**Coding Dojo notes: HTML & CSS**

01/23/17 (Day 1): **HTML & GIT**

Front-end technologies:

* HTML: HyperText Markup Language (today)
* CSS: Cascading Style Sheets (next few days)
* JS: Javascript (next week)

Back-end technologies (web server):

* Python
* Java
* PHP
* C#
* Go
* Ruby

Logic / Framework [chef] performs request-response with database (also a server) [pantry].

JSON: Javascript object notation

Example head tag:

<head>

<meta charset="utf-8">

<title>My Awesome Webpage</title>

<meta name="description" content="The text here describes what the webpage

is about. It's what will show up in search results for search engines like

google under the title of the webpage. It's important that this is relevant

to your page and well written!" >

<link rel="stylesheet" href="my\_css\_file.css">

<script src="my\_javascript\_library.js"></script>

<script src="another\_javascript\_file.js"></script>

</head>

|  |  |
| --- | --- |
| <head> | Opening head tag to indicate that we are about to talk about the properties of the document |
| <meta charset=”utf-8”> | Declare encoding to a browser through a meta tag in the header. Without this tag, browser may not know to switch to the proper encoding and characters may be displayed as gibberish. |
| <title></title> | Title of your web page: name of bookmark, name of tab, and what is displayed in search result of search engine. |
| <meta name = “description” content = “description content”> | Description meta tag is used by search engines when displaying results. |
| <link rel=”stylesheet” href=”my\_css\_file.css”> | Links a stylesheet to page. |
| <script src=”my\_javascript\_library.js”></script> | Links a JavaScript or jQuery file. |
| </head> | Closing head tag. Done talking about properties of page and can move on to body. |

Common body tags:

|  |  |
| --- | --- |
| <h1></h1> | A heading is a section title. Each section (<div>) will have a heading. There are **6** levels of heading, each indicating importance of its section. |
| <p></p> | Any chunk of text is a paragraph and therefore, needs to be encapsulated in paragraph tags. |
| Images | There are two ways that we use images on a web page: as **page elements** (such as album art in Pandora, or the photos in your Facebook feed), or as **background images** (this is covered in CSS section).  It has two required attributes: **src and alt**. The **src** attribute stands for source. This is the link to where the image is residing. The **alt** attribute stands for **alternate**. This is a few words of text to describe the image, in case it fails to load. This is also used by screen readers, for the vision-impaired. If (and only if) we also specify its **height and width attributes**, this text will show up where the image should be in case the image fails to load. |
| Links / <a></a> | <a> tag (anchor tag) with **href** attribute.   * Absolute URL: points to another website * Relative URL: points to a file within a website * Anchor URL: points to an anchor inside a page |
| <ol></ol> | Ordered lists (numbered) |
| <ul></ul> | Unordered lists (bullets) |
| <li></li> | List item |
| Tables: <table>, <thead>, <th>, <tbody>, <tr>, and <td> | Table parts:   * A table head (<thead>), which contains rows (<tr>) and column names (<th>). * A table body (<tbody>), which contains rows (<tr>) filled with table data (<td>). |

**Forms:**

One of the most important HTML tags to learn. Responsible for all data exchange between user and server. Form’s job is to take in user input and send it to the back-end to be processed. Form is declared by using the **<form>** tag, which will have attributes action and method that decide where the form info gets sent, and how it gets sent respectively.

Taking in the input is done by using input fields **(<input>)** tag. Each input will typically also have a label (**<label>**) that is the name of the field. To make sure that a specific label is linked/associated to a specific input element, we must add corresponding ***for attribute*** on the label with input’s ***id attribute***. *Including a label tag around the input field is a convention that allows us to click on the label to focus on the input field.*

A **name attribute** will typically go hand in hand with your input tags. They are mainly used for sending form data to the back end.

Examples of input types:

* When the user needs to enter a short amount of text, such as an email address or name.

The appropriate input type here is **text**.

<label for="first\_name">First Name:</label>

<input type="text" id="first\_name" name="first\_name">

<label for="last\_name">Last Name:</label>

<input type="text" id="last\_name" name="last\_name">

<label for="email">Email:</label>

<input type="text" id="email" name="email">

* A password field

The appropriate input type here is **password**.

<label for="password">Password:</password>

<input type="password" id="password" name="password">

* When the user can choose only 1 option from a variety of options. A good example is a gender selector.

One appropriate input type here is **radio** buttons. Example (button on right of text):

<label for="male">Male</label>

<input type="radio" id="male" name="gender" value="male">

<label for="female">Female</label>

<input type="radio" id="female" name="gender" value="female">

Another option is a dropdown menu, which uses **<select>** and **<option>** tags. Example (checkbox on right of text):

<select name="gender">

<option value="male">Male</option>

<option value="female">Female</option>

</select>

* When the user can choose multiple things from a variety of options, such as choosing their favorite 3 colors from 5 options.

The appropriate input type here is **checkboxes**.

