the Master Course

{CODENATION}

Introduction to Node.js

{CODENATION}

{CODENATION}

Learning Objectives

To understand what NodeJS is used for.

To be able to run your javascript files in a Node Environment.

To understand modularity in Javascript.

To be able to export and require modules.

To understand what NPM is and how to use it.

Node.js

A javascript runtime environment

An environment which understands javascript
outside of the browser.

{CODENATION}

Node.js

And that is pretty awesome.

{CODENATION}

Node.js

Node is a huge reason for javascript's surge in popularity because it lets us do things we could never do before.

{CODENATION}

Node.js

Javascript was designed to run in the browser, which really limited its wider application.

Node.js takes javascript server-side.
And this really is massive.

{CODENATION}

Node.js

Some cool things about Node:

It's really fast.
It's event driven
It's non-blocking (can run JS asynchronously)

{CODENATION}

Node.js

Some cool things about Node:

It's powered by Google's V8 engine and is written in C++.

It takes the javascript you write and converts it into machine code.

Node.js

We can run our javascript files in node.
We simply use the node keyword and the path to the js file we want it to run.

Example:
node app.js

{CN}®

Node.js

In a browser Javascript can **add interactivity** to a webpage but can't really do too much else.

{CODENATION}

Node.js

Node means we can connect to databases, access the computer's file system etc - basically do what traditional languages like C and Java can do.

There are many JavaScript libraries and frameworks on the web.

They are simply JavaScript files that other developers have written.

If we pull the files into our own project, we can use their files as if we had written them ourselves.

Node has a modular system.

And these node modules are just like the javascript libraries we spoke about before.

There are core modules built in to Node and there are (lots) more we can install too.

Node.js

We will look at both: core modules and ones we can install.

We will also look at creating our own modules.

You should start thinking about how you could separate your code in a modular way. Splitting functionality into their own separate .js files.

Reminder

You must always read the documentation when learning a new technology.

{CODENATION}

Node.js

A bit more about modules:

A module is really just a javascript file with some functionality (it does something) and you can use that functionality in your javascript code to do things. win.

{CN}®

Node.js

Local modules (made by us!)

Core modules (built in to node)

Third party modules (made by ace people in the node community)

{CN}®

Node.js

Question:

Say I have a file called dansFunctions.js with two functions, add( ) and subtract( )

And I want to use those functions in my main.js file.

How is my main.js file going to know where the dansFunctions.js file is and what functions it has?

{CN}®

Node.js

Welcome module.exports and require( )

{CN}®

Node.js

Essentially what we need to do is export our functions so other files can use them. Then in a different file we require them. Ace! I'll show you how.

{CN}®

dansFunctions.js ● | main.js

```
1
2
3   let add = (num1, num2) => {
4       let result = num1 + num2;
5       return result;
6   }
7
8   let subtract = (num1, num2) => {
9       let result = num1 - num2;
10      return result;
11  }
12
13  module.exports = {
14      add,
15      subtract
16  }
17
18
```

We will use the **module.exports object** to list which functions, variables, objects, arrays etc that we want to export (make available outside of this file).

{CN}®

dansFunctions.js | main.js

```
1
2   const dansFuncs = require('./dansFunctions');
3
4   dansFuncs.add(2,3);
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

Now in the main.js file I create a variable and store a reference to the dansFunctions file using the require method.

I can now access the functions in the dansFunctions.js file. Notice how I use them using the dot notation we're now very familiar with!

dansFunctions.js | main.js

```
1
2   const {add, subtract} = require('./dansFunctions');
3
4   add(2,3);
5   // returns 5
6
7
8   subtract(6, 3);
9   //returns 3
10
11
12
13
14
15
16
17
```

Sometimes you may know exactly which functions you want, in which case you can require them directly like this in curly brackets. Now they're available in main.js

This uses something called Object Destructuring if you want to look it up.

{CN}®

This was an example of using a local module in our project.

TASK: create a new file called maths.js - inside create a function that takes two numbers and multiplies them together. Export this function and import it into your main.js file. Run the file in there to check it works.

Node.js

Let's have a quick look at core modules. Built right in to Node for us to use!

{CODENATION}

Node.js

Let's do something really cool. Like access data on our computer. Using Javascript!

{CODENATION}

Node.js

Let's require the os module and the fs module.

Can you remember how?

{CODENATION}

Node.js

Let's require the os module and the fs module.

```
const os = require('os');
const fs = require('fs');
```

{CODENATION}

{CODENATION}

```
let userDetails = os.userInfo().username;

fs.appendFile('oh-hi.txt', `Hello ${userDetails}`, (err) => {
    if (err){
        console.log('oops');
    }
})
```

Try this out and tell me what it does.

Reminder

You must always read the documentation when learning a new technology.

os and fs have different methods - read up if you want to know what else you can do with them.

Node.js

Modules can be contained nicely in a package.

We can install packages in our projects and make use of all the goodness they offer. Ace.

{CN}®

Node.js

Third-party modules are put in easy to use **packages**.

But do we need to go to each developer's website and manually click on a download button for each one?

{CN}®

Node.js

Nope! NPM has us covered

{CN}®

NPM

Node.js

Node Package Manager:
We can use NPM directly in the terminal to install any packages we want to use (ace).

NPM is included with node so we don't have to install it (also ace).

Node.js

Node Package Manager:
The first thing we should do when we start a new project is create a package of our own.

TASK: First we must initialise our project folder using the command: npm init
Press enter a few times for now to keep the defaults.

{CN}®

NPM Node.js

After running the npm init command, you'll have a brand-spanking new package.json file.

Congrats! You've just created a cool new package for your app. Very nice. Let's check out the file.

NPM Node.js

```
{
  "name": "yetanothertest",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

This is our app information file. Let's install a dependency (third party module).

{CN}®

NPM Node.js

Third-party modules we use are referred to as dependencies. When we install them they are included in a folder called node_modules which is created in our project directory after initialising our package.

{CN}®

Run the command: npm install lodash

lodash is a cool utility library which makes a lot of things quicker for us.

Add this module to your js file using the require method. Usually we use a single underscore for the variable name.

```
const _ = require('lodash');
```

NPM Node.js

```
{
  "name": "yetanothertest",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "lodash": "^4.17.11"
  }
}
```

In our package.json file we should now have a dependencies property with lodash listed (and it's version).

{CN}®

TASK:
Go to lodash.com read the documentation and see what it does.
Try out at least 2 of its methods (some of the array ones are cool, like reverse( ) or pull( ))

If you've done that, get stuck into the docs at nodejs.org and npmjs.com

{CN}®

Node.js

A few extra bits:
node_modules
shouldn't be shared or pushed to git

{CODENATION}

After we've initialised our repo using git init, we should create a .gitignore file (with the dot before it). Inside this file, we can list the files or folders we want git to ignore.
You should add node_modules

Our dependencies are stored in the package.json file. If I deleted the node_modules folder, then ran npm install it will add back the node_modules folder.

{CODENATION}

Node.js

So you run npm install to get the node_modules (dependencies) listed in the package.json file.

Remember to do this if you clone someone else's app from github.

{CODENATION}

Revisiting Learning Objectives

To understand what NodeJS is used for.

To be able to run your javascript files in a Node Environment.

To understand modularity in Javascript.

To be able to export and require modules.

To understand what NPM is and how to use it.