wdi-14

ga-wolf

Published
with GitBook

Table of Contents

WDI 14

Daily Stuff

Week 01

Week 01, Day 01.

What we covered today:

- Introduction | Orientation | Housekeeping
- Structure of the Course
- Introduction to the Command Line

Introduction | Orientation | Housekeeping

Jack's slides

Jack Jeffress - Lead Instructor - jack.jeffress@ga.co

Sherif Gamal - Teaching Assistant sherif.gamal@generalassemb.ly

Kane Mott - Teaching Assistant kane.mott@generalassemb.ly

Gigi Tsang - Producer - gigi@ga.co

Meggan Turner - Assistant Course Producer - meggan.turner@ga.co

Lucy Barnes - Outcomes Producer - lucy.barnes@generalassemb.ly

The Office is typically open from 8am to 9pm.

Classroom Culture

- Every time Jack makes a pun, someone else has to make one (One for one puns).
- Be collaborative, be supportive.
- Phones on silent.
- Ensure good personal hygiene.
- No smelly food in the classroom.
- Keep your sense of humour.
- Be open to supportive gestures.
- Don't be afraid to fail.
- No stupid questions.
- No spoilers => Except in code.

Geek | Hacker Culture

Share and enjoy.

Books, movies and TV series, stupid memes

The Jargon File

The Tao of Programming

"Kicking a dead whale up a beach"

Necessary Links, Meetups and Newsletters

Links

- Ruby on Rails
- Ruby on Rails Community
- #rubyonrails on Freenode IRC

Newsletters

- Ruby Weekly
- Javascript Weekly
- Versioning
- Sidebar

Meetups

- RORO
- SydJS

Also, check out these

- Web Design Field Manual
- Web Design Stack
- Panda App

Other

What will go wrong? Everything. This won't be easy for anyone.

> "If debugging is the practice of removing bugs from software... Then programming must be the practive of adding them." – E. W. Dijkstra

The best thing you can learn as a beginner is **how to debug.**

A Typical Day here at GA...

Time	What?
09:00 - 10:00	Warmup Exercise
10:00 - 01:00	Code Along
01:00 - 02:00	Lunch
02:00 - 02:30	Review
02:30 - Beyond	Labs / Homework

Breaks for morning and afternoon tea last for twenty-ish minutes and are whenever works best.

In terms of homework, we like to keep you busy until 9 or 10.

We have office hours here in Sydney (except during Week 6 - we have the Spit to Manly.)

Structure of the Course

- Week 01 - Front End
- Week 02 - Front End
- Week 03 - Project 00
- Week 04 - Ruby
- Week 05 - Ruby on Rails
- Week 06 - Project 01
- Week 07 - Advanced Front End
- Week 08 - Advanced Front End
- Week 09 - Project 02
- Week 10 - Advanced Back End / Advanced Everything
- Week 11 - Advanced Back End / Advanced Everything
- Week 12 - Project 03

The Command Line

It's probably worth downloading iTerm 2.

Web programmers have to live on the command line. It gives us fast, reliable, and automatable control over computers. Web servers usually don't have graphical interfaces, so we need to interact with them through command line and programmatic interfaces. Once you become comfortable using the command line, staying on the keyboard will also help you keep an uninterrupted flow of work going without the disruption of shifting to the mouse.

The command-line interface, is often called the CLI, and is a tool, that by typing commands, performs specific tasks. It has the potential to save you lots and lots of time because it can automate things, loop through items etc.

`date` - Will print the current date and time

`which date` - Will show the relevant file (will probably return **/bin/date**)

`pwd` - Stands for **Print Working Directory**, will show you where you are in your computer

`mkdir` - Stands for **Make Directory**

`rmdir` - Stands for **Remove Directory**

`clear` - Will clear the screen (ctrl + l will do this as well)

`reset` - Will reset your terminal

`cd` - Stands for **change directory**

`cat filename` - Will show you the contents of the specified file

`whoami` - Will show the logged in user

`ps` - Will you show you all running processes

`ps aux` - Will show you all of the running processes with more details

`top` - Will show you the **Table of Processes**

`grep` - Stands for **Global Regular Expression Print** - useful for finding files or content

`ls` - Short for List. This will show you all of the files and folders in the current directory

`ls /bin` - Will show you all terminal commands

`man` - Stands for **Manual**. To use it, follow the man command with another command (i.e. `man grep`).

Most commands will have additional **flags**. A flag is a request for more information.

A good example of this is the following:

```
> ls
Applications Documents Desktop etc.
> ls -l
drwx------   6 jackjeffress  staff   204 16 Mar 15:39 Applications
etc.

#https://developer.apple.com/library/mac/documentation/Darwin/Reference/ManPages/man1/ls.
```

```
# cd can do a lot of things!

> cd
# Will take you back to your "home folder"
> cd /
# Will take you back to your "root folder"
> cd FolderName
# Will take you into the specified folder name
> cd FolderName/AnotherFolderName
# Will take you through FolderName and then into AnotherFolderName
```

```
# We can make folders in the CLI by using mkdir

> mkdir Projects
# Then we can move into it
> cd Projects
# ls will show most of your files (ones that aren't prefixed by a .)
> ls
# ls -la will show every file (even hidden files)
> ls -la
# This will change to the current directory
> cd .
# This will open the current directory in Finder
> open .
# This will go back to the previous directory
> cd ..
# If you hit tab at this point, it would autocomplete for you
> cd pro
# This will create a markdown file called README
> touch README.md
# To open an application in terminal (can use any application that you have)...
> open -a "Sublime Text"
```

```
# This will open the file and show the contents
> cat books
# Will show you the contents, pipe it into the sort program. This doesn't change the orig
> cat books | sort
# The pipe character pipes the output to a command
# This will sort the contents and books, and put the sorted contents into the sorted_book
> cat books | sort > sorted_books
# This will rename the sorted_books to books (and will overwrite the books file if it alr
> mv sorted_books books
# This will open and show the contents and books, and shows only lines that have the word
> cat books | grep script
# To copy a file, the command is cp (this needs parameters - or arguments. It needs a sou
# cp source destination
> cp books my_books
# To remove things, use the rm command (this doesn't get moved to your trash! It will del
> rm my_books
```

What happens when we run commands?

```
# It will go through all of the folders and files that are shown when we run the followin
> echo $PATH
```

Here is a basic bash profile.