<label for="blue">Blue</label>

<input type="checkbox" id="blue" name="color" value="blue">

<label for="green">Green</label>

<input type="checkbox" id="green" name="color" value="green">

<label for="red">Red</label>

<input type="checkbox" id="red" name="color" value="red">

<label for="black">Black</label>

<input type="checkbox" id="black" name="color" value="black">

<label for="purple">Purple</label>

<input type="checkbox" id="purple" name="color" value="purple">

* When the user might want to enter longer text. This can be used in forums for comments, or for user profile descriptions.

In this case we use the **<textarea>** tag.

<textarea name="description"></textarea>

* When a form needs a submit more than just user input.

Input type **hidden** is similar to text fields, except they do not show on the page and users can not enter anything into them. This is useful for the back-end authentication and passing data.

<input type="hidden" name="id" value="7">

* When a date needs to be inputted.

Appropriate input type is **date.**

<input type=”date” name=”bday”>

* To create a submit button.

Appropriate input type is **submit**.

<input type="submit" value="Submit">

*Sample form source code:*

<form action="process.php" method="post">

<p>Please Register</p>

<label for="first\_name">First Name:</label>

<input type="text" id="first\_name" name="first\_name">

<label for="last\_name">Last Name:</label>

<input type="text" id="last\_name" name="last\_name">

<label id="email">Email:</label>

<input type="text" id="email" name="email">

<p>Select your gender:</p>

<label for="male">Male</label>

<input type="radio" id="male" name="gender" value="male">

<label for="female">Female</label>

<input type="radio" name="gender" value="female">

<p>Select 3 of your favorite colors:</p>

<label for="blue">Blue</label>

<input type="checkbox" id="blue" name="color" value="blue">

<label for="green">Green</label>

<input type="checkbox" id="green" name="color" value="green">

<label for="red">Red</label>

<input type="checkbox" id="red" name="color" value="red">

<label for="black">Black</label>

<input type="checkbox" id="black" name="color" value="black">

<label for="purple">Purple</label>

<input type="checkbox" id="purple" name="color" value="Purple">

<p>Say a few words about yourself:</p>

<textarea name="description"></textarea>

<label for="password">Password:</label>

<input type="password" id="password" name="password">

</label>

<label for="pw\_confirm">Password Confirmation:</label>

<input type="password" for="pw\_confirm" name="password\_confirmation">

<input type="submit" value="Click here to register">

</form>

\*\*Other label-input Declaration:

<label>

Text <input type=”text” name=”name”>

</label>

Notice that the **input** element is now **nested inside the label element** and we no longer need to link the two using the label’s for attribute and the input's id attribute.

**<div> & <span>:**

*Span and div signify no specific meaning besides the generic grouping of content* and are, therefore, more appropriate for creating an organization or stylistic additions without signifying superfluous meaning.

Divs define logical sections of a document and should be used sparingly under other circumstances. Ideally, div tags are used to group a number of elements that are meaningful together. They are used for layout purposes. Use them for styling (CSS) purposes only (no child elements) when absolutely necessary.

Example doc template:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>My Awesome Webpage</title>

<meta name="description" content="The text here describes what the

webpage is about. It's what will show up in search results for search

engines like google under the title of the webpage. It's important

that this is relevant to your page and well written!">

<link rel="stylesheet" href="my\_css\_file.css">

<script src="my\_javascript\_library.js"></script>

<script src="another\_javascript\_file.js"></script>

</head>

<body>

<div id="wrapper">

<div id="header">

<h1>Logo</h1>

<ul>

<li>Home</li>

<li>Contact Us</li>

<li>Information</li>

</ul>

</div>

<div id="main-content">

<h2>Article One</h2>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. gravida augue,

a eleifend dolor. Sed sagittis ut odio vitae porta. Mauris

et sapien lacinia, ornare nulla sit amet, tincidunt turpis. Integer

fringilla sem a odio condimentum, vel ullamcorper nulla sagittis.</p>

<h2>Article Two</h2>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. gravida augue,

a eleifend dolor. Sed sagittis ut odio vitae porta. Mauris

et sapien lacinia, ornare nulla sit amet, tincidunt turpis. Integer

fringilla sem a odio condimentum, vel ullamcorper nulla sagittis.</p>

</div>

<div id="side-menu">

<img src="img/dojo\_image.png" alt="picture of the dojo">

<ul>

<li>Page 1</li>

<li>Page 2</li>

<li>Page 3</li>

<li>Page 4</li>

<li>Page 5</li>

</ul>

</div>

<div id="footer">

<a href="/home">HOME</a>

<a href="/contact">CONTACT US</a>

<a href="/info">IFORMATION</a>

</div>

</div>

</body>

</html>

The **span** tag is often used for small bits of text inside paragraphs. Example:

<p>A great quote by Arthur C. Clarke states that <span id="quote"> any

sufficiently advanced technology is indistinguishable from magic.</span></p>

As opposed to *divs, which are block elements*, *spans are inline elements* only intended for small bits of information. They are still container elements, like divs, but only encompass the text they surround.

