Table of Contents

Daily Stuff

Modules

WDI 14

Daily Stuff

Modules

- Heroku
- Cloudinary
- Underscore

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.1.html
```

```
# 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 o
riginal!
> 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_b
ooks file (it will create the file if necessary)
> cat books | sort > sorted_books
# This will rename the sorted_books to books (and will overwrite the books file if it
already exists)
> mv sorted_books books
# This will open and show the contents and books, and shows only lines that have the w
ord script in them (case sensitive!)
> cat books | grep script
# To copy a file, the command is cp (this needs parameters - or arguments. It needs a
source and a destination).
# cp source destination
> cp books my_books
# To remove things, use the rm command (this doesn't get moved to your trash! It will
delete it permanently and is impossible to undo)
> 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 follo
wing command, and use the contents of the files to decide whether it can run that part
icular program or command
> 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 expr
ession into the console.

// 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 t
he function.
```

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", "Marce
l Proust", "Jean-Paul Sartre", "Stendhal", "Anäis Nin", "Simone de Beauvoir", "Rene De
scartes", "Montesquieu" ];

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 "Monte
squieu" (the last element)

// 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", "
Wolfgang Amadeus Mozart" ];

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", "Marce
l Proust"];

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", "Marce
l Proust"];

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
 element.
// END POP //

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

// REVERSE //
amazingFrenchAuthors.reverse(); // Reverses the order of the elements of an array — th
e first becomes the last, and the last becomes the first.
// END REVERSE //

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

// UNSHIFT //
amazingFrenchAuthors.unshift(); // Adds one or more elements to the front of an array
and returns the new length of the array.
// 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 Pro
ust"];
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-paragr
aph.another-class {} -->
<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 usi
ng -->
<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 j
avascript objects) that provide additional information. They look like this. This is a
 link by the way (or anchor tag) -->
<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

Week 02, Day 04.

What we covered today:

- Warmup Exercise
  - Solution
  - More dancing cats!
    - Slides
  - The window
  - Animations in JS
  - Events
  - jQuery
    - Slides

Window

When you run JS in the browser, it gives you the window object with many useful properties and methods:

```
window.location.href; // Will return the URL
window.navigator.userAgent; // Tell you about the browser
window.scrollTo(10, 50);
```

Lots of things that we have been using are actually a part of the window object. As the window object is the assumed global object on a page.

```
window.alert("Hello world!"); // Same things
alert("Hello World!");

window.console.log("Hi"); // Same things
console.log("Hi");
```

Animation in JS

The standard way to animate in JS is to use these 2 window methods.

To call a function once after a delay:

```
window.setTimeout(callbackFunction, delayMilliseconds);
```

To call a function repeatedly, with specified interval between each call:

```
window.setInterval(callbackFunction, delayMilliseconds);
```

Commonly used to animate DOM node attributes:

```
var makeImageBigger = function() {
  var img = document.getElementsByTagName('img')[0];
  img.setAttribute('width', img.width+10);
};
window.setInterval(makeImageBigger, 1000);
```

Animating Styles

It's also common to animate CSS styles for size, transparency, position, and color:

```
var img = document.getElementsByTagName('img')[0];
img.style.opacity = 1.0;
window.setInterval(fadeAway, 1000);
var fadeAway = function() {
  img.style.opacity = img.style.opacity - .1;
};

// Note: opacity is 1E9+ only (plus all other browsers).
//
var img = document.getElementsByTagName('img')[0];
img.style.position = 'absolute';
img.style.top = '0px';
var watchKittyFall = function() {
  var oldTop = parseInt(img.style.top);
  var newTop = oldTop + 10;
  img.style.top = newTop + 'px';
};

window.setInterval(watchKittyFall, 1000);
// Note: you must specify (and strip) units.
```

Stopping an Animation

To stop at an animation at a certain point, store the timer into a variable and clear with one of these methods:

```
window.clearTimeout(timer);
window.clearInterval(timer);
```

```
var img = document.getElementsByTagName('img')[0];
img.style.opacity = 1.0;

var fadeAway = function() {
  img.style.opacity = img.style.opacity - .1;
  if (img.style.opacity < .5) {
    window.clearInterval(fadeTimer);
  }
};
var fadeTimer = window.setInterval(fadeAway, 100);
```

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);
```

Get into this exercise!

Javascript Libraries

What is a Library?

A collection of reusable methods for a particular purpose. You just reference a javascript file with a particular library in it - and away you go!

jQuery is the most common library...

Have a crack at these exercises.

jQuery

What is it?

An open source JavaScript library that simplifies the interaction between HTML and JavaScript. A Javascript library is collection of reusable methods for a particular purpose.

It was created by John Resig in 2005, and released in January of 2006.

Built in an attempt to simplify the existing DOM APIs and abstract away cross-browser issues.

Why pick jQuery?

- Documented
- Lots of libraries
- Small-ish size (23kb)
- Everything works in IE 6+, Firefox 2+, Safari 3+, Chrome, and Opera 9+ (if you are using 1.11.3 or previous, greater than that scraps up until IE8 support)
- Millions and millions of sites using it. According to BuiltWith.com, 52,339,256 live sites are using it. 26.95% of sites apparently use it.

What does it do for us?

- Data Manipulation
- DOM Manipulation
- Events
- AJAX
- Effects and Animation
- HTML Templating
- Widgets / Theming
- Graphics / Chart
- App Architecture

Why use it?

```
// No library:
var elems = document.getElementsByTagName("img");
for (var i = 0; i< elems.length; i++) {
  elems[i].style.display = "none";
}

// jQuery:
$('img').hide();
```

The Basics

Select -> Manipulate -> Admire

```
var $paragraphs = $("p")
$paragraphs.addClass('special');

// OR
$("p").addClass("special");
```

How to select things? All CSS selectors are valid, plus a whole heap more.

Some common ones are:

- :first
- :last
- :has()
- :visible
- :hidden

Reading Elements

If we had this element in the HTML...

```
<a id="yahoo" href="http://www.yahoo.com" style="font-size:20px">Yahoo!</a>
```

We can select it using `$("a#yahoo")`

We can store it using `var $myLink = $("#yahoo");`

We can get the content within it using `$("#yahoo").html()`

We can get the text within it using `$("#yahoo").text()`

We can get the HREF attribute using `$("#yahoo").attr("href")`

We can get the CSS attribute using `$("#yahoo").css('font-size')`

Changing Elements

```
$("#yahoo").attr("href", "http://generalassemb.ly")
```

```
$("#yahoo").css("font-size", "25px")
```

```
$("#yahoo").text("General Assembly")
```

Create, Manipulate and Inject

```
// Step 1: Create element and store a reference
    var $para = $('<p></p>'); // You can create any element with this!

// Step 2: Use a method to manipulate (optional)
    $para.addClass('special'); // So many functions you could use

// Step 3: Inject into your HTML
    $('body').append($para); // Also could use prepend, prependTo or appendTo as well
```

Regular DOM Nodes to jQuery Objects

```
var $paragraphs = $('p'); // an array

var myParagraph = paragraphs[0]; // a regular DOM node

var $myParagraph = $( paragraphs[0] ); // a jQuery Object
// or
var $myParagraph = $paragraphs.el(0); // This is the preferred method!

// We can also loop through our array...
for( var i = 0; i < paragraphs.length; i++ ) {
   var element = paragraphs[i];
   var paragraph = $(element);
   paragraph.html(paragraph.html() + ' wowee!!!!');
};

// Or use jQuery to do it - this is preferred
$paragraphs.each(function () {
  var $this = $( this );
  $this.html( $this.html() + " wowee!!!" );
});
```

The Ready Event

```
$(document).ready(function(){

});
```

In order to do cool jQuery stuff, we need to make sure that all of the content so put all your DOM related jQuery code in the document ready.

Have a crack at these exercises.

Homework

This and this.

Events

```
var onButtonClick = function() {
  console.log('clicked!');
};

$('button').on('click', onButtonClick); // Attaching an event handler with a defined f
unction to the button

$('button').on('click', function () { // Attaching an event handler with an anonymous
function to the button
  console.log('clicked!');
});

$('button').click(onButtonClick)
```

Other Event Types

- Keyboard Events - 'keydown', 'keypress', 'keyup'
- Mouse Events - 'click', 'mousedown', 'mouseup', 'mousemove'
- Form Events - 'change', 'focus', 'blur'

Arguments get passed into callbacks by default.

```
var myCallback = function (event) {
    console.log( event );
    // The event parameter is a big object filled with ridiculous amounts of details a
bout when the event occurred etc.
};

$('p').on('mouseenter', myCallback);
```

Preventing Default Events

```
$('a').on('click', function (event) {
    event.preventDefault();
    console.log('Not going there!');
});
$('form').on('submit', function (event) {
    event.preventDefault();
    console.log('Not submitting, time to validate!');
});
```

Effects and Animations

```
$('#error').toggle(1000);

$('#error').fadeIn();

$('#error').show(1000, function(){
    $(this).addClass('redText')
});
```

jQuery Patterns and Anti-Patterns

```
// Pattern: name variables with $
var $myVar = $('#myNode');

// Pattern: bind events to the document or "body"
$("body").on('click', 'a', myCallback);

// Storing References
// Pattern: store references to callback functions
var myCallback = function(argument) {
  // do something cool
};

// $(document).on('click', 'p', myCallback);

// Anti-pattern: anonymous functions
$("body").on('click', 'p', function(argument) {
  // do something anonymous
});
```

Chaining jQuery Functions

```
banner.css('color', 'red');
banner.html('Welcome!');
banner.show();

// Is the same as:
banner.css('color', 'red').html('Welcome!').show();

// Is the same as:
banner.css('color', 'red')
      .html('Welcome!')
      .show();
```

Read here for more information

Have a crack at these exercises.

Plugins with jQuery

To Find Plugins

- Go through the jQuery plugin website
- Or here, I prefer this one

How to select a good plugin

- Well documented
- Flexible
- Community
  - Number of forks and issues
- Actively maintained?
- File size
- Browser compatibility
- Responsive

For more details, go here.

How to use a plugin

- Download the plugin and associated files from the site or Github repository
- Copy them into your project folder
- In the HTML, reference any associated CSS
- After you reference jQuery, and before you reference your own code - add the script tag that references the plugins JS file(s)

Give this a go

- Create a new Javascript file and link to it with a script tag at the bottom.
- Create a variable to store a reference to the img.
- Change the style of the img to have a "left" of "0px", so that it starts at the left hand of the screens.
- Create a function called catWalk() that moves the cat 10 pixels to the right of where it started, by changing the "left" style property.
- Call that function every 50 milliseconds. Your cat should now be moving across the screen from left to right. Hurrah!
- Bonus #1: When the cat reaches the right-hand of the screen, restart them at the left hand side ("0px"). So they should keep walking from left to right across the screen, forever and ever.
- Bonus #2: When the cat reaches the right-hand of the screen, make them move backwards instead. They should keep walking back and forth forever and ever.

- Bonus #3: When the cat reaches the middle of the screen, replace the img with an image of a cat dancing, keep it dancing for 10 seconds, and then replace the img with the original image and have it continue the walk.

Homework!

MOAR CATS

Additionally, look at these:

Try jQuery jQuery - codeacademy Learn jQuery jQuery fundamentals

Week 02, Day 05.

What we covered today:

- Cat Demos
- Events
- jQuery
- Happy Hour
- PROJECT ZERO!

ip)

Events

```
var onButtonClick = function() {
  console.log('clicked!');
};

$('button').on('click', onButtonClick); // Attaching an event handler with a defined f
unction to the button

$('button').on('click', function () { // Attaching an event handler with an anonymous
function to the button
  console.log('clicked!');
});

$('button').click(onButtonClick)
```

Other Event Types

- Keyboard Events - 'keydown', 'keypress', 'keyup'
- Mouse Events - 'click', 'mousedown', 'mouseup', 'mousemove'
- Form Events - 'change', 'focus', 'blur'

Arguments get passed into callbacks by default.

```
var myCallback = function (event) {
    console.log( event );
    // The event parameter is a big object filled with ridiculous amounts of details a
bout when the event occurred etc.
};

$('p').on('mouseenter', myCallback);
```

Preventing Default Events

```
$('a').on('click', function (event) {
    event.preventDefault();
    console.log('Not going there!');
});
$('form').on('submit', function (event) {
    event.preventDefault();
    console.log('Not submitting, time to validate!');
});
```

Effects and Animations

```
$('#error').toggle(1000);

$('#error').fadeIn();

$('#error').show(1000, function(){
    $(this).addClass('redText')
});
```

Try these.

jQuery Patterns and Anti-Patterns

```
// Pattern: name variables with $
var $myVar = $('#myNode');

// Pattern: bind events to the document or "body"
$("body").on('click', 'a', myCallback);

// Storing References
// Pattern: store references to callback functions
var myCallback = function(argument) {
  // do something cool
};

// $(document).on('click', 'p', myCallback);

// Anti-pattern: anonymous functions
$("body").on('click', 'p', function(argument) {
  // do something anonymous
});
```

Chaining jQuery Functions

```
banner.css('color', 'red');
banner.html('Welcome!');
banner.show();

// Is the same as:
banner.css('color', 'red').html('Welcome!').show();

// Is the same as:
banner.css('color', 'red')
      .html('Welcome!')
      .show();
```

Read here for more information

Have a crack at these exercises.

Plugins with jQuery

To Find Plugins

- Go through the jQuery plugin website
- Or here, I prefer this one

How to select a good plugin

- Well documented
- Flexible
- Community
    - Number of forks and issues
- Actively maintained?
- File size
- Browser compatibility
- Responsive

For more details, go here.

How to use a plugin

- Download the plugin and associated files from the site or Github repository
- Copy them into your project folder
- In the HTML, reference any associated CSS
- After you reference jQuery, and before you reference your own code - add the script tag that references the plugins JS file(s)

Give this a go

HOMEWORK!

Project 0: https://gist.github.com/ga-wolf/efe1026df941ae49fc1da5871f712a3f

Libraries that may help:

- Sweet alert
- Animate CSS

Week 04

Week 04, Day 01

Stuff from project week:

- Brice Lechâtellier's slides on CSS
- Deploying to gitHub pages

What we covered today:

- Warmup Exercise
  - Solution
- RVM and Ruby installation guide
- AirBnB style guide
- Ruby
  - Data Types
  - Variables
  - Conditionals
  - Control Structures
  - Methods (Functions in JS)

How to get Ruby

Install Developer Tools from Xcode (this should have happened for most of you)

```
xcode-select --install
```

Access a specific URL using a secured line and run the downloaded program

```
curl -sSL https://get.rvm.io | bash -s stable
```

Restart the terminal, and try running the command `rvm`.

If it doesn't work...

- Open the bash_profile up in Sublime

  ```
  subl ~/.bash_profile
  ```

- Add these lines into the bottom of the bash_profile and save it

  ```
  [[ -s "$HOME/.rvm/scripts/rvm" ]] && source "$HOME/.rvm/scripts/rvm" `
  export PATH="$PATH:$HOME/.rvm/bin"
  source ~/.profile
  ```

Restart the terminal again

`rvm`

`rvm list known`

`rvm get stable --auto`

Go here and find the most recent version - https://www.ruby-lang.org/en/downloads/

Latest version at the time of writing was 2.2.2, swap those in for the latest stable version that shows on that website

`rvm install ruby-2.2.3`

`rvm --default use 2.2.3`

Let's test that it has all worked.

- `ruby -v`
- `rvm -v`
- `which ruby` - should not return anything in the /usr/local/bin
- `which rvm`

If all of this has worked, run...

`gem install pry`

Common Commands

`ruby -v` - Will return the current version of Ruby

`which ruby` - What is the path to the version of Ruby you are using

`ruby hello_world.rb` - Runs the hello_world.rb file

`irb` - Runs a ruby console

`pry` - Runs a better console

`<CTRL> + D` - Ends a file in irb or ruby

Data Types

Strings

Again, delimited by quotes.

`"string"` `'string'`

Numbers

There are multiple types:

- Integer (Fixnum)
- Float
- Bignum

Logical Operators

Lots of them, but the basic ones are:

`+=` - Add then assign `-=` - Minus then assign `*=` - Multiply then assign `/=` - Divide then assign `**` - To the power of

Comparison Operators

All the same ones...

```
>
>=
<
<=
== # Normally stick to two equals in Ruby
===

<=> (returns -1 if less than, 0 if equal, and 1 if greater than)
etc.
```

Variables

No need for the var keyword.

`ruby = "is nice"`

Much harder to make global variables, it isn't the default behaviour in Ruby

Methods (Functions in JS)

`puts "this is like console.log` `print "this is also like console.log` `p "this is a bit more complex`

Brackets are mostly optional, occasionally necessary (only in method or function chaining)

`puts("this is like console.log)`

Basic Naming Conventions

snake_case_everywhere - very rare to see camelCase!

Variable Interpolation in Strings

Interpolation just means you can put code inside

```
name  = "gilberto"
drink = "scotch"

"My name is #{ name } and I drink #{ drink }!"
# A lot nicer than "my name is " + name + " and I drink " + drink
# Which is the way you would do it in JS
```

Interpolation only works with double quotes!! Single quotes means leave this string alone, this is mine

Comments in Ruby

```
# This is is a single line comment

# This is
# a multiline
# comment

# OR (don't do this)

=begin
This is also a multi line comment
You can't have any an empty line between the =begin and the start of the comment
=end
```

Getting User Input

In JS, we have prompt etc.

```ruby
# Initial greeting
puts "What is your first name?"

# first_name = gets
# This will wait for user input, and include the new line in the variable

first_name = gets.chomp
# This will wait for user input, and strip the new line from the variable
# For more documentation on chomp - http://ruby-doc.org/core-2.2.0/String.html#method-i-chomp

puts "Your first name is #{ first_name }."

puts "What is your surname?"
surname = gets.chomp
puts "Your surname is #{ surname }."

puts "Your full name is #{ first_name } #{ surname }"
# fullname = "#{ first_name } #{ surname }"
# Sames as ... puts "Your full name is #{ fullname }"

puts "What is your address?"
address = gets.chomp
puts "Your name is #{ fullname } and you live at #{ address }"

# INTERPOLATION ONLY WORKS ON DOUBLE QUOTES!
```

Conditionals

IF STATEMENTS

```ruby
if 13 > 10
    p "Yep, it is a bigger number"
end

grade = "A"

if conditional
    # To do
elsif conditional
    # To do
else
    # To do
end

p "Yep, it is a bigger number" if 13 > 10 # This only works in single line statements

# It's called a modifier (if modifier)
```

UNLESS STATEMENTS

```
x = 1
unless x > 2
    puts "x is less than 2"
else
    puts "x is greater than 2"
end

code_to_perform unless conditional
```

CASE STATEMENTS

Think of these as shorter if statements, but don't overuse them (particularly in JS)

```
grade = 'B'
case grade
when 'A'
    p 'Great Job'
when 'B'
    p 'Good Job'
when 'C'
    p 'Adequate Job'
else
    p 'Talk to the Hand'
end

case expression_one
when expression_two, expression_three
    statement_one
when expression_four, expression_five
    statement_two
else
    statement_three
end

# Very similar to the switch statement in Javascript!
```

Now that we know this stuff, give these exercises a go.

WHILE LOOPS

```
while conditonal
    statement
end

while true
    p "OMG"
end # BAD IDEA

i = 0
while i < 5
    puts "I: #{ i }"
    i += 1
end
```

UNTIL LOOPS

```
until conditional
    statement
end

i = 0
until i == 5
    puts "I: #{ i }"
    i += 1
end
```

ITERATORS

So, so common in Ruby.

```
5.times do
    puts "OMG"
end

5.times do |i|
    puts "I: #{ i }"
end
# The thing that times wants to pass into me is stored as the parameter between the pi
pe characters

5.downto(0) do |i|
    puts "I: #{ i }"
end
```

FOR LOOPS (No one ever uses these)

```
# Don't ever use them

for i in 0..5
   puts "I: #{ i }"
end
```

Generating Random Numbers

```
Random.rand # Generates a number between 0 and 1
Random.rand(10) # Generates a random number up to 10 (including zero and 10)
Random.rand(5..10) # Generates a number between 5 and 10 (also includes them)
Random.rand(5...10) # Does not include 5 and 10
```

Now that you know this stuff, have a crack at these exercises.

Methods or Functions

```
def hello
    # A plain method
end

hello # called like this

def hello( name )
    # A plain method that takes a parameter
    # When calling this, you MUST pass in a parameter or it will throw an error
end

hello "Wolf"    # Called this way
hello("Wolf")   # Or this

def hello( name = "World" )
    # A function with a default parameter
    # This won't throw an error in the case that you don't pass a parameter in
end

hello           # Works this way
hello("Wolf")   # Or this
```

Methods in Ruby have an implicit return, meaning that you don't need to actually use the return keyword, it does it automatically.

Parentheses are optional!

Here is the homework for tonight!

Week 04, Day 02

Stuff from Toby What we covered today:

- Warmup
  - Solution
- Demos
- History of Ruby
- Collections
  - Arrays
  - Hashes
- Lab (MTA)

History of Ruby

Created in 1993 by Yukihiro Matsumoto (Matz). He knew Perl and he also knew Python. He thought that Perl smelt like a toy language apparently, and he disliked Python because it wasn't a true object-oriented programming language. Ruby was primarily influenced by Perl and SmallTalk though.

