Notes for 4th March

TODO

1. Push the code in git
2. Revise the html part once more
3. Revise Node.js once more
4. Take a look at the remaining course content
5.

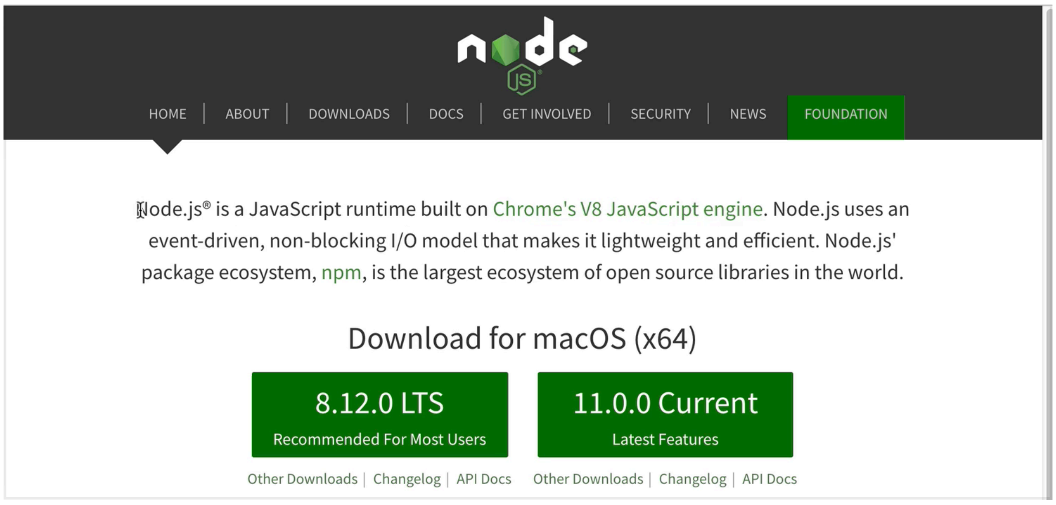
1. What is vh - it is viewport height
2. Go to api.openweathermap.org
3. Pull out the apiKey - 2d3aa7eb5d3cf4e429f6c85a67e1b355
4. We will try on the browser api.openweathermap.org/data/2.5/weather?q=bengaluru&appid=2d3aa7eb5d3cf4e429f6c85a67e1b355
5. First try out with a sample endpoint, on the js file, use console.log and then use the console command on the browser to show the response of the api

What is Node.js?
Came about when the original developers took javascript which till that point we could run only on our browser and they allowed it to run as a standalone process on our machine
Before Node, js couldn't be used for as a more general purpose programming language.
It was limited to what the browser allowed it to do
For example: we could use javascript to add a click event to a button as we saw in our example from the first class
But there was no way to use javascript outside the browser to build things like the Web Server that can access file systems and connect to databases.
All of these are things that other programming languages can do without any problem

All of these ended up changing with the introduction of Node.js
With node.js js developers can now use javascript on the server side, so they can use the same Javascript programming language to create web servers, command line interfaces, application backends and more. All of which we will end up covering in the class.

So node is a way to run javascript code on the server as opposed to being forced to run it on the client.

Now the question that has to be asked is - How is this possible??
We will understand the definition by looking at a one sentence summary



Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

Download for macOS (x64)

8.12.0 LTS
Recommended For Most Users

11.0.0 Current
Latest Features

Other Downloads | Changelog | API Docs
Other Downloads | Changelog | API Docs

We are going to start exploring by going through this one sentence summary - this line still available on the current node.js homepage.
We are going to read this, break this down and this is going to shed some light on what exactly is happening here.

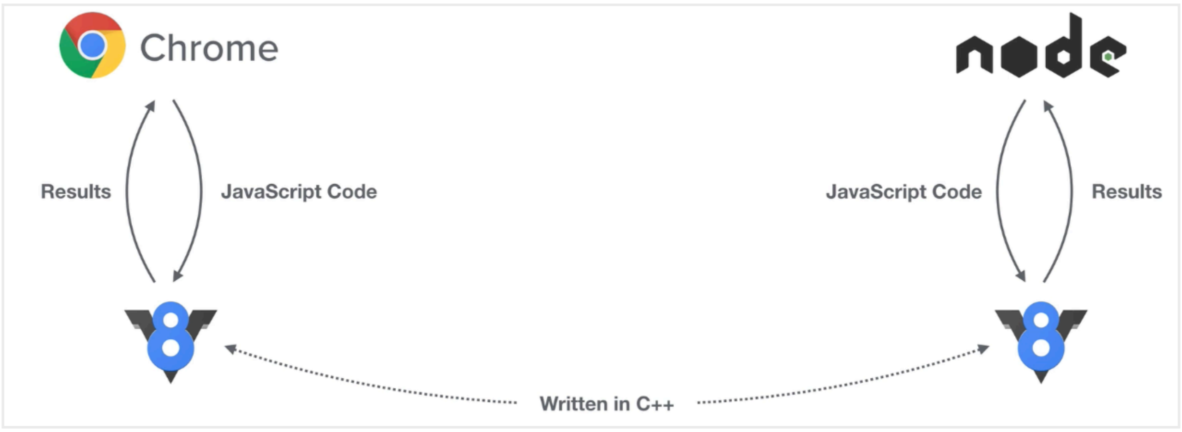
So node.js is a Javascript runtime built on Chrome's V8 engine. And it all comes back to that V8 Javascipt engine. There are all sorts of javascript engines out there, pretty much every major browser has their own javascript engine.

