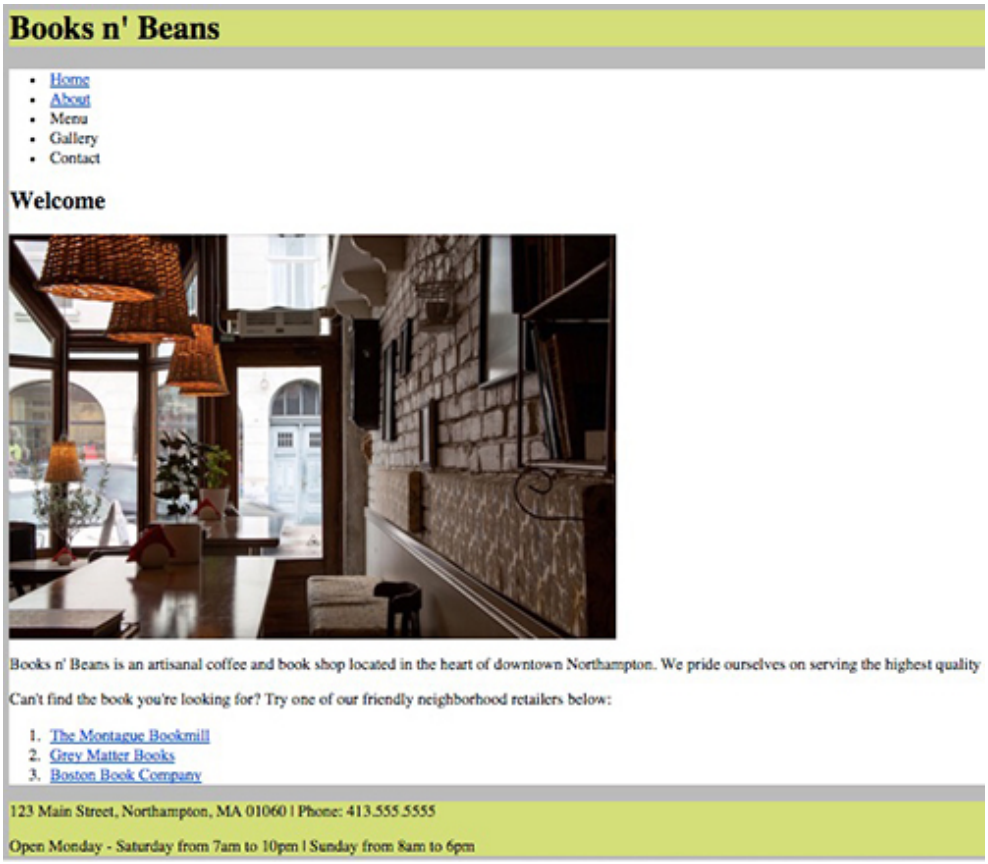
Here's what my Books n' Beans Home page looks like now:



Next, I'd like you to add a background color to the header div, main div, and footer div of your site. Remember, if you've added the correct markup to your pages, the changes you make to your style sheet should be reflected on both the Home and About page. Not seeing any CSS styles on the About page?

Check to make sure you've added a link to the external style sheet in the head of the page!

Here's a preview of the Home page with background colors assigned to #header, #main, and #footer:



Okay, so it's not beautiful, but it's getting there! A few CSS pro tips before moving on:

If you're using the same value for multiple CSS selectors, there's no need to list each selector separately. Rather than adding each selector to your style sheet like this:

```
#header {
        background-color: #660066;
}

#footer {
        background-color: #660066;
}
```

You can combine multiple selectors in a list, like this:

```
#header, #footer {
        background-color: #660066;
}
```

You can also shorthand hex color values that use repeating patterns. This will help decrease the size of your style sheet, and it's a lot faster than typing out all six characters! Instead of typing out #660066, the purple header and footer color shown in the previous example, you can simply type #606. Other shorthand examples include:

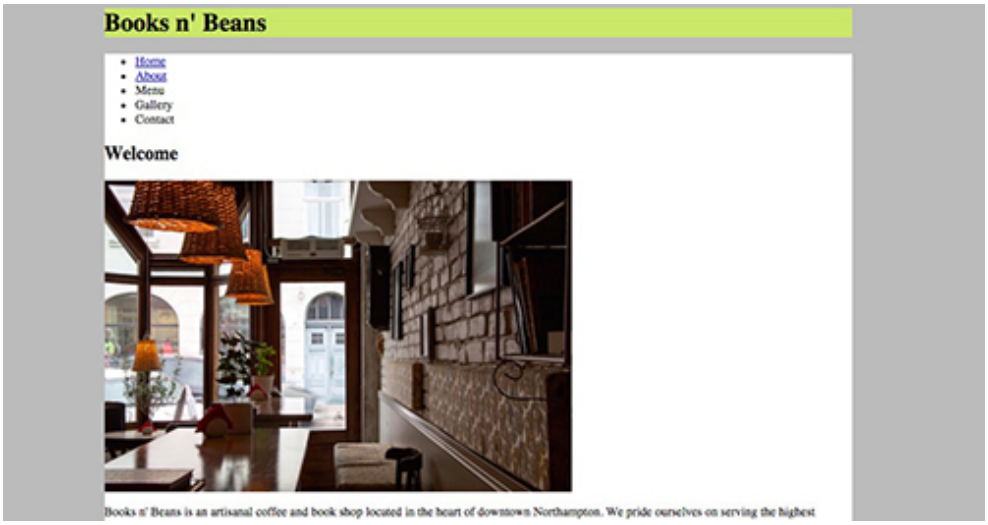
#ffffff - #fff (white)
#000000 - #000 (black)
#669900 - #690 (lime green)
#ff0066 - #f06 (hot pink)

And so on…

5. Restrict the Container Width

Before we start customizing font styles, I'd like you to add one helpful CSS rule that will make your page easier to read. At the beginning of the exercise, I asked you to wrap all of the page content in a div called "container." Now, let's see what happens when we assign a width and a centered margin to that div. Copy/paste the following CSS rule into your style sheet:

```
#container {
        width: 960px;
        margin: 0 auto;
    background-color: #fff;
}
```



The result is starting to look much closer to a traditional Web page layout. Assigning a width of "960px" to the container div effectively restricted the width of everything inside the container—which includes the header, menu, main div, and footer. Assigning a margin of "0: auto" centered the div in the middle of the page. Now, no matter the size of the browser window, the container will always appear centered.

(We'll be learning all about widths, heights, margins, and padding in the next lesson, so you don't need to memorize the above style rules. For now, you can just appreciate the visual effect it has on the site design!)

6. Customize Font-family, Font-size, and Color

It's time to let your heart run wild with CSS font styles! First, I'd like you to choose font-family styles, font sizes, and font colors for the headings, paragraphs, and lists. It's okay to use black text if it works with your site design as long as you don't rely on the default browser styles. If you'd like to use black paragraph text, add the color value #000 to the p selector in your style sheet. When choosing font families, remember to use complete font stacks to ensure cross-browser compatibility.

Include custom styles for each of the following selectors in your stylesheet:

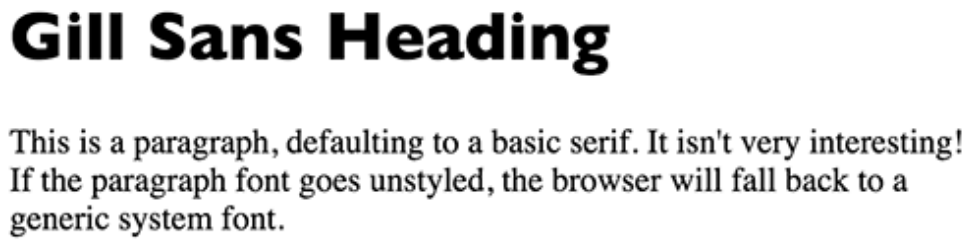
h1
h2
p
ol
ul

Example:

```
h2 {
        font-family: Verdana, Geneva, sans-serif;
        font-size: 24px;
        color: #555;
}
```

Reminder: To change the background color of an element or div, use the CSS "background-color" property. To change the font color, use the "color" property.

If the font family isn't declared for all elements on your site, the browser will default back to a basic system font—usually a generic "times" or "serif". This can lead to websites with lovely headings and navigation menus, but lackluster body text!



Rather than apply a custom font to every single element, you can cover all bases by declaring a font for the entire site body. This will make it so no text on your site goes unstyled!

```
body {
font-family: Tahoma, Verdana, Segoe, sans-serif;
font-size: 14px;
}
```

The body font-family can be overwritten for individual elements, like links, list items, or headings, but it's an important global style for your website!

Adjust letter-spacing and line-height as needed. You don't need to demonstrate the use of every text property you learned about in Lecture Two, but I'd like to see at least two out of the six properties used from this list:

- letter-spacing

- line-height
- font-weight
- font-variant
- text-transform
- text-align

Check out the Books n' Beans header, menu, and Welcome text pictured above. The page headings have been changed to a deep purple and I've added a letter-spacing of 2px to give the headings an airy feel, and I've used the font-variant property "small-caps" for a professional, distinguished look.

My lists and paragraph text are using a dark gray font color with a size of 1.1em and a custom line-height of 1.3em.

I don't expect to see these exact styles on your site, but I do want to see you transform any default browser font styles into something more creative!

7. Style the Anchor Links and Lists

Finally, I'd like you to add some custom styles to the page links and lists. Let's start by removing the default underline that appears under anchor links. The underline used to be an important signifier that said, "Hey! This is a clickable link!" Now, with the power of CSS, we can use other visual signifiers like color, font weight, and link state changes.

To remove the underlines from all anchor links, add the following rule to your style sheet:

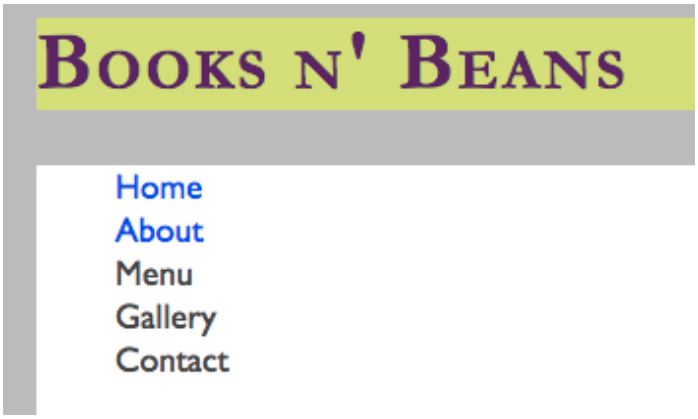
```
a {
        text-decoration: none;
}
```

That rule will remove every single link underline from the site. If you wanted to, say, remove the underlines from the navigation menu links but leave the underlines in the ordered list links at the bottom of the page, you'd specify like this:


```
ul a {
        text-decoration: none;
}
```

While we're at it, let's remove the bullet points from the navigation menu at the top of the page!

```
#menu ul {
        list-style-type: none;
}
```



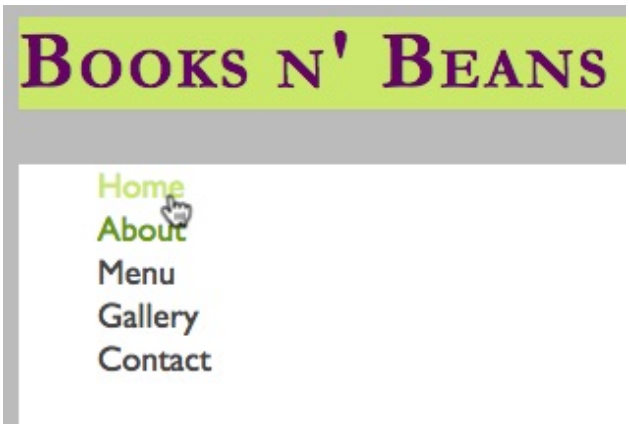
Removing bullets from the navigation list

Hey, looking good! But we're still looking at those default blue anchor links. Start by changing the color to something that's a better match for your page.

```
a {
    color: #690;
}
```

Because we've removed the underlines, it's a good idea to give visitors another visual indicator to let them know these are clickable links. Why not add a link hover pseudo-class?

```
a:hover {
    color: #cbe86b;
}
```

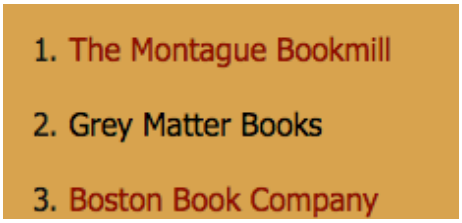


There are four pseudo-classes for the anchor element, but hover is the only one that's required for this exercise:

- `a:link` (unvisited link)
- `a:visited` (visited link)
- `a:hover` (moused-over link)
- `a:active` (selected link)

Remember that link pseudo-classes must appear in a certain sequence in order to work properly. `a:hover` must be listed after `a:link` and `a:visited`, and `a:active` must be listed after `a:hover`.

Though link pseudo-classes must appear in a certain order, you're not required to assign a custom color or style to every link state. In fact, adding a separate visited link color can be confusing to site visitors!



In this design, a dark brown color has been applied to visited links. If previously visited the Grey Matter Books site, the link will appear dark brown, instead of red. There's only three links here, so there's really no reason to apply a visited link color. The new color disrupts the design, and may confuse visitors, who don't know what the color change represents.

Styling visited links to display in a different color is really only useful on large content aggregator sites like Reddit, where a visitor is scrolling through many different links at once.

Visited links shown in brown, unvisited shown in blue. It's easy to get lost scrolling through hundreds of cute animal pictures, so these different link styles are helpful!

Be sure to style links in a different color from the rest of your page text! A visitor should never have to go on a scavenger hunt to find your page links, and obscuring that information can lead to an accessibility nightmare!

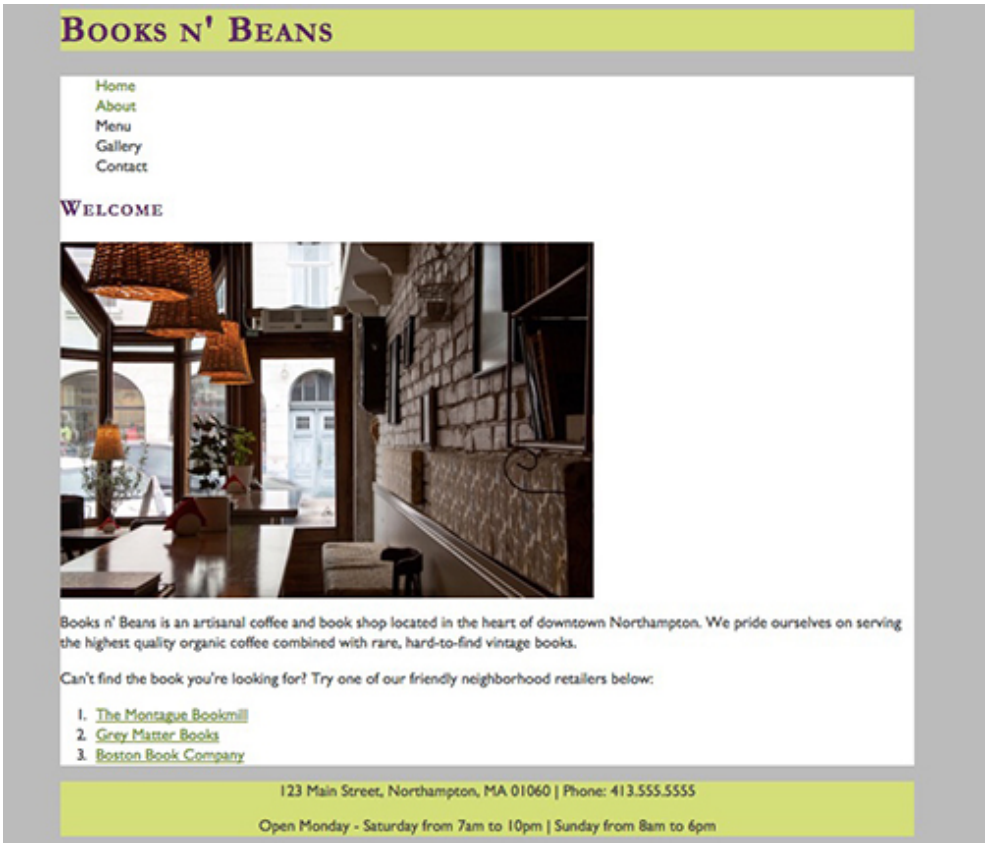
There's a link in there, but it's hidden among the rest of the text!

Much better! I've added a bold font-weight as well, just to make it extra clear.

Resist the temptation to change the vertical navigation menu to a horizontal menu in this exercise. We'll be working with horizontal menus later in the course, and it's important to keep this menu vertical in preparation for creating floated sidebars in exercise 3. The current navigation menu may not look beautiful, but we'll be getting there soon!

8. Upload and Validate Your Work

Save your CSS style sheet and refresh your HTML page in the browser to view the final changes. Make sure to view all pages to make sure your CSS rules are working properly across the site.



Before submitting your site, double check to make sure your site includes:

- The correct markup added to both the Home and About page (#container, #header, #main, #menu, and #footer)

- CSS styles applied to element selectors (headings, paragraph text, lists, links, etc)
- Valid font stacks with font sizes

If you're getting validation errors, here are some troubleshooting questions:

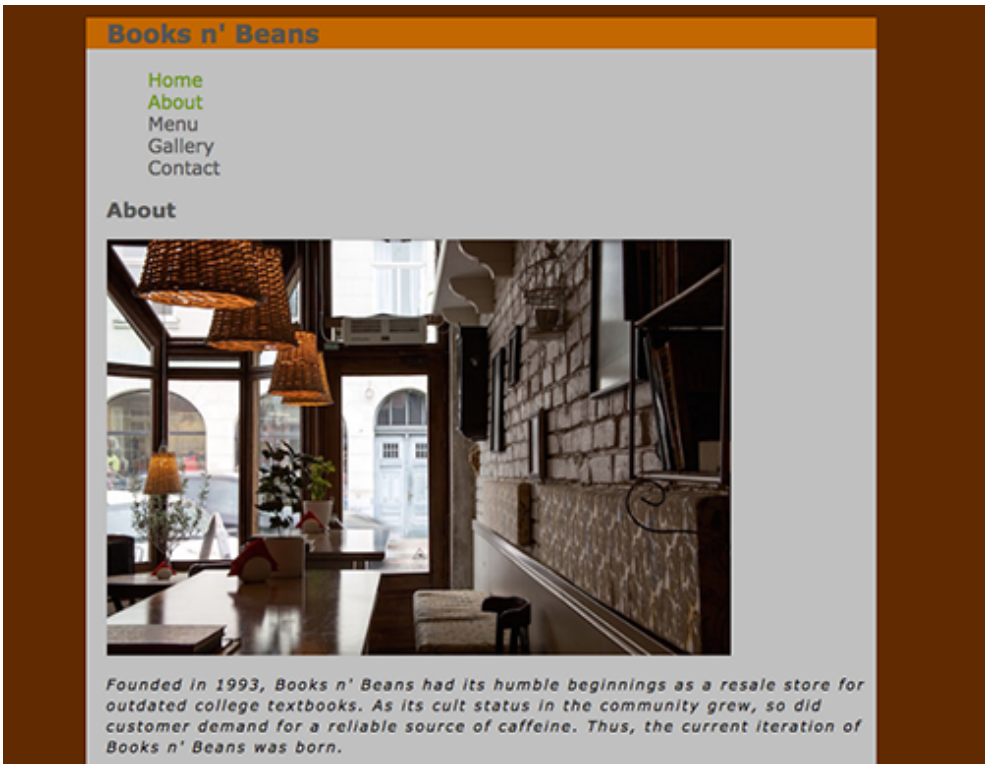
- Have you included the correct HTML5 doctype at the very top of the page? Are there any spelling or formatting errors in the doctype?
- Have you included quotations in the right places? Make sure div names, anchor links, images, and alt text are wrapped in the proper quotations. Double-check to make sure that you're using code-friendly quotes, and not "smart quotes" or "curly quotes."
- Have you closed any open divs? Have you opened and closed divs in the right order?
- Have you checked divs for spelling errors? Your stylesheet won't be able to communicate with your HTML document if div names are spelled incorrectly.

Describe Your Experience

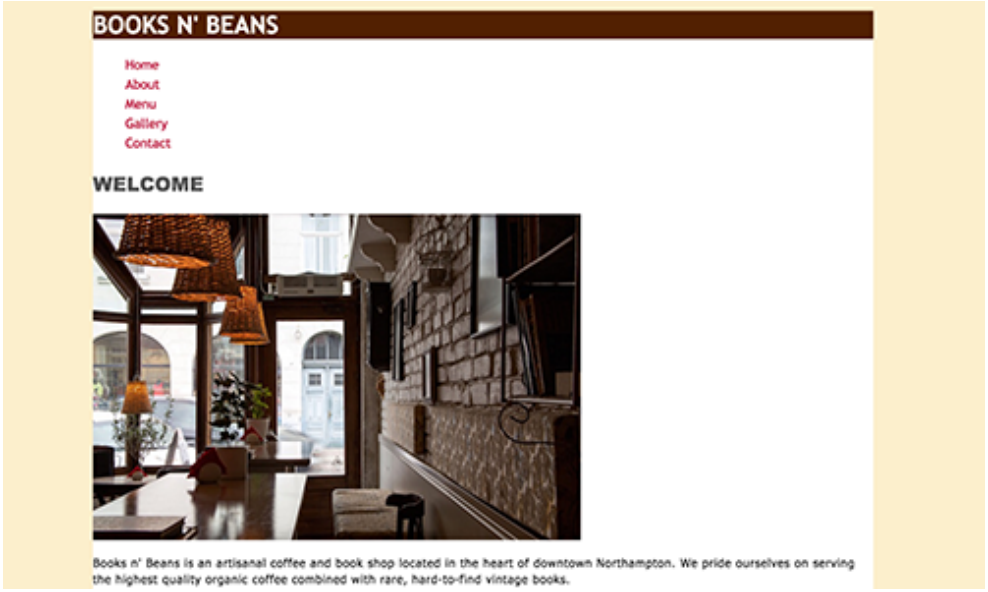
When learning code, it's important to reflect on what worked and what didn't. You can learn as much from your mistakes as your successes. When you post your work, include your answers to the following questions :

1. Which CSS style do you feel had the biggest impact on improving the site design?
2. What changes do you hope to make to further improve this design? What do you think is still missing?
3. Which parts about writing CSS were clear to you, and which parts (if any) are still confusing?

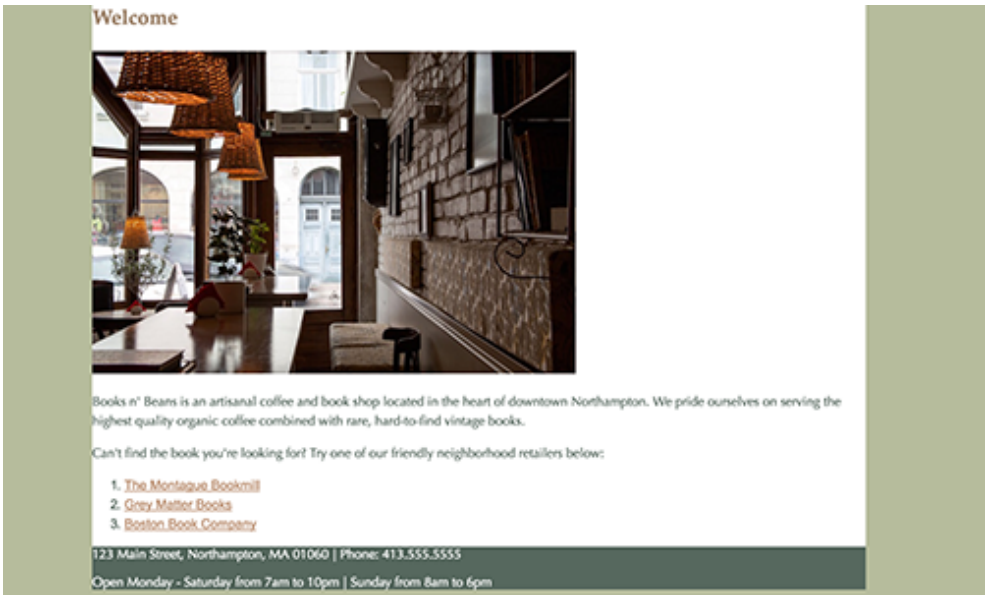
Our Books n' Beans site is well on its way, and we'll be expanding on its layout design in the next exercise. For now, I look forward to seeing your work!



Student Ishrat Jahan used a coffee-colored background combined with a green accent color for navigation links. The san-serif body text is italicized, with customized letter-spacing and line-height.



Student Binh Bau combined two browser-friendly san-serif fonts, Trebuchet MS and Tahoma, in this Books n' Beans design. The color palette is warm without feeling heavy, and the uppercase headings (styled with text-transform CSS) add a nice touch.



Student Roger Young combined muted greens to create a largely monochromatic color palette. Note the serif "Welcome" heading (Palatino) combined with the san-serif body text (Optima).

Grading Criteria:

What your instructor expects you to do:

☐ Demonstrate the ability to attach CSS styles to a site using an external stylesheet.

☐ Show the ability to develop an attractive, readable visual design for your site using color and font styles.

☐ Show the ability to correctly implement a range of CSS rules to style a site's color, typography, and links.

☐ Demonstrate the ability to create clean, accurate, and standards compliant code in your HTML and CSS documents.

☐ Demonstrate the ability to FTP a site to the Web, testing and troubleshooting each page and fixing any issues.

How to Post:

Once you're done, go to the Dropbox for this exercise and post your URL for this assignment along with your written comments.

If you have a question before sending your completed exercise for grading, be sure to contact your instructor via the People area.

I look forward to seeing your work!

sessionscollege
FOR PROFESSIONAL DESIGN

Lecture 3

Page Layout with CSS

Laying out a page in CSS

CSS has transformed the way that designers create Web page layouts. In this lesson, we'll learn how to take our page layouts to the next level with box model formatting and CSS positioning techniques.

We'll begin with an introduction to the CSS box model, then we'll progress to techniques like relative and absolute positioning and floats, and one and two-column layouts.

This is a tricky lesson, but once you've grasped these concepts, you'll have learned techniques used on most Web sites designed today.

Learning Objectives

In this lecture, you can expect to:

- Learn how to apply the box model to style elements using borders, padding, and margins.
- Learn how to apply relative positioning, absolute positioning, and floats to layout page sections.
- Learn how to create fixed and fluid one and two-column layouts.
- Learn some basic troubleshooting tips for CSS positioning and floats.

The CSS Box Model

Thinking Inside and Outside the Box

The CSS box model is one of my favorite CSS principles, but it takes time and practice to understand.

The first step in understanding the box model is to imagine every block-level HTML element as a box. **Block-level elements** are elements that start on a new line and extend the entire width of their container. Examples of block-level elements include divs, h1 through h6 headings, lists, and paragraph tags.

The opposite of a block-level element is an **inline element**. Inline elements don't start on a new line, and take up only as much space as needed. Examples of inline elements include spans, anchor links, and image tags.

note

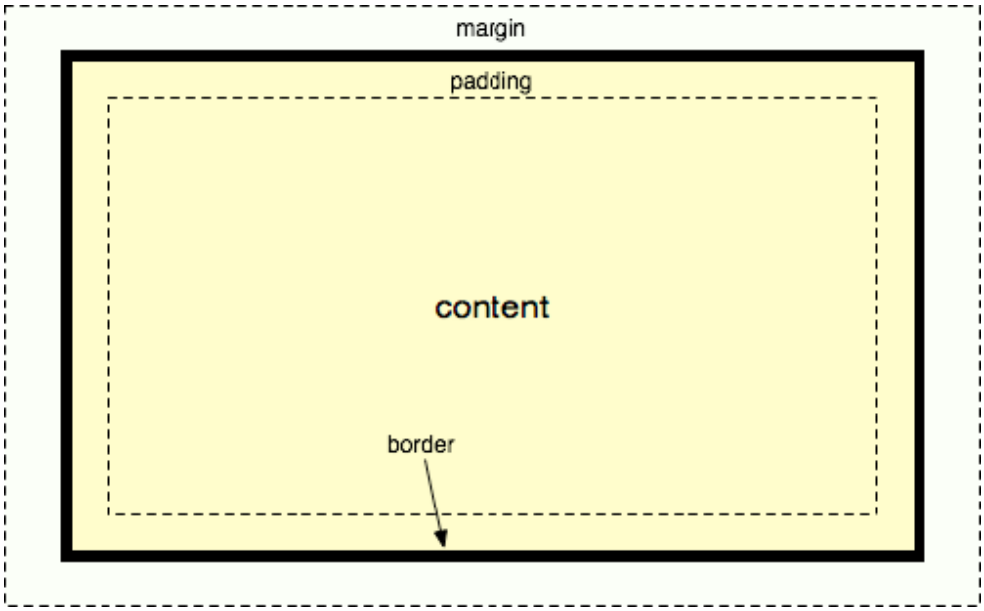
Every block-level element in HTML can be imagined as a box.

On a Web page, every block-level element can have adjustable margins, padding, and a border. These parts may not be always visible, but they're always there.

Here's a diagram showing how the CSS box model works:

note

Block level elements can have padding, a border, and a margin.



We can break down the CSS box model into four parts: the content, border, padding, and margins.

The **content** inside the box could include headings, paragraph text, images, links, lists, and so on. Surrounding the box content is a **border** (pictured above in black). The border's thickness, style, and color can be adjusted using CSS (and can be invisible, as well).

Between the content in the box and the border is a space created by the box's **padding**. Padding allows you to leave a cushion of space between the edge of a box and the content inside the box. Without padding, text and images will bump up against the edges of their container.

Finally, between the element's border and any surrounding elements on the page is the **margin**. The margin is what provides some "breathing room" between box elements that are next to one another.

Sound confusing? Let's use our alice.html page from Lesson One to get to the bottom of this rabbit hole.

First, open up your alice.html page and set up some tags to divide the page into sections. Let's create wrap the entire visible page in a container and then wrap the following areas in divs: image, contents, and text.

```
<head>
    <title>Alice's Adventures in Wonderland</title>
    <link rel="stylesheet" type="text/css" href="css/style.css" />
</head>
<body>
<div id="container">
    <h1>Alice's Adventures in Wonderland</h1>
    <h2>By Lewis Carroll</h2>
<div id="image">
    <a href="https://www.gutenberg.org/">
<img src="images/alice.gif" alt="Alice and rabbit in hallway">
</a>
</div>
<div id="content">
<h3>Contents</h3>
```

Adding divs to the Alice page

note

Designers use the CSS box model to style all kinds of block-level elements.

Now open up your stylesheet and let's see how to style each block element using the box model.

tip!

You can control the width, color, or style of a border, or set it to none.

We'll begin by styling the container div by assigning a fixed width and a centered margin to that div. But we will also add a 5 pixel solid red border to the container. Copy/paste the following CSS rule into your style sheet:

```
#container {
width: 960px;
margin: 0 auto;
border: 5px solid red;
}
```

Next, see what happens when you add different colored borders (with margins and padding) to style each of the CSS selectors in your document.

```
#content {
border: 5px solid orange;
margin: 30px;
padding: 30px;
}
```

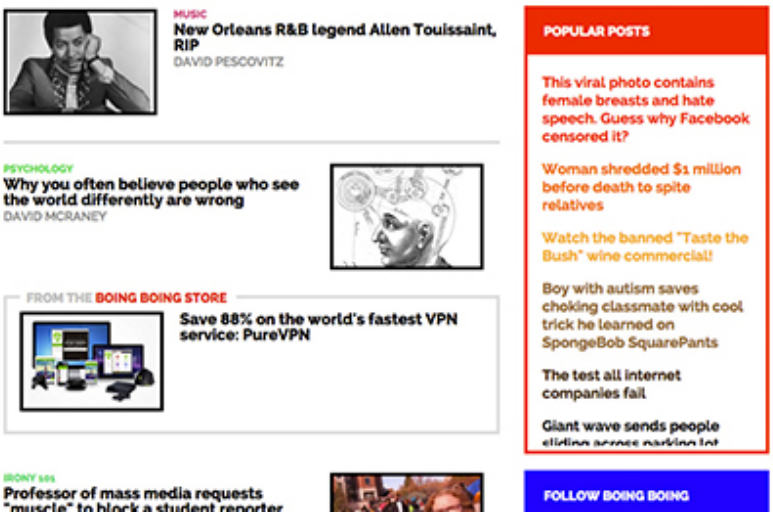
Try this approach with your <p> tags too. You'll see that each block-level element on the page can be styled as its own box, with its own border, padding, and margins.

Once you get a grasp of the CSS box model, you'll start to notice it in action all around the Web. In the boingboing blog (below), the designers use padding on the left and right to create a generous amount of whitespace around the site content.

note

Width, color, and style properties can be

combined in one border property.



Padding, margins, and borders, oh my!

They also use borders around most of the content boxes and images, and separate block-level elements by ample margins.

Applying Styles to Borders

CSS provides many options for styling the borders around elements. Keep your CSS document open and follow along, applying each style to your alice.html page.

Let's take a look at the different properties related to border styling:

note

Padding refers to space inside the box; margin refers to space outside.

1. Using border-width
As its name suggests, the border-width property changes the width of the border. You can assign the values *thin*, *medium*, or *thick*, or you can set the width with a length value (such as 2px). I recommend using a pixel value, as it allows for more fine-grained control over your design.

2. Using border-color
The border-color property sets the color of the border. The values of the color can be specified in plain English (red, blue, and so on), or as a hexadecimal value.

3. Using border-style
The **border-style** property changes the style of the border. Border-style takes one of the following values:

- solid
- double
- groove
- dotted
- dashed
- inset
- outset
- ridge

- hidden (or none)

tip!

Think about how much breathing room you need around block-level content.

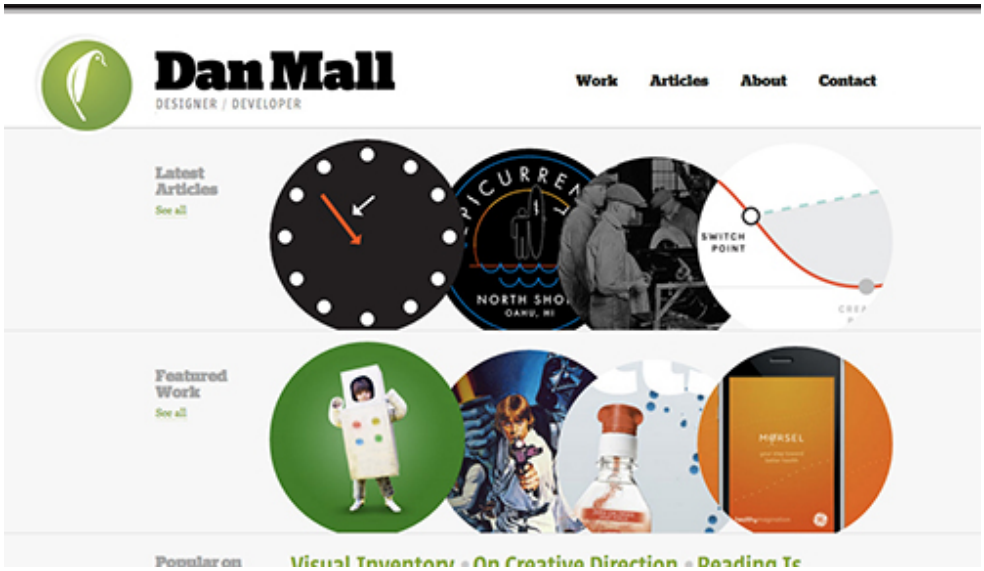
Want to save some space in your style sheet? You can combine border width, color, and style properties into one with the **border** property:

```
p {
    border: 3px dotted red;
}
```

With any of these properties, you can also pick out just one side of the element, by specifying top, bottom, left, or right:

```
p {
    border-left: 2px dashed blue;
}
```

Borders applied to just one or two sides of an element are useful for separating content into readable sections, or for adding a subtle design flair. Here's a great real world example:



Designer Dan Mall divides content with borders applied to the top of each section. He's included a thick black border at the top of the site as a design flourish.

note

Background colors and images can also be used to style block-level elements.

While Dan Mall's design includes generous margins and padding, the boxes on our Alice page need a bit of padding to separate the borders from the content. That's up next!

Margins and Padding

Looking for some breathing room? That's good, because white space is the designer's best friend.

You can adjust the margins and padding of an element using the **margin** and **padding** properties. They both take as their values lengths, percentages, or auto. Like the border property, you can specify which side of the element you are setting the margin or padding for.

You should have a div ID wrapped around the content on your main text area of your Alice page: `<div id="content"></div>`. Let's explore how to apply margins and padding. First, set the margin and padding at a comfortable 30 pixels on all sides.

note

A is used to define a block-level section, while a defines an inline section.

```
#content {
    border: 5px dotted red;
    margin: 30px;
    padding: 30px;
}
```

Scale the margins and padding values up and down until you get comfortable with how they work.

Next, let's isolate and try to control the specific padding on each side. (We can do the same for margins.)

```
#content {
    border: 5px dotted red;
    padding-top: 20px;
    padding-right: 40px;
    padding-bottom: 20px;
    padding-left: 10px;
    margin: 30px;
}
```

Or, if all four sides have different values, we can set the top, right, bottom, and left padding as follows:

```
#content {
    padding: 20px 40px 20px 10px;
}
```

If the top/bottom and left/right values are the same, we can condense our code. Using two values sets the top and bottom margin to the first value, and the left and right margin to the second value:

```
#content {
    padding: 20px 40px;
}
```

note

Used with unique ID attributes, divs are ideal for breaking up your document into meaningful sections (header, footer, and so on).

These principles apply to both margin and padding properties.

Background Colors

In the last lesson, we learned how to apply a background color to an element. This gives you interesting options for any block-level element. Experiment with adding background colors to our Alice page.

```
#content {
    padding: 20px 40px;
    background-color: #e8f7f8;
}
```

Background Images

To add a background image to an element, use the **background-image** property. For instance, you could have your alice.gif image display in the text div area like so:

```
#content {
    padding: 20px 40px;
```

note

When block-level elements are stacked, their vertical margins collapse, applying just the largest margin.

```
    background-color: #e8f7f8;
    background-image: url(../images/alice.gif);
}
```

(Note: To create a relative link from the CSS to the image, the background image link address uses a new relative path (../) that specifies "go back to the root folder" and open the images folder.)

