Lecture 2: Intro to JavaScript

CS-546 – WEB PROGRAMMING

Recap and Intro

Recap

The course has begun!

We will be running off of Node.js for the semester

The web runs off a series of requests and responses

There will be 5 assignments, 10 labs, and a Final Project

This week

JavaScript

- About JavaScript
- General Syntax
- Strings
- Numbers
- Booleans
- Arrays
- Objects
- Tons of other stuff!

Slack

Several students have requested that we use Slack to communicate more easily throughout the term!

Slack is a chat client that is becoming very popular in the tech world.

If you wish, you can sign up for the class Slack account at https://cs546spring.slack.com/signup

This will be shared between CS-546-A and CS-546-WS.

This tool is useful for the final project, as you will be able to setup private conversations with your group. It will also allow you to more easily contact myself and the CAs and have us weigh in on technical issues you may be discussing.

Reintroducing Node.js

What is Node.js?

There are many languages and environments that you can develop web applications in, each with great strengths and great weaknesses.

Node.js is a JavaScript runtime environment that runs on a computer, rather than in a browser.

- In simple terms, it's JavaScript being run as a script on a computer.

You can run these scripts from the command line.

Node.js often comes bundled with npm, the Node.js Package Manager, which allows you to add dependencies

- In node, you use the *require* function to require other modules (from packages, or from other files)

Why are we using Node.js?

Node.js has been chosen for this course for a number of reasons:

- It is particularly easy to setup
- For the sake of learning, it will be much easier on you to learn one programming language for both the frontend and backend rather than to learn one programming language for the frontend and another on the backend.
- There are many node packages available for you to use
- Node.js promotes extremely modular code, making it easy to organize your code
- Node.js scripts can be run via the command line, making it very easy to test your code without building out your actual web applications, but rather making and running other scripts. You can then phase your code into a web application in a more natural way.

What is a module?

Generally, a module is an individual unit that can be plugged into another system or codebase with relative ease. Modules do not have to be related, allowing you to write a system that allows many different things to interact with each other by writing code that glues them all together.

They are very flexible, and allow you to organize your code very well!

In Node.js, you will be using modules *everywhere*. In our case, a module will be a specific object (think, an instance of a class) that has certain methods and data that you can access from other scripts. You will create your first module today.

What are packages and npm?

Node.js has a *massive* repository of published code that you can very easily pull into your assignments (where applicable) through the *node package manager* (npm).

You will require the modules that your packages export, and use code that other people have created, tested, and tried. You will then use these packages to expand on your own applications and build out fully functional applications.

Using Git to get today’s code

By this point, you should have also installed git. Git is a version control software, that you should be able to use via your command line. As part of this course, you will be learning how to version control your software.

Through your command line, issue the following command:

```
git clone https://github.com/Stevens-CS546/Lecture-2.git
```

By cloning this repository (a codebase with a version history) you will make a local copy of the code in a folder called Lecture-2.

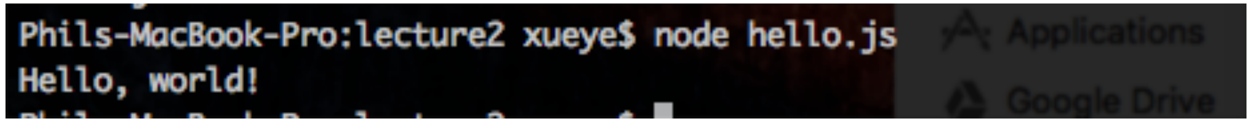
Navigate into this folder, so that you may run the following node scripts together.

Testing Your Install

No programming environment is complete without our next step! Let's check that you configured and installed Node.js properly by running our first file, `hello.js`.

You can do this with the command `node hello.js`

If you look at this file, you will notice that it is only one line. You will be able to log messages to your terminal using the `console.log` function.



```
Phils-MacBook-Pro:lecture2 xueye$ node hello.js
Hello, world!
```

JavaScript Syntax / Intro

Some basic facts about JavaScript

JavaScript is a loosely typed language, a concept that you may have seen before.

- Loose typing means that you don’t strictly declare types for variables, and you can change the type of data that that you store in each variable.

There are five primitives currently, with a sixth (Symbol) on the way

- Boolean
- Number
- String
- Null
- Undefined

JavaScript also has Objects, which all non-primitives fall under

- Functions in JavaScript are types of Objects
- Objects are prototypical

Basic Syntax and Strings

Let’s open up `strings.js` to take a look at basic syntax and some string operations. Variables are assigned with the `var` keyword, and each line ends with a semicolon.

Run `strings.js` the same way as you ran `hello.js`; you’ll see how to store variables, make comments, and do some basic string operations.

Booleans and Equality

JavaScript is a *truthy* language; that means that many things can evaluate as “true” or “false”.