Node.js uses the V8 js engine and it is the same javascript engine that powers the chrome browser and V8 is a google open source project. The job of the javascript engine whether it is V8 or the other ones is to take in javascript code and compile it down to machine code that our machine can actually execute.
Now the V8 engine is written in the C++ programming language. That means anyone out there can write a c++ application that could incorporate the V8 js engine into their application and they could extend the functionality that javascript provides.

And that is exactly what both chrome and node.js do. Both chrome and node.js are largely written in C++.
Now we will notice that Node.js is a javascript runtime. Node.js is not a programming language. And this is true, so all of the code that we are going to write in this class is indeed going to be javascript code and that is the reason why the first part of the course is invested in teaching students javascript because that's exactly what we will use to write our code in.
A runtime is something that provides custom functionality meaning specific tools and libraries specific to an environment.
So in the case of chrome it provides V8 with various objects and functions that

allowed javascript developers in the chrome browser to do things such as **add button click events or** manipulate the DOM

What is a DOM - DOM stands for Document Object Model - The **Document Object Model** (**DOM**) is a cross-platform and language-independent interface that treats an XML or HTML document as a tree structure wherein each node is an object representing a part of the document. The DOM represents a document with a logical tree. Each branch of the tree ends in a node, and each node contains objects. DOM methods allow programmatic access to the tree; with them one can change the structure, style or content of a document.
Now neither of these features make sense for node where we neither have buttons and we don't have a DOM. So node doesn't provide those things.
Instead node runtime provides various tools that node developers need.
For example: libraries for setting up web servers, integrating with the file system so that we can read and write from and to a disk and in the end both chrome and node.js are creating their own modified version of javascript.
In the end of the day: it's the same core javascript language but with custom functions and objects injected.



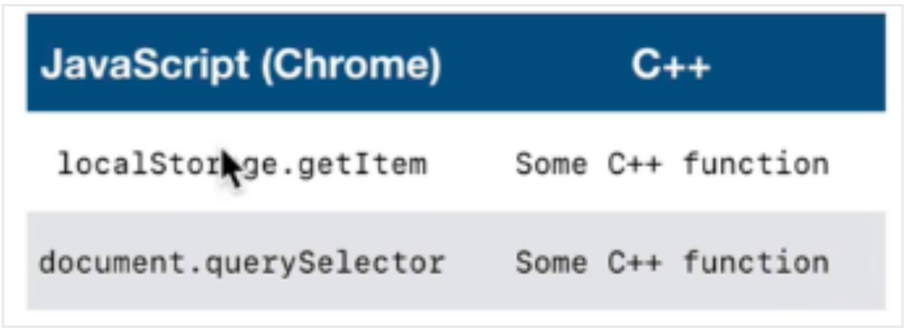
As I mentioned we have the chrome browser and the chrome browser has the V8 javascript engine. Now when Chrome needs to run some javascript code for a webpage it doesn't run the javascript itself, instead it uses the V8 engine. So it passes the code to V8 and it gets the result back. Similarly node doesn't know how to run javascript code. It uses that V8 engine, so when node needs to run a javascript code, it passes the javascript code to the V8 engine and get the results back.

Now I mentioned that V8 is written in C++ and that node and chrome are both largely written in C++ and once again that's no coincidence. The reason why both chrome and node.js is written in c++ is that they are both providing bindings when they are instantiating V8 engine. This is what allows them to create their own javascript runtime with interesting and noble features.

This is what allows Chrome to interact with the DOM when the DOM isn't part of javascript and it allows node to interact with the filesystem when the filesystem isn't part of javascript either.

So V8 doesn't have idea how to interact with the filesystem or the DOM. It is upto Node and Chrome to provide implementations for those when running V8.

What does this process look like??