By default you will notice that the browser tiles the image behind the content. I can control the tiling of the image using the **background-repeat** property:

```
#content {
    padding: 20px 40px;
    background-color: #e8f7f8;
    background-image: url(../images/alice.gif);
     background-repeat: repeat-y;
}
```

To tile an image just horizontally, or just vertically, use the values repeat-x (tile horizontally) or repeat-y (tile vertically). Handy for non-intrusive background patterns!

Or, I can set that property to no-repeat, which means the image will not be tiled and place the image off to the right in the empty area.

```
#content {
    padding: 20px 300px 80px 40px;
    background-color: #e8f7f8;
    background-image: url(../images/alice.gif);
    background-repeat: no-repeat;
}
```

We can more precisely position a background image by using the background-position property. This property takes as a value lengths, percentages, or keywords (such as "top right" and "bottom left"):

```
#content {
    padding: 20px 300px 80px 40px;
    background-color: #e8f7f8;
    background-image: url(../images/alice.gif);
    background-repeat: no-repeat;
     background-position: 700px 50px;
}
```

note

Use the position and float properties to move an object out of normal flow.

This positions the image 700 pixels over, and 50 pixels down, from the upper-left hand corner of the box. Alternatively, I could use "top right" for quick and easy placement:

```
#content {
    padding: 20px 300px 80px 40px;
    background-color: #e8f7f8;
    background-image: url(../images/alice.gif);
    background-repeat: no-repeat;
    background-position: top right;
}
```

As you can see, the CSS box model offers consistent styling of block level elements. It also plays a role in page layout, our next topic. Let's revisit some of the selectors we encountered in Lesson Two.

note

Relative positioning positions an object relative to its location in normal flow.

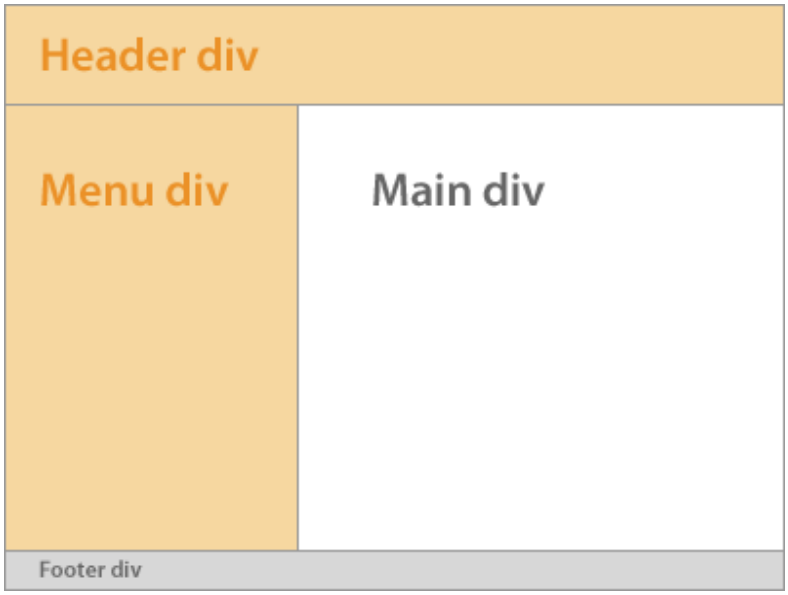
The Basics of CSS Positioning

Using Div Tags

In the last lecture, we learned that `<div>` defines a block-level section, while `<span>` defines an inline area.

The `<div>` tag, by itself, works somewhat like a paragraph tag in that it puts a line break before and after the content that it wraps. It differs semantically from a paragraph, however, as it's intended to mark out sections of an HTML document.

These sections can be as broad as the general content of the Web page, or as narrow as the copyright information in a footer. Used in conjunction with unique ID attributes, `<div>` elements are ideal for breaking up your document into meaningful sections (header, footer, and so on), which can then be styled and positioned using CSS.



Standard layouts like this one can be created by positioning elements. Unusual layouts, too!

Setting Positioning Properties

note

Absolute positioning positions an object relative to the edges of its containing element.

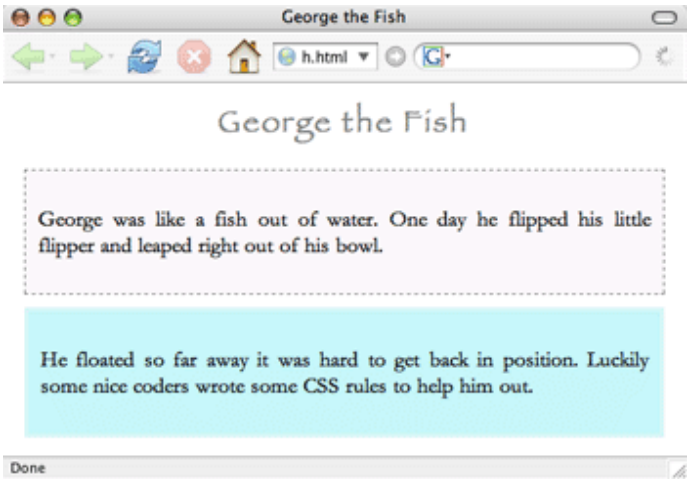
Before we go into the various properties used to position elements on a page, it's helpful to know how elements are positioned in the normal document flow: the default method the browser uses to layout elements if other position properties aren't specified.

Download the Lesson 3 zipped folder and open the fish folder inside. What we have is an HTML document with two specified divs. In the CSS document, you'll note the following properties are assigned `margin: 10px;` and `padding: 10px;`, as well as some basic borders and text styles.

In normal flow, block-level elements (like `<div>`) flow vertically from top to bottom, in the order that they appear in the markup:

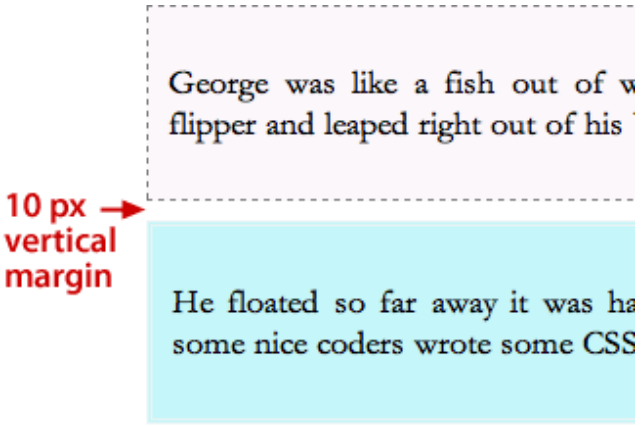
important!

Absolute positioning can cause problems in responsive layouts.



Two div elements in normal flow

Let's review the difference between inline elements and block-level elements. Inline elements like `<span>` or `<em>` flow from left to right. When block-level elements are stacked vertically, their vertical margins collapse—instead of adding the bottom margin of one element to the top margin of the element below it (10 + 10 = 20 pixels) the browser will just apply the largest margin (in this case, 10 pixels).



You can move an element out of normal flow using the **position** and **float** properties.

These two properties are the key to moving around content within your Web pages. Designers use boxes to position all sorts of content within a Web layout. The main column of text on a Web page could have any number of smaller content items associated with it, each precisely or relatively positioned (and styled) using divs.

note

Content that follows a floated element wraps around it.

Let's explore the position and float properties.

The Position and Float Properties

Use the position property to—surprise!—precisely position block-level elements on a page. You can position elements relatively or absolutely.

Relative Positioning

We'll start with **relative** positioning. Take a look at the markup in the HTML document:

```
<div id="fish">
<p>George was like a fish out of water.
One day he flipped his little flipper
and leaped right out of his bowl.</p>
</div>
<div id="bowl">
<p>He floated so far away it was hard to get back in position.
Luckily some nice coders wrote some CSS rules to help him out.
</p>
</div>
```

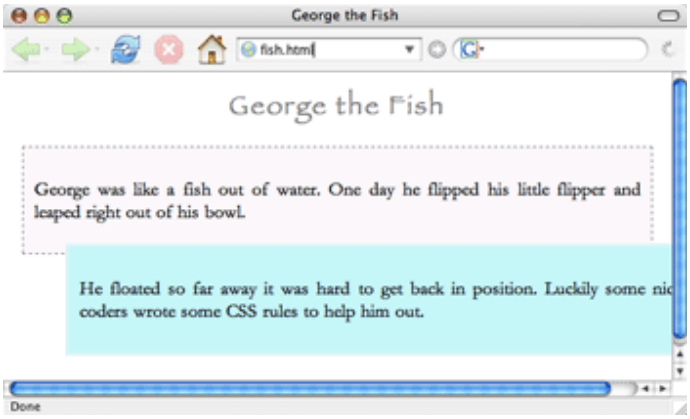
Here we have two divs: "fish" and "bowl". Suppose we apply the following CSS rules:

```
#bowl {
    border: 3px double #e9f4f5;
    position: relative;
    top: -20px;
    left: 40px;
    background-color: #c5f6f9;
    text-align: justify;
    margin: 10px;
    padding: 10px;
}
```

What have we done here? We've set the position property of the "bowl" div as "relative." This means that the browser will initially place that div within the normal flow (right after the fish div).

The **top** and **left** properties specify where it will go from there. (You can also use **bottom** and **right** instead of top or left, though it's recommended that you pick one pair or the other.)

The rules will *offset* the bottom box -20 pixels from the top of its default position (moving it up) and 40 pixels from the left of its default position (moving it right) resulting in the following:



Relative positioning: -20 pixels from the top, 40 pixels from the left of default position

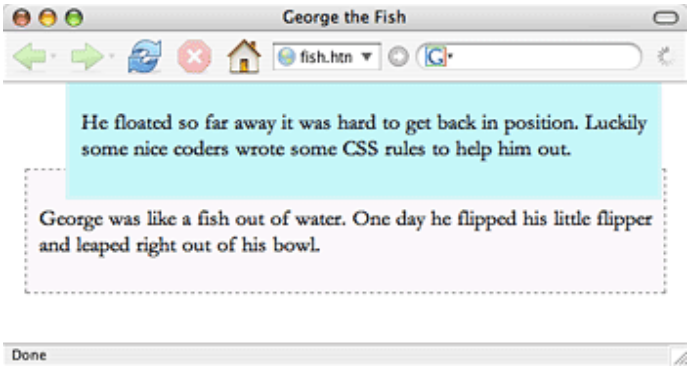
Before you continue, play around with the values to move the bowl div around the page! Notice that however you resize your browser, the location of the top left corner of the positioned box relative to the other box will not change.

Absolute Positioning

Let's now explore **absolute positioning**. We'll just use the same code, except we'll change the position value from "relative" to "absolute":

```
#bowl {
    border: 3px double #e9f4f5;
    position: absolute;
    top: -20px;
    left: 40px;
    background-color: #c5f6f9;
    text-align: justify;
    margin: 10px;
    padding: 10px;
}
```

Yikes, what happened? The header and top box "fish" stayed in the same place, but the bottom box "bowl" moved up to (and off) the top of the page, and it just won't budge:



Absolute positioning: -20 pixels from the top, 40 pixels from left of the container

Here's why. When a browser positions an element absolutely, it takes it completely *out of the normal flow*, and begins its positioning from the edges of its containing element (that is, the element it's inside).

You define the edges it is starting from by using the properties "top", "bottom", "left", or "right". And if you specify, say, "right: 50px", it will move 50 pixels from the right edge of the container. In this case, the containing element of *this* div is the browser window.

Why is that? Well, the containing element of an absolutely positioned element is its nearest positioned ancestor (that is, the nearest ancestor that has a position value of absolute, relative, or fixed). The direct ancestor of this div element is the body element, but it doesn't have a position value. So, the container element becomes the browser window instead: that is why the padding specified for the body element doesn't apply to this div (and why it is so big). A div could also have another div as its nearest positioned ancestor.

In our example, the div element is offset 20 pixels up from the top left corner of the browser screen and 40 pixels from the left of the screen. Thus, the top of the element is cut off from view, as it is pushed up inside of the top of the browser window.

Before you move on, try toggling between relative and absolute settings and notice what happens when you change the values.

So, to summarize:

1. A relatively positioned element is offset relative to its place in normal flow.

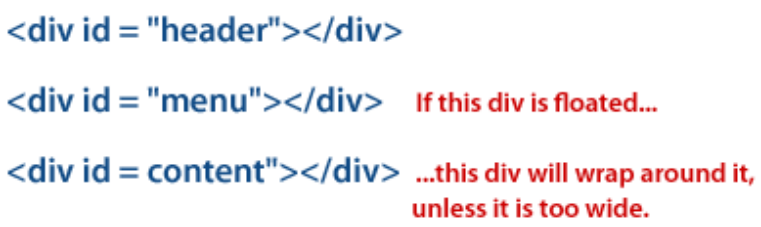
2. An absolutely positioned element is offset from the edges of its containing element.

Absolute positioning can be a lot of fun to work with, as it allows very precise control over the placement of elements on a page. This is great when working with static page layouts, but gets tricky when working with **responsive layouts**, whose elements are designed to smoothly and flexibly adapt to many different screen sizes.

We'll talk more about responsive design in a future lesson! In the meantime, let's plan on using absolute positioning for small design flourishes and avoid using it to position large block-level elements on a page.

Floats

Floated elements are positioned vertically within normal flow, but are horizontally pushed all the way to the left or the right of the containing element. Content that follows a floated element in the markup then wraps around it. If the content following the float is too big to fit next to the float (too wide, for example), it will appear below the float.



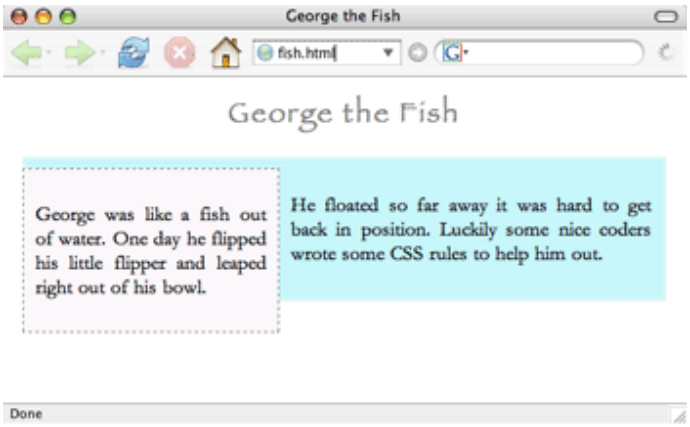
Content that follows a floated element wraps around it.

Let's try this out with our fish page. First, remove the positioning property from the "bowl" div to restore it to its original location.

Next, let's float the first div element:

```
#fish {
    width: 200px;
    float: left;
    border: 1px dashed gray;
    background-color: #fbf7fb;
    text-align: justify;
    margin: 10px;
    padding: 10px;
}
```

Now, our page looks like this:



Floating the fish div

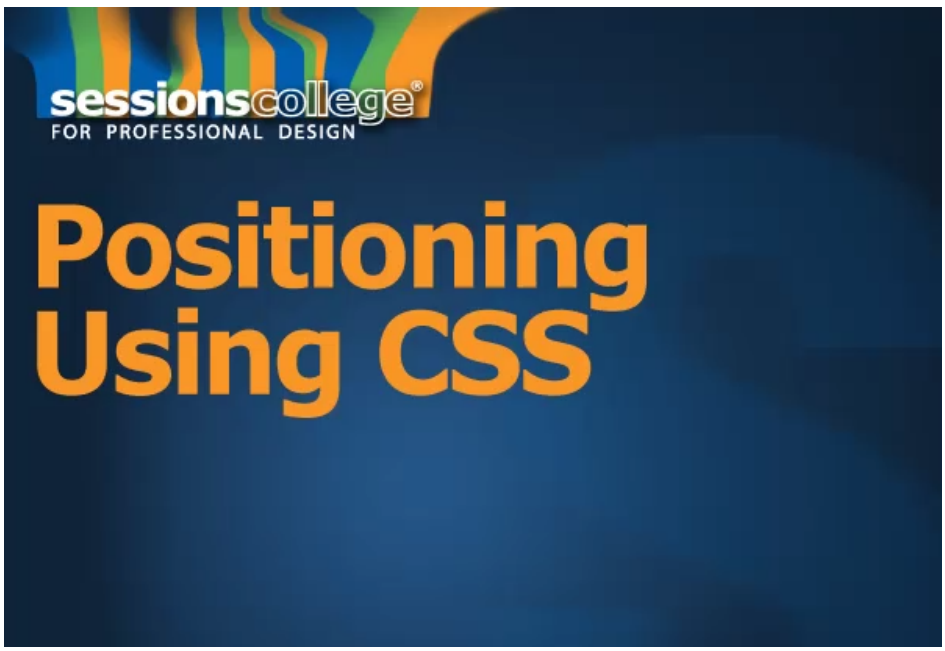
The "fish" div, now 200 pixels wide, is pushed to the left edge of the containing element (here, the body), and the content following it (inside the "bowl" div) now flows around it. Resize your browser to see how it works.

To float an element, you need to assign "left" or "right" for the **float** property, and then specify a width. All floats *must* have a width (though a width can be a relative unit, like a percentage). If you float an image, though, there's no need to specify a width in the CSS, as all images come with a width already!

Experiment with different property settings, applying them to each div, to see the results. For extra practice, add div tags to create a page element and apply positioning properties to the element. CSS positioning can be a little mind bending—and sometimes unpredictable!

Before we move on to our next topic, watch this video to see how absolute and relative positioning can be used with floats and box model styles to create a page layout:

VIDEO TUTORIAL: Positioning with CSS



0:00 / 3:28

Column-Based Layouts

note

Most Web sites today feature column based layouts.

Now that you've learned how to float elements on a page, let's look at how to create columns in your layouts. Most professional Web sites feature some version of the column-based layout. A more minimalist site may get away with a single column, but a robust site with more information will probably be split into two or more columns.

Let's look at a couple of professional examples.

forbetter.coffee is a single-page, single-column website. As the visitor scrolls, the coffee bean in the center follows them down the page.

theblacktruffle.com.au uses a two-column layout, with the sidebar navigation floated to the left and the main content to the right.

The main strategy for creating a column-based layout is to:

note

Column-based layouts are created by splitting up an HTML page into sections using div tags and the ID attribute, then styling and positioning the divs.

- Split up the HTML page into sections (header, menu, and so on) using div tags and the ID attribute.
- Size and position those divs using CSS.

Fixed and Fluid

There are two main kinds of layouts: fixed and fluid.

note

Column-based layouts can be fixed or fluid. Fixed layouts have fixed column widths, fluid layouts, columns that resize.

A **fixed layout** is one in which the columns are a static width; they remain the same size no matter how big the browser window is. Fixed columns are created by setting a width to the relevant div. If we want our "main" div to be 350 pixels wide, we would write the div ID like this:

```
#main {
    width: 350px;
}
```

A **fluid layout** is one in which the columns stretch or shrink as the browser window is resized. They are created by assigning a percentage to the relevant div. Below, we're telling the browser that the "main" div is 45% of the *width* of the containing element:

```
#main {
    width: 45%
}
```

One advantage of a fluid layout is that it takes advantage of all the space on the screen; you aren't left with large swaths of empty space. And, all of your content fits inside the browser window; you don't have to worry about text being cut off, or forcing your users to scroll horizontally in order to see the rest of the page.

A disadvantage of a fluid layout is that you ultimately have less control over the design when you can't predict the exact size of the elements on the screen. And, if text is squeezed into too small a space, or columns are stretched out over too large an area, your page can look odd or even be illegible.

In short, both fixed and fluid layouts have their advantages and disadvantages, and you'll have to decide which one to implement according to the needs of your particular site (or your particular design tastes).

Let's work through creating a few fixed and fluid layouts. Open the condos.html file in your downloads folder and create a CSS document called condos.css to follow along.

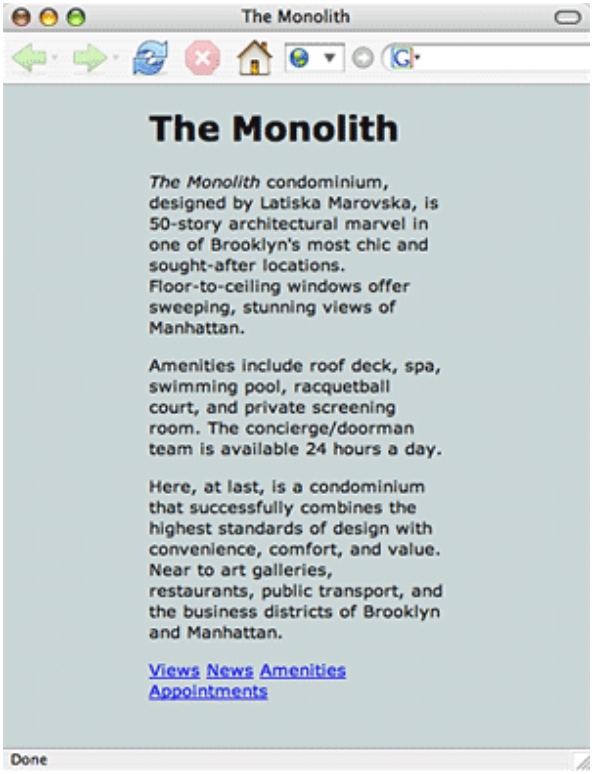
One-Column Fluid Layout

The easiest layout to create is the one-column fluid layout. You just have to assign a percentage to the left and right margin of the body element.

For example, you can use this CSS to style the body element of the page:

```
body {
    font: 13px verdana, sans-serif;
    background-color: #cad6d7;
    margin: 0 25%;
}
```

The result in your browser is this:



The margin is set to be 0 at the top and bottom and 25% on the sides.

One-Column Fixed Layout

Creating a centered, fixed one-column layout is a bit more involved. First, wrap all the content in your page body in a div, and give it an ID attribute:

```
<div id="content"></div>
```

Then, apply a width, some padding, alignment, background color, and border to this div:

```
#content {
    width: 500px;
    padding: 15px;
    text-align: justify;
    background-color: #cad6d7;
    border: 1px solid black;
}
```

Next, add a 50% padding to the body element:

```
body {
    font: 13px verdana, sans-serif;
    padding-left: 50%;
}
```

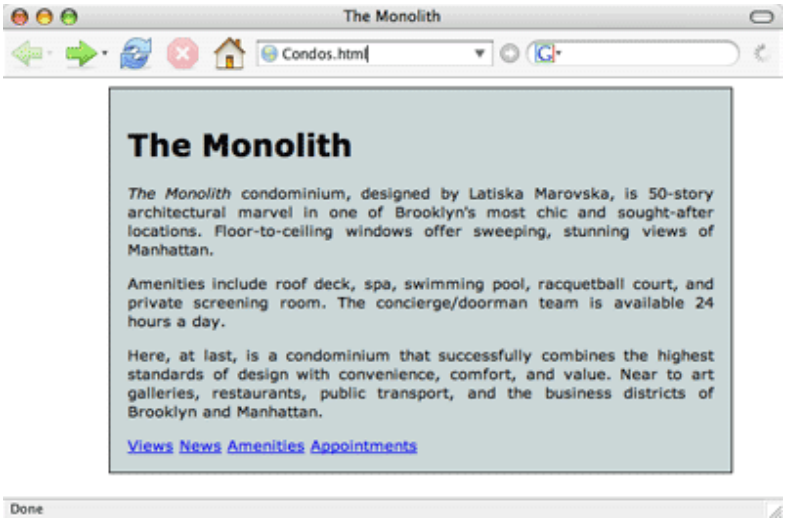
Finally, add a left margin to the div that is equal to half the width of the column:

```
#content {
    width: 500px;
    margin-left: -250px;
```



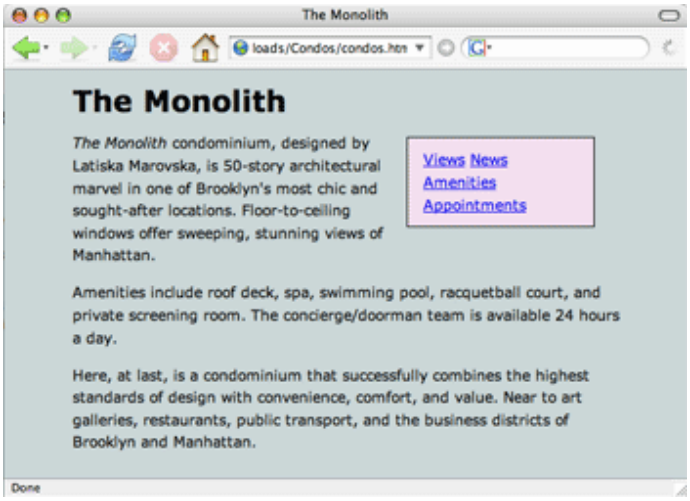
```
    padding: 15px;
    text-align: justify;
    background-color: #cad6d7;
    border: 1px solid black;
}
```

This gives us a page that looks like this:



Two-Column Fluid Layout

Let's move on to creating a two-column fluid layout which, when we're through, will look something like this:



The main column should be fluid, but the navigation column's width will be fixed. Delete your "content" div, and remove everything from your CSS file:

You'll see that your document contains some other divs, sectioning off the header, main text column, and navigation:

```
<div id="header">
<h1>The Monolith</h1>
</div>
<div id="main">
<em>The Monolit
...
```

```
</div>
<div id="menu">
<a href ="blank">Views</a>
...
</div>
```

Essentially, what you're going to do is float the menu to the right. As all floated elements need a designated width, we are going to assign the menu column a width, as well as some other attributes:

```
#menu {
    float: right;
    width: 150px;
    margin-right: 1.67em;
    border: 1px solid black;
    padding: 10px 1em 10px 1em;
}
```

We're also going to add a left and right-hand margin to the body element to squeeze the columns in a little, as well as specify a font and background color:

```
body {
    margin-left: 10%;
    margin-right: 10%;
    font: 0.9em/1.5em Verdana, sans-serif;
    background-color: #cad6d7;
}
```

The only thing left to do is to tweak some of the lesser aesthetic properties of the menu:

```
#menu {
    float: right;
    width: 150px;
    margin-top: 5px;
    background-color: #f3deee;
    margin-right: 1.67em;
    border: 1px solid black;
    padding: 10px 1em 10px 1em;
}
```

Notice that the menu is floated, but far down the page because its div in the HTML code is after the main div. This is an easy fix: simply cut out the entire menu div and paste it above the main div in the HTML page. The order of your divs in your code matters.

Two-Column Fixed Layout

You can create a fixed-width two-column layout similarly. Instead of setting the width of the main column using percentages, however, you set it using a length (like pixels).

Then, just give the body element a width, which will thereby set the width for the header and navigation column as well (the navigation column will take up the remainder of the width of the body—here, 360 pixels—and the header will span the whole length of the body):

```
body {
    margin: 10px;
    padding: 0;
    width: 800px;
    font: 0.9em/1.5em Verdana, sans-serif;
    background-color: #cad6d7;
}
#header {
    margin-left: 10px;
}
#main {
    float: left;
    width: 440px;
    padding: 10px 1em;
}
#menu {
    padding: 30px;
}
```

The page will remain the same as the browser resizes to be smaller. When the browser size becomes narrower than 800px, the width of the body, a scroll bar will appear so that the user can scroll to the right to see any content on the page that is outside the browser window.

Want some extra practice? Create your own page layout with fixed and fluid columns. Open up your alice.html document and create a two-column layout. Create a div for the main content area, and place the table of contents in the second column. Then float the image so that the page text wraps around it.

Troubleshooting CSS Positioning and Browser Inconsistencies

Despite how powerful, versatile, and easy-to-use CSS is, designers often become quite frustrated when taking a stab at creating their first CSS page layouts. The "rules" that browsers are supposed to follow when floating, positioning, and stacking elements can be quite complicated, and, to make matters worse, not all browsers actually *follow* the rules.

This can result in pages that behave unexpectedly, erratically, and differently in different browsers. Here are some guidelines for troubleshooting your layouts.

CSS Troubleshooting Basics

So, what do you do when you're having trouble with a CSS-based layout? Take a deep breath, relax, and try the following tips:

1. Make sure your code validates

Most browsers try to make sense of invalid or otherwise bad markup, but not all of them reconcile errors in the same way or to the same degree. Coding errors can make your page behave unpredictably, and, given all the tools out there to clean up your code, there's little reason to have invalid code.

Always run your site url through the HTML5 validator:
https://html5.validator.nu/

2. Always test your sites in different browsers

While standards are generally becoming more unified across the Web, variations still exist in the way different browsers display content. Unfortunately, older versions of Internet Explorer (IE 5 and 6) are known for browser bugs and issues with the box model and generated content selectors.

3. Don't reinvent the wheel

If you're having problems getting a page to look right in Internet Explorer (or any other browser), and you know that your HTML and CSS validates, you might want to check out a list of common browser compatibility bugs and their solutions.

Professional designers have already come up with clever and ingenious ways to circumvent the various glitches and browser bugs that make trouble for CSS-based layouts.

Here are three to get you started:

1. 5 CSS Tricks to Avoid Cross Browser Issues
2. Quick Steps for Finding CSS Issues
3. Tips for Debugging HTML & CSS

Finally, now that we are getting more proficient with ID selectors, here are a couple of technical points to remember.

Creating and Using ID Selectors

Using ID selectors along with div elements is an ideal way to split up your page into logical chunks, based on a content section's *role* on a page (whether it be a page header, footer, or navigational element). Not only does it make it easy to manipulate these different parts of a page using CSS, but breaking down the content by function also makes more *meaningful* sense of the HTML code.

There are a couple of technical rules to keep in mind, however:

1. Remember that you can only assign one HTML element a particular ID tag.

If you call one div element "navigation", you cannot use that ID name in another div on the Web page. If you need to assign a particular role to more than one element on a page, use the class attribute instead.

2. Don't imply presentational aspects in your ID names.

Even if you do plan on putting your navigation menu to the right of your main content, don't name it something like "rightcol". You want to assign an ID name based on what the element does, rather than what it looks like. Your navigation menu should be called something like "nav".

That way, if you (or your client) ever decide to change the location of the navigation menu (like to the left of the main content instead), you aren't stuck with an ID name that implies it should be somewhere else! The same goes for ID names that include colors. Avoid names like "redbox" or "bluesidebar", as these colors may someday change!

This hearkens back to golden rule: Presentation should be strictly separated from content. Nothing in your HTML should control (or imply) how the page looks. You've got CSS for that.

in the next lesson

- Learn how HTML uses semantic elements to structure page content.
- Learn guidelines for using semantic elements.
- Learn how to add audio and video content using HTML5.
- Learn principles for testing on multiple platforms and browsers.

Coming Up Next:

Discussion
Share your thoughts and opinions on using CSS for layout in the Discussion area.

Exercise
Use CSS to create a two-column layout for your coffee site and style the margins and padding for site content.

HTML and CSS | Model Menus

sessionscollege
FOR PROFESSIONAL DESIGN

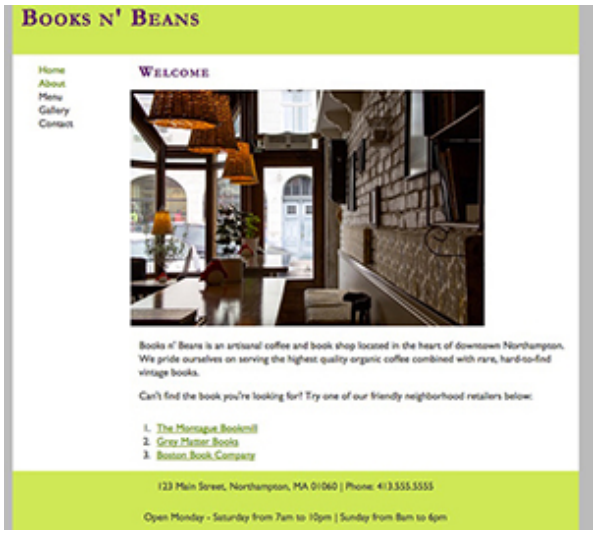
Exercise 3

Model Menus

In the last lecture, we covered a *lot*: everything from the CSS box model, to absolute and relative positioning and floats, and one and two-column layouts.

In this exercise, we'll put it all together by designing the layout of our Books n' Beans site.

In the second part of this exercise, we'll add a Menu page to the Books n' Beans Web site. Then, we'll add custom CSS classes and borders to the page, checking our work against the rest of the site to make sure every page feels like part of a whole.



Let's add some columns to our coffee shop.

Important: Before beginning Exercise 3, make sure to duplicate your files so you don't overwrite your Exercise 2 website! The easiest way to do this is to copy the files and past them into a folder called "exercise3".

Performance Objectives

- Reset the default styles for your coffee shop site.
- Style the height, margins, and padding of element selectors and divs.
- Implement a two-column layout for the site.
- Use CSS selectors (class and span) to apply styles to selected site areas.
- Consolidate and comment your site code.

note

Assignments are evaluated for creativity, technical proficiency, and understanding of concepts covered in the lecture.

Project Brief

Part 1: Incorporating the CSS Box Model

As you were completing Exercise Two, I'm sure you noticed some spacing issues in the Books n' Beans Web site. Without any CSS layout design, the text bumps right up against the edge of the container div, while the header, main, and footer div are separated by an unnecessary gap.

Depending on what browser you're using, you might have gaps between the divs on your Books n' Beans site that look something like this.

These issues can be remedied by applying the correct margins and padding to our site. The CSS box model comes to the rescue.

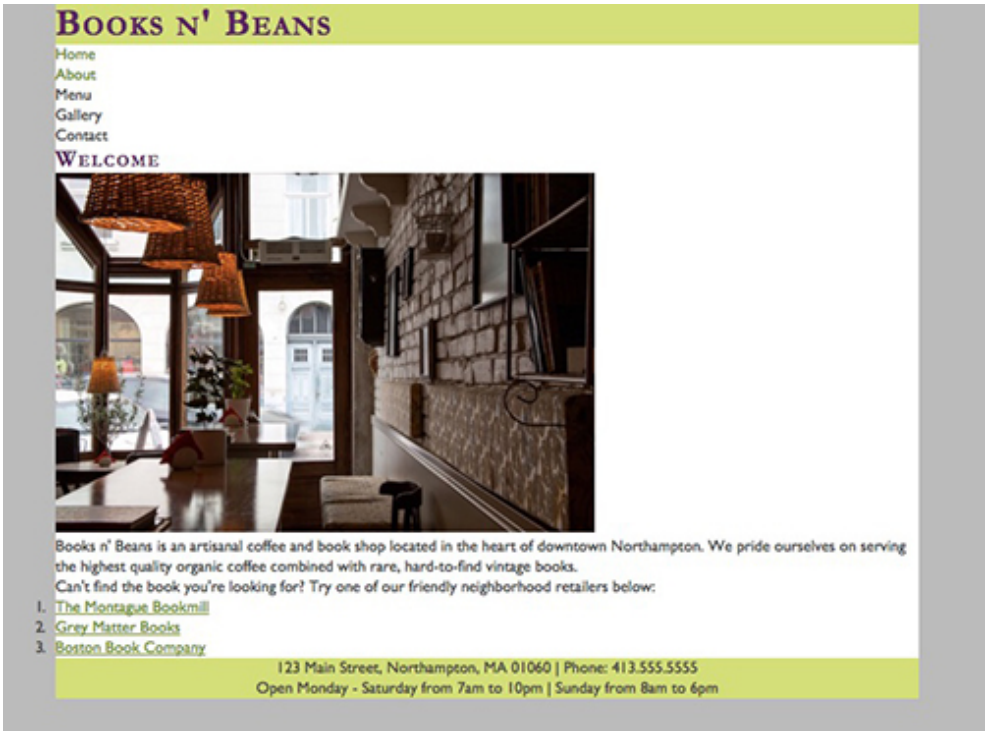
These gaps are caused by your browser just trying to be helpful. If the browser doesn't see certain CSS styles in your style sheet, it defaults back to the **user agent style sheet**, which is simply the browser's own default style sheet. CSS is so essential to displaying documents on the Web that even your browser has its own default styles!

Resetting CSS Styles

This is why customizing each and every CSS style is so important. Every browser's default style sheet is a little bit different, so if we want our site to look and act the same way across the Web, it's our responsibility to customize these default styles. We can get rid of this unwanted space by adding a few "reset" styles to our CSS file.

Reset styles are, quite simply, CSS styles that will overwrite the default browser styles. If you do a quick search online, you'll find a number of helpful reset style sheets created by different Web designers. The important thing to know about reset styles is that they're meant as a starting point. Reset styles are designed to provide you with a blank slate, so you can troubleshoot spacing and sizing issues one element at a time. Let's see what happens when we add some sample reset styles to our style sheet:

```
html, body {
        margin: 0;
        padding: 0;
}

h1, h2, h3, h4, h5, h6 {

margin: 0;
}

p {
        margin: 0;
}

ul, ol {
        margin: 0;
        padding: 0;
}

li  {
        margin: 0;
        padding: 0;
}
```



The Books n' Beans Web site with margins and padding reset

Hmm. It looks like our reset styles have solved some of our spacing problems, but created some others. Reset styles can remove unwanted space between divs, but they can also squish paragraphs and bullet lists too close together. After adding reset styles to your site, it's your responsibility as a designer to go through and customize each style. We'll do that in just a moment! First, let's take a look at our site header and footer.

Assigning a Height to the Header and Footer

Notice that the header and footer divs on the Books n' Beans site are exactly the height of the text inside of them. If there's no height assigned to a div, the div will stretch to fit whatever's inside of it. This is helpful, but it doesn't give the text a lot of breathing room. You should already have CSS selectors in your stylesheet for the #header and #footer div. Let's add a height of 100px to each of these selectors.

```
#header, #footer {
        height: 100px;
}
```

Note: Depending on the way you've formatted your selectors, your style sheet may look different from the styles I've included above. Rather than copy/pasting the above code snippet, try adding the height to the existing #header and #footer styles in your style sheet.

Adding Margins and Padding to Element Selectors

Now, let's add margins and padding to our headings, paragraphs, and list elements. Change this style rule:

```
h1, h2, h3, h4, h5, h6 {

margin: 0;
}
```


To this:

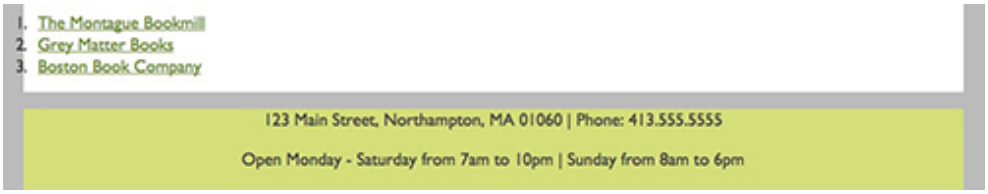
```
h1, h2, h3, h4, h5, h6 {

margin: 0;
padding: 15px;
}
```

In the CSS box model, margins control space outside of block-level elements, while padding controls the space inside blocks. By keeping our heading margins at 0, the site header can stay firmly planted at the top of the page, but the text on the inside of the header and main content area gets 15px of breathing room on all sides. Adding a margin might move the actual blocks around, and adding padding just moves the content inside the block away from the edges.