Let’s take a look at `equality.js` and take a look at some if statements.

Running `equality.js` will show you all the things that are true and false in JavaScript.

Numbers

Numbers in JavaScript follow the same pattern. You'll have access to several basic mathematical operators (`*-+/`) out of the box, and several more via the `Math` global variable.

You can read about the Math variable on the MDN

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

Now open up `numbers.js` to take a look at how to handle numbers.

Functions in JavaScript

Functions are one of the most fundamental building blocks of JavaScript.

In JavaScript, variables can store functions; this makes it simple to do things like pass callbacks to functions. A callback is when a function takes a function as a parameter, to call it later.

Now open up `functions.js` to take a look at how we use functions.

Functional and Global Scope in JavaScript

Functions are one of the most fundamental building blocks of JavaScript.

We often want to isolate our scope in JavaScript, particularly when we write browser-based JavaScript code in order to avoid conflicts between libraries.

While Node.js scripts isolate their variables between files, all top-level variables in a browser-environment become global variables, even across different files.

In JavaScript, scope is not defined by block: it is defined by the function you are in. This can cause loops to behave unexpectedly if you assume block-level scoping!

Run `scope.js` to take a look at how functional scoping works.

Objects in JavaScript

Objects in JavaScript are incredibly dynamic and incredibly flexible, and highly readable.

You can create a basic type of Object, an “Object Literal”, very simply, using the `{}` syntax when creating a variable. You can add properties to an object at any time.

Let’s take a look at `objectLiteral.js` to see very simple objects.

Objects in JavaScript

Objects can also be created using functions, which act as constructors.

In order to create an object using a function, you will use the `new` keyword. We will cover the `new` keyword more later, to get into exactly what it does.

Another object of note is that in your constructor function, you attach properties to the object using the `this` keyword. By using the `new` keyword, each time you create a new object using the constructor function, `this` will be a new object with no defined properties.

Without the `new` keyword, this will default to the global, top level scope.

Let’s take a look at `objectFromConstructor.js` to see objects from a constructor function.

Arrays in JavaScript

Arrays in JavaScript also inherit from Objects, and can store elements of any type.

You can create an array using the very simple syntax of `var myArray = [1, 2, 3];`

Like an object, you do not have to populate anything in the initial array; you can manipulate it freely after creation.

Arrays have a number of built-in methods, which you can see by running through `arrays.js`.

Types and Prototypes

In JavaScript, everything is either an object or a primitive; even functions are objects!

This is because JavaScript is a *prototypical* language.

JavaScript is a prototypical language, which means that every object follows a blueprint; these blueprints can point upwards any number of times. This is similar to the Object Oriented Pattern, but not identical.

Prototypes in Objects

As JavaScript is a dynamic language, there are no classes; instead, each object follows a blueprint referred to as the Prototype.

If you go up the prototype chain, you will encounter the Prototype of Object, followed by the Prototype of null.

Whenever you access a property on an object, the interpreter will first check for that property directly on the object. Failing to find it here, the interpreter will then check for properties on that object's prototype; failing to find it there, it will then check the parent prototype, the grandparent, and so on until Object and null.

You can see a demonstration of how to use the Prototype chain by analyzing `proto.js`.

Modules in Node.js

Require

There is a special, global function called *require*, which will allow you to import code from other files, packages, etc.

When you require a file / package, you will be accessing whatever the programmer assigned to be exported in that file. From there, you can use the code.

This allows you to make very small, isolated code that performs related functions.

How do I make my code ‘requirable’?

There is another global variable called `module`, which has a property called `exports` on it.

When you require a file / package, it will take whatever is assigned to the `module.exports` variable in a package. You can export anything you want: a function, a number, or an object that allows you to do any combination of these things.

You can see an example of this in the `my_module.js` and `requiring_my_module.js` files.

You can use those files in order to get started making your own modules!

Why would I use modules?

The more unrelated code you have together, the messier your application will become and the harder it will be to maintain.

- You can have accidental name collisions
- It becomes harder to follow what the related components are in a large file
- It becomes less readable overall

Modules allow for many great things:

- You can strictly define what code is exported to be used, allowing you to make entire files with a defined structure
- You can change the internal workings of a module to make it more performant and add more features while not updating external code.

Preparing for Next Week

Readings For Next Week

Take a look at the Express module, which you will use as your web server

- http://expressjs.com/

Take a look at the Request-Promise module, which you will use next week to make requests from servers.

- https://github.com/request/request-promise

Assignment 1

You can find Assignment 1 on Canvas!

For assignment 1, you will be making several modules and requiring them from one central file.

It will be due on 2/12

Lab 1

Lab 1

This week, you will have your first lab!

This lab is estimated to take about 60 minutes; you can find it on Canvas. You will have 24 hours to complete it.

Questions?