G11 Course Curriculum

Michael Herman

Contents

*

Chapter 1

Week 1, Day 1

Welcome!

Today's Goals

- Get to know your instructors.
- Get to know your classmates.
- Introduce habits and mindsets that improve your ability to learn.
- Describe what it takes to be a kind, courteous Galvanize member.
- Setup hardware.
- Take a headshot.
- Get financials in order.
- Setup dev environment.
- Develop metacognition about learning and teaching

Schedule

8:30am

- Mingle - say "Hi"
- Breakfast provided by Galvanize

9am

- "Welcome" - Michael
- Staff Intros -Michael, Zach, Reyna, Casy, Lydia, Joanna, Evan, Kelly, James
- Student Testimonial/Advice - Kenneth To (from G6)

9:30am

- Icebreakers!

11am

- “Learn to Learn” with Evan

12pm

- Lunch! (provided by Galvanize)

1pm

Divide into groups...

- Computers (network access, download Chrome, and Atom) with Michael
- Joanna for money/contracts
- Photos with Reyna
- Tours/Fob with Casy/Kelly/Matt
- Outcomes with James
- Speed Pong with Zach

3pm

- Computer setup - Homebrew, iterm2 and zsh, Node and Git (via brew), Spectacle

4:30pm

- Wrap up
- Happy Hour

Course Overview

Link

What makes a good developer?

- Resourcefulness - http://whathaveyoutried.com
- Passion

Advice from previous gSchool students

- Ask a lot of questions
- Check your ego at the door
- Help others and don't be afraid to ask for help EVER!
- get comfortable with feeling uncomfortable/lost/confused
- Write a list describing the kind of developer you want to be and what you want to be an expert at(sometimes it helps to look at your dream jobs role descriptions). Make sure that you set aside a little bit of time every week to work on one small thing on the side that addresses that(whether it's reading about CSS mark-up or making a miniapp using a non-relational database, doing a one hour tutorial on how to use redis etc).
- my advice would be to not be intimidated, everyone starts somewhere. People that seem to know it all, knew little at first.
- it's important to actually do the homework. It helps to retain new information and gives you a chance to practice!
- when you run into a small bug that you think you can solve using trial and error, fire up a debugger instead and try to solve it in one shot.
- "the problem isn't solving the problem, it's figuring out how to finding the answer."
- Get sleep and take consistent breaks, ESPECIALLY if you feel like you're in a rut. You'll often think of a solution while you're taking a break, exercising, or waking up from a reasonable night's sleep.
- Don't just focus on learning syntax but make sure you understand the concepts of how mvc frameworks operate and pay attention to high level concepts. I would also throw in there to make sure to actively practice communication since lots of programmers do not know how to communicate or relate with people.

Homework

- Finish the pre-work!

Chapter 2

Week 1, Day 2

Today's Goals

- Agree on classroom norms.
- Acknowledge graduation criteria.
- Become familiar with the basic terminal commands
- Demonstrate the basic git commands/workflow
- Understand what a Version Control System (VCS) is
- Understand the difference between git and Github
- Explain how UNIX commands work.
  - Identify the parts of a UNIX command and explain the purpose of each. (command)
  - Discuss the UNIX philosophy (small programs that do one thing well, can be chained together).
  - Describe how to stop a running process with CTL+C or CTL+D
  - Learn how to quickly get help on a command without leaving the comfort of the terminal - COMMAND –help or COMMAND -h
- Navigate the file system using the command line.
- Explain how the mac file system is set up like a tree, with / at the root.
- Use pwd to print your working directory.
- Use cd to change your working directory.
- Explain the difference between absolute and relative paths.
- Explain . and ..
- Navigate to a specific location using both absolute and relative file paths.
- Discuss what types of files are in the home and root directories. (your files vs. system files)
- Use cd ~ to jump home.
- Use TAB to autocomplete paths
- Use up arrow to view previous commands
- Manage files using the command line.
- Chain and Pipe commands to work more efficiently.

9am Standup

- Events, Interestings, Happenings

9:15am Warm-up

- Type racer

10am Daily/Weekly Schedule and Expectations/Attendance

- Expectations/Schedule with Michael
- Daily/Weekly Schedule

Daily

- 9 to 9:30 - standup and warmup
- 9:30 to 12 - class
- 12 to 1 - lunch
- 1 to 5 class

Weekly

1. Code Reviews - Thursday
2. Lighting Talks - Friday
3. Interview/Career Prep - Friday
4. Weekly re-cap - Friday

Attendance

DO NOT miss more than 6 days!