Matz wanted a language that:

- Syntactically Simple
- Truly Object-Oriented
- Had Iterators and Closures
- Exception Handling
- Garbage Collection
- Portable

Programming Mottos

Perl - There's More Than One Way To Do It (T.M.T.O.W.T.D.I)

Python - There Should Be One And Only One Way To Do It

Ruby - Designed For Programmer Happiness

Here is a pretty good history of Ruby.

Arrays

Creation of an Array

```ruby
# LITERAL CONSTRUCTOR

bros = []
bros = [ 'groucho', 'harpo', 'chico' ]

# CLASS KEYWORD

bros = Array.new # => []
bros = Array.new( 3 ) # => [ nil, nil, nil ]
bros = Array.new( 3, true ) # => [ true, true, true ]

bros = Array.new(4) { Hash.new } # => [{}, {}, {}, {}]
bros = Array.new(2) { Array.new(2) } # => [ [nil, nil], [nil, nil] ]

# CHARACTER MODIFIERS - http://en.wikibooks.org/wiki/Ruby_Programming/Syntax/Literals

%w{ Hello World }
%w[ Hello World ]
%w/ Hello World /
```

Accessing Elements

```ruby
arr = [1, 2, 3, 4, 5, 6]
arr[2]      # => 3
arr[100]    # => nil
arr[-1]     # => 6
arr[-3]     # => 4
arr[2, 3]   # => [3, 4, 5]
arr[1..4]   # => [2, 3, 4, 5]
arr[-1..-2] # => [5, 6]

# These work for reassignment as well!

arr[0] = 0
arr[0] = 1

arr.at(0)   # => 1

arr.first   # => 1
arr.last    # => 6

arr.take(3) # => [1, 2, 3] - Grabs the first three elements
arr.drop(3) # => [4, 5, 6] - Grabs the last three elements

arr.fetch(100) # => IndexError: index 100 outside of array bounds: -6...6
arr.fetch(100, "ERROR") # => "ERROR"
```

Adding Items to an Array

```
arr = [1, 2, 3, 4]
arr.push(5) # => [1, 2, 3, 4, 5]
arr << 6    # => [1, 2, 3, 4, 5, 6] Uses push behind the scenes

arr.unshift(9) # => [0, 1, 2, 3, 4, 5, 6] Adds an element to the start

arr.insert( 3, 'Serge' ) # => [ 0, 1, 2, 'Serge', 3, 4, 5, 6 ]
arr.insert( 4, 'didnt marry', 'Jane') # =>  [0, 1, 2, 'Serge', 'didnt marry', 'Jane',
3, 4, 5, 6]
```

Removing Items from an Array

```
# Pop removes the last element and returns it (it is destructive)

arr = [1, 2, 3, 4, 5, 6]
arr.pop     # => 6
arr         # => [1, 2, 3, 4, 5]

# To retrieve and at the same time remove the first item

arr.shift # => 1

# Delete at a particular index

arr.delete_at( 2 )

# To delete a particular element anywhere

arr = [1, 2, 2, 3]
arr.delete(2) # => [1, 3]

# Compact will remove nil values

arr = ['foo', 0, nil, 'bar', 7, 'baz', nil]
arr.compact  #=> ['foo', 0, 'bar', 7, 'baz']

# Remove duplicates

arr = [2, 5, 6, 556, 6, 6, 8, 9, 0, 123, 556]
arr.uniq    # => [2, 5, 6, 556, 8, 9, 0, 123]
```

Iterating Over Arrays

```ruby
arr = [1, 2, 3, 4, 5]

arr.each do |el|
    puts el
end

arr.each { |el| puts el }

arr.reverse_each do |el|
    puts el
end

arr.reverse_each { |el| puts el }

# The map method will create a new array based on the original one, but with the value
s modified by the supplied block

arr = [1, 2, 3]
arr.map { |a| 2 * a }  # => Returns [ 2, 4, 6 ] but doesn't change the original
arr.map! { |a| 2 * a } # => Changes the original and returns it

# DON'T DO IT THESE WAYS!

arr = [1,2,3,4,5,6]
for x in 0..(arr.length-1)
    puts arr[x]
end

# or, with while:
x = 0
while x < arr.length
    puts arr[x]
    x += 1
end

for el in arr
    puts el
end
```

Selecting Items from an Array

Elements can be selected from an array according to criteria defined in a block. The selection can happen in a destructive or a non-destructive manner. While the destructive operations will modify the array they were called on, the non-destructive methods usually return a new array with the selected elements, but leave the original array unchanged.

```
arr = [1, 2, 3, 4, 5, 6]
arr.select { |a| a > 3 }        # => [4, 5, 6]
arr.reject { |a| a < 4 }        # => [4, 5, 6]

# You can use these two with the exclamation mark to make them destructive

# The next two are destructive!

arr.delete_if { |a| a < 4 }     # => [4, 5, 6]
arr.keep_if { |a| a < 4 }       # => [1, 2, 3]
```

Ruby Array Comparison Tricks

```
array1 = ["x", "y", "z"]
array2 = ["w", "x", "y"]

array1 | array2
# Combine Arrays & Remove Duplicates(Union)
# => ["x", "y", "z", "w"]

array1 & array2
# Get Common Elements between Two Arrays(Intersection)
# => ["x", "y"]

array1 - array2
# Remove Any Elements from Array 1 that are contained in Array 2.(Difference)
# => ["z"]
```

Thanks Rodney and this site

At this point, have a crack at these exercises

Destructive Methods vs Non-destructive Methods

There are destructive methods and non-destructive methods in Ruby. Destructive methods will affect the original, whereas non-destructive will leave it alone and just return an altered copy. Destructive methods normally end with an !.

Predicate Method

More or less, predicate methods are those that return a boolean value. They always end with a ?.

What are blocks?

The content in iterators are called blocks, they are quite similar to anonymous functions in javascript.

```
arr = [1, 2, 3]

# The content between the do and the end is the block
arr.each do |el|
    puts el
end

# The content between the curly brackets is the block
arr.each { |el| puts el }
```

Object IDs, Strings vs. Symbols, plus False Interlude

In Ruby, every single thing is an object. Absolutely everything. Doesn't matter if it is a string, boolean or anything - they are all objects and all get assigned an object_id. Everytime a new one is created, even if it looks identical, a new object_id (a new place in memory) will be created.

```
"Wolf".object_id
# => 70131971988560

{}.object_id
70131953807740

false.object_id
0
```

Each time you reference a new string, hash or array, it declares a new object_id - meaning that it takes up more memory in your Ruby program. If you imagine a database with thousands of entries, these small things add up to huge amounts of memory.

Symbols aren't like that, they are assigned a static place in memory (meaning that they don't need to be redefined). They behave in the exact same way as strings are easily translated back. Always use symbols for keys on objects.

```
:wolf.object_id
1147228
```

```
# Conversions!

:wolf.to_s
# => "wolf"

"wolf".to_sym
# => :wolf
```

False Interlude

The only things that are considered false in Ruby are the boolean false, and the nil value.

Hashes

Creation of a Hash

```
# Literal Constructor
# "=>"" is called a hash rocket

hash = {}
serge = {
    :name => "Serge",
    :nationality => "French"
}
serge = {
    "name" => "Serge",
    "nationality" => "French"
}
serge = { # Keys stored as symbols!
    name: "Serge",
    nationality: "French"
}

# Class Constructor

hash = Hash.new

# Normally a hash will return nil if the property is undefined
# We can pass in default values to this quite easily though

hash = Hash.new( "WOLF" )
hash["Jack"] #=> Will return "WOLF"

# If you create the hash using the literal though...

hash = {}
hash.default = "WOLF"
hash["JACK"] #=> Will return "WOLF"
```

Accessing Elements

```
serge = { # Keys stored as symbols!
    name: "Serge",
    nationality: "French"
}

serge[:name]

serge = {
    "name" => "Serge",
    "nationality" => "French"
}

serge["name"]
```

Adding Items to a Hash

```
# Notice no hash rocket!

serge[:counterpart] = "Jane (temporarily)"

# This is the same way as you access them!

p serge[:counterpart] # => "Jane (temporarily)"
```

Removing Items from a Hash

```
serge.delete(:counterpart)
```

Iterating Over Hashes

```
serge = { # Keys stored as symbols!
    name: "Serge",
    nationality: "French"
}

# Will run for keys and values
serge.each do |all|
    puts all
end

# Will run for each key and value pair
serge.each do |key, value|
    puts "Key: #{key} and Value: #{value}"
end

# Return the current key
serge.keys.each do |key|
    puts key
end

# Return the current value
serge.values.each do |value|
    puts value
end

# Thousands of other ways to do this though
```

Have a crack at these exercises. and these.

Here is the homework! MTA again!

Week 04, Day 03

What we covered today:

Sinatra

What is it?

Sinatra is a **D**omain **S**pecific **L**anguage (DSL). That doesn't mean that much, but more or less Sinatra is a web application framework - it is a gem (or library) that we include to give us the methods that we need to be able to construct simple web applications (with minimal effort apparently).

How do we use it?

It is a gem like anything else!! We need two gems for this though - we have `sinatra` itself (which gives us all of those necessary methods), and we have `sinatra-contrib` - which is anything worthy that has been contributed to the Sinatra gem. This is what includes the live reloader etc.

Run these two lines first!

```
gem install sinatra
gem install sinatra-contrib
```

For a basic Sinatra application, we don't need that much. We just need to require the necessary gems. We call the main Sinatra file `main.rb`.

```
require 'sinatra'           # Gets sinatra itself
require 'sinatra/reloader'  # Live reloader from sinatra-contrib
```

This is all that is required to technically run the server (but it won't do anything or respond to anything).

To run the server, we just use to ruby to run the file - `ruby main.rb`. Normally this will run a server at localhost:4567.

Routes

In Sinatra, a route is an HTTP method paired with a URL matching pattern. Each one is linked to a block.

```
get '/' do
    "This is a get request to the root path - visited at localhost:4567/"
end

get '/anything' do
    "This is a get request - visited at localhost:4567/anything" # This is returned as
 a response
end
```

Routes are matched in the order they are defined. The first route that matches the request is invoked.

These are literal matches. We actually have to visit `/anything`, this is annoying because we don't always not what we are going to receive.

The way we solve this is by using named parameters. Whatever is matched by the thing prefixed with the colon is stored in the params hash (which is automatically generated for us).

```
get '/hello/:name' do
  # matches "/hello/foo" or "/hello/bar" or anything else that starts with "/hello/""
  # params['name'] might be 'foo' or 'bar'

  "Hello #{ params['name'] }!"
end
```

Views / Templates

At this point, we are just rendering a piece of text to the page - that isn't ideal. We want it to be well presented!! In its most basic form, we use ERB (embedded ruby).

ERB is very smart, but requires a particular setup. For it to work, we need a `views` folder. This is where all of our HTML (stored as ERB) will go.

The first thing we always do is create a `layout.erb` file. This will be used by default everytime Sinatra sees the erb command. This is where all of the stuff that you want to be on every page should go.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Our Basic Sinatra App</title>
</head>
<body>
    <nav>
        <ul>
            <li>Home</li>
            <li>About</li>
            <li>Gallery</li>
            <li>Contact</li>
        </ul>
    </nav>

</body>
</html>
```

It might look something like that. This will get loaded regardless of what we want though.

If you imagine a blog situation, you might want a navigation that is on every page - this goes on the layout.erb page. But you might also want to show the actual post as well. This is what ERB is great at! You pass in a symbol (the name of the file you'd like to load in) to the ERB method and it goes and grabs it for you.

```
get '/post' do
    erb :post
end
```

If you just did this though, it would get the file but wouldn't know where to put the contents of that particular file in the layout! As such, it will only show the layout.erb file. We use the yield method to solve this problem! This grabs the contents of the file requested and puts it straight over itself.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>

    <%= yield %>

</body>
</html>
```

For more information about Sinatra, see here.

Here is your homework!

Week 04, Day 04

What we covered today:

- Warmup
  - Solution
- Demos
- SQL (Specific Query Language)
  - Slides
- CRUD
- Butterfly Lab

SQL

This is a really difficult topic and not one that we expect you to be able to write out - as long as you can get it in terms of principles - it is all good.

When we talk about tables and databases, there is really only 4 tasks that we need to do.

- **C**reate
- **R**ead
- **U**pdate
- **D**elete

This is called CRUD.

Involved in every database, there are a couple of things. We have:

- The Database itself
- Individual tables
- Individual records on tables

Before we can do any actual action we need to create the database to work with...

```
CREATE TABLE table_name (
    -- Comma seperated list of attributes with a type and a list of options
    column_name DATATYPE
);

CREATE TABLE person (
    id INTEGER PRIMARY KEY,
    first_name TEXT,
    last_name TEXT,
    age INTEGER
);
```

This normally goes in a file with a file name ending with .sql. You might call this person.sql, and obviously we can have lots of these. As many as you want.

This hasn't created the database though, we need to be explicit about that.

`sqlite3 desired_database_name.db < add_this_table.sql`

`sqlite3 database.db < person.sql`

This line will create the database.db file if necessary, and if not - it will just add whatever is defined in the .sql file specified. It imports the details from the .sql into the database.db.

To make sure this has worked, type in `sqlite3 database.db` and hit enter in the terminal. This will open up a direct line to the database in the current folder. If you type `.schema` it will show the current tables.

CREATE STEP

Once we have the table defined, we need to figure out how to actually put records into it.

```
INSERT INTO table_name ( comma, seperated, columns ) VALUES ( commas_value, seperated_
value, column_value);

INSERT INTO person (id, first_name, last_name, age) VALUES ( 0, "Zed", "Shaw", 37 );

-- We don't need to tell the attributes though, it can look just like this...

INSERT INTO person VALUES (0, "Zed", "Shaw", 37);
```

This is the creation step. If you wrote this in a file, we can import that SQL into the database - `sqlite3 database_name.db < insert_stuff.sql`

READ STEP

This is pretty annoying to write.

```
-- SELECT what FROM what_table;
-- SELECT what FROM what_table WHERE options;

SELECT * FROM person; -- this will select all attributes and all records from the pers
on database

SELECT name FROM person; -- only show the name attributes

SELECT * FROM person WHERE first_name == "Zed"; -- show all attributes from records in
 the person database where the first_name is "Zed"
```

UPDATE STEP

And this is pretty annoying to write.

```
UPDATE table SET attribute_name = attribute_value WHERE attribute_name = attribute_val
ue;

UPDATE person SET first_name = "WOLF" WHERE first_name = "Zed";
```

DELETE STEP

AND this is pretty annoying to write.

```
DELETE FROM what_table WHERE what_attributes = what_value;

DELETE FROM person WHERE first_name = "Zed"; -- Delete all records in the person table
 where the first_name is equal to "Zed"
```

That is the basics of SQL, for more - see here. It's all about the principles though, as long as you understand the fact that you need a database to have tables, and tables to have records - that is all good.

In terms of actual structure of an application, this is the structure of a CRUD application. 7 views for all of this! The #new and the #edit are just ways to show the actual form.

	VERBS	URLS	SQL	NAME
CREATE	POST	/butterflies	INSERT	#create
		/butterflies/new		#new
READ	GET	/butterflies	SELECT	#index
	GET	/butterflies/:id	SELECT	#show
UPDATE	POST	/butterflies/:id	UPDATE	#update
		/butterflies/:id/edit		#edit
DELETE	(Delete)	/butterflies/:id	DELETE	#delete

CRUD is the foundation of most applications on the web, it is the thing that powers it! Important to get the principles of it.

Here is the lab we did. HOMEWORK!

Week 04, Day 05

What we covered today:

Object Oriented Programming (OOP)

Brief Intro to OOP

Basically, OOP is an approach to development that tries to replicate real life. It is pretty much always done using objects or classes as namespaces and treats them as a way to make your code "modular".

Small, clean methods are also a major part of it.

Basic OOP in Ruby

Up until now, we have only been modifying objects. We can create an empty one and then add methods and features to construct a particular one.

```
o = {}
def o.silly_method
    puts "I'm silly!"
end
```

This is not ideal. It means we always have to rewrite a whole bunch of code.

To get around this we use classes!

Classes

Everything in Ruby inherits from a class (is an instance of a class) and in some capacity inherit from Object. This doesn't pop up that regularly, but to see what I mean...

```
{}.class
[].class
"".class
# etc.

# If you want to see everything a data type inherits from...
{}.class.ancestors
[].class.ancestors
"".class.ancestors
```

Treat classes as a factory or a blueprint, something that gives another thing all the details that it needs. We use them to stop duplicate code, make manageable, easy-to-debug code.

What do they look like?

```
class Person
end
```

The Real Power...

...Comes from being able to add methods! We can encapsulate functionality with classes.

```
class Person
    def speak
    end

    def laugh
    end
end
```

Unlike javascript though, we can't call the functions or methods on the class itself. This is how we could do it in javascript...

```
var Person = {
    talk: function () {
        console.log( "I can talk!" );
    }
}

Person.talk();
```

In Ruby, we need to create an instance of the class. Think of the class itself being a blueprint, and the instance being the house that is built from it (it is the thing with all the power and functionality).

```
class Person
    def speak
        puts "Speak"
    end

    def laugh
    end
end

person = Person.new
person.speak # Will work!
```

Obviously, though. We want to be able to store something on the person themselves! Maybe we want to know their name, age or gender for example.

We use what are called "getters" and "setters" to do this.

```
class Person
    def name=(name)
        @name = name
    end

    def name
        @name
    end

    def age=(age)
        @age = age
    end

    def age
        @age
    end
end

person = Person.new
person.name=( "Jane" )
person.age=( 80 )

person.name # Will return "Jane"
person.age # Will return 80
```

Surely, you can all see the duplication though. There must be an easier way! There is.

```ruby
class Person
    attr_accessor :name, :age
end

person = Person.new
person.name = "Serge"
person.name # Will return "Serge"

person.age = 100
person.age # Will return 100
```

But it is a hassle to define each one of these like this. We are lucky to have a solution for this.

On a creation of a new instance in Ruby, when the .new method is called, it will also call automatically an initialize method.

```ruby
class Person

    attr_accessor :name, :age

    def initialize( name, age )
        @name = name
        @age  = age
    end

end
```

This will do everything that we were doing before.

We can add heaps of methods and see what we have to work with as well.

```
class Person
    def method_one
    end
    def method_two
    end
    def method_three
    end
end

person = Person.new

# To access all the methods:

person.class.instance_methods

# If you pass in the value false to the instance methods method, it will only show you
 the methods you have defined.

person.class.instance_methods( false )
```

Active Record

Preface - Object Relational Mapping

One of the most signifcant principles in Object-Oriented Programming is the idea of rich objects - things that store data, and allow it to be retrieved in logical and concise ways. This pattern is what Active Record strives for. They call it Object Relational Mapping, or ORM. It's basic principle is that rich objects in your application should be connected with database tables. Using ORM, the properties and relationships of the objects in an application can be easily stored and retrieved from a database without writing SQL statements directly and with less overall database access code. Also is far more secure due to lessened risks in regards to SQL Injections.

What does Active Record do?

Active Record, as an ORM, gives us several mechanisms, the most important being the ability to:

- Represent models and their data.
- Represent associations between these models.
- Represent inheritance hierarchies through related models.
- Validate models before they get persisted to the database.
- Perform database operations in an object-oriented fashion.

Convention over Configuration

Convention over configuration is big in development anyway, but it is particularly full on when it comes to Active Record. That is because it uses the names to sort out assosciations etc. Make sure you follow these rules!!

- **Database Tables** - Plural with underscores seperating words (articles, line_items etc.)
- **Model / Class Names** - Singular with the first letter of each word capitalized (Article, LineItem etc.)

How do you work with Active Record?

Well, at the end of the day, Active Record is a gem like any other. So we need to require it!!

```
require 'active_record' # make sure this is at the top of the file!
```

After requiring it, we also need to actually establish the connection to the database. These are annoying lines that will be done automatically in Rails! Just copy and paste them.

```
# Sets up our connection to the database.db we have created
ActiveRecord::Base.establish_connection(
  :adapter  => 'sqlite3',
  :database => 'database.db'
)

ActiveRecord::Base.logger = Logger.new(STDERR) # Logs out the Active Record generated SQL in the terminal. This isn't necessary but very helpful and cool to see what it is actually running
```

Brief Interlude of Random Stuff

	SQL	HTTP Verbs	Active Record
Create	INSERT	(Put) or POST	.create or .new/.save
Read	SELECT	GET	.find or .find_by
Update	UPDATE	(Patch) or POST	.update or .find/.save
Delete	DELETE	(Delete) or POST or GET	.destroy or .destroy_all

CRUD with Active Record

Once we have our models defined (i.e. our classes), we can get into the CRUD stuff. There are always a lot of options of how to do this!