**Avoiding headaches for HTML:**

* Use proper document structure (i.e. <!DOCTYPE html>)
* Constantly validate your code
  + <https://validator.w3.org/#validate_by_uri>
* Organize HTML Syntax:
  + Use lowercase letters within element names, attributes, and values.
  + Indent nested elements.
  + Use double quotes, not single or completely omitted quotes to store in values for HTML attributes.
* Avoid using too many divs.
* Make use of Semantic Elements.
  + Semantic HTML is the use of HTML markup to reinforce the semantics, or meaning, of the information in webpages rather than merely to define its presentation or look.
* Keep your tab names lowercase.
* Use *alt* attribute with images.
  + The alt information is helpful when a user cannot view your image.
  + <img id="logo" src="images/logo.png" alt="My First Website's logo">

**Additional Tags:**

* DOCTYPE:

|  |  |
| --- | --- |
| HTML 4.01 | HTML 5 |
| <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd"> | <!DOCTYPE html> |

* Character Encoding

|  |  |
| --- | --- |
| HTML 4.01 | HTML 5 |
| <meta http-equiv="content-type" content="text/html; charset=UTF8"> | <meta charset="utf-8"> |

* Script and Link Tags

|  |  |
| --- | --- |
| HTML 4.01 | HTML 5 |
| <script type="text/javascript" src="main.js"></script>  <link rel="stylesheet" type="text/css" href="style.css"> | <script src="main.js"></script>  <link rel="stylesheet" href="style.css"> |

* Canvas

Once an image has been served up to a browser, its contents cannot be updated. You can create animated gifs or update the image's styles, but its contents cannot be updated. The canvas element can now be used *for creating dynamic images*.

<canvas id="work\_of\_art" width="400" height="200">

<p>Your browser does not support canvas</p>

</canvas>

* Audio

Embedding an audio file in an HTML5 document is simple.

<audio src=”desperado.mp3”>

</audio>

Problem with the audio element isn’t the specification, it is audio formats. MP3 format is widely used everywhere, but it isn’t an open format. This is why Safari will play back MP3 files while Firefox will not. The Vorbis codec (.ogg) is an open format. Firefox supports Ogg Vorbis, but Safari does not.

Workaround…

<audio controls>

<source src="desperado.ogg">

<source src="desperado.mp3">

</audio>

* Video

Flash plugin is currently most popular of displaying video content, but HTML5 might change that. Video element works like the audio element. Main difference is that you may want to provide dimensions for your video element. Once again, there is a battle amongst video formats: .ogv and .mp4 files.

<video width="400" height="200" poster="picture.jpg" controls>

<source src="caddyshack.ogv">

<source src="movie.mp4">

</video>

One of the limitations of relying on a plug-in for rich media is that plug-in content is sandboxed away from the rest of the web page. By having native rich media elements in HTML means that we can have our media content play nicely with other browser technologies such as CSS and JavaScript.

**FAQ’s:**

* What is the difference between an id attribute and a name attribute?

The **id attribute** identifies your element for the **front-end** (CSS and JavaScript.) It can be used on **ANY element**.

The **name attribute** belongs **ONLY on form elements** and is used in the back-end (PHP, Ruby on Rails, etc.) to identify your form's values.

* What is DOM?

The **Document Object Model (DOM)** is a structured representation of your HTML as generated by the browser, allowing access to the elements of your web page so they may be manipulated. *Generally, it is JavaScript that does this manipulation.* The DOM is notoriously hard to define, so here's a metaphor:

Let's say we have a piece of paper that says "Coding Dojo Rocks!". We take out some magnetic letters, go to our fridge, and write out "Coding Dojo Rocks!", as per our paper. That paper is your HTML, and the magnets can be considered our DOM.

In a lot of cases, the magnets will remain the same as what is written on our paper, so there will be almost no difference between HTML and the DOM. In other cases, we will want to change the words on the fridge, or alter the color of the magnets, or add other neat functionality. Changing those things is manipulating the DOM; the HTML paper we started with is unchanged, but the DOM is no longer identical.

When you see JavaScript that changes the DOM or hear people talking about DOM manipulation, they are basically talking about **interacting with those fridge magnets, after they had copied off of our HTML paper**.

Terminal:

|  |  |
| --- | --- |
| pwd | Present Working Directory: Use this command any time you are unsure where you are currently in your file structure. |
| ls | To see all the files and folders that are in current directory. |
| cd [folder name] | “Change directory to [folder\_name]” |
| cd .. | Traverse up to parent folder |
| cd / | Change current directory to root directory of computer |
| cd ~ | Change current directory to root directory of current user |
| mkdir [folder name] | Makes a new directory |
| rmdir [folder name] | Delete an empty folder |
| rm –rf [folder name] | Delete a non-empty folder. The *r* stands for recursive; the *f* stands for force. |
| touch [New File] | Creates new, empty file with the name and extension type that you specify. |
| rm [file name] | Removes the file without any prompt. Add “-i” to the command to prompt whether or not file should be deleted. |
| mv [Current File Name] [New File Name] | Rename file. |
| mv [File to Move] [location and name to move the file] | Move file to another folder. |
| cp [file to copy] [name of new copied file] | Copies a file into its current directory with the new name. |
| cp –rf [folder to copy] [Folder destination to copy into] | Copies a directory with “-rf” option to copy all the internal contents of the directory. |
| mv [folder to move] [Folder destination to move into] | Moves a directory. |
| open . | Opens current directory in your Finder |