Some recommended readings when it comes to the Command Line Interface (CLI):

- http://en.flossmanuals.net/command-line/index/
- http://cli.learncodethehardway.org/book/
- http://cristal.inria.fr/~weis/info/commandline.html
- https://quickleft.com/blog/tag/command-line/page/4/
- http://en.wikipedia.org/wiki/The_Unix_Programming_Environment
    - This PDF download might work...

Some other useful links...

- 15 Use Cases of Grep
- 40 Terminal Tips and Tricks
- Terminal Cheatsheet)

Homework

- Track down the Terminal City murderer
- Make a start reading up on the command line documentation of your choice (select from the links in recommended readings just above!)

- Finish the WDI Fundamentals pre-work and bring any questions for tomorrow Read and work through one or more of the following:+

https://en.wikipedia.org/wiki/The_Unix_Programming_Environment
https://quickleft.com/blog/tag/command-line/page/4/ (reverse chronological order, start at the bottom) http://en.flossmanuals.net/command-line/ http://cli.learncodethehardway.org/book/
https://en.wikipedia.org/wiki/In_the_Beginning..._Was_the_Command_Line

Finish off the prework! End of Day 1.

Week 01, Day 02.

What we covered today.

- Git & Github
- Alex Swan & further Git
  - @alxswan
  - Slides
  - Gist
- Javascript Date Types
- Javascript Control Structure
  - Javascript control flow and logical operator slides

Git and Github

A very short history...

A Version Control System designed and developed by a Finnish guy, Linus Torvalds, for the Linux kernel in 2005. Apparently, he named it Git because of the British-English slang meaning "unpleasant person". Torvalds said: "I'm an egotistical bastard, and I name all my projects after myself".

What does it do?

- A way to snapshot - you get a time machine
- A way to collaborate - working in parallel
- Saves us from moving code around on Floppy Disks
- Automates merging of code
- Lets you distribute a project

N.B. It is quite hard to understand, and you won't understand most of the problems that it solves yet! Just trust us.

Git is on your local computer. Github is in the cloud - it is the central repository!

For a nice introduction to it, see here. P.S. Octocat is the logo!

Some basic Git commands:

- `git init` - This initializes the git repository (behind the scenes it creates a .git file in the folder - it has called .git because any file prefixed by a . is hidden in finder and ls). Make sure you don't do this in another git repository!
- `git status` - This will tell you what is happening the current project. Commit status,

untracked files etc.

- `git add .` - This will add all files in the current directory into the "staging area". Git is now paying attention to all files. You can alternatively specify a particular file - `git add octocat.txt`. We could also add all files of a particular type - `git add '*.txt'` (this needs quotes!) or multiple files at a time - `git add octocat.txt blue_octocat.txt`.
- `git commit -m "You're commit message` - This is the thing that takes the snapshot. You must give it a message!
- `git log` - Will show you all of the snapshots in the current project
- This has all been happening locally! So we need to connect it to Github...
- `git remote add origin https://github.com/try-git/try_git.git` - It takes a *remote name*, the word origin, and a *remote repository*, the URL.
- `git push -u origin master` - The first time you run git push (which moves it up to github), make sure you specify `-u origin master`. This tells your local machine to always use the remote link you specified in the step before. Once you have done this - you can run `git push` from now on
- `git pull` - this brings the code down from Github
- `git diff HEAD` - this will show the differences between the current commit and the previous commit
- `git diff --staged` - You can show the differences between the currently staged files by using this command
- `git reset FILENAME` - Will remove the specified files or folders from the staging area (so they won't be pushed!)
- `git checkout -- octocat.txt` - This will go back to the last commit involving the octocat.txt file using this.
- `git branch clean_up` - This allows us to work in an alternate reality - changes you make here won't effect anything in other branches
- `git checkout clean_up` - Will move to the clean_up branch.
- `git rm FILENAME FILENAME` - Will not only delete the file on your computer, but also remove it from the staging area with Git.
- `git add .` then run `git commit -m "COMMIT NAME"`. This will take a screenshot of the current branch position.
- `git checkout master` - This will move to the master branch.
- `git merge clean_up` - Will merge the clean_up branch into the current branch.
- Now that everything has been merged, we can delete the clean_up branch - `git branch -d clean_up`.
- We can run `git push` now! Everything is up there.

A really basic process: (Follow these steps every time!)

- `git add .`
- `git commit -m "Commit message`

- `git pull`
- `git push`

Some good github tutorials:

- A tutorial by the people at Code Academy
- A tutorial by the folks at Atlassian
- A tutorial by Roger Dudler
- A tutorial by Git Tower

And of course Alex's Gist and slides:

- Slides
- Gist
- She can also be found on the mentor channel, as 'alex-mentor', or on Twitter.

Introduction to Javascript

Jack's intro slides can be found here.

> "Java is to Javascript, as ham is to hamster" - *Joel Turnbull*
>
> "We can't break the Web" - *Joel Turnbull*

Javascript is the most popular programming language in the World by a long way. It works everywhere.

To be a programming language, a language needs to do the following things:

- Add 1 to a number
- Check if a number is equal to zero
- Branch - can alter flow

SOMETHING REALLY IMPORTANT

There are two audiences that see any program that you will write. There is the computer, and there is the next human who sees the code (who could well be you). Prioritize the human! You need to make code readable and logical.

> "If given a program that works but is unreadable, and another that doesn't work but makes sense - I would take the one that doesn't work. I can still work with it and I understand it. It isn't a nightmare."

A short history of Javascript.

