Udemy: Learn and Understand NodeJS

1. Introduction and the Goal of this Course
2. Asdf
3. Asdf
4. Conceptual Aside – The Command Line Interface
5. Command Line References
6. Conceptual Aside – Processors, Machine Language, and C++
   1. When we give the microprocessor on your computer instructions, we give them to it in the specific language it understands. Otherwise it wouldn’t be able to run them. There’s a variety of instruction sets/languages that microprocessors might be designed to speak. The processor’s a physical machine that accepts instructions and carries them out.
   2. Machine code: programming languages spoken by computer processors. Every program you run on your computer has been converted (compiled) into machine code.
   3. So ultimately whenever we write a programming language, there’s another process running that converts the code you write into machine code
   4. The first thing to understand about node: Node is written in C++. The reason why is that V8 (the JavaScript engine) is C++. V8 is the thing that converts JavaScript into machine code.
7. JavaScript Aside – JavaScript Engines and the ECMAScript specification
   1. ECMAScript = the standard that JavaScript is based on. Need a standard since there are many engines. ECMAScript is the name for the core standard that JavaScript is based on. Note: V8 is actually just one of many JavaScript engine
   2. JavaScript Engine: A program that converts JavaScript code into something the computer processor can understand. And it should follow the ECMAScript standard on how the language should work and what features it should have.
8. Adding features to JavaScript
   1. V8 is able to be embedded in other C++ programs, so if you wrote a program you could put V8 inside it and use it. You could write a C++ program where other people can write JavaScript and your C++ program can take it and run it through V8.
   2. Since V8 is just C++ that just takes JavaScript and doing things with it, V8 allows us to write our own C++ code that’s available to JavaScript. Ie. We can make it so that if someone writes something in particular in their JavaScript code, that will cause my C++ code to be run. This allows us to essentially add features to JavaScript by embedding V8 into our C++ program, so our C++ program will read and understand more than what the ECMAScript standard specifies. This is very powerful because C++ has far more features than JavaScript. So, we can write things in C++ that JavaScript can’t do, and then make anything that we write available to JavaScript
   3. All this allows us to attach C++ code to our own custom JavaScript keywords. Essentially we can add features to JavaScript. This is what node.js is – a C++ program with V8 embedded that has added a wealth of features that make it suitable to be a server technology
9. Conceptual Aside: Servers and Clients
   1. Nodejs is a server technology. It’s designed to be able to use JavaScript to write server code.
   2. A **server** is a computer that’s performing services. It’s just a computer performing jobs requested of it. The **client** asks for those services (and may also do some work).
   3. The client sends a request for a service (for some work to be performed) and the server responds, with perhaps verification that the work’s been performed, or error messages, or data. And that request (and the response also) is in a standard format that both the client and server and programmed to understand.
   4. In the case of the internet, we have web servers, and the usual client is a web browser.
10. What does JavaScript need to manage a server?
    1. What are the features we need to have accessible in JavaScript in order to do the kind of things that a web server normally needs to do?
       1. Better ways to organize our code into reusable pieces
       2. Ways to deal with files
       3. Ways to deal with databases
       4. The ability to communicate over the internet
       5. The ability to accept requests and send responses (in the standard format)
       6. A way to deal with work that takes a long time (we want people to continue to be able to access the web server while it deals with a big job)
11. The C++ Core
    1. The C++ core is a core of features, of utilities, built in C++ and made available to JavaScript via the hooks in the V8 engine
    2. The mental model you should have is not that nodejs is JavaScript, but rather that it accepts JavaScript and allows us to write more features in it than usual
12. The JavaScript Core
    1. There are plenty of C++ utilities in nodejs, but there’s also pure JavaScript that’s been written for you to help make using those C++ features easier as well as other common tasks and needs you have in js development
13. The JavaScript Core
14. Downloading Lecture Source code
15. Let’s Install and Run Some JavaScript in Node
    1. You’re always going to give node an entrypoint, a single JavaScript file it’ll run
16. Conceptual Aside – Modules