10:30am Terminal Commands

Quick Review with Zach

How do UNIX commands work?

- $ command <optional flags>
- −h or −−help
- Stop a running process
- Chaining commands

Navigating the file system

- pwd, cd, cd ~, cd .., ls −a
- Absolute vs relative paths
- TAB autocomplete
- UP arrows

Managing Files

- mv move vs rename
- cp
- touch
- rm

11am

Command Line Hard Way (first 13 chapters)

12pm Lunch

Eat food (if that's your thing).

1pm Git

With Reyna

- Cheatsheet: Git Essentials
- Git Assessment

Git = Version Control

Version control is a developer's safety net. If you're proficient at version control, you don't have to worry about "breaking your app" because you can always revert to a previous state. This means that you can take risks and play more freely when you're trying something new! Version control is what we call a "learning multiplier". It helps you behave in a way that is conducive to learning as quickly as possible. Move fast and break things!

Watch What is VCS?

Introduction to Git, GitHub

- git init
- git status
- git add –A
- git commit –m "This is a commit message"
- git log
- git remote add <remote name> <SSH URL>
- git remote –v
- git push <remote name> <branch>

Git Config

Add your name, email and a gitignore file to your global Git Configuration

- git config ––global user.name "<Your Name>"
- git config ––global user.email "<your_email@example.com>"
- git config ––global core.excludesfile ~/.gitignore_global
- git config ––global –l

We will add some things to be ignored by git in our gitignore file.

- atom ~/.gitignore_global
- Add this line and save:
- .DS_Store
- junk

Concepts

1. Git vs. Github
2. Git States - working directory, staging, Git Repo

Feature Branch Workflow (time permitting)

1. https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow

Bonus!

- Play: Try Git (challenges 1-8)
- Command Line Mystery
- Try Githug. Go as far as you can before you get stuck.

Homework

1. Finish pre-work and Command Line Hard Way (first 13 chapters)
2. Read: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Introduction
3. Watch: Semantic HTML
4. Watch: Parent Child HTML Elements
5. Watch: Simple Project Structure
6. Ask yourself: What is CSS? Why is it important?

Chapter 3

Week 1, Day 3

Today's Goals

- Review how to use HTML to build a webpage
- Understand the difference between HTML tags, elements, and attributes
- Be able to articulate the difference between HTML syntax and semantics
- Build a simple webpage using semantic markup

  NOTE: We will just be going over the very basics of HTML and CSS; we will take a much deeper dive later in the cohort.

9am Standup

- Events, Helps, Happenings

9:15am Warm-up

- Type racer (Robbie vs ??)

9:30am How to Submit Assignments/Student Site Overview

- Gist
- With Zach

10am HTML/CSS Exercise

- HTML and CSS Tutorial (only do the HTML part!)

12pm Lunch

Go eat.

1pm HTML and CSS - primer

With Michael or Zach or Reyna

HTML Boilerplate

Build a basic site, starting with-

```
<!— Defines the html version (for the browser) —>
<!DOCTYPE html>

<!— Sets up the html element and defines the language to use —>
<html lang="en">

  <!— Meta information about our html document —>
  <head>

    <!— Defines the character set to use in the document —>
    <meta charset="utf-8">

    <!— Sets the title of the page —>
    <title>Awesome Title</title>

    <!— Pull in stylesheet normalization for cross-browser support —>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/normalize/3.

    <!— Pull in our own custom css file —>
    <link rel="stylesheet" href="main.css">

  </head>

  <!— Defines the page content - e.g., what the end user sees —>
  <body>

    <!— A header tag and element —>
    <h1>This is an Awesome header</h1>

    <!— Use classes to identify unique-ish elements —>
    <h2 class="header-alt">Another header</h2>
```

```
    <h3 class="header-alt">Sub header</h3>

    <!-- Use an ID to identify actual unique elements -->
    <h4 id="header-primary">This is an h4</h4>

    <!-- Hierarchy -->
    <div>
      <p>(INSIDE) Lorem ipsum dolor sit amet, consectetur adipisicing elit.</p>
    </div>

    <p>(OUTSIDE) Adipisci vero quae, a possimus impedit praesentium eaque beatae mag

  </body>
</html>
```

What is HTML?