Command Prompt:

|  |  |
| --- | --- |
| cd | Change directory; can also display current directory |
| dir | See all files and folders in directory |
| cd .. | Traverse up to parent folder |
| cd / (or cd \) | Change current directory to root directory of computer |
| md [folder name] | Make a new directory |
| rd [folder name] | Delete empty folder |
| rd /s [folder name] | Delete non-empty folder |
| copy NUL [file name] | Create new file (by copying empty file) |
| copy [original file name] [copy file name] | Copy a file |
| del [file name] | Remove a file. Add “-p” for a prompt. |
| ren [old file name] [new file name] | Rename a file. |
| xcopy [source folder] [destination] | Copy a folder and all of its contents to another location.   * /s : copies directories and subdirectories, unless they’re empty * /e : copies all subdirectories, even if they are empty * /l : If Source is a directory or contains wildcards and Destination doesn’t exist, xcopy assumes destination specifies a directory name and creates a new directory. Then, xcopy copies all specified files into new directory. By default, xcopy prompts you to specify whether Destination is a file or a directory. |
| move [source folder] [new location] | Physically move file or folder to new location. |
| start . | Open current directory in file explorer |
| dir /b /s > [text file name] | Save directory listing to text file |

**Git:**

Git Commands:

|  |  |
| --- | --- |
| git init | Initialize the repository |
| git add | Add all the files that were changed since the last back up to the staging area |
| git status | Shows you all the files that were changed since the last backup and which ones are already added to the staging area |
| git commit –m “…” | Commits the changes to the repository |
| git checkout \_\_\_\_\_ | Switches to the branch name provided in your git repository. This will create a new branch if the name provided doesn’t exist. |
| git branch | Shows all of your git branches and marks the one you are currently on |
| git log | Shows all the backups created in the repository |
| git blame \_\_\_ | Shows who wrote which line of code or in other words who is to be blamed for that particular line of code |
| git remote add origin \_\_\_ | Tells git to add a remote place called ‘origin’ to a remote URL \_\_\_ |
| git push | Pushes the changes in your local repository to the remote repository |
| git pull | Pulls the changes in a remote repository to your own local repository |
| git clone \_\_\_\_ | Clones a remote repository in \_\_\_ to your own local folder |

Repository is place where version control system keeps track of all the changes you make. Most Version Control Systems (VCS) store current state of code, along with when each change was made, who made it, and a text log message that explains why they made the change.

**Working tree** is current view into the repository. It has all the files from your project: source code, build files, unit tests, and so on.

Commits are the individual pieces of history stored by the repository. Each one marks the progression of your code. Git stores your name and email address—from the configuration we did earlier—and adds a message to each commit.

That’s what the -m and string in the earlier command are for. The commit message is 'aboutMe.txt added'. *Properly written log messages are the killer feature of any version control system. They’re a place to explain the reason of your commit.* What does the new file do? Why did you change the code?

Example of git log entry:

commit bdf8ce93d2af928e8a430629225e3bd754429bd8

Author: Kari Ekenes <coolKari@gmail.com>

Date: Sun Jun 30 09:52:58 2013 -0700

index.html added

The first line here shows the commit name (commit hash). Commit names are SHA- 1 hashes generated by Git to keep track of a commit. Git uses these hashes to make sure that each commit identifier is completely unique. The second line in the git log output is the commit’s author info, the third is the date the commit was made, and the final bit of information is the commit log message from the commit.

\*NOTE: Commit history is displayed in reverse chronological order (like a blog).

To view code that changed in conjunction with log message, *add –p option to display diff that revision created*. Often you need to view only a set number of commits. You *can add -1 to the log command to limit the log to one commit*, -2 to limit it to two commits, and so on. For example, git log -10 shows the last ten log entries.

“git diff --stat [commit hash]” : see what changes have been made since a particular commit

Changing the history after you’ve shared it can lead to major headaches when others try to sync against your changes. One of the values of completely distributed development is that you *share only what is ready*. Make sure changes are ready before you push them. By keeping all your changes local until they are ready, you leave yourself the option of rewriting your repository’s history without worrying how it affects others.

\***Try to use revert over reset whenever possible**.

The simplest way to revert an existing commit is the git revert command. It “reverts” a commit by *creating a new commit in your repository that reverses all the changes made by the original commit*.

Normally Git commits the reversal immediately, but you can *add the -n parameter to tell Git not to commit*. This is useful when you need to revert multiple commits. Just run multiple git revert commands with the -n parameter, and Git stages all the changes and waits for you to commit them.

Example:

git revert –n HEAD

You must provide it with a commit name so it knows what to revert. For example, if you want to revert the commit 540ecb7 and HEAD, use the following. **Always revert backward—the most recent first**. That makes sure you don’t have any unnecessary conflicts to work through when reverting multiple commits.

Git assumes we make mistakes like this and allows us to reset the repository to the state we want with reset. The command git reset takes a commit name as its parameter or defaults to HEAD if you don’t provide one. The **first leading seven characters** that are displayed by git log are normally unique enough that you can use them without having to use the full forty-character hash.