Create

```
plant = Plant.new
plant.name = "Hibiscus"
plant.flowers = true
plant.save # YOU MUST RUN THIS LINE! (When using the new approach)

# OR

plant.create( :name => "Hibiscus", :flowers => true )

# OR

user = User.new do |u|
  u.name = "David"
  u.occupation = "Code Artist"
end # THIS WILL RUN THE SAVE AUTOMATICALLY
```

Read

```
Plant.all
Plant.first
Plant.last
Plant.find( 10 ) # Find with an ID
Plant.find_by( :name => "Hibiscus" ) # Returns the first plant that this works with
Plant.where( :name => "Hibiscus" ) # Returns all instances where this is appropriate
```

Update

```
plant = Plant.find_by( :name => "Hibiscus" )
plant.name = "Hibiscus 2"
plant.save

plant = Plant.find_by( :name => 'Hibiscus 2' )
plant.update( :name => 'Hibiscus' ) # This will save automatically
```

Delete

```
plant = Plant.find_by( :name => 'Hibiscus' )
plant.destroy

Plant.destroy_all
```

Homework!

Week 05

Week 05, Day 01

What we covered today:

- Warm up!
  - solution
- Demos
- Rails
  - Intro
  - CRUD
  - Helpers & form helpers
- Homework
  - Getting Started with Rails
  - Games on Rails

Rails

History of Rails

David Heinemeier Hansson extracted Ruby on Rails from his work on the project management tool Basecamp at the web application company also called Basecamp. He released it as open source in July 2004 (but didn't give out commit rights until February 2005).

Obviously there is heaps more, see here for it

What is it?

Rails is a web application framework, built with Ruby. It's a similar approach to a Javascript library with Javascript itself. It is designed to make programming all web applications much, much easier by making lots of assumptions about what you are going to need to get started. It means you have to write a lot less code, while building a lot more. Similar to Ruby, it makes development fun.

Rails is very opinionated (the same as its creator), it makes an assumption there is a 'best' way to do it, and it encourages that way (but still allows for flexibility). If you follow the 'Rails Way', it will be much easier.

The Rails philosophy includes three major guiding principles, which really just stem from Ruby itself (except the MVC pattern):

- *Don't Repeat Yourself (D.R.Y)* - DRY is a principle of software development, not

exclusive to Ruby (though Ruby is very good at it, as is its community). It's main thing is that "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system". That makes it sound a lot more complex than it really is. All it means is that by not writing the same code over and over again, our code is more maintable, extensible and less buggy.

- *Convention Over Configuration (C.O.C)* - Rails, as previously mentioned, is very opinionated. If you don't follow its guidelines (naming conventions etc.), it is difficult. But if you do, the set up is very efficient. It has the capacity to do most things by default as long as you follow the right approach.
- *Model, View and Controller (M.V.C)* - This is probably the most important. Rails structures everything like this, so it is important to understand this. MVC is a software architecture pattern that breaks the code into small manageable chunks and stems from the problem when trying to modularize a user interface functionality so that it is maintanable and extensible. Those chunks are Models, Views and Controllers as you have probably guessed. But what are they?
    - Models - The model manages the behavior and data of the application domain, also where the database classes are created.
    - Views - The view manages the display of information.
    - Controllers - The controller interprets the user behaviour (it is like the switchboard)
    - An easy way to understand MVC: the model is the data, the view is the window on the screen, and the controller is the glue between the two. -- ConnellyBarnes
    - But is all about the interactions. A controller can send commands to the model to update the model's state (e.g., editing a document). It can also send commands to the associated view to change the presentation (and content). A model stores data that is retrieved by the controller and displayed in the view. Whenever there is a change to the data it is updated by the controller. A view requests information from the model that it uses to generate an output representation to the user.

How to install it?

Simple! `gem install rails`, make sure you don't add sudo at the start! Obviously this needs to have Ruby as well, if you don't have that - see [here.(../week_04/wk04_day01.md). It takes a long time to load everything in Rails.

Let's create a new Rails project

First step is always planning, before you get into a Rails project - it is very important to map out the way that a user will be using your site. Once you have figured that out, you can begin creating the project.

To create the Rails project, we use the `rails new project_name` command in terminal. This takes a lot of options but for now we will just stick with that. We need to then move into our project (`cd project_name`).

After this, we alter our Gemfile to be more suited for debugging. The code we tend to use is this...

```
group :development do
  gem 'pry-rails'
  gem 'pry-stack_explorer'
  gem 'annotate'
  gem 'quiet_assets'
  gem 'better_errors'
  gem 'binding_of_caller'
  gem 'meta_request'
end
```

We then run `bundle` in our terminal, that will sort out all our gems and get all the dependencies loaded which is quite important. The Gemfile is a more complex and extensible way of doing requires at the top of a Sinatra app.

Now we get into the Routes (found in config/routes.rb). This is like our phone directory or the 'gets' in Sinatra. There are a million different ways of approaching this. But at its most basic it will look something like the following.

```
Rails.application.routes.draw do
  # controller#method
  root :to => 'pages#home'
  # route      controller#home
  get '/home' => 'pages#home'
  # DYNAMIC ROUTES WITH VARIABLE BITS IN PARAMS (JUST LIKE SINATRA)
  get '/auto/:color' => 'auto#color'
end
```

Lets load up the server - `rails server` or `rails s`. If we go to `localhost:3000`, you can see an error. We are up to an Error Driven Development point now, so if we load the root page (which is pages#home), we let the errors guide where we go next.

It will first say that there is an uninitialized Constant. This means that we haven't created the associated controller. In our routes, for our root page we have said that it wants "pages#home". This says we need a Pages Controller and a home method within it. So within the app/controller folder, we need to create a file called pages_controller.rb. All our controllers inherit from ApplicationController - this is what gives it all of its functionality. Within the PagesController, at its most basic, it is going to look something like this...

```
class PagesController < ApplicationController
    def home
    end
end
```

Once we have this setup, we can need to create our views (the error is missing template). We need to create a pages folder in our app/views folder, and within that we need a home.html.erb. Once we have that, it will be required by default for us through the method (that's why naming conventions are so important with Rails).

This is the sort of setup you need to go through for each particular page in your application (there are many ways to generate them automatically though). Obviously, you can do it a million times if necessary!

The Basic Order of Things (without a database)

- Planning
- `rails new project_name`
- Edit Gemfile for debugging
- Work out your routes - config/routes.rb
- Create controllers
- Create methods in controllers
- Create appropriate views
- Repeat as necessary

Have a crack at this.

Naming Conventions

Models are singular, with a capitalized class name, located in app/models, have an extension of .rb, and inherit from ActiveRecord::Base.

Controllers are plural, with a CamelCase class name, located in app/controllers, have an extension of .rb, and inherit from ApplicationController.

Views are a bit more difficult. They reside in a folder in app/views, the name of that folder comes from the associated controller and is plural. The name of the actual file is the name of the associated action (or method), plus .html.erb. They don't inherit from anything. This may make things clearer...

Type	File Name	Location	Class Name	Inher
Model	planet.rb	app/models	Planet	ActiveRe
Controllers	planets_controller.rb	app/controllers	PlanetsController	Applicati
Views	index.html.erb	app/views/planets	N/A	N/A

The Power of Rails in Terminal

There are a few commands that are absolutely essential for Rails development. The more you know them, the better it is!

At its most basic:

- `rails new` - Generate a new application
- `rails server` - Runs the server
- `rails generate` - Generate a whole heap of things within an application
- `rails console` - Opens up a console
- `rails dbconsole` - Opens up a direct connection to SQL
- `rake` - Does thousands of things
- `bundle` - Install gems and their dependencies

Running any command with `-h` or `--help` at the end will show you the documentation for that particular command. But the real power comes from...

Customization and Automation!

RAILS NEW

```
rails new app_name
rails new app_name -T # Skips the Test Suite
rails new app_name --database=postgresql # Specifies the Database (changes it from sql
ite3)
rails new app_name -d postgresql
```

RAILS SERVER

```
rails server
rails s # Shorthand for rails server
rails server -p 3001 # Specifies another port (have multiple servers at once!)
rails server -e production # Changes the state of the application (different gem sets
etc. - don't worry about this one)
```

RAILS GENERATE

```
rails generate controller ControllerName list of actions
rails generate controller Greetings index create
rails g controller Greetings index create
# THIS WILL CREATE VIEWS, JS, CSS AND ACTIONS IN CONTROLLERS (PLUS TESTS)

rails g model ModelName field:type
rails g model Painting name:string year:date
# THIS WILL CREATE MODELS, MIGRATIONS, AND TESTS

rails g scaffold ModelName field:type field:type
rails generate scaffold Painting name:string year:date
# THIS WILL CREATE EVERYTHING
```

RAILS CONSOLE

```
rails console # OPENS UP YOUR RAILS APP IN PRY OR IRB
rails c # SHORTHAND

rails console staging # OPENS UP A SPECIFIC ENVIRONMENT

rails console --sandbox # CAN'T MAKE ANY ACTUAL CHANGES
```

RAILS DBCONSOLE

```
rails dbconsole # OPENS UP A DIRECT CONNECTION TO YOUR DATABAS
rails db # SHORTHAND
```

BUNDLE

```
bundle install # INSTALLS GEM AND DEPENDENCIES
bundle # SHORTHAND
```

RAKE

This is crazy powerful and does a million things, but here are some of the more important ones that you might need to know. We will go into this in a lot more detail though!

```
rake --tasks # Lists everything it can do

rake about # Lists everything about your Rails app

rake db:drop # DROPS THE DATABASE
rake db:create # CREATES THE DATABASE
rake db:migrate # MIGRATES TABLES INTO THE DATABASE (FROM db/migrations)
rake db:rollback # GOES BACK ONE STEP IN THE DATABASE (BACK ONE MIGRATION)

rake routes # LIST ALL OF YOUR ROUTES

rake stats # LINES OF CODE ETC.

rake notes # SEE HERE - http://guides.rubyonrails.org/command_line.html#notes
```

Form Helpers

At its most basic...

```
<%= form_tag("/search", method: "get") do %>
  <%= label_tag(:q, "Search for:") %>
  <%= text_field_tag(:q) %>
  <%= submit_tag("Search") %>
<% end %>
```

Binding a form to an object...

```
# OUR CONTROLLER
def new
  @article = Article.new
end
```

```
<!-- OUR ASSOCIATED VIEW -->

<%= form_for @article, url: {action: "create"}, html: {class: "nifty_form"} do |f| %>
  <%= f.text_field :title %>
  <%= f.text_area :body, size: "60x12" %>
  <%= f.submit "Create" %>
<% end %>
```

TAG HELPERS!

```
<%= radio_button_tag(:age, "adult") %>
<%= label_tag(:age_adult, "I'm over 21") %>
<%= text_area_tag(:message, "Hi, nice site", size: "24x6") %>
<%= password_field_tag(:password) %>
<%= hidden_field_tag(:parent_id, "5") %>
<%= search_field(:user, :name) %>
<%= telephone_field(:user, :phone) %>
<%= date_field(:user, :born_on) %>
<%= datetime_field(:user, :meeting_time) %>
<%= datetime_local_field(:user, :graduation_day) %>
<%= month_field(:user, :birthday_month) %>
<%= week_field(:user, :birthday_week) %>
<%= url_field(:user, :homepage) %>
<%= email_field(:user, :address) %>
<%= color_field(:user, :favorite_color) %>
<%= time_field(:task, :started_at) %>
<%= number_field(:product, :price, in: 1.0..20.0, step: 0.5) %>
<%= range_field(:product, :discount, in: 1..100) %>
<%= select_tag(:city_id, '<option value="1">Lisbon</option>...') %>
<% cities_array = City.all.map { |city| [city.name, city.id] } %>
<%= options_for_select(cities_array) %>
<%= time_zone_select(:person, :time_zone) %>
<%= select_date Date.today, prefix: :start_date %>
```

Thousands of things you can do, go through here!

HOMEWORK

- https://gist.github.com/ga-wolf/cbcfaf16c5e675eef6b8ae6a2191b169

Week 05, Day 02

What we covered today:

Helpers

These are things that you should use over and over, it is a collection of commonly used patterns. They can only be used in our views!!! Remember that these obviously need to be wrapped in ERB tags.

Number Helpers

```
number_to_currency( value )
number_to_human( value )
number_to_phone( value, options )
number_to_phone( value, :area_code => true )
```

Here are all of the number helpers.

Text Helpers

```
pluralize( value, 'singular_case' )
pluralize( @person_count, 'person' )

truncate( value, options )
truncate( @story, :length => 50 )

cycle( list_of_values )
cycle( 'red', 'green', 'orange', 'purple' )
```

Here are all of the text helpers.

Assets Helpers

```
image_tag( 'path', options )
image_tag( 'funny.jpg' )
image_tag( 'http://fillmurray.com/500/500', :class => "oh-bill" )
```

Here are all of the asset helpers.

URL Helpers

```
link_to( 'Home', root_path )
link_to( 'Work Path Show', work_path( work.id ) )
link_to( 'Work Path Show', work_path( work ) )
link_to( 'Work Path Show', work )
button_to( 'Test Path', root_path, :method => 'GET' )
```

Here are all of the url helpers.

The Power of Rails in Terminal

There are a few commands that are absolutely essential for Rails development. The more you know them, the better it is!

At its most basic:

- `rails new` - Generate a new application
- `rails server` - Runs the server
- `rails generate` - Generate a whole heap of things within an application
- `rails console` - Opens up a console
- `rails dbconsole` - Opens up a direct connection to SQL
- `rake` - Does thousands of things
- `bundle` - Install gems and their dependencies

Running any command with `-h` or `--help` at the end will show you the documentation for that particular command. But the real power comes from...

Customization and Automation!

RAILS NEW

```
rails new app_name
rails new app_name -T # Skips the Test Suite
rails new app_name --database=postgresql # Specifies the Database (changes it from sql
ite3)
rails new app_name -d postgresql
```

RAILS SERVER

```
rails server
rails s # Shorthand for rails server
rails server -p 3001 # Specifies another port (have multiple servers at once!)
rails server -e production # Changes the state of the application (different gem sets
etc. - don't worry about this one)
```

RAILS GENERATE

```
rails generate controller ControllerName list of actions
rails generate controller Greetings index create
rails g controller Greetings index create
# THIS WILL CREATE VIEWS, JS, CSS AND ACTIONS IN CONTROLLERS (PLUS TESTS)

rails g model ModelName field:type
rails g model Painting name:string year:date
# THIS WILL CREATE MODELS, MIGRATIONS, AND TESTS

rails g scaffold ModelName field:type field:type
rails generate scaffold Painting name:string year:date
# THIS WILL CREATE EVERYTHING
```

RAILS CONSOLE

```
rails console # OPENS UP YOUR RAILS APP IN PRY OR IRB
rails c # SHORTHAND

rails console staging # OPENS UP A SPECIFIC ENVIRONMENT

rails console --sandbox # CAN'T MAKE ANY ACTUAL CHANGES
```

RAILS DBCONSOLE

```
rails dbconsole # OPENS UP A DIRECT CONNECTION TO YOUR DATABAS
rails db # SHORTHAND
```

BUNDLE

```
bundle install # INSTALLS GEM AND DEPENDENCIES
bundle # SHORTHAND
```

RAKE

This is crazy powerful and does a million things, but here are some of the more important ones that you might need to know. We will go into this in a lot more detail though!

```
rake --tasks # Lists everything it can do

rake about # Lists everything about your Rails app

rake db:drop # DROPS THE DATABASE
rake db:create # CREATES THE DATABASE
rake db:migrate # MIGRATES TABLES INTO THE DATABASE (FROM db/migrations)
rake db:rollback # GOES BACK ONE STEP IN THE DATABASE (BACK ONE MIGRATION)

rake routes # LIST ALL OF YOUR ROUTES

rake stats # LINES OF CODE ETC.

rake notes # SEE HERE - http://guides.rubyonrails.org/command_line.html#notes
```

Form Helpers

At its most basic...

```
<%= form_tag("/search", method: "get") do %>
  <%= label_tag(:q, "Search for:") %>
  <%= text_field_tag(:q) %>
  <%= submit_tag("Search") %>
<% end %>
```

Binding a form to an object...

```
# OUR CONTROLLER
def new
  @article = Article.new
end
```

```
<!-- OUR ASSOCIATED VIEW -->

<%= form_for @article, url: {action: "create"}, html: {class: "nifty_form"} do |f| %>
  <%= f.text_field :title %>
  <%= f.text_area :body, size: "60x12" %>
  <%= f.submit "Create" %>
<% end %>
```

TAG HELPERS!

```
<%= radio_button_tag(:age, "adult") %>
<%= label_tag(:age_adult, "I'm over 21") %>
<%= text_area_tag(:message, "Hi, nice site", size: "24x6") %>
<%= password_field_tag(:password) %>
<%= hidden_field_tag(:parent_id, "5") %>
<%= search_field(:user, :name) %>
<%= telephone_field(:user, :phone) %>
<%= date_field(:user, :born_on) %>
<%= datetime_field(:user, :meeting_time) %>
<%= datetime_local_field(:user, :graduation_day) %>
<%= month_field(:user, :birthday_month) %>
<%= week_field(:user, :birthday_week) %>
<%= url_field(:user, :homepage) %>
<%= email_field(:user, :address) %>
<%= color_field(:user, :favorite_color) %>
<%= time_field(:task, :started_at) %>
<%= number_field(:product, :price, in: 1.0..20.0, step: 0.5) %>
<%= range_field(:product, :discount, in: 1..100) %>
<%= select_tag(:city_id, '<option value="1">Lisbon</option>...') %>
<% cities_array = City.all.map { |city| [city.name, city.id] } %>
<%= options_for_select(cities_array) %>
<%= time_zone_select(:person, :time_zone) %>
<%= select_date Date.today, prefix: :start_date %>
```

Thousands of things you can do, go through here!

A Basic Rails Guide

Treat this as a really rough guide, definitely don't always follow it. You'll figure out your approach soon.

- `rails new app-name`
- `cd app-name`
- Add `gem 'pry-rails'` into your Gemfile (in the development group)
- run `bundle`
- run `rake db:create`
- Generate your first migrations and models - either `rails g model Model name:string type:string` etc. or:
  - `rails g migration create_tables`
  - Add the fields you need to that file - make sure you include t.timestamps
  - `rails g model ModelName`
- Generate your controllers and your views - either `rails g controller Users index new create delete show` etc. or:
  - `rails g controller Users`
  - Add your methods into the controller

    - Create views that correspond with the method names in the view folder for that particular controller
- Work out your routes file
- Repeat this stuff as necessary

Week 05, Day 03

What we covered today:

- Warmup!
  - Solution
- Helpers
- TUNR
  - Has and Belongs to Many
  - Has Many Through

Helpers

These are things that you should use over and over, it is a collection of commonly used patterns. They can only be used in our views! Remember that these obviously need to be wrapped in ERB tags.

Number Helpers

```
number_to_currency( value )
number_to_human( value )
number_to_phone( value, options )
number_to_phone( value, :area_code => true )
```

Here are all of the number helpers.

Text Helpers

```
pluralize( value, 'singular_case' )
pluralize( @person_count, 'person' )

truncate( value, options )
truncate( @story, :length => 50 )

cycle( list_of_values )
cycle( 'red', 'green', 'orange', 'purple' )
```

Here are all of the text helpers.

Assets Helpers

```
image_tag( 'path', options )
image_tag( 'funny.jpg' )
image_tag( 'http://fillmurray.com/500/500', :class => "oh-bill" )
```

Here are all of the asset helpers.

URL Helpers

```
link_to( 'Home', root_path )
link_to( 'Work Path Show', work_path( work.id ) )
link_to( 'Work Path Show', work_path( work ) )
link_to( 'Work Path Show', work )
button_to( 'Test Path', root_path, :method => 'GET' )
```

Here are all of the url helpers.

Associations!

We need associations to make common tasks easier and more readable. In Rails, an association is a connection between two models that inherit from ActiveRecord::Base (called Active Record models for understandable reasons). There are six types of associations and we will briefly look at them now.

- belongs_to
- has_one
- has_many
- has_many :through
- has_one :through
- has_and_belongs_to_many

For a far deeper dive though, see here.

Belongs To

A belongs_to association sets up a one-to-one connection with another model, such that each instance of the declaring model "belongs to" one instance of the other model. This is often used in conjunction with has_one or has_many. Remember that belongs_to associations must be in the singular term!

```
class Order < ActiveRecord::Base
    belongs_to :customer
end
```

Has One

A has_one association also sets up a one-to-one connection with another model, but with somewhat different semantics (and consequences). This association indicates that each instance of a model contains or possesses one instance of another model.

```
class User < ActiveRecord::Base
  has_one :account
end
```

Has Many

A has_many association indicates a one-to-many connection with another model. You'll often find this association on the "other side" of a belongs_to association. This association indicates that each instance of the model has zero or more instances of another model. Note that the name of the other model is pluralized when declaring a has_many association.

```
class User < ActiveRecord::Base
  has_many :orders
end
```

Has Many Through

A has_many :through association is often used to set up a many-to-many connection with another model. This association indicates that the declaring model can be matched with zero or more instances of another model by proceeding through a third model.

```
class Doctor < ActiveRecord::Base
  has_many :appointments
  has_many :patients, through: :appointments
end

class Appointment < ActiveRecord::Base
  belongs_to :doctor
  belongs_to :patient
end

class Patient < ActiveRecord::Base
  has_many :appointments
  has_many :doctors, through: :appointments
end
```

Has One Through

A has_one :through association sets up a one-to-one connection with another model. This association indicates that the declaring model can be matched with one instance of another model by proceeding through a third model.

```
class Supplier < ActiveRecord::Base
  has_one :account
  has_one :account_history, through: :account
end

class Account < ActiveRecord::Base
  belongs_to :supplier
  has_one :account_history
end

class AccountHistory < ActiveRecord::Base
  belongs_to :account
end
```

Has And Belongs To Many

A has_and_belongs_to_many association creates a direct many-to-many connection with another model, with no intervening model.

```
class Assembly < ActiveRecord::Base
  has_and_belongs_to_many :parts
end

class Part < ActiveRecord::Base
  has_and_belongs_to_many :assemblies
end
```

Homework!

