**Learning Node**

**Modules – part 1.**

Modules are essential to the way that Node puts together functionality. Node has both user defined modules (either local modules, which you defined, or third party-modules), and core modules.

Our key topics here are:

1. **An overview of modules**
2. **Where do I find and how do I choose modules?**
3. **How do I install modules?**
4. **How do modules provide functionality to other parts of my application (i.e. *export*  functionality and *require(some\_module)*  )?**
5. **How are dependencies of one file on a module documented (i.e. *package.json*)**
6. **Exactly what is export**ed **? and some simple examples**
7. **An overview of modules**When you install node.js (see next paragraph) certain **core modules** will automatically be put on your machine.   
   You can find out what these are by going to the command line interface and typing   
    **npm**
8. **Where do I find and how do I choose modules?**There are several places to find modules and their documentation.

<http://eirikb.github.io/nipster/>

Another search tool - beyond [nodejs.org](http://nodejs.org/)'s homepage and <https://www.npmjs.com/> - is at <http://blago.dachev.com/modules>  
  
At <http://nipstr.com/> you may search for modules and see who wrote them, how popular they are, etc.  
  
You can find documentation on modules at <https://docs.npmjs.com/>  or by typing npm docs moduleName on your command line.  
  
Other places to find lists of modules are:

<https://www.codementor.io/ashish1dev/tutorials/list-of-useful-nodejs-modules-du107mcv3> List of useful modules  
 <https://nodejs.org/docs/latest/api/modules.html> Documentation for all modules

1. **How do I install modules?**To install a module, no matter what folder you are in, you type  
     
    **npm install *my\_module***If you want the module to be available globally you type  
     
    **npm install *my\_module* –g** (The –g flag may go either before or after the name of the module)  
     
   Usually global installs are recommended only for modules whose functions will be used on the command line.  
     
   <http://www.tutorialspoint.com/nodejs/nodejs_npm.htm> states that “Globally installed packages/dependencies are stored in system directory. Such dependencies   
   can be used in CLI (Command Line Interface) function of any node.js but can not be imported using require() in Node application directly. “ but my  
   experience is that you can use require() after a global install.  
     
   If you have already set up your package.json file (see item 5 below) and want myModule to be installed and added to the dependencies you type  
     
    **npm installl *my\_module*  --save  
     
   NOTE: You must access npm outside node – i.e. you must first stop node (clt+c in Windows)**
2. **How do modules provide functionality to other parts of my application (i.e. *export*  functionality and *require(‘some\_module’)*  )?**
   1. **What is a module?** When you first use a module you may think of it as a library of functions. You will ***require (****import) those functions with   
       let someID = require('./someID') ;*This will give you a new object named someID, and you can access its functions as someID.func1(), etc.   
      In actuality, and perhaps not surprisingly, modules are objects and on the prototype for a Module are important and useful methods and properties.   
      One such property is the **exports** object. Initially this is empty, but as your module does its work, it will acquire a value (which might be a function, some JSON, or an object with several key-value pairs, where the values may be functions. ) Using the export functionality , the module will be able to pass these to a script which uses
   2. **What other properties does a module have?** It has the ability to construct a **full path** to itself. So, for example, it knows what directory it sits in. It also has a **require method** which allows it to import functionality from still other modules.
   3. **The parameter for require() has quote marks around the name or path and name.   
      If you do not specify a file extension Node will assume .js, or for a folder look for an index page.**
   4. If a module myModule.js sits in the same directory as your app.js then you import its functionality with   
       **myModule = require(‘./my\_module’);**  
      Note: If myModule is a standard Node module you will say:  
        
        **myModule = require(‘my\_module’);**

**Please notice that the path is required for the modules you coded, even if they are in the same folder as your app.js, but is not needed for the core modules (b/c they are global) and other installed node modules.**

* 1. One critical part of modules, is that each module has its own context and name space. That is, the identifiers for functions and variables are not part of the global name space.   
     The module may have functions (helper functions) other than those which are exposed in the exports object – these (helper) functions are private to the module. For example, you can't call one of the modules functions unless it has been exported.  
     Of course, you know from our work with closures, that the functions which are exported still have access to the module's context and may make use of that context, including the private parts of the module.

1. **How are dependencies of one file on a module documented (i.e. *package.json*)   
   a. Basic answer**: When you are in the folder with your main application (traditionally named app.js) you type  
     
    **npm init**This will generate a series of questions and after you have answered them it will generate a **package.json** file.  
   The package.json file will document these answers, including dependencies (see numbers 3 and 4 above.)  
     
