**Assignments for Unit 8**

**Material you will be using for these assignments --- as always, I like to give you a choice of materials. Here are 4 resources I particularly like:** (The first two are more elementary, and the last two are more complete. Different authors treat the Core Modules of Node.js in different orders – so use the table of contents for the books.)

i. Mosh's Video at [https://www.youtube.com/watch?v=TlB\_eWDSMt4&t=1982s](about:blank) This provides a great introduction to Node.js

ii. The tutorialspoint pages at [https://www.tutorialspoint.com/nodejs/index.htm](about:blank) These go somewhat farther than Mosh's video --- but they are not as detailed in their explanations

iii. Beginning Nodeljs by Basarat Ali Syed at   
<https://acm.percipio.com/books/08c96e50-f32f-11e6-ad5d-0242c0a80607#epubcfi(/6/20!/4/2%5Bepubmain%5D/2%5Bch2%5D/2/8%5Bover-02%5D/2/1:0)>   
This is an older but excellent book. Chapters 1 and 2 review some fundamental facts about JavaScript. For the rest, please use the Table of Contents.

iv. Get Programming with Node.js by Jon Wexler at   
<https://acm.percipio.com/books/84e7ae8b-ede5-4c6e-beb9-c145b43f94a2#epubcfi(/6/4!/4/2%5Bepubmain%5D/2%5Bg269b43ad-56bf-478c-98a5-10dad5ae422c%5D/2/2/1:0)>   
As with the previous book, use the Table of Contents to find what you are interested in.

Of course, the **References** page has additional references, but these four are likely to be the first place you go. In places I will also make use of excerpts from the Brad Dayley book which should be in the Simmons Library.  
  
Finally ,you may find [**https://www.tutorialspoint.com/nodejs/index.htm**](about:blank) and [**https://nodejs.org/api/**](about:blank)  
to be useful for quick and more detailed references to look up names of methods, etc.

**Assignment 0**: **Preparing for Node; What is Node.js and how is it put together?**

**You will be using the files for items 0 -5 on the Chapter 16 page** [**http://web.simmons.edu/~menzin/CS321/Unit\_8\_Server\_Side\_Node/**](about:blank)

**Item 0 (which has 2 files) is about 'preparing for node' and items 1-5 are about how node is put together. In addition, I am asking you to watch (with my guidance on this page) 37 minutes of a video. (The video is also about how node is put together.)**You will be downloading node and copying some code from the video, but you won't (yet) be creating any code of your own.

This should all go very smoothly, but if you have any problems, please shoot me an email and/or ask in class.

**Preparing for Node:  
Read:** ES6 Reminders and a Few New Pieces of Syntax (the link is on the chapter page) and make note of anything mysterious.  
**Review**: Commands You Will Need for Using the Terminal and Writing Node (also linked to from the chapter page.)  
 **What is Node.js and how is it put together?**  
**Read** The files in items 0-5 most of which are review.  
 Commands for Writing Node.js - in this reading the first three paragraphs should be review.   
 The fourth paragraph Is something we will use in a little while.

We are going to use the video at [https://www.youtube.com/watch?v=TlB\_eWDSMt4](about:blank)   
This is an hour long video, so you will watch it in pieces.  
Please **watch thru minute 33.**

I think the introduction to node (minutes 0-13) is particularly good.  
 Around minute 13, Mosh write his first node application.   
 He creates a new *top level* folder , cd's over to it and then by typing **code .** he is in VS Code with that folder open. You can do this anyway you want, but I suggest that you make a top-level directory *myNode* and then make a folder in it for each project. So you might have a folder first\_app  
Inside that folder you will write your JavaScript code and save it as app.js (Note you could create these folders in the terminal or even before you open the terminal. In fact, if you save app.js in the right place, then in the first\_app folder, in the terminal, you could run it by typing **node app  
  
 After** the first app is completed around minute 15, you should try to replicate it, first in VS Code and then my opening up your terminal, navigating to the firstApp folder and typing **node app.js**