Javascript Primitive Value Types

Javascript value slides

Strings - an immutable string of characters

```
var greeting = "Hello Kitty.";
var restaurant = "McDonalds";
```

Numbers - whole (6, -102) or floating point (5.8727)

```
var myAge = 35;
var roughPi = 3.14;
```

Boolean - Represents logical values true or false

```
var catsAreBest = false;
var dogsAreBest = false;
var wolvesAreBest = true;
```

undefined - Represents a value that has not been defined.

```
var notDefinedYet;
```

null - Represents something that is explicitly undefined.

```
var goodPickupLines = null;
```

Variable Names

- Begin with letters, $ or _
- Only contain letters, numbers, $ and _
- Case Sensitive
- Avoid reserved words
- Choose clarity and meaning
- Prefer camelCase for multipleWords (instead of under_score)
- Pick a naming convention and stick with it

```
// All Good!
var numPeople, $mainHeader, _num, _Num;

// Nope.
var 2coolForSchool, soHappy!;
```

Everything in Javascript returns the result of an expression.

```
4 + 2;
// Returns 6.

8;
// Returns 8.

console.log("Hello World.")
// Passes a string into the console.log function. Then returns the result of this express

// Variables can store the result of expressions as well.
var courseName = "W" + "D" + "I";
var testMultiplication = 5 * 7;

var x = 2 + 2;
var y = x * 3;

var name = "Pamela";
var greeting = "Hello" + name;
```

Javascript is a *Loosely Typed* language.

What this means is that Javascript will figure out the value of a variable by looking at the type of the result of the expression.

A variable can be of only one type at one time, but can be reassigned.

```
var y = 2 + ' cats';
console.log(typeof y);
```

For a much more complex overview of the differences between loosely typed and strongly typed languages, see here.

Remember that they say you need to write 10000 words before you write a novel. All of the stuff you are learning right now is a part of that 10000 words. It's not meant to be perfect!

Here is an exercise for you to muck around with. and here are Sherif's solutions.

Comments

Comments are human-readable lines of text that the computer will ignore.

In atom, `<CMND> + L` will toggle the comments.

```
// This is a single line comment in JS

/*
    This is a block level comment
    i.e. multiline
*/
```

Control Flow with Javascript

Javascripts Comparison Operators

Use these operators to compare two values for equality, inequality or difference.

Operator	Meaning	True Statements
==	Equality	28 == '28', 28 == 28
===	Strict Equality	28 === 28
!=	Inequality	28 != 27
!==	Strict Inequality	28 !== 23
>	Greater than	28 > 23
>=	Greater than or Equal to	28 >= 28
<	Less than	24 < 28
<=	Less than or Equal to	24 <= 24

We can all of these in any statement or evaluation. Very useful for if's etc.

We should also use the strict equality and inequality operators. They compare type (i.e. strings, numbers etc.) as well as content.

Logical Operators

These are typically used in combination with the comparison operators, they are commonly used to group multiple conditions or for special types of variable declarations.

Operator	Meaning	True Expressions
&&	AND	4 > 0 && 4 < 10
| |	OR	4 > 0 | | 4 < 3
!	NOT	!(4 < 0)

In any control flow statement in Javascript - we can use all logical and comparison operators.

N.B. Empty strings (""), 0, undefined and null are all considered false in javascript!

If you want to find out everything there is to know about Javascript Equality, see here.

The if Statement

We can use an if statement to tell js which statements to execute, based on a condition.

If the condition (the things within the parentheses) evaluates to true, it will run whatever is within the curly brackets. Otherwise it will skip over it.

```
var x = 5;

if ( x > 0 ) {
  console.log( "x is a positive number!" );
}

// This would run, unless the variable x had a value less than or equal to zero.
```

We can give an if statements multiple branches...

```
var age = 28;
if (age > 16) {
  console.log('Yay, you can drive!');
} else {
  console.log('Sorry, but you have ' + (16 - age) + ' years til you can drive.');
}
```

It will always run one of them! But will never run both.

You can also use else if if you have multiple exclusive conditions to check:

```
var age = 20;
if (age >= 35) {
  console.log('You can vote AND hold any place in government!');
} else if (age >= 25) {
  console.log('You can vote AND run for the Senate!');
} else if (age >= 18) {
  console.log('You can vote!');
} else {
  console.log('You have no voice in government!');
}
```

Now that we have done a bit of an introduction to if, else if, and else statements, have a go at these exercises.

Homework

- These tasks
- Bonus
  - Head start for tomorrow here
  - Read this
  - Or this
  - Link to download both

Week 01, Day 03.

What we covered today:

- Warmup Exercise
- Documented Solution
- Talk with Lucy
  - Slides
- Functions in Javascript
- Loops in Javascript

Javascript Functions

Functions slides

Functions are way to make a collection of statements re-usable. This is the most powerful thing in Javascript.

You need to declare them!

```
function sayMyName () {
    console.log( "Hello Jane" );
}

// But functions are also data types, so they can be stored in a variable
// This is my favourite way of declaring functions! Stick to this.
var sayMyName = function () {
    console.log( "Hello Jane" );
}
```

If you want to see the difference between the function declarations, see here and here.

```
// We need to call functions though, we do this by having the parentheses at the end
sayMyName();
```

Parameters or Arguments

Functions in Javascript can accept as many named parameters (or arguments) as it wants. We can then use the arguments from within the function. This is crazily powerful.

```
var sayMyName = function ( firstName, lastName ) {
    console.log( "Hello, " + firstName + " " + lastName + "!");
}

sayMyName( "Jane", "Birkin" );

// We don't have to pass in plain data types, we can also pass in variables.

var serge = "Serge";
var gainsbourg = "Gainsbourg";
sayMyName( serge, gainsbourg );
```