This

Week 05, Day 04

What we covered today:

- Warmup
  - Solution
- TUNR
  - Sessions
  - Authentication
  - Validations

Authentications!

Authentication is rough, and rarely (if ever) will you be asked to do it for yourself. There are plenty of gems that will do that sort of thing, but if you want to do it for yourself - there are a bunch of things you need to do.

First things first, we don't store passwords in plain text, we store it in something called a password_digest. This is something that has been encrypted.

Let's create our User model!!

```
rails generate model User name:string password_digest:string age:integer email:string
# That will create our model and our migrations

rake db:migrate # This will create our User model
```

Now, we need to use our Gemfile to get something that does the encrypting. We normally use bcrypt for this. Let's add it!

```
gem 'bcrypt'
```

In our User model, we need to add the following line. Make sure it goes before the end!

```
has_secure_password
```

In our forms, we still need to do reference :password and :password_confirmation. But it will sort everything out behind the scenes for us!

This is quite a good description!

Validations

As with virtually every problem in the universe, Active Record has a solution. Instead of trying to create by hand validations for data that is getting submitted to a database, let us work with Active Record and find out whether it has the solution (and it will).

These validations go within the model that you are trying to validate against (for example user.rb).

Why should we use them?

Because the internet is a scary, scary world and there are scary, scary things that people will try and put in your database.

When do validations take place?

Validations occur when any of the following methods take place:

- create
- create!
- save
- save!
- update
- update!

If you want to skip validations, they won't take place when using the following methods.

- decrement!
- decrement_counter
- increment!
- increment_counter
- toggle!
- touch
- update_all
- update_attribute
- update_column
- update_columns
- update_counters

You can also pass `:validate => false` to the save method.

How to test validity?

There are methods!

- valid?
- invalid?

If a model is invalid, you can access the errors easily as well. That is in the `.errors.messages`.

Alright, enough of this. Now for some Validation Helpers

```
class User < ActiveRecord::Base
  has_many :books
  validates :column_name, :key => value
end
```

acceptance

This method validates that a checkbox on the user interface was checked when a form was submitted. This is how terms of service are accepted for example.

```
validates :terms_of_service, acceptance: true

# It defaults to "1", if you want to accept something else

validates :terms_of_service, acceptance = { :accept => "yes" }
```

validates_associated

If you have associations, you can validate those associations by using validate associated.

```
validates_associated :column_name
```

Don't do this on both sides of the associations! It will cause an infinite loop.

confirmation

You should use this helper if you have two text fields that should receive exactly the same content. This validation creates a virtual attribute whose name is the name of the field but with "_confirmation" at the end.

```
validates :email, :confirmation => true

# This should always work with a nil check as well

validates :email_confirmation, :presence => true
```

exclusion

This helper validates that the attributes' values are not included in a given set. It requires an :in option that receives the set of values that cannot be included, as well as an optional :message option to show how you can include the attribute's value.

```
validates :subdomain, :exclusion => {
                        :in => [ "www", "us", "ca", "jp" ],
                        :message => "%{ value } is reserved. "
                      }

# Notice the interpolation with value, that takes the excluded value
```

format

This helper validates the attributes' values by testing whether they match a given regular expression, which is specified using the :with option.

```
validates :legacy_code, format: {
                          :with => /\A[a-zA-Z]+\z/,
                          :message => "only allows letters"
                        }
```

Alternatively, you can require that the specified attribute does not match the regular expression by using the :without option.

length

This helper validates the length of the attributes' values. It provides a variety of options, so you can specify length constraints in different ways:

```
validates :name, length: { minimum: 2 }
validates :bio, length: { maximum: 500 }
validates :password, length: { in: 6..20 }
validates :registration_number, length: { is: 6 }
```

numericality

This helper works with numericality.

```
validates :count, numericality: { :only_integer => true }
```

Besides :only_integer, you can also work with the following:

- :greater_than - Specifies the value must be greater than the supplied value. The default error message for this option is "must be greater than %{count}".

- :greater_than_or_equal_to - Specifies the value must be greater than or equal to the supplied value. The default error message for this option is "must be greater than or equal to %{count}".
- :equal_to - Specifies the value must be equal to the supplied value. The default error message for this option is "must be equal to %{count}".
- :less_than - Specifies the value must be less than the supplied value. The default error message for this option is "must be less than %{count}".
- :less_than_or_equal_to - Specifies the value must be less than or equal the supplied value. The default error message for this option is "must be less than or equal to %{count}".
- :odd - Specifies the value must be an odd number if set to true. The default error message for this option is "must be odd".
- :even - Specifies the value must be an even number if set to true. The default error message for this option is "must be even".

presence

Checks whether the value of a particular set of attributes is nil or an empty string.

```
validates :email, :presence => true
```

absence

Does the opposite or presence, makes sure it is not there.

```
validates :email, :absence => true
```

uniqueness

Makes sure there is only one of these columns with this particular value.

```
validates :email, uniqueness: true
```

Working with Validation Errors

Taking an invalid person as example...

```
person.valid? # Returns false

person.errors # Returns an instance of the class ActiveModel::Errors containing all er
rors. Each key is the attribute name and the value is an array of strings with all err
ors.

person.errors.messages # Returns all of the messages associated
```

There are plenty more ways to validate things, so see here!

HOMEWORK

HERE

Week 05, Day 05

What we covered today:

- Warmup!
  - solution
- Install PostgreSQL
- LolCommits
- Project Brief

PostgreSQL Installation

First things first. Download the application from here. Once the download is complete, double click the ZIP file and run through the installation. Make sure you drag the Postgres.app file into your Applications folder!

After that, open up the Postgres.app, you will know that it has worked when you see an elephant in the top bar of your computer.

Now, this is the more difficult bit. Open up your .bash_profile with Sublime - `subl ~/.bash_profile` should do the trick, and copy this line of code into there `export PATH=$PATH:/Applications/Postgres.app/Contents/Versions/9.4/bin` . Save that file and open up a new terminal window. If typing `postgres` doesn't throw an undefined error, you are all good!

Remember that we should always use PostgreSQL for your Rails applications, it is far more scalable, pretty much just as easy to set up and works on Heroku (where we will be hosting our applications). To start a Rails application with postgres - `rails new app_name --database=postgresql` or `rails new app_name -d postgresql` - should do the trick.

Working with Rails and PostgreSQL

Create your application - `rails new app_name -d postgresql`

Find out your username with `whoami`

Open up your database.yml file and add the following lines into the development bit...

```
host: localhost
username: the_result_of_the_whoami_command
```

Now run `rake db:create`

You can get into your migrations now!

To check that your migrations have worked, run `rails dbconsole` or `rails db`, and then type in `\d table_name;` (eg. `\d songs;`). This will show you the schema for it.

`\q` - will exit the dbconsole!

Then run `annotate`

LolCommits

For some reason, people love this thing... See here.

Run `brew install imagemagick` .

Then run `gem install lolcommits` .

To enable lolcommits within one git repo - `lolcommits --enable`

If you would like this to apply to all git repos on your local computer, see here.

Week 07

Week 07, Day 01

What we covered today:

- Warmup
  - Solution
- TDD
- Callbacks
- AJAX / XHR
- Movies III - Army of Darkness

Testing

Testing is huge in the Ruby and Rails communities. It came out with Rails version 1 and has been a big part ever since. Following the principles of T.D.D (Test Driven Development) is completely vital for large apps for tiny changes can change everything (as you all now know). It is also vital because your final project needs to have 100% test coverage.

You normally follow the "Red-Green-Refactor Cycle". Write the tests, let them fail first, make them pass, then refactor if necessary.

The most common testing suite for Ruby is Rspec.

Install that! `gem install rspec` .

Let's create a new project. `mdkir tdd-bank && cd tdd-bank` . In a non-Rails environment, i.e. pure Ruby, we need to do a few things to get it the testing suite working. Run `rspec --init` . This will make a spec folder and a .rspec file for us. The .rspec has all our configurations and the spec_helper is the thing that kicks everything off for us.

So, what does it actually look like?

Let's make a bank_spec.rb file to show how this all works.

```
| spec
|    | bank_spec.rb
|    | spec_helper.rb
| .rspec
| bank.rb
```

```
describe Bank do
  # This sets up a description of a class (i.e. it will search for a class with the na
me Bank)
end
```

But, there isn't a bank anywhere so you need to require the necessary files! You do this by adding `require_relative "../bank"` at the very top of the file.

We need to actually test individual pieces of everything in here (make sure they are tiny bits at a time!).

```
require_relative "../bank"

describe Bank do
  let (:bank) { Bank.new( 'TDD Bank' ) }
  # This is like a before_action, it will run everytime any new describe gets run.

  describe ".new" do
    # Class methods are prefixed by a dot when describing them
    it "Creates a new Bank object" do
      expect( bank ).to_not eq nil
    end
  end

  describe "#create_account" do
    it "Can create a new account" do
      # yadda yadda yadda
    end
  end
end
```

Basically, testing is broken up into lots of little bits. A describe block for the class as a whole, describe blocks for each individual class and instance method (remember that class methods are prefixed with a . and instace methods are prefixed with a #), and then for each small piece - you should have "it" blocks.

Remember to run all of this crazily regularly, just run `rspec` .

Callbacks and Scope

Callbacks are ridiculously important in Javascript. They are absolutely everywhere and it is really important to get a solid understanding of them. Remember that javascript is an asynchronous programming language, more than one thing can happen at once.

There are a couple of keys to understanding them...

The Problem with Asynchronous Javascript

```
var do_first = function () {
    console.log("The do_first function was called");
}

var do_second = function () {
    console.log("The do_second function was called");
}

do_first();
do_second();

// RESULT
// "The do_first function was called"
// "The do_second function was called"
```

So that works! The order has been kept. But what happens if the do_first function takes a little while longer than the do_second?

```
var do_first = function () {
    setTimeout( function(){
        console.log("The do_first function was called");
    }, 1000 );
}

var do_second = function () {
    console.log("The do_second function was called");
}

do_first();
do_second();

// RESULT
// "The do_second function was called"
// "The do_first function was called"
```

Well that isn't ideal. Obviously the do_first is the function that we want to have completed before the do_second. There are a bunch of different ways around this. Callbacks are probably the main one!

Functions are Objects

Well, they are! When push comes to shove, they are Function objects created using a Function constructor. Interestingly enough, behind the scenes - a Function object contains a random a string which contains javascript code. Due to this fact, we can store functions as strings - and therefore pass things in as parameters.

Passing in a Function as a Parameter

```
var do_first_and_second = function ( callback ) {
    console.log( "Do First." );
    callback();
}

var do_second = function () {
    console.log( "Do Second." );
}

do_first_and_second( function () {
    console.log( "Do Second." );
} );

// OR! (and this is better)

do_first_and_second( do_second );
```

To work with the timing of the things...

```
var do_first_and_second = function ( callback ) {
    setTimeout(function () {
        console.log( "Do First." );
        callback();
    }, 1000);
}

var do_second = function () {
    console.log( "Do Second." );
}

do_first_and_second( do_second );
```

Now that we can pass functions around, we know when to call functions a lot more! This is how things like $(document).ready... etc. work.

XMLHttpRequests (X.H.R - Extensible Markup Language HTTP Requests)

XHR was something that was designed by Microsoft, and something that has since been adopted by Apple, Mozilla and Google. It is also now standardized with the W3C.

It provides a way for us to get information from a given url without refreshing the page, it can update just a part of a page. It's a main part of AJAX (Asynchronous Javascript And XML).

Despite the name, XMLHttpRequests can be used to retrieve any type of data - including files.

The Approach

- Create an instance of the XMLHttpRequest object
- Open a URL (on the instance)
- Send the Request (on the instance)

```
var request = new XMLHttpRequest();
// request.open( "PROTOCOL", "URL" );
request.open( "GET", "http://www.a-particular-website.com" );
request.send();
```

Due to the asynchronous nature of Javascript, we need to figure out a way to retrieve the data. On the request object, after a request has been sent, there is a readystate key. This gets a number between 0 and 4 as its value.

- 0 - request not initialized (not sent)
- 1 - server connection established
- 2 - request received
- 3 - processing request in the browser
- 4 - request finished and the response is ready to be interacted with

We are provided with a handy "onreadystatechange" function, which will be called automatically.

```
var request = new XMLHttpRequest();
// request.open( "PROTOCOL", "URL" );
request.open( "GET", "http://www.a-particular-website.com" );
request.send();

request.onreadystatechange = function () {
    console.log( "Ready State: " + request.readystate );
    // This will print out a number between 0 and 4
}
```

What we really want to do though, is make sure that everything has been loaded.

```
request.onreadystatechange = function () {
    if ( request.readystate === 4 ) {
        var received_data = request.responseText;
        // This is how we access the received data.
        $( "body" ).append( received_data );
    }
}
```

There are short hands for this though. We can always do it in a way that uses callbacks.

```
request.onload = function () {
    // Success handler goes in here!
}
```

For more details, see here, here and here.

Week 07, Day 02

What we covered today:

- AJAX with jQuery

AJAX with jQuery

What is AJAX?

Even if you haven't heard of AJAX before, you definitely would have used an AJAX-based application somewhere along the way - Gmail for example. It stands for Asynchronous Javascript And XML (XML - stands for Extensible Markup Language) and it is a technique for handling external data through Javascript asynchronously without reloading the entire page.

How to do it

There are so many ways. The most flexible is using a function called $.ajax... $.ajax has been around since jQuery 1.0. All of the other AJAX functions that you can use with jQuery use this behind the scenes.

```
// It can look like the following...
$.ajax( url, { OPTIONS } );
// e.g.
$.ajax( "/stats", { method: "GET" } );

// But this is the preferred way!
$.ajax({ OPTIONS });
```

For a more complex example...

```
// Using the core $.ajax() method
$.ajax({

    // The URL for the request
    url: "/post",

    // The data to send (will be converted to a query string)
    data: {
        id: 123
    },

    // What type of request it is
    type: "GET",

    // The type of data we expect back
    dataType : "json",

    // Code to run if the request succeeds;
    // the response is passed to the function
    success: function( data ) {
        $( "<h1>" ).text( data.title ).appendTo( "body" );
        $( "<div class=\"content\">").html( data.html ).appendTo( "body" );
    },

    // Code to run if the request fails; the raw request and
    // status codes are passed to the function
    error: function( xhr, status, errorThrown ) {
        alert( "Sorry, there was a problem!" );
        console.log( "Error: " + errorThrown );
        console.log( "Status: " + status );
        console.dir( xhr );
    },

    // Code to run regardless of success or failure
    complete: function( xhr, status ) {
        alert( "The request is complete!" );
    }

});
```

We could also use promises for this, instead of encapsulating the handlers (success, error etc.), we can chain functions and jQuery promises us that these will get run.

```
$.ajax({...}).always(function () {
    // This will always run
}).done( function (data) {
    // This will run on success
}).fail( function (data) {
    // This will run if there is an error
});
```

For more information, see here and here.

jQuery also provides lots of convenience methods if you don't need something quite as customizable as $.ajax. These include:

- $.get
- $.getJSON
- $.getScript
- $.post
- $.load

They are worth getting to know.

Check this thing out!!!

Same Origin Policy

Same Origin Policy - your javascript can only talk to the current domain (the server that this site is loaded from). If you try and talk to another domain, it might respond with a 'No Cross-Origin Resource Sharing' error (or CORS).

There are three ways to deal with this:

- Allow CORS (like OMDB API)
- Support JSONP
- The sight can host your code

This is called a CORS error, and is a really hard one to deal with.

HOMEWORK

DO ISS STUFF. Space things!

People in space

ISS Lat/Long

List the ISS position and/or people currently in space. Bonus: Make it update in real time on google maps and anything else you can cram in.

Alternatively: Do something interesting with these.

Week 07, Day 03

What we covered today:

Turbolinks

What is it?

Turbolinks, as you might image, makes your links turbo. It is made to make following links in your application faster. Instead of letting the browser get all HTML, CSS and Javascript again, it uses AJAX and just replaces the things that are necessary - changes the title of the page and fills in the body tag with whatever is necessary.

What is wrong with it?

- It can bloat your javascript memory. It doesn't destroy variables and that means that everything you have ever defined exists for as long as a user uses the site.
- It means things like $(document).ready etc. won't work - which is my major gripe with it.
- Events can end up being added lots and lots of times
- setIntervals and setTimeouts can end up still running
- Turbolinks will store the last 10 pages that were loaded, which can be good, but it means your Javascript keeps all of these details in it
- Requires ridiculous amounts of CSS specificity if you are working with some of the things it provides

Because of all of this stuff, it means that you have to write javascript very differently when dealing with Turbolinks.

Using it...

As previously mentioned, Turbolinks doesn't completely refresh the page - which means that we can't rely on $(document).ready running. To solve this, Turbolinks triggers alternate events on the document object.

The important events...

- page:fetch - Starting to fetch a new page

- page:receive - The page has been received but not parsed
- page:change - The page has been changed to the new version
- page:load - Fired at the end of the loading process (the same as $(document).ready)

The one that you would see most is this...

```
$(document).on("ready page:load", function () {  });
```

Caching with Turbolinks

```
Turbolinks.pagesCached(); // Returns how many pages are cached
Turbolinks.pagesCached(20); // Sets how many pages are allowed to be cached

Turbolinks.cacheCurrentPage();
// If you are going to be dynamically adding stuff with AJAX, you can cache a page at
a particular point using this function

// Up until now, the pages are loaded instantaneously if they are cached.  What happen
s if we want it to load straight away and then fetch it just in case?  We run the foll
owing line of code...
Turbolinks.enableTransitionCache();
```

Disabling Turbolinks on a Page or particular Links

```
<!-- On a particular link -->
<a href="/">Home (via Turbolinks)</a> <!-- Will use Turbolinks -->
<a href="/" data-no-turbolink>Home (via Turbolinks)</a> <!-- Won't use Turbolinks -->

<div id="some-div" data-no-turbolink> <!-- None of the a tags within will use turbolin
ks -->
  <a href="/">Home (without Turbolinks)</a>
  <a href="/">Home (without Turbolinks)</a>
  <a href="/">Home (without Turbolinks)</a>
</div>

<!--
If you find that a page is causing problems, you can have Turbolinks skip displaying t
he cached copy by adding data-no-transition-cache to any DOM element on the offending
page.
-->
```

The Turbolinks Progress Bar

Because it skips the browser reload, it won't show the traditional loading bar. Turbolinks gives you the capability in javascript though.

```
Turbolinks.ProgressBar.disable();
Turbolinks.ProgressBar.enable();
```

The Progress Bar is added to the HTML element's pseudo :before element. It can easily be customized with CSS so it looks as you want it to.

```
html.turbolinks-progress-bar::before {
  background-color: red !important;
  height: 5px !important;
}
```

To manually control the progress bar with javascript...

```
Turbolinks.ProgressBar.start();
Turbolinks.ProgressBar.advanceTo(value); // where value is between 0-100
Turbolinks.ProgressBar.done();
```

Allowing Particular Links to use Turbolinks

Only .html files are cached by default with Turbolinks. If you want to do other stuff...

```
Turbolinks.allowLinkExtensions();                 // => ['html']
Turbolinks.allowLinkExtensions('md');             // => ['html', 'md']
Turbolinks.allowLinkExtensions('coffee', 'scss'); // => ['html', 'md', 'coffee', 'scss
']
```

Also! Turbolinks is the last click handler to be added to any element so things like event.preventDefault, event.stopPropagation and event.stopImmediatePropagation will work. This also means that you can still use jquery-ujs (Unobtrusive Javascript - something that comes with Rails) without issues - data-remote, data-confirm, and data-method will all still work.

Working with jQuery

If you have a lot of existing jQuery and it all works due to $(document).ready or other stuff that is no longer working, there is a gem called jquery-turbolinks that can help out.

Steps for getting this working...

- Add this to your gemfile - gem 'jquery-turbolinks'
- Run Bundle
- Add this stuff to your application.js file (in this order!)

```
//= require jquery
//= require jquery.turbolinks
//= require jquery_ujs
//
// ... your other scripts here ...
//
//= require turbolinks
```

See here for more information.

Tracking Assets

By adding **data-turbolinks-track** to a link or script tag. It will always check to see if anything has changed. If the file has changed, it will load the whole page without using Turbolinks. (This will cause the full files to be loaded twice though - once to check if there is a change, and then again if there is a change).

Evaluating Script Tags

Script tags will always be evaluated when using Turbolinks (as long as they don't have a type attribute or the type attribute is equal to text/javascript).

If we only want script tags to be evaluated the first time...

```
<script type="text/javascript" data-turbolinks-eval=false>
    console.log("I'm only run once on the initial page load");
</script>
```

Turbolinks won't evaluate script tags on back/forward navigation unless they have data-turbolinks-eval="always" on it.

```
<script type="text/javascript" data-turbolinks-eval=always>
    console.log("I'm run on every page load, including history back/forward");
</script>
```

Triggering Turbolinks

You can use `Turbolinks.visit(path)` to go to a URL through Turbolinks.