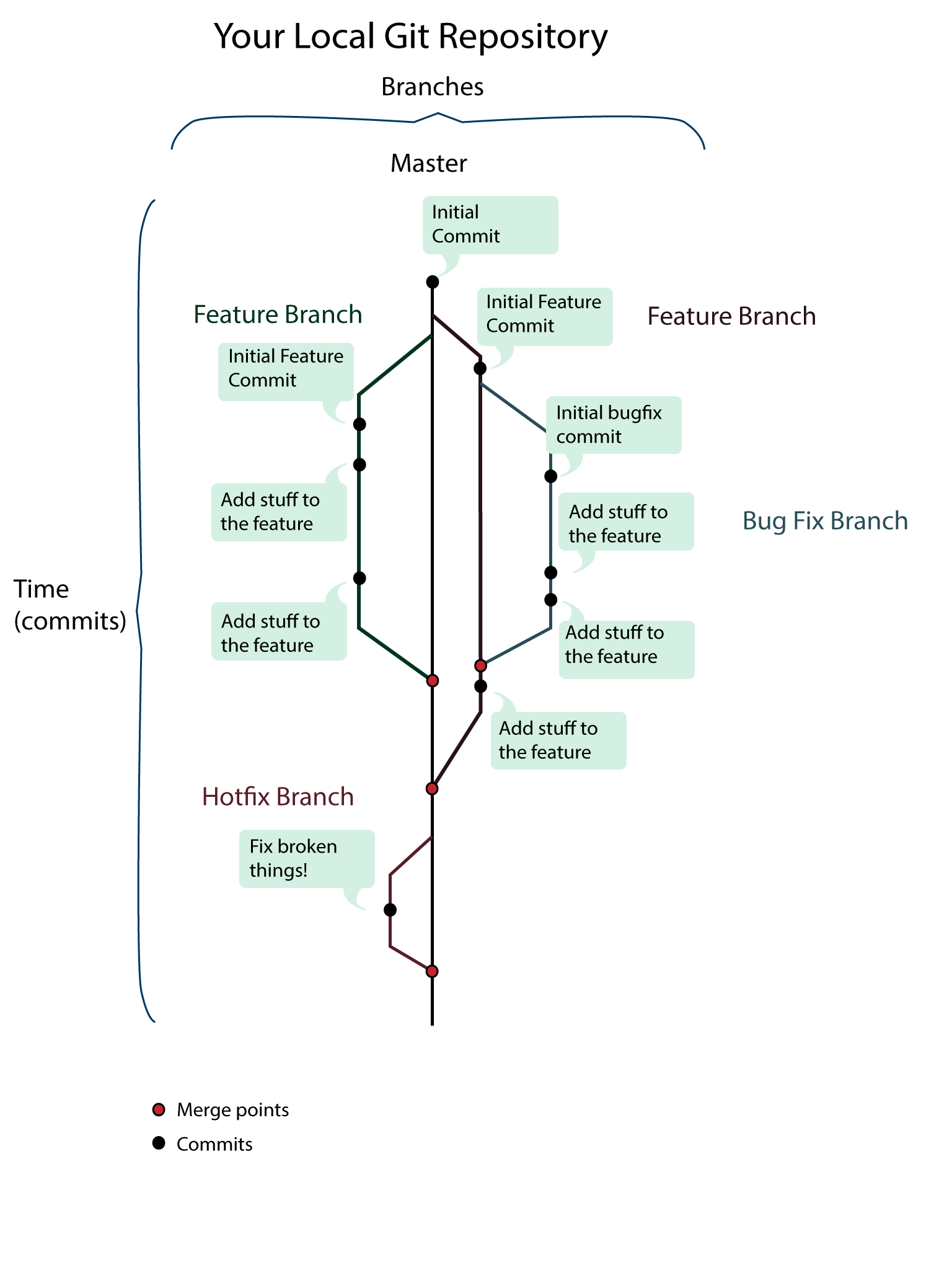
Example of resetting to a particular commit:

git reset <commit hash> --hard

Example of resetting changes to a particular file and not the whole project:

git reset <commit hash> <filename> --hard

*Warning!!! When we revert the changes using this command, the changes we revert are GONE FOREVER, ERASED FROM THE LOG, UNRECOVERABLE. This is what makes reset perfect for those situations when you want to UTTERLY AND COMPLETELY ERASE SOMETHING, but A TERRIBLE IDEA FOR ANYTHING ELSE.*



Git has three main states that your files can reside in: **committed, modified, and staged.**

* Modified: You have changed file in Working Directory but have not added it yet.
* Staged: You have marked a modified file to go into the Staging Area for your next commit.
* Committed: Data is safely stored from the Staging Area into your local .git Directory.
  + Working Directory: A single checkout of the project. These files are pulled out of the compressed database in the Git Directory and placed on disk for you to use or modify.
  + Staging Area: Is a simple file, generally contained in your Git directory, that stores information about what will go into your next commit -- basically an "index" of the staged files.
  + Git Directory (Repository): Metadata & Object Database of the project (a compressed reference "skeleton" of the project). This is the essential part of Git *-- it's what is copied when you clone a repository* from GitHub or another computer.

Basic Git workflow:

1. Create new file or modify existing files in working tree.
2. Stage files by adding them to staging area.
3. Do a commit, which takes files as they are in staging area and stores that snapshot permanently to your Git directory.