Now let's add 15px of padding to the left and right of the paragraph element so the text doesn't bump up against the sides of the page. We'll control the space between paragraphs using margins:

```
p {
margin-top: 1em;
        padding-left: 15px;
        padding-right: 15px;

}
```

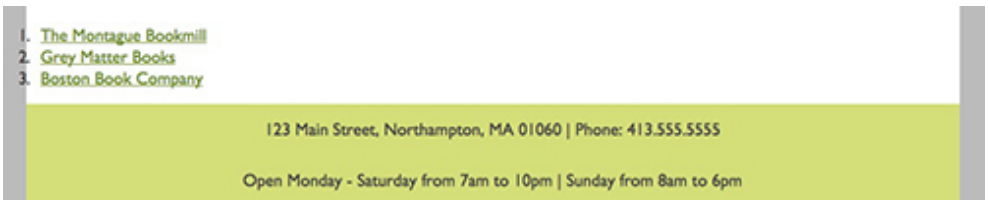
Our paragraph text should now be aligned with the heading text, with a healthy 1em margin between each paragraph. But wait! You may notice that adding a margin back into the paragraph element pushed the footer away from the main content area again.



How do we remove the unwanted margin above the footer paragraph text without impacting the rest of the margins on our site? This is where descendent selectors come to save the day!

```
#footer p {
        margin: 0;
        padding; 15px;
}
```

Ah, that's better! Now, the footer snaps back to the main content area and there's a nice bit of padding around all of the text.



I used a descendant selector to style all paragraph text in the footer div

Now I'd like you to go through and add margins and padding to the ul, ol, and li elements in your style sheet. It can be useful to shorthand margins and padding when possible, but remember that your lists shouldn't always have equal margins on all sides, so you'll need to try different combinations until you figure out which works best with your design.

Here's a refresher:

Equal padding on all sides:

```
ol, ul {
margin: 30px;
}
```

Margins styled individually:

```
ol, ul {
        margin-top: 0;
        margin-right: 10px;
        margin-bottom: 0;
        margin-left: 30px;
}
```

The same margins as above, written in shorthand:

```
ol, ul {
        margin: 0 10px 0 30px;
}
```

If you are unsure of where your divs are located and how your changes are affecting your page layout, try adding a border to a specific box element:

```
ol, ul {
        margin: 0 10px 0 30px;
    border: 5px solid orange;
}
```

You can adjust margins and padding to your heart's content, and remove the border when done!

When you're finished, you should have a clean, well-formatted design with custom styles applied to the following selectors:

- h1 through h6 headings
- p
- ul
- ol
- li
- #header
- #footer
- #footer p

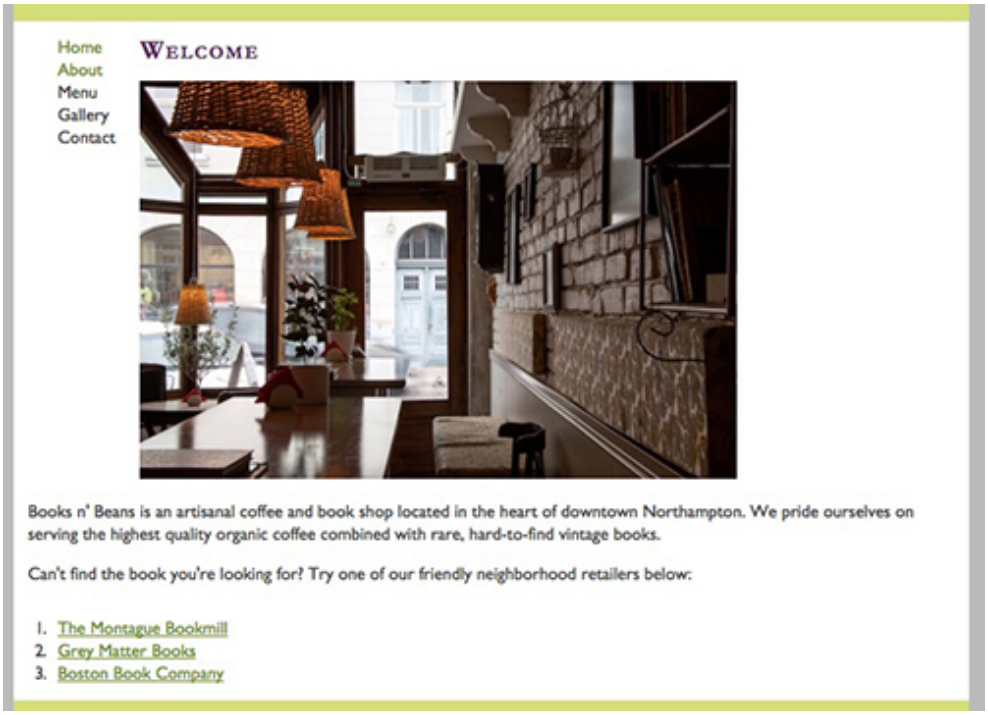
You can combine multiple selectors in whatever way works best for your style sheet, and you can style additional selectors (and add descendent selectors) as needed. I'll be looking for clean, compact, commented code in your submission, so take some time to review the rules for condensing CSS styles.

Looking good? Let's add some floats to incorporate a column-based layout into our design.

Formatting with Floats

To get the two-column layout we're looking for, we'll want to start by floating the navigation menu to the left of the page. This can be accomplished by adding a left float to the #menu div. We'll also want to assign a width to the menu, as floated elements without set widths can act in unpredictable ways!

```
#menu {
        float: left;
        width: 200px;
}
```



This moves the menu to the left of the page, but to get the full effect, we've got to float the text and image in the main content area to the right. There's just one problem—the #menu div is currently inside of the #main div! Trying to float the #main div to the right with the menu still inside of it is going to cause a whole host of layout problems.

To work around this problem, we need to edit our HTML documents and simply add a new div called #content inside of the #main div, right before the Welcome heading.

```
<div id="main">
<div id="menu">
<ul>
<li><a href=#>Home</a></li>
<li><a href=#>About</a></li>
<li>Menu</li>
<li>Gallery</li>
```



```
<li>Contact</li>
</ul>
</div>

<div id="content">
<h2>Welcome</h2>
```

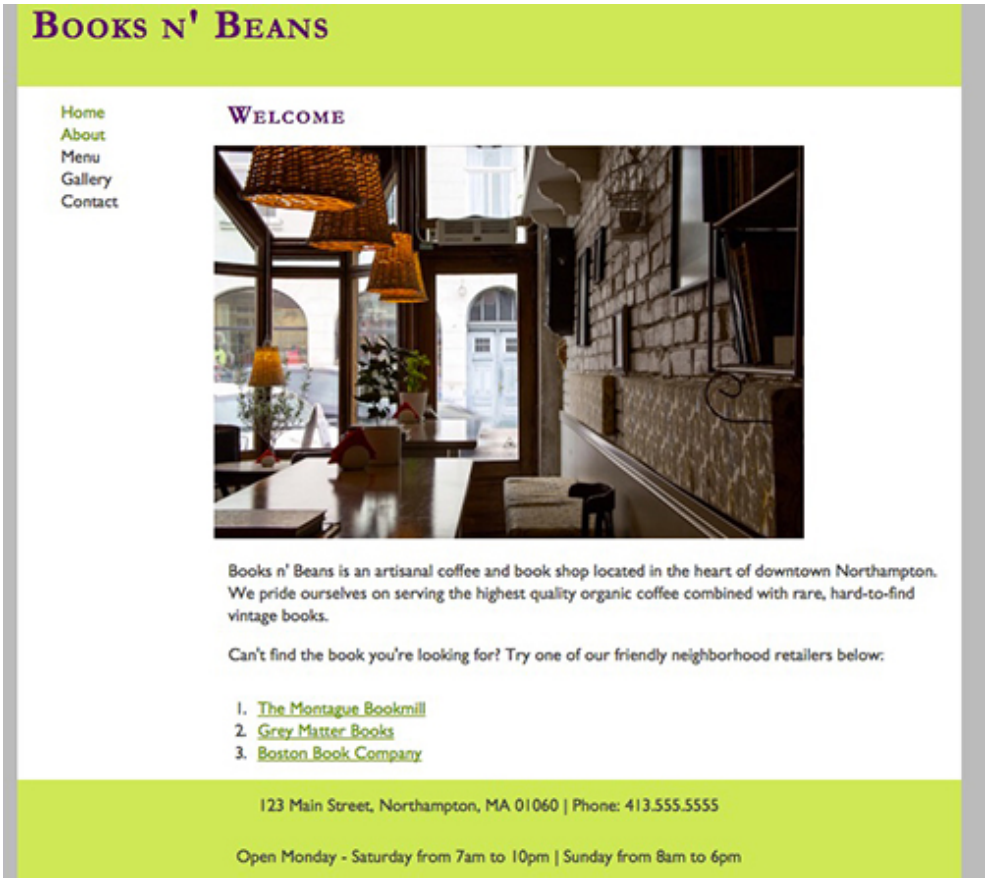
Make sure to close the #content div right before the closing tag of the #main div, and before the beginning of the footer. After adding the new #content div to your HTML markup, it's time to style the #content selector. Start by floating the content to the right.

```
#content {
        float: right;
        width: 720px;
}
```

If you preview the page at this point, you'll notice that the content is all over the place, with the navigation menu mashed up against the header and the footer at the top of the page. To add a bit of order to this chaos, we'll need to assign the **clear** property to the #header and #footer div.

```
#header, #footer {
        height: 100px;
        clear: both;
}
```

The clear property forces the header and footer to "clear" the floated elements so they don't try to jump up adjacent to the floats.



Remember to add the new markup to your About page before moving on. Your CSS style sheet will control the styles across your site, but without the correct HTML, there won't be anything there to style!

Part 2: Adding Style and Class to the Menu Page

At this point, I'd say our Books n' Beans page is looking pretty good! Now, let's add a Menu page to the site to indulge our coffee-loving patrons. Rather than starting from scratch, I'd recommend starting from a copy of the About page. Duplicate the about.html file and rename the file menu.html. Download the menu file from the Exercise 3 downloads and customize the About page content with the menu text, like so:

- Menu: Wrap the Menu heading in h2 tags
- Coffee, Tea, Pastries, and Specialty Items: Create a custom heading style using descendant selectors, or create a custom class
- Individual menu items: Create a custom paragraph style or list style using descendant selectors, or create a custom class

After choosing some basic styles for the Menu text, we'll put our knowledge of div classes to use on this page. Remember the difference between div IDs and classes? IDs can only be used once per page (like our #content div), but classes can be used again and again. ID selectors always begin with a pound sign (#), while class selectors always begin with a period (.).

I'd like you to wrap each of the food and beverage sections in a class called "menubox," starting with the Coffee section and repeating the process for Tea, Pastries, and Specialty Items.

```
<div class="menubox">
Coffee

Espresso
Americano
Macchiato
Latte
Cappuccino
</div>
```

After wrapping each section in a "menubox" class, add the .menubox class selector to your style sheet. The example below includes a border. Please customize the border thickness and color to match your site design.

```
.menubox {
        border: 1px solid #999;
}
```

Customize the .menubox styles to your liking. Be sure to include:

- A border
- Margins and padding, as needed
- A background color (optional)
- Custom text styles that match with the overall design

When you're happy with your Menu page, remember to add a link to the page in the navigation menu on all pages of your site.

Part 3: Creating the Gallery and Contact Page

By this point you may have noticed that we're missing the final two pages of our site: the Gallery page and the Contact page.

In the Exercise 3 downloads folder, which you downloaded at the beginning of this exercise, you'll also find the text and images for the Gallery page, and the Contact page text. Let's start by creating the Gallery page. Duplicate an existing Books n' Beans HTML file (I recommend the Home or About page) and rename the file gallery.html.

Formatting the Gallery Page

Add the Gallery page heading to your new HTML document. Wrap the heading in the correct heading tags and wrap the remaining text from the gallery.txt file in paragraph tags. As you can see, there's very little text to format on this page. The meat of this page is in the images, which we'll be laying out in a clean floated gallery.

In your Exercise 3 downloads, browse through the **gallery-images** folder and select four images for the gallery page. We're looking for fresh content here, so make sure you're not using images that have appeared elsewhere on your site! At least one of the images you choose should feature real people. Images of people will help your site feel more alive, which is important for a business that caters to humans!

Add the four images you've selected to the gallery page HTML, and give each image a descriptive alt tag.

```
<img src="coffeeshop.jpg" alt="coffee shop">
```

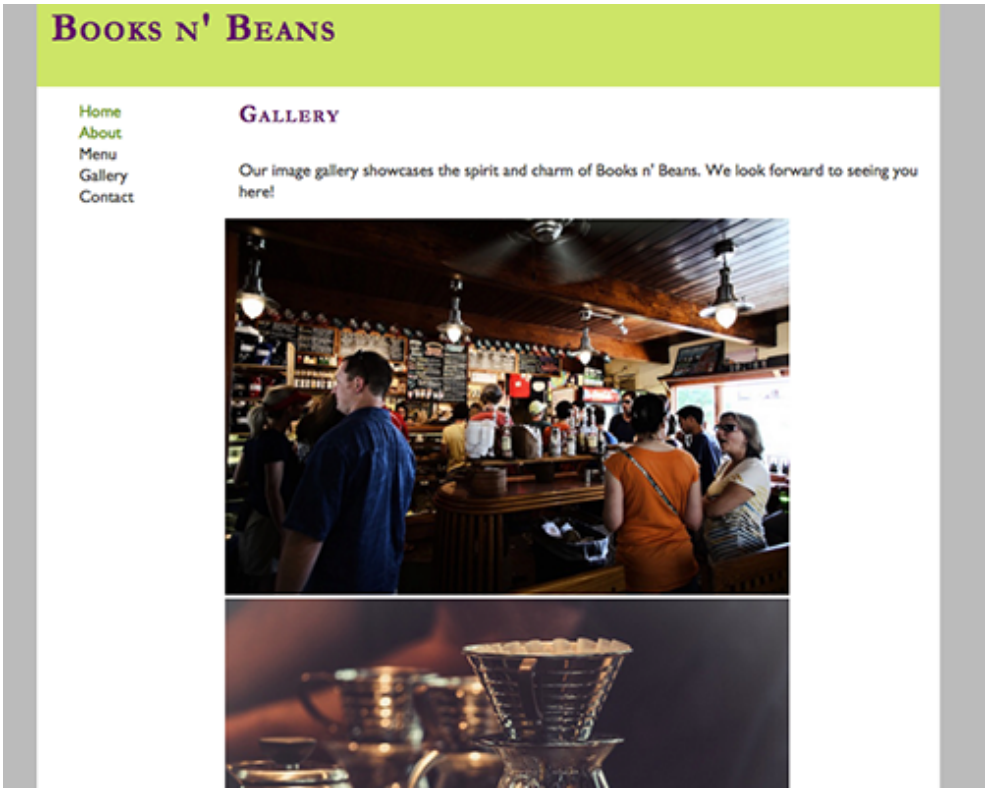
Next, we'll add a special image class to the beginning of each image. This will allow us to target just images with the class name of "gallery" in our CSS, instead of targeting every image on the site.

```
<img class="gallery" src="image.jpg" alt="my image">
```

See the snippet below for an example of what the image code might look like in your document. You may be using different images in your design, so don't copy the code word-for-word. Just use this example as a guide:


```
<p>Our image gallery showcases the spirit and charm of Books
n' Beans. We look forward to seeing you here!</p>

<img class="gallery" src="busycafe.jpg" alt="busy coffee shop">
<img class="gallery" src="coffee-fancy.jpg" alt="drip coffee">
<img class="gallery" src="coffeecups.jpg" alt="coffee cups">
<img class="gallery" src="barista1.jpg" alt="woman barista">
```

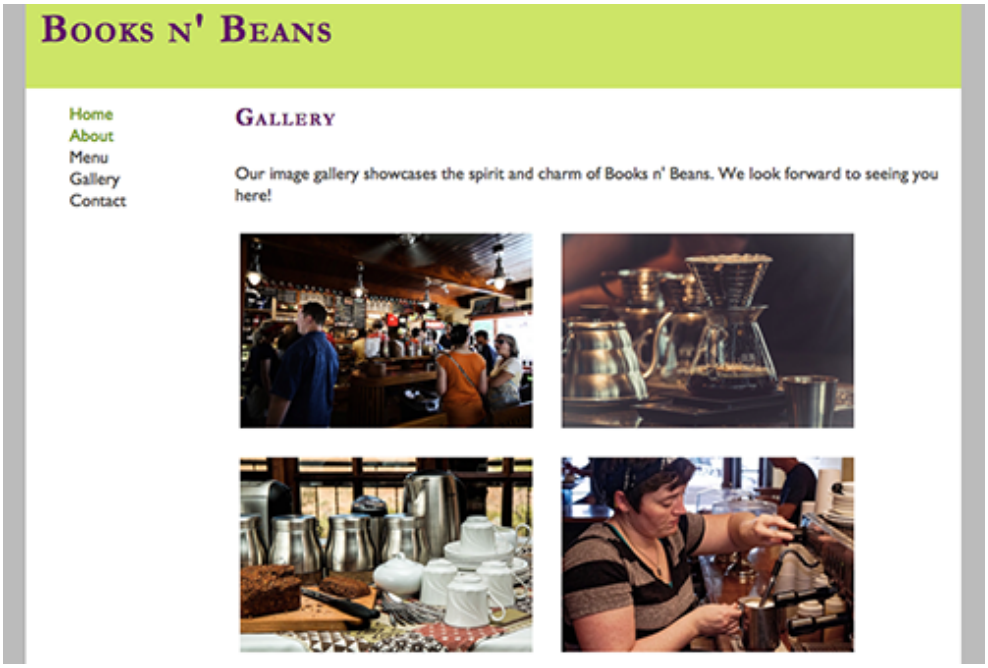
Now, let's take a look at the results.



View larger image

Hmmm. Our high-resolution images look very nice, but this isn't a very user-friendly gallery! Let's use CSS to decrease the size of the images and lay these out in an appealing grid.

```
img.gallery {
        width: 300px;
        float: left;
        padding: 10px 10px 20px 20px;
}
```



View larger image

Using the above style rules, I've restricted the width of the images to 300px. I then used the **float: left;** property to flow the images to the left, creating a 2x2 image grid. Finally, I applied padding to the top, left, bottom, and right of the images to give them some breathing room. Without any padding, the images would appear as a solid block—not very pretty. Using these rules as a guide, experiment with different widths, padding, and margin settings until you find the sizes and spacing that works best with your design.

Formatting the Contact Page

When the Gallery page is complete, duplicate one more Books n' Beans page and save the file as contact.html. Customize the page with the Contact page text included in your Exercise 3 downloads. Wrap the Contact page heading in the correct heading tag and wrap the rest of the text in paragraph tags.

The Contact page should provide some great opportunities to use spans or emphasis tags. There's not a lot of content on this page, so I'd like to see you approach the text creatively here!

When you're finished, remember to add a link back to the Gallery and Contact pages in the navigation menu on all pages of your site. Upload all files to your Web host along with any associated image files.

Remember to validate your work using the HTML5 validator, and check to make sure all links are working properly before submitting your Web site to the Dropbox for evaluation.

Design Concepts

At this point you should have three functional, well-designed pages on your site: the Home page, the About page, and the Menu page. When you preview the Books n' Beans Web site in your browser, do you find that the site looks consistent from page to page? Consider the following questions as you review your design:

- Does each page look like it's part of the same Web site?

- Are there any big differences in the color styles, font styles, or sizes that might be confusing to users?
- Have I made appropriate color choices for this design?
- Do my color choices follow good color-matching principles?
- Do my type styles work well together?
- Do the headings feel distinct from the paragraph text?
- Do my design choices follow Web accessibility standards?
- Does the code validate, is the text readable, and are there any color choices or background patterns that might impact usability?

If you're not sure about the answer to one of these questions, remember that there are many great color and type-matching resources online. Here are a few to get you started:

- Adobe Color CC: Explore different color rules and time-tested combinations.
- Colour Lovers: Search palettes by keyword, or organize by highest-rated color combos.
- I Font You: Examples of different font combinations, including many browser-safe fonts.

Kick It up a Notch

Before publishing your Books n' Beans site to the Web, I'd like you to incorporate one design element that will take this design to the next level. Using the resources listed above for inspiration, choose one of the following design elements from the list below to incorporate into your final site:

- Background image. Add a background image to a block-level element, to a custom div, or to the body of your site. Choose a high-quality graphic from Pixabay, and make sure you've chosen something that won't interfere with the readability of the text.
- Image captions. Add a custom class to incorporate descriptive caption text below the images on each page.
- Social media icons. Add social media icons to the Books n' Beans header or footer so patrons can find you on the Web. These icons don't need to link to real social media accounts. Just the images will do!

(Try iconfinder.com.)

- Testimonials. Add patron testimonials to the site. Wrap text in the HTML <blockquote> element and assign custom styles to this element to give your blockquotes some pizzazz. Get creative with this one, and have fun making up some quotes!

Describe Your Experience

When learning code, it's important to reflect on your experience. When you post your work, include your answers to the following questions:

1. What CSS concepts are you having the most fun with? What concepts are you struggling with?
2. Which CSS layout concepts do you find easiest/hardest to grasp or put into practice?

We'll be moving onto different projects later in the course. For now, I look forward to seeing your work!

Grading Criteria:

What your instructor expects you to do:

☐ Demonstrate the ability to reset default styles and apply appropriate height, margins, and padding styles to a site.

☐ Show the ability to implement an attractive two-column layout for a site using positioning and floats.

☐ Show the ability to class and span selectors to apply styles to specific site areas.

☐ Demonstrate your knowledge of clean, standards-compliant code practices by consolidating your CSS code, including clear comments in your file, and validating/troubleshooting your site.

How to Post:

Once you're done, go to the Dropbox for this exercise and post your URL for this assignment along with your written comments.

If you have a question before sending your completed exercise for grading, be sure to contact your instructor via the People area.

I look forward to seeing your work!

sessionscollege
FOR PROFESSIONAL DESIGN

Lecture 4

HTML5 Semantic Elements

In this course, we've talked a lot about the importance of separating page structure (HTML) from presentation (CSS). We've also touched on some of the ways that HTML5 has been evolving to streamline and improve Web coding practices.

In this lesson, we'll delve into HTML semantic elements and find out how this new(ish) approach to structuring pages is helping developers and browsers make better sense of Web pages.

Another kind of box: HTML semantic elements

We'll also discuss how to use HTML5 to implement audio and video content.

Learning Objectives

In this lecture, you can expect to:

- Learn how HTML uses semantic elements to structure page content.
- Learn guidelines for using semantic elements.
- Learn how to add audio and video content using HTML5.
- Learn principles for testing on multiple platforms and browsers.

Semantic Versus Non-Semantic Elements

Semantics is the study of the meaning of words. So what is semantic HTML? Simply, **semantic elements** are HTML elements with inherent meaning.

As we've discovered in this course, many HTML tags have very specific meanings. Examples of semantic elements that you've learned about include:

- `<img>`
- `<em>`
- `<h1>` - `<h6>`

Each one of those tags tells the browser something specific about the information on the page: this is an image, this is emphasized, this is a header.

Others tags have no meaning. Div IDs and classes, for example, don't have any semantic meaning on their own. If you wanted to, you could wrap div tags around any part of a Web page and call that part of the page `<div id="nav">`.

note

Semantic tags such as <h1> and <img> tell you something about the meaning of the content.

While it wouldn't make sense to label your footer "nav", you could do it! The name of a div or class is a name of your choosing. It doesn't look or behave differently until you apply CSS styles to its selector in your style sheet. And most importantly, the Web browser does not look at your `<div id="nav">` and interpret it as navigation.

note

HTML5 semantic elements describe the meaning of a part of a Web page.

Examples of non-semantic elements that you've learned about include:

-
- <class>
-

Along Comes HTML5

In HTML5, new semantic elements were created to better describe the different parts of a Web page. Those elements include, among others:

- <header>
- <nav>
- <section>
- <article>
- <aside>
- <footer>

These HTML5 semantic elements, which are supported in all modern browsers, serve the function of helping to streamline the way we describe the structure of a Web page. Previously, a Web designer might have used any of the following to describe a site navigation bar:

```
<div id="nav">
<div id=navigation">
<div class="nav">
<div id="menu">
```

Now, a designer can simply use the HTML5 tag:

note

The header tag can specify the header for a Web page or section of a page.

```
<nav>
  <a href="#">Link 1</a>
  <a href="#">Link 2</a>
  <a href="#">Link 3</a>
  <a href="#">Link 4</a>
</nav>
```

HTML5 semantic elements are clean, simple, and help make the Web a more functional and accessible place.

Using HTML5 Semantic Elements

So, how do you go about using these semantic elements on your Web site? The <footer> element should just replace the footer div, right? That's correct, but their use isn't always straightforward. There are special rules for how each of these elements should be used, when each element should be used, and where these elements should appear in the structure of your document.

Let's go through them, one by one.

The <header> Element

important!

Each header must contain at least one heading tag.

The <header> element contains important introductory information about your page. The <header> element can be used to specify a single header area for your entire Web page, or it can be used to describe the header of an individual page section. Whereas you were only allowed to use one per document, the <header> element can be used multiple times in one document, as long as it's being used correctly!

Example #1:

```
<header>
    <h1>My Fabulous Web Site</h1>
    <h2>About Me</h2>
    <p>Some introductory text about me.</p>
  </header>
```

Example #2:

```
<header id="banner">
  <h1>Website Title</h1>

  <nav><ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">About</a></li>
    <li><a href="#">Portfolio</a></li>
    <li><a href="#">Contact</a></li>
  </ul></nav>

</header>
```

As shown in the examples above, the header element can be used alone <header> or combined with an ID attribute <header id="banner">, which can be useful if you have more multiple <header> elements that require different styles. CSS styles for the <header> element might look something like this:

note

The nav element is designed to include the primary navigation on a site.

```
header {
        width: 960px;
        height: 100px;
        background-color: #eee;
}

header#banner {
        width: 960px;
        height: 100px;
        background-color: #eee;
}
```

In order for the <header> element to properly validate, each <header> must include at least one heading tag (<h1>-<h6>). <header> elements can also include images (like a site logo), paragraph text, and site navigation links <nav>. A <header> element cannot be placed inside a <footer> or another header tag.

Here's an example in the wild:

note

HTML5 semantic elements are helpful for search engines and screen readers.

The Lost World's Fairs site was created to showcase the beauty of HTML5. It uses the <header> element and other HTML5 tags.

```
<header id="header">
        <div id="mtn">
                <div class="container">
                        <div class="clearfix title">
                        <h1>Lost World's Fairs</h1>
                        <div id="button">
                                <span id="txt_nowwith">now with</span>
                                <span id="txt_woff">WOFF</span>
                        </div>
                        </div>
                        <div id="banner">
                                <div id="txt_beauty">Beauty of the Web</div>
                        </div>
                </div>
        </div>
</header>
```

The header element

The <nav> Element

The <nav> element contains the site navigation menu, or other navigation functions that will take you to a different page within the site. Only the primary site navigation links should go inside the <nav> element—not all anchor links on the site.

For example, it's not uncommon for a site footer to contain a list of navigation links. Because those are secondary or tertiary navigation links, the <nav> element isn't needed.

note

The section element groups related content on a page.

In HTML5 markup, the <nav> element can often be found inside the <header> element, though it isn't required there. The <nav> element can be used in place of div tags like or , as in the example below:

```
<nav>
<ul>
<li><a href="#">Home</a></li>
<li><a href="#">About</a></li>
<li><a href="#">Contact</a></li>
</ul>
</nav>
```

Because the <nav> tag has semantic meaning (where does not), it communicates to the browser that this particular list of links is a

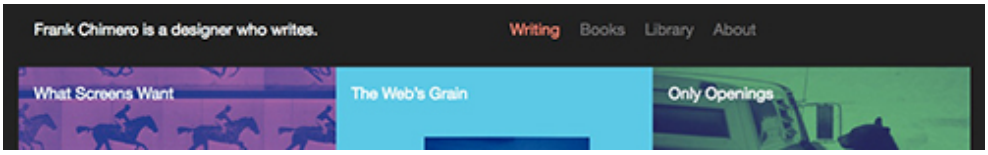
important!

The section element should not be used for styling purposes.

primary navigation menu. This can be especially useful for SEO (Search Engine Optimization) and screen readers!

CSS styles applied to the <nav> element might look something like this:

```
nav {
display: block;
height: 100px;
background-color: #000;
}
```



Designer Frank Chimero uses the <nav> element inside of the <header> element for a clean, single line navigation bar.

```
<nav role="navigation">
    <ul>
        <li><a href="/writing/">Writing</a></li>
        <li><a href="/books/">Books</a></li>
        <li><a href="/library/">Library</a></li>
        <li><a href="/about/">About</a></li>
    </ul>
</nav>
```

The nav element

The <section> Element

The <section> element is used to group related sections of a document together. This one can be a little tricky to grasp, so I'll give a few different examples to illustrate how it's used.

Imagine that you're about to undertake a big Web design project. Before diving in, you'll probably want to outline your page content so you know where everything's going to. Your draft outline might look like this:

- My Portfolio
  - Animation Projects Section
    - Descriptions of animation projects with images
  - Audio Projects Section
    - Descriptions of audio projects with sound clips
  - Illustration Projects Section
    - Descriptions of illustration projects with images

note

The article element is designed to tag things like blog posts and news articles.

Resulting in the following HTML5 markup:

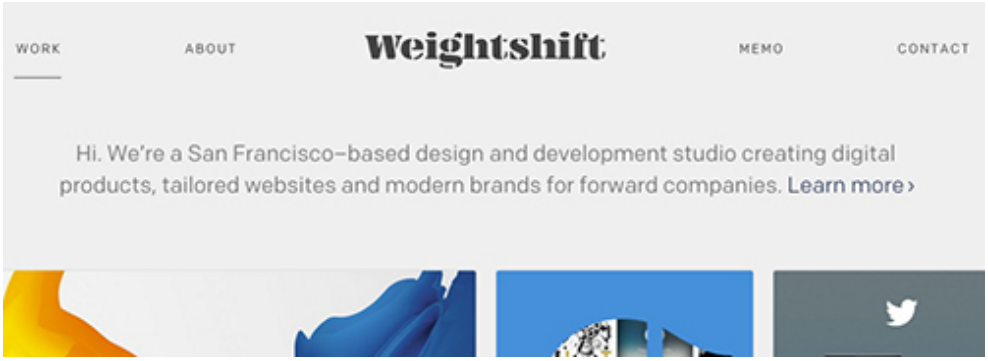
note

The article element should generally contain at least one heading.

```
<h1>My Portfolio</h1>

<section>
<h2>Animation Projects Section</h2>
<p>Descriptions of animation projects with images.</p>
</section>

<section>
<h2>Audio Projects Section</h2>
<p>Descriptions of audio projects with sound clips.</p>
</section>

<section>
<h2>Illustration Projects Section</h2>
<p>Descriptions of illustration projects with images.</p>
</section>
```

As you can see, there's no limit to the number of times you can use the <section> tag per page. There are also no hard and fast rules about what defines a document section, so you'll have to use your best judgement when grouping thematic content together. A section can even contain a <header> or <footer> tag, as long as the content is all related.

One thing that's important to note is that the <section> element should not be used for styling purposes. Let section tags do their semantic work behind the scenes. Any sections of the page that need special styling should be styled using div tags. Remember, even if you can't see what your section tags are doing on the page, your Web browser can read and interpret their meaning.



Weightshift studio in San Francisco uses multiple <section> elements to divide their page content. There is a <section> tag wrapped around the intro message shown above.

note

The aside element is designed for pullquotes and sidebars, like me!

```
115     <section id="intro">
116         <div class="container center">
117             <p>Hi. We're a San Francisco&ndash;based design and
    development studio creating digital products, tailored websites and
    modern brands for forward companies. <a href="/about/">Learn
    more</a></p>
118         </div>
119 </section>
```

The section element

The <article> element

The <article> element is for self-contained content that could be read independent of the Web site. Examples include things like blog posts, news articles, forum posts, and recipes.

It's easy to confuse <article> and <section> tags, in that both tags group related content together. It's especially confusing because article tags can contain multiple sections and section tags can contain multiple articles. So, when should you use which tag?

When trying to figure out whether an article or a section is more appropriate, ask yourself whether the content could easily be printed from the Web site as a complete piece. If the answer is no, the <section> tag is probably more appropriate. If the answer is yes, the <article> tag might be a great choice!

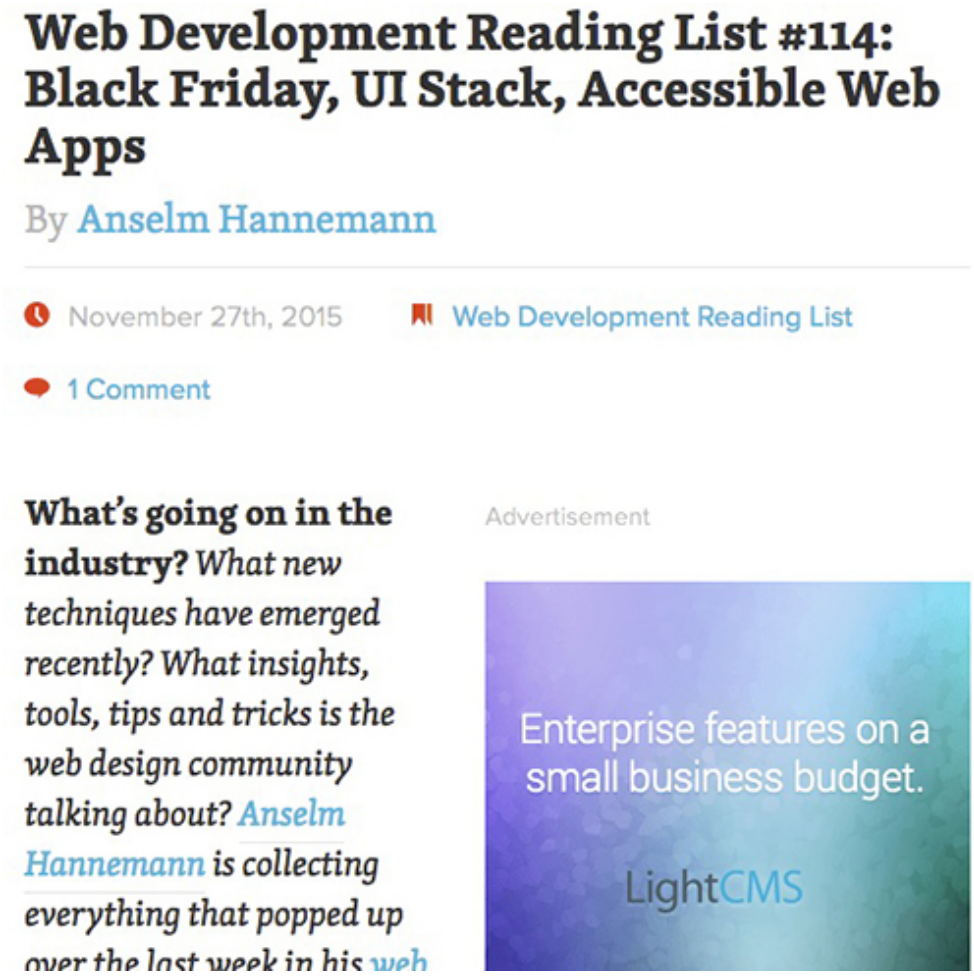
Example #1:

```
<article>
  <h1>Rabbit Habitats</h1>
  <p>Rabbit habitats, or "rabbitats", include meadows, woods,
  forests, grasslands, deserts and wetlands. Rabbits live in
  groups, and the best known species, the European rabbit,
  lives in underground burrows, or rabbit holes.</p>
</article>
```

Example #2:

```
<article>
  <h1>Rabbit Habitats</h1>
  <p>Rabbits live in many different habitats...</p>

  <section>
    <h2>Meadows</h2>
    <p>Sometimes they live in meadows...</p>
  </section>

  <section>
    <h2>Woods</h2>
    <p>Often they live in woods...</p>
  </section>
</article>
```

The <article> element should typically contain at least one heading. Similar to the section element, we must let article elements do semantic work but not style work. If styling is needed, you'll want to use a div.