Return Values

If you use the return keyword, we can return a value to wherever the function was called (it also leaves the function).

```
function addNumbers( num1, num2 ) {
  var result = num1 + num2;
  return result; // Anything after this line won't be executed
}

var sum = addNumbers( 5, 2 );
console.log( sum ); // Logs 7.

// We can take this further though, because we can use functions calls in expressions.

var biggerSum = addNumbers( 2, 5 ) + addNumbers( 3, 2 );
// biggerSum is declared as 12.

// We can take it further again! Because we can call functions from within functions
var hugeSum = addNumbers( addNumbers( 5, 2 ), addNumbers( 3, 7 ) );
// hugeSum is declared as 12 here as well.
```

Scope

JS Variables have "function scope". They are visible in the function where they're defined:

A variable with "local" scope:

```
function addNumbers( num1, num2 ) {
  var localResult = num1 + num2;
  console.log( "The local result is: " + localResult );
}

addNumbers( 5, 7 );
console.log( localResult ); // This will throw an error as localResult is defined in the
```

A variable with "global" scope:

```
var globalResult;
function addNumbers( num1, num2 ) {
  globalResult = num1 + num2;
  console.log( "The global result is: " + globalResult );
}

addNumbers( 5, 7 );
console.log( globalResult );
// Because this wasn't defined in the function, it is available. Will log 12.
```

Three things to know about functions:

- You can pass things in as parameters (or arguments)
- You can return things (to allow for chaining or usage in expressions)
- Variables declared in a function (or passed in as parameters) have local scope (or function scope) - i.e. only accessible within it.

For a more in-depth dive into Javascript variable scope, see here (this is very good), here and here.

Coding Conventions

Use newlines between statements and use indentation to show blocks. We aim for readability.

```
// BAD
function addNumbers(num1,num2) {return num1 + num2;}

function addNumbers(num1, num2) {
return num1 + num2;
}

// GOOD
function addNumbers(num1, num2) {
  return num1 + num2;
}
```

For information relating to javascript style, see the following:

- Idiomatic JS - the best style guide
- All encompassing, check this out
- AirBnB Style Guide - quite good

Loops in Javascript

The While Loop

The while loop will tell JS to repeat the statements within the curly brackets until the condition is true. It is ridiculously easy to make infinite loops with these. Beware! They'll crash your browsers and crush your spirits.

You need a condition in the parentheses, and you need something within the body (between the curly brackets) that will eventually change the condition to be false.

```
var x = 0;

while (x < 5) {
  console.log(x);
  x = x + 1;
}
```

The For Loop

Loops slides

A for loop is another way of repeating statements, more specialized than while.

It looks like the following:

```
for (initialize; condition; update) {

}

var x = 0; // This is the initialize value
while (x < 5) { // This is the condition (in the parentheses)
    console.log(x);
    x = x + 1; // This is the update
}

// To change this while loop into a for loop...

for (var x = 0; x < 5; x = x + 1) {
    console.log( x )
}
```

The Break Statement

To prematurely exit any loop, use the break statement:

```
for (var current = 100; current < 200; current++) {
    // current++ is the same current += 1 and current = current + 1
    // current-- also exists (minus 1)
    // Called syntactic sugar

    console.log('Testing ' + current);
    if (current % 7 == 0) {
        // The % stands for the modulus operator, it finds the remainder
        console.log('Found it! ' + current);
        break;
    }
}
```

Now that you have done a bit more on loops, have a crack at these exercises.

Homework

- Finish off exercises homework.
- And do as many of these these as you can.
- Here are some additional readings
  - MDN
  - Quite a good one
  - Speaking Javascript
  - Way more than you'll ever need

Week 01, Day 04.

What we covered today:

- Warmup Exercise
- Demos
- Collections in Javascript
  - Collections slides
  - Javascript Arrays
  - Javascript Objects

Warmup Exercise

Leap year.

- Requirements
- Warmup Exercise and Documented Solution

Collections

The Array Data Type

An array is a type of data that holds an ordered list of values, of any type. They are sequences of elements that can be accessed via integer indices starting at zero (it can be zero elements along as well). More or less, it is a special variable that can hold more than one value at a time.

Repeating myself... But the indexes start at zero!

How to create an array

- `var testArray = [ 1, 2, 3 ];` - This is an array literal - definitely the way you should do it.
- `var testArray = new Array( 1, 2, 3 );` - Uses the Array constructor and the keyword new. Does the same thing, but stick to the other way.

How to access arrays

```
var amazingFrenchAuthors = [ "Alexandre Dumas", "Gustave Flaubert", "Voltaire", "Marcel P

console.log( amazingFrenchAuthors[0] ); // Logs "Alexandre Dumas"
console.log( amazingFrenchAuthors[3] ); // Logs "Marcel Proust"

// You can use variables and expressions to access elements in arrays as well!
var theBestOfTheBest = 4;
console.log( amazingFrenchAuthors[ theBestOfTheBest ] ); // Logs "Jean-Paul Sartre"
console.log( amazingFrenchAuthors[ amazingFrenchAuthors.length - 1 ] ); // Logs "Montesqu

// This will turn a string into an array, with each element defined by the space
var bros = "Groucho Harpo Chico Zeppo".split(" ");
```

How to iterate through elements in an array

Stick to the for loop in most cases, but there are always thousands ways of doing things.

```
var greatPeople = [ "Louis Pasteur", "Jacques Cousteau", "Imhotep", "Sigmund Freud", "Wol

for ( var i = 0; i < greatPeople.length; i++ ) {
    console.log( greatPeople[ i ] ); // Will log out the "i-th" element
}


[ 'a', 'b', 'c' ].forEach( function (elem, index) {
    console.log(index + '. ' + elem);
});

// This will return:
// 0. a
// 1. b
// 2. c
```

Have a crack at these exercises.

And solutions are here.

How to set elements in an array

```
var amazingFrenchAuthors = [ "Alexandre Dumas", "Gustave Flaubert", "Voltaire", "Marcel P

amazingFrenchAuthors[0] = "Stendhal"; // Just access them and reassign!

console.log( amazingFrenchAuthors );
// Logs [ "Stendhal", "Gustave Flaubert", "Voltaire", "Marcel Proust"]
```

Common Methods and Properties for Arrays!