**NOTE: the .js is superfluous --- Node assumes (unless told otherwise) that every file is js.**

[**https://vscode.readthedocs.io/en/latest/nodejs/nodejs-tutorial/#:~:text=Node.js%20Tutorial%20in%20VS%20Code%20Node.js%20is%20a,TypeScript%20languages%20out-of-the-box%20as%20well%20as%20Node.js%20debugging**](about:blank#:~:text=Node.js%20Tutorial%20in%20VS%20Code%20Node.js%20is%20a,TypeScript%20languages%20out-of-the-box%20as%20well%20as%20Node.js%20debugging)**.** will remind you about running scripts and debugging them (including Node.js modules) in VS Code.

Leading up to minute 33, Mosh slowly and clearly goes thru

* **how to create a module**
  + In Node, every js file is a module, but you won't be able to use it unless you **export** some of its functions and values.
  + Every module (i.e. every js file) starts with a **module** property – and earlier in the video Mosh has shown you show to look at it with console.log
  + That module has a property **exports**. That is, **module.exports** is an *empty object.*You can assign key-value pairs to it in the usual ways --- that is you can say  
     module.exports.func1 = func1; //where func1 was defined earlier in the module  
     module.exports.func2 = func2; //etc  
    Or you can say  
     module.exports = {func1:func1,  
     func2:func2}  
    Sometimes, when you are exporting only one function you will write  
     module.exports = oneFunc;  
    This last line kills off the object you used to have so be careful about using it.
* **how to use what that module exposed**
  + We use the **require** method to grab what our module has exported.
  + The custom is to use a const which has the same identifier as the name of the module you are importing/requiring.
  + If it is a **core module** such as http you code:  
      
     const http = require('http'); //Note that Node assumes any file is js unless   
     //you specify a different extension

//Note that for **core modules there are quotes but no  
 //path information.**

* + if it is a **local module in the same folder** (named someModule.js) you code:  
      
     const someModule = require('./someModule') //Note the ./ and quote marks!!
  + If the module is local module in a **different folder** you write the path to it *inside the quotes* – e.g. start with .. to go up one level, etc.
  + Mosh notes that if someModule exported an object (which you have now imported and named someModule) then you access its content using the usual dot notation – e.g.  
    someModule.func1  
    But, *if someModule exported a single function, not in an object* then someModule is that function.

Please **watch** the video up to minute 37:39

**Reminder about modules --- they all have the following properties:**

* **module.exports = { } //**what we just discussed above
* **require //**so we can require other modules
* **\_\_filename //**full path to this module, starting w/ 2 underscores
* **—dirname //**full path to this directory, starting w/ 2 underscores.

The video continues on with brief introductions to several core modules: path, os, fs, events   
 (with the EventEmmitter class) , and http.   
 *Please pay particular attention to his discussion of events and event emitters.*  
 Emitting an event is just like triggering an event in jQuery. You will see that our work on   
 events in jQuery stands us in good stead here. You also need to watch this more than once ---   
 watch it all the way thru to the point where he summarizes, and then go back and watch it   
 again. What he is really showing you is how many modules in Node have extended the   
 EventEmitter class.  
  
 If you want a different, short explanation, look at   
 [https://www.tutorialspoint.com/nodejs/index.htm](about:blank) through the page on the Event Loop.

**In class we will make sure that you were able to download and install node & we will do a few very simple apps.**

*Coming attractions*: After assignment 1 we will go thru an example of an event emitter such as that in tutorialspoint. As Mosh's video indicates, event emitters play an important role in Node. (Good thing we spent time on them when we studied jQuery!) Node uses the on() method to attach event handlers, just like in jQuery and has an emit() method which corresponds to the trigger() method of jQuery.

**Assignment 1- Getting to Know Important Core Modules and Building a Simple Project**The first three pieces I am asking you to read (both of which are "Chapter 3"s is a review and deepening of the material from Assignment 0. The second part of this assignment is new material.  
  