Smashing Magazine, a blog for Web professionals, wraps every blog post in an <article> tag.

```
154 <article class="post-230553 post type-post status-publish format-standard has-post-
    thumbnail hentry category-general tag-web-development-reading-list"
    vocab="http://schema.org/" typeof="TechArticle">
155
156                 <h2>
157                 <a property="url"
    href="http://www.smashingmagazine.com/2015/11/black-friday-ui-stack-accessibility-
    performance/" rel="bookmark" title="Read 'Web Development Reading List #114: Black
    Friday, UI Stack, Accessible Web Apps'"><span property="name">Web Development Reading
    List #114: Black Friday, UI Stack, Accessible Web Apps</span></a>
    ...
    original_referer=http%3A%2F%2Fwww.smashingmagazine.com%2F2015%2F11%2Fblack-friday-ui-
    stack-accessibility-
    performance%2F&source=tweetbutton&text=Web%20Development%20Reading%20List%20%23114%3A
    %20Black%20Friday%2C%20UI%20Stack%2C%20Accessible%20Web%20Apps&url=http%3A%2F%2Fwww.s
    mashingmagazine.com%2F2015%2F11%2Fblack-friday-ui-stack-accessibility-
    performance%2F&via=smashingmag">Tweet it</a><a class="sot single"
    href="http://www.facebook.com/sharer/sharer.php?
    u=http%3A%2F%2Fwww.smashingmagazine.com%2F2015%2F11%2Fblack-friday-ui-stack-
    accessibility-performance%2F">Share on Facebook</a></p></p>
266
267 </article>
```

The article tag

The <aside> Element

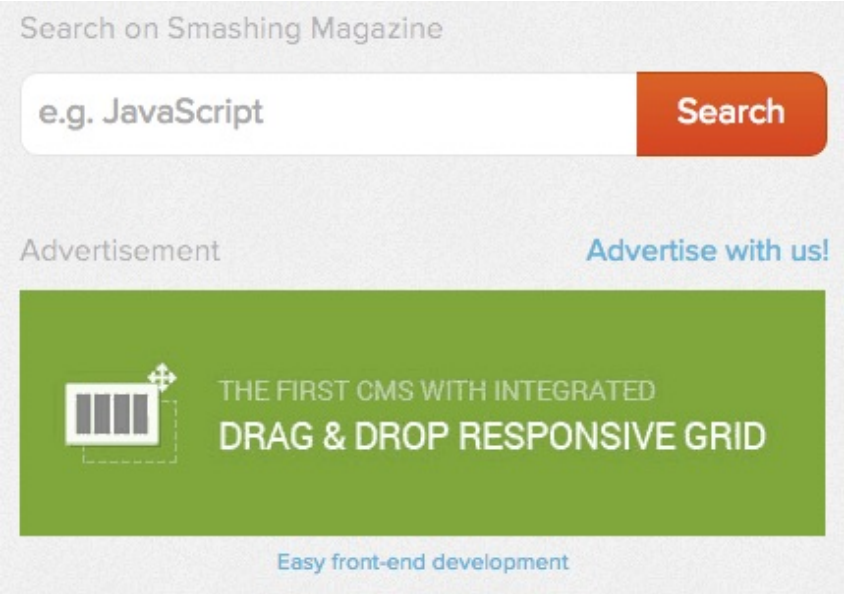
Still with me? Good! The `<aside>` element is a cousin of the `<article>` or `<section>` element. Use the `<aside>` tag when indicating content that's related to, but separate from, the content it's inside of. A good use of the aside element might include a pull-quote, a bit of sidebar information, or a special factoid.

Example:

```
<p>Rabbits often live in burrows.</p>

<aside>
  <h4>Did you know?</h4>
  <p>A burrow is another name for a rabbit hole.</p>
</aside>
```

If you're wondering whether the <aside> tag is appropriate, ask yourself whether the information you'd like to pull aside is needed to understand the main content. If removing the content will negatively impact a person's understanding of the page content, <aside> probably isn't appropriate. If removing the content won't impact someone's understanding, then give it a whirl!

You'll often find the aside element nested inside the article tag, but it can be used elsewhere as well. When used inside the article element, the content in the aside tag should relate to the article. When used elsewhere, the content inside the aside tag should relate to the Web page or the site as whole (like a group of sidebar links or advertisements).



Smashing Magazine uses the <aside> element in this complementary sidebar, which includes a search bar and advertisements.

```
477 <aside id="wpsidebar" class="sb" role="complementary">
478         <div class="col side">
479                 <div class="mise sf clearfix" >
480         <form role="search" id="search_2" method="get"
action="http://www.smashingmagazine.com/search-results/"
target="_top">
481     <label class="sl" for="searching_2">Search on Smashing
Magazine</label>
...
567                 <div class="clearfix">
568   <div class="oa_zone--ad sidebar_sky" id="sidebar_sky-3"
data-ad-name="Sidebar 3" data-ad-zone="40"></div>
569 </div>
570
571         </div>
572 </aside>
```

The aside element

The <footer> Element

The <footer> element contains footer information for a Web page or section. It can be used in place of and generally contains copyright information, information about the site creator or authors, contact information, or secondary site navigation links.

Example:

```
<footer>
        <p>Copyright 2015 Rabbit Burrow Studios.</p>
        <p>Contact:  <a href="mailto:rabbit@website.com"</a>.</p>
</footer>
```

Just like you're allowed multiple <header> elements in one document, you can include multiple <footer> elements, as long as they're used correctly. For example, each <article> or <section> on a page could include its own <footer> tag.

Lost World's Fairs takes advantage of a large <footer> area, which includes author names and bios. The information in the <footer> element can be simple or complex, as long as it comes at the end of a section or at the bottom of the site.

```
78          <footer id="footer">
79                  <div class="container clearfix"><img class="fom"
src="img/friendsofmighty.png" alt="friends of mighty"/>
80                  <div class="grid_6 mighty_box">
81
82                          <div class="bio_row clearfix">
83                                  <div class="grid_2 alpha">
84                                  <img class="biopic" src="img/jasonsantamaria.png"
alt="Jason Santa Maria"/>
...
116                 </div><p id="copyright">&copy; 2010 <a
href="http://madebymighty.com">Friends Of Mighty</a><br/><a
href="http://typekit.com/colophons/ops5zjr" class="footer_tk_link">View the fonts used
on this page</a></p>
117
118         </footer>
```

The footer element

Those are just some of most important semantic tags introduced in HTML5. For a complete list of HTML5 elements, check out the helpful Element Index at HTML5 Doctor.

Adding Audio and Video Using HTML5

important!

Only the newer versions of the main browsers support HTML5 audio and video.

An exciting new development in HTML5 is the ability to play audio and video without using third-party plugins such as Flash or QuickTime. HTML5 has defined `<audio>` and `<video>` tags that enable developers to easily encode players that play these media files natively within Web documents.

Despite this new potential promise for headache-free media deployment on the Web, there are a few snags.

First, there is currently limited browser support for these elements. Only the newest versions of Firefox, Safari, Chrome, and Internet Explorer (as well as the iOS and Android) support `<audio>` and `<video>`. And while we would like to imagine a world where everyone is using the newest browser, browser usage statistics show us that simply isn't true. For example, Internet Explorer users are known for using a version one or two generations old. Second, even for browsers that support `<audio>` and `<video>`, different browsers support different audio and video formats.

A third factor to bear mind is that the advent of video-sharing Web services such as YouTube and Vimeo has changed the dynamic in video and audio publishing on the Web. Both Youtube and Vimeo present compelling advantages to self-publishing media content:

- It's easy to upload/publish.
- Your movies will play on almost any system.
- Your movie can reach a wide search audience and be found through searches for related content.
- Movies published on YouTube or Vimeo can *also* be published on your site, by grabbing provided code.

Keeping this mind, it's important to learn how to make use of these nifty HTML5 features, as they may become more important over time, and certain clients/projects may require it.

The <audio> Element

note

The audio element has a controls attribute that enables you to create a simple interface.

Adding audio to a page couldn't be simpler with the HTML5 `<audio>` element. Download the Lesson 4 folder. Create a new basic HTML file inside it called audio.html and place it inside the audio folder that you downloaded. Remember to add your HTML5 doctype and charset. Good habits!

There is an MP3 file in that folder that you will place in your new HTML page. Write the following code in the body of your new page.

```
<audio controls src="iwannago.mp3">
    Browser not supported.
</audio>
```

important!

Not all browsers support all audio file formats, also called codecs.

Let's break down the simple code you just added. We have the <audio> tag here along with some essential attributes, the most important of which is the src attribute. Similar to its use in an img tag, this defines the URL of the audio file to be played, in this case the MP3 file in your download folder.

In the above example, we've also included the **controls** attribute, which displays a simple user interface for your media. The exact appearance of the interface varies according the browser.

The player on the left appears in the Safari browser when the controls attribute is used. The player on the right is from Firefox.

If you didn't want to include a set of controls, you would most likely include the **autoplay** attribute instead, which automatically plays the audio file in the browser when it is loaded.

How about that "Browser not supported" line in the code example? This is an error message for the user in case he or she is using a browser that doesn't support the <audio> element.

Codec Support

Unfortunately, not all browsers support all audio file formats (or codecs). The MP3 file in the above code, for example, won't play in older version of Firefox. The chart below shows the browser support for popular audio file formats at the time of this writing.

	Firefox	Safari	Opera	Chrome	IE	Microsoft Edge
MP4	No Native Support	Yes	No Native Support	Yes	Yes	Yes
Ogg Vorbis	Yes	No Native Support	Yes	Yes	No	Yes
WAV	Yes	Yes	Yes	Yes	No	Yes

important!

Users should always have some control over the playback of your media.

As you'll notice, there isn't one audio codec that all browsers support. So, in order to ensure cross-browser support, we have to encode our audio in more than one codec, and reference both audio files within the <audio> tag. A good choice is to include both an MP3 (.mp3) and an **Ogg Vorbis** (.ogg) file (which is an open-source audio codec similar in size and quality to MP3). WAV files are not as practical, as this format can be much larger than the other two.

Your download folder contains an Ogg Vorbis file of the audio. Adjust the code on your page to reference both the .mp3 and the .ogg files. The code should look something like what you see below. Note that I have also added the autoplay attribute:

Important: Users should always have some level of control over the playback of your media.

```
<audio autoplay controls>
        <source src="iwannago.ogg" />
        <source src="iwannago.mp3" />
```

important!

Make sure you test any audio or video you add to a page in multiple browsers.

```
        Browser not supported.
</audio>
```

(Did you notice the self-closing <source> tags above?)

The browser will try to play the first audio file it comes to. If it can't play the first file, it will skip it and move on to the next file.

Adding Your Own Controls with JavaScript

If you don't like the default controls for your audio player (or, if you just want the functionality to be more consistent across browsers), you can create your own quite easily using JavaScript. Don't worry! It may sound advanced, but we're going to work with just some very straightforward controls.

First, add the following button below your closing <audio> tag (on the next line):

```
<input type=button value="Play/Pause" tabindex="0" />
```

This line of code displays a simple "Play/Pause" button below the audio player.

The Play/Pause button in Firefox, under the controls

Now, we have to attach some actions to this button so something happens when the user clicks it.

Include the following attribute in the existing input tag (in bold below):

```
<input type=button onclick="playPause()" value="Play/Pause"
tabindex="0" />
```

This tells the browser to run a piece of JavaScript called "playPause" when the button is clicked. So, now, we'll need to include that script inside the <head> tags of our HTML document. Remember, you don't actually need to understand JavaScript right now to use it. This JavaScript can be used and reused as often as you like for your audio pages without any modifications.

tip!

You can include backup options for users whose browser doesn't support video.

```
<script>
        function playPause() {
      var myAudio = document.getElementsByTagName('audio')[0];
      if(myAudio.paused)
      myAudio.play();
      else
      myAudio.pause();
        }
</script>
```

This script plays the audio if it is paused, and pauses the audio if it is already playing. How does it do this? It assigns a variable to the <audio> tag (called "myAudio"). Then there is a simple if/else statement that essentially says, "If myAudio is paused, play it; if myAudio is playing, pause it."

You can add an opening frame to a page with the poster attribute.

Once we've created the button and added the JavaScript, we really don't need the default browser controllers anymore. So we can just get rid of the controls attribute in the `<audio>` tag, leaving only the autoplay attribute:

```
<audio autoplay>
```

Try out your HTML page in at least two browsers now. If you have trouble, check out the finished version of the page in your download folder, which also includes some CSS styling. When you gain more scripting experience, you can create more elaborate JavaScript controls than these. But this example shows how easy it is to add your own customized controls.

Important: You do not need to know how to write your own JavaScript to use the audio and video controllers shown in this lecture.

The <video> Element

Before we start with a page that incorporates video, let me give you some best practice tips for using video so that you don't scare your users away. (These apply to audio as well.)

- Don't autoplay media files unless you want to annoy people. You know, old people like myself.
- Video files are rather large, so keep their use to a minimum unless they are necessary to your site. If you are serious about putting videos online, look into movie compression and video codecs to squeeze more movie out of your files.
- Provide a backup option in case a visitor's browser doesn't support native video support.

Create a new page called video.html and place it in the video folder you downloaded for this lecture.

The syntax and available attributes of the `<video>` element are very similar to those of the `<audio>` element. So let's dive right in and try a simple example.

```
<video controls>
   <source src="Kissing.mp4" type="video/mp4" />
        Browser not supported.
</video>
```

We've included the default control set that the browser provides (the `controls` attribute), and have sourced an MPEG-4 video file (Apple's popular video format). Try playing it now, by opening your video.html page in the Safari browser.

The video playing in the Safari browser with basic browser controls.

Codec Support

As with audio, different browsers support different video formats.

Every video file you play is actually a "container" that encases both an audio and video codec. For example, an MP4 video file is a container that usually includes an H.264 video codec and an AAC audio codec.

While different containers can encase various combinations of audio and video codecs, for our purposes we'll narrow our consideration to the two widely supported video containers: MP4 and Ogg. As I just noted, MP4 contains the H.264 video file format (which is also used in Blu-ray) and the AAC audio file format. Ogg is an open-source solution that contains a Theora video file and a Vorbis audio file.

Here's what browser support for the two major video codecs looks like at the time of writing.

	Firefox	Safari	Opera	Chrome	IE	Microsoft Edge
MP4	Yes	Yes	Yes	Yes	Yes	Yes
Ogg	Yes	Yes	Yes	Yes	No	No

While there isn't any one format that works on all browsers, including both file formats should provide a working video for all of these browsers:

```
<video controls>
        <source src="Kissing.ogv" type="video/ogg">
        <source src="Kissing.mp4" type="video/mp4">
        Video Not Supported
</video>
```

Setting Additional Attributes

A great feature of adding videos in HTML5 is that you can add your own opening frame for the video. This is called a **poster frame**, which displays if the browser can't play the video, or while the video is loading.

In your video folder is a file called poster.jpg. This is simply a graphic created in Photoshop at the same dimension as the video itself. To add this to your video, simply include the poster attribute in the <video> tag, and have it point to the poster image file:

```
<video controls poster="poster.jpg">
```

Open the page in a browser to see the results.

The poster frame in Firefox

In addition to adding the poster frame, you can further customize your video. First, you can adjust the size of your video by setting the height and width in the <video> tag:

```
<video controls poster="poster.jpg" width="320" height="240">
```

And if you just can't get enough of this kissing clip, you can loop the video so that it plays over and over again:

```
<video controls poster="poster.jpg" width="320"
height="240" loop>
```

Controlling Your Video with JavaScript

We can create custom controls for our video, much like we did for our audio clip. In fact, the code is nearly identical and can be used as-is without any fancy JavaScript expert modifications.

As you did for audio, start off by adding a play/pause input button just under your closing video tag.

```
<input type=button onclick="playPause()" value="Play/Pause"
tabindex="0" />
```

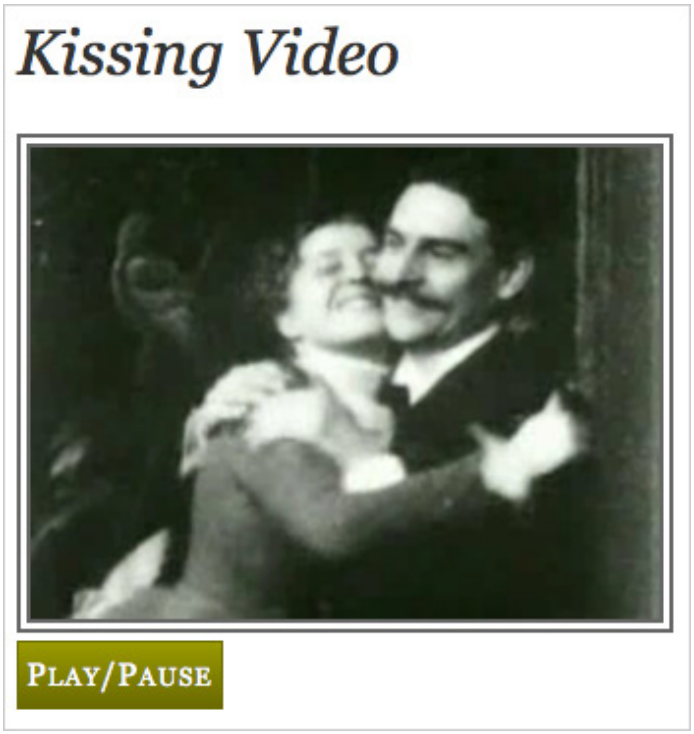
Then, add the following JavaScript in the head of the HTML document:

```
<script>
    function playPause() {
      var myVideo = document.getElementsByTagName('video')[0];
      if(myVideo.paused)
      myVideo.play();
      else
      myVideo.pause();
```

```
    }
</script>
```

The only crucial difference between this script and the one we had for the audio player is that we have to make sure to assign the variable to the `<video>` (not the `<audio>`) element, which you can see in the fourth line. Then you can remove your controls attribute from the `<video>` tag, because you now have your own JavaScript controller.

If you'd like to put your knowledge of CSS to use, you'll see that further customizations can be made to the overall style of the input button and the page itself. To customize my controller even further, I added my own button graphic, subtle border, and CSS-styled text. If you're feeling adventurous, check out my code in the video_finished.html file contained in your video download folder.



A highly customized controller

On Browser Support

It's true: Concerns about browser support can sometimes throw a wet blanket over our eagerness to embrace new features.

So just to be safe, and to accommodate the widest possible variety of users, you will want to provide alternative paths to any audio and video content that you include on your pages. For example, include a fallback video or audio player such as a Flash player, or at least a direct link to the file itself within the `<audio>` and `<video>` tags. While this may be a bit of a hassle—especially considering how wonderfully simple the HTML5 code is—this ensures true cross-browser compatibility for your media players.

And on that note, let us now turn our attention to some other things we can do to ensure that pages look as good as possible, for as many users as possible.

Troubleshooting Web Pages

Test on Multiple Browsers

While looking at your page in a browser does not necessarily alert you to coding errors, getting your page to look right in a browser is certainly an essential step in Web page design. Unfortunately, completing this step is often easier said than done.

The most frustrating thing about designing Web pages is the varying ways in which different browsers render the same code. Even the same browser in a different operating system can interpret a page differently. Despite the existence of Web standards that state how browsers should interpret and render code, different browsers comply with these standards to varying degrees.

Internet Explorer, the browser that until a few years ago made up the majority of the computer user base, is, ironically, notorious for its non-compliance with Web standards, though each version is typically is a major step up from its predecessor, and its successor Microsoft Edge is improved.

What this means, practically speaking, is that a page that looks great in one browser may appear "broken," or look just awful, in another, even if the code is technically valid. The Web designer is forced to test a site in various browsers (on various platforms) and sometimes employ various hacks and workarounds to get the site to look relatively consistent across browsers, as you saw in our audio and video examples. Hopefully, as Web standards compliance improves, the need for such rigorous testing and use of hacks will lessen.

In the meantime, it is still imperative that you test your HTML documents on various browsers, various browser versions, and, ideally, on different operating systems even if the page validates (conforms to Web standards).

It is best to start with browsers that more strictly adhere to Web standards, fix what is necessary to get the page working, then move on to less compliant browsers. If you start off by accommodating your code to browsers that don't interpret the code correctly to begin with, you tend to end up with code that is not clean and not compliant.

Start coding Web designs for browsers like Chrome, Firefox, and Safari, which are very standards-compliant. Internet Explorer is known to flout Web standards, so you'll want to test your pages on a couple of versions of IE once you get them working in Chrome, Firefox, and Safari.

Test your work in Chrome, Firefox, Safari... then Internet Explorer.

Getting Access to Different Browsers (on the Cheap)

The bigger your Web design projects get, the more you will want to add cross-platform and cross-browser testing into the mix. For example, a client may be able to give you Google Analytics data on the main operating systems and browsers, as well as mobile devices, used by site visitors.

Being able to test different browsers on different platforms can be a challenge. Internet Explorer, for example, is not available for Mac or Linux, and you generally can't install more than one version of Internet Explorer on Windows.

Seeing how pages look on different operating systems can be problematic for the majority of us who only use one computer.

There are, however, workarounds to see your pages in various browsers:

- **Install a virtual machine or emulator.** Using Parallels for OS X or Wine for Linux allows you to run Windows programs while running your native operating system. (And if you use Linux, check out the IEs4Linux project, which installs several versions of Internet Explorer on systems that run Wine.)
- **Use a Web-based service.** The media site Mashable maintains a great list of Web based services for cross-platform and multiple browser testing. Many of them have free versions you can use, and as you get bigger projects or work in teams you might consider purchasing a product.
  - Browsershots.org provides you with screenshots of your page running in various browsers, on various operating systems.
  - NetRenderer.com provides browser views of your pages for Internet Explorer 5.5 through 11, Useful for Mac users!
  - Spoon.net provides a simple looking browser sandbox for PC users.

In the next lesson

- Learn how to create and style horizontal navigation using HTML and CSS.
- Learn how to use CSS height and width properties to standardize layout proportions.
- Learn how to add Web fonts to your site's HTML and CSS.

Coming Up Next:

Discussion
Share your thoughts and opinions on semantic HTML in the Discussion area.

Exercise
Apply semantic markup and coding as you style an informational Web page.

HTML and CSS | Semantic Layouts

sessionscollege
FOR PROFESSIONAL DESIGN

Exercise 4

Semantic Layouts

In Lecture Four, you learned how to use semantic elements to structure a Web page. You also learned about the HTML5 multimedia tags that were created to help streamline the sharing of video and audio on the Web.

This exercise will give you practice in using semantic elements and embedding video using HTML5 on a Web page. First, you'll add semantic and non-semantic markup to the page. Then, you'll add CSS styles and a video to give the page a bit of color, style, and interactivity.

Do you have a nose for HTML5?

Performance Objectives

- Mark up a page using semantic and non-semantic HTML elements.
- Style the page with an internal stylesheet.
- Apply styles to the typography, color, and links.
- Use HTML5 to add a video to the page.

note

Assignments are evaluated for creativity, technical proficiency, and understanding of concepts covered in the lecture.

Project Brief: Semantic Smell-O-Vision

In the first part of this assignment, take the content from the text document provided below and transform it into an HTML5 document with a fixed width layout.

The text is from the Project Gutenberg EBook of The Art of Perfumery, by G. W. Septimus Piesse, published in 1857. The text includes headings, paragraphs, pullquotes, links, and lists, so it will challenge you to explore the concepts you've been learning.

Open a new HTML document in Sublime and begin with the following layout. You'll customize the design from here, but let's start with some basics:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Untitled Document</title>
<style type="text/css">
#frame {
width: 800px;
margin: 0 auto;
}
</style>
```

```
</head>
<body>
<div id="frame">
.Page content goes here.
</div>
</body>
</html>
```

Remember to title the document and add the document text between the #frame div tags. You'll notice that this exercise uses an internal style sheet. Since you're only formatting one page, you'll write the CSS rules into the top of the document. In most other projects, you'd use an external style sheet.

Mark Up the Document

Start by applying HTML5 semantic markup to the content. You may wish to print out a paper copy of the article and mark up the components you'll want to style. Recall that HTML5 provides <header>, <nav>, <section>, and <footer> tags, and that you can still use and tags as needed. The HTML5 elements you'll be using in this document include:

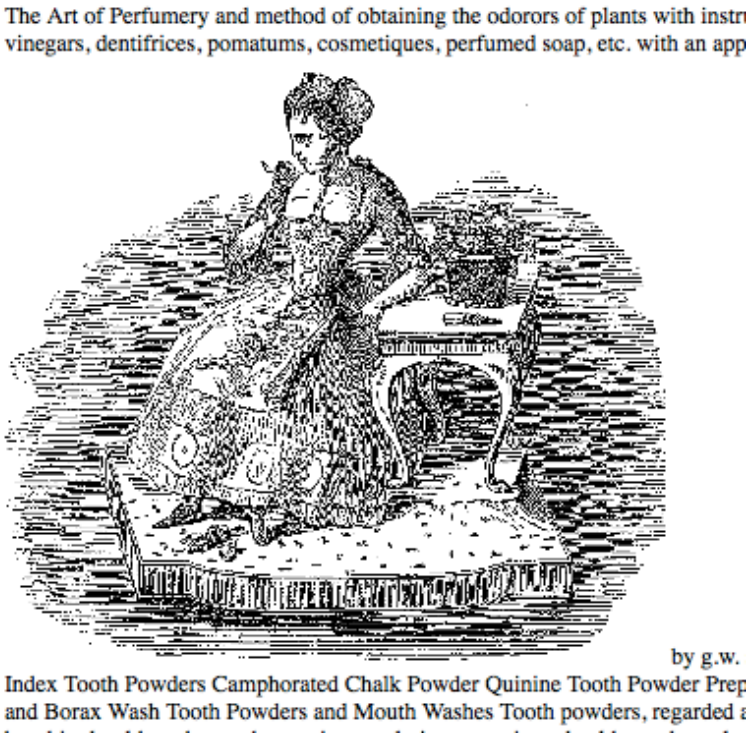
- <header>
- <nav>
- <article>
- <section>
- <aside>
- <footer>

1. Header

Wrap <header> tags around the opening text, beginning with the title, "The Art of Perfumery," and ending after the author's additional credits ("author of The Odors of Flowers...").

In the exercise 4 downloads folder, you'll find an image entitled perfume.png. Insert this image in the header above the author name. Remember to include alt text so the image validates.

```
<img src="perfume.png" alt="a noblewoman sits smelling a rose">
```



Let's get these semantic elements in here quickly so we can start to organize this text!

2. Nav

You'll often find the <nav> element inside of the <header> element, but because we have such a large introductory header area, let's place the nav between <section> tags.

```
<section>
<nav>
Index

Tooth Powders
Camphorated Chalk Powder
Quinine Tooth Powder
Prepared Charcoal Powder

Mouth Washes
Eau Botot Wash
Botanic Styptic Wash
Myrrh and Borax Wash
</nav>
</section>
```

We'll come back and format the navigation links later. For now, let's get our major semantic elements organized on the page.

3. Article

We'll break down this page into individual sections, but first, let's wrap the main text content in an <article> tag, beginning above "Tooth Powder and Mouth Washes" and ending before the Project Gutenberg footer credit.

```
<article>
Tooth Powders and Mouth Washes
```

```
...

</article>
Text and image courtesy Project Gutenberg
```

4. Sections

Now we can subdivide the article into two major sections: Tooth Powders and Mouth Washes. Wrap opening and closing `<section>` tags around the entire Tooth Powders section and around the entire Mouth Washes section. The final section tag should end before the closing article tag.

5. Aside

This page wouldn't be complete without some helpful pullquotes about 19th century dental health. **Pullquotes** are longer lines that repeat key phrases or tips that are stated in the body text. In magazines, they are often "called out" of the text and italicized with quotation marks to attract the attention of a browsing or casual reader.

Wrap the following helpful reminder in `<aside>` tags and choose an appropriate placement for it somewhere in the Tooth Powders section:

> "On account of the volatility of camphor, the powder should always be sold in bottles, or at least in boxes lined with tinfoil."

Follow the same steps with this useful tip in this Mouth Washes section:

> "All these tinctures should be made with grape spirit, or at least with pale unsweetened brandy."

There's a bit of an art to the placement of pullquotes. They should support the main text, but they shouldn't appear so close to the original quote that they feel redundant. Use your judgement and feel free to move the quotes around later when you begin styling your document.

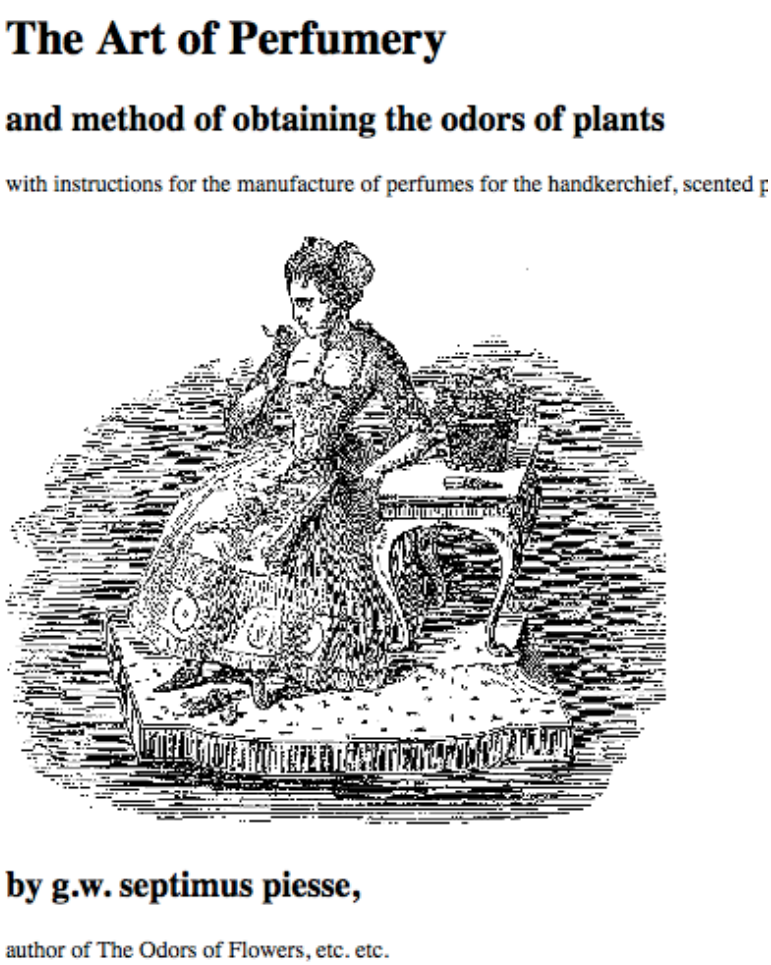
6. Footer

To complete the HTML5 semantic markup, wrap the Project Gutenberg credit line at the bottom of the page in `<footer>` tags.

Look back over the HTML5 tags to make sure all tags have been opened and closed in the right order. After you're happy with the overall structure, it's time to go through and add markup to the rest of the page.

7. Headings, paragraphs, lists, and links

There are three levels of headings in the document: `<h1>`, `<h2>`, and `<h3>`. Wrap the page title in `<h1>` tags and use your judgement in determining the heading hierarchy for the rest of the page. Enclose all paragraph text in `<p>` tags.

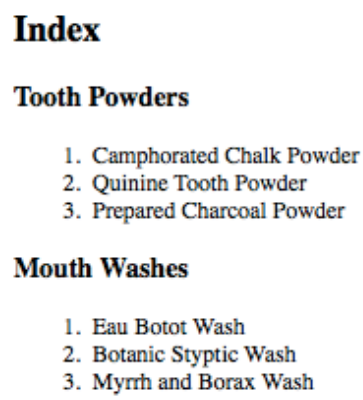


Things are looking better with some headings and paragraph tags.

Next, mark up the lists. The <nav> section should include two ordered lists, formatted like so:

```
Tooth Powders
 <ol>
<li>Camphorated Chalk Powder</li>
<li>Quinine Tooth Powder</li>
<li>Prepared Charcoal Powder</li>
</ol>

Mouth Washes
 <ol>
<li>Eau Botot Wash</li>
<li>Botanic Styptic Wash</li>
<li>Myrrh and Borax Wash</li>
</ol>
```

The list headings, "Tooth Powders" and "Mouth Washes" should be placed in heading tags.



Index

Tooth Powders

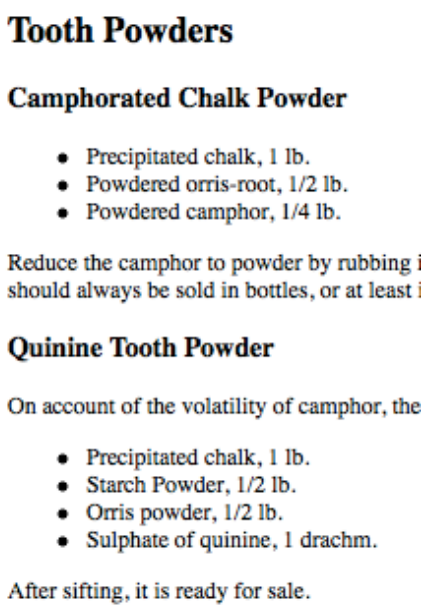
1. Camphorated Chalk Powder
2. Quinine Tooth Powder
3. Prepared Charcoal Powder

Mouth Washes

1. Eau Botot Wash
2. Botanic Styptic Wash
3. Myrrh and Borax Wash

Lower on the page, the lists of tooth powders and mouth washes should be wrapped in unordered lists, like so:

```
<h3>Camphorated Chalk Powder</h3>

<ul>
<li>Precipitated chalk, 1 lb.</li>
<li>Powdered orris-root, 1/2 lb.</li>
<li>Powdered camphor, 1/4 lb.</li>
</ul>
```



Tooth Powders

Camphorated Chalk Powder

- Precipitated chalk, 1 lb.
- Powdered orris-root, 1/2 lb.
- Powdered camphor, 1/4 lb.

Reduce the camphor to powder by rubbing i
should always be sold in bottles, or at least i

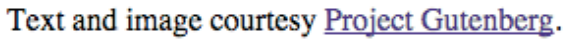
Quinine Tooth Powder

On account of the volatility of camphor, the

- Precipitated chalk, 1 lb.
- Starch Powder, 1/2 lb.
- Orris powder, 1/2 lb.
- Sulphate of quinine, 1 drachm.

After sifting, it is ready for sale.

Finally, be sure to add a link back to the Project Gutenberg Web site in the footer: http://www.gutenberg.org.



Text and image courtesy Project Gutenberg.

Note: If you're seeing unexpected gaps between the block-level elements on your page, you can use CSS reset styles to override the default margins from your browser's user agent style sheet. Apply resets to the body element, headings, paragraphs, and listsâ€”then go through and apply your own custom margins and padding.

Add CSS Styles

To style your page, you will be adding CSS rules (selectors, properties, and values) to the head of your document, as shown at the beginning of this exercise.

1. Adjust Column Width and Margins

We've already set up an ID attribute to style all the page content. The first thing you might do is adjust column width and margins to your liking. The page is set to 800 pixels wide. Wide text columns can be tiring on the eye, so experiment with a narrower column. Also, you'll note that margins are set to 0 at the top and bottom and auto (equal reset) on the left and right. Experiment with those settings and use individual margin settings if you wish to position your text differently.

2. Choose Font-family Styles/Sizes for Headers and Paragraphs

Now it's time to select your fonts. You might try contrasting styles for headers and paragraphs (using serif for one and sans serif for the other). Remember that you can specify more than one font-family and combine selectors if you want to:

```
h1, h2, h3 {
font-family: Verdana, Arial, sans-serif;
}
p {
font-family: Georgia, "Times New Roman", serif;
}
```

You can mix and match type styles using the default system fonts on your computer, or find interesting combinations using CSS Font Stack.

Next, select your font size. You'll recall this can be done many different ways, but that designers often use pixels or ems, like so:

```
h1 {
font-size: 24px;
}
```

Also apply styles to your lists and `<aside>` pullquotes. Start by setting them in the same font-family and size as your body text, then experiment. Pullquotes should be larger than the body text to attract attention.

You'll also recall that you can set font-weight, font-style, font-variant, and text-decoration. Pullquotes might be italicized using the font-style: oblique; option. Since pullquotes should look separate from the text, you can play around with setting them to text-align: right; or center.

5. Letter-Spacing and Line-Height

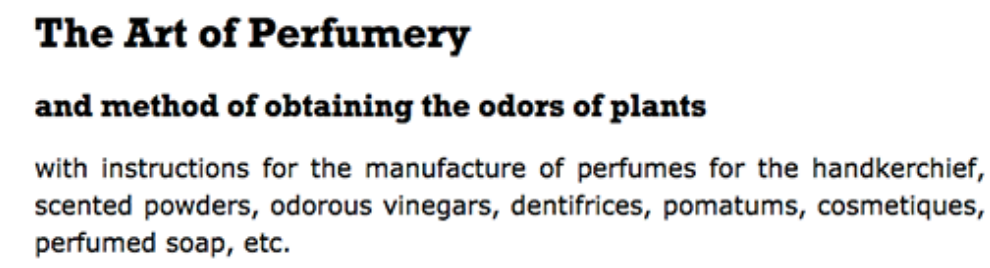
Now let's make sure that the letter-spacing in headings is effective and the line-height in our paragraphs makes the text readable. You'll recall that the spacing between letters in headings can be globally adjusted:

```
h1 {
letter-spacing: 2px;
}
```

Experiment with positive and negative values until your headings look punchy. Then apply line-height to various pieces of body text, starting with paragraphs:

```
p {
line-height: 16px;
}
```

Experiment with different values until the text has enough space around it to "breathe" without losing the visual coherence of each paragraph.



I'm using different line heights for the headings and paragraphs here. Play around until you find what works for your design

6. Style Lists

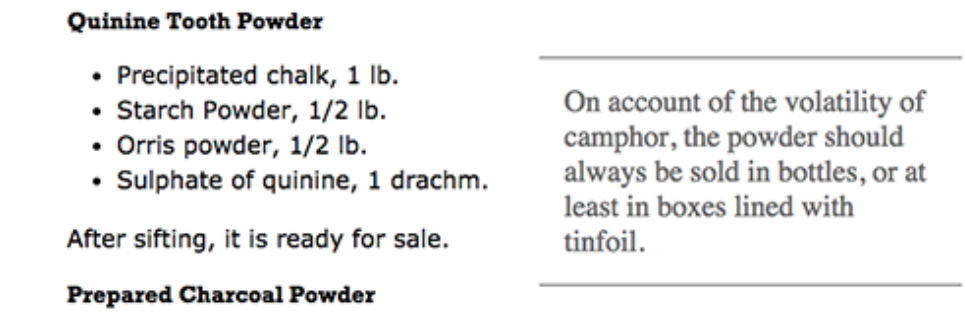
Next, it's time to apply some styles to lists. You'll recall that lists can have unconventional bullets:

```
ul {
list-style-type: square;
}
```

7. Style Pullquotes

OK, time to style the `<aside>` pullquotes. There are lots of great ways to style pullquotes, and I encourage you to get creative with them. Define a width, choose appropriate text alignment, add borders, margins, padding, and adjust the font family and size. Here's an example:

```
aside {
width: 400px;
margin: 0 auto;
padding: 20px;
border-left: 3px solid #690;
font-style: oblique;
}
```



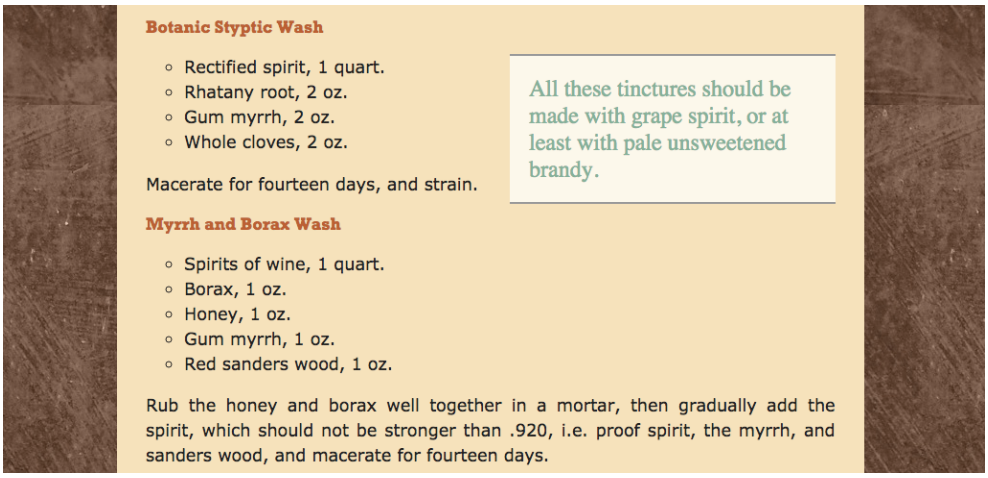
I've added a right float to the pullquotes so they're set apart from the flow of the text.

8. Background and Text Color

Now it's time to work on the background color and text color. Remember—you can set the background color and text color for the page body, headers, paragraphs, as well as the pullquotes.