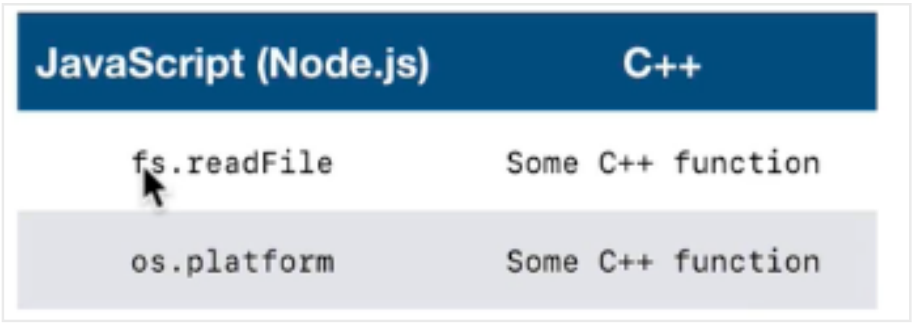


JavaScript (Chrome)	C++
`localStorage.getItem`	Some C++ function
`document.querySelector`	Some C++ function

Well we have a table, it's not really a table but it is convenient to think of it as a table
We have a set of javascript methods and objects and we have a set of C++ functions.

In the left we have two very popular javascript methods. Here we have localStorage.getItem which will fetch an item of a local storage
And I have document.querySelector which would allow me to query some elements from the DOM (if you remember we have already used this method in our example previously) and then do something with them, like deleting them or changing their contents. But it may surprise you that neither of these are part of the javascript programming language itself. You can pull out the specification and search for them but you are not gonna find them that's because they are actually Implemented by the chrome runtime. When Chrome runs a javascript file that uses either of these, in the end of the day some C++ code gets executed behind the scenes which is responsible for taking care of the functionality. That's why chrome has so much C++ code. Now chrome needs to tell V8 what to do when these methods are called so Chrome's not passing just js code to V8 engine, it is also passing down the C++ bindings creating the context for which the js code should be executed.

Now the exact same thing is true with Node.js. Over here I have picked up a couple of node.js methods that we will all be familiar with in future.



JavaScript (Node.js)	C++
fs.readFile	Some C++ function
os.platform	Some C++ function

We have fs.readFile where fs stands for FileSystem and readFile allows us to read the contents of a file from disk. We also have os.platform where os stands for operating system and platform gets us the platform we are running on such as Linux, Windows or Mac. Now neither of these methods are part of the javascript language. These are provided to V8 by the node.js runtime. So javascript doesn't know how to read a file from disk but C++ does. So when someone uses js code in nodejs to read a file from disk it just defers all of that to the c++ function that can read a file from disk and get the results back where they need to be..
So once again we have a series of methods that can be used in our JS code, which are in reality just running C++ code behind the scenes.

Now let's wrap up the understanding by going over a little code example by illustrating some of the differences and similarities between javascript in the browser and js inside of node.js
Now let's fire up the browser and open up a console and try out some things

The console will allow us to run individual javascript statements. So for example we can type 2 + 3 and get result 5 back. I can type my name in here and then use the string method to convert the string to upper case and that will convert the string to uppercase.

Now we can do the exact same thing in node as well. For now we will try things on the terminal itself, later we will write multiple scripts in files, using Visual Studio Code. We are just going to type node and hit enter
Now what we get here is a little place where we could run individual node statements also known as REPL which stands for Read, Eval, Print, Loop, like the terminal it is waiting for us to run some command but these are actually going to be node statements and not bash commands.

So we can see that all the core js features we are used to are still available while using node.js because those are provided by the V8 engine itself.

Now let's explore a couple of the differences. You might not be familiar with this but there is something called the window object. Window object is a reference to a browser object. If we type window and press enter, it will dump all of the object to the console. Now we can scroll through and see some of the methods. Now all of these make sense in context of the browser because we actually have a window. What happens we type window in node.js. Window is something provided specifically by the Chrome runtime when js is running in the chrome application. Window is not provided in Node.js, because Node.js doesn't need a window and hence we get the error. With node we have something similar, a variable called "global" . Stores a lot of the global things that we are going to use throughout the course. We are not going to dive into these now but we will definitely explore a few of them as we go through the course.

Now does a browser have global? Well the browser doesn't have global. In the browser we have access to document. Document allows us to work with the DOM. We have a whole bunch of elements that make up the page and we can use document to manipulate them in whatever we see fit. Now again, this makes sense for a browser where we actually have a DOM, but it doesn't make sense for node where we don't have a DOM. Once again reference error. We have something similar to document called **Process.** Process gives us various properties and methods for manipulating the node process that is running. We have a method called exit, which will let us exit the process. We can actually use this to shutdown our node terminal.

We need to enter **process.exit()** and hit enter. Well we will be brought back to the standard terminal. And we can see that process will not be available in the browser. So we saw some of the differences between js in the browser here specifically chrome and node.js which is js running on the server.

Let's take a quick moment to summarise, Node.js is javascript on the server. This is possible because node.js uses the V8 engine to run all of the js code they provide. Now node is able to teach js new things by providing c++ bindings to V8. This allows js in essence to do anything that c++ can do and C++ can indeed access the filesystem and thus javascript can access the filesystem.