Properties, Methods and Functions

```
var amazingFrenchAuthors = [ "Alexandre Dumas", "Gustave Flaubert", "Voltaire", "Marcel P

console.log( amazingFrenchAuthors.length ); // Returns 4 - doesn't use a zero index

// POP //
amazingFrenchAuthors.pop(); // Removes the last element from an array and returns that el
// END POP //

// PUSH //
amazingFrenchAuthors.push(); // Adds one or more elements to the end of an array and retu
// END PUSH //

// REVERSE //
amazingFrenchAuthors.reverse(); // Reverses the order of the elements of an array — the f
// END REVERSE //

// SHIFT //
amazingFrenchAuthors.shift(); // Removes the first element from an array and returns that
// END SHIFT //

// UNSHIFT //
amazingFrenchAuthors.unshift(); // Adds one or more elements to the front of an array and
// END UNSHIFT //

// JOIN //
amazingFrenchAuthors.join(); // Joins all elements of an array into a string.
// END JOIN //

// SPLICE //
amazingFrenchAuthors.splice(); // Adds and/or removes elements from an array.
amazingFrenchAuthors = ["Alexandre Dumas", "Gustave Flaubert", "Voltaire", "Marcel Proust
amazingFrenchAuthors.splice(1,1);
// returns ["Gustave Flaubert"]
// amazingFrenchAuthors is ["Alexandre Dumas", "Voltaire", "Marcel Proust"]

amazingFrenchAuthors.splice(1,1, "Gustave Flaubert");
// Returns the deleted items, and adds in the next parameters ["Voltaire"]
// amazingFrenchAuthors is ["Alexandre Dumas", "Gustave Flaubert", "Marcel Proust"]
// END SPLICE //

amazingFrenchAuthors = ["Alexandre Dumas", "Gustave Flaubert", "Marcel Proust"];

// INCLUDE //
amazingFrenchAuthors.include( "Alexandre Dumas" ); // Returns true.
amazingFrenchAuthors.include( "Montesquieu" ); // Returns false.
// END INCLUDE //

// INDEX OF //
amazingFrenchAuthors.indexOf("Alexandre Dumas"); // Returns 0.
amazingFrenchAuthors.indexOf("Anäis Nin"); // Returns -1 if it doesn't find anything
// END INDEX OF //
```

Javascript Objects

In Javascript, an object is a standalone entity - filled with properties and types (or keys and values). It is very similar in structure to a dictionary.

So most javascript objects will have keys and values attached to them - this could be considered as a variable that is attached to the object (also allows us to iterate through them).

They are sometimes called assosciative arrays. Remember that they are not stored in any particular order (they can change order whenever).

How to create a Javascript Object

```
// With object literal
var newObject = {};

// Using Object
var newObject = new Object();
```

How to add Properties

```
// Remember to seperate by commas!
var newObject = {
  objectKey: "Object Value",
  anotherObjectKey: "Another Object Value",
  objectFunction: function () {

  }
};

var newObject = {};
newObject.objectKey = "Object Value";
newObject.objectFunction();
newObject["anotherObjectKey"] = "Another Object Value";

// Can also use Constructors and Factories - see Week 1 Day 5 notes.
```

How to access properties

Like all JS variables - both the object name and property names are case sensitive.

```
var favouriteCar = {
  manufacturer: "Jaguar",
  year: 1963,
  model: "E-Type"
}

favouriteCar.year

// Or

favouriteCar["year"]
```

How to iterate through an object

```
Object.keys(newObject); // Returns an array of all the keys in the specified object.
Object.getOwnPropertyNames(newObject); // So does this

var obj = {
  a: 1,
  b: 2,
  c: 3
};

for (var prop in obj) {
  console.log( "o." + prop + " = " + obj[prop] );
}
```

Deleting Properties

```
var favouriteCar = {
  manufacturer: "Jaguar",
  year: 1963,
  model: "E-Type"
}

delete favouriteCar.year;
```

Comparing Objects

In JavaScript objects are a reference type. Two distinct objects are never equal, even if they have the same properties. Only comparing the same object reference with itself yields true.

```
// Two variables, two distict objects with the same properties
var fruit = { name: "apple" };
var fruitbear = { name: "apple" };

fruit == fruitbear; // return false
fruit === fruitbear; // return false

// Two variables, a single object
var fruit = { name: "apple" };
var fruitbear = fruit;  // assign fruit object reference to fruitbear

// Here fruit and fruitbear are pointing to same object
fruit == fruitbear; // return true
fruit === fruitbear; // return true
```

Here are some exercises involving objects.

No simple way though. Underscore js has an implementation - "_.isEqual", lots of alternatives here, but I would stick to the underscore method. I love underscore.

Homework

- Arrays
  - Array and Function exercises
  - Array documentation on MDN
  - Speaking Javascript on Arrays
  - Javascript.info on Arrays
  - Eloquent Javascript: Arrays
- Objects
  - Geometry Lab
  - Sitepoint: Objects
  - Speaking Javascript: Objects
  - Speaking Javascript: Objects and Inheritance
  - MDN: Objects
  - Eloquent Javascript: Data
  - Eloquent Javascript: Objects
  - Code Academy: Object and Arrays
  - Object Playground
- MTA
- Keep on reading Eloquent JavaScript or Speaking JavaScript

Week 02

Week 02, Day 01.

What we covered today:

- Warmup Exercise
  - Solution
- MTA Demos
- This
- Slides for this & factories
- Making Objects
  - Factories
  - Constructors
- HTML
- CSS
- HTML & CSS slides

This

This is one of the most powerful things in Javascript, but also one of the hardest to understand.

In Javascript, this will always refer to the owner of the function we are executing. If the function is not within an object, or another function - this will refer to the global object - or window. Window is an object that exists in every browser, applying keys and values to this will make them globally accessible. Don't do it regularly though.

```
// GLOBAL THIS //
var doSomething = function () {
    console.log( this );
    // Will log the window object
}

// OBJECT THIS //
var objectFunction = {
    testThis: function () {
        console.log( this );
        // Would log the objectFunction object
    }
}
objectFunction.testThis();

// EVENT THIS //
var button = document.getElementById("myButton");
// This is a basic click handler - you aren't expected to understand this yet!
button.addEventListener( "click", function() {
    console.log( this );
    // Will log the HTML element that this event ran on (button with id myButton)
});
```

More or less...