```
body {
        background-color: #444;
        color: #222;
}
```

You can also use background graphics in block-level elements. Remember, if you're stuck for color combinations, try looking up hex colors at one of the many color resource sites on the Web. You might start by choosing a dominant color then thinking about two or three complementary colors that will work well with it. Good places to start are colourlovers.com and colorcombos.com.



Using complementary colors

I would encourage you to experiment but avoid the usual traps: bright colors, clashing colors, too much contrast, or insufficient contrast. You may experiment with background graphics too, but it's essential that you keep your page easy to read. I'm looking for appropriate, effective design choices in this assignment.

9. Style Links

Lastly, you'll need to style your links, particularly if you used a page background. You can target this specific state of a link by using pseudo-classes. There are four pseudo-classes for the anchor element:

- `<a:link>` (unvisited link)
- `<a:visited>` (visited link)
- `<a:hover>` (moused-over link)
- `<a:active>` (selected link)

You can set the attributes for each link state at the top of your document, like so:

```
a:hover {
text-decoration: none;
color: red;
background-color: #ffff99;
}
```

Do the site colors work well together? Do the link styles function as expected and do the links take you to the right place? If you feel good about the overall flow, then it's time for the final step—embedding a video on the page.

Add Video to the Page

Insert the video using the HTML5 <video> tag and the control features of your choice, making sure that both versions are provided, as you learned in the lecture. Only the basic controls are required.

When you insert the video, you can either use absolute links (links that begin with http://) to the video files or you can use relative paths that point to the files on your own server (for example, videos/perfume.mp4).

The videos are found in the exercise 4 downloads folder for this exercise. Not all servers are equipped to handle the Ogv format, so if you run into trouble getting the Ogv version to play, please use the absolute links instead.

Place the video near the bottom of the page, right before the footer begins.

Optionally, you can try out some of the JavaScript in the lecture to add custom controls and/or create a poster frame graphic for the video.

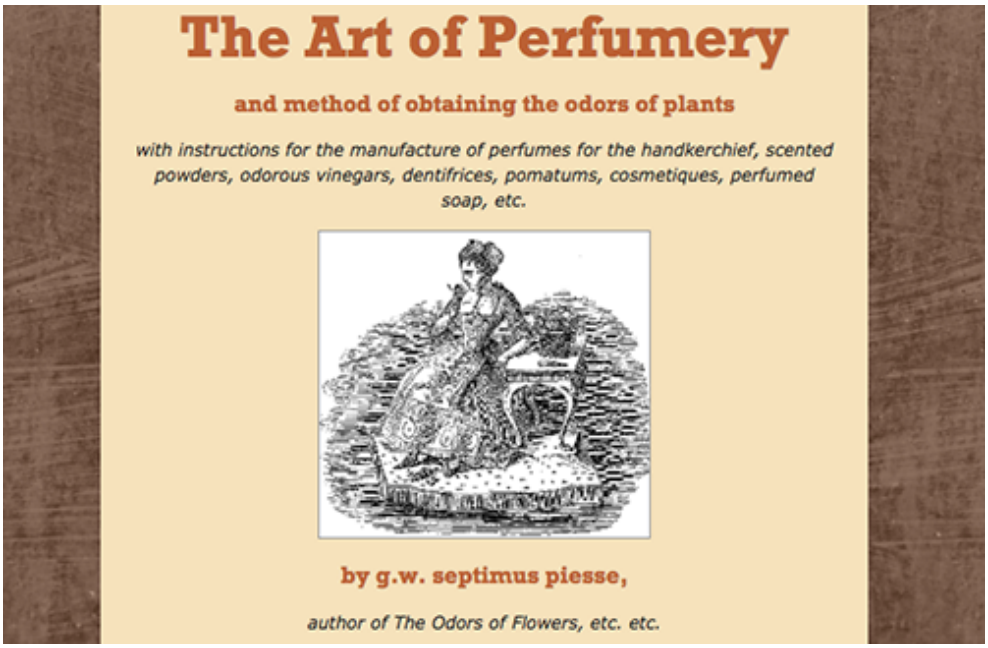
Depending on your page layout, you'll probably want to center both the perfume video and the header image at the top of the page. To do that, we have to add style rules that instruct the browser to treat these as block-level elements, instead of inline elements (elements inside a block). The CSS to accomplish that looks like this:

```
img, video {
display: block;
margin: auto;
}
```

That CSS snippet tells the browser to treat both the image and video as blocks, and to apply equal margins that center the content on both sides.

When you're finished, upload the page (with image and video files in their location) to your Web host and run through the validator again. Be sure that everything is valid and working properly. Remember to test your work in multiple browsers as well!

Below is one possible version of this page. What I'm looking for in your assignment is solid technical coding combined with the ability to create an engaging, readable layout. I look forward to seeing your work!



Describe Your Experience

When learning code, it's important to reflect on your experience. When you post your work, include your answers to the following questions:

1. Which aspects of semantic coding do you find helpful or confusing, if any?
2. What applications do you see for applying video and audio using HTML5?

We'll be moving onto different projects later in the course. For now, I look forward to seeing your work!

Grading Criteria:

What your instructor expects you to do:

☐ Demonstrate the ability to mark up a page with semantic and non-semantic HTML elements.

☐ Show the ability to use an internal stylesheet to control color, typography, and link styles.

☐ Show the ability to add a video to a Web page using HTML5.

☐ Demonstrate the ability to create clean, accurate, standards-compliant code in your documents.

☐ Demonstrate the ability to create an appropriate, attractive, and readable page design.

How to Post:

Once you're done, go to the Dropbox for this exercise and post your URL for this assignment along with your written comments.

If you have a question before sending your completed exercise for grading, be sure to contact your instructor via the People area.

I look forward to seeing your work!

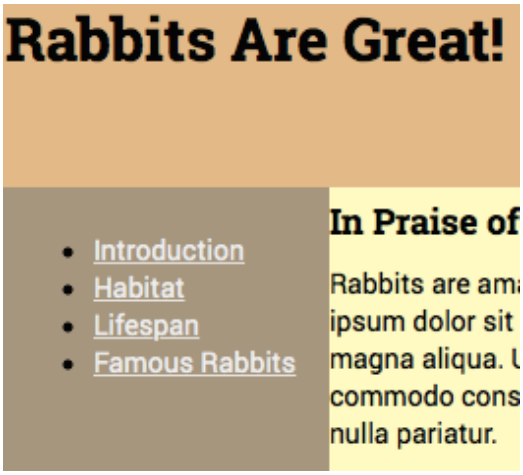
sessionscollege
FOR PROFESSIONAL DESIGN

Lecture 5

Advanced CSS Techniques

Now that you've learned the basics, it's time to take on some of the more nuanced CSS rules. This lesson will cover new techniques for creating navigation bars, using CSS dimension properties to streamline your design, and elevating your site typography with cool, contemporary Web fonts.

Finally, you'll learn how to add CSS3 effects—like rounded corners and opacity—to further customize your work. Some changes may seem subtle, but incorporating these new techniques into your site will have a big impact on the overall design!



Down the rabbit-hole some more!

Learning Objectives

In this lecture, you can expect to:

- Learn how to create and style horizontal navigation using HTML and CSS.
- Learn how to use CSS height and width properties to standardize layout proportions.
- Learn how to add Web fonts to your site's HTML and CSS.
- Learn how to apply simple CSS3 effects.

Horizontal Navigation Menus

In previous lessons, you learned how to create attractive 2-column layouts and vertical navigation menus. But how do we create horizontal navigation?

If you've been wondering how the heck you turn a vertical list-based nav into a horizontal bar, then wonder no more! In this lesson we will give you many tools for styling navigation.

Let's take a look at a few different Web site layouts that utilize horizontal navigation, then we'll dive into the code.

note

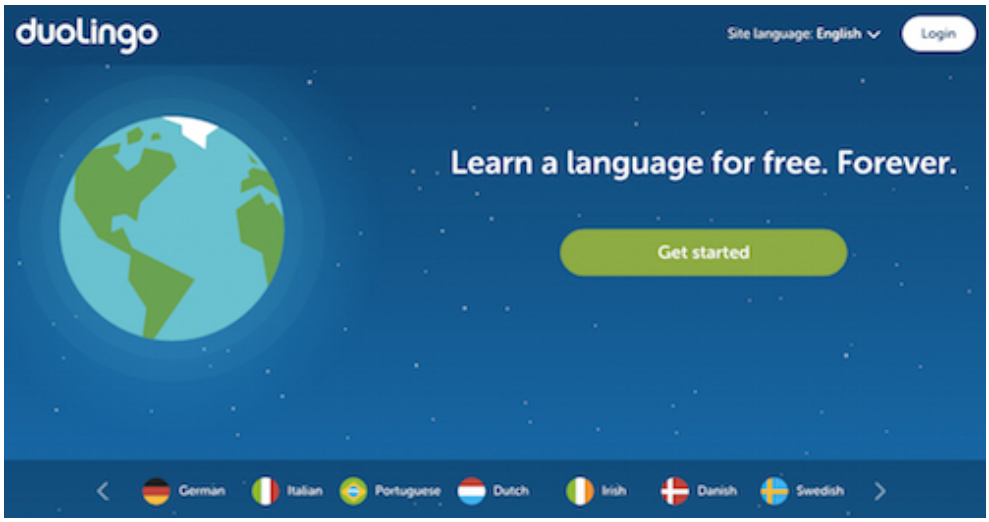
List-based navigation can be styled to display horizontally.

Horizontal navigation at the top of the page

note

Most large sites incorporate multiple navigation menus.

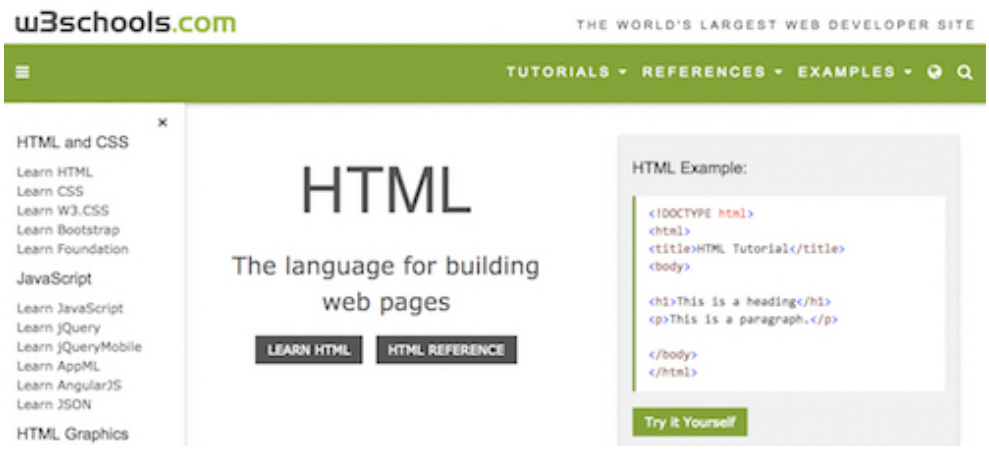
The Mt. Holyoke College Web site displays a large blue header area at the top of the page with a centered horizontal navigation menu beneath a centered logo. There's also a small, left-aligned secondary navigation menu at the very top of the page. You've probably seen many variations of this layout before!



Horizontal navigation at the bottom of the browser window

The Duolingo Web site uses a more unconventional horizontal navigation scheme. Rather than placing a menu at the top of the page, the navigation menu is at the bottom of the browser window, below a minimal home screen.

This isn't a traditional layout, but it works due to the clean simplicity of the site design.



Horizontal and vertical navigation

note

The display; inline; rule removes the line breaks between block-level elements.

W3Schools takes advantage of both horizontal and vertical navigation menus. There's a simple, dropdown horizontal menu featured at the top of the page, aligned to the right. There's also a vertical navigation menu featured in the left sidebar. It's hard to say which is the primary and which is the secondary menu here. They both seem to serve different important functions!

Let's explore how to use CSS to style navigation.

Using the Inline Display Property

The HTML code for a horizontal navigation menu can start off much like a vertical navigation menu. For example:

important!

If you remove link underlines, make sure that it's clear your navigation is clickable.

```
<ul>
  <li><a href="#home">Home</a></li>
  <li><a href="#about">About</a></li>
  <li><a href="#blog">Blog</a></li>
  <li><a href="#contact">Contact</a></li>
</ul>
```

Looks pretty familiar, right? The difference is in the CSS. Particularly, in a CSS rule called "inline" styling. Remember that most HTML elements are, by default, block-level elements? The **display: inline**; CSS rule removes the line breaks between block elements, essentially turning each list item into a series of inline elements that will appear all on the same line.

Here's a simple, vertical navigation menu before adding the display: inline; property:

And here's the same navigation menu with some very basic inline styling:

Ready to give it a try? Open a new HTML document in Sublime. Add the HTML5 doctype, a document head, and wrap the document in the required HTML and body tags. Then, copy/paste the HTML for our sample navigation menu:

note

The HTML5 nav element displays a navigation menu as horizontal by default.

```
<ul>
  <li><a href="#home">Home</a></li>
  <li><a href="#about">About</a></li>
  <li><a href="#blog">Blog</a></li>
  <li><a href="#contact">Contact</a></li>
</ul>
```

If you'd like, you can add a heading, a body background color, and customize the font colors as I did above. It's not required, but I find it easier to troubleshoot a nice-looking design!

Add the following CSS rules to the document head. You're welcome to use an external style sheet, or use an internal style sheet for simplicity:

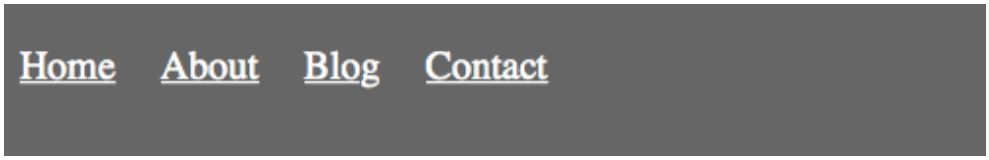
```
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
}
```

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

Now, you may have noticed that the work doesn't end here. As it stands, this navigation menu is very difficult to read.

What not add some padding to the right of every list item to give these navigation links some breathing room?

```
li {
    display: inline;
    padding-right: 20px;
}
```



Ah, that's more like it! What if we wanted to make our horizontal navigation bar look more like, well...a bar? To accomplish that effect, we'll add a background color and some padding to the ul element, like so:

```
ul {
    list-style-type: none;
    margin: 0;
    padding: 10px;
    background-color: #000;
    }
```

Finally, let's remove those link underlines! This is clearly a navigation menu, and visitors don't need those pesky lines cluttering up that lovely navigation bar. We'll do that using a descent selector that targets list item anchor links:

```
li a {
                text-decoration: none;
}
```

Remember to add hover styles to your links when using text-decoration: none. The underlines aren't very pretty, but if you are not using the standard underlines, you need to give site visitors a clear indication of what's clickable text and what isn't.

HTML5 Horizontal Navigation

A neat feature of the HTML5 nav element is that its contents will display inline by default, as long as the individual navigation items aren't placed inside HTML list tags. So, without any additional CSS, this:

```
<nav>
<a href="#home">Home</a>
<a href="#about">About</a>
<a href="#blog">Blog</a>
<a href="#contact">Contact</a>
</nav>
```

Will display like this in your Web browser:

Home About Blog Contact

You'll still have to add padding and margins, and customize your navigation menu in interesting ways, but you can bypass the process of transforming a vertical, list-based navigation into a horizontal navigation bar.

You can decide whether you'd like to use plain anchor links wrapped in an HTML5 nav tag or a list-based navigation scheme. If you decide to use a list-based navigation scheme, wrap the entire list in <nav> tags so your Web page follows good semantic practices.

Advanced CSS Dimension Properties

note

CSS width and height properties can ensure your pages look consistent when the content varies from page to page.

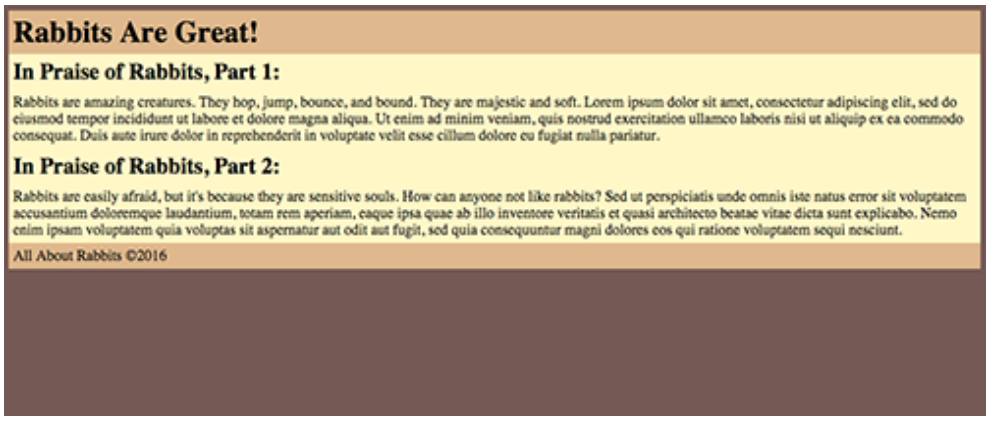
The ability to control the dimensions of block-level elements is an important (and sometimes essential) CSS technique. We briefly introduced the CSS height and width properties in Exercise Three. Let's spend a little more time getting to know these properties.

Click here to download the lesson 5 downloads file and extract the rabbits.html file, which includes the HTML and CSS for a very basic Web site about, well, rabbits. I've included extra spaces between the lines of code for better readability.

Open the rabbits document in Sublime and view the page in your browser. It should look something like this:

note

The min-height property defines the minimum height of an element.

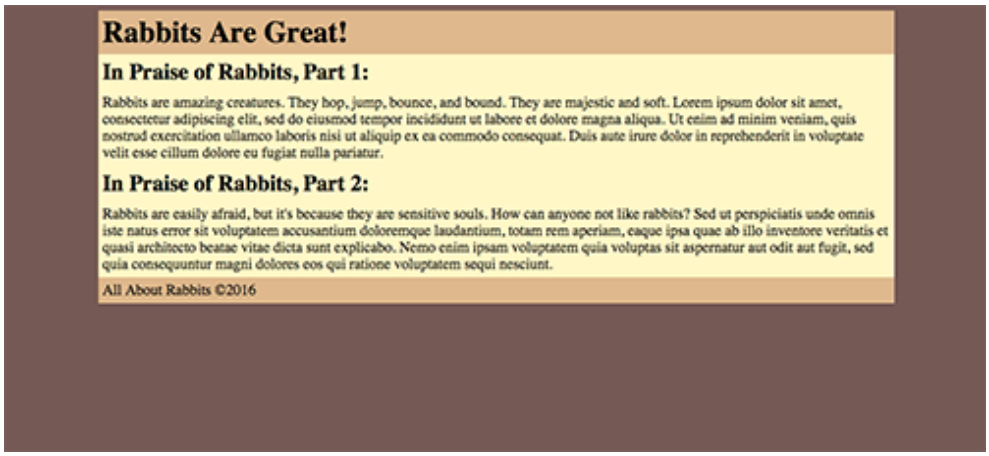


View larger image

Aside from a bit of padding that's already been applied to the headings and paragraph elements, this document is feeling pretty cramped. Without any height property specified, the block-level elements will only stretch as tall as their contents. Without any width property specified, the lines of text will stretch across the full width of the browser window, making them difficult to read. But we can fix that with a few simple CSS rules. Let's start by applying a width and margin to the #frame div, to reign in and re-center the main text content.

```
#frame {
        background-color: #FFFCC1;
        width: 860px;
        margin: 0 auto;
}
```

Then, apply the same width and margins to the **header** and **footer** selectors to create a uniform design.



View larger image

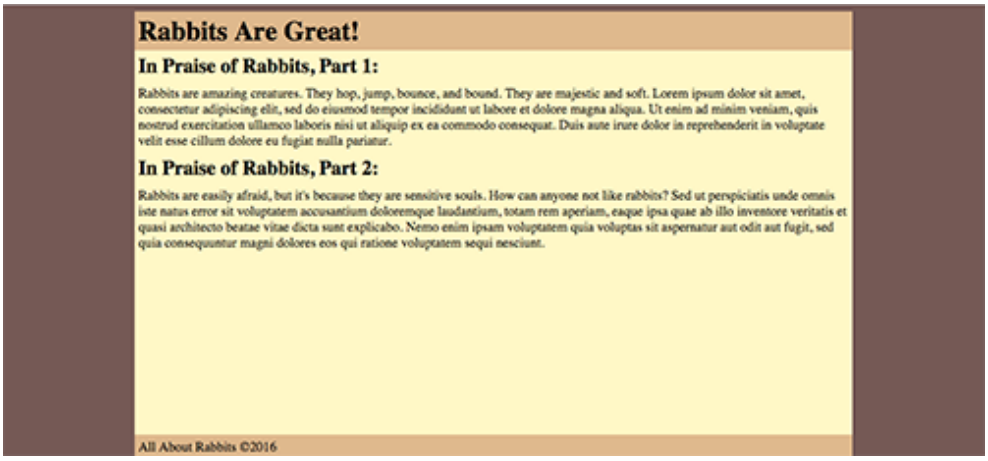
We're on the right track, but the header, footer, and frame div are all looking a bit too short. Even though there's not much text on the page, we want the frame to extend down past the content to fill more empty space. How about this:

tip!

Height and width properties can also be

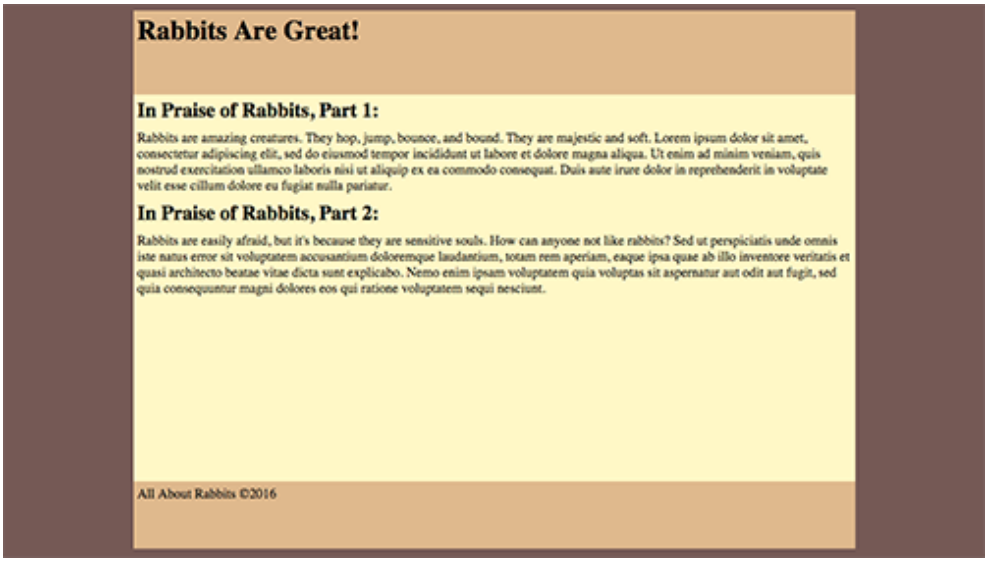
```
#frame {
        background-color: #FFFCC1;
        width: 860px;
        margin: 0 auto;
        height: 460px;
}
```

useful when working with column layouts and floats.



View larger image

Now we're getting somewhere! Apply a custom height to the header and footer selector to give these elements some breathing room as well.



View larger image

A height of 460px is fine for this page on our rabbits site, given that there's not much content on this page. But what if we wanted to include a blog page, with many articles about rabbits that stretch on and on? That's where the min-height property comes into play. Rather than assigning a static height to the element (forcing every frame on every page to stretch down to 460px), min-height defines the minimum height of the element.

```
#frame {
        min-height: 460px;
}
```

A minimum height of 460px applied to the frame div will ensure that the frame div on every page is always at least 460px tall (no smaller), but can extend taller than 460px when there's more than 460px worth of content in the frame.

The complete list of CSS dimension properties includes:

- width
- min-width

note

Web fonts are fonts that reference a font library on the Web.

- max-width
- height
- min-height
- max-height

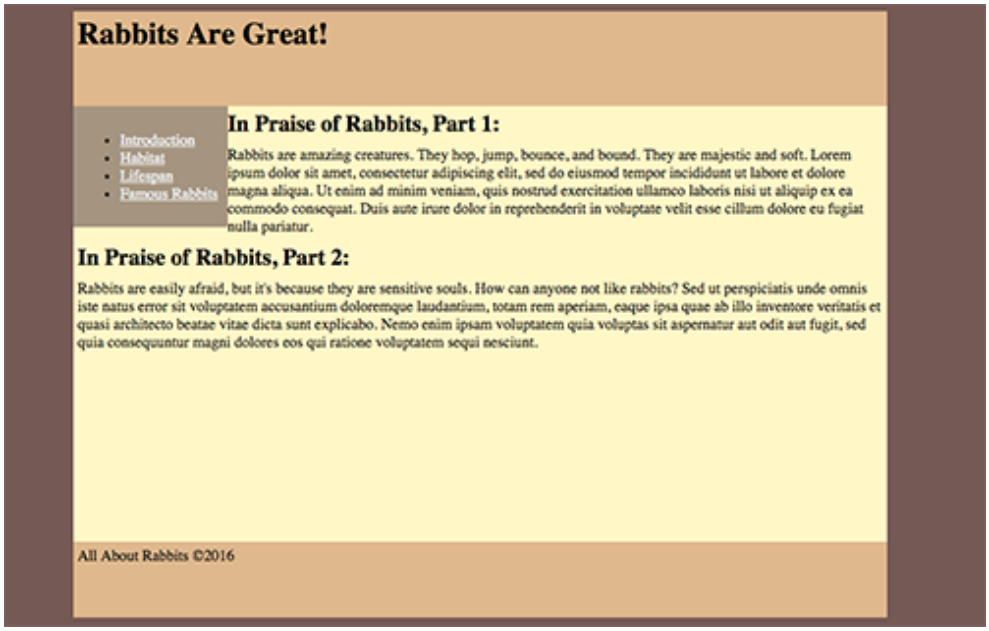
The min-height property is fairly common, and has many practical uses. However, you'll want to be careful with the min-width property. Using min-width will allow the element to expand to the width of the user's browser window, undoing all the hard work you may have done to keep lines of text at a readable length.

When in doubt, you'll want to use the basic width property instead of the min-width property to assign a width to an element.

Practical Uses for CSS Dimension Properties

So, why is it important to always declare height and width properties? Height and width in CSS, more than any other CSS property, go a long way to giving your designs a consistent look and feel across your site. You'll find the min-height property especially useful when you're working with page content that varies in length.

You'll also find the height and width properties useful when working with column layouts and floats. Take our rabbit page, for example. If I were to add a floated sidebar menu to the rabbit page without declaring a width or height, the sidebar would look something like this:



View larger image

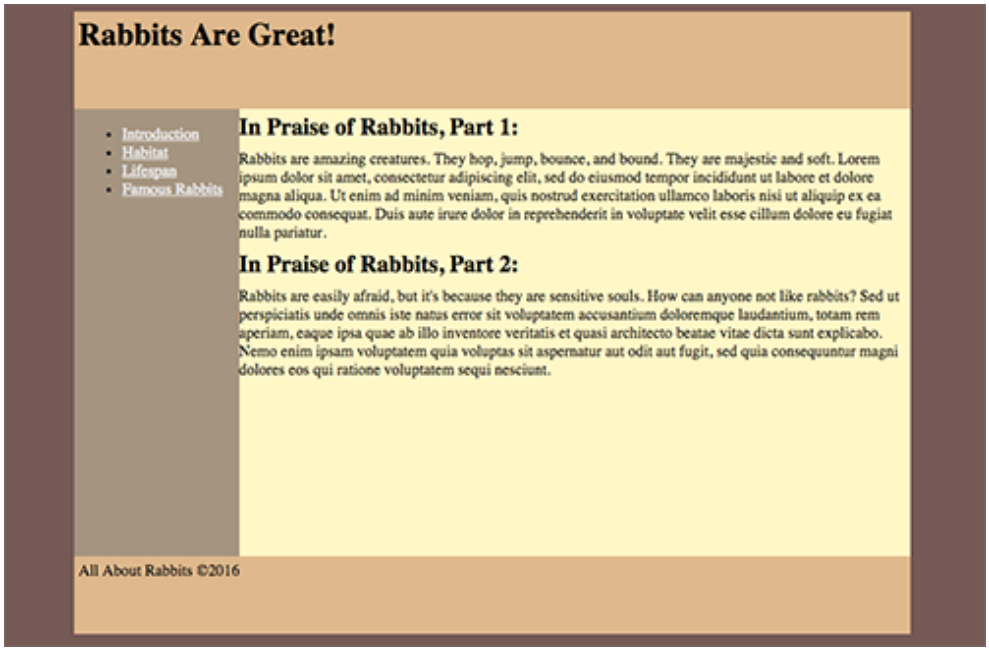
But after assigning a width of 150px and height of 460px to my navigation menu...

```
nav {
        width: 150px;
        height: 460px;
}
```

important!

Using too many Web fonts can slow down the loading of your Web pages.

The sidebar looks like this:

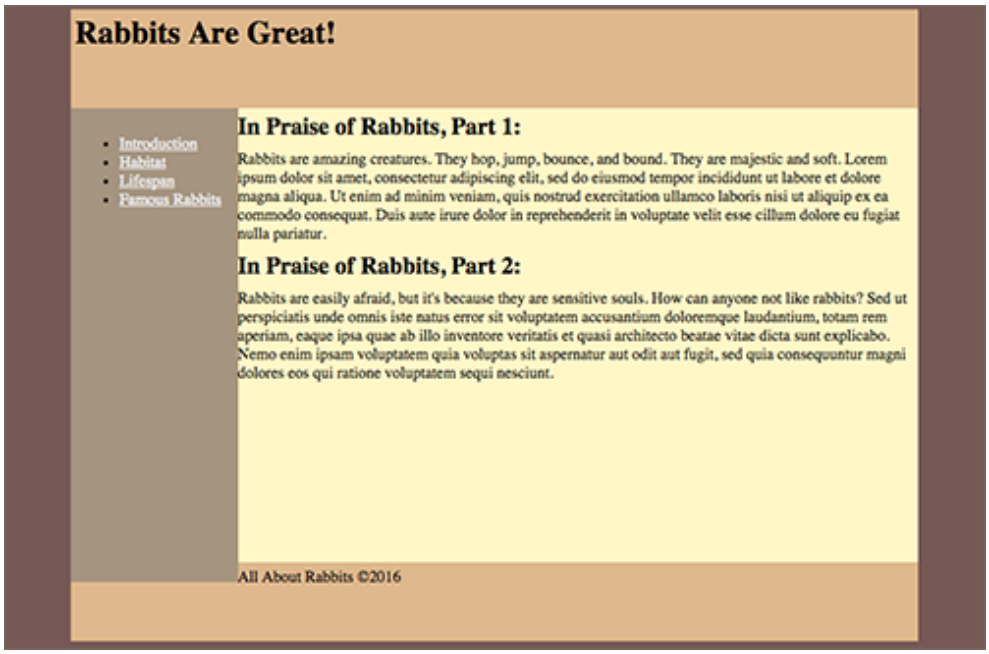


View larger image

As you work with CSS dimension properties, remember to keep the CSS box model in mind. Padding, margins, and borders get added to the width and height of your element, which can easily push boxes out of whack if you're not paying attention. Because the #frame div on the rabbits page is exactly 460px tall, if I add any padding to the top or bottom of the navigation menu, the entire sidebar will extend past the bottom of the frame.