**The review and deepening material:**

**Read the 6th file: Modules in Node- Part 2, which is on the page for Chapter 16.**

**Read: Chapter 3 in** [Get Programming with Node.js](about:blank) which is at <https://acm.skillport.com/skillportfe/main.action#summary/BOOKS/RW$275850:_ss_book:147200> This very short chapter will review the structure of a Node application --- note that the aurhor uses main.js for his main javascript file, though we will use **app.js** (which is the custom in Node, although you will also see index.js.)   
We will use only two of the npm commands: **npm init** and **npm install *somePackage -someFlag***  
BTW. I really like his diagram near the end of the chapter.

NOTE: In Node.js a **module** is a standalone js file or files which may be accessed by using require(). A **package** is a collection of modules (i.e. a folder or directory) , with a package.json file (which describes it). The formal description of this is at [https://docs.npmjs.com/about-packages-and-modules](about:blank) The important piece here is that a module does not need to have a package.json description and is typically a single file.

**Read Chapter 3 in** [Beginning Node.js](about:blank) **which is at** [**https://acm.skillport.com/skillportfe/main.action#summary/BOOKS/RW$39899:\_ss\_book:77629**](https://acm.skillport.com/skillportfe/main.action#summary/BOOKS/RW$39899:_ss_book:77629)Some of this you need to understand thoroughly now, and some (from Important Globals on) you should read to know it is there, but don't need to grapple with the details yet.   
It is important to understand how data is shared across modules --- this is discussed as "shared" and "factory" folders and we will review it together in class.  
The later part of the Chapter is a brief overview of some of the most useful parts of the Core modules of Node. Js

**Starting a project – a review of the steps**

* Make a folder for your project
* In that folder put an app.js file (even if it is empty)
* Navigate to your folder and type  
   npm init  
  This will create a package.json file --- and you 'll need to fill iin your name etc.  
  This file will also keep track of your dependencies.
* We will use npm install to install (in the node\_modules folder in our project folder) the modules we will be using in this project  
   npm install *packageName*

Then when we reload package.json whatever you install'ed gets added as dependencies in package.json   
Alternatively, when you install a new module or package, you may say  
 npm install *packageName –save*and the package.json file will be updated.

**The new material**

**Watch** the rest of the Mosh video at [https://www.youtube.com/watch?v=TlB\_eWDSMt4&t=1982s](about:blank)   
Again, there is an overview of the Core modules.

One of the modules which we will use heavily is the npm module. It is used to create a server.

Let's start, however, with a simpler module, the Path module.

The general documentation for all modules is at [https://nodejs.org/api/](about:blank)

Look at the Path module at [https://nodejs.org/api/path.html](about:blank) Look at the example at   
 [https://nodejs.org/api/path.html#path\_path\_parse\_path](about:blank#path_path_parse_path) (POSIX is a standard for OS'es) and at  
[https://nodejs.org/api/path.html#path\_path\_relative\_from\_to](about:blank#path_path_relative_from_to)

Now let's think about the http module.  
  
 When you come to the *http* module, you should know that many people use **req** and **res** to stand for **request** and **response.** You can name those parameters anything you want --- for the built in methods that use them the first parameter will be the request and the second will be the response. But, it's probably easiest to stick with req and res for their names.

The http module is based on (is an extension of ) the event module. That is because receiving a get or post is an event and we want to listen for it and respond to it. (There are other modules which also extend the event module, and you may wish to define your own events.) So let's start by understanding the event module and its EventEmmitter. Please notice that emit(*someEvent*) is like the trigger( ) method we used in jQuery.  
  