- HTML *describes* and applies *structure* to a page; it's the skeleton.
- Browsers *parse* and then *render* the HTML so that it's human-readable.

Tags Make Elements

HTML tags are used to wrap content - generally text.

```
<p>Some text.</p>
```

The tags above are & .

When tags are used, they form an HTML element. The use of tags p tags surrounding text above creates an HTML element.

Self-closing or void tags

Some tags are self-closing and do no need to wrap content.

```
<img src="foo.com/thing.png">

<input value="Input␣Here!">

<hr>
```

Attributes

elements often have attributes to generate content or to act as a reference for other technologies like CSS & JS.

This image has a source and alt attribute.

```
<img src="foo.com/thing.png" alt="A cool image of a foo thing.">
```

This anchor link has an href attribute.

```
<a href="/contact">Contact Page</a>
```

This title has a class attribute.

```
<h1 class="content-title">You Will Never Guess What This Puppy Does With Her Brunch.
```

Building Blocks

1. html
2. head
3. body
4. script
5. link
6. title
7. meta
8. style

Media

1. img
2. video
3. audio

Typography

1. p
2. h1 - h6
3. a
4. em
5. strong
6. small
7. blockquote
8. ul
9. li
10. ol

Layout

1. main
2. section
3. article
4. header
5. footer
6. hr
7. nav
8. div

Semantic Markup

1. article
2. aside
3. header
4. footer
5. nav
6. main
7. section

CSS

CSS = Cascading Style Sheets. Why?

```
/* Select by tagname */
h1 {
  color: red;
}

/* Select by classname */
.header-alt {
  border-bottom: 2px solid red;
}

/* Select a single element by its ID */
#header-primary {
  text-transform: uppercase;
}

/* Select all H1s AND all .header-alts */
h1,
.header-alt {
  border-top: 2px solid blue;
```

```
}

/* Use descendent selectors */
div p {
  font-style: italic;
}

/* Target all p's and a's inside of divs */
div p, div a {}
```

2pm Exercises

- Mock Ups
- Card Flip

4pm Website Presentation

- Present your personal site in groups! (start with the finished product, then highlight the code)

Bonus!

- Dive Into HTML5

Homework

- Refactor your personal website to use **semantic markup** (push to Github)
- Read Introduction and Chapter 1 in Eloquent JS

Chapter 4

Week 1, Day 4

Today's Goals

1. Start *loving* JavaScript!

- Understand the JavaScript primitives - numbers, strings, booleans, null, undefined
- Understand the difference between a value and an expression
- Utilize JS primitives, expressions, and variables in action
- Write and evaluate grammatical JS statements featuring variables, data types, operators, and values.

1. Learn how to get quick feedback from the JS Console

9am Standup

- Events, Helps, Happenings

9:15am Warm-up

- Type racer

9:30 - Javascript Primer Part 1

With Zach

1. History of Javascript

2. Why Should YOU Care About Javascript?
3. DevTools/JavaScript Console
4. Comments
5. Numbers
6. Strings
7. Booleans
8. Values and Expressions
9. Special Number Operators
10. Variables
11. undefined & null

History of Javascript

- Not to be confused with Java
- Created in 10 days in May 1995 by Brendan Eich
- It's an exciting time to learn Javascript! It's the language that enables web pages to respond to user interaction beyond the basic level.
- The language today is viewed quite differently than how it was 10 years ago
- The famous Douglas Crockford gives a thorough introduction of Javascript

Why Should YOU Care About Javascript?

- You can use the same language on the front-end and the back-end with Node.js
- Many popular libraries built with JS - jQuery, Underscore.js, etc.
- Javascript allows us to make our pages interactive and dynamic and awesome. If HTML is the skeleton and CSS is the skin, then JavaScript is the heart.
- http://blog.makersquare.com/2015/05/31/the-top-3-reasons-to-learn-javascript/
- Here are some very common uses for JS on the front-end:
- Password Strength Meter
- Simple Navigation Menu
- Overlay Effects
- Page Loading Effects
- Image Carousel
- Parallax Scroll and Blur
- Some other more complex uses for JS:
- Interactive Music Video
- Jam With Chrome
- Patatap
- Arcade Fire Reflektor Music Video
- Walmart Website

DevTools/Javascript Console

- Allows you to easily interface with your app to run JS commands and display log messages for help with debugging - GET QUICK FEEDBACK!
- Shortcut to open JS console & bring focus to console
- Mac: Cmd + Opt + I
- Use Tab for autocompletion!