```
nav {
        width: 150px;
        height: 460px;
        padding: 10px;
}
```

Look at what just 10px of padding on all sides does to the footer element:



note

CSS3 includes a range of visual effects and animations.

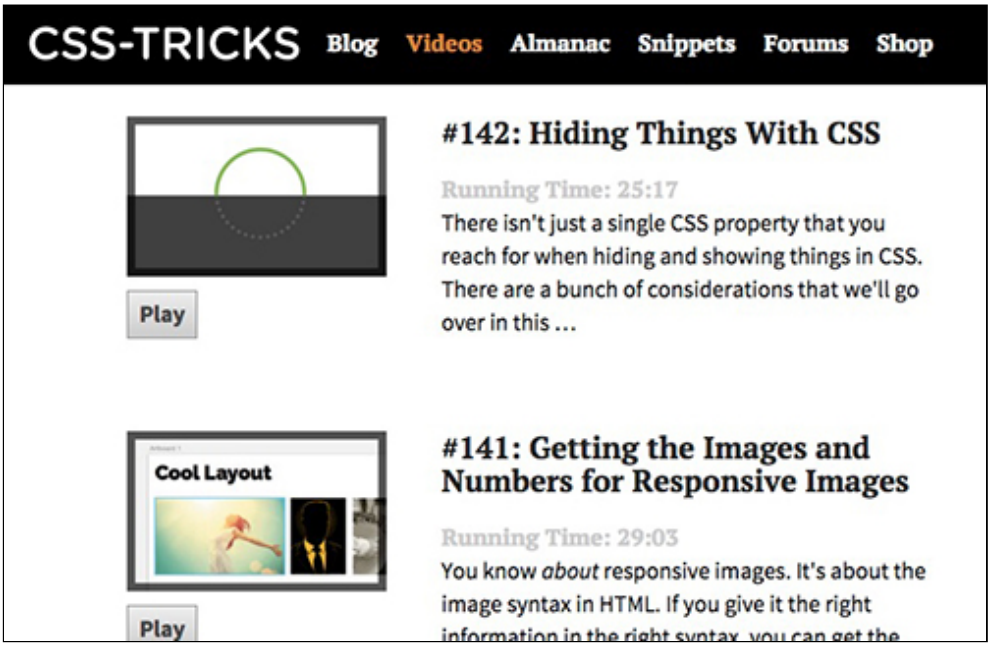
Notice that the sidebar has started to overtake the footer element, and has pushed the footer text to the side? - View larger image

It's subtle, but it's enough to throw off a good design. Mind the box model as you go and everything should be ship-shape!

Using Web Fonts

You've already got a good handle on using CSS to transform text. Why not take that typography to the next level?

Not to be confused with Web safe fonts, **Web fonts** aren't pre-installed on a visitor's operating system, but instead reference an external font library on the Web. That font is then rendered in the visitor's Web browser. If the typography on your favorite blog has a hip, contemporary feel, chances are that Web site is using Web fonts. Here's an example:



The CSS Tricks Web site uses a mix serif and san-serif Web fonts. Headings are in "Pt Serif" and body text is "Source Sans Pro."

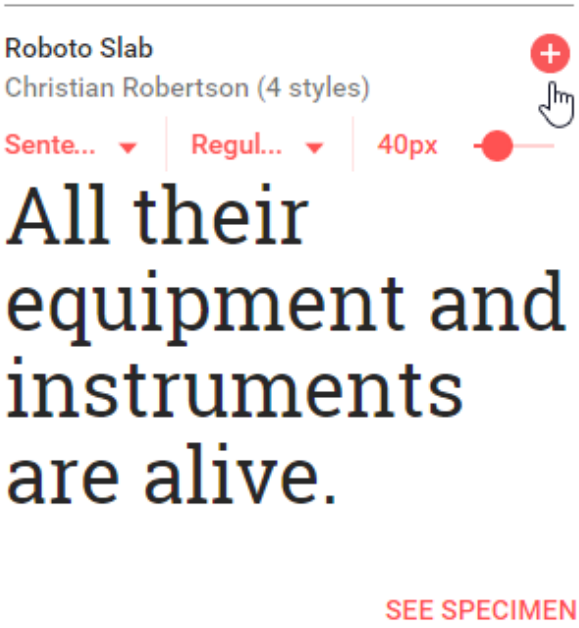
The benefit of Web fonts is that the possibilities are endless. While many sites offer paid subscription models, there are free options as well—the most notable being Google Fonts.

Note that different online font foundries use different methods for linking these fonts to your stylesheet, and these methods are often changing. With that said, it's often as simple as copy/pasting a line of code into the <head> section of your HTML document, and listing the font in your font-family CSS property.

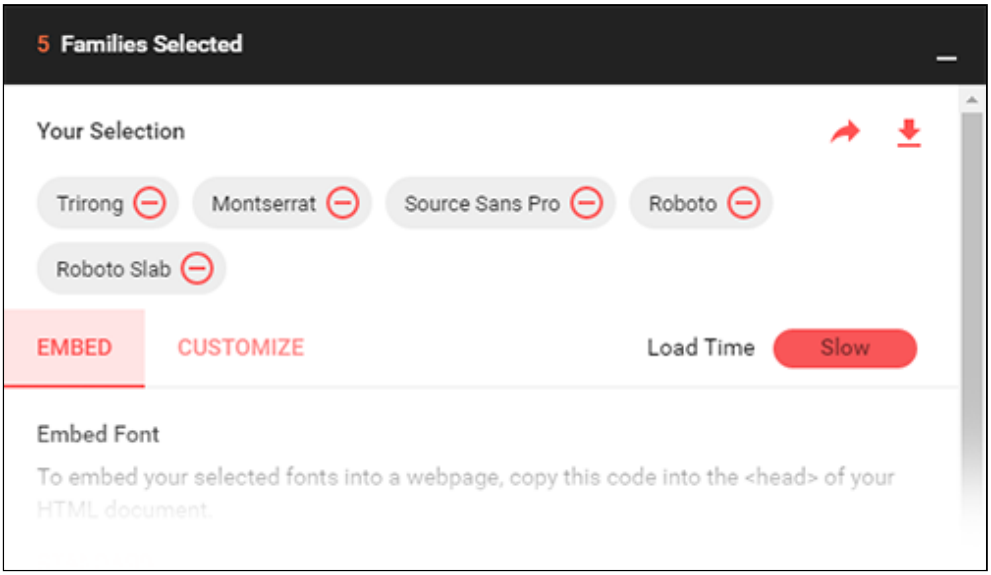
To incorporate Google fonts into the rabbits.html page, browse through the font library until you find something that strikes you. Click on the plus sign in the upper right corner to save that font for use on your Web site. In the image below, I've selected "Roboto Slab."

important!

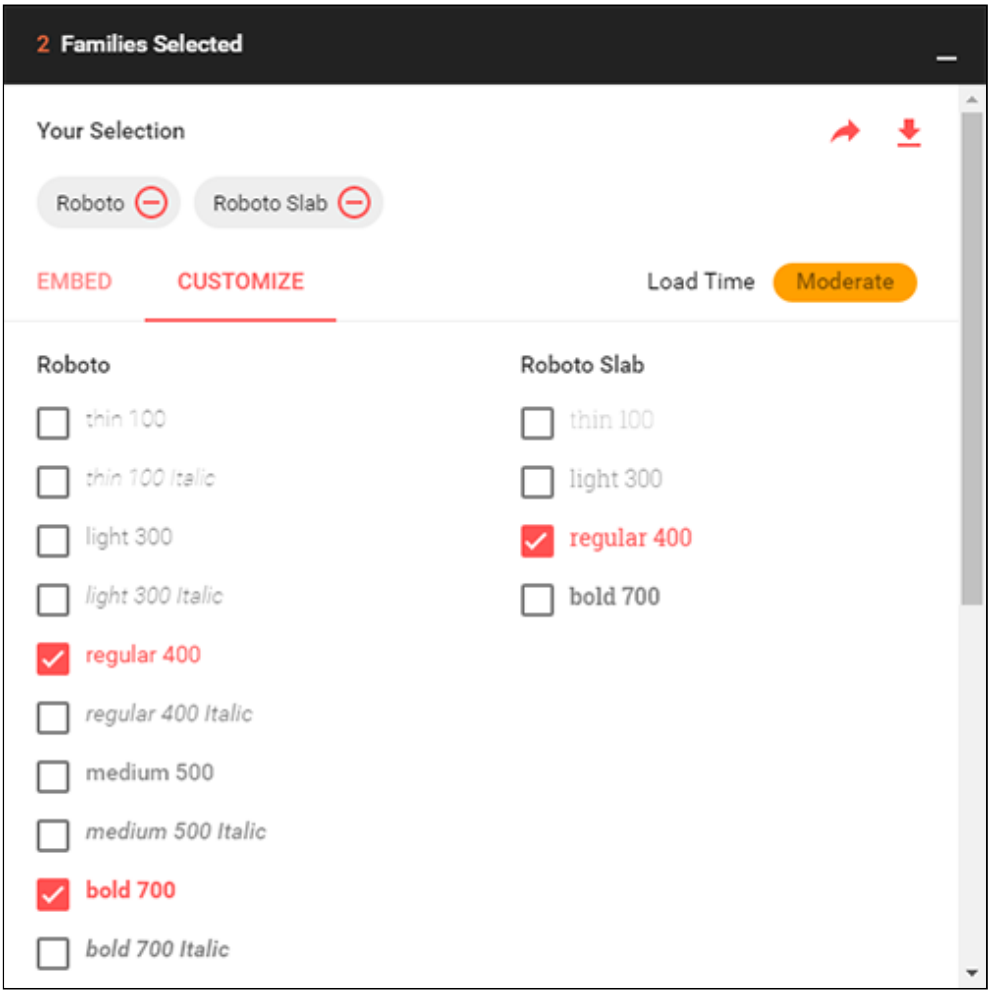
Make sure you are using CSS3 effects for a good reason, and avoid overuse.



You can save multiple Web fonts to your collection before adding them to your site. But proceed with care! Because Web fonts must reference external font libraries, too many Web fonts can slow down the load time of your site. It can be hard to resist the urge to go font crazy when you first start using Web fonts. At this stage, I recommend adding no more than two Web fonts to your site.

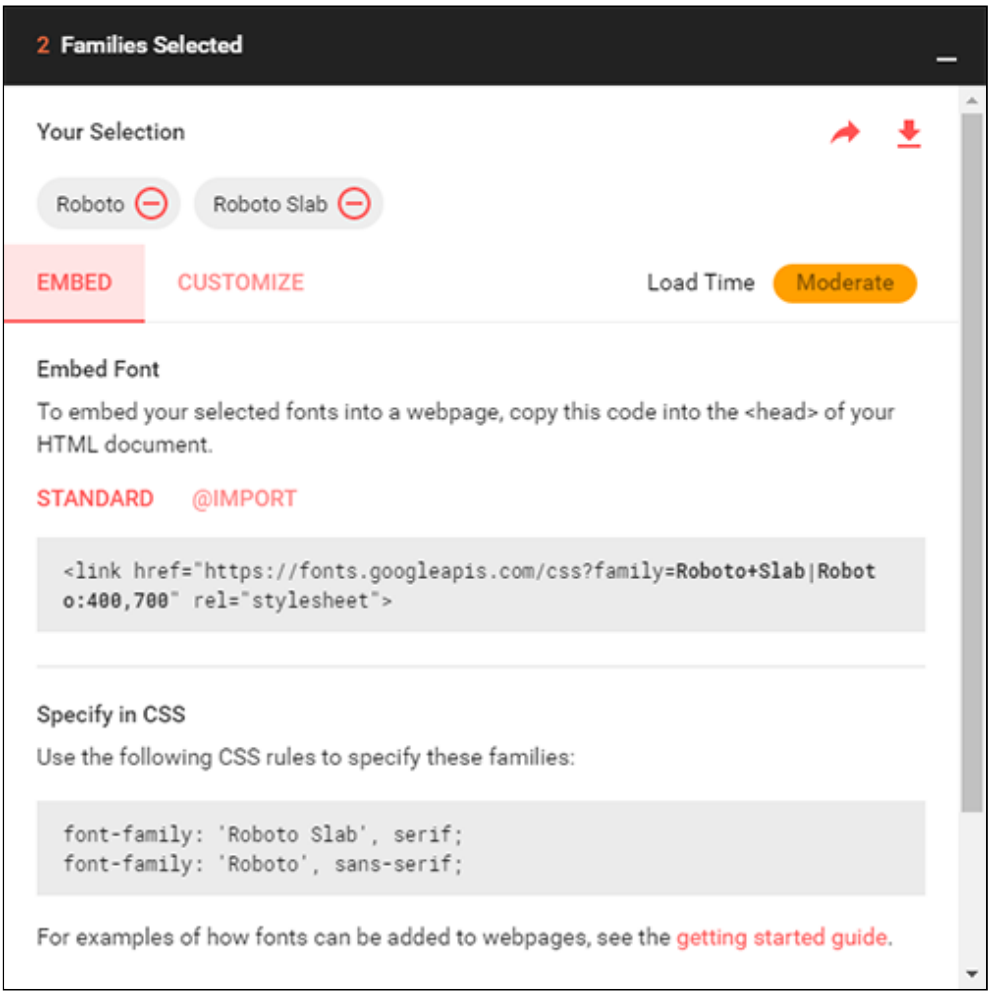


When you're happy with your font selection, navigate to the "Customize" tab. This section will give you the option to choose the exact font styles you want. For example, if you know for a fact that you won't be using italicized text on your Web site, you can deselect the italics option. This will save on page load time, but if you do choose to use italics in the future, italic text may have a broken or blurry appearance on your site!

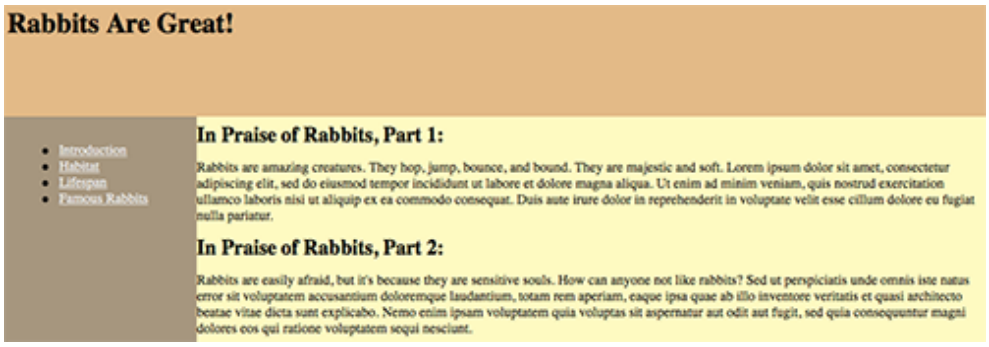


You can select the exact styles you want within a font

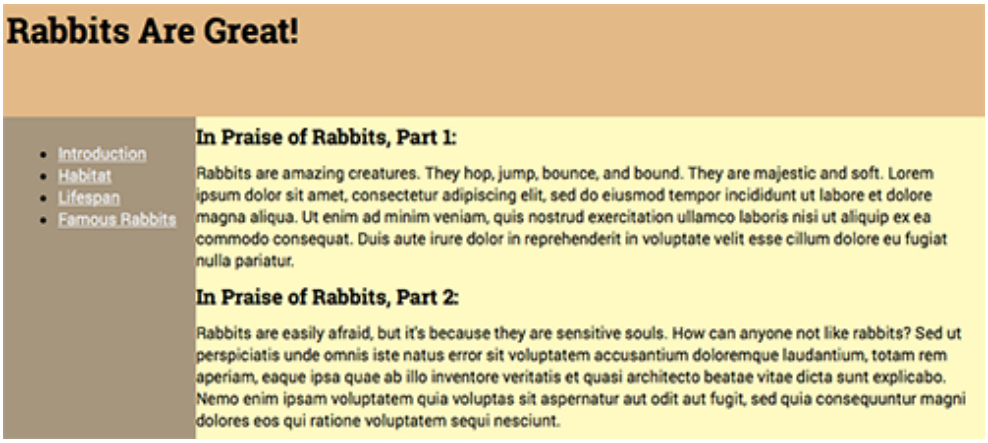
Finally, when you're ready to add your new fonts to your site, return to the "embed" tab to grab the necessary code. Add the HTML code snippet to the <head> of your HTML document, and add the font names to your CSS styles just as you would with any other font. I chose "Roboto Slab" and "Roboto," which will pair nicely in my headings and paragraph text.



Here's what my rabbits.html page looks like before and after adding Web fonts:



Before Web fonts - View larger image



After Web fonts - View larger image

CSS3 Techniques

In this lecture, you've picked up a few new advanced techniques: horizontal navigation, min and max dimension properties, and Web fonts. Now, I'd like to introduce you to a powerful resource for your Web design arsenal: CSS3.

CSS3 doesn't encompass a single style rule. Rather, it's the most recent iteration of the CSS language. CSS3 includes lots of neat effects and add-ons that bring greater interactivity to Web pages. Some of these more complex effects include things like transition effects and animation. Other CSS3 capabilities include rounded corners, color transparency values, and shadow effects for text and box-level elements.

Let's look at a few useful CSS3 techniques now!

Border-radius

The CSS3 **border-radius** property is used to apply rounded corners to an element. The rounded corners can be subtle (like a gentle softening of the edges of a box) or extreme (turning a box into a perfect circle). Here's some sample code for the border-radius property:

HTML:

```
<div id="corners1"> ... </div>
```

CSS:

```
#corners1 {
        border-radius: 10px;
        background: #f60;
}
```

And here are a few different examples of what the border-radius property looks like in action!

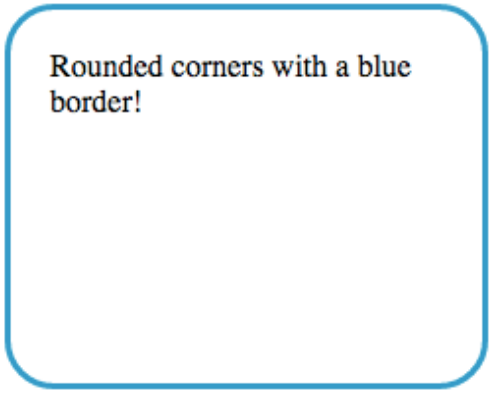
Rounded corners applied to an element with a blue background color and white text:

```
.box1 {
        border-radius: 25px;
        background: #09c;
    color: #fff;
    padding: 20px;
```

```
    width: 200px;
    height: 150px;
}
```

Rounded corners applied to an element with a blue border:

```
.box2 {
        border-radius: 25px;
        border: 3px solid #09c;
    padding: 20px;
    width: 200px;
    height: 150px;
}
```



Rounded corners applied to an element with a background image:

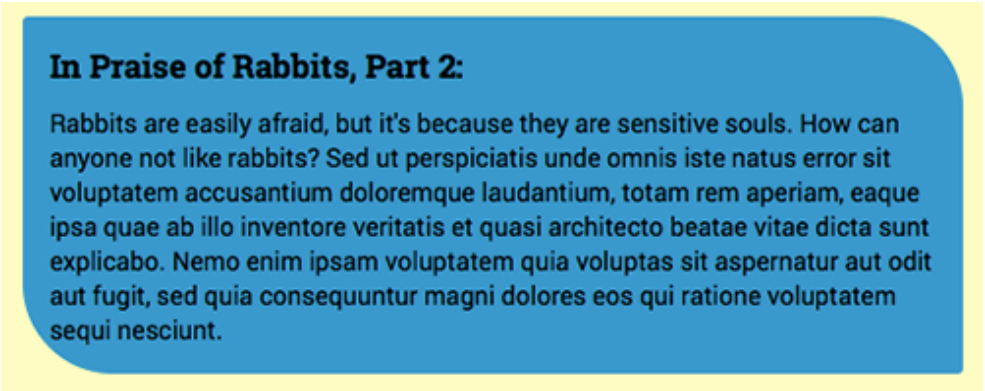
```
.box3 {
        border-radius: 25px;
        background-image: url(cat.jpg);
    color: #fff;
    background-position: center;
    background-repeat: none;
    padding: 20px;
    width: 200px;
    height: 150px;
}
```

Note: You'll only notice the border-radius property in action if you've assigned a different background color or border color to the box element in question. If the box element is white and the rounded corners are white, they'll be invisible!

Finally, you can apply different values to the box element to control each corner of the box. Assigning to different values creates a neat "leaf" effect!

```
#rcorners1 {
        border-radius: 3px 50px;
    background: #09c;
    color: #fff;
    padding: 20px;
    width: 200px;
    height: 150px;
}
```

Open up a new HTML document, or try adding a few boxes with rounded corners to your rabbits.html page.



Keep this new document open, as we'll be adding more CSS3 to the page momentarily!

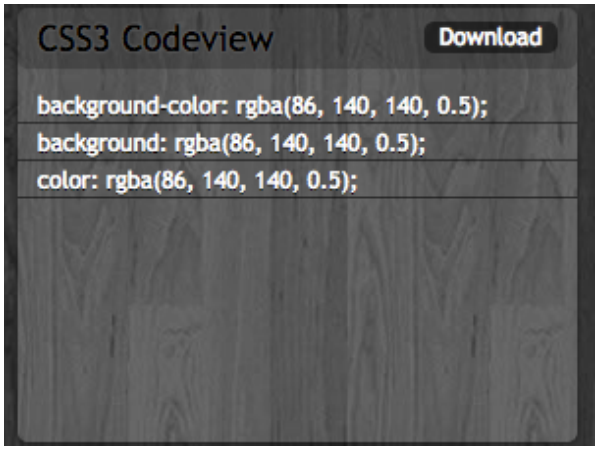
RGBA Colors

You've already learned about using HTML hex codes (like #fff or #40ff00) to choose color values on the Web. Now, what if you wanted to assign a color to an element with a bit of transparency?

RGBA colors are RGB color values that include an alpha channel. RGB stands for red, green, and blue. RGB represents the different ways red, green, and blue light get added together to display all the colors you see on the Web. The A in RGBA stands for alpha. The alpha channel represents transparency. When defining RGBA color values, 1 represents completely opaque (you can't see through the color at all) and 0 represents completely transparent (the color is invisible).

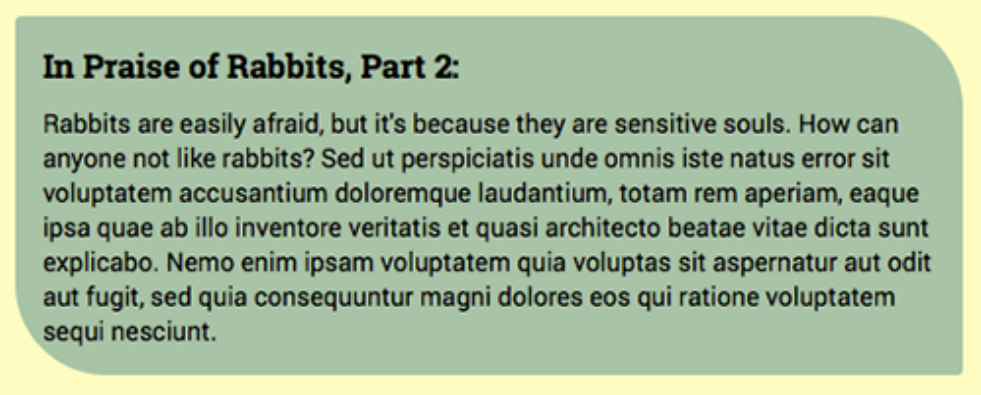
One of the great things about generating CSS3 effects is that there are many tools on the Web that help shortcut the process. When adding RGBA color to a Web site, I turn to my favorite CSS3 resource, css3maker.com.

I've created the transparent aquamarine color shown above by dragging the RGB and Opacity slides in the left-hand column. When I'm ready to add these values to my Web site, I can copy the appropriate code snippet from the right-hand column:



The first value list represents the red channel, the second green, the third blue, and the fourth represents opacity. 0.5 is equivalent to 50% opacity.

Here's what the background-color RGBA code looks like applied to a box element on my rabbits.html page:



Text-Shadow

Last but not least, CSS3 offers some neat effects for adding captivating text shadows to your site! Here's the code for a simple white text shadow, which I've incorporated into my rabbits.html page:

```
text-shadow: 3px 2px 6px #ffffff;
```

The first value of 3px represents the horizontal length of the shadow. The second value of 2px represents the shadow's vertical length. The third value of 6px specifies the shadow's blur radius (how crisp or blurry the shadow appears). The fourth and final value you should recognize as the shadow's color. Shadows are commonly gray or black, but you can play around with different color values to achieve different shadow effects.

As with RGBA opacity, you can get a bit of help creating text shadows using the css3maker Web site: http://www.css3maker.com/text-shadow.html

When applying shadows to text, it's important to remember that *less is more*. Whereas drop shadows, gradients, and beveled edges used to be very popular on the Web, current design trends lean in the favor of flatness and minimalism. Use text shadows with care, and use them in ways that support the readability of the text and match the tone of the design.

Consider how shadow effects will impact the tone of your site. My rabbits.html header text, rebooted for a death metal band!

Ready to give it a try? Hop on over to Exercise Five to put your advanced CSS knowledge to the text with horizontal navigation, dimension properties, newly improved font styles, and CSS3 effects!

In the next lesson

- Learn some important benefits of responsive Web design.
- Learn how media queries work and how to attach them to a style sheet.
- Learn the benefits of

Coming Up Next:

Discussion
Share your thoughts and opinions on Web fonts and CSS3 in the Discussion area.

working with a responsive framework or boilerplate.
- Learn how to set up the Skeleton framework in preparation for a Web design project.

Exercise

Pick one of two clients and create a five-page Web site for that client.

HTML and CSS | Choose Your Own Adventure

sessionscollege
FOR PROFESSIONAL DESIGN

Exercise 5

Choose Your Own Adventure

In Lecture Five, we explored some advanced CSS techniques to take your designs to the next level. We talked about using the inline property to create horizontal navigation menus, about the importance of using CSS dimension properties, and about ways to incorporate Web fonts into your design. We also looked at some nifty CSS3 styles.

In this exercise, you'll choose between two different clients in need of a new Web site. The first client is the Mountain Zen Center, a fictional Zen retreat center nestled in the beautiful Catskill Mountains of New York. The second client is Lift This! Fitness, a private weight lifting gym in Los Angeles, California with a focus on women's fitness.

You'll choose between two very different clients for this exercise.

After choosing which client you'd like to work with, you'll create a five-page website for the client of your choice, putting your new advanced CSS knowledge into practice.

Performance Objectives

- Create a five-page Web site for one of two clients.
- Code your site using HTML5 markup, including header, nav, content, section, and footer tags.
- Create a CSS design with a horizontal navigation, two column layout, and floated secondary navigation.
- Include properly sized images, at least one Web font, and at least one CSS3 style.

note

Assignments are evaluated for creativity, technical proficiency, and

Project Brief

Whichever client you choose, your Web site should reflect design principles that match the content. The color palette, images, and font styles for a Zen retreat center will likely be very different from those used for a weight lifting gym. Choose whichever client you're most excited to work with, though I encourage you to push yourself out of your design comfort zone!

Regardless of which project you select, your site should contain:

- Valid HTML5 markup, including a header element, nav element, a main content area with the appropriate section tags, and a footer
- A primary horizontal navigation menu
- A two-column layout that makes use of floats

understanding of concepts covered in the lecture.

- A secondary vertical navigation menu in the floated column
- Properly sized images
- At least one Web font
- At least one CSS3 style

Sketch Out Your Design

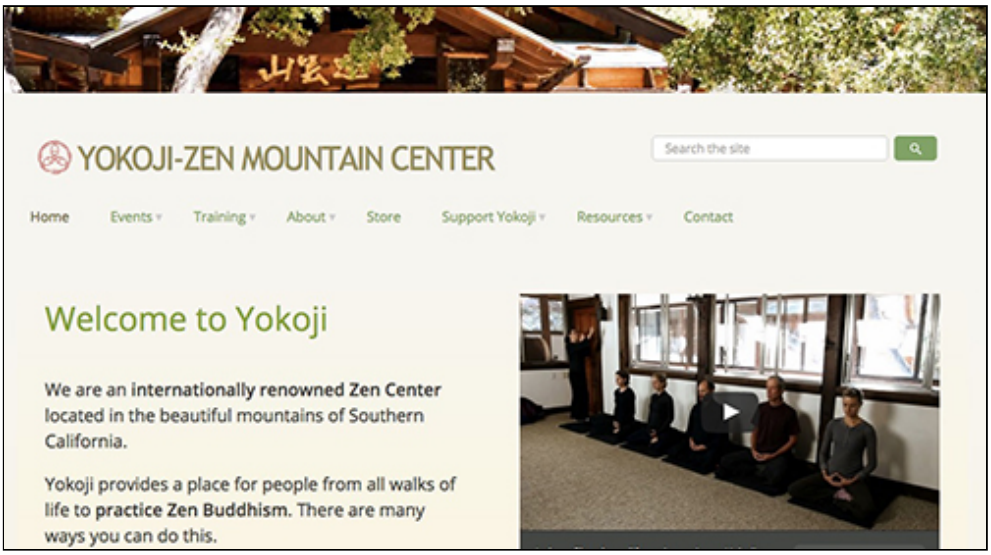
Before jumping into marking up your content, take some time to sketch out the site layout. You can use Photoshop, or grab a notebook and try good, old-fashioned pen n' paper. Remember to include a clear header area, main content area, floated sidebar, and a footer.

An essential part of the design process is searching for inspirationâ€”and searching for potential competition! Who are your client's competitors? What do similar Web sites look like?

Search online for Zen retreat centers in the eastern United States, or for women's gyms and fitness centers in the greater Los Angeles area. If you can't find anything that matches your search criteria, expand the location to do a wider search. Make sure to bookmark these links, as I'll be asking you to share your design inspirations with me later!

Here are two examples of sites that match the look and feel we're going for:

The Yokoji-Zen Mountain Center: The Yokoji-Zen Mountain Center site features a left-aligned title and logo, a horizontal primary navigation scheme, a main content area to the left, and a wide floated sidebar to the right with videos, a newsletter subscription form, and a list of upcoming events.

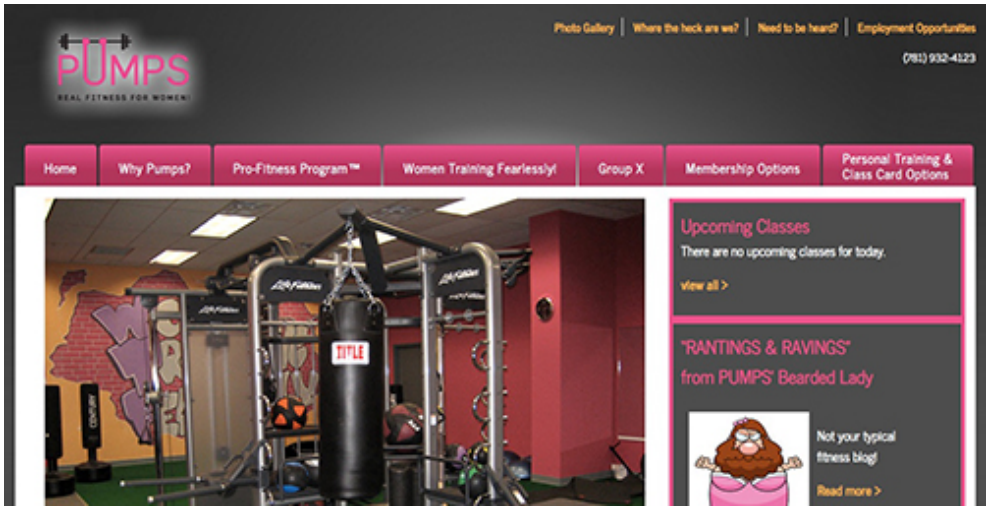


A mindful design

The site footer contains the address and contact information, a link to their location in Google maps, links to social media accounts, and copyright information.

PUMPS Gym: This women-only gym features a tabbed horizontal navigation scheme below a left-aligned site title and logo, a main content area to the left,

and a floated sidebar to the right with information about upcoming classes, and secondary navigation links.



Heavy lifting required

The footer contains the location, hours of operation, and social media links.

Mark Up the Pages

Download the Exercise 5 zip file and extract the contents. In the Exercise 5 folder you'll find two additional foldersâ€”one called Zen and the other called Fitness. Each folder contains all the text and image files you'll need to get started. Remember, you're only making one Web site, so make sure you're working from the right folder as you go.

Open a new document in Sublime and start by the marking up the content for each page in HTML5. Each page should include a header, a primary horizontal navigation menu, a main content area divided into appropriate sections, and a footer. Start with this basic document structure and work from there:

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>My Web Page</title>
<link rel="stylesheet" type="text/css" href="css/style.css"/>
</head>
<body>
--Site content will go here.--
</body>
</html>
```

You'll only need one external style sheet for the entire site, but the style sheet must be linked from each page. Remember to customize the page title in the head of the document for every page of your site.

Header and Primary Navigation

Enclose the header area in HTML5 <header> tags and wrap the site name in <h1> tags. Below the title is a brief description of the site. You might choose to wrap these in a smaller heading tag or a paragraph tag, so long as you make consistent choices as you go.

Decide whether you'd like to include the primary navigation menu in the header of your site, as shown below, or whether you'd like to include it in the main content area.

```
<body>
<header>
<h1>Site Title</h1>
<p>site description</p>
<nav>
...
...
...
</nav>
</header>
```

Wherever you decide to include the primary navigation, remember to enclose the menu in HTML5 <nav> tags. The menu should be formatted horizontally, and must include the following links, in this order:

Home - About - Classes - Hours - Contact

You can use either a list-based navigation scheme wrapped in <nav> tags, or utilize the default inline <nav> display described in the lecture. Each navigation tab must link to the correct page, so be sure to add the correct hyperlinks to each page and check your work as you go.

Main Content Area

The main content area of every page should include heading tags, section tags, and paragraph tags. Be sure to include an image on every page. There are lots of great images to choose from in the exercise folder!

Secondary Navigation

Because this navigation isn't of primary importance, it should not be wrapped in <nav> tags. You will, however, want to wrap the secondary navigation links in tags so it can later be customized to your specifications! Without a custom class or div ID, you'll have trouble floating this navigation menu down the road.

The text and anchor links for the secondary navigation menus are included in the exercise folder. Be sure to format these properly and test each link before moving on.

```
<a href="http://wikipedia.com/">This is a link to Wikipedia</a>
```

Footer

The footer area should come after the main content area and secondary navigation, and should be enclosed in HTML5 <footer> tags. Be sure to wrap the footer content in the appropriate paragraph tags or divs.

Style the Pages

With the basic markup in place, it's time to give this Web site some style. You should have already added a placeholder link to a style sheet in the head of your document, so create that style sheet now and save the file as style.css.

```
<head>
<title>My Web Page</title>
<link rel="stylesheet" type="text/css" href="css/style.css"/>
</head>
```

You don't have to use the file structure pictured above, as long as you maintain the same file structure between your local machine and your Web server.

Next up, begin assigning widths and heights to your page elements. If you're feeling fatigued assigning an exact width to the header, footer, and main content area, try wrapping all of the main content in a div called "wrapper" or "frame" and assign a width only to this containing div.

Create a two-column layout by specifying a width for either the main content area or the secondary sidebar navigation div. Then, float the sidebar div to the left or right. As a reminder, floated elements are positioned vertically within normal flow, but are horizontally pushed all the way to the left or the right of the containing element. Content that follows a floated element in the markup then wraps around it. If the content following the float is too big (too wide, for example) to fit next to the float, it will appear below the float. Make use of the CSS height property to set a consistent height for the sidebar, which will prevent page text from flowing around it!

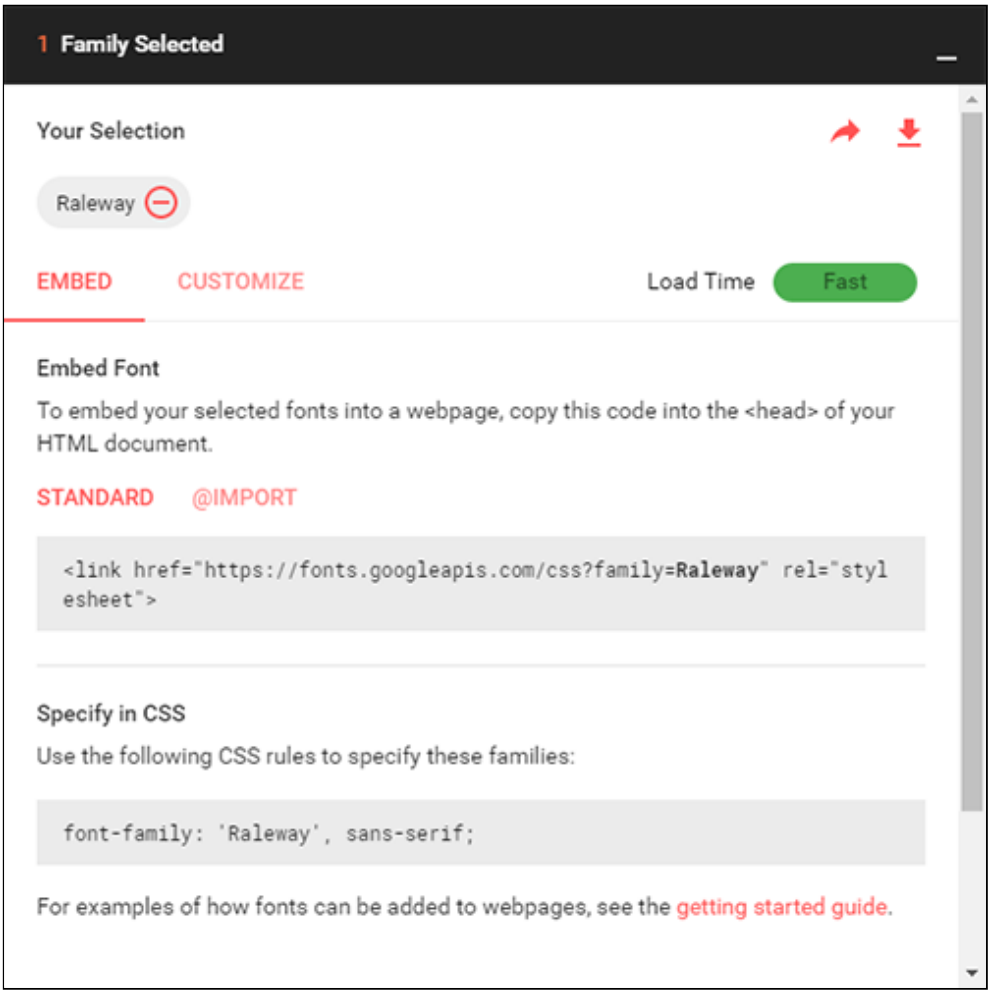
Your page design should have a consistent height, regardless of how much content is on each page. Remember to use **min-height**, and other advanced CSS dimension properties you learned about in the lecture, to streamline your design.

Now, choose font-family styles and sizes for your header and paragraphs and adjust the letter-spacing and line-height as needed. You're required to use at least one Web font in this design, so take some time browsing the Google fonts collection to find appropriate font styles. You can save multiple Web fonts to your collection to review before use, but proceed with care, as too many Web fonts can slow down the load time of your site.

Raleway is a modern, elegant font choice. Perfect for a Zen retreat center. Not so great for a weight lifting gym, though!

When you're happy with your font collection, navigate to the "customize" tab. Select the font styles you want and uncheck any styles that you won't be needing. Add the Google fonts HTML code snippet to the <head> of your HTML

document, and add the font names to your CSS styles just as you would with any other font.



Add the HTML code at the top to the <head> section of your document. Then, integrate the CSS below into your style sheet.

Set background colors and text colors to match the feel of your content. I encourage you to use resources like COLOURlovers to find tried-and-true color combinations!

Be sure to style your lists and links, and add margins and padding so your content has room to breathe. Borders and background images are optional, but they can go a long way in adding depth to your design. Try searching Pixabay for free-to-use background images that might work well with your content.

Experiment with applying one of the CSS3 techniques described in the lecture: rounded corners, transparency, or text shadow. I'd like to see you integrate one of those styles into your site.

Remember to check your work in your browser as you go! Preview the file often and watch for these possible design pitfalls:

- Head and footer misaligned: Did the header and/or footer of your site pop out of alignment when you floated the sidebar? You may need to add the CSS property clear: both; to re-align both of these elements!
- Unwanted space: Is there unwanted blank space between your header, footer, and main content

area? Remember to add CSS reset styles to override the margins added by your browser's user agent style sheet.

Test Your Site

After uploading your Web site to your server, remember to test your work in multiple browsers. Before submitting, run the link through the HTML5 validator to be sure you've included properly formatted HTML5 tags, and that all tags are in the right order.

Describe Your Experience

When you upload your site to the Dropbox, I'd like you to share your inspiration links with me and answer the following questions:

- Which Web site or sites did you use for inspiration? Please share the Web site urls.
- Which elements of the design most inspired you? For example, did you take inspiration from the color palette, the unique layout, the use of images, the typography, etc?

I look forward to seeing your masterful creation!

Grading Criteria:

What your instructor expects you to do:

☐ Demonstrate the ability to use HTML5 to structure your site content, including header, nav, content, section, and footer tags.

☐ Show the ability to design a CSS layout using a horizontal navigation, a two column layout, and a floated secondary navigation.

☐Demonstrate the ability to include properly sized images, and at least one Web font and CSS3 style.

☐ Demonstrate the ability to make consistent and attractive design choices in color, typography, and images are appropriate for your selected client.

☐ Demonstrate the ability to create clean, consistent, valid, and standards-compliant code.

How to Post:

Once you're done, go to the Dropbox for this exercise and post your URL for this assignment along with your written comments.

If you have a question before sending your completed exercise for grading, be sure to contact your instructor via the People area.

I look forward to seeing your work!

sessionscollege
FOR PROFESSIONAL DESIGN

Lecture 6

Intro to Responsive Web Design

The answer to a streamlined browsing experience lies in responsive design.

Today, more people than ever are browsing the Web on the go. According to a 2015 article in the Guardian, smartphones are responsible for one-third of Internet access in the UK—a percentage that's jumped more than 20% since 2012. Similarly, two-thirds of adults now own a smartphone, a jump of nearly 40% since 2012.

With so many people accessing the Web on mobile devices, modern smartphones have stepped up to the challenge, offering bigger screens, better user interfaces, and higher resolution than ever before.

As a Web designer, it's important to offer users the same easy, accessible browsing experience whether they're accessing your site from a desktop computer, a tablet, or a phone. So, how do you accomplish such a feat? Through the power of responsive design.

Learning Objectives

In this lecture, you can expect to:

- Learn some important benefits of responsive Web design.
- Learn how media queries work and how to attach them to a style sheet.
- Learn the benefits of working with a responsive framework or boilerplate.
- Learn how to set up the Skeleton framework in preparation for a Web design project.

What is Responsive Web Design?

Responsive Web design is all about designing Web sites with a wide range of viewing devices in mind. A responsive site is a Web site that will adapt fluidly to the size of the browser window. As the browser gets smaller, the images will scale proportionally and sidebar columns will rearrange down to a single column of text.

Before responsive Web design took hold, there were two different ways to design mobile sites. The first way was to not design a mobile site at all. As a designer, you just crossed your fingers and hoped that your fixed-width Web site wasn't too small to be readable on a mobile phone! The second method involved creating an entirely separate, special site just for visitors accessing your site on mobile. Neither of these methods was very good.