Please read the following tutorials *in this order:*

[https://code.tutsplus.com/tutorials/using-nodes-event-module--net-35941](about:blank)

[http://www.tutorialsteacher.com/nodejs/nodejs-eventemitter](about:blank)   
 (all the way to the end with  2 nice patterns)

[https://www.hacksparrow.com/nodejs/eventemitter-tutorial-examples.html](about:blank)

   Note: to understand this, do the following:

* Make a folder hacksparrow
* Copy the code for radio.js into a file in that folder

//radio.js file from https://www.hacksparrow.com/nodejs/eventemitter-tutorial-examples.html

var util = require('util');

var EventEmitter = require('events').EventEmitter;

// @station - an object with `freq` and `name` properties  
var Radio = function(station) {

// we need to store the reference of `this` to `self`, so that we can use the current context in the setTimeout   
 //(or any callback) functions

// using `this` in the setTimeout functions will refer to those funtions, not the Radio class  
 var self = this;

// emit 'open' event instantly  
 setTimeout(function() {  
 self.emit('open', station);  
 }, 0);

// emit 'close' event after 5 secs  
 setTimeout(function() {  
 self.emit('close', station);  
 }, 5000);

// EventEmitters inherit a single event listener, see it in action  
 this.on('newListener', function(listener) {  
 console.log('Event Listener: ' + listener);  
 });  
};

// extend the EventEmitter class using our Radio class  
util.inherits(Radio, EventEmitter);

// we specify that this module is a refrence to the Radio class  
module.exports = Radio;

* Copy the code for example.js into another file in that folder.  
   var Radio = require('./radio.js');

// a station object  
var station = {  
 freq: '80.16',  
 name: 'Rock N Roll Radio',  
};

// create an instance of the Radio class  
var radio = new Radio(station);

// add an 'open' event listener  
radio.on('open', function(station) {  
 console.log('"%s" FM %s OPENED', station.name, station.freq);  
 console.log('♬ ♫♬');  
});

// add a 'close' event listener  
radio.on('close', function(station) {  
 console.log('"%s" FM %s CLOSED', station.name, station.freq);  
 });

* In either the terminal or VS Code, type npm init and work thru what it asks
* Type node radio
* Type node example

You should be able to see what happened --- both the emitting of the events in radio.js & their event handlers firing and the emitting of events in example.js and their event handlers firing.  
  
NOTE: In the hacksparrow tutorial the only modules which were required were core modules (util, events) and one you created (radio). If you had needed other modules, then *before* you asked for npm init you should have installed those modules. You may also choose to use certain versions of core modules. To see this in action, make a new folder hacksparrow2 and copy into it the radio.js and example.js files. Now type  
 npm install util  
 npm install events  
 npm init  
Now examine the files in your hacksparrow2 folder, You should find a folder node\_modules. Open it to see all the files which were added to your project. (This is why we don't copy the core modules when we publish our module.)

[https://www.sitepoint.com/nodejs-events-and-eventemitter/](about:blank) OPTIONAL

Then look at the documentation for event emitters at [https://nodejs.org/api/events.html#events\_emitter\_on\_eventname\_listener](about:blank#events_emitter_on_eventname_listener)   
Browse the list of methods, and look at the examples for event.on and event.off   
What is the newListener event?

***Make a note of* anything which is not clear, to discuss in class.**

**Assignment 2**

Our goal is to create the back end of an interactive web site.

Let's think about the tasks we need to carry out:

1. **Set up a server.**
2. **The server needs to know when a request has been made**
3. **The server needs to know about the request - which page was it made from**
4. **The server needs to know if data was sent to it**
   * **Was it a get or a post?**
   * **What values were sent to it**
5. **Depending on the results of item 4, the server needs to put together an appropriate response**
6. **This may require that the server communicate with a database**
7. **The server needs to send off the response.**

We have already seen some parts of this.

We know that the http module will set up a server, and that we can make it listen on a port.  
  