   The references listed below are also in the References document.   
   **Please read the first one (**at freecontent.manning.com)  
   **REFERENCES**<http://freecontent.manning.com/managing-node-js-project-dependencies-with-npm-and-a-package-json-file/> on managing dependencies. Clear, straightforward.  
   [http://www.infoworld.com/article/2984358/application-development/inside-npm-building-and-sharing-javascript- packages.html?phint=newt%3Dinfoworld\_daily&phint=idg\_eid%3Dae8e3ea0bca8a3c279ed6a8235932ec2#tk.IFWNLE\_nlt\_daily\_pm\_2015-09-22](http://www.infoworld.com/article/2984358/application-development/inside-npm-building-and-sharing-javascript-%20%20packages.html?phint=newt%3Dinfoworld_daily&phint=idg_eid%3Dae8e3ea0bca8a3c279ed6a8235932ec2#tk.IFWNLE_nlt_daily_pm_2015-09-22) on what npm will do for you, but less detailed.  
   <https://www.packtpub.com/books/content/using-nodejs-dependencies-nwjs?mc_cid=23ec57dbd9&mc_eid=9aa1af28be> on what modules are shipped with node

**b. Details about versions of modules**These can be found either in the freecontent link above or in Chapter 3 of the Ali Syed book (see references or class by class assignments) Beginning Node.  
  
If at some point you want to write your own module, and add it to the Node.js ecosystem, then the API for modules is at <https://nodejs.org/api/module.html> and for general documentation about modules <https://nodejs.org/api/modules.html>   
  
There are 3 types of modules- core, local, and third-party.

* **Core modules** provide the most important functions, and they are automatically loaded whenever Node starts up. A clear description of them is at <https://www.tutorialsteacher.com/nodejs/nodejs-modules> and they are summarized in:

| Core Module | Description |
| --- | --- |
| [http](https://nodejs.org/api/http.html) | http module includes classes, methods and events to create Node.js http server. |
| [url](https://nodejs.org/api/url.html) | url module includes methods for URL resolution and parsing. |
| [querystring](https://nodejs.org/api/querystring.html) | querystring module includes methods to deal with query string. |
| [path](https://nodejs.org/api/path.html) | path module includes methods to deal with file paths. |
| [fs](https://nodejs.org/api/fs.html) | fs module includes classes, methods, and events to work with file I/O. |
| [util](https://nodejs.org/api/util.html) | util module includes utility functions useful for programmers. |

* **Local Modules** are the ones you write yourself  
  **NOTE: As of about 2019 module names can NOT have capital letters in them (although you may use dashes).**
* **Third party modules** are the ones, other than the Core ones, which you will download from npm. They include many wonderful modules for creating templates for web pages, etc.  
  **NOTE: As of about 2019 module names can NOT have capital letters in them (although you may use dashes).**

**c. Modules and versions**  
 Node.js uses a common system for numbering versions. The versions are numbered as X.Y.Z The Z indicates bug fixes (i.e. each version starts at X.Y.0 and then becomes X.Y.1 etc.) The Y indicates minor updates and the X indicates major updates. When X is *odd* that version is supported for 6 months. When X is *even,* that is an LTS (Long Term Support) version --- and that is what you want.  
 So when you start out you want the highest even number version.  
 As you know, backwards compatibility is always an issue. The folks who run Node.js know this and they have a slightly atypical solution to this. You can have different versions of Node and of Node modules running on different projects. That way, you don't need to worry about a new version killing your project.   
 On the other hand, it makes life a little more complex than with ES6+, where you need only one version, knowing that the ECMAScript people will always maintian backwards compatibility. For example, in Node you will always need to show the dependencies on specific versions of modules.   
 If you want more information about Node versions, go to <https://tamalweb.com/which-nodejs-version>

**d. CommonJS Modules and ESM (EcmaScript Modules) – you can use either.  
 Common JS:** For a long time (i.e. prior to Node) there was a module format known as the CommonJS module. These modules are all JavaScript files, and so they have a **.js extension**.  
 In order to use one of these modules we ***require*** the module.  
 **const some\_module = require(‘./some\_module’);**

**#for some\_module in my local folder**

**const some\_module = require(‘some\_module’);  
 #for some\_module which is a core module**Further documentation on this is at <https://nodejs.org/api/modules.html#modules-commonjs-modules>   
 **ECMAScript Modules: (or ES Modules)** Since the advent of ES6 (aka ES2015), these modules have become the official type. While they are all JavaScript files they have a

**.mjs extension.**

In order to use one of the modules we ***import*** the module.  
 **const some\_module = import from ‘./some\_module’ ;  
 #for some\_module in my local folder**  **const some\_module = import from ‘some\_module’ ;  
 #for some\_module which is a core module.**

Further documentation on this is at <https://nodejs.org/api/esm.html>

**What type of module should I use?** For purposes of this course, it doesn’t matter. All the core modules (& any others you are likely to need) are available in both formats. In addition, node knows (are you using ‘require’ or are you using ‘import’?) which type of module you want.  
 *We will use the CommonJS modules and require in this course.* There are some other differences between the two types*. CommonJS modules are loaded synchronously, while ES Modules are loaded asynchronously.* A good disucssion of this may be found at <https://www.howtogeek.com/devops/how-to-use-ecmascript-modules-with-node-js/#:~:text=ECMAScript%20or%20ES%20Modules%20%28ESM%29%20rely%20on%20the,export%20const%20helloWorld%20%3D%20%28%29%20%3D%3E%20%22Hello%20World%22%3B>