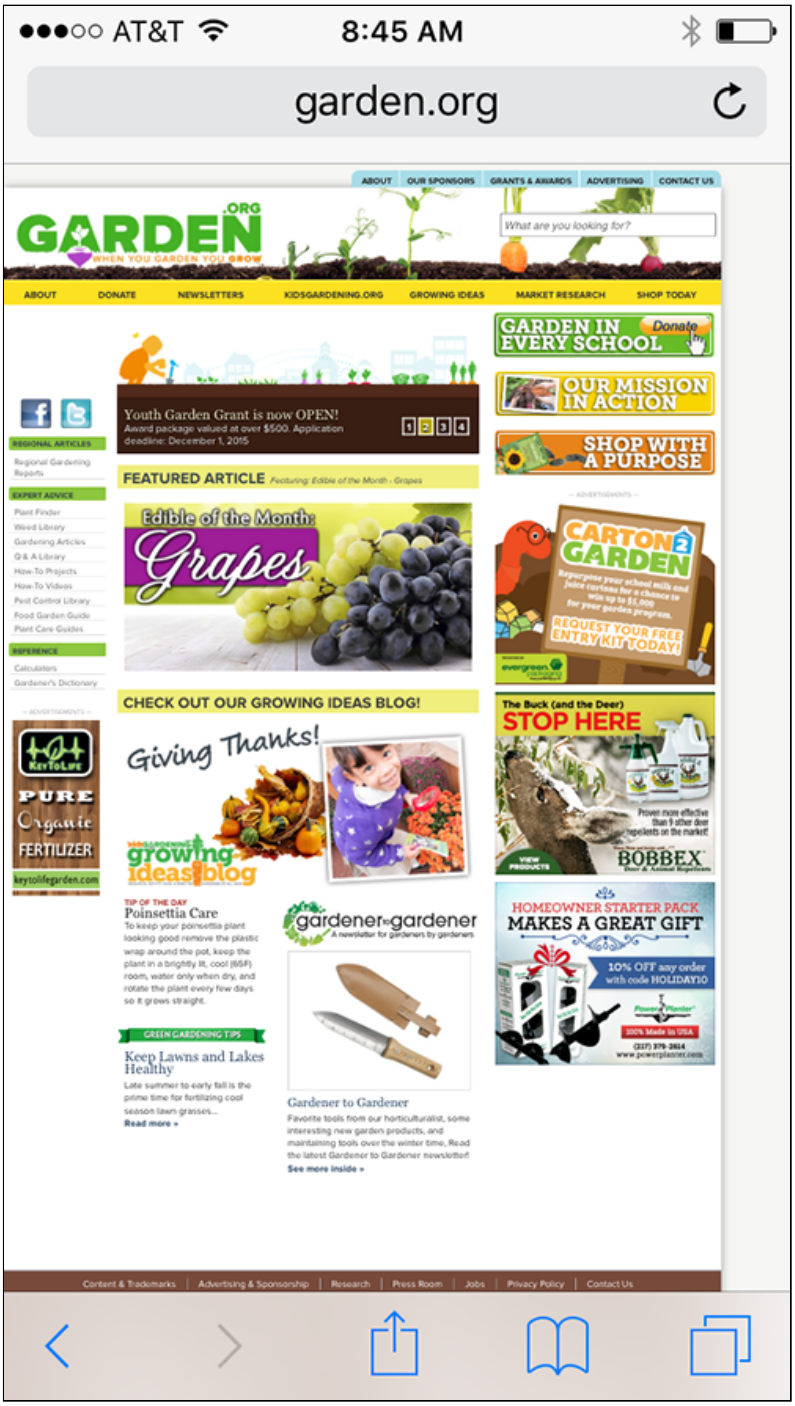
Mobile-only Web sites tended to confuse visitors. They often didn't look much like the original site, and many were made in haste to meet the rising demands of mobile viewing. Fixed-width Web sites frustrated visitors, forcing people to zoom in on text with their fingers just to read a menu, or to find an important phone number. Here's what the fixed-width site, garden.org, currently looks like on a smartphone:

note

A responsive Web site adapts to the size of the browser window.

note

A responsive design makes a Web site more accessible than a traditional site by making it easier to access content across a range of devices.



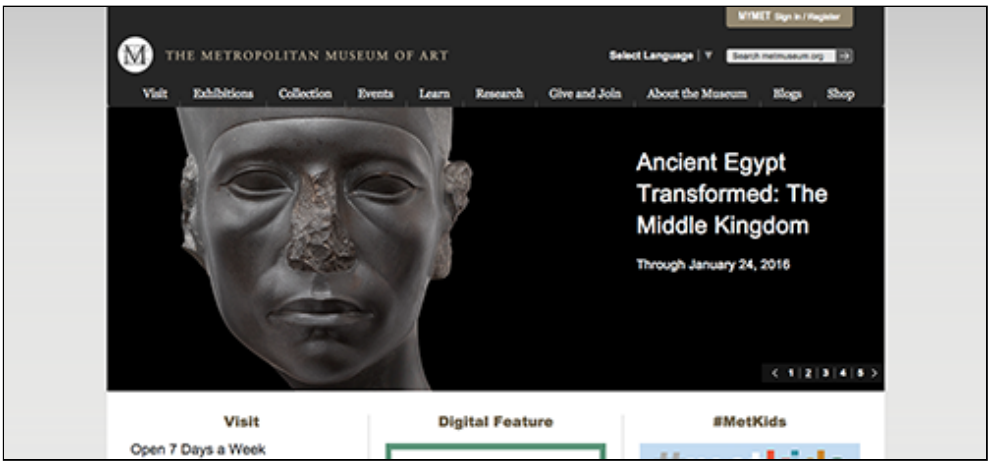
Unfortunately, this isn't a very accessible browsing experience! I can read the text on the image slider and the text in some of the sidebar advertisements, but I can't make out any of the navigation links without zooming in. The site also drifts to the left in the device window, instead of displaying in the center. If you browse the Internet on a smartphone, you've probably noticed that there are still many fixed-width sites on the Web. It's a lot of work to upgrade a site to a responsive design, but it's time for designers to answer the call.

You've probably seen many responsive sites before, though you may not have realized it. The power of responsive Web design is best demonstrated through examples, so let's take a look at a few non-responsive and responsive sites.

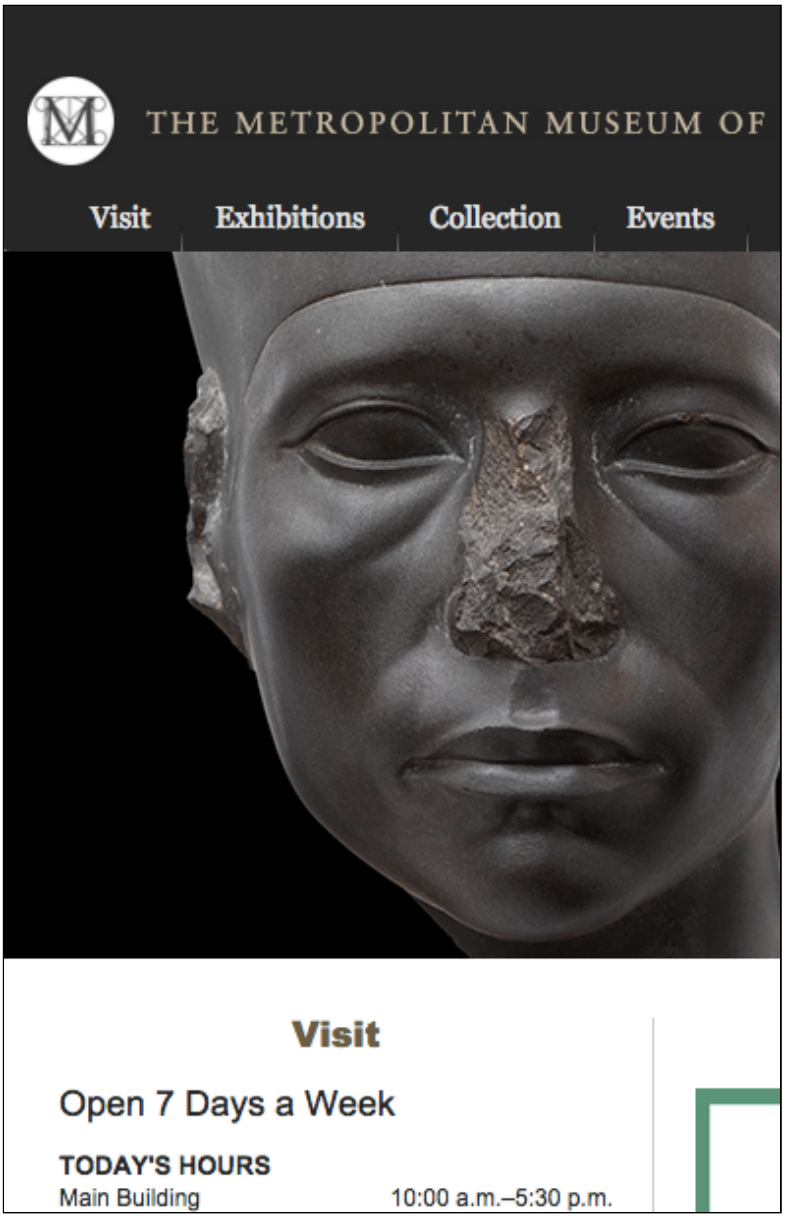
Here's what the non-responsive Metropolitan Museum of Art Web site looks like on my laptop. All of the content is viewable on my screen, and the text is clear and readable.

note

Responsive design can be essential for a site where the majority of visitors are mobile visitors.



And here's what happens when I scale my browser window to a smaller size:

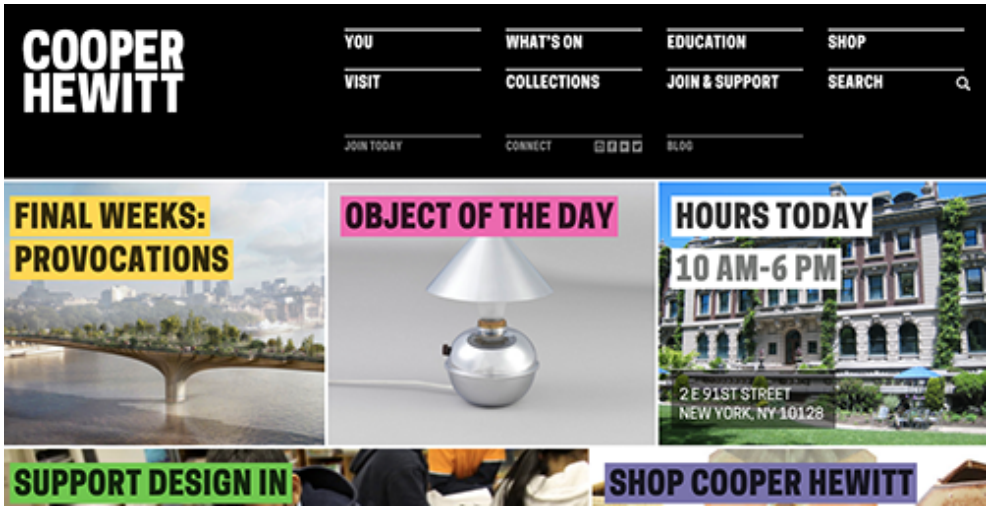


Oops! This view isn't very helpful! I can see part of the slider image, which is good, but I only see half of the primary navigation menu. The left sidebar is visible, but if I want to access the main content area, I'll have to scroll to the right.

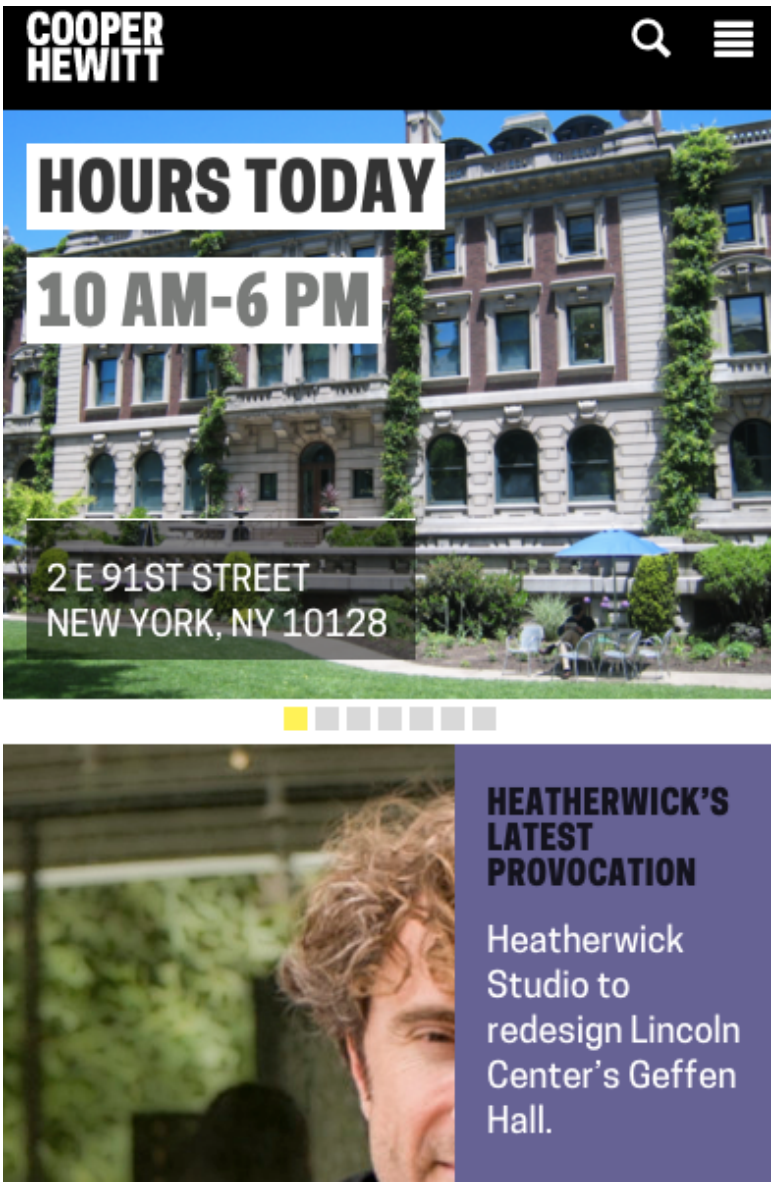
note

Media queries are filters that display different content depending on the user's screen size or device.

Let's take a look at another example. Here's what the responsive Cooper Hewitt Museum site looks like on my laptop:



And here's what it looks like when I scale the window of my browser down to a mobile screen size:

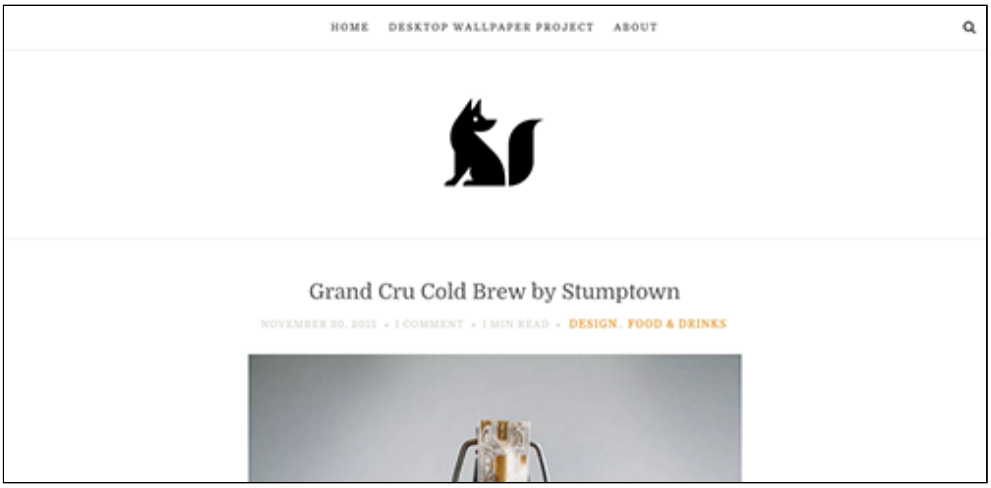


note

Media queries can be created in a new external style sheet, or simply added to an existing one.

Notice how the grid layout on the Home page shifted into a single column? The hours and address are still up at the top, and all of the text is clear and easy to read. Cooper Hewitt uses a somewhat nontraditional layout, which may be confusing to some visitors in its own right. But all the information I need is right here, and I don't need to zoom in to see it.

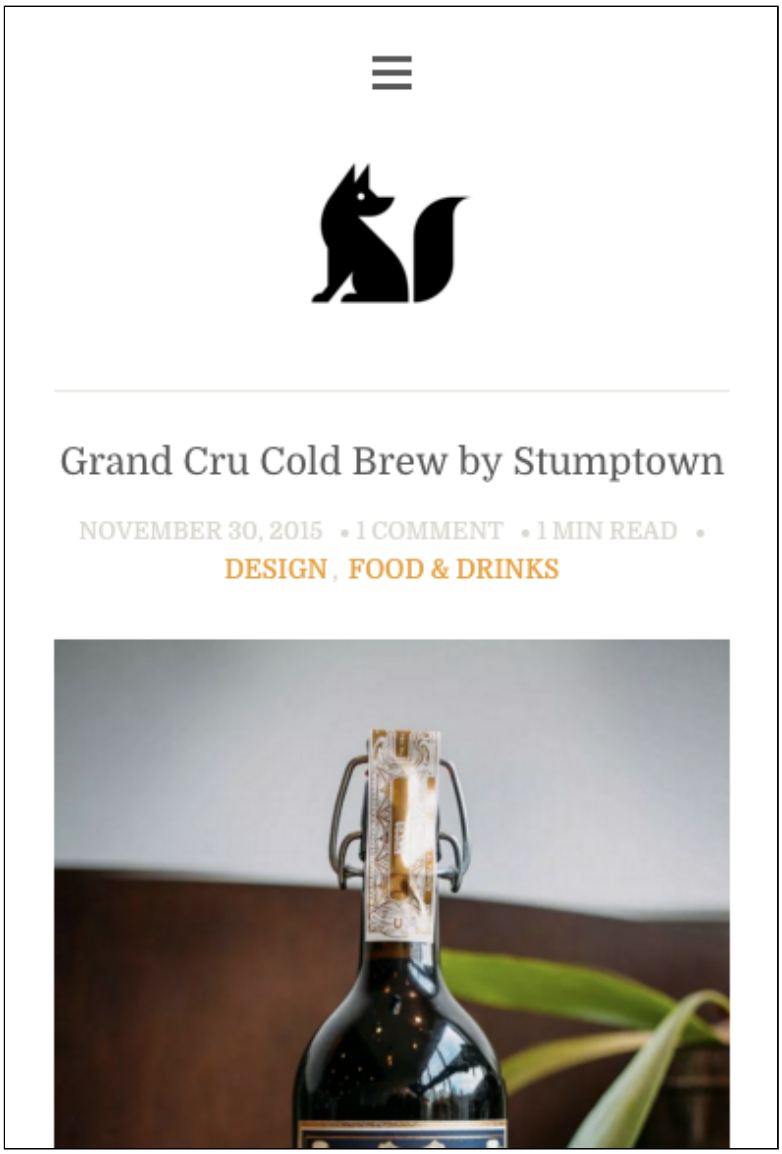
Let's look at one more example of responsive Web design in action, featuring a more traditional layout. Here's the popular design blog The Fox Is Black on my laptop:



And here's what the site looks like when I shrink the browser window to a smaller size:

note

Most designers use responsive frameworks or boilerplates as a starting point for a responsive design.



As you can see, the site looks virtually the same. Instead of getting smaller, the text size has actually increased for mobile viewing. Notice that little 3-line icon at the top? That icon represents a mobile navigation menu. Introducing new symbols can be a challenge, as it's difficult to make a symbol that's universally understood. The 3-line "hamburger" icon confused some people at first, but over time, it's been accepted as a familiar sign of the mobile Web.

It's Not Just About Looks...

Though it's certainly appealing, responsive Web design isn't just about looks; it's about quick access to information and functionality on the go. When out-of-town visitors are trying to choose a restaurant in an unfamiliar city, they're more likely to choose a restaurant with a menu that's readable on a smartphone. And while choosing a restaurant is rarely an emergency (even when you're really hungry), there are situations in which responsive Web design can actually save lives.

The AIDS.gov Web site did a major responsive redesign when they discovered that most emergency searches on the site were coming from mobile phones, and that over 60% of visitors were accessing the site on mobile.

note

The Skeleton boilerplate uses a 960 grid system, with 12 or 16 column options.

A screenshot of the AIDS.gov Web site on mobile. Life-saving information about service locations, prevention, and the Affordable Care Act is front and center.

With a wealth of information quite literally at our fingertips, the easier it is to access that information, the better. So, how do responsive Web sites work, and how do you go about making one?

How Do Responsive Layouts Work?

Head on over to the Cooper Hewitt Web site and try dragging your browser window to a smaller size. At a certain screen size, you'll notice that the content begins to shift around. Responsive layouts work by sending different CSS rules to the Web browser depending on the size of the viewport.

Media queries check for the capabilities of device, not the exact type of the device.

We send these different instructions using something called **media queries**. Media queries are conditional filters that make it easy to display different content

Using rems enables you to override a browser's defaults for font size.

based on the size of the user's screen or device. Media queries tell the browser things like, "If the screen is smaller than 500px, then display the smaller version of this logo." Or, "At 900px and smaller, move the sidebar below the main content area instead of to the side of it." Rather than having to create an entirely separate mobile Web site for visitors, media queries in responsive Web design allow us to create content that adapts fluidly for any device, anywhere.

As you know, there are a lot of different devices that people use to access the Web. Media queries don't try to search for the exact type of device a user is using. There's just too many of them, and new devices are coming out every day. Instead, media queries check for the capabilities of a user's device. Using media queries, you can check for things like:

1. The width and height of a user's device
2. The width and height of the browser window
3. The screen resolution of the device
4. The physical orientation of the device (is it turned in portrait or landscape mode?)

By using media queries to check if a user is accessing our Web site on an iPhone, tablet, PC, or other device, we can leverage the power of CSS to standardize a user's browser experience across the Web, ensuring that our Web site will look and function well whether the user is browsing on a tiny screen or a large desktop computer.

Using Media Queries

There are two different recommended ways to incorporate media queries into your Web site: via a separate external style sheet, or by placing media queries at the end of an existing style sheet. Let's take a look at both methods.

Adding Media Queries via External Style Sheet

The first method involves including a link in the head of your document to separate the style sheets that control your responsive styles.

```
<link rel="stylesheet" media="(max-width: 640px)"
 href="max-640px.css">
```

The above rule says that when the browser is between 0px and 640px wide, the styles included in the max-640px.css style sheet will be applied to the site. You'd place all of the styles for the max-640px screen size in this style sheet. All other styles would go elsewhere.

Adding Media Queries to an Existing Style Sheet

The second method, and the one I prefer, involves using the @media tag to place media queries in their own section at the end of an existing style sheet. Here's an example:

```
@media only screen and (max-width: 480px) {
   h1 {
     font-size: 16px;
```

```
    }
  }
```

This style rule says that when a user is viewing the site on a screen smaller than 480px wide, the h1 headings should drop down to 16px, instead of displaying at a larger font size.

This second method requires no additional style sheets, but it does require some special formatting in your primary style sheet. Notice anything different about the above CSS rule?

@media CSS style rule

```
@media only screen and (max-width: 480px) {
   h1 {
     font-size: 16px;
   }
}
```

Standard CSS style rule

```
h1 {
   font-size: 16px;
}
```

The @media rule is enclosed in double brackets, while the standard style rule is enclosed in single brackets. Think of your @media rules as their own special style sheet nestled inside the existing style sheet. As long as the @media rules are listed after your primary CSS styles, the rules will cascade down, and the responsive rules will overwrite the desktop rules at the appropriate sizes.

The order is important here! If you place your @media rules before the standard CSS rules, they won't work. Here's an example of what a well-formatted cascade might look like in an actual style sheet:

```
header {
        height: 160px;
        background-image: url(rabbit-logo.jpg);
}

h1 {
        font-size: 32px;
}

@media only screen and (max-device-width: 480px) {
header {
        height: 100px;
        background-image: none;
}

h1 {
        font-size: 18px;
}
        }
```

In the example above, the header element is set to 160px tall at the desktop size, with a rabbit logo image set to display in the background. All h1 headings are set to display at a size of 32px. At the bottom of the style sheet, the mobile media queries in double brackets. These rules say that on mobile devices with a maximum screen width of 480px, the header should shrink to 100px tall, the logo image should disappear, and h1 headings should display at a font size of 18px.

Working with Responsive Boilerplates

The key to successful responsive design is a mix of precision and flexibility. It's extremely challenging to build a robust responsive layout from the ground up. In fact, I wouldn't recommend any beginning Web designer attempt to design a responsive site from scratch.

Instead, after building a solid foundation in CSS (as you have done in this course), I would recommend getting started with responsive design using a responsive framework. Today, there are a number of amazing tools that exist to support the creation of clean, accessible, responsive Web sites. These tools, which are sometimes called responsive frameworks or **boilerplates**, are a designer's best friend.

HTML5 boilerplates come bundled with all the files you need to jumpstart the responsive design process. A good boilerplate generally includes a basic HTML index file, a CSS style sheet (or multiple style sheets) with responsive media queries already in place, and sometimes JavaScript files and image icon files.

Why Use a Boilerplate?

So, why use a boilerplate instead of building from the ground up? For one, a boilerplate will significantly speed up the development process. With a boilerplate, you can have a very simple responsive Web site up and running in minutes, instead of the hours it would take to code responsively from scratch. Boilerplates provide a solid foundation to build from, while still allowing you—the designer—to control the important design decisions that make your site unique.

Because boilerplates are built with clean HTML5 code, you can rest assured that you'll be working with a standards-compliant design. The boilerplate we'll be using in this Lesson, and in the upcoming Exercise, is called Skeleton (http://getskeleton.com/). Skeleton is "a dead simple boilerplate" that makes it easy to start designing responsively, providing you all the @media rules and custom classes that you'll need to create a site that functions similarly across many different devices. Let's take a look at all the neat features Skeleton has to offer!

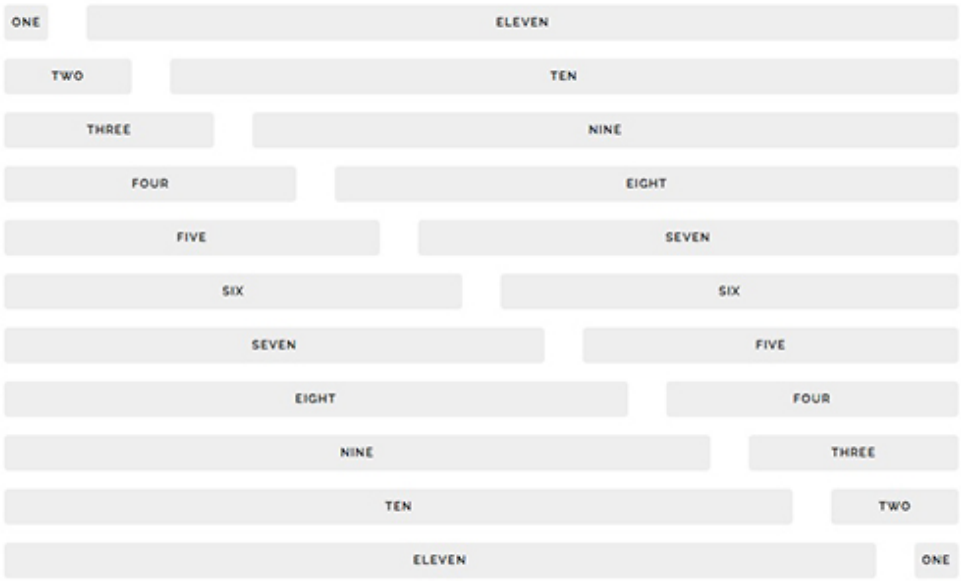
The Skeleton Boilerplate

The Skeleton boilerplate is a cross-browser compatible responsive framework. Skeleton includes default CSS styles and CSS reset styles, so it takes some of the tedium out of the early design process, allowing you to customize your Web site from a functional, attractive starting point.

Using the 960 Grid

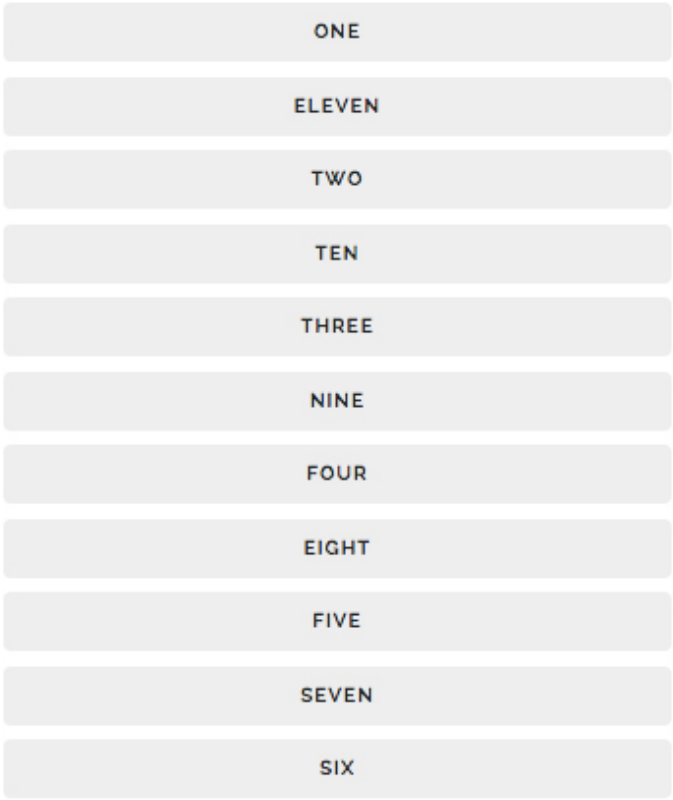
The Skeleton boilerplate makes use of something called the **960 Grid System**. The 960 Grid System, which was developed to help streamline responsive Web development workflow, divides a Web site into a grid-based layout, with a maximum width of 960 pixels for the entire grid. There are two different variants on the 960 Grid: a 12-column layout and a 16-column layout.

Skeleton uses the 12-column layout, which smoothly adapts to fit the device when viewing on a smaller screen size. Here's a demonstration of Skeleton's columns at a large screen size:



A single header might span the entire width of Skeleton's 12-column layout, while a main content area might span eight columns next to a sidebar that spans four.

And here's how those columns shrink down on the mobile layout:



As you're working with Skeleton, you'll want to imagine these columns in your head, as they'll determine how you divide up the content on your page. Will you create a single-column design that spans the entire width of the 960px container, or will you divide the columns into even halves, or thirds?

Skeleton also takes a lot of the guesswork out of setting the correct margin width. Each column inside the 960px container is based on a specific width, with a 10px margin on either side. As long as you leave the native column width and default margins intact, you shouldn't have to add any margins of your own around the various content areas.

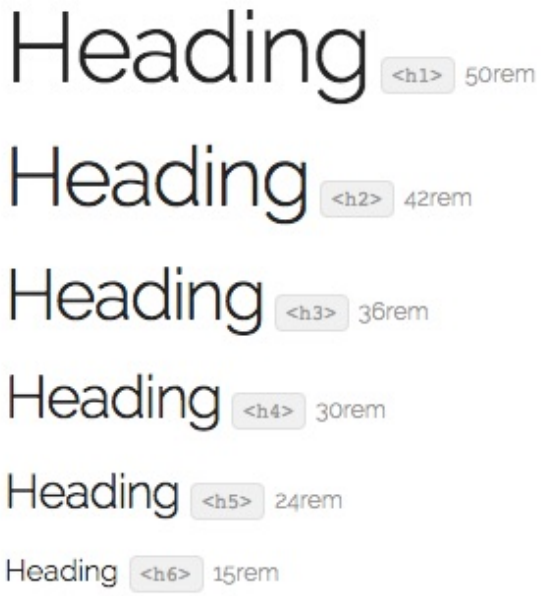
You'll learn how to adjust page columns in Skeleton momentarily. First, let's finish our tour of Skeleton's basic features.

Typography in Skeleton

Typography in Skeleton is set in **rems**. We haven't done much work with rem-based text in this course, so let's review what they are and how they work.

When responsive Web design came to town, designers started searching for flexible alternatives to pixel-based font sizes. Ems and percentages helped with the problem, but they didn't solve everything. Whereas Ems are based on the default browser text size, rems allow you to declare the default text size in your Web page's HTML element, overriding the browser's size defaults.

Setting rem font sizes from scratch requires some careful calculation, but Skeleton has made these calculations easier by setting the `<html>` font-size property to a base of 10. Using the Skeleton calculations, `<h1>` headings with a 5.0rem font-size equals a font-size of 50px (5 x 10 = 50). Text set to 1.4rem equals 14px (1.4 x 10 = 14), 2rems equals 20px, and so on.



In addition to its base font sizes, the Skeleton boilerplate takes advantage of Web fonts, and comes preloaded with the Raleway Google font. We'll talk more about customizing these font styles shortly.

Media Queries

Skeleton uses media queries to help you style your site across a variety of different devices. The default sizes Skeleton uses for queries are:

- Desktop HD: 1200px
- Desktop: 1000px
- Tablet: 750px
- Phablet: 550px (smartphones with large screens)
- Mobile: 400px

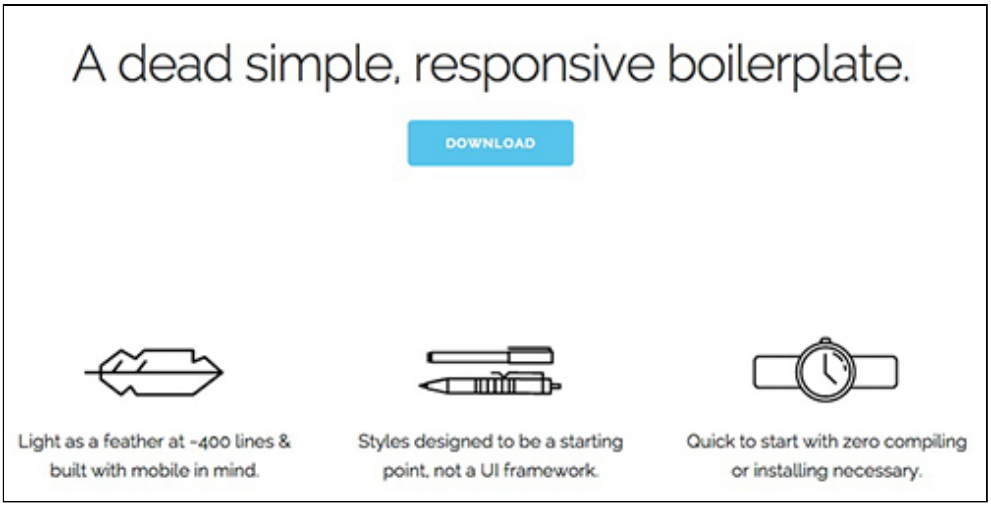
Because Skeleton uses **mobile-first** queries (queries which are optimized for mobile viewing), small devices don't have to run through a bunch of unused CSS before loading the mobile styles. You shouldn't have to change or customize these media queries in Skeleton. As long as you work within Skeleton's guidelines, the content you add should scale smoothly no matter the browser size.

Getting Started with Skeleton

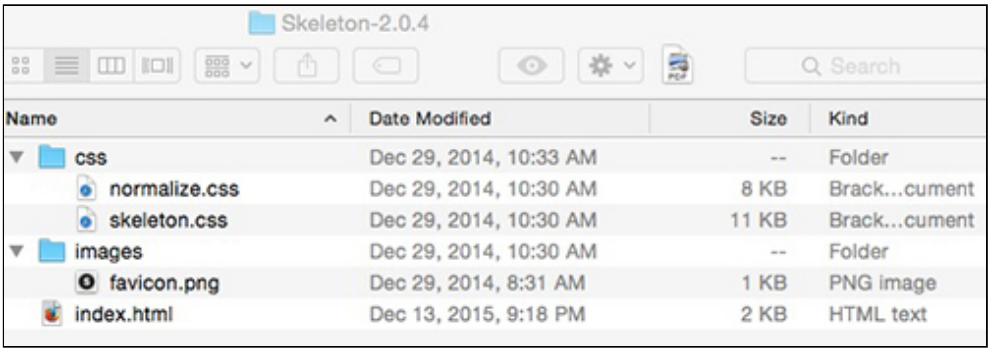
note

The Skeleton download folder has all the components you need to get started.

To get started with Skeleton, head on over to the Web site getskeleton.com and download the Skeleton zip file from the main page.



When the download is complete, double click to unzip the file. The Skeleton folder contains all the basics you need to get started. Take a moment to browse through the file structure to get the lay of the land.



The Skeleton subfolders and files, viewed on a Macbook

Some of these files, like index.html, should look pretty familiar. Let's take a closer look at each of these files to see what we're getting ourselves into.

index.html file - The basic Skeleton HTML page that includes all the necessary initial markup.

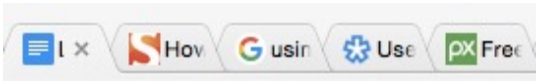
css folder - The normalize.css file contains CSS reset styles and styles that correct common cross-browser compatibility bugs. The skeleton.css file

note

The viewport meta tag controls the dimensions of a Web site on a mobile device.

contains the style rules that keep the Skeleton grid system working. You shouldn't have to mess around with either one of these files!

images folder- The images folder contains the Skeleton favicon. A favicon is a small icon associated with a Web site that gets displayed in a browser tab or bookmark list. You'll learn how to create a custom favicon for your Web site in this lesson.



Favicons make sites easier to identify at a glance.

The Skeleton Document Head

Open the Skeleton index.html file in Sublime. Because there's some additional code and code comments in the <head> section of the document, it may take a moment to get oriented.

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

  <!-- Basic Page Needs
  ------------------------------------------------------ -->
  <meta charset="utf-8">
  <title>Your page title here :)</title>
  <meta name="description" content="">
  <meta name="author" content="">

  <!-- Mobile Specific Metas
  ------------------------------------------------------ -->
  <meta name="viewport" content="width=device-width, initial-scale=1">

  <!-- FONT
  ------------------------------------------------------ -->
  <link href="//fonts.googleapis.com/css?family=Raleway:400,300,600" rel=

  <!-- CSS
  ------------------------------------------------------ -->
  <link rel="stylesheet" href="css/normalize.css">
  <link rel="stylesheet" href="css/skeleton.css">

  <!-- Favicon
  ------------------------------------------------------ -->
  <link rel="icon" type="image/png" href="images/favicon.png">

</head>
```

The gray lines of text that begin with exclamation points are code comments. These are notes about the HTML code that won't show up on the page. View large image

Page Title and Description

Let's take our customizations one step at a time, starting with the page title and meta information. In the <head> section called Based Page Needs, customize the page title to read "Skeleton Practice Site." Then, customize the description content to read "Skeleton boilerplate practice" and customize the author content with your own name.