**Add**: Add files to staging area to have them included in next commit. If you add a file and then make more changes to it, you must re-add the file to include all of your recent changes.

**Commit**: Every file added to staging area will be updated with changes you have made. With every commit, you create another local version of your project. Git clears staging area every time you do a commit.

**GitHub:**

GitHub is a *Git repository hosting service*, but it has many features.

While Git is a command line tool, GitHub has a Web-based graphical user interface (GUI). It also provides access control and several collaboration features, but for the early parts of this bootcamp, we'll be using GitHub to mostly host our completed assignments.

***As a programmer, your GitHub account serves as your portfolio***.

*Pushing to GitHub lets you put a copy of your repository onto GitHub*.

1. Create GitHub repository: When you log in to your GitHub account, at the right of your username is a button that allows you to create a new repository. Once you’ve created one, you can push your local repository to GitHub.
2. Navigate to local git repository in command line.
3. Add newly created GitHub repository remotely by running the following:

git remote add origin https://github.com/<github username>/<github repo name>.git

The word **remote** is just a shorthand for remote repository - i.e. a repository that's not on your local machine. **Origin** is the conventional name for the GitHub copy of your repository.

1. Push the changes from local to GitHub:

git push origin master

\***Origin** refers to GitHub repository. **Master** refers to local repository.

Other information:

* You can add as many remote repositories as you want by using git remote add <remote name>
* You can also delete remote repository that you've added by using git remote remove <remote name>
* To list all the remote repositories you are connected to, run git remote
* To see more information about a remote repository, use git remote show <remote name>



The flagship functionality of GitHub is **forking** – copying a repository from one user’s account to another. This enables you to copy a project, which you don’t have a write access to and modify it under your own account. If you make changes you’d like to share, you can send a notification called a **pull request** to the original owner. That user can then, with a click of a button, **merge** the changes found in your repo with the original repo.

01/24/17 (Day 2): **CSS**

3 ways of attaching CSS to a document: inline, internal, and external. Inline and internal CSS are considered to be bad practices. Only use *external* stylesheets, linked from HTML document using <head> tag.

Ex:

<link rel=”stylesheet” href=”css\_1.css”>

CSS selectors are used for declaring HTML elements to which a style will apply. Most common selectors are:

* all elements with a specific HTML tag (e.g. p, h1)
* elements specified by following attributes:
  + id (preceded by # in CSS)
    - Must be unique and can only be used once in your page
  + class (preceded by . in CSS)
    - Can be used multiple times to share repeating CSS code

Declaration block is a list of declarations contained within braces. **Each individual declaration consists of a property, a colon, and a value**. If block has multiple declarations, separate with semi-colon.

Default priority order of CSS:

**1. element (1 point)**

**2. .class (10 points)**

**3. #id (100 points)**

“Inspect Element” tool in Browsers:

* Elements tab
  + View HTML that is being rendered by browser. You can see all classes and id’s.
* Styles tab
  + Shows all of the rendered CSS for a particular element

Each HTML element by default is either inline or block. Inline items are laid out one after the other horizontally across the available space until there is no more room. There is no height property. Block elements stack vertically.

**Block elements: <body>, <div>, <form>, <h1>-<h6>, <p>, <table>, <ul>, <ol>, <li>**

**Inline elements: <a>, <img>, <span>, <input>, <label>, <select>, <textarea>**

***It’s invalid to put block elements inside inline elements***.

Example:

Not valid: <a href="www.google.com"><h1>Google</h1></a>

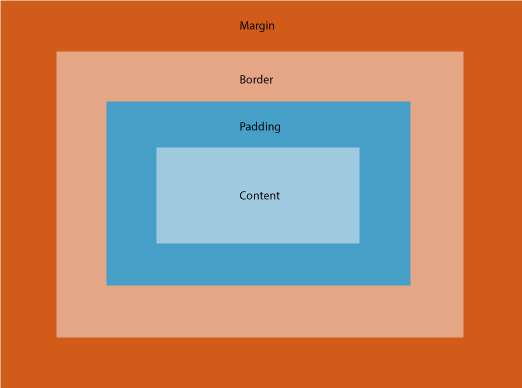
Valid: <h1><a href="www.google.com">Google</a></h1>

Display properties:

* inline
* block
* none
* inline-block: Display elements inline, but allow them to retain their block-level characteristics

All web layouts are accomplished with block elements. There are only 3 rules:

1. **Total area**: space an element occupies and affects
   * Total width is how much horizontal space a block occupies, including margin, border, and padding.
   * Box model consists of properties margin, border, and padding. Margin is outside block elements, while padding is within them.



Example:

div{

padding-top: 10px;

padding-right: 10px;

padding-bottom: 10px;

padding-left: 10px;

}

The shorthand property flows clockwise: top -> right -> bottom -> left.

