# Getting Started with Node

In this lab, you'll change some of the programs you've written to run in NodeJS.

Actually, that's not true – you'll mostly change the *files* your JavaScript is in; you'll make *minimal* changes to your JavaScript code. Your code is in *HTML* files. Since we're running outside the browser, you'll *copy the JavaScript into separate .js files* to run with NodeJS.

Your tasks will be:

- Download and install NodeJS (If we already haven't done so)
- Run NodeJS with some small examples to check if it's installed properly
- Create and use a NodeJS *module*
- Run the code from the JavaScript ES6+ lab, fixing errors along the way, guided by NodeJS diagnostics.

## Download and install NodeJS

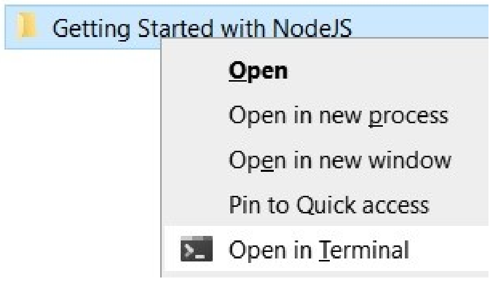
If we haven't done this yet go to *www.nodejs.org* and follow instructions. The Ohs on page 9 and 10 have the info.

As an aside, when I installed some earlier Node versions, the (stoopid) installer wanted to place my node installation in **C:\Program Files (x86)**. This is NOT the location for a 64 bit install! The correct location is **C:\Program Files**.

Good idea to check your %PATH% entry through the command line (shown on OH 11) or in the Environment variables panel in Control Panel.

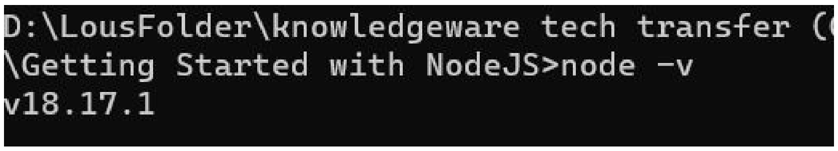
## Run NodeJS

Might as well navigate to the lab directory - *Labs/Getting Started with NodeJS* - and open a command window (or open a command window and navigate there....).



A quick way to get to the directory is to open the labs folder, shift-right-click and select *Open in Terminal* (Might say *Open command window here)*

In the command window, enter **node -v**



```
D:\LousFolder\knowledgeware tech transfer (
\Getting Started with NodeJS>node -v
v18.17.1
```

So far, so good.

If you like you can enter the JavaScript code in the Node REPL as shown on pages 13 through 18; feel free to skip this step.

## Create and Use a NodeJS Module

Here you'll see how to *import external JavaScript code* into your script – analogous to using <script src=...> in an HTML page.

Node imports external code as **modules**. A Node module has *exportable* items that may be referenced by other code. *Any code in a module not specifically exported is private to the module.*

To create a module, code some JavaScript in an external file and *export* the items you want known outside the module. Here's a step-by-step for you to try:

1. Create a file containing code to be used (imported). You already have that code in *makemeamodule,js*. The code is:

```
// myfirstmodule.js
// Just a Node module to show how its done
function fibonacci(n) {
   return n < 1 ? 0
        : n <= 2 ? 1
        : fibonacci(n - 1) + fibonacci(n – 2);
}

var names = ["Lou", "Jake", "Mary", "Cuthberth", "Melissa" ] ;

// This will be PRIVATE to the module
function reverseNames( ) {
   return names.reverse( ) ;
}
```

2. Tell Node that we want to *export* certain functions/variables/classes in our module. Here is the revised makemeamodule.js saved in a different file: *myFirstModule.js*:

```
module.exports.fib =
  function fibonacci(n) {
      return n < 1 ? 0
           : n <= 2 ? 1
           : fibonacci(n - 1) + fibonacci(n – 2);
   }
```

```
module.exports.names = ["Lou","Jake","Mary",
"Cuthberth","Melissa"];

// This will be PRIVATE to the module
function reverseNames( ) {
   return names.reverse( ) ;
}
```

Short story – Append an identifier to the *module.exports* object in the entities you want to expose to other Node code. As an aside, **exports.fib** and **exports.names** work here, too.

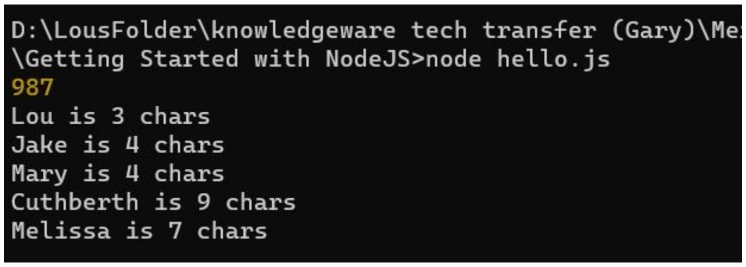
3. Let's use this module in our *hello.js* script.

```
// Hello.js
// Modified to use myfirstmodule.js
var moduleObject = require('./myfirstmodule.js') ;
// Execute the fibonacci function
console.log( moduleObject.fib( 16 ) )  ;

// Do something with the names array
moduleObject.names.forEach( function (name) {
            console.log(name  + " is " + name.length + " chars" )
                        } ) ;
```