In a simple function (one that isn't in another function or object) - "this" stays as the default - window.

In a function that is within an object, "this" is defined as the object - it's immediate parent.

In an event handler (a function that is called based on browser interaction), "this" is defined as the element that was interacted with.

Lizzie the Cat example

Further This Reading

- Todd Motto
- MDN
- Javascript is Sexy
- Quirks Mode

Making Objects / Constructors / Factories

First off, both constructors and factories are "blue prints". They bootstrap development. Often they are more hassle than they are worth though - so be wary. Think about whether all the code to get this running efficiently is actually worth it. Objects can get you through 95% of the time.

What is a constructor?

Essentially they are factories for objects. Normally JS objects are only maps from strings to values - however JS also supports inheritance - something that is truly object-oriented. They are quite similar to classes in other languages.

They become a constructor factory for objects if they are invoked via the new operator.

What does it entail?

The initial set up - this is where you set up the instance data...

```
var Point = function ( x, y ) {
    this.x = x;
    this.y = y;
}
```

The application of methods or functions... These will apply to any instance.

```
Point.prototype.dist = function () {
    return Math.sqrt( this.x * this.x + this.y * this.y );
};
```

The invocation of a new instance.

```
var p = new Point( 3, 4 );
console.log( "Point X: " + p.x );
```

We can also check to see if an object is an instance of a constructor:

```
typeof p
// Returns "object"

p instanceof Point
// true
```

- Phrogz
- [HTML5]

HTML

- The First Website Ever
- The History of Web Design
- A much nicer view of the History of the Web

What is HTML?

- Stands for Hyper Text Markup Language
- Currently at Version 5 (as of October 2014)
- Made up of tags (opening and closing usually), which in turn create elements

```
<!--    This whole thing is an element      -->
<tagname attribute="attribute_value"></tagname>

<!-- A paragraph tag with two classes, could be selected in CSS using p.default-paragraph
<p class="default-paragraph another-class">Some content in here</p>
```

What does an HTML document need?

```
<!doctype html> <!-- Always have this - it describes which version of HTML you are using
<html>
    <head></head> <!-- This is the meta data of the page, often invisible -->

    <body></body> <!-- This is where the actual content is -->
</html>
```

Common Elements

Actual Content

```
<!-- Heading Tags - getting less important the higher the number -->
<h1></h1>
<h2></h2>
<h3></h3>
<h4></h4>
<h5></h5>
<h6></h6>

<p></p> <!-- A paragraph tag -->

<!-- An HTML element can have attributes.  Attributes are key value pairs (just like java
<a href="generalassemb.ly">General Assembly</a>

<img src="" />
<video src=""></video>

<br /> <!-- a new line -->
<hr /> <!-- a horizontal line -->

<button></button>
<input />

<pre></pre> <!-- Preformatted text -->
<code></code>

<textarea></textarea>

<ul> <!-- Unordered list -->
    <li></li> <!-- List item -->
</ul>

<ol> <!-- Ordered list -->
    <li></li>
</ol>
```

Dividing Content

```
<div></div> <!-- A division, this is just a way to group content -->
<section></section>
<header></header>
<main></main>
<nav></nav>
<!-- etc. -->
```

For more information, see here.

Placeholder Stuff

TEXT:

- Meet the Ipsums
- Monocle Ipsum
- Samuel L. Jackson Ipsum
- Wiki Ipsum
- Social Ipsum
- 56 Other ones

IMAGES:

- Placeholdit
- Place Bear
- Dummy Image
- Place Kitten
- Fill Murray
- Nice Nice JPG - Vanilla Ice
- Place Cage

CSS

What is CSS?

- Stands for cascading style sheets
- It defines how HTML elements are to be represented
- Styles were added to HTML 4
- Currently at version 3 (CSS3)

```
selector {
    /* A Declaration */
    property:    value;
}

p {
    color: red;
}
```

CSS Selectors

Here are the basics of CSS Selectors, for more - go here.

```
p {} /* Selects all paragraph tags */

p.octocat {} /* Selects all paragraph tags with the class octocat */

.octocat {} /* Selects any tag with the class octocat */

p#octocat {} /* Selects any paragraph tag with the ID octocat */

#octocat {} /* Selects any element with the ID octocat */

* {} /* Selects all elements */

div p {} /* Selects all paragraph tags that are within div tags */

p, a {} /* Selects all p tags and all a tags */

p:hover {} /* Selects all p tags when they are hovered over */
```

CSS Specificity

Due to CSS's cascading nature, CSS rules will be overwritten. You need to have a bit of a handle on this.

- 1 point for an elements name
- 10 points for a selection based on a class
- 50 points for a selection based on an ID

For more details:

- Smashing Magazine - CSS Specificity
- CSS-Tricks - Specifics on CSS Specificity

This is a great way to learn selectors.

Floats and Clears

I'm not going to go into this that much - but these two articles will help explain this. For the most part - avoid floats and clears, stick to display: inline-block or inline instead. Only in the case that you need text to wrap around images should this be used.

- CSS Tricks - All About Floats
- Smashing Magazine's The Mystery of Floats

Homework

Hell is other people's HTML

- Understand: Separation of Concerns

- Play: CSS Diner
- Read: Learn Layout
- Memorise: 30 CSS Selectors
- Work through: Discover Dev Tools - Chapters 1 and 2
- Complete: Brook & Lyn
- Complete: Busy Hands
- Complete: eCardly, both versions

Week 02, Day 02.

What we covered today:

- Warmup Exercise
  - Solution
- Review
- Atom Package Control and Good Packages
- Advanced CSS
  - Slides
  - Positioning
  - Display
  - Transitions
  - Google Fonts
  - Custom Fonts
- Semantic HTML
- Guest speaker - Daisy!
  - Slides
  - Twitter!