1. **Float, clear and overflow**
   * *Float* property specifies whether or not an element should float
   * *Clear* property used to control the behavior of floating elements. Elements after a floating element will flow around it. Use clear to avoid this.
   * *Overflow* property specifies whether to clip content or to add scrollbars when the content of an element is too big to fit in a specified area.
     + Visible – Default. Overflow is not clipped. It renders outside the element’s box.
     + Hidden – Overflow is clipped, and the rest of the content will be invisible.
     + Scroll – Overflow is clipped, but a scrollbar is added to see the rest of the content.
     + Auto – Overflow is clipped, a scrollbar should be added to see the rest of the content.
2. **Nested elements**
   * According to box model, total width of an element is: *(Margin\*2) + (Border\*2) + (Padding\*2) + Content Width*. Calculating height is tricky because *vertical margins collapse*. When the vertical margins of two elements are touching, ***only the margin of the element with the largest margin value will be honored,*** while the margin of the element with the smaller margin value will be collapsed to zero.
   * There are other situations where elements do not have their margins collapsed:
     + floated elements
     + absolutely positioned elements
     + inline-block elements
     + elements with overflow property set to anything other than visible (They do not collapse margins with their children.)
     + cleared elements (They do not collapse their top margins with their parent block’s bottom margin.)

CSS offers 5 position values:

* 1. Static
     + Default. Any element that has position: static applied is in the normal document flow. A statically positioned element will ignore any values for the properties, top, right, bottom, and left, as well as any z-index declarations.
  2. Relative
     + Relatively positioned elements are positioned based on the same top, right, bottom, and left properties, but are simply shifted from where they would normally sit. In a sense adding relative positioning is similar to adding a margin with one very important difference: *the elements around a relatively positioned element act as though that shift didn’t exist*. They ignore it.
  3. Absolute
     + Completely removed from normal document flow. As far as elements around them are concerned, absolutely positioned element doesn’t exist. It’s as though the display property of the element was set to **none.**
     + Set location of absolutely positioned element through the top, right, bottom, and left properties. You’ll usually define only two of these.
     + ***An absolutely positioned element is positioned relative to the first parent element that has a position other than static applied to it. If no parent element up the chain meets that condition the absolutely positioned element is positioned relative to the document window.***
  4. Fixed
     + Acts similarly to absolute positioning.
     + *Fixed positioned element is always positioned relative to the browser window* and takes top, right, bottom, and left properties.
     + *Fixed positioned elements are fixed. They don’t move as page is scrolled*.
  5. Inherit
     + Property should inherit value from parent element.

Two ways to center an element inside a container:

#container{

width:200px;

height:200px;

background: blue;

position:relative;

}

#center-me{

background: red;

height: 100px;

width: 100px;

margin: auto;

position: absolute;

left: 0;

bottom: 0;

top: 0;

right: 0;

}

OR

#container{

width:200px;

height:200px;

background: blue;

position:relative;

}

#center-me{

background: red;

height: 100px;

width: 100px;

margin: 0 auto;

}

Styling Texts:

* Color – text color specified with color property
* Alignment – Used to set the horizontal alignment of a text
  + Center, right, left, justify
* Text-decoration – add/remove underlines, overlines, line-throughs, etc.
* Text-indent – indentation
* Font-family – specifies font for an element
  + Two types of font family names:
    - Family-name: times, courier, arial, etc.
    - Generic-family: serif, sans-serif, cursive, fantasy, monospace
  + Web-safe fonts:
    - **Sans-serif**: fonts without serifs: Verdana, Arial, Trebuchet MS
    - **Serif:** fonts with serifs: Times New Roman, Georgia
    - **Monospaced:** fonts in which every glyph takes up the same space: Andale mono, Courier new
    - **Cursive:** fonts that have a decorative, often handwritten-looking style: Comic Sans
    - **Fantasy:** fonts that have a bold, often ornamental or quirky style, which are meant to be used for headings, not body: Impact
  + Browser reads font stack from left to right until it finds a font installed on system.
  + **Fonts with more than one word in their name need surrounding in quotes.**
* Font-size – font size
  + Units that can be used: **px, em, pt,** and **%**. Recommended to use is **em** because it responds to resizing well.
  + *In general, 12 pt = 16 px = 1 em = 100%.*
* Font-style – italics
  + Possible values: normal, italic, oblique, initial, and inherit
* Font-weight – Bold
  + Possible values: normal, bold, bolder, lighter, inherit, and numerical values 100 – 900

**LEGO Concept:**

Way to visualize the different elements in your website as LEGO pieces. You want to put the big blocks down first and then place smaller blocks inside of them. By focusing on the big blocks first, you can think of the general layout of your website. After placing the big blocks, you determine which smaller boxes will exist inside of the bigger boxes and position them accordingly. It might seem faster to look at a screenshot and start coding HTML/CSS right away but drawing out the different lego pieces on the whiteboard first (bigger ones first, then smaller ones) will help you construct HTML/CSS more

quickly.

***\*IMPORTANT***: Different browsers may give default attributes to HTML elements, causing sites to look different based on the version of the browser or whether you're seeing the site in IE, Firefox, Chrome, etc. To solve many cross-browser issues, it's encouraged that you use 'RESET' and 'NORMALIZE' and, basically, define what the default attributes should be (instead of relying on the default values that come with the browser).

RESET: Using a well-crafted set of global CSS reset styles enables designers to make assumptions about the default behavior of browsers. These presentational assumptions greatly simplify the process of creating a “universally” consistent CSS design using only one set of CSS rules.