Note: It is common to use 3000 for development work. We use 80 for http and 443 for https.  
A list of the most common ports can be found at [https://opensource.com/article/18/10/common-network-ports](about:blank) and a more complete list (including for specific databases) is at [https://geekflare.com/default-port-numbers/](about:blank)

**So we know how to take care of items 1 and 2 with**

**const http = require('http');  
 const server = http.createServer( (req, resp) => {  
 //do stuff  
 } );  
server.listen(3000);**

**We actually know how to do item 3 also, even if our initial way to do so is a bit clumsy.** (Express will make this more elegant.). To see this, examine the code in our server2 example:

**myNode/server2/app.js**const http = require('http');

const server = http.createServer((req,res) => {

  if (req.url === '/about')

     res.end('The about page')

  else if (req.url === '/contact')

     res.end('The contact page')

  else if (req.url ==='/')

     res.end('The home page')

  else {

     res.writeHead(404);

     res.end('Page not found')

  }

  })

  server.listen(3000)

Here you can see what we do:

**Look at req.url and test it to see which url made the request.**

Pretty clearly, http will be an important module. You can find its documentation at [https://nodejs.org/api/http.html](about:blank)   
If you scroll down to the Incoming Message section, you will find the methods which may be applied to req, the request. For a **GET** we will be able to parse the url and find the appended information from a form, the username and password, etc. WARNING: A number of these methods were deprecated in 2021, and beware of code from older sites and books. (We will do this in detail soon, using appropriate methods, but you can see that Node.js has long had methods for handling input from forms, whether from a GET or a POST.

If you want to see a review of this and a complete example, a pretty clear one is in Lesson 4 of   
*GET proramming with Node.js* at <https://acm.skillport.com/skillportfe/main.action#summary/BOOKS/RW$275850:_ss_book:147200>

There is another problem beyond that of avoiding deprecated methods. That is that, on a large website, we may be receiving requests from many different pages and we would like to have an efficient way to manage those requests. So in Assignment 3 we are going to learn about Express, which will solve both those problems for us.

Before we turn to our overview of the file system, there is something we can do to make our lives more efficient. In Node there is a package **nodemon** which will automatically restart your server whenever it detects a change in your project.

There are 2 ways to use nodemon:

1. Install nodemom globally      *node install -g nodemon*

    In the terminal (rather than typing *node app*)  type *nodemon app.*

This willl make nodemon run your app (rather than node) and give you the automatic restarts.

*2.*Install nodemon either locally or globally (okay to install it with the --dev flag)

    In your package.json file, by hand edit it so that the scripts property includes a dev script which has *nodemon app.js* or *nodemon app*For example, if you installed nodemon in dev mode only (*npm install nodemon --save -dev*if you are installing after the init) your package.json file might look like:

{

"name": "abc",

"version": "0.0.1",

"description": "my server",

**"scripts": {**

**"dev": "nodemon app",  
 “start”: “node app”**

**},**

"devDependencies": {

"nodemon": "~1.3.8",

},

"dependencies": {

}

}

Notice that in the package.json file I have edited the scripts object. If I type *npm run dev* then the dev script (i.e. my app under nodemon) and if I type *npm run start* then it will run under node. It may seem silly to bother with editing the scripts tag rather than just using node or nodemon to run our app, but in fact, you may have several other scripts in there – e.g. scripts for testing, etc.

If you want a short video on this, there is a clear one at <https://www.youtube.com/watch?v=kV6MJ9W4whM> (You don’t need to watch the material about configuring nodemon which starts around minute 3.)

If you want to see examples of multiple scripts for linting, testing, etc. look at <https://discordjs.guide/improving-dev-environment/package-json-scripts.html#adding-your-first-script>

The file system: At this point, I would like you to have an overview or what we are going to do.

