Node is an environment in which one can write server-side scripts in JavaScript.

* For example, servers
* Uses ES6 ....so you will see **const, let and => notation,** which you are familiar with from our earlier work.
A node.js script is a JavaScript script which makes use of many small pieces of functionality in **modules**.

The modules are stored as **packages** on the **npm** or **node package manager**.

* The npm has become so useful that other packages are also stored there - for example jQuery.
* There are several places where you can look for packages for your needs - it is a huge ecosystem.
* Often one module will make use of  - or depend on - another module, which introduces **dependencies**.
* Some of the modules we will use most often are **http, eventemitter, fs** (which stands for file system).

Typically - see <https://adrianmejia.com/blog/2016/08/12/getting-started-with-node-js-modules-require-exports-imports-npm-and-beyond/> .
In the first block of code you will see:

|  |  |
| --- | --- |
| 123456 | const fs = require('fs');fs.readFile('./file.txt', 'utf-8', (err, data) => { if(err) { throw err; } console.log('data: ', data);}); |

Here we are using the module **fs** , and getting it with the key word **require**.
**It is the customary in Node to name your variable (or const) with the same name as the module – as annoying as that may seem to begin with.**

The fs module is actually returning an object and the object has functions (key-value pairs where the value is a function. One of those functions is (the value for) readFile. Now, readFile has 3 parameters: the first is a path to the file being read (here ‘./file.text’), the second is the encoding (here ‘utf-8’) and the third is an anonymous function which is defined using arrow notation. More about that in a minute.

The important thing here is that we use **require** to get something from the **fs module.** The something we get is an object with methods and attributes, which we can access (as usual) using the dot notation.

Because of everything you learned about closures, your fs object is able to use the variables and functions in the object (the fs module) which created it – remember in a closure the object or function which is returned has access to the entire context in which it was created --- but the code in the creating module is something your object can’t change. In other words, the code in the fs module is protected from you, but the functions which are returned to you are functions you can use that as needed.

In the next box on the same page, you see how a module **exports** objects to a user.

circle.js

|  |  |
| --- | --- |
| 123 | const PI = 3.14159265359;exports.area = (radius) => Math.pow(radius, 2) \* PI;exports.circunference = (radius) => 2 \* radius \* PI; |

Here two methods, area and circumference, have been attached to the object which are module is exporting. The module also has a constant PI, which is not exported, but you can see that both area and circumference make use of that const in their calculations.

Of course, the way to read the arrow notation is that area is a function which takes one parameter, radius, and returns Math.pow(radius, 2) \* PI. Similarly for the circumference function.

If you look back at the first box of code, you will see that the third parameter of readFile is an anonymous function which takes two parameters, named err and data, which then executes the code in { if (err)…….}

This anonymous function is actually a **callback function** which is executed after readFile has found and read the file and returned two values – which get assigned to err and data.

**Another somewhat annoying custom in Node is to name the main (driver) program app.js** In other words, you will end up with lots of different files (in different folders) named app.js --- just as when you design a large site you may end up with lots of pages named index.html

**What is meant by: Node is an event-driven runtime environment which uses a single-threaded asynchronous R-E-P-L ?** Let's take this one term at a time.

***Node is event-driven:*** Well, we learned a lot about event-driven code when we studied jQuery. What is meant here is that what happens in a Node script is determined (driven) by events. The events may be things like a form being posted to the Node script, or some other request being made.

***Node is a run-time environment:*** Node starts up and keeps running, and handling these requests until you shut it down. In that sense, Node works the same way an operating system does (but please don't stop your OS.) That is, Node allows you to run JavaScript (really ES6+) *outside your browser.*

***Node is single-threaded:*** There is only one process which is running. That one process handles all the requests which come into it, makes all the requests of other resources as needed, and then responds to the requests made of it. This is one of the distinctive features and also why ***Node is asynchronous***. Let us suppose that you have a Node script which responds to requests from a particular web page (e.g. a site for selling used books.) Various users will be reading that page on their machines and will make a request – e.g. What copies do you have of a certain book? Each request comes to the Node script, and that script will, in turn make a query against a database. The Node script keeps getting more requests from web pages and forwarding the queries to the database. After a while, the database will respond to query 37. Node then takes the information the database returned, constructs a response to the user whose request generated that query, and sends the response back to the user. Query 37 has been taken care of (and may be removed from the list of pending queries) and Node just 'hangs out' waiting for more requests from users, and results from the database. It's pretty clear why Node's response to the users is asynchronous. Another way to say that Node is asynchornous it to say that it is ***non-blocking.***Because of this structure, Node is very well suited for applications with a lot of I/O (e.g. airline reservations, on-line shopping, social networks, etc.) --- and is not well suited for operations which are CPU intensive (e.g. pattern matching)

***R-E-P-L*** stands for Read-Evaluate-Process-Loop. That is exactly what a Node script does. It reads the request (from the user), evaluates it, and processes it (send the request to the database, wait for the answer, construct and send off the new page to the user.) And then Node loops back waiting for the next request (which it will read, evaluate. Process, etc.)

**To summarize:**Node.js allows us to write JavaScript files and run them outside a browser. Typically the file which we will run in Node.js is named app.js

Our app.js will use modules (libraries ) which bundle together such things as setting up a file server, reading and writing files, constructing web pages to send back to our user, etc. In other words, the power of Node.js resides in all these wonderful modules.

We will get access to the functionality these modules provide by using the *require( )* function.

Often a module will depend on other modules, and these *dependencies* are described in the package.json file which Node.js will put together for us when we *init* a new Node application.
The package.json file and the modues we have required will be in the same folder as our app.js

The Node package manager is named ***npm***. There are several sites which allow us to search for what we want on npmjs.com In fact, npm has become so popular that many other modules, such as all the jQuery plug-ins, are also on npm. (Search for jquery-plugin)