You can also use `redirect_to path, turbolinks: true` in Rails to perform a redirect via Turbolinks.

Although there is a lot of stuff going on with Turbolinks, if you are writing a Javascript-heavy application it is definitely best to avoid it. If you want more information about Turbolinks though, see here.

Homework:

- Continue on with the Flickr project
    - Add a button at the bottom of your page to continue generating more images.
    - Make it an infinite scroll. Convert your button to a trigger that generates the next page of results.
        - Hints:
            - Scrolltop
            - Window & document .height
    - Refresh all your results and reset your page count when a new search is added rather than appending more images.
    - Add a debounce, try underscore
    - Add a lightbox for images when you click them.

Other stuff: Underscore. Learn underscore, read the docs.

Week 07, Day 04

What we covered today:

Underscore

What is Underscore?

It's a programmer's toolbelt. It provides a whole range of useful utility functions that you will end up using thousands and thousands of times. It was developed by a guy called Jeremy Ashkenas (who also made Backbone and Coffeescript).

Using it in Rails

- Add this to your Gemfile - `gem 'underscore-rails'`
- Run `bundle`
- Add this line before the `require tree` stuff - `//= require underscore`

Common Things with Underscore

Best to download this from here, and have a muck around with it.

```
// ////////////////////////////////// //
// Common Methods - Arrays or Objects //
// ////////////////////////////////// //

// Each

// Iterates through each thing in the passed in collection.
// _.each( collection, iteratee_function  );
// The iteratee function receives an element, an index, and en entire collection as pa
rameters

console.log( "UNDERSCORE EACH WITH ARRAY" );
var arr = [ 1, 2, 3 ];
_.each( arr, function ( element, index, entire_array ) {
  console.log( "\tElement: ", element );
  console.log( "\tIndex: ", index );
  console.log( "\tEntire Array: ", entire_array );
} );
console.log("");
```

```
console.log( "UNDERSCORE EACH WITH OBJECT" );
var obj = {
  four: 4,
  five: 5,
  nine: 9
}
_.each( obj, function ( value, key, entire_object ) {
  console.log( "\tValue: ", value );
  console.log( "\tKey: ", key );
  console.log( "\tEntire Object: ", entire_object );
} );
console.log("");

// Map

// Iterates through each thing in the passed in collection and passes back an altered
collection.
// _.map( collection, iteratee_function  );
// The iteratee function receives an element, an index, and en entire collection as pa
rameters

console.log( "UNDERSCORE MAP WITH ARRAY" );
var arr = [ 1, 2, 3 ];
var arrByFive = _.map( arr, function ( element, index, entire_array ) {
  return element * 5;
} );
console.log( "\tOriginal Array: ", arr );
console.log( "\tArray By Five: ", arrByFive );
console.log( "" );


console.log( "UNDERSCORE MAP WITH OBJECT" );
var obj = {
  one: 1,
  two: 2,
  three: 3
};
var objByFive = _.map( obj, function ( element, index, entire_array ) {
  return element * 5;
} );
console.log( "\tOriginal Object: ", arr );
console.log( "\tObject By Five (turned into an array by map): ", objByFive );
console.log( "" );


// Reduce

// Iterates through each thing in the passed in collection and returns a single sum.
// _.reduce( collection, iteratee_function, starting_value  );
// The iteratee function receives a sum, a current value, and en entire collection as
parameters

console.log( "UNDERSCORE REDUCE WITH ARRAY" );
```

```
var arr = [ 1, 2, 3 ];
var reducedArr = _.reduce( arr, function ( sum, value, index, list ) {
  return sum, value;
}, 0 );
console.log( "\tOriginal Array: ", arr );
console.log( "\tReduced Array (to a sum): ", reducedArr );
console.log( "" );

console.log( "UNDERSCORE REDUCE WITH OBJECT" );
var obj = {
  one: 1,
  two: 2,
  three: 3
};
var reducedArr = _.reduce( arr, function ( sum, value, index, list ) {
  return sum, value;
}, 0 );
console.log( "\tOriginal Object: ", obj );
console.log( "\tReduced Object: ", reducedArr );
console.log( "" );

// Find

// Iterates through each thing in the passed in collection and returns true in the pas
sed in function.
// _.find( collection, iteratee_function  );
// The iteratee function receives a current value as a parameter

console.log( "UNDERSCORE FIND WITH ARRAY" );
var arr = [ 1, 2, 3, 4, 5, 6 ];
var firstEven = _.find( arr, function ( num ) {
  return num % 2 === 0;
} );
console.log( "\tOriginal Array: ", arr );
console.log( "\tFirst Even in the Array: ", firstEven );
console.log( "" );

console.log( "UNDERSCORE FIND WITH OBJECT" );
var obj = {
  one: 1,
  two: 2,
  three: 3
};
var firstEven = _.find( obj, function ( num ) {
  return num % 2 === 0;
} );
console.log( "\tOriginal Object: ", arr );
console.log( "\tFirst Even Value in the Object: ", firstEven );
console.log( "" );

// Filter

// Iterates through each thing in the passed in collection and returns everything that
```

```
 returns true in the passed in function.
// _.filter( collection, iteratee_function  );
// The iteratee function receives a current value as a parameter

console.log( "UNDERSCORE FILTER WITH ARRAY" );
var arr = [ 1, 2, 3, 4, 5, 6 ];
var allEvens = _.filter( arr, function ( num ) {
  return num % 2 === 0;
} );
console.log( "\tOriginal Array: ", arr );
console.log( "\tAll Evens in the Array: ", allEvens );
console.log( "" );

console.log( "UNDERSCORE FILTER WITH OBJECT" );
var obj = {
  one: 1,
  two: 2,
  three: 3,
  four: 4
};
var allEvens = _.filter( obj, function ( num ) {
  return num % 2 === 0;
} );
console.log( "\tOriginal Array: ", obj );
console.log( "\tAll Evens in the Array: ", allEvens );
console.log( "" );

// Where

// Iterates through each thing in the passed in collection and returns everything that
 has the same key and values
// _.where( collection, object  );

console.log( "UNDERSCORE WHERE WITH ARRAY FILLED WITH OBJECTS" );
var books = [{
  author : "Gustave Flaubert",
  title  : "Sentimental Education"
}, {
  author : "Marie-Henri Beyle",
  title  : "Lucien Leuwen"
}];
var gustave = _.where( books, { author: "Gustave Flaubert" } );
console.log( "\tOriginal Books Array: ", books );
console.log( "\tGustave Flaubert's Books: ", gustave );
console.log( "" );

// FindWhere

// Same as where except returns the first one.

// Reject

// Iterates through each thing in the passed in collection and removes everything that
```

```
 returns true in the passed in function.
// _.reject( collection, iteratee_function  );
// The iteratee function receives a current value as a parameter

console.log( "UNDERSCORE REJECT WITH ARRAY" );
var arr = [ 1, 2, 3, 4, 5, 6 ];
var odds = _.reject( arr, function ( num ) {
  return num % 2 === 0;
} );
console.log( "\tOriginal Array: ", arr );
console.log( "\tOdds Array: ", odds );
console.log( "" );

// Contains

// Iterates through each thing in the passed in collection and returns true if it has
the passed in value
// _.contain( collection, value  );

console.log( "UNDERSCORE CONTAIN WITH ARRAY" );
var arr = [ 1, 2, 3 ];
var containsThree = _.contains( arr, 3 );
console.log( "\tOriginal Array: ", arr );
console.log( "\tIt contains three? ", containsThree );
console.log( "" );

// Pluck

// Iterates through each thing in the passed in collection and returns just the reques
ted key
// _.pluck( collection, key  );

console.log( "UNDERSCORE PLUCK WITH ARRAY OF OBJECTS" );
var books = [{
  author : "Gustave Flaubert",
  title  : "Sentimental Education"
}, {
  author : "Marie-Henri Beyle",
  title  : "Lucien Leuwen"
}];
var authors = _.pluck( books, 'author' );
console.log( "\tOriginal Books Array: ", books );
console.log( "\tAll Authors: ", authors );
console.log( "" );

// Max

// Returns the maximum value in the array
// _.max( collection );

console.log( "UNDERSCORE MAX WITH ARRAY" );
var arr = [ 1, 292898, 4, 232.223 ];
var maxNum = _.max( arr );
```

```
console.log( "\tOriginal Array: ", arr );
console.log( "\tMaximum Number: ", maxNum );
console.log( "" );

console.log( "UNDERSCORE MAX WITH OBJECT" );
var people = [{
  name: "Marcel",
  age: Infinity
}, {
  name: "Roger",
  age: 34
}];
var oldestPerson = _.max( people, function ( person ) {
  return person.age;
} );
console.log( "\tOriginal Array: ", people );
console.log( "\tOldest Person: ", oldestPerson );
console.log( "" );

// Min

// Returns the minimum value in the array
// _.min( collection );

console.log( "UNDERSCORE MIN WITH ARRAY" );
var arr = [ 0.1, 292898, 4, 232.223 ];
var maxNum = _.min( arr );
console.log( "\tOriginal Array: ", arr );
console.log( "\tMinimum Number: ", maxNum );
console.log( "" );

console.log( "UNDERSCORE MIN WITH OBJECT" );
var people = [{
  name: "Marcel",
  age: Infinity
}, {
  name: "Roger",
  age: 34
}];
var youngestPerson = _.min( people, function ( person ) {
  return person.age;
} );
console.log( "\tOriginal Array: ", people );
console.log( "\tYoungest Person: ", youngestPerson );
console.log( "" );

// SortBy

// Iterates through each item in the collection and sorts them using the given function

// _.sortBy( collection, iteratee_function )

console.log( "UNDERSCORE SORTBY WITH ARRAY" );
```

```
var arr = [ 0.1, 292898, 4, 232.223 ];
var sortedArray = _.sortBy( arr, function (num) {
  return num;
} );
console.log( "\tOriginal Array: ", arr );
console.log( "\tSorted Array: ", sortedArray );
console.log( "" );

console.log( "UNDERSCORE SORTBY WITH OBJECT" );
var people = [{
  name: "Marcel",
  age: Infinity
}, {
  name: "Roger",
  age: 34
}];
var youngestPerson = _.sortBy( people, "age" );
console.log( "\tOriginal Array: ", people );
console.log( "\tSorted by Age: ", youngestPerson );
console.log( "" );

// Shuffle

// Shuffles the given collection
// _.shuffle( collection );

console.log( "UNDERSCORE SHUFFLE WITH ARRAY" );
var arr = [ 1, 2, 3, 4, 5, 6 ];
var shuffledArr = _.shuffle( arr );
console.log( "\tOriginal Array: ", arr );
console.log( "\tShuffled Array: ", shuffledArr );
console.log( "" );

// Sample

// Picks a number (default 1) of elements from the given collection
// _.sample( collection, to_return );

console.log( "UNDERSCORE SAMPLE WITH ARRAY" );
var arr = [ 1, 2, 3, 4, 5, 6 ];
var sample = _.sample( arr );
var threeSample = _.sample( arr, 3 );
console.log( "\tOriginal Array: ", arr );
console.log( "\tSample: ", sample );
console.log( "\tThree Sampled: ", threeSample );
console.log( "" );


// ////////////////////// //
// Common Methods - Arrays //
// ////////////////////// //
```

```
// First

// Returns the first number of element(s) in the collection
// _.first( collection, to_pick );

console.log( "UNDERSCORE FIRST WITH ARRAY" );
var arr = [ 1, 2, 3, 4, 5 ];
var firstOne = _.first( arr );
var firstThree = _.first( arr, 3 );
console.log( "\tOriginal Array: ", arr );
console.log( "\tFirst One: ", firstOne );
console.log( "\tFirst Three: ", firstThree );
console.log( "" );

// Last

// Returns the last number of element(s) in the collection
// _.last( collection, to_pick );

console.log( "UNDERSCORE LAST WITH ARRAY" );
var arr = [ 1, 2, 3, 4, 5 ];
var lastOne = _.last( arr );
var lastThree = _.last( arr, 3 );
console.log( "\tOriginal Array: ", arr );
console.log( "\tLast One: ", lastOne );
console.log( "\tLast Three: ", lastThree );
console.log( "" );

// Compact

// Removes all falsey values in an array
// _.compact( collection );

console.log( "UNDERSCORE COMPACT WITH ARRAY" );
var arr = [ 0, 1, false, 2, '', 3, undefined, NaN ];
var compactedArray = _.compact( arr );
console.log( "\tOriginal Array: ", arr );
console.log( "\tCompacted Array: ", compactedArray );
console.log( "" );

// Flatten

// Turns an array of arrays into one flat array ( can be specified to just flatten to
one level )
// _.flatten( arr, flatten_to_one_level_true_or_false );

console.log( "UNDERSCORE FLATTEN WITH ARRAY" );
var arr = [ [1], [2], [[[1]]] ];
var flattenedArray = _.flatten( arr );
var flattenedArrayToOneLevel = _.flatten( arr, true );
console.log( "\tOriginal Array: ", arr );
console.log( "\tFlattened Array: ", flattenedArray );
console.log( "\tFlattened Array (to one level): ", flattenedArrayToOneLevel );
```

```
console.log( "" );

// Without

// Returns the array without the specified pieces
// _.without( collection, remove_this, remove_this... );

console.log( "UNDERSCORE WITHOUT WITH ARRAY" );
var arr = [ 1, 2, 1, 0, 3, 1, 4 ];
var withoutOnes = _.without( arr, 1 );
var withoutOnesAndTwos = _.without( arr, 1, 2 );
console.log( "\tOriginal Array: ", arr );
console.log( "\tWithout Ones Array: ", withoutOnes );
console.log( "\tWithout Ones and Twos Array: ", withoutOnesAndTwos );
console.log( "" );

// Union

// Returns unique values from all given arrays in an array
// _.union( collection, collection, collection );

console.log( "UNDERSCORE UNION WITH ARRAY" );
var arr1 = [ 1, 2, 3 ];
var arr2 = [ 101, 22, 303.2 ];
var arr3 = [ 1, 2 ];
var uniqItems = _.union( arr1, arr2, arr3 );
console.log( "\tArray 1: ", arr1, " Array 2: ", arr2, " Array 3: ", arr3 );
console.log( "\tUnique Items: ", uniqItems );
console.log( "" );

// Intersection

// Returns values that are present in all given arrays as an array
// _.intersection( collection, collection, collection );

console.log( "UNDERSCORE INTERSECTION WITH ARRAY" );
var arr1 = [ 1, 2, 3 ];
var arr2 = [ 101, 2, 1, 10 ];
var arr3 = [ 2, 1 ];
var intersectedItems = _.intersection( arr1, arr2, arr3 );
console.log( "\tArray 1: ", arr1, " Array 2: ", arr2, " Array 3: ", arr3 );
console.log( "\tItems in all the Arrays: ", intersectedItems );
console.log( "" );

// Uniq

// Returns just unique values from the given array
// _.uniq( arr );

console.log( "UNDERSCORE UNIQ WITH ARRAY" );
var arr = [ 1, 2, 1, 4, 1, 3 ];
var uniqueItems = _.uniq( arr );
console.log( "\tOriginal Array: ", arr );
```

```
console.log( "\tUnique Items in Array Above: ", uniqueItems );
console.log( "" );

// Zip

// Creates an array of arrays but changes the placement of the elements.  Will make an
 array of the first elements, then all the second elements etc. Will return undefined
if there are more elements in the first array
// _.zip( first_array, other_array, other_array );

console.log( "UNDERSCORE ZIP WITH ARRAYS" );
var arr1 = [ 'moe', 'larry', 'curly' ];
var arr2 = [ 30, 40, 50 ];
var arr3 = [ true, false, false ];
var zippedArrays = _.zip( arr1, arr2, arr3 );
console.log( "\tArray 1: ", arr1, " Array 2: ", arr2, " Array 3: ", arr3 );
console.log( "\tZipped Arrays: ", zippedArrays );
console.log( "" );

// Object

// Makes an object with the key coming from the first array and the value coming from
the second array.  Keeps going like that
// _.object( arr1, arr2 );

console.log( "UNDERSCORE OBJECT WITH ARRAYS" )
var arr1 = [ "moe", "larry", "curly" ];
var arr2 = [ 30, 40, 50 ];
var createdObject = _.object( arr1, arr2 );
console.log( "\tArray 1: ", arr1, " Array 2: ", arr2 );
console.log( "\tCreated Object: ", createdObject );
console.log( "" );

// IndexOf

// Returns the index of the specified value. Will return -1 if it isn't present
// _.indexOf( collection, target );

console.log( "UNDERSCORE INDEXOF WITH ARRAYS" );
var target = 2;
var arr = [ 1, 2, 3 ];
var indexOfTarget = _.indexOf( arr, target );
console.log( "\tTarget to find index of: ", target, " Array: ", arr );
console.log( "\tIndex of Target: ", indexOfTarget );
console.log( "" );

// Range

// Creates an array using a range
// _.range( starting_value, ending_value, step );

console.log( "UNDERSCORE RANGE WITH ARRAYS" );
console.log( "\tPassing in 10 to range: ", _.range( 10 ) );
```

```
console.log( "\tPassing in 10 and 20 to range: ", _.range( 10, 20 ) );
console.log( "\tPassing in -1, -11, and -1 to range: ", _.range( -1, -11, -1 ) );
console.log( "" );



////////////////////////////
// Common Methods - Objects  //
////////////////////////////

// Keys

// Returns an array of keys
// _.keys( collection );

console.log( "UNDERSCORE KEYS WITH OBJECTS" );
var obj = {
  one: 1,
  two: 2,
  six: 6
};
var objectKeys = _.keys( obj );
console.log( "\tOriginal Object: ", obj );
console.log( "\tObject Keys: ", objectKeys );
console.log( "" );

// Values

// Returns an array of values
// _.values( collection );

console.log( "UNDERSCORE VALUES WITH OBJECTS" );
var obj = {
  one: 1,
  two: 2,
  six: 6
};
var objectKeys = _.values( obj );
console.log( "\tOriginal Object: ", obj );
console.log( "\tObject Values: ", objectKeys );
console.log( "" );

// Pairs

// Returns an array of arrays with the key being the first element and the value being
 the second element
// _.pairs( collection );

console.log( "UNDERSCORE PAIRS WITH OBJECTS" );
var obj = {
  one: 1,
  two: 2,
  six: 6
```

```
};
var objectKeys = _.pairs( obj );
console.log( "\tOriginal Object: ", obj );
console.log( "\tObject Pairs in Array Form: ", objectKeys );
console.log( "" );

// Invert

// Returns the opposite object. Keys become values
// _.invert( collection );

console.log( "UNDERSCORE PAIRS WITH OBJECTS" );
var obj = {
  one: 1,
  two: 2,
  six: 6
};
var objectKeys = _.invert( obj );
console.log( "\tOriginal Object: ", obj );
console.log( "\tInverted Object: ", objectKeys );
console.log( "" );

// Pick

// Returns an object with just the passed in keys (white listing)
// _.pick( collection, key_to_keep, key_to_keep );

console.log( "UNDERSCORE PICK WITH OBJECTS" );
var obj = {
  name: "Moe",
  age: 50,
  userID: 142423
}
var whiteListedObject = _.pick( obj, "name", "age" );
var functionWhiteListedObject = _.pick( obj, function ( value, key, object ) {
  return _.isNumber( value ); // Returns true if it is a number
} );
console.log( "\tOriginal Object: ", obj );
console.log( "\tWhite Listed Object (passing in keys): ", whiteListedObject );
console.log( "\tWhite Listed Object (passing in a function that returns a boolean): ",
 functionWhiteListedObject );
console.log( "" );

// Omit

// Returns an object without the passed in keys (black listing)
// _.omit( collection, key_to_remove, key_to_remove );

console.log( "UNDERSCORE OMIT WITH OBJECTS" );
var obj = {
  name: "Moe",
  age: 50,
  userID: 142423
```

```
}
var omittedKeys = _.omit( obj, 'name' );
var omittedKeysWithFunction = _.omit( obj, function ( value, key, object ) {
  return _.isNumber( value );
} );
console.log( "\tOriginal Object: ", obj );
console.log( "\tOmitted Keys (specified with key): ", omittedKeys );
console.log( "\tOmitted Keys (specified by a function): ", omittedKeysWithFunction );
console.log( "" );

// Has

// Returns true if the object has the specified key
// _.has( collection, key );

console.log( "UNDERSCORE HAS WITH OBJECTS" );
var obj = {
  name: "Moe",
  age: 50,
  userID: 142423
}
var hasName = _.has( obj, "name" );
console.log( "\tOriginal Object: ", obj );
console.log( "\tDid it have the name key? ", hasName );
console.log( "" );



// ///////////////// //
// Common Functions //
// ///////////////// //

// Delay

// Calls a function after a specified amount of time
// _.delay( function, time_in_ms );

console.log( "UNDERSCORE DELAY WITH FUNCTION" );
var toCall = function () {
  console.log( "\tDelayed function Called." )
}
_.delay( toCall, 1000 );
console.log( "" );

// Throttle

// Says that the specified function can be called only every so often (will call it st
raight away when it reads this line).  Throttle guarantees that the given function act
ually runs
// _.throttle( function, time_in_ms );

console.log( "UNDERSCORE THROTTLE WITH FUNCTION" );
var showScrollTop = function () {
```

```
  var scrollTop = $( window ).scrollTop();
  console.log( "\tScroll Top is: ", scrollTop );
}
var throttledShowScrollTop = _.throttle( showScrollTop, 100 );
$( window ).scroll( throttledShowScrollTop );
console.log( "" );
// This function can only call every 100 milliseconds but will call straight away when
 defined

// Debounce

// Says that the specified function can be called only every so often (won't call stra
ight away)
// _.debounce( function, time_in_ms );

console.log( "UNDERSCORE DEBOUNCE WITH FUNCTION" );
var calculateLayout = function () {
  var windowWidth = $( window ).width();
  console.log( "\tWindow Width is: ", windowWidth );
}
var debouncedCalculateLayout = _.debounce( calculateLayout, 300 );
$(window).resize( debouncedCalculateLayout );
// This function can only call every 300 ms but won't call straight away

// Once

// Says that the specified function can be called only once
// _.once( function );

console.log( "" );
console.log( "UNDERSCORE ONCE WITH FUNCTION" );
var createApplication = function () {
  console.log( "\tCreate Application called." );
}
var initialize = _.once( createApplication );
initialize(); // This will call
initialize(); // This won't call
console.log( "" );

// Times

// Call the passed in function a specified amount of times (receives an index as a par
ameter)
// _.times( num_of_times, function );

console.log( "UNDERSCORE TIMES WITH FUNCTION" );
_.times( 3, function ( index ) {
  console.log( "\tIndex: ", index );
} );
console.log( "" );

// Random
```

```
// Generates a random value between 0 and the passed in number if just one value is pa
ssed in.  Or between the first and second values.  Best to be explicit and pass in 0 i
f necessary
// _.random( starting_point, ending_point );

console.log( "UNDERSCORE RANDOM WITH FUNCTION" );
var upToOneHundred = _.random( 100 );
var betweenOneHundredAndTwoHundred = _.random( 100, 200 );
var betweenOneHundredAndMinusTwoHundred = _.random( 100, -200 );
var betweenMinusOneHundredAndMinusTwoHundred = _.random( -100, -200 );
console.log( "\tUp to 100: ", upToOneHundred );
console.log( "\tBetween 100 and 200: ", betweenOneHundredAndTwoHundred );
console.log( "\tBetween 100 and -200: ", betweenOneHundredAndMinusTwoHundred );
console.log( "\tBetween -100 and -200: ", betweenMinusOneHundredAndMinusTwoHundred );
console.log( "" );



// ///////////////// //
// Predicate Methods //
// ///////////////// //

// isEqual

// Checks whether two collections are equal
// _.isEqual( collection_one, collection_two );

console.log( "UNDERSCORE ISEQUAL WITH COLLECTION" );
var arr1 = [ 0, 1, 2 ];
var arr2 = [ 0, 1, 2 ];
var returnedEquals = arr1 === arr2; // Returns false
var returned = _.isEqual( arr1, arr2 ); // Returns true
console.log( "\tThing 1: ", arr1, " Thing 2: ", arr2 );
console.log( "\tThing 1 and Thing 2 compared with three equals: ", returnedEquals );
console.log( "\tThing 1 and Thing 2 compared with isEqual: ", returned );
console.log( "" );

// isMatch

// Tells you if the keys and values in properties are contained in object.
// _.isMatch( collection, obj );

console.log( "UNDERSCORE ISMATCH WITH OBJECT" );
var obj = {
  name: "Roger"
};
var matched = _.isMatch( obj, { name: "Roger" } ); // Returns true
console.log( "\tObject: ", obj );
console.log( "\tObject has a name: ", matched );
console.log( "" );

// isEmpty
```

```
// Returns true if there is nothing in the array or the object
// _.isEmpty( thing );

console.log( "UNDERSCORE ISEMPTY WITH COLLECTION" );
var emptyArr = [];
var notEmptyObj = {
  name: "Roger"
};
var emptyArrMethod = _.isEmpty( emptyArr ); // Returns true
var filledObjMethod = _.isEmpty( notEmptyObj ); // Returns false
console.log( "\tArray is: ", emptyArr, ". Is it empty? ", emptyArrMethod );
console.log( "\tObject is: ", notEmptyObj, ". Is it empty? ", filledObjMethod );
console.log( "" );

// isArray

// Returns true if it is an array
// _.isArray( thing );

console.log( "UNDERSCORE ISARRAY WITH THING" );
var arr = [];
var obj = {};
var arrMethod = _.isArray( arr ); // Returns true
var objMethod = _.isArray( obj ); // Returns false
console.log( "\tThing 1: ", arr, ". Is it an array? ", arrMethod );
console.log( "\tThing 2: ", obj, ". Is it an array? ", objMethod );
console.log( "" );

// isObject

// Returns true if it is an object
// _.isObject( thing );

console.log( "UNDERSCORE ISOBJECT WITH THING" );
var arr = [];
var obj = {};
var arrMethod = _.isObject( arr ); // Returns false
var objMethod = _.isObject( obj ); // Returns true
console.log( "\tThing 1: ", arr, ". Is it an object? ", arrMethod );
console.log( "\tThing 2: ", obj, ". Is it an object? ", objMethod );
console.log( "" );

// isFunction

// Returns true if it is a function
// _.isFunction( thing );

console.log( "UNDERSCORE ISFUNCTION WITH THING" );
var myFunc = function () {};
var arr = [];
var funcMethod = _.isFunction( myFunc ); // Returns true
var arrMethod = _.isFunction( arr ); // Returns false
console.log( "\tThing 1: ", myFunc, ". Is it a function? ", funcMethod );
```

```
console.log( "\tThing 2: ", arr, ". Is it a function? ", arrMethod );
console.log( "" );

// isString

// Returns true if it is a string
// _.isString( thing );

console.log( "UNDERSCORE ISSTRING WITH THING" );
var myFunc = function () {};
var str = "";
var funcMethod = _.isFunction( myFunc ); // Returns false
var strMethod = _.isFunction( str ); // Returns true
console.log( "\tThing 1: ", myFunc, ". Is it a string? ", funcMethod );
console.log( "\tThing 2: ", str, ". Is it a string? ", strMethod );
console.log( "" );

// isNumber

// Returns true if it is a number
// _.isNumber( thing );

console.log( "UNDERSCORE ISNUMBER WITH THING" );
var myFunc = function () {};
var num = 1;
var funcMethod = _.isNumber( myFunc ); // Returns false
var numMethod = _.isNumber( num ); // Returns true
console.log( "\tThing 1: ", myFunc, ". Is it a number? ", funcMethod );
console.log( "\tThing 2: ", num, ". Is it a number? ", numMethod );
console.log( "" );

// isFinite

// Returns true if it is a finite value
// _.isFinite( thing );

console.log( "UNDERSCORE ISFINITE WITH COLLECTION" );
var infiniteThing = -Infinity;
var finiteThing = 1;
var infiniteMethod = _.isFinite( infiniteThing ); // Returns false
var finiteMethod = _.isFinite( finiteThing ); // Returns true
console.log( "\tThing 1: ", infiniteThing, ". Is it finite? ", infiniteMethod );
console.log( "\tThing 2: ", finiteThing, ". Is it finite? ", finiteMethod );
console.log( "" );

// isBoolean

// Returns true if it is a boolean value
// _.isBoolean( thing );

console.log( "UNDERSCORE ISBOOLEAN WITH COLLECTION" );
var isTrue = true;
var str = "";
```

```
var trueMethod = _.isBoolean( isTrue ); // Returns true
var strMethod = _.isBoolean( str ); // Returns false
console.log( "\tThing 1: ", isTrue, ". Is it a function? ", trueMethod );
console.log( "\tThing 2: ", str, ". Is it a function? ", strMethod );
console.log( "" );

// isNaN

// Returns true if it is NaN
// _.isNaN( thing );

console.log( "UNDERSCORE ISNAN WITH COLLECTION" );
var str = "";
var nope = NaN;
var strMethod = _.isNaN( str ); // Returns false
var nanMethod = _.isNaN( nope ); // Returns true
console.log( "\tThing 1: ", str, ". Is it NaN? ", strMethod );
console.log( "\tThing 2: ", nope, ". Is it NaN? ", nanMethod );
console.log( "" );

// isNull

// Returns true if it is null
// _.isNull( thing );

console.log( "UNDERSCORE ISNULL WITH COLLECTION" );
var str = "";
var nope = null;
var strMethod = _.isNull( str ); // Returns false
var nullMethod = _.isNull( nope ); // Returns true
console.log( "\tThing 1: ", str, ". Is it null? ", strMethod );
console.log( "\tThing 2: ", nope, ". Is it null? ", nullMethod );
console.log( "" );

// isUndefined

// Returns true if it is undefined
// _.isUndefined( thing );

console.log( "UNDERSCORE ISUNDEFINED WITH COLLECTION" );
var str = "";
var nope = undefined;
var strMethod = _.isUndefined( str ); // Returns false
var undefinedMethod = _.isUndefined( nope ); // Returns true
console.log( "\tThing 1: ", str, ". Is it undefined? ", strMethod );
console.log( "\tThing 2: ", nope, ". Is it undefined? ", undefinedMethod );
console.log( "" );
```