Atom!

Atom is really powerful, but one of the things that makes it great is the versatility provided by packages. To install packages, navigate to:

Atom > Preferences > Install.

The first package that we need is called "Emmet", to find it, navigate to install and search for "emmet", it should be the top result with the most downloads.

From there, simply click install and Atom should do the rest.

Emmet is really helpful when writing HTML, it automates a lot of stuff for us. Check out here and here. They will give you lots of information about how to use Emmet.

Brief Intro to Emmet

Everything with Emmet comes from writing down a shortcut and then hitting tab at the end of the shortcut.

Tag Name

Whether it is a `p`, a `div`, or anything else. If you type the tag name, and then hit tab, it will create the element.

Classes and IDs (# or .)

```
div.className
Makes <div class="className"></div>

div#tagName
Makes <div id="tagName"></div>

div.firstClassName.secondClassName
Makes <div class="firstClassName secondClassName"></div>

div.className#secondClassName
Makes <div class="className" id="secondClassName"></div>
```

Children (>)

This is for nesting elements!

```
div>p
Makes <div><p></p></div>

header>nav>p
Makes <header><nav><p></p></nav></header>
```

Sibling (+)

This is for creating elements next to each other.

```
header+div.container
Makes <header></header><div class="container"></div>
```

*Multiplication (*)*

This is for making multiple elements at once.

```
div>ul>li*3
Makes <div>
    <ul>
        <li></li>
        <li></li>
        <li></li>
    </ul>
</div>
```

Climb Up (^)

This is to climb out of a nesting.

```
header>p^div
Makes <header>
    <p></p>
</header>
<div></div>
```

Grouping (())

This is to group chunks of elements so you don't need to worry about climbing.

```
(header>h1)+(nav>a)
Makes <header>
    <h1></h1>
</header>
<nav><a href=""></a></nav>
```

Attributes ([])

This is to give custom attributes.

```
img[src="" title="" alt=""]
Makes <img src="" alt="" title="">
```

Text ({})

This is to add text to things.

```
a{This is a link to something}
Makes <a href="">This is a link to something</a>
```

These things can all be used together!

Display

Every element on a page is a rectangular box (this is called the box-model). The display property is the thing that determines how that box behaves. There are a bunch of things that can be given to it, but the main ones are:

Inline

An inline element will accept margin and padding, but the element still sits inline as you might expect. Margin and padding will only push other elements horizontally away, not vertically. Inline elements will allow things to sit next to them and is the default value for some elements (em, span, and b).

An inline element will not accept height and width. It will just ignore it.

Inline Block

An element set to inline-block is very similar to inline in that it will set inline with the natural flow of text (on the "baseline"). The difference is that you are able to set a width and height which will be respected.

Block

A number of elements are set to block by the browser UA stylesheet. They are usually container elements, like

,
, and
. Also text "blocks" like
and

. Block level elements do not sit inline but break past them. By default (without setting a width) they take up as much horizontal space as they can.

Things won't sit next to block-level elements!

None

Totally removes the element from the page. Note that while the element is still in the DOM, it is removed visually and any other conceivable way (you can't tab to it or its children, it is ignored by screen readers, etc).

Positioning

Originally designed for scripting animation effects, this is not designed for layouts (but it is very possible!) - stick to display for that.

There are a bunch of different values for this, these are the most common ways:

Static

Static is the default value. It lets the element use the normal behaviour (what it is supposed to do). The top, right, bottom, left and z-index properties do not apply. It really does nothing.

Relative

It treats position: static as its starting point and, without changing any other elements position, allows us to move it around based on its static position.

For example, if we added top: -20px to an element, it would move that element up the page by 20px.

Absolute

Position absolute is very different to relative and static. If we add this property, the browser will not leave space for that element. Instead it references its nearest positioned parent (non-static) - this will often reference the body element.

Make sure you reference top and left or bottom and right when you use this property. Remember that this will change the document flow!

Fixed

This is quite similar to position absolute. You reference the top, bottom, left and right - except it sticks to the place that you tell it.

This is how they create fixed navigation bars etc. Remember that it references its nearest positioned element!

```
nav {
    position: fixed;
    top: 0;
    left: 0;
    height: 80px;
}
```

Get to know this stuff! And, go here for more information.

Google Fonts

- Go through here and Add the fonts that you want to your Collection
- Once you have selected all your fonts, click Use (bottom right)
- Choose the styles that you would like, and the character set
- Choose @import, and copy and paste the code into the top of your CSS file that it shows

- Reference the font with the code provided

Font Awesome

- Go here and copy the CDN link - `<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/font-awesome/4.3.0/css/font-awesome.min.css">`
- If you aren't using a server (Node or Ruby or anything else), add https at the front of the href
- Put this in the head of your HTML page
- Go through here and click on the icons that you want
- That will show you the HTML that you need

Custom Fonts

To reference custom fonts, you need to have the fonts saved in your project. Reference them in this way - make sure this is at the top of the CSS file! Reference this particular font by using the font-family name you referred to.

```
@font-face {
  font-family: 'GT Pressura';
  src:  url('GTPresurra.eot');
  src:  local('GT Pressura'),
        url('GTPressura.eot#iefix'),
        url('GTPressura.eot') format("truetype"),
        url("GTPressura.otf") format("opentype"),
        url("GTPressura.woff") format("woff"),
        url("GTPressura.woff2") format("woff2"),
        url("GTPressura.svg") format("svg");
}
```

To convert fonts, use this tool.

Fontello

This will generate a custom font for you, that you will then need to reference in your CSS. Once you have selected all the icons that you want, give it a family name (top right) and select download webfont. Make sure you have all the formats that you need (from the custom fonts section above), and reference each icon with the code that it gives you.

Variadic Attributes (margin etc.)