    1. Module = A reusable block of code whose existence does not accidentally impact other code (JavaScript didn’t have this before ES6, at the time of node’s creation). Since these weren’t around when node was created, node needed to give us the ability to build code modules (reusable blocks where we could put code, where it wouldn’t impact other code unintentionally)
    2. NodeJS implements CommonJS modules. CommonJS Modules = An agreed upon standard for how code modules should be structured
17. JavaScript Aside – First-Class Functions and Function Expressions
    1. First-class functions means that everything you can do with other types you can do with functions. You can use functions like strings, numbers, etc (pass them around, set variables equal to them, put them in arrays, and more)
    2. An expression = a block of code that results in a value. Function expressions are possible in JavaScript because functions are first-class.
18. Let’s build a module
    1. Require() takes a string that’s the location of the file you want to import
    2. When you use require to pull in a module that has functions defined within it, you won’t be able to straight up use them in your main file – they won’t be defined.
    3. Module.exports is a special place to put anything that you want to make available to other things that use this module. Only the things we attach to this will be made available outside of this module for use. It determines what we expose.
    4. Then, where we use require, we set a variable equal to the require function. The require function returns module.exports
    5. Module and require and exports are available in the Node javascript core
19. JavaScript Aside – Objects and Object literals
    1. Name/Value pair: a name which maps to a value. The name may be defined more than once, but can only have one value in any given context. That value may also be more name/value pairs.
    2. In javascript, an object is just a collection of name/value pairs
    3. Object literal: name/value pairs separated by commas and surrounded by curly braces. Quick shorthand for creating objects
20. JavaScript Aside – Prototypal Inheritance and function constructors
    1. Every object has a property that actually points to another object. It’s the object’s prototype, the thing it inherits from. It’s a different object, and the first object will have access its properties. And that prototype can be connected to another prototype, etc., and we can access properties on any of those objects down the line without having to specify which one it is.
    2. Function constructors: a normal function that’s used to construct object. The ‘this’ variable points to a new empty object, and that object is returned from the function automatically.
    3. What happens when you use a function constructor is that any object created from the function constructor, its prototype will point to the prototype property of the function that you used to create the object
21. Asfd
22. Afds
23. How do node modules really work: module.exports and require
24. JavaScript Aside - JSON
    1. JSON = A standard for structuring data that’s inspired by JavaScript object literals. JavaScript engines are built to understand it.
    2. Note that we don’t put functions in JSON because JSON is just pure data. Also have to put our names in quotes. JavaScript comes built in with the ability to take JSON and turn it into a JS object and vice versa
25. More on require
    1. You can require (‘greet’) and if there’s no greet.js, require will instead look for a greet folder. Then it will look inside that greet folder for a file called index.js
    2. If you pass require a JSON file, it will create an object from the file and pass it to the variable that you’re assigning require()’s result to
26. Module Patterns
    1. When you require the same exports object multiple times, you get multiple references to the same object, not copies
    2. Revealing module pattern = exposing only the properties and methods you want via a returned object. A very common and clean way to structure and protect code within modules.
27. Exports vs Module.exports
    1. Both point to the same object, or at least they start out pointing to the same object. But modifying one doesn’t change the other
28. Requiring Native (Core) Modules
29. Modules and ES6
    1. Nodejs added the feature of modules to JavaScript because it didn’t have it. But now the new specification of javascript includes modules, so engines like V8 are supporting the new module structure. So, eventually we’ll see a move towards ES6 modules
30. Web Sever Checklist
31. Conceptual Aside – Events
    1. Event = something that has happened in our app that we can respond to. In node we actually talk about two different kinds of events. But we have a tendency to think about both as if they were the same kind.
    2. System events: come from the C++ Core, specifically libuv. The library of C++ code that’s part of NodeJS that deals with events coming from the computer system, like, I finished reading a file, I’ve received data from the internet, things like that the JavaScript doesn’t have. Those C++ events allow us to respond to those things happening with code