Homework:

Finish these if you haven't:

- One
- Two
- Three
- Four
- Five

If you finish those and are bored, research Backbone, read the docs.

Week 07, Day 05

What we covered today:

Backbone

The Problem

When you work on lots and lots of javascript, your files always end up all over the place - spaghetti code. When asked to add a feature, you might just have to find a random place and give it a go.

You also end up tying your data to the D.O.M (the Document Object Model). It's very easy, and happens ridiculously regular, that javascript applications become "tangle piles of jQuery selectors and callbacks", and they are all trying to keep data in sync between your JS, Database and the UI itself. For rich client-side applications, a far more structured approach is necessary.

The Solution

Using a front end framework! There are lots and lots of options here, this makes it difficult to pick the right one. Here we really like Backbone. It is unopinionated, lightweight, flexible and works brilliantly in lots of lots of situations. Other options include Ember.js (developed by Yehuda Katz), Angular.js (a Google thing) and React.js (which is a little different - developed by Facebook).

Backbone - A Ridculously Brief History

Backbone was developed by Jeremy Ashkenas, the creator of Underscore.js and Coffeescript, while he was at DocumentCloud (I believe - it might have been while he was at the New York Times as well).

Worth checking out the annotated source. Here is its website, and here is its github.

MVC x MV*

Seperation of concerns is a huge deal in programming and that is something that both the M.V.C and M.V.* approaches follow. In a M.V.C approach, thins are broken up into:

- Models - The data
- Views - The representation
- Controller - The orchestrator

M.V.C has additional bits as well like the Router etc. This is the way that Rails works. The M.V.* approach is a little different. It breaks everything up into:

- Models - the data
- Views - the actual User Interface
-
    -
        - Everything else. The router, collections of data etc.

A Brief Interlude... Literate Programming

Literate programming is a different style of programming compared to what we have seen before. It was introduced by Donald Knuth with this book. It's main principle is that a program is given as an explanation of the program logic in a natural language, whether that is English or otherwise, interspersed with snippets of code.

The Manual Approach - Back to Backbone

Often used in conjunction with jQuery, Backbone relies on Underscore.js. We need to include those things.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Basic Backbone App Structure</title>
</head>
<body>


    <!-- OTHER SCRIPTS. -->
    <!-- jQuery from cdnjs.com -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/1.11.3/jquery.min.js"></
script>
    <!-- underscore.js from cdnjs.com -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.3/underscore
-min.js"></script>
    <!-- backbone.js from cdnjs.com -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.2.1/backbone-min
.js"></script>

    <!-- Your Scripts Here -->
    <script src="js/main.js"></script>
</body>
</html>
```

We need to create a seperate javascript file for all this stuff (normally we would have a lot more files than just one). So let's create one now called `main.js` and reference it in the html.

```
FILE STRUCTURE!

backbone-intro
| css
| js
  | main.js
index.html
```

Models are the building blocks of Backbone. This, again, is a way to create data in a structured manner. Models are the individual pieces (whether it is a user, animal etc.). They look like this...

```
var Animal = Backbone.Model.extend({  }); // This is the blueprint (or constructor)

var animal = new Animal();
```

We can put lots of stuff in here. Defaults for example...

```
var Animal = Backbone.Model.extend({
    defaults: {
        name      : "Generic Animal",
        ecosystem : "",
        stripes   : 0
    }
});

var animal = new Animal();
console.log( animal ); // It has the default values
```

We can also assign values quite easily. We do this by passing the new Animal function an object. This works even if there are default values.

```
var Animal = Backbone.Model.extend({  });

var animal = new Animal( {
    name        : "Wolf",
    ecosystem   : "Hearts and Minds",
    stripes     : "As many as it desires"
} );
console.log( animal ); // It has all the values that we passed in
```

Now it's not ideal to console.log things, we actually want to let javascript get and set the values after creation. We can't access the keys and values through normal notation, we use get and set.

```
var Animal = Backbone.Model.extend({  });
var animal = new Animal( {
    name        : "Wolf",
    ecosystem   : "Hearts and Minds",
    stripes     : "As many as it desires"
} );

// To get attributes...
animal.get( "name" ); // Returns "Wolf"
animal.get( "ecosystem" ); // Returns "Hearts and Minds"

// To set attributes...
animal.set( "name", "Vargen" );
animal.set( "stripes", "earnt" );

// We can also retrieve all of it's attributes...
animal.attributes; // Returns the object itself (we can then work with it normally) -
don't do it this way!
```

On creation of a new Animal, we can call a function. We do this using an initialize function.

```
var Animal = Backbone.Model.extend( {
    initialize: function ( new_animal ) {
        // This will run every time.
        console.log( "New animal created." );
    }
} );

var animal = new Animal(); // Will run the initialize function
```

We can also add "watchers". Watchers wait for changes in attributes in Backbone Models and can fire functions.

```
var Animal = Backbone.Model.extend( {
    initialize: function ( new_animal ) {
        console.log( "New animal created." );

        // This function listens for a change in the name attribute on any animal crea
ted with this Animal constructor.
        this.on( "change:name", function ( model ) {
            // We need to receive model (as a paramter) so we know which model changed.

            var type = model.get( "type" );
            console.log( "I am now a " + type );
        } );
    }
} );

var animal = new Animal( { name: "Wolf" } ); // Will fire the initialize method
animal.set( "name", "Vargen" ); // Will fire the change:name function
```

One of the greatest things about Backbone is the way that you can group models together. We use Backbone.Collection to do this...

```
var Animal = Backbone.Model.extend({  });
var Zoo = Backbone.Collection.extend( {
    model : Animal // We need to tell Backbone what model to use.
} );

var animal1 = new Animal();
var animal2 = new Animal();
var animal3 = new Animal();
// Create three instances of the Animal model.

var myZoo = new Zoo( [ animal1, animal2, animal3 ] );
// This creates the instance of the collection and passes in the three models we creat
ed
```

These things are now grouped. The next great thing about Backbone is the view layer. Views and Collections work really nice together.

```
var Animal = Backbone.Model.extend({  });
var Zoo = Backbone.Collection.extend( {
    model : Animal // Most of the time we tell Backbone what model to use.  This helps
 us in the long run so do it this way
} );

var animal1 = new Animal();
var animal2 = new Animal();
var animal3 = new Animal();

var myZoo = new Zoo( [ animal1, animal2, animal3 ] );
// This creates the instance of the collection and passes in the three models

var ZooView = Backbone.View.extend( {
    el : "#main",
    // Backbone Collections work nicely with jQuery.  When you set a el key on it (lik
e above), it will allow you to access it with el or $el (the jQuery selected version).

    initialize: function () {
        // Every time a new view is created, this will be run.
        console.log( "View Initialized" );
    },

    render: function () {
        var view = this;
        // An each loop redefines "this" so we need to keep a reference to it.  In an
each loop - "this" is the current piece of the collection.  When it comes to "this", i
f it has a value that is actually really important and you need to keep referring to i
t - save it in a variable that means more.

        // view.collection gets defined when we create the new instance of the view (
a couple of lines below ).
        view.collection.each( function ( model ) {
            // Iterate through each model in the collection
            var name = model.get( "name" ); // Retrieve the current model's name
            var $h2 = $( "<h2 />" ).text( name ); // Create an h2 with the name in it
            view.$el.append( $h2 ); // Use the jQuery version of el (defined above) an
d append the h2 we just created.
        } );
    }
} );

var zooView = new ZooView( { collection : myZoo } ); // We pass in the collection so t
hat the render can do what we want
```

Backbone Blog

An Approach for Building Backbone Apps

- Lots and lots of thinking
- Create a basic HTML page

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Basic Backbone App Structure</title>
</head>
<body>

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

  <!-- OTHER SCRIPTS. -->
  <!-- jQuery from cdnjs.com -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/1.11.3/jquery.min.js">
</script>
  <!-- underscore.js from cdnjs.com -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.3/undersco
re-min.js"></script>
  <!-- backbone.js from cdnjs.com -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.2.1/backbone-m
in.js"></script>

  <!-- Your Scripts Here -->
  <script src="js/main.js"></script>
</body>
</html>
```

- Referencing all of your files
- Create the router blueprint

```
var AppRouter = Backbone.Router.extend( {
  routes: {
      "" : "index"
  },
  // Any functions required for the routes
  index: function () {

  }
} );
```

- Create a new instance of the AppRouter constructor, and tell it to start listening. Make sure you don't do this until the document has loaded!

  ```js
  ```

var router; $(document).ready(function () { router = new AppRouter(); // Creates a new instance Backbone.history.start(); // Tells it to start listening });

```
```

- Decide what you want in each view. Let's pretend we are working with the index page
- We want that to show the intro page, so we are going to create an AppView and then render it.
- Create your main view, give it an el, an initialize method, and a render method. Make sure in the render method that you save a reference to the view.

```
var AppView = Backbone.View.extend( {
  el : "#main",
  initialize: function () {  },

  render: function () {
      var view = this;
      var $welcomeText = $( "<h1 />" ).text( "Welcome to our App" );
      view.$el.html( $welcomeText );
  }
} );
```

- Actually create an instance of that view in the Router blueprint (AppView)

```
var AppRouter = Backbone.Router.extend( {
  routes: {
      "" : "index",
      "posts/:id" : "showPost"
  },
  // Any functions required for the routes
  index: function () {
      var appView = new AppView();
      appView.render(); // Have this written ready to pass in a collection if nece
ssary
  },

  showPost: function ( id ) {
      var post = blogPosts.get( id );
      var postView = new PostView( { model : post } );
      postView.render();
  }
} );
```

- Let's work with templates though, so let's make some of them

```
<script id="appTemplate" type="text/template">
  <h1>Welcome to our Application</h1>
  ul.posts
</script>
```

- Change the render function on the AppView to use the template

```
render: function () {
  var view = this;
  var $welcomePageHTML = $( "#appTemplate" ).html();
  view.$el.html( $welcomePageHTML );
}
```

- Now define your main model and give it some default values (if necessary)

```
var Post = Backbone.Model.extend( {
  defaults : {
      title   : "Untitled Post",
      content : "Untitled Post Content"
  }
} );
```

- Create your collection - passing the appropriate model

```
var Posts = Backbone.Collection.extend( {
  model : Post
} );
```

- Create some sample data (models that are given to the collection) to make sure that it all has worked.

```
var blogPosts = new Posts([
  new Post( { id: 1, title: "Post 1", content: "Post 1 Content" } ),
  new Post( { id: 2, title: "Post 2", content: "Post 2 Content" } ),
  new Post( { id: 3, title: "Post 3", content: "Post 3 Content" } ),
]);
```

- Check that all the connections have worked!!

```
console.log( "Blog Posts object: ", blogPosts );
console.log( "Blog Posts length: ", blogPosts.length );
console.log( "Models in our blogPosts collection: ", blogPosts.models );
```

- Change the index function to show all the dummy data

```
var AppView = Backbone.View.extend( {
  el : "#main",
  initialize: function () {  },
  render: function () {
      var view = this;
      var appHTML = $( "#appTemplate" ).html();
      view.collection.each( function ( post ) {
          console.log( post.toJSON() );
      } );
  }
} );
```

- Change the creation and render of this particular view to pass in a collection

```
var appView = new AppView( { collection : blogPosts } );
appView.render();
```

- Create smaller views if necessary (more localised - for one post for example)

```
var PostListView = Backbone.View.extend( {
  tagName : "li",
  render: function () {
      console.log( "The model passed in - ", this.model.toJSON() );
  }
} );
```

- Change the appView render to create new instances of this PostListView

```
render: function () {
  var view = this;
  var appHTML = $( "#appTemplate" ).html();
  view.collection.each( function ( post ) {
      var postListView = new PostListView( { model: post } );
      // Create a new instance of PostListView, passing in a model
      postListView.render()
  } );
}
```

- Now that we are going to want interpolation with the templating stuff, let's add a different style of interpolation (using curly brackets) to underscore templates by adding this code at the very top of the file

```
_.templateSettings = {
    evaluate : /\{\[([\s\S]+?)\]\}/g,     // {[ console.log("Hello"); ]} - runs
    interpolate : /\{\{([\s\S]+?)\}\}/g   //  - interpolates
};
```

- And let's create our template, remembering to put it at the bottom of the HTML page in

script tags with a type of "text/template"

```
<script class="#postListTemplate" type="text/template">
  <h3>
  <p>
</script>
```

- In our PostListView render function, we need to actually use this template to work our stuff

```
render: function () {
  var postListTemplate = $( "#postListTemplate" ).html();
  var postListHTML = _.template( postListTemplate );
  this.$el.html(  postListHTML( this.model.toJSON() ) );
}
```

- Now we need to give some events to our PostListView

```
var PostListView = Backbone.View.extend( {
  events: {
      "click" : "showPost"
  },
  showPost: function () {
      router.navigate( "posts/" + this.model.get("id"), true ); // Change URL with
 Backbone to posts/:id
  }
} )
```

- We actually need to listen for this URL. Add a few things to your Router!

```
var AppRouter = Backbone.Router.extend( {
  routes: {
      "" : "index",
      "posts/:id" : "showPost"
  },
  // Any functions required for the routes
  index: function () {
      var appView = new AppView();
      appView.render(); // Have this written ready to pass in a collection if nece
ssary
  },

  showPost: function ( id ) {
      var post = blogPosts.get( id );
      var postView = new PostView( { model : post } );
      postView.render();
  }
} );
```

- We have said that we need to render a postView, lets create it!

```
var PostView = Backbone.View.extend( {
  el: "#main",
  render: function () {
      var postTemplate = $( "#postTemplate" ).html();
      var postHTML = _.template( postTemplate );
      this.$el.html( postHTML( this.model.toJSON() ) );
  }
} );
```

Must Haves in Backbone

- Backbone.View.extend({ });
  - el
  - initialize (maybe)
  - render
- Backbone.Collection.extend({ });
  - model
  - initialize (maybe)
- Backbone.Model.extend({ });
  - defaults (maybe)
  - initialize
  - watchers - change etc.