Go to Google and try pasting the following code into your console:

```
var logo = document.getElementById('hplogo');
logo.onclick = function () { this.src = "http://cdn.howtogeek.com/wp-content/uploads
```

And then try this:

```
function makeWider() {
  var logo = document.getElementById('hplogo');
  logo.width += 5;
}

setInterval(makeWider, 41.67)
```

Lastly, try this:

```
javascript:document.body.contentEditable='true'; document.designMode='on'; void 0
```

Comments

Comments come in two forms - line comments and multiline comments

```
// descriptive stuff

/*
  These
  are
  comments on
  many lines
*/

  ```

How do you comment out a line in Atom? What does "comment out" even mean?

Numbers

Numbers are one of the *types* of **values** we want to be able to interact and play

Integers

..., -1,0, 2, 3, 4, 5, ...

Floats (or Decimal numbers)

2.718, 3.14, .5, .25

Strings

Strings are collections of letters and symbols known as **Characters**, and we use th

“John”, “Jane”

Booleans

A boolean represents logical values — **true** or **false**

var catsAreGreat = true; var dogsRule = false;

Values and Expressions

Values are the simplest components in JavaScript. ‘‘‘1‘‘‘ is a value, ‘‘‘true‘‘‘ is

Types of values like ‘Number‘ or ‘String‘ are not very useful without being able to

Try your favorite number operators

1 + 1 => 2 2 - 1 => 1

You can also create expressions with strings using addition

“Hello,” + “world!” => “Hello, world!”

This is called **String Concatentation.**

Special Number Operators

Javascript can be a little cheap with the number of operations it allows you to do.

* Taking a number to some ‘power‘? Then just use ‘Math.pow‘

// 3^2 becomes Math.pow(3,2) => 9 // 2^4 becomes Math.pow(2,4) => 16

* Taking a square root

// √(4) becomes Math.sqrt(4) => 2

* Need a ‘random‘ number? Then use ‘Math.random‘.

// The following only returns a random decimal Math.random() => .229375290430 /** The following will return a random number between 0 and 10 */ Math.random()10

* Since Numbers can be **Floats** or **Integers** we often want to get rid of remain

// Remove the decimal Math.floor(3.14) => 3 Math.floor(3.9999) => 3

Variables

Having made some expressions it becomes evident we want to store these values.

To store values we use things called **variables**.

The word ’variable’ means ’can change’ and is used because variables can store many

var myNumber = 1; var myString = “Greetings y’all!”

The main note to make here is that these variables should always have the ‘var‘ keyw

Variables can also store the result of any ”expression”.
For example:

var x = 2 + 2;

or

var name = ‘Momo’; var greeting = ‘Hello’ + name;

undefined & null

undefined: Represents a value that hasn’t been defined

var notDefinedYet;

A variable that has not been assigned a value is of type undefined. A method or state

null: Represents an explicitly empty value

var dogsRule = null;

```
## 12pm Lunch

Food.

## Exercises/Assessments

- Assessment: JavaScript Variables
- Assessment: JavaScript Fundamentals

> In english, we write SENTENCES, composed of WORDS, which have TYPES, (noun, verb,

### Basics

1. Create a blank, valid html5 document.
1. Create a JavaScript file
1. Use `var`, `prompt`, `string` literals, the `+` operator, and `alert` such that w

## Bonus!

https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html

## Homework
- Read [Chapter 2](http://eloquentjavascript.net/02_program_structure.html) in Eloqu

# Week 1, Day 5

## Today's Goals

- Review the week in small groups.
- Take 2 small assessments.
- Become familiar with JS loops (while and for) and conditionals (if/else)!

## 9am Standup

- Events, Helps, Interestings

## 9:15am Warm-up

- Survey (with Reyna)

## 9:30- Assessments
- Command Line
- Git basics
- JavaScript variables
```