NORMALIZE: What does it do?

* Preserves useful defaults, unlike many CSS resets.
* Normalizes styles for a wide range of elements.
* Corrects bugs and common browser inconsistencies.
* Improves usability with subtle modifications.
* Explains what code does using detailed comments.

Technical recruiters will ask. Answer: what reset/normalizing means and why they are used. It’s important to validate HTML/CSS often, as validating the HTML/CSS gives insight on how you can fix issues that may lead to cross-browser compatibility issues.

READ: <https://perishablepress.com/a-killer-collection-of-global-css-reset-styles/>

Normalize.css location: <https://github.com/necolas/normalize.css/>

Removing or changing item markers: **list-style-type property**. Possible values: none, circle, square, upper-alpha, etc. Most common is “none.” For custom item markers, use: **list-style-image** property.

Ex:

ul{

list-style-image : url('star.gif');

}

Link selectors:

|  |  |  |
| --- | --- | --- |
| :hover | a:hover | When user hovers over an element |
| :link | a:link | Selects all unvisited links |
| :visited | a:visited | Selects all visited links |
| :active | a:active | Selects the active link (a link is “active” when it’s clicked on) |

Background properties:

|  |  |
| --- | --- |
| background-color | Specifies the background color to be used. RGB and hex values are both supported, along with some English words. |
| background-size | Specifies the size of the background images.  Possible values:   |  |  | | --- | --- | | Length | Sets the width and height of the background image. The first value sets the width, the second value sets the height. If only one value is given, the second is set to "auto"  Ex:  background-size: 400px 500px; | | Percentage | Sets the width and height of the background image in percent of the parent element. The first value sets the width, the second value sets the height. If only one value is given, the second is set to "auto".  Ex:  background-size: 80% 80%; | | Cover | Scale the background image *to be as large as possible so that the background area is completely covered by the background image*. Some parts of the background image may not be in view within the background positioning area  Ex:  background-size: cover; | | Contain | Scale the image to the *largest size* such that both its width and its height can *fit inside the content area*.  Ex:  background-size: contain; | |
| background-repeat | Specifies how to repeat the background images.  Possible values:   |  |  | | --- | --- | | Repeat | The background image will be repeated both vertically and horizontally. This is the default. | | Repeat-x | The background image will be repeated only horizontally | | Repeat-y | The background image will be repeated only vertically | | No-repeat | The background-image will not be repeated | | Inherit | Specifies that the setting of the background-repeat property should be inherited from the parent element | |
| background-image | Specifies ONE or MORE background images to be used. It takes a URL as a value. Ex:  background-image: url('my\_background\_image.gif'); |

Border radius: This is for rounded corners. The greater the number, the more rounded the corners will appear.

Syntax: border-radius: 30px;

*FAQ:*

Q: Can I add **multiple classes to an element**?

A: Yes. Simply separate the class names with space. Ex: <div class="nav side-bar"></div>

Q: Can I **add a class and an id** to the same element?

A: Yes. An element can have an id and multiple classes. Ex: <div id="primary-nav" class="nav side-bar"></div>

Q: Do I have to add a class or an id to target a nested element?

A: Not as a rule. Consider the following example:

<ul id="nav-menu">

<li><a href="#">Home</a></li>

<li><a href="#">Dashboard</a></li>

<li><a href="#">About</a></li>

<li><a href="#">Contact Us</a></li>

</ul>

To target the links and remove the underlines from them we can do:

#nav-menu li a{

text-decoration: none;

}

**In general, make your code easier to read by avoiding adding too many classes and ids.**

**CSS3:**

Series of modules that are designed to be implemented separately and independently from each other. We can group our website's visual elements into two categories: *critical and non-critical*. Visual elements that are **critical** include **branding, usability, accessibility and layout**. Visual elements that are **non-critical** include **interaction, visual rewards, feedback, and movement**. We have to make sure we *apply CSS3 to non-critical areas* because it will be risky to use CSS3 on critical areas.

WebKit is the engine behind Safari and Chrome so we use the **-webkit-** prefix for these browsers and Gecko is the engine behind Firefox which uses -moz- prefix.

Border Radius: round corners of an element with a specified radius value.

Ex:

.awesome {

-webkit-border-radius: 10px;

-moz-border-radius: 10px;

border-radius: 10px;

}

Box Shadow: add shadow to an element with options for the direction, the amount of blur, and color of the shadow.

Ex:

.awesome {

-webkit-box-shadow: 10px 10px 5px #333;

-moz-box-shadow: 10px 10px 5px #333;

box-shadow: 10px 10px 5px #333;

}

Opacity: 1 means that the element is fully opaque (fully filled) while a value of 0 means transparent (invisible).

Ex:

.awesome {

opacity: 0.5;

}

RGBA: adding the ability to specify a level of opacity along with an RGB color value.

Ex:

.awesome {

background-color: rgba(0, 0, 0, 0.8);

}

Multiple background images: CSS3 allows you to apply multiple background images on an element (separated with commas). This is how people create the parallax scrolling effect.

Ex:

body {

background: url(first\_image.png) no-repeat top left,

url(second\_image.png) repeat-x bottom left;

}