HOMEWORK:

Make this

All this

Week 08

Week 08, Day 01

What we covered today:

Backbone

Structure

One of the main reasons to use Backbone is because of Seperation of Concerns. But so far, we have kept everything in one file. That is really not ideal!

The basic folder structure for any Backbone project...

```
|   YourJavascripts
        backbone.js
        underscore.js
        jQuery.js

        |   models
        |   collections
        |   views
        |   routers
```

So, we have folders for models, collections, views and routers. By creating this broken up structure, we end up having small, concise files that do one thing (hopefully, well).

Another thing that is very common for working with Backbone in multiple files is using a global app object, something that encompasses all of our data. There are lots of reasons for doing this, but stopping pollution of the global namespace (not having heaps of variables visible) is a big one, also reducing the change for errors. To do this, there are three things we need to do.

First off, you need to have this line at the top of each file:

```
var app = app || {};
```

That says uses the or operator to act as a way of providing a default. If app is defined, it will just use that, otherwise it will be defined as an empty object.

Secondly, you need to scope all of your variables. It may end up looking something like this:

```
var app = app || {};

app.Posts = Backbone.Collection.extend({}); // for example
app.Post = Backbone.Model.extend({}); // another example
```

And finally, you need to work with the way that files are required. The order normally is:

- Libraries (jQuery, then Underscore, then Backbone)
- All of your models
- All of your collections
- All of your views
- Your router
- The main app file

Backbone and Rails - The Really Important things

Backbone works really well with Rails, but does rely heavily on a RESTful API - which Rails all but enforces.

Let's create an application! `rails new backboneblog-rails`

Because Rails handles the requiring of all our javascript files, we need to customize our application.js file to work with the correct order.

```
//= require jquery
//= require jquery_ujs
//= require underscore
//= require backbone
//= require_tree ./models
//= require_tree ./collections
//= require_tree ./views
//= require_tree ./routers
//= require_tree .
```

The next thing we encounter is the fact that ERB uses the same style of interpolation as underscore does in the interpolation of templates, so we need to customize them:

```
_.templateSettings = {
    evaluate : /\{\[([\s\S]+?)\]\}/g,     // {[ console.log("Hello"); ]} - runs code (
for if statements, loops etc.)
    interpolate : /\{\{([\s\S]+?)\}\}/g   // {{ key }} - interpolates
};
```

After we have customized a bunch of these things to make sure it all works, it is time for us to actually create a database and create a post model.

```
rails generate scaffold Posts content:text title:text
rake db:migrate
```

Now, we have a fully functioning posts database table to work with! Start up the server and create a few posts to work with (this should be on localhost:3000/posts/new).

There are a bunch of functions that allow us to interact with the server, but they won't work until we actually provide the URLs, so let's go ahead and do that.

```
app.Post = Backbone.Model.extend({
    urlRoot: "/posts",
    defaults: {
        title: "Hello World",
        content: "I exist"
    }
});

app.Posts = Backbone.Collection.extend({
    url: "/posts",
    model: app.Post
});
```

The really important things in here are:

- You must provide a `urlRoot` to a model, that specifies the base RESTful route to use
- You must provide a `url` to a collection, which specifies the base RESTful route to use

Now that we have specified those things, we can start to use all of the useful functionality that Backbone offers.

Some of the methods that Backbone provides on models are:

- save
- fetch
- destroy

Some of the methods that Backbone provides on collections are:

- create
- fetch
- sync

But before we go ahead and do that, let's create an application from scratch!

Backbone and Rails - A Full Tutorial with Secrets

- Create the app - `rails new secrets-app`
- Move into the app directory - `cd secrets-app`
- Create a scaffold for individual secrets - `rails generate scaffold Secret content:text`
- Run the migration - `rake db:migrate`
- Let's create some secrets, so start up the server and visit `localhost:3000`
- Once that is done, let's add some stuff into the config/routes.rb file. We just want to add a custom root. `root :to => 'secrets#home'` or `root 'secrets#home'`
- That will fail, because we haven't added that action. Add a home method in the app/controllers/secrets_controller.rb

```
def home
end
```

- That will now complain about the view not existing, so let's create our file. `touch app/views/secrets/home.html.erb`
- Let's navigate to the javascripts folder - `cd app/assets/javascripts`
- It is now time to create our folder structure... `mkdir models collections views routers`
- We then need to download Backbone and Underscore and add a main javascript file
  - curl https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.2.3/backbone-min.js > backbone.js
  - curl https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.3/underscore-min.js > underscore.js
  - touch main.js
- We also need to alter our application.js file so that it requires files in the correct order, it needs to look somewhat like this:

```
//= require jquery
//= require jquery_ujs
//= require underscore
//= require backbone
//= require_tree ./models
//= require_tree ./collections
//= require_tree ./views
//= require_tree ./routers
//= require_tree .
```

- Let's start doing some work in the main.js file (our main file)

```
var app = app || {};

_.templateSettings = {
    evaluate : /\{\[([\s\S]+?)\]\}/g,
    interpolate : /\{\{([\s\S]+?)\}\}/g
};
```

- We then get start creating our models, so touch a secret.js file in the app/assets/javascripts/models folder - `touch models/Secret.js`

```
var app = app || {};

app.Secret = Backbone.Model.extend({
    urlRoot: "/secrets",
    defaults: {
        content: ""
    }
});
```

- Test that that has worked by trying this in the console of the browser - `var secret = new app.Secret({ content: "Hello World." });`
- Now we want to have a collection of secrets, so make a secrets.js file in the collections folder - `touch collections/Secrets.js` . In there, put the following code:

```
var app = app || {};

app.Secrets = Backbone.Collection.extend({
    url: "/secrets",
    model: app.Secret
});
```

- Let's test that that has worked. Try running this in the console - `var secrets = new app.Secrets();` , then run `secrets.fetch()` . If there are no errors, then we are all good!
- It's time to move onto the Router, `touch routers/AppRouter.js` . And let's put some code in there!

```
var app = app || {};

app.AppRouter = Backbone.Router.extend({
    routes: {
        '': 'index'
    },

    index: function () {
        console.log( "You have reached the index." );
    }
});
```

- That won't work because we never have started up the router, so let's do that. Open up the main.js file and put this at the bottom:

```
$(document).ready( function () {
    app.secrets = new app.Secrets();
    app.secrets.fetch().done(function () {
        app.router = new app.AppRouter();
        Backbone.history.start();
    });
});
```

- There are lots of things going on in there, so let's have a look at it
  - `app.secrets = new app.Secrets();` - creates a new instance of our Secrets collection
  - Get them all from the url defined, and once that is done:
  - Create a new instance of the router
  - And get Backbone to start listening to the urls
- We now need to create our view to render our app. `touch views/AppView.js` - Put the following stuff inside there:

```
var app = app || {};

app.AppView = Backbone.View.extend({
    el: '#main',
    render: function () {
        var appViewTemplate = $("#appViewTemplate").html();
        this.$el.html( appViewTemplate );
    }
});
```

- Because we are referencing the element with an ID of main, we better to add that element into the home.html.erb file (in app/views/secrets). We also want to add our templates

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

<script id="appViewTemplate" type="template/underscore">
    <h1>Secrets</h1>
    <div id="secretForm">textarea view goes here</div>

    <ul id="secrets">
        <li>Secrets go here as list items</li>
    </ul>
</script>
```

- Now that we have created our elements, in the Backbone router that we created, put the following code in the index function:

```
var appView = new app.AppView({ collection: app.secrets });
appView.render();
```

- Let's go ahead and create an individual secret view. So make a file called SecretView.js in app/assets/javascripts/views. It's going to look something like this:

```
app.SecretView = Backbone.View.extend({
    tagName: "li",
    render: function () {
        this.$el.text( this.model.get('content') );
        this.$el.prependTo( '#secrets' );
    }
});
```

- And then we need to actually render the individual secrets using that view. So let's make the AppView's render function look like this:

```
render: function () {
    var appViewTemplate = $("#appViewTemplate").html();
    this.$el.html( appViewTemplate );

    var secretInput = new app.SecretInputView();
    secretInput.render();

    this.collection.each(function (secret) {
        var secretView = new app.SecretView({ model: secret });
        secretView.render()
    })
}
```

- We now want to put a way to create secrets, so let's make a SecretInputView - make a file called SecretInputView.js in the app/assets/javascripts/views folder. It's going to look like this:

```
var app = app || {};

app.SecretInputView = Backbone.View.extend({
    el: "#secretView",
    render: function () {
        var secretInputViewTemplate = $("#secretInputViewTemplate").html();
        this.$el.html( secretInputViewTemplate );
        this.$el.find('textarea').focus();
    }
});
```

- But we don't have that template, so put this in your home.html.erb

```
<script id="secretInputViewTemplate" type="template/underscore">
    <textarea placeholder="Tell me all your secrets"></textarea>
    <button>Submit</button>
</script>
```

- We want this last view to do a bunch of work, so let's have a look at that:

```
var app = app || {};

app.SecretInputView = Backbone.View.extend({
    el: "#secretView",
    events: {
        'click button' : 'createSecret'
    },
    render: function () {
        var secretInputViewTemplate = $("#secretInputViewTemplate").html();
        this.$el.html( secretInputViewTemplate );
        this.$el.find('textarea').focus();
    },
    createSecret: function () {
        var secret = new app.Secret();
        secret.set("content", this.$el.find('textarea').val() );
        secret.save();

        app.secrets.add( secret );
    }
});
```

- Finally, for the sake of this tutorial, let's add this into the initialize method of your collection (Secrets.js in app/assets/javascripts/collections)

```
this.on('add', function (secret) {
   var secretView = new app.SecretView({model: secret});
   secretView.render();
});
```

There are lots of things going on in here, and here is the project in it's entirety. Well worth going through that link as we didn't write every line of code in this tutorial.

Homework:

Burning Airlines!

Your groups:

["Dan", "Jeremy", "Paula"] ["Sigmund", "Simon", "Michael"] ["Jane", "Fabio", "Tall Jess"] ["Donn", "Monali", "Cedric"] ["Lex", "Harrison", "Nik"] ["Little Jess", "Caylie", "Yumi"]

The brief:

Hurr.

Week 08, Day 02

Burning Airlines!

Week 08, Day 03

Burning Airlines!

Week 08, Day 04

What we covered today:

- Burning Airlines demos
- Delivered Project Brief
- Do your thing!

Week 10

Week 10, Day 01

What we covered today:

- Warmup
  - solution
- Ron talked to us about UXDI
  - Slides
- Responsiveness
  - Viewports
  - Fluid sizing
  - Grids
  - Units
  - Media queries
- Advanced CSS
  - Animations
  - Frameworks and libraries. Slides

Advanced CSS:

Responsiveness

Most people think that responsive web design comes from this article here, it was posted on A List Apart on May 25, 2010 and written by Ethan Marcotte (twitter and website).

But what is it in response to?

- *"Aim for eternity" - Christopher Wren*
- *"We can't break the web."*

What is it?

Well, firstly, what is the web? Designing websites that respond to the needs of users and the devices that they are using Responsive websites respond to their environments

Adaptive: Multiple fixed width layouts

Responsive: Multiple fluid grid layouts

There are a couple of different approaches:

- Content-first design
  - Where the focus is primarily on presenting content, the content shapes your design.
- Desktop-first design
  - Where the focus is primarily on users who will be interacting with the site through their PCs and laptops.
- Mobile based design
  - Where the focus is on mobile users working with limited screen space.

Key components:

Viewports:

What is it? The viewport is user's visible portion of the screen.

Why does it exist? When mobile devices first became popular, it became necessary to scale websites down to the smaller screen sizes so everything could be seen.

Add this to your header:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<!-- width sets the actual width -->
<!-- initial-scale sets the initial zoom -->
```

Fluid sizing of elements:

This just means make the sizing of the elements adapt to the browser they are being viewed with.

When adding elements to the page, be careful to not hard-code their sizing. This could just be the differences between setting widths and max-widths -- it is particularly important for images.

Grids:

Breaking a site down in to horizontal segments can make the transition to responsiveness relatively easy.

Think of Bootstrap's grid system.

Units:

- em
- rem
- percentages

- vw
- vh
- vmin
- vmax

- **Em** -- Historically, this is the width of a capital 'M' in a typeface, now 1 em means the current font-size in the current element. If you have not set a font size, it will use the browser's default size (16px).
- Why use ems?
  - They make it simple to create responsive sizes.
  - They can make ratios simpler to understand.
  - They are very friendly to browsers (compatible back to ES6).
- Downsides?
  - They cascade terribly. Enter: rem.

- **Rem** -- Rem behaves exactly like Em, except it is relative to the root element (the HTML element), meaning you do not run into the issue of Ems multiplying each other.

- **Percentages** -- Relative to the width, height (if set) and font-size of the parent element.

New CS3 values:

- vh - The view height of the viewport.
- vw - Viewport width.
- vmin - The smallest value of either the vh or vw.
- vmax - The largest value between vh & vw.

Media Queries:

A syntax for attaching styles based on certain conditions (window width, viewing device etc).

```
@media (min-width: 700px) { ... }
@media (min-width: 700px) and (orientation: landscape) { ... }
@media (min-width: 700px), handheld and (orientation: landscape) { ... }
@media only screen and (color) { ... }
@media (not (tv)) { ... }
```

The future of Responsive Web Development:

Responsive web development is here to stay, as technology changes, and with a greater number of devices with which to browse and interact with the web, it is something you will need to know.

The good news is, its foundations are things that you already know.

It will take time to learn, and there is more to follow, but there are a lot of great resources.

Advanced CSS:

Vendor Prefixing:

Vendor prefixing is used to access features that are not yet standard and still running through their experimental stages. They are based on the drafts of new versions of CSS.

Vendor prefixing looks like this:

```
div {
 -webkit-transition: all 4s;
 -moz-transition: all 4s;
 -ms-transition: all 4s;
 -o-transition: all 4s;
 transition: all 4s;
}
```

Each of those properties refers to handlers for different versions of web browsers, each one is looking for one of those values to work with. If you are uncertain about whether a browser will support a feature, you can always check here. There's also this Emmet guide which may make your life easier.

Other new CS3 stuff:

Box shadows:

Box shadows can be used to add shadows to virtually any element. If you want to add multiple shadows, they must be chained. **Structure**:

```
box-shadow: offset-x | offset-y | blur radius | spread radius | color
```

```
blur-radius | spread-radius | color;
div {
box-shadow: 2px 2px 2px 2px;
}
```

Text shadows:

... Like box-shadow... But for text!

```
h1 {
 text-shadow: 2px 2px 2px 2px hotpink;
}
```

Transitions:

The CSS property transition is a shorthand property for a bunch of other things:

- Transition-property
- Transition-duration
- Transition-timing-function
- Transition-delay

In action:

```
div {
  transition: all 0.5s;
  transition: width 0.2s, background 0.3s;
  transition: margin-left 4s linear 1s;
}
```

Animations:

Animations are a 2 step process.

- Firstly, define your animation:

  ```
  @keyframes fade-in-and-out {
  0% { opacity: 0; }
  100% { opacity: 1; }
  }
  ```

- Secondly, add it to your element:

  ```
  div {
    animation: fade-in-and-out 5s infinite;
  }
  ```

Together:

```
@keyframes just-keep-spinning {
 0% { transform: rotate(0deg); }
 100% { transform: rotate(360deg); }
}
div {
 animation: just-keep-spinning 3s infinite linear;
}
```

You can also add multiple animations by chaining them together with commas.

Here are a few things that can make generating these effects easier:

- Bounce.js
- AnimateCss
- CssMatic generators

There are plenty of other libraries out there to help you with these things, not limited to Bootstrap, Materialize or BassCSS.

There are also preprocessors (also post) which can help you get even more power from CSS3. Some of these include Sass and Less.

Homework:

Portfolios:

- Personal
- Class

Week 10, day 02

What we covered today:

- Warmup
  - solution
- jQuery Plugins
- Regex
  - Slides

jQuery Plugins

But wait, what is it?

A jQuery plugin is another method that can use to extend jQuery. We do this by attaching functions to the jQuery prototype, which means that anything that inherits from the jQuery prototype will inherit it.

The normal idea of a plugin is to do something to a collection of elements - think .hide or .fadeOut.

You can make your own, so let's figure out how to do that.

The Real Basics of jQuery

Before you write a plugin, you must really understand the basics of jQuery.

```
$( "p" ).css( "color", "red" );
```

This is some relatively basic jQuery code, but what is actually happening?

- Whenever we use the $ function (or the jQuery function) to select elements, it returns a jQuery object.
  - This inherits from the jQuery prototype which in turn means that it (the prototype) contains all the methods.
  - If we want to add jQuery object methods, we need to attach them to the jQuery prototype ($.fn).

Let's see an example

```
// Create a new function on the jQuery prototype called greenify.
$.fn.greenify = function() {

  // Select the element(s) that this method was called upon.
  //  Notice we don't have to select this again with jQuery - like $(this) - that is d
one automatically.
  this.css( "color", "green" );

  // Return the selected element(s) so that we can chain methods.
  return this;
};

$( "a" ).greenify(); // Makes all the links green.
```

Lots of additional stuff to learn with this sort of stuff - see here.

REGEX

Metacharacters

In regular expressions, there are characters that mean far more than literal characters. This can prove problematic (but there are ways to escape metacharacters), but also the source of some of the most powerful parts.

Character Classes

These are often used to check for capitalized and uncapitalized versions, but can also be used to check for multiple letters. The metacharacters for these are `[]`.

```
"bob" =~ /[Bb]ob/  # Returns 0
"Bob" =~ /[Bb]ob/  # Returns 0
"cob" =~ /[Bbc]ob/ # Returns 0
"dog" =~ /[Bb]ob/  # Returns nil
```

For this sort of stuff, we can also use ranges. These are quite useful.

```
"A" =~ /[A-Z]/      # Returns 0
"a" =~ /[A-Z]/      # Returns nil
"a" =~ /[A-z]/      # Returns 0
```

There are lots of inbuilt ranges, and these are really useful to know. Some of them are...

- `\s` - Will match any space characters (spaces, new lines etc.)
- `\S` - Anything other than space characters
- `\w` - Will match any word character (i.e. actual letters or numbers)

- `\W` - Will match any non-word character

```
"Wolf" =~ /[\w]/    # Returns 0
"Wolf" =~ /[\W]/    # Returns nil
"Wolf " =~ /[\W]/   # Returns 4

"Wolf " =~ /[\s]/   # Returns 4
"Wolf " =~ /[\S]/   # Returns 0
```

We can obviously add lots and lots of things between those square brackets. But there are other ways we can do this as well. We can use `()` and `|` to check for multiple things.

```
"Jane" =~ /(Jane|Serge)/    # Returns 0
"Serge" =~ /(Jane|Serge)/   # Returns 0
```

The round brackets are metacharacters in Regexp as well! They are a way to say this or this (or this or this or this). The pipe is the delimiter for saying "or".

There are a lot more metacharacters. The `.`, for example, is a wildcard, it will match anything.

```
"jane" =~ /.ane/            # Returns 0
"zane" =~ /.ane/            # Returns 0
"Serge and Jane" =~ /.ane/  # Returns 10
```

This is still relatively hardcoded though, we need to specify where the characters are and what they are. To help solve this problem, there are quantifiers.

Quantifiers

Quantifiers are a way to check in Regexp whether things exist, or exist more than once etc.

The ones that you will actually use:

- `+` means one or more
- `?` means one or zero
- `*` means zero or more

It can look like the following:

```
"Hi there" =~ /i+/      # Returns 1
"Hi there" =~ /the?/    # Returns 3
"Hi theeere" =~ /the*/  # Returns 3
```

These are regularly used for existence checks.