```
<!-- Basic Page Needs
------------------------------------------------------ -->
<meta charset="utf-8">
<title>Skeleton Practice Site</title>
<meta name="description" content="Skeleton boilerplate practice">
<meta name="author" content="Your name goes here">
```

note

A favicon is the little graphic that appears in your browser's URL bar.

Make sure you've added your customizations between the right set of apostrophes!

Viewport Specifics

The viewport meta tag was introduced in HTML5 to give designers greater control over the viewport area of a user's mobile device. The viewport meta tag controls the dimensions of a Web site on a mobile device.

```
<!-- Mobile Specific Metas
  ================================================== -->
<meta name="viewport" content="width=device-width, initial-scale=1">
```

When the viewport tag is doing its job, the Web site content should scale so a user doesn't have to scroll horizontally to read the text, and doesn't have to zoom in to read the text. You shouldn't have to change the viewport information in Skeleton, but it's important to know what's going on behind the scenes!

Web Fonts

In Lesson Five, you learned how to incorporate Web fonts into your design. The Skeleton boilerplate makes it even easier, as it comes preloaded with the Raleway Web font from Google fonts.

```
<!-- FONT
  ================================================== -->
<link href="//fonts.googleapis.com/css?family=Raleway:400,300,600" rel="stylesheet" type="text/css">
```

View larger image

If you'd like to customize or change this font, then you already know the drill. Select the font (or fonts) of your choice from Google fonts and copy/paste the code into the head section of your document. Then, add the corresponding CSS rules to your style sheet.

But which style sheet? When working with Skeleton, you generally won't want to edit any of the code in the **normalize.css** or **skeleton.css** files. Instead, you'll want to add all custom styles to a new style sheet. Open a new file in Sublime now and save the file as **custom.css**. Make sure to add this file to the Skeleton CSS folder to keep files organized and to maintain a consistent folder structure. Before we start adding custom styles, let's link this new file to the head of our document.

Style Sheet Links

The next lines of code in the head the Skeleton index.html are the links to the normalize.css and skeleton.css files. The order that these files are listed in matters, so don't rearrange the link order or change the current folder structure.

```
<!-- CSS
  ================================================== -->
<link rel="stylesheet" href="css/normalize.css">
<link rel="stylesheet" href="css/skeleton.css">
```

Add a link to your new style sheet below the normalize.css and skeleton.css files, like so:

```
<!-- CSS
-------------------------------------------------- -->
<link rel="stylesheet" href="css/normalize.css">
<link rel="stylesheet" href="css/skeleton.css">
<link rel="stylesheet" href="css/custom.css">
```

Great! Now any CSS that we add to customize our site, from fonts to color styles, to borders and background images, can be added to our custom.css style sheet. This will keep our styles clean and easy to read, and will keep them separate from the core CSS files that keep Skeleton working properly.

Site Favicon

The final bit of code before the closing head tag is for the site favicon. A custom favicon isn't required, but it's a nice touch, and it makes it easier for a user to navigate quickly back to your Web site when they have multiple tabs open in their browser.

If you have access to image editing software, creating a custom favicon is simple. Create a graphic in an image editor of your choice and crop the graphic so the dimensions are perfectly square (all four sides should be the same length). Resize the graphic to 16 x 16 pixels and save the file as favicon.ico. Favicons can also be saved as png or gif files, but no other extensions than .png, .gif, or .ico will work.

```
<!-- Favicon
-------------------------------------------------- -->
<link rel="icon" type="image/png" href="images/favicon.png">
```

The favicon that comes preloaded with the Skeleton boilerplate is a .png image. You're welcome to keep this image for your site, or play around and try making a favicon of your own.

The Skeleton Body

Phew! Now that you're familiar with the head section of the Skeleton index.html file, it's time to move on to the body. As with any Web site, the main content of your site should be placed inside the opening and closing body tags.

The Skeleton boilerplate operates on a grid-based design. All of the content inside the body is organized into rows, and all the rows should be placed inside the Skeleton container class, which has a maximum width of 960px. Here's what the Skeleton index.html file looks like without any modifications made to the code or text content inside the body tags:

```
<body>

  <!-- Primary Page Layout
  -------------------------------------------------- -->
  <div class="container">
    <div class="row">
      <div class="one-half column" style="margin-top: 25%">
        <h4>Basic Page</h4>
        <p>This index.html page is a placeholder with the CSS, font and favicon. It's just waiting
        for you to add some content! If you need some help hit up the <a href="http://www.getskeleton.
        com">Skeleton documentation</a>.</p>
      </div>
    </div>
  </div>

<!-- End Document
  -------------------------------------------------- -->
</body>
</html>
```

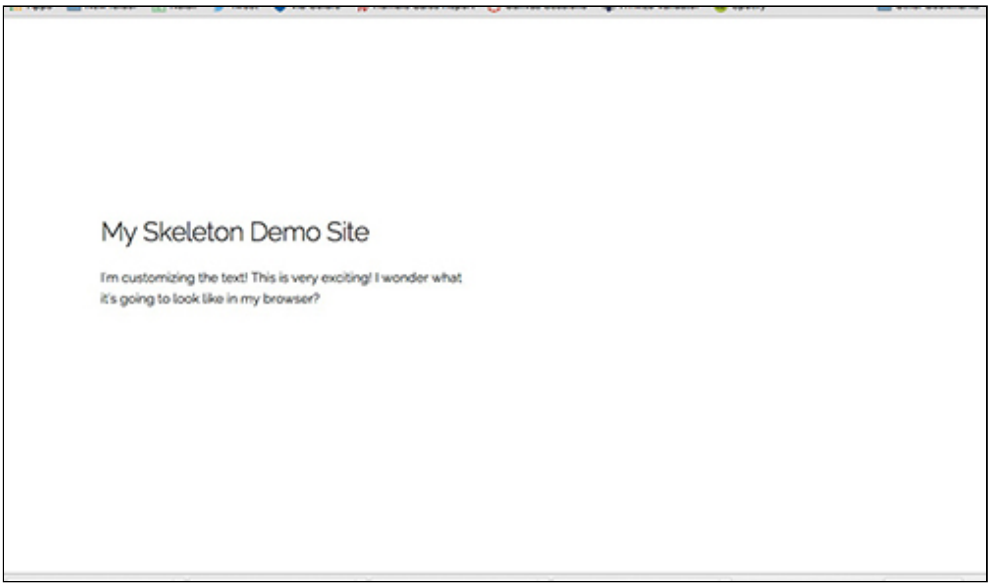
View larger image

And here's what the document looks like in my Web browser:

Take a moment to preview the document in your browser before making customizations. Then, we'll start by making some very simple changes to this page. Without editing the column class or margins, try customizing only the <h4> heading text and paragraph text between the <p> tags with your own content. Here's my custom content added to the source code:

View larger image

And here's a preview of those changes in the browser:



The heading reads "My Skeleton Demo Site," and the custom paragraph text reads "I'm customizing the text! This is very exciting! I wonder what it's going to look like in my browser?"

Bravo! We're off to a good start, but this layout isn't utilizing the full potential of Skeleton's amazing responsive grid. What if we wanted to add a header area that spans the entire width of the page?

To do that, we'll start by adding a new row div. Then, inside the row div we'll tell Skeleton that this row should span across all twelve columns.

```
<div class="container">
   <div class="row">
      <div class="twelve columns">
      <header>
        <h1>Header Title</h1>
      </header>
      </div>
   </div>
   <div class="row">
      <div class="one-half column" style="margin-top: 25%">
        <h4>My Skeleton Demo Site</h4>
        <p>I'm customizing the text! This is very exciting!
I wonder what it's going to look like in my browser?</p>
      </div>
   </div>
</div>
```

Here's what the demo site looks like with a row added for the header:

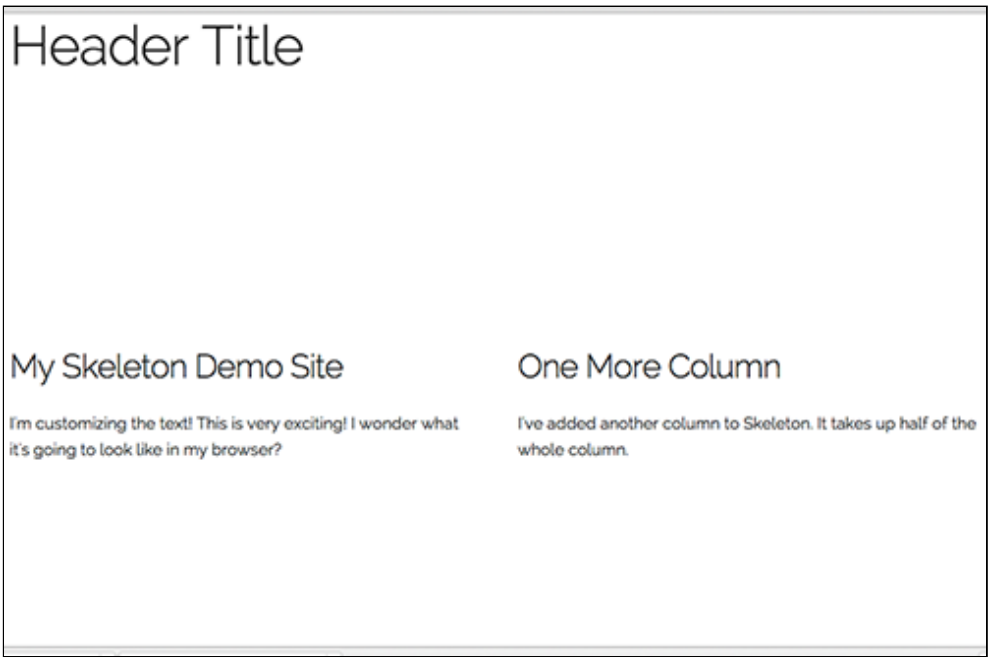
Now, what if we wanted to add another column to the main content area of this page? For two evenly-sized columns, we could merely add another one-half column div next to the one-half column div that's already there.

```
<div class="row">
   <div class="one-half column" style="margin-top: 25%">
      <h4>My Skeleton Demo Site</h4>
      <p>I'm customizing the text! This is very exciting!
I wonder what it's going to look like in my browser?</p>
   </div>
   <div class="one-half column" style="margin-top: 25%">
```

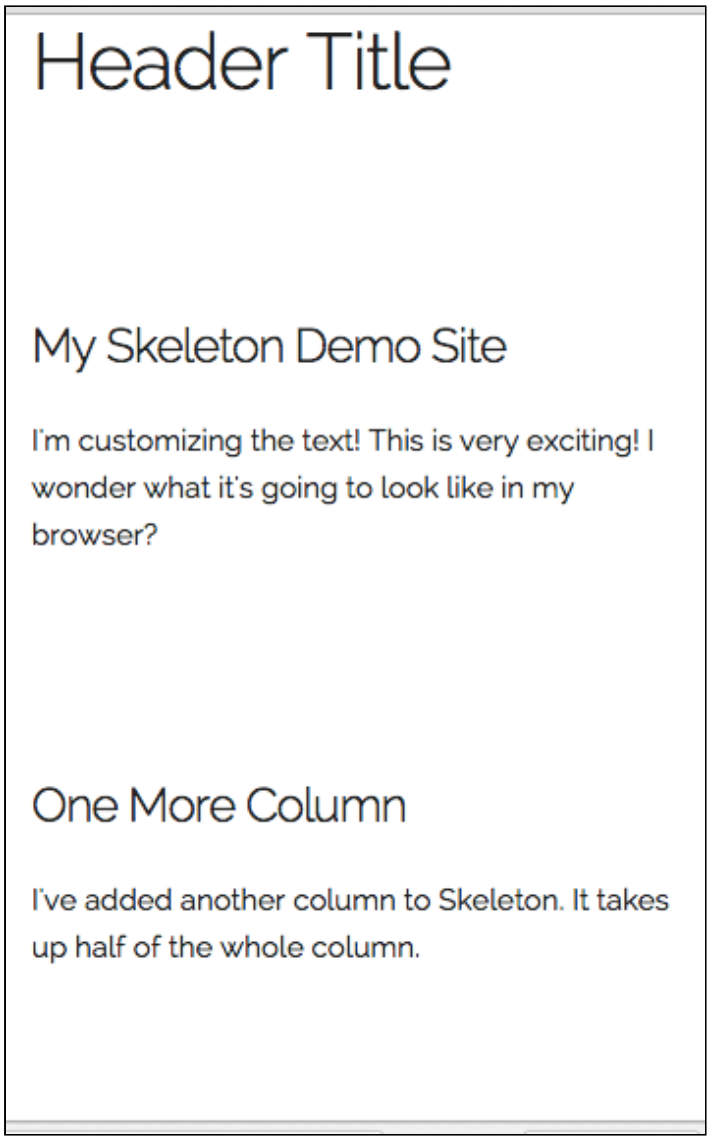


```
      <h4>One More Column</h4>
      <p>I've added another column to Skeleton. It takes up
half of the whole column.</p>
   </div>
</div>
```

Here's what the new, two-column design looks like in the browser on a desktop computer:



And here's what it looks like with the browser scaled to a mobile screen size:



Notice how the right-hand column has shifted position and now appears below the left column? Looks like our media queries are doing good work behind the scenes!

Skeleton provides many custom classes for different-sized column layouts. We won't go over all of them here, but let's review one layout that you should be very familiar with—a Web site layout with a wide main content area next to a narrow sidebar. To accomplish this layout, we don't want to use a row with even half-columns. Let's try a two-thirds and one-third column layout instead!

```
<div class="two-thirds column">
<div class="one-third column">
```

While we're at it, let's remove the inline margin styles from our columns. Those inline styles are there so the Skeleton preview will look nice when you first open the site in your browser. As you customize your site, you'll want to add element margins and padding to your custom.css style sheet.

```
<div class="row">
   <div class="two-thirds column">
      <h4>My Skeleton Demo Site</h4>
      <p>I'm customizing the text! This is very exciting!
I wonder what it's going to look like in my browser?</p>
   </div>
```

```
   <div class="one-third column">
      <h4>One More Column</h4>
      <p>I've added another column to Skeleton. It takes up
half of the whole column.</p>
   </div>
</div>
```

Check it out! Our margins need some work, but now we've got a decent sidebar here! If you're partial to a layout with a left-hand sidebar, just insert the **one-third column** class into your code before the **two-thirds column**.

I've removed the inline margins from the code, as we should be doing all style customization in our custom.css file.

If we wanted to add a footer to our Skeleton site, the steps are similar to those we used for our adding a header. We'd simply add another twelve column row below our existing row, right before the closing div tag for the container class.

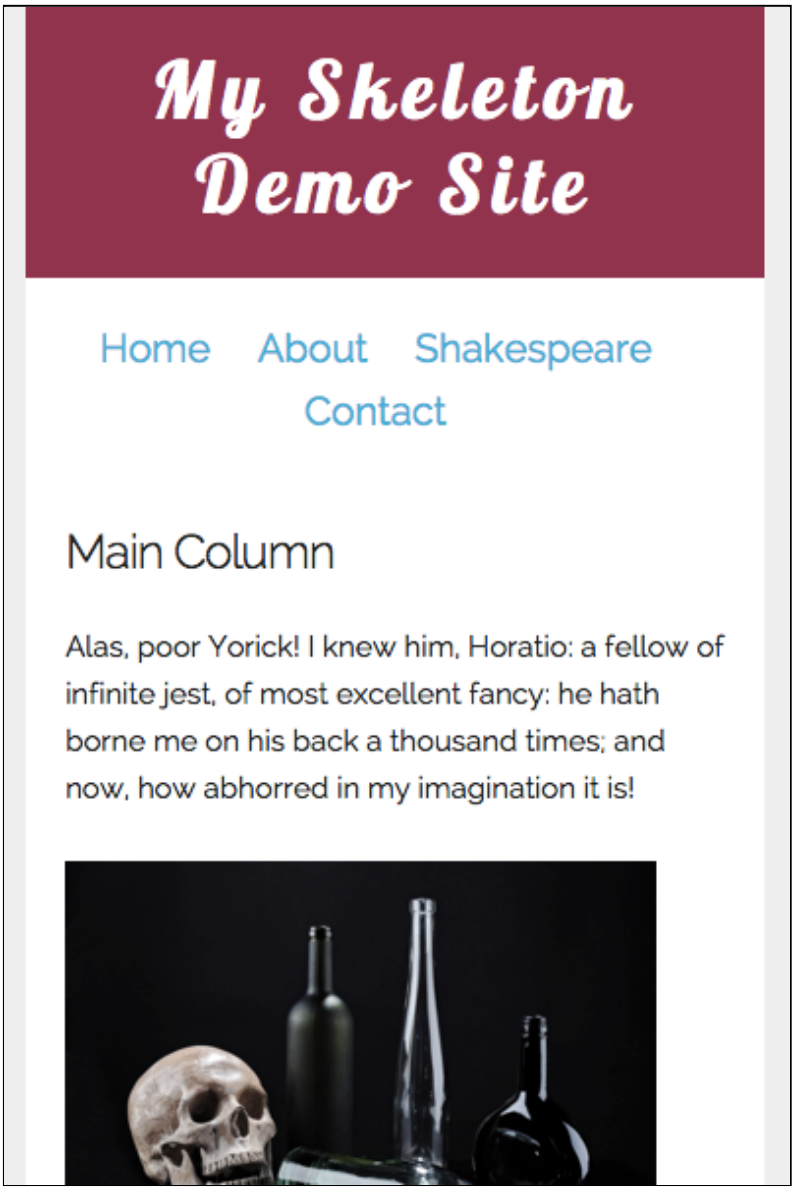
```
<div class="row">
   <div class="twelve columns">
   <footer>
      <p>Footer and ©copyright information goes here.</p>
   </footer>
   </div>
</div>
```

Skeleton Custom CSS

You'll be customizing your own Skeleton site in the upcoming exercise! For now, I'll get you started on the road to responsive design by providing a walkthrough of a simple Skeleton demo site.

Click here to view a finished version of the site, and keep the site open in a separate browser window while we run through the code and styles.

The responsive Skeleton demo site viewed at full size on a laptop computer - View larger image

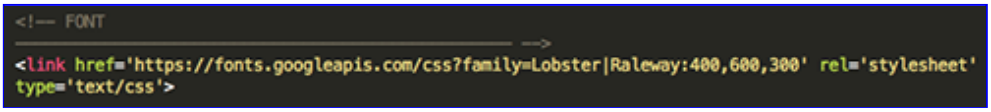


The responsive Skeleton demo site at the mobile screen size.

Demo site HTML

If you take a look at the Skeleton HTML code, you'll notice that I've made virtually no changes to the structure of the Skeleton code. The demo site is using the same two-column layout demonstrated earlier in the lecture (two-thirds column paired with one-third column) and the header, footer, and main container area are all virtually the same.

There's only two notable changes I've made to the core HTML code. The first change is that I've added an additional Web font in the <head> section of the document. The cursive font you see in the site header is called Lobster, and it's a lovely pairing with the default Raleway font style that comes bundled with the Skeleton boilerplate.



View larger image

I invite you to experiment with different Web font pairings on your own responsive site. It's always nice to see different font styles used to separate site headings from paragraph text.

The second notable change I've made is to the footer code. Previously, I'd placed the site footer footer in a row, like so:

```
<div class="row">
   <div class="twelve columns">
      <footer>
         <p>Footer and ©copyright information goes here.</p>
      </footer>
   </div>
</div>
```

There's nothing wrong with placing the footer in a row, but if the footer is part of the main container (along with all of the other rows), you may run into trouble when you start applying custom styles to the container div. For example, if you wanted to apply the min-height property to the container div to streamline the height of all pages, because the footer is just another row in the container, it will jump up into the middle of the page!

There's no code that's inherent in the <footer> element that tells it to stay put at the bottom of your site. If you want to the footer to stay put, it should be placed inside a new container div.

On left, footer placed inside a row in main container. On right, footer placed inside a row in a new container div.

Here's the code I added to keep the footer in place (note the new container):

```
<div class="container">
   <div class="row">
      <div class="twelve columns">
       <footer>
       <p>Footer and ©copyright information goes here.</p>
       </footer>
      </div>
   </div>
</div>
```

And here's the code I changed at the top of the document so both containers could be styled independent of each other:

```
<div class="container main">
   <div class="row">
      <div class="twelve columns">
       <header>
       <h1>My Skeleton Demo Site</h1>
       </header>
      </div>
   </div>
</div>
```

Note that I assigned the name "main" to the first container? This will come into handy when I want to apply special styles (like height) to the main content container without applying those styles to the footer container as well.

Demo site CSS

Now, let's take a look at some of the custom CSS styles on the Skeleton demo site.

```
body {
        background-color: #eee;
}

.container {
```

```
        background-color: #fff;
}

.container.main {
        min-height: 500px;
}
```

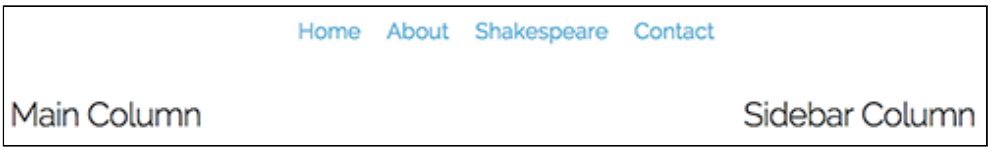
I've styled the entire body element of the site with a light gray background color, with pure white applied to the foreground container. Then, I assigned a min-height of 500px to the main container (.container.main) to streamline the height of the main content area across all pages.

I was noticing that the columns were a little cramped on the demo site, so I applied a generous margin to the top of the column div.

```
.column {
        margin-top: 4rem;
}
```

Here's what the column spacing looks like with and without that top margin:

No top margin applied to column class



4rem top margin applied to column class

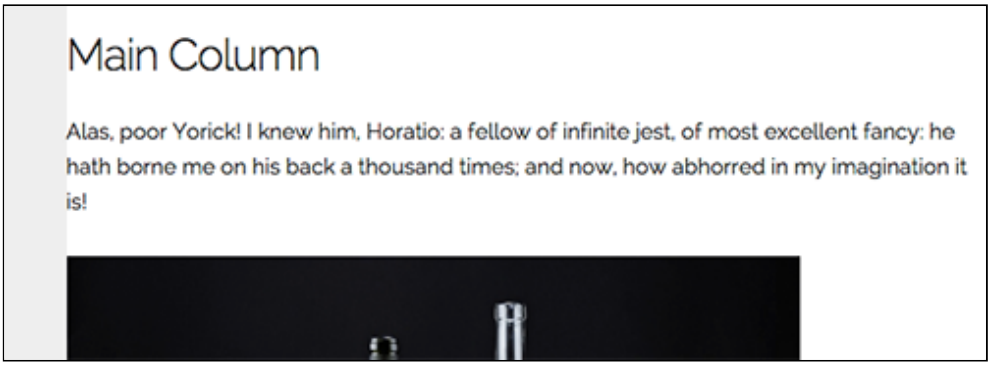
I assigned a fixed height to the header and footer element and grouped some of the styles together, as shown below:

```
header {
        height: 100px;
        padding: 20px;
}

footer {
        height: 80px;
}

header, footer {
        text-align: center;
        background-color: #9E1E4C;
        color: #fff;
}
```

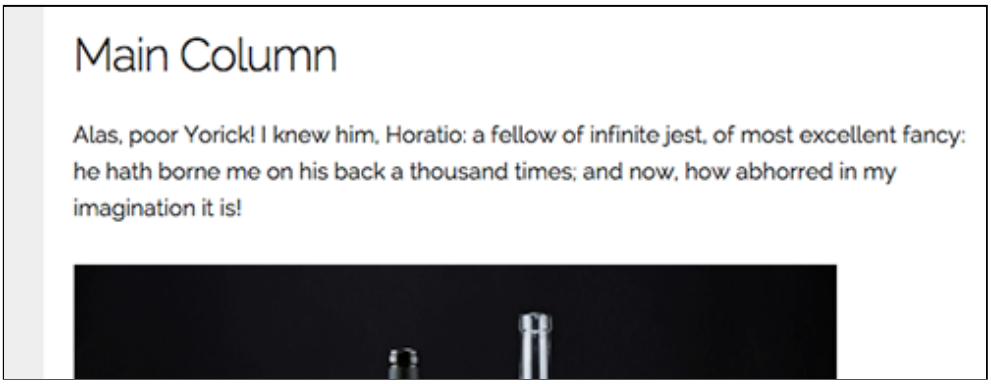
Because the header and footer both share centered white text and a wine-colored background, there's no need to style these separately. Remember that multiple selectors can be styled at one time, separated with a comma.

As you begin styling block-level elements on your own Skeleton site, you may find that the text needs a bit of breathing room. I noticed that the text and images were bumping up against the edges of the container, so I added a bit of padding to remedy the problem:

```
h4, p, img, ul {
        padding-left: 20px;
        padding-right: 20px;
}
```



Text and image before adding 20px of padding to the left and right sides.



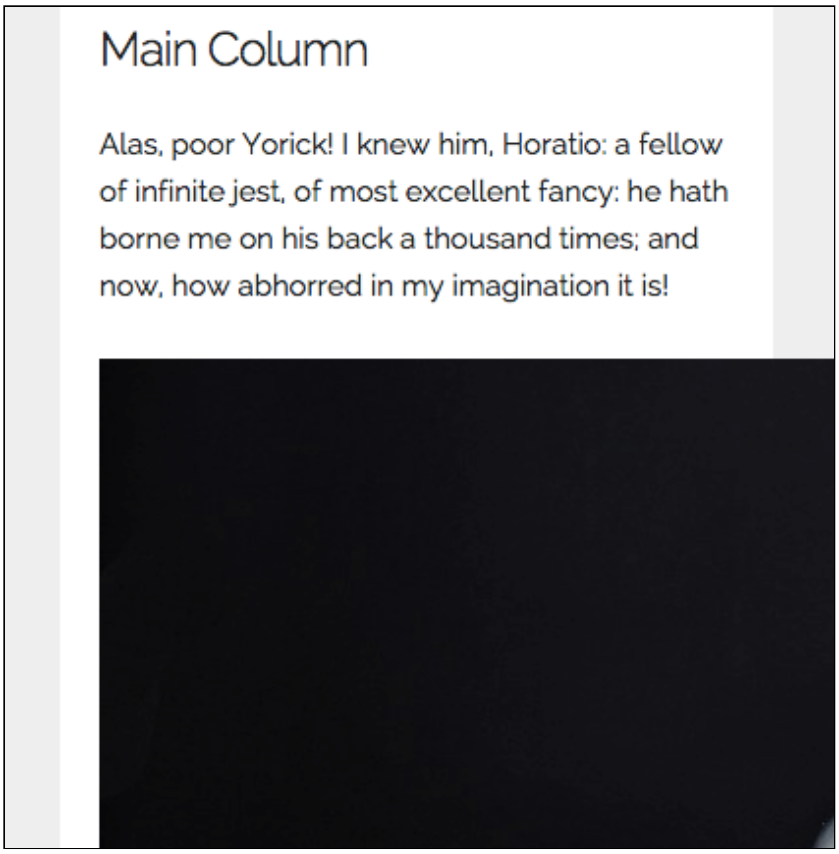
Text and image after applying 20px of padding to the left and right.

Speaking of the demo site image, notice that the image scales smoothly and proportionally no matter the size of the site? To achieve that effect, you'll need to add responsive image scaling to your custom.css style sheet. Scaling images responsively requires only two lines of CSS, but forgetting to include those lines of code can lead to a broken mobile site!

```
img {
        width: 80%;
        height: auto;
}
```

The above CSS rule says that all images on the site should be scaled to 80% of their full width, and that the height should automatically scale in proportion to the width. The width property on your own site may change based on the original size of the images. You may set the width to 100% or you may set it to 50%. The important thing is that the height property is set to "auto."

Want to see what the Skeleton demo site looks like without responsive image scaling?



Oops! The image is so big that I can't even see the Skeleton, and the image extends beyond the boundaries of the container.

It's important to remember that setting a width of 80% or 50% in your style sheet doesn't change the original size of the image—it only changes the size the image displays on the page. If the original images are large, high-resolution files, I recommend first resizing your images in a photo editing program like Photoshop. Your site will load faster for the user, and you won't have to do extensive image resizing in the code.

I've applied many custom styles to the Skeleton demo site, and I won't be showing every one here. It's your responsibility to look through the custom.css style sheet and to match styles with the various elements on the page. Take note of the different font sizes and letter spacing, link hover styles, and the customized blockquote. Note which of the core Skeleton styles I've overwritten in the demo, and which of the core styles I've left intact.

Take some time to play around with these sample styles. I expect you to wow me with your final responsive design, so you'll want to have a good handle on how to customize your CSS styles in Skeleton before diving into the final project!

Coming Up Next:

Discussion
Discuss the benefits and challenges of responsive Web design in the Discussion.

Exercise
Redesign a five-page site as a responsive site, using the Skeleton framework.

HTML and CSS | Responsive Redesign

sessionscollege
FOR PROFESSIONAL DESIGN

Exercise 6

Responsive Redesign

Congratulations! You've now traveled through the dark forest of Web design, and you've emerged victorious on the other side.


Prince Hamlet ponders the complex depths of the Skeleton boilerplate.

Before you take on the final assignment, I'd like you to take a moment to reflect on how far you've come. In a short period of time, we've elevated our skills from basic HTML markup, to semantic HTML5, to advanced CSS techniques and responsive Web design.

For the final assignment, you'll be working with some familiar content in an entirely new way. Working with the site you designed in Exercise Five (either the Zen Center or Lift This! Fitness, you'll revise the site, migrating its content into a clean, responsive design, using the Skeleton boilerplate as a base.

Your new Web site will contain the same text content, images, and number of pages as the previous iteration but will incorporate the responsive design principles you learned about in this lecture: scalable images, grid-based layouts, media queries, rems, and more.

Performance Objectives

- Redesign a five-page site as a responsive Web site.
- Using the Skeleton framework, create a two-column layout with a horizontal primary navigation menu, and a vertical secondary navigation menu.
- Make sure the site scales effectively at different browser widths and presents information in a logical, readable manner.
- Incorporate in your design a Web font, CSS3 style, and a favicon.

note

Assignments are evaluated for creativity, technical

Project Brief: Responsive Web Site Project

To design responsively, or not to design responsively, that is the question!

When responsive design is an option, creating a Web site that looks good and functions properly on a mobile device is always the answer. In an age of on-the-go browsing, everyone with a Web site is looking to upgrade their design to a more mobile-friendly format. Are you up for the challenge?

In this project, you'll begin by creating a responsive template for the Home page of your site. After testing to make sure your page validates properly, and that the layout scales well in your browser, you can replicate that page layout to build out the other pages on your site. After cloning the basic layout for every page, you'll need only to customize the page content with the correct text and images.

proficiency, and understanding of concepts covered in the lecture.

Here are the primary requirements for this assignment:

- Use the text content and images from the site you created in Exercise Five. The text files and images are available to download here.
- Build a two-column responsive layout utilizing Skeleton's 960 grid system.
- Include a horizontal primary navigation menu, and a vertical secondary navigation menu in the sidebar.
- Use valid HTML5 semantic tags (header, footer, section) to mark up your content. You may use article and aside tags as well, but these are not required.
- Test your design in multiple browsers to make sure the layout scales for mobile.
- Include at least one CSS3 feature and one Web font.
- Customize the site typography to match the look and feel of your design.

If you worked with the Zen site in the last exercise, and you'd like to switch it up, you're welcome to work with the content for the fitness site this time around. And vice versa.

Getting Started

1. Make a Plan

Before you start adding content to the Skeleton boilerplate, take the time to plan the layout of your new site. Are you going to use an evenly divided two-column layout, or a two-thirds and one-third column layout for the main content and sidebar? You can also use the Skeleton Demo site from Lesson 5 to help you plan.

Think about which design elements from your last site you'd like to carry over to your responsive redesign, and which elements you'd like to change. If you feel like the color palette and typography didn't quite create the feel you were going for, now is the time to try something new.

2. Create the Site Template

You should have already downloaded the Skeleton boilerplate. If you want to start with a blank slate, download the files again to work from a pristine index.html file.

Open the Skeleton index.html in Sublime. Starting with the Home page, customize the title and meta information in the document head. Then, begin incorporating the text and images into the Skeleton boilerplate. The order of divs is very important in Skeleton, so you'll want to be mindful of where you're placing your content, and where you're closing those div tags.

Remember to visualize your design in rows and columns as you work. Begin with the Skeleton **container** class. Then, place rows inside the container and

place columns inside rows. You may be able to incorporate all of the site content inside a single container, but if you have trouble positioning the footer, you can begin a new container class to anchor the footer at the bottom of the page.

Be sure to validate your work before moving on to the next steps.

3. Style the Site Template

With the proper markup in place, it's time to get styling! Create a new CSS style sheet called **custom.css** and save the file in the Skeleton CSS folder. Remember, you shouldn't need to edit any of the core Skeleton CSS files, but you're welcome to browse through them to see which classes you can use to customize your design. Add a link to the stylesheet in the head of the Skeleton index.html file, after the normalize.css and skeleton.css links.

```
<!-- CSS
  ------------------------------------------------------------------ -->
<link rel="stylesheet" href="css/normalize.css">
<link rel="stylesheet" href="css/skeleton.css">
<link rel="stylesheet" href="css/custom.css">
```

Customize your typography: Skeleton comes prepackaged with the elegant Raleway Web font. Raleway is a lovely font, but it's not going to be appropriate for every design.

If you're working with the Zen site content, and you'd like to keep Raleway on your site, I'd like to see it paired with one additional font. If you're working with the fitness site content, I'd like to see you use a different Web font (or fonts) in your design. Use no more than two different font families on your site, and feel free to combine the Web font of your choice with a browser-safe font choice from CSS font stack.

- google.com/fonts
- cssfontstack.com

Use different styles for the site headings and paragraph text to help the headings stand out. Use rems to control the size of the various headings, paragraphs, and lists.

Customize your palette: If you like the palette you worked with in Exercise Five, then there's no reason to reinvent the wheel. Assign background colors and, if desired, background images to the various block-level elements in your design. Whichever colors or background images you choose, make sure the text is readable in the foreground.

Watch for common Web color combo faux pas, like red-on-green, busy backgrounds, and vibrating colors.

Style your images: You should already have added an image to the Home page, but you may have noticed that the image extends beyond the boundaries of its containing column. Assign responsive CSS rules to the img element in your custom.css style sheet so the images on your site scale properly.

```
img {
        width: 100%;
        height: auto;
}
```

Begin with a width of 100% and work backwards from there. The auto height property will maintain the proportions of the image so the image looks balanced at any size. If you need to style multiple images in different ways, you can add custom classes as needed.

Want to include a CSS background image to kick this design up a notch? Search Pixabay to find a high-quality, free-to-use background texture. (pixabay.com)

Adjust borders, padding, and margins: You'll probably need to adjust the margins on your headings and paragraphs so the text content doesn't bump up against the edge of the container. Tweak margins and padding to your liking, being mindful of the native dimensions included for the Skeleton columns and container. If you start adding margins to a column without decreasing the column width, you may find that the content on your page pops out of alignment.

Adjust links and list styles as desired: Finally, remember to customize link styles and list styles on your page. Skeleton has done us the favor of taming down the default blue browser link styles to a gentle aquamarine, but you'll need to add your own special styles to these links. Remember to include link pseudo classes to add visual interest and interactivity!

```
a:link
a:visited
a:hover
a:active
```

4. Test Your Site (And Test Again!)

Happy with your design? Now's the time to test it on multiple browsers before you try to create any additional pages from your template. Check your work in Google Chrome, Firefox, Safari, and IE or Microsoft Edge, looking for any formatting issues or inconsistencies. Are all of the responsive elements functioning properly?

If you have a smartphone or tablet, now is the perfect time to upload your site to see what it looks like on a mobile device. If you don't have access to a mobile device, you can check the responsive functionality of your site just by dragging the browser window to a smaller size on your desktop or laptop computer.

Is your Web site attractive and accessible even at a smaller screen size?

If your page has some issues, try to correct them yourself. Save an extra version of your pages and try to isolate the problem in your code. You don't want to create any additional pages until your design has "passed" this step.

If you've tried to correct your page and are still unable to get it to work, you can send me the link to your uploaded page via the Canvas mail inbox (don't submit it via the Dropbox). I'll try to troubleshoot the issue before you proceed.

5. Duplicate Your Work

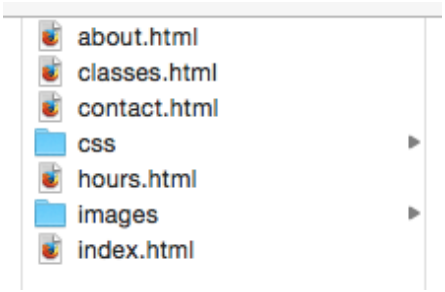
After checking and validating your Home page, it's time to incorporate all of the text and images onto the other site pages. In your Skeleton folder, right-click (PC) or control-click (Mac) on your **index.html** page and select Save As from the dropdown. (For newer Mac users, the Save As function may replaced by Edit/Duplicate and Rename.)

Save the file to create the other pages in your site.

Make sure the files have lower-case names with no spaces, and are all located in the same local folder. For example:

index.html
about.html
classes.html
hours.html
contact.html

When your work is complete, the local folder structure should look like this:



Once the pages are created, you will need to carefully place the content for each page in its place. Page titles, headings, paragraph content, and images will need to carefully be placed inside the appropriate tags.

Check your work as you go by previewing the HTML pages in the browser. And, I'll say it just once more for good measure, be sure to validate your work.

Before uploading your work, finalize the primary site navigation, and check that every page links up properly with every other page. Are all internal links working?

Do all external links lead to the correct pages? Do the site images display correctly? Does the site look amazing and you just can't wait to share it with the world? That's great!

Out of the rabbit-hole, and into the world!

I look forward to seeing your work and hearing about your experience. Though this is the final assignment for this course, this is only the beginning of your Web design journey.

Grading Criteria:

What your instructor expects you to do:

☐ Show the ability to redesign a five-page non-responsive site into a responsive Web site.

☐ Demonstrate the ability to create a two-column layout with a horizontal primary navigation menu, and a vertical secondary navigation menu, using the Skeleton framework.

☐ Show the ability to create a responsive site that scales effectively, presents information in a logical and readable manner, at different browser sizes.

☐ Show the ability to create a responsive site with attractive and consistent layout, color, and typography choices for your chosen client.

☐ Show the ability to create a clean, standards-compliant code and test your site so that it has a minimum of bugs or errors.

How to Post:

Once you're done, go to the Dropbox for this exercise and post your URL for this assignment along with your written comments.

If you have a question before sending your completed exercise for grading, be sure to contact your instructor via the People area.

I look forward to seeing your work!