Variadic attributes are just shorthands to apply a number of properties. Take margin for example:

```
h1 {
    /* Applies to all four sides */
    margin: 1em;

    /* vertical | horizontal */
    margin: 5% auto;

    /* top | horizontal | bottom */
    margin: 1em auto 2em;

    /* top | right | bottom | left */
    margin: 2px 1em 0 auto;
}
```

Homework

Make a fan site

Week 02, Day 03.

What we covered today:

- Warmup Exercise
  - Solution
- Javascript Review
- The Document Object Model (DOM)
  - Events
  - Selectors

The Document Object Model (DOM)

What is the DOM?

It's the HTML that you wrote, when it is parsed by the browser. After it is parsed - it is known as the DOM. It can be different to the HTML you wrote, if the browser fixes issues in your code, if javascript changes it etc.

How do we use it?

The document object gives us ways of accessing and changing the DOM of the current webpage.

General strategy:

- Find the DOM node using an access method
- Store it in a variable
- Manipule the DOM node by changing its attributes, styles, inner HTML, or appending new nodes to it.

Here are some exercises to test this stuff.

Get Element by ID

```
// The method signature:
// document.getElementById(id);

// If the HTML had:
// <img id="mainpicture" src="http://placekitten.com/200/300">
// We'd access it this way:
var img = document.getElementById('mainpicture');

// DON'T USE THE HASH!
```

Get Elements by Tag Name

```
// The method signature:
// document.getElementsByTagName(tagName);

// If the HTML had:
<li class="catname">Lizzie</li>
<li class="catname">Daemon</li>
// We'd access it this way:
var listItems = document.getElementsByTagName('li');
for (var i =0; i < listItems.length; i++) {
  var listItem = listItems[i];
}
```

Query Selector and Query Selector All

```
// The HTML5 spec includes a few even more convenient methods.
// Available in IE9+, FF3.6+, Chrome 17+, Safari 5+:

document.getElementsByClassName(className);
var catNames = document.getElementsByClassName('catname');
for (var i =0; i < catNames.length; i++) {
  var catName = catNames[i];
}

// Available in IE8+, FF3.6+, Chrome 17+, Safari 5+:
document.querySelector(cssQuery);
document.querySelectorAll(cssQuery);
var catNames = document.querySelectorAll('ul li.catname');
```

Remember, some of these methods return arrays and some return single things!

Will return Single Elements

```
getElementById()
querySelector() * returns only the first of the matching elements
  var firstCatName = document.querySelector('ul li.catname');
```

Will return an Array

Others return a collection of elements in an array:

```
getElementByClassName()
getElementByTagName()
querySelectorAll()
  var catNames = document.querySelectorAll('ul li.catname');
  var firstCatName = catNames[0];
```

Do these exercises

Changing Attributes with Javascript

You can access and change attributes of DOM nodes using dot notation.

If we had this HTML:

```
<img id="mainpicture" src="http://placekitten.com/200/300">
```

We can change the src attribute this way:

```
var oldSrc = img.src;
img.src = 'http://placekitten.com/100/500';

// To set class, use the property className:
img.className = "picture";
```

Changing Styles with Javascript

You can change styles on DOM nodes via the style property.

If we had this CSS:

```
body {
  color: red;
}
```

We'd run this JS:

```
var pageNode = document.getElementsByTagName('body')[0];
pageNode.style.color = 'red';
```

CSS property names with a "-" must be camelCased and number properties must have a unit:

```
//To replicate this:

// body {
//   background-color: pink;
//   padding-top: 10px;
// }
pageNode.style.backgroundColor = 'pink';
pageNode.style.paddingTop = '10px';
```

Changing an elements HTML

Each DOM node has an innerHTML property with the HTML of all its children:

```
var pageNode = document.getElementsByTagName('body')[0];
```

You can read out the HTML like this:

```
console.log(pageNode.innerHTML);
```

```
// You can set innerHTML yourself to change the contents of the node:
pageNode.innerHTML = "<h1>Oh Noes!</h1> <p>I just changed the whole page!</p>"

// You can also just add to the innerHTML instead of replace:
pageNode.innerHTML += "...just adding this bit at the end of the page.";
```

DOM Modifying

The document object also provides ways to create nodes from scratch:

```
document.createElement(tagName);
```

```
document.createTextNode(text);
```

```
document.appendChild();
```

```
var pageNode = document.getElementsByTagName('body')[0];

var newImg = document.createElement('img');
newImg.src = 'http://placekitten.com/400/300';
newImg.style.border = '1px solid black';
pageNode.appendChild(newImg);

var newParagraph = document.createElement('p');
var paragraphText = document.createTextNode('Squee!');
newParagraph.appendChild(paragraphText);
pageNode.appendChild(newParagraph);
```

Have a crack at these exercises

Events

Adding Event Listeners

In IE 9+ (and all other browsers):

```
domNode.addEventListener(eventType, eventListener, useCapture);
```

```
// HTML = <button id="counter">0</button>

var counterButton = document.getElementById('counter');
var button = document.querySelector('button')
button.addEventListener('click', makeMadLib);

var onButtonClick = function() {
    counterButton.innerHTML = parseInt(counterButton.innerHTML) + 1;
};
counterButton.addEventListener('click', onButtonClick, false);
```

Some Event Types

The browser triggers many events. A short list:

- mouse events (MouseEvent): mousedown, mouseup , click, dblclick, mousemove, mouseover, mousewheel , mouseout, contextmenu
- touch events (TouchEvent): touchstart, touchmove, touchend, touchcancel
- keyboard events (KeyboardEvent): keydown, keypress, keyup
- form events: focus, blur, change, submit
- window events: scroll, resize, hashchange, load, unload

Getting Details from a Form

```
// HTML
// <input id="myname" type="text">
// <button id="button">Say My Name</button>

var button = document.getElementById('button');
var onClick = function(event) {
    var myName = document.getElementById("myname").value;
    alert("Hi, " + myName);
};
button.addEventListener('click', onClick);
```

Have a crack at these exercises and here are some slides that may help.

Homework!

CATS

INSPIRATION FOR CATS