Capturers

These are difficult to understand. Basically, you match something in parentheses and can refer back to them. You capture something by using round brackets, and refer back to them using a `\` and an integer (that mimics the order of capture).

```
"WolfWolf" =~ /(....)\1/
# Matches any four characters, but then needs to have the same four characters straigh
t after.  This returns zero.

"ArcticWolf ArcticWolf" =~ /(......)(....) \1\2/
# Matches "Arctic" in the first brackets, then "Wolf" in the second brackets.  "Arctic
" is saved as \1 and "Wolf" is saved as \2
```

A few resources related to Regular Expressions...

- Ruby Docs
- MDN
- Regex One
- Learn Code the Hard Way
- Regex Crossword
- Rubular
- See this Gist

Other things you might like to look at:

- https://gist.github.com/wofockham/9fb80a265a60c865b308
- https://toddmotto.com/everything-you-wanted-to-know-about-javascript-scope/
- https://robots.thoughtbot.com/back-to-basics-anonymous-functions-and-closures

Homework:

Finish your custom jQuery plugins, then finish off this.

If you get all that done, look at this

Week 10, Day 03

What we covered today:

Rspec

We have done a little bit of Rspec now, but it has all been with pure Ruby, no Rails. We want to get it working with Rails, and luckily, that isn't difficult.

We need to include the gem...

`gem 'rspec-rails'` - and make sure it goes in the development and test environments!

Next we ant to generate a model to actually test, so we run `rails g model Contact first_name:text last_name:text email:text`

We then run `bundle` and `rake db:migrate`.

This still hasn't installed Rspec, so we need to do that... `rails generate rspec:install` or just `rails g rspec:install`. This does a bunch of things, it generates us a spec folder (where all of our tests will live), but it also changes the behaviour for other rails generators. It will add functionality. For example, when you generate a model, it will now generate a test for it as well! Same thing for controllers etc.

From here we want to make a few more changes before we get started:

In `.rspec` we want to add the following line at the end of the file: `--format=documentation` this tells rspec to be verbose, to tell us more about the errors it actually encounters.

From here we can actually start testing our models. Using the red/green factoring cycle, we will be writing tests that fail and then begin trying to get them to pass.

Our first test is in `contact_spec.rb`. Firstly, we tell it what we expect it to do:

```
RSpec.describe Contact, type: :model do
  it "is valid with a first_name, last_name & email"
```

If we run rspec from the terminal, we get the following:

```
1) Contact is valid with a first_name, last_name & email
    # Not yet implemented
    # ./spec/models/contact_spec.rb:4

    Finished in 0.00049 seconds (files took 1.75 seconds to load)
    1 example, 0 failures, 1 pending
```

Next we specify the conditions we want our test to meet, in this case we want to make sure we can make our models and that they are valid when all the information is supplied for them.

```
RSpec.describe Contact, type: :model do
  it "is valid with a first_name, last_name & email" do
    contact = Contact.new( first_name: "Tom",
                           last_name: "Selleck",
                           email: "Tom@GA.co" )
    expect(contact).to be_valid
  end
```

And when we run our rspec again:

```
Contact
  is valid with a first_name, last_name & email

Finished in 0.09143 seconds (files took 1.66 seconds to load)
1 example, 0 failures
```

Success! We have a passing test.

Our next test is to test for an invalid entry, we want to exclude bad data.

```
it "is invalid with no last_name" do
  contact = Contact.new( first_name: "Tom",
                         last_name: nil,
                         email: "Tom@GA.co" )
  expect(contact).to be_invalid
end
```

However, if we run that - we hit a failing test:

```
1) Contact is invalid with no last_name
     Failure/Error: expect(contact).to be_invalid
       expected `#<Contact id: nil, first_name: "Bill", last_name: nil, email: "Bill@G
A.co", created_at: nil, updated_at: nil>.invalid?` to return true, got false
```

In order to ensure our model actually has values on creation, we can add `# validates :last_name, presence: true` to our `contact.rb`.

This once again gets our tests passing. There are a lot more validations and tests to be made, it is well worth looking through the documentation.

Faker and Factory Girl

Faker and Factory Girl are automators for testing databases normally. They make generation of things like users easy and A basic factory may look like the following...

```
FactoryGirl.define do
  factory :user do
    name "Craigsy"
    email "craigsy@hotmail.com"
  end
end
```

If we run `FactoryGirl.create( :user )`, it will actually create a new user and put it in the database. If we wanted to create lots at a time, we could do the following... `FactoryGirl.create_list( :user, 10 )` and that will make 10 users (but they all have the same name!).

Now, this is where Faker comes in. Worth having a look at the docs for that, but there are lots of things that it helps with.

Some common uses of Faker.

```
Faker::Name.name
Faker::Internet.email

Faker::Company.name
Faker::Company.catch_phrase # Very good

Faker::Date.between(2.days.ago, Date.today)
Faker::Date.forward(23) # Up to 23 days away
Faker::Date.backward(14) # No more than 14 days ago

Faker::Lorem.paragraphs
```

Everytime you run this, it will generate a random thing of some description, so you think it would be relatively straight forward to create unique users. Maybe something like this?

```
FactoryGirl.define do
  factory :user do
    name Faker::Name.name
    email Fake::Internet.email
  end
end
```

We can still run this using `FactoryGirl.create_list( :user, 50 )` but they are still all the same! We solve this problem by using sequences, this means that they get evaluated every time.

```
FactoryGirl.define do
  factory :user do |f|
    f.sequence(:name) { Faker::Name.name }
    f.sequence(:email) { Faker::Internet.email }
  end
end
```

That will do exactly what we want. But what if we wanted to make some tweets that are associated with a particular user? We can do that too!! (Assuming we have the right associations).

```
FactoryGirl.define do
  factory :user do |f|
    f.sequence(:name) { Faker::Name.name }
    f.sequence(:email) { Faker::Internet.email }

    factory :user_with_tweets do
      after(:create) do |u|
        FactoryGirl.create_list(:tweet, Random.rand(10..100), :user => u)
      end
    end
  end

  factory :tweet do |f|
    f.sequence(:content) { Faker::Lorem.sentence }
  end
end
```

To run this, and this seems weird, we run `FactoryGirl.create_list( :user_with_tweets, 20 )`. This will create lots of tweets for lots of users and associate them correctly.

Homework:

https://gist.github.com/ga-wolf/24cc1f7fdc9d0e14d4dfc15527303ba0

Heroku

What is it?

Github only supports static front-end sites, so we need something more robust and scalable for our Rails apps. Heroku has everything that we need for this:

- Dynos - run virtually any language
- Database - always postgres
- Tools and components
- Connects with heaps of other applications (Salesforce etc.)
- Good support

See here for more.

Heroku is completely based on Git, meaning that it relies on us pushing code up etc. It can be difficult, though it is ten times easier than making our own server.

Steps for Deploying a Rails app to Heroku

- Make an account - here.
- Install the heroku toolbelt - see here or use `brew install heroku`
- Make sure that it has worked
  - `heroku --version`
  - `which heroku`
- Log in - `heroku login`
- Make your Rails app
  - 'Deploy Early, Deploy Often'
  - Set the app up with PostgreSQL
    - `rails new app_name --database=postgresql`
    - `rails new app_name -d postgresql`
- Change into the app directory and run `bundle install` to install all Gems and dependencies
- At the end of the Gemfile, add this line - `gem 'rails_12factor', group: :production` and then run `bundle install` again
- Make it a git repository
  - Run `git init` in the root folder of the project
  - `git add .`, `git commit -m "Your commit message."`

    - Run `git status` to make sure everything has worked
- Run `heroku create`
- Run `git config --list | grep heroku` to make sure it has worked
- Run `git push heroku master`
- Run `heroku run rake db:migrate`
    - Any command prefixed by `heroku run` will obviously run on the heroku terminal
- Ensure that you have a running server, run `heroku ps:scale web=1`
- Run `heroku ps` to make sure that that has all worked
- Run `heroku open` to do exactly what you imagine

For this in far more detail, see here.

Common Heroku Commands

- `heroku run rake db:etc.` (create, seed, migrate etc.)
- `heroku run rails console`
- `heroku logs` - Shows the rails server (once off)
- `heroku logs --tail` - Shows the live server

Cloudinary

What is it?

Cloudinary is an image and video management system in the cloud. It includes:

- Storage
- Upload
- "Powerful Administration"
- Image Manipulation
- A fast CDN

It is used by companies like Gizmodo, GQ, Answers, Redbull, Ebay etc.

See here for more.

It's also quite simple to set up so let's get that going.

How to get it up and running

First things first, sign up for an account here.

- Let's set up a new rails project - making sure to start it off with postgresql (just as a habit) - `rails new cloudinary-test -d postgresql`
- We will use some Rails generators to make all our migrations etc. - `rails generate model Animal name:string image:string`
- We then need to run `rake db:create` and `rake db:migrate`
- We need to add `gem 'cloudinary'` to our Gemfile
- Then we need to run `bundle`
- Because Cloudinary is an API, we need to identify ourselves with it. Luckily they have a file that we can download that will do this for us - go here (if it doesn't do anything, make sure you are logged in).
    - This will download a file called `cloudinary.yml` - we need to save that in the config folder. This will be run everytime your server gets run
- Let's change our Routes ...

```
Rails.application.routes.draw do
  root 'animals#index'
  get 'animals/new'
  post 'animals' => 'animals#create'
end
```

- Let's create some views for us to work with - `rails g controller Animals index new`

```
def index
    @animals = Animal.all
end

def new
    @animal = Animal.new
end
```

- And then make those views
- `new.html.erb`

```
<%= form_tag @animal, url: "/animals", multipart: true do %>
  <%= label_tag "Animal Name: " %>
  <%= text_field_tag :name %>

  <br />
  <br />
  <%= label_tag "Animal Image: " %>
  <%= cl_image_upload_tag :avatar %>

  <br />
  <br />
  <%= submit_tag "Create Animal." %>
<% end %>
```

- In this form, we have said that we are posting it to /animals, so we need to make sure our routes have something for that - `post 'animals' => 'animals#create'`
- In our create method, it should look something like this:

```
req = Cloudinary::Uploader.upload( params[:file] )
@animal = Animal.create( name: params[:name], avatar: req["url"] )
```

- We have an error, and that is because we haven't given Cloudinary the appropriate details
- Add this into the head tag of your application.html.erb - `<%= cloudinary_js_config %>`.

This will append your form with a bunch of cloudinary related configuration things.

Underscore.js

Important links for these slides

- These slides
- First Exercises
- Second Exercises
- Third Exercises
- Fourth Exercises
- Fifth Exercises

What is it?

Homepage

> "Underscore is a JavaScript library that provides a whole mess of useful functional programming helpers"
>
> "It's the answer to the question: "If I sit down in front of a blank HTML page, and want to start being productive immediately, what do I need?"

It's a utility belt, over hundred functions that will help you get stuff done.

Who built it?

- Jeremy Ashkenas (@jashkenas)
  - Twitter
  - Github
- He also built:
  - Backbone
  - Coffeescript

Why do we teach it?

- It's incredibly useful
- Brings a lot of Ruby's style and functionality across (as does Coffeescript)

- It is a dependency of Backbone (with one small caveat)
  - There is another library called `lo-dash`
  - It is a fork of underscore and is a direct replacement
- Saves you from repeating code

What's the approach?

- Just like jQuery has the `$`, Underscore uses the `_`
- All of it's functions are scoped, or nested, within that `_`
- Relies heavily on **predicate** methods - ones that always return `true` or `false`

```
_.each( /* ... */ );
_.sortBy( /* ... */ );
_.where( /* ... */);
```

What's the approach?

It breaks the functions down into six categories:

- Collections
- Arrays
- Objects
- Utilities
- Functions
- Chaining

Because there are over a hundred of these things, we aren't going to go through them all.

Let's look at some of them though!

How do we use it?

Reference it just like any other Javascript library!

- Download the file || Get it from the CDN || Or use a gem
- Make sure you reference it before any JS that uses it!

e.g.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.3/underscore-min
.js"></script>
```

Collections

_.each()

```
// Works on arrays...

_.each( [1, 2, 3], function (num) {
  console.log( num );
});

_.each( [1, 2, 3], console.log );

// Works on objects!

_.each( { one: 1, two: 2, three: 3 }, function ( value, key ) {
  console.log( value );
});
```

Collections

.map(), .reduce()

```
// Works on arrays...

_.map( [1, 2, 3], function (num) {
  return num * 3;
});

// Works on objects... but returns an array

_.map( { one: 1, two: 2, three: 3 }, function ( value, key ) {
  return value * 3;
});

_.reduce( [1, 2, 3], function (sum, num) {
  return sum + num;
}, 0 ); // => 6

// reduce is an alias for inject
```

Collections

.where(), .findWhere(), *.filter(),* .reject(), _.find()

```
var data = [
  { id: 22, username: "Martin", active: true },
  { id: 23, username: "Max",    active: false},
  { id: 24, username: "Linda",  active: false}
];

_.where( data, { active: false } );     // => [{ id: 23, ... }, { id: 24, ... }]
_.findWhere( data, { active: false } ); // => { id: 23, ... }

var nums = [ 1, 2, 3, 4, 5, 6 ];

_.filter( nums, function ( num ) {
  return num % 2 === 0;
}); // => [ 2, 4, 6 ] - _.find will return the first one of this

_.reject( nums, function ( num ) {
  return num % 2 === 0;
}); // => [ 1, 3, 5 ]
```

Exercise Time!

Here they are!

Collections

_.sortBy(), .groupBy()

```
var stooges = [
  {name: 'moe', age: 40},
  {name: 'larry', age: 50},
  {name: 'curly', age: 60}
];
_.sortBy(stooges, 'name'); // => [{/* Curly */}, {/* Larry */, {/* Moe */}]

_.groupBy( [1.3, 2.1, 2.4], function(num) {
  return Math.floor(num);
}); // => { 1: [1.3], 2: [2.1, 2.4] }
```

Collections

.every(), .some(), _.contains()

```
var data = [ 1, 2, 3, 4, 5 ];

_.every( data, function ( num ) {
  return num % 2 === 0;
}); // => false

_.some( data, function ( num ) {
  return num % 2 === 0;
}); // => true

_.contains( data, 3 ); // => true
```

Collections

.pluck(), .max(), _.min()

```
var stooges = [
  {name: 'moe', age: 40},
  {name: 'larry', age: 50},
  {name: 'curly', age: 60}
];

_.pluck( stooges, 'name' ); // => [ 'moe', 'larry', 'curly' ]

_.max( stooges, 'age' ); // => 60
_.min( stooges, 'age' ); // => 40
```

Collections

.countBy(), .shuffle(), *.sample(),* .size()

```
var data = [ 1, 2, 3, 4, 5 ];

_.shuffle( data ); // => [ 3, 2, 5, 4, 1 ] (random every time)
_.size( data ); // => 5
_.sample( data ); // => 3 (random every time)
_.sample( data, 3 ); //=> [ 3, 5, 1 ] (random every time)

_.countBy([1, 2, 3, 4, 5], function(num) {
  return num % 2 == 0 ? 'even': 'odd';
}); // => { odd: 3, even: 2 }
```

Exercise Time!

Here they are!

Arrays

.first(), .last(), *.initial(),* .rest()

.compact(), .flatten()

- `_.first()` and `_.last()` do exactly what you'd expect
- `_.initial()` returns everything except the last element(s)
- `_.rest()` returns everything except the first element(s)

```
_.compact([0, 1, false, 2, '', 3]); // => [1, 2, 3]

_.flatten([1, [2], [3, [[4]]]]); // => [1, 2, 3, 4];
_.flatten([1, [2], [3, [[4]]]], true); // => [1, 2, 3, [[4]]];
```

Arrays

.without(), .union(), *.intersection(),* .difference(), _.uniq()

```
_.without([1, 2, 1, 0, 3, 1, 4], 0, 1);
// => [2, 3, 4]

_.union([1, 2, 3], [101, 2, 1, 10], [2, 1]);
// => [1, 2, 3, 101, 10] - all unique items

_.intersection([1, 2, 3], [101, 2, 1, 10], [2, 1]);
// => [1, 2] - all items that are present in all arrays

_.difference([1, 2, 3, 4, 5], [5, 2, 10]);
// => [1, 3, 4] - all items that are present in the first array and nowhere else
```

Arrays

.zip(), .unzip(), _.object()

```
_.zip(['moe', 'larry', 'curly'], [30, 40, 50], [true, false, false]);
// => [["moe", 30, true], ["larry", 40, false], ["curly", 50, false]]

_.unzip([['moe', 30, true], ["larry", 40, false], ["curly", 50, false]])
// => [['moe', 'larry', 'curly'], [30, 40, 50], [true, false, false]]

_.object(['moe', 'larry', 'curly'], [30, 40, 50]);
// => {moe: 30, larry: 40, curly: 50}

_.object([['moe', 30], ['larry', 40], ['curly', 50]]);
// => {moe: 30, larry: 40, curly: 50}
```

Arrays

.indexOf(), .lastIndexOf(), *.findIndex(),* .findLastIndex(), _.sortedIndex()

- `_.indexOf()` and `_.lastIndexOf()` find a value and return it's index (or -1)
- `_.findIndex()` returns the index of the first thing that passes a predicate function

```
var stooges = [{name: 'moe', age: 40}, {name: 'curly', age: 60}];
_.sortedIndex(stooges, {name: 'larry', age: 50}, 'age'); // => 1

// _.sortedIndex() tells you where you should insert an element
```

Arrays

_.range()

```
_.range(10);
// => [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

_.range(1, 11);
// => [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

_.range(0, 30, 5);
// => [0, 5, 10, 15, 20, 25]

_.range(0, -10, -1);
// => [0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

Exercise Time!

Here they are!

Objects

.keys(), .mapObject(), *.pairs(),* .invert()

```
var data = { name: "Unknown", location: "Unknown" };
_.keys( data ); // => [ "name", "location" ]

var data = { start: 100, end: 200 };
_.mapObject( data, function (value, key) {
  return value * 2;
}); // => { start: 200, end: 400 }

_.pairs(data); // => [["start", 100], ["end", 200]]

_.invert( data ); // => { "100" : "start", "200" : "end" }
```

Objects

.pick(), .omit(), *.defaults(),* .has(), *.isEqual(),* .isMatch()

```
var data = { name: "N/A", location: "N/A", description: "N/A" };

_.pick( data, "name", "location" ); // => { name: "N/A", location: "N/A" }

_.omit( data, "description" ); // => { name: "N/A", location: "N/A" }

_.defaults( { name: "N/A" }, { drinksWater: true } ); // => { name: "N/A", drinksWater
: true }

_.has( { name: "N/A" }, "name" ); // => true

_.isEqual( { name: "N/A" }, { name: "N/A" } );

_.isMatch( { name: "N/A", location: "N/A" }, { location: "N/A" } ); // => true
```

Objects

_.is**()

- isEmpty
- isElement
- isArray
- isObject
- isArguments
- isFunction
- isString
- isNumber
- isFinite
- isBoolean
- isDate
- isRegExp
- isNaN
- isNull
- isUndefined

Exercise Time!

Here they are!

Utilities

.times(), .random(), *.escape(),* .unescape()

```
_.times( 3, function () { console.log( "Hi" ); });

_.random(0, 100); // This is inclusive!

_.escape('Curly, Larry & Moe');
// => "Curly, Larry &amp;amp; Moe"

_.unescape('Curly, Larry &amp;amp; Moe');
// => "Curly, Larry & Moe"
```

Utilities

.now(), .template()

```
_.now() // Returns a timestamp of the current time

var templateString = "<p> Hello <%= name %>! </p>";
var template = _.template( templateString );
var compiledTemplate = template( { name: "Jane" } );
```

Functions

.delay(), .once()

```
_.delay(function () { console.log("Hi"); }, 1000);

var createApplication = function () {
  console.log("Hi");
}

var initialize = _.once(createApplication);
initialize();
initialize();
```

Functions

.throttle(), .debounce()

Very weird functions.

- Throttling enforces a maximum number of times a function can be called. As in execute this function at most once every 100 milliseconds.
- Debouncing enforces that a function not be called again until a certain amount of time has passed without it being called.

Use cases:

- Infinite Scroll
- Double Clicks - see here
  - Throttling won't filter double clicks
  - Debouncing will

See here for a visual representation.

Functions

.throttle(), .debounce()

```
var throttled = _.throttle(updatePosition, 100);
$(window).scroll( throttled );

var lazyLayout = _.debounce(calculateLayout, 300);
$(window).resize(lazyLayout);
```

Chaining

- Two approaches

```
_( [1, 2, 3] ).map( function (num) {
  return num * 3;
}); // => [ 3, 6, 9 ]
// Only works for one additional method!

_.chain( ["one", "two", "three"] )
 .map( function (word) {
   return word + " mapped";
 })
 .map( function (word) {
   return word.toUpperCase();
 }); // => [ "ONE MAPPED", "TWO MAPPED", "THREE MAPPED" ]
 // This approach is much better!
```

That is Underscore!

Don't expect to remember all of that, but do remember that the documentation for Underscore is really good.

Important thing is, just as with Ruby, always see if a function exists that does what you are trying to do!

Here are the final exercises!