

Lecture 1: Intro to JavaScript

CS-546 – WEB PROGRAMMING

Introducing 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).

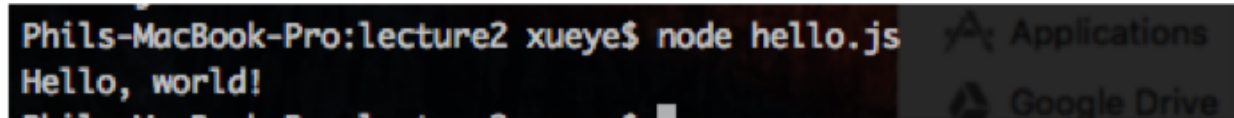
You will require the modules that your packages export, and use code that other people have created, tested, and tried. You will then use these packages to expand on your own applications and build out fully functional applications.

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.

A screenshot of a macOS terminal window. The prompt is 'Phils-MacBook-Pro:lecture2 xueye\$'. The command 'node hello.js' has been entered and executed. The output 'Hello, world!' is displayed on the line below the command. To the right of the terminal window, there are two dock icons: 'Applications' and 'Google Drive'.

```
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 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

Defining Variables

There are three keywords used to define variables in JavaScript:

Keyword	Scope	Explanation	Example
const	Block	Initializes a block scoped variable that cannot get overwritten. Most common used in this course.	<code>const twelve = 2 + 10;</code>
let	Block	Initializes a block scoped variable that can get overwritten.	<code>const numbersToAdd = [1, 2, 3]; let squareSums = 0; numbersToAdd.forEach(num => { const squared = num * num; squareSums = squareSums + squared; });</code>
var	Functional	Initializes a variable that can get overwritten; not used in course.	<code>var twelve = 2 + 10; twelve = 24 / 2;</code>

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 `boolean.js` and take a look at some if statements.

Running `boolean.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 (\ast $-$ $+$ $/$) 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 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 unless using the keyword *let* to define a variable; when using *var*, it is defined by the function you are in.

- <http://coffeegrammer.com/understanding-functional-scope/>

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 `objects.js` to see very simple objects.

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 `let 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`.