    3. Custom Events: come from a JavaScript library of events that deals with events that I can create for myself, where I can say in my code, hey this happened and respond to something happening. This is the event emitter inside the JavaScript core. It’s the JavaScript file that contains the code for what’s called the event emitter. The event emitter is where we have custom events (JavaScript), and libuv is sending events that are happening inside the computer system (lower level, closer to the machine). But, the JavaScript code often wraps code to the C++ side of node, so oftentimes when an event occurs in libuv, it generates a custom JavaScript event to make it easier to deal with.
    4. Really, the JavaScript side is faking it. JavaScript doesn’t have an eventing concept, there’s no event object in JavaScript. But, we can create an event library, our own way of creating, telling the code that something has happened and responding to it via the technique that the node event emitter uses.
32. JavaScript Aside – Object Properties, First Class Functions, and Arrays
33. The Node Event Emitter Part 1
    1. Event listener = the code that runs when an event happens. In JavaScript’s case, the listener will be a function.
    2. On is basically for adding a listener
34. Asfd
35. Asfd
36. Asdf
37. Asdf
38. Sadf
39. Asdf
40. Ads
41. Fa
42. JavaScript Aside: JavaScript is Synchronous
    1. Node does things asynchronously. V8 does not.
    2. JavaScript is synchronous. Think of it as only one line of code executing at a time.
43. Conceptual Aside – callbacks
    1. Callback = a function passed to some other function that we assume will be invoked at some point. The function ‘calls back’ invoking the function you give it when it’s done doing its work (or at some other point)
44. Libuv, the event loop, and non-blocking asynchronous code
    1. System events are handled by the libuv library. The idea here is managing events coming from the operating system, close to the machine.
    2. Inside node, we have V8, and javascript is synchronous, so V8 runs code, essentially, one line at a time, and that keeps happening as functions are called.
    3. Also inside node is libuv, another library in node written to deal with lower-level events occurring in the operating system. Libuv connects by requesting something from the operating system. Inside of it, libuv has a queue of events that have completed.
    4. There’s an event loop in libuv. It means libuv is constantly checking the queue to see if anything’s happened. At some point, the OS completes an event and that event gets placed in libuv’s queue. Every time in the event loop, libuv checks the queue to see if there’s anything in there. And if things have been finished and put in the queue, libuv processes it and runs a callback, i.e. code meant to be run when that event completes.
    5. That callback usually involves running javascript code. V8 will wait to run that code until it finishes running its current code, because JavaScript is synchronous
45. Conceptual Aside – Streams and Buffers
    1. Buffer = a temporary holding spot for data being moved from one place to another
    2. Stream = a sequence of data made available over time. Pieces of data that eventually combine into a whole
46. Conceptual Aside – Binary Data, Character Sets, and Encoding
    1. Character Set = a representation of characters as numbers. Each character gets a number. Unicode and ASCII are character sets where characters get a number assigned to them.
    2. Character encoding = how characters are stored in binary. Really how many bits we use to represent each number
47. Buffers
    1. Var buf = new Buffer
    2. We can give new buffers a size value or a string + encoding. This asks the buffer to take the string and convert it to binary using whatever encoding we give it.
    3. Can do buf.toString() to convert to string or buf.toJSON()
48. JavaScript Aside - ES6 Typed Arrays
49. JavaScript Aside – Callbacks

**Lynda: Up and Running with Node.js**

Module.exports is the object that’s actually returned as the result of a require call.

The exports variable is initially set to that same object, so in the module code you would usually write something like this:

var myFunc1 = function() {…};

exports.myFunc1 = myFunc1;

to export (or “expose”) the internally scoped functions myFunc1 and myFunc2.

And in the calling code you would use:

Var m = require(‘mymodule’);

My.myFunc1();

**Lynda: Node.js Essential Training**

* Every node.js file we create is its own module, and every variable we create is scoped to that module. They aren’t available in the global namespace
* The process object is available to use globally is the process object. It contains functionality that allows us to interact with information about the current process instance.

**Tutorialspoint**

A Node.js application consists of the following three important parts:

* Import required modules – we use require directive to load a Node.js module
* Create server – a server which will listen to a client’s request similar to an Apache HTTP Server
* Read request and return response – the server created in the earlier step will read HTTP requests made by a client (can be a browser or console) and return the response.