10:00 Weekly Review
– small groups

Javascript Basics: Conditionals, Loops
With Zach

Objective

Learn and be able to __apply__ conditionals and loops to problem solving.
Learning the syntax will be the first step, but once you've got that down, focus on

Conditionals

Conditionals control the flow of a program. Conditionals decide which code statemen
An example from everyday life would be:

If you spend $100 or more, then you get 20% off, otherwise the purchase is full price

In the example above, the input to the conditional is the total amount of your purch

If

The most basic control flow statement is the ‘‘‘if‘‘‘ statement. Here is our exampl

```
var total = 284; // Some value
if ( total >= 100 ) { total = total * .8; }
// More code to display the total to the user
console.log("Your total is: $" + total.toFixed(2));
```

Exercise

Write a program that prompts for a users name, then says hello to the user. The progr
Here is an example input and output:

Please enter your name: Zach

Hello Zach. What are you doing here?

Please enter your name: Michael

Hello Michael!! Welcome back.

Multiple If Statements

A program can have multiple if statements when we have multiple control flow needs. For example, we could have a special greeting for multiple names:

var name = prompt("Please enter your name:");

if (name === "Michael") { console.log("Michael Welcome Back."); }

if (name === "Zach") { console.log("Hello Zach! Stop messing with my computer"); }

if (name !== "Colt" && name !== "Tim") { console.log("Hello" + name + ". What are you doing here?"); }

If, else

The multiple if statement code we've written so far can get a little confusing. It is really more suited for an if else control flow statement. If else statements Here is the basic flow and syntax:

if (/*some expression* /) {

// If the expression is true, run this code

} else {

// If the expression was false, run this code instead.

}

We can string together if else statements like this:

if (/*some expression* /) { // run this code } else if (/* some other expression */) { // run this code only if the first expression is false and this expression is true } else { // run this code if the first two expressions were false }

Booleans and Comparison Review

Let's take a minute to review booleans and comparisons. What does the following exp

var bankAccount = 7000; // I'm rich in most cities other than SF var RENT = 3000; var CAR_INSURANCE = 250; var COMCAST = 115; // Way too expensive var CAR_PAYMENT = 120; var NEW_PET_COST = 3000; // An extravagant pet var funSpending = prompt("How much fun money are you going to spend?"); var areYouFinanciallyWise = prompt("Are you financially wise?");

if (areYouFinanciallyWise === “no”) { areYouFinanciallyWise = false; } else { areYouFinanciallyWise = true; }

if ((bankAccount > RENT + CAR_INSURANCE + CAR_PAYMENT + COMCAST + funSpending) || !areYouFinanciallyWise) { console.log(“Congrats, you are buying a new pet liger (half tiger, half lion) name fluffy.”); } else { console.log(“Sorry, no pets for you”); }

Loops

Loops are essential to programming. They allow us to repeat an operation many times.

For Loops

For loops are common in C style programming languages. They are a concise way to ex
The general syntax is below:

for (initialization; conditional; post loop increment) { // Some code to run. }

Here is an example where we print the numbers 1 to 10:

for (var i = 0; i < 10; i++) { console.log(i + 1); }

While loops

A while loop is another way of constructing a loop. Here is the syntax:

while (/* Boolean expression */) { Execute code }

Here is an example:

var timesForPhrase = 10; var phrase = prompt(“What do you want to say” + timesForPhrase + ” times¿‘);

var i = 0; while (i < timesForPhrase) { console.log(phrase); i++; }

12pm Lunch

function eat() { if(hungry){ return “GO EAT” } }; “‘

Exercises

1. Write code to output the numbers 1 to 10. This time make sure to use a while loop instead of a for loop.
2. JavaScript Conditionals
3. JavaScript Loops

4:00 Friday EOD

- Lightning Talks Intro (what are they and when) with Reyna
- EOD with Mike

Homework

Create new repos for each of these.

- **99 Bottles:** Write code to print out the lyrics to 99 bottles of beer on the wall. Pay attention to pluralization!
- **Odd Numbers** Write code to print all the odd numbers between 1 and 1337.
- **Temperature Convertor** Write code that asks the user for a temperature in Celsius and converts it to Fahrenheit. Bonus: ask the user first if they want to convert from F to C or C to F.