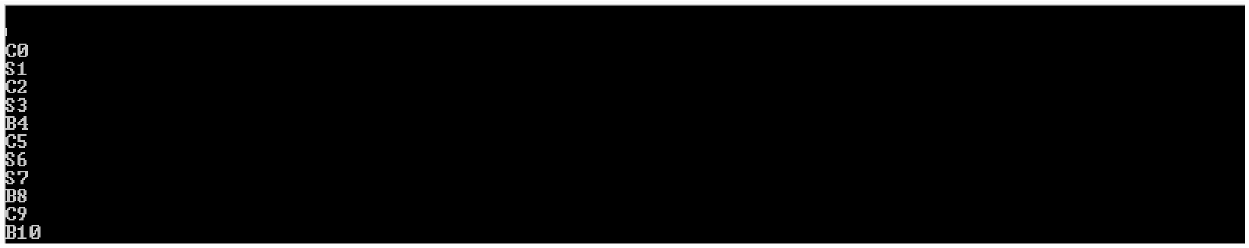
Reference exported items by using the names of *properties of the object* you coded in your *require* call.

4. And run hello.js:



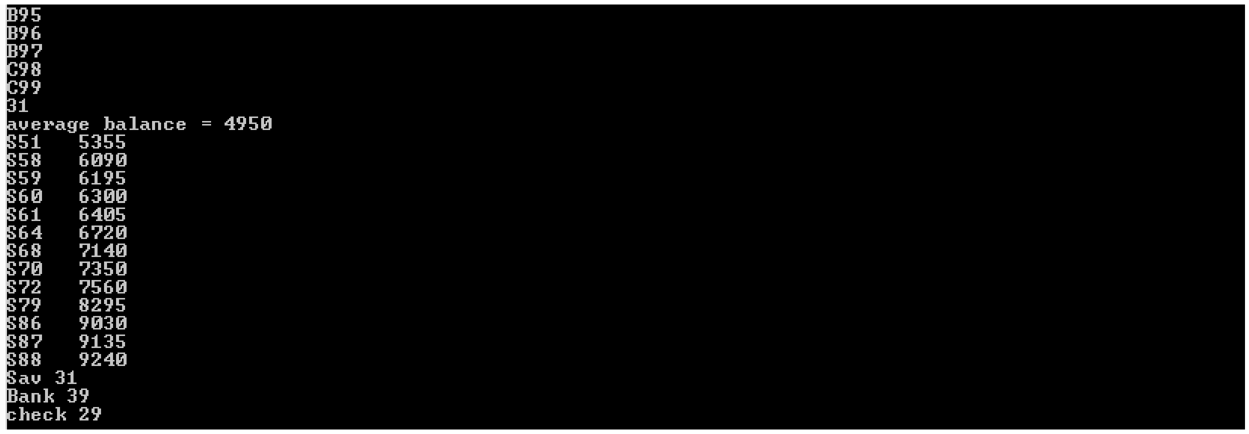
```
D:\LousFolder\knowledgeware tech transfer (Gary)\Me
\Getting Started with NodeJS>node hello.js
987
Lou is 3 chars
Jake is 4 chars
Mary is 4 chars
Cuthberth is 9 chars
Melissa is 7 chars
```

Ok. Final step before you try this yourself. Run *the node-ready version* (solution) of a lab you coded in the JavaScript class. Execute *labSolution.js* with Node. You'll see something resembling:



```
C0
S1
C2
S3
B4
C5
S6
S7
B8
C9
B10
```

Buncha lines omitted



```
B95
B96
B97
C98
C99
31
average balance = 4950
S51   5355
S58   6090
S59   6195
S60   6300
S61   6405
S64   6720
S68   7140
S70   7350
S72   7560
S79   8295
S86   9030
S87   9135
S88   9240
Sav 31
Bank 39
check 29
```

## Now you Try It!

Your task is to reproduce the results shown in the previous screenshot. In other words, you'll take the code you labored over in a previous JavaScript lab and get it to run under NodeJS.

Here's what you need to do:

1. Use the code in *useES6Classes.html,* extract the JavaScript code and save in a file named **mylab.js** (or remove the HTML and save as mylab.js).

   The **only change you need to make** to mylab.js for Node is to **make your account classes known via *require* function calls.** Before you do, you need to change the code in the account class files to **expose the class**.

2. The folder *classesnotasmodules* has the JavaScript code as ES6 classes used in the JavaScript lab. **Each class file must expose the class coded within**. All you need do is code *one line at the bottom of each class file.*

   We are exporting only one item from each file – the class. We can get away with a Node shorthand for this by coding in each file:

   module.exports = <the class name>

   Make the necessary change and save the files.

   The folder *classesasmodules* has a copy of the class files modified as described above (part of the lab solution, actually)

3. Import the revised account classes using ***require*** and you should be good to go!

4. So – to recap: export each class via a **module.exports** statement and use calls to **require** anywhere an external reference is required. Remember to **assign the target of require** to the **class name used in mylab.js.**