**READ:**

* Go to [https://www.w3schools.com/nodejs/nodejs\_http.asp](about:blank) and **read** the sections on the http module, the file sytem and the url module.   
  *NOTE: As of version 17 (2021) certain parts of the URL module were deprecated/marked “legacy”. We will use the current WHATWG URL API, but beware of older material which uses the then-current URL functionality. You will see current code in the exercise for the synchronous work after this assignment.*  
  It's already clear that we need the http and url modules. Actually, we also need the fs module (the file system) in order to write pages which are more than one line long.
* Just as we needed to examine *events* to understand the http module, we need to look at *streams*  and *buffer* in order to understand the fs module.   
  A nice overview of buffer, fs and streams can be found at [https://www.tutorialspoint.com/nodejs/nodejs\_buffers.htm](about:blank) Please **read** these pages. Don't worry if the material on buffers is a little obscure.   
  I hope to keep this as high level as possible – but that said you should understand how these modules are put together.  
  The important points you need to understand are that:
  + **Buffers** are used for binary data – imagine images etc.
  + When you declare a buffer you need to know what size it will be.
  + You can do the kinds of thingg with bufffers that you expect to do to files – read from them, write to them, concatenate them, etc.
  + **Streams** are the underpinning for the file system.
  + **Streams** may be readable, writable, duplex (both) or transform (take an input steam, do something to it and produce an output steam.)
  + Streams may be piped – i.e. the output from one stream is sent as the input to the next operation. This is similar to chaining operations.
  + **File** operations all come in pairs – synchronous and asymchronous. I hope to stay away from synchronous.
  + The tutorialspoint page on the File system has a table of the flags (modes ) for working with a file. There are a few subtle differences – e. g. both r+ and w+ allow reading and writing to a file, but w+ will create a file If it doesn't exist and r+ will raise an exception then. You should be aware that there are such subtleties, but certainly not memorize them. We will mainly be doing very straightforward reading and writing.

An *optional* but very nice discussion of Streams is found in the last part of Chapter 5 of Syed's Beginnning Node.js book <https://acm.skillport.com/skillportfe/main.action#summary/BOOKS/RW$39899:_ss_book:77629> (Use ctl-f to look for Buddha, or start right after Listingi 5-39.)

**Read** the very short chapter 5 of the *GET proramming with Node.js* book <https://acm.skillport.com/skillportfe/main.action#summary/BOOKS/RW$275850:_ss_book:147200> . If you think about a complex web site, clearly we will need to handle many requests from a lot of different pages – Listing 5.6 is just the beginning   
Do not let his use of template literals in lines like the one below bother you:

console.log(`Method: ${getJSONString(req.method)}`);

Remember that this is just a shorthand way to say for   
  
 console.log(`Method: ` + getJSONString(req.method));

And that req.method will return get etc. as appropriate.

**Do: Create an account for yourself at npmjs.com  
 Poke around a little – what are the most popular frameworks?   
 Search (in the search box) for jQuery.**

We don't have time to learn a lot about npm, but here are a few links should you wish to go further:

* [https://docs.npmjs.com/](about:blank) of course, the documentation
* [https://www.youtube.com/playlist?list=PLWkguCWKqN9PI8zI57E4FsxHbcPbrL8rp](about:blank) Bogdan Stashchuck's tutorial on npm (free).
* [https://docs.microsoft.com/en-us/visualstudio/javascript/npm-package-management?view=vs-2019](about:blank) Microsoft's information about using npm with Visual Studio
* [https://www.sitepoint.com/npm-guide/](about:blank) Sitepoint's very basic tutorial (you know most of this already, but there is info about updating modules etc.)
* [https://docs.npmjs.com/creating-and-publishing-scoped-public-packages](about:blank) how to create and upload a public package (The menu on the left will also link to creating private packages.)
* Some people prefer yarn to npm. A reasonable introduction is at [https://www.sitepoint.com/yarn-vs-npm/](about:blank) but if you search for *npm vs yarn* you will find many other articles.

**Assignment 3 – on Express and handlebars**

**Express allows us to**

* **Handle multiple routes (endpoints)**
* **Extract data from forms sent with a GET or a POST**
* **Write and assemble in any order we choose small pieces of code called *middleware.***For example, different routes may use different pieces of middleware
* **Make use of a template engine so that we can effficiently code a complex but coherent website.** For example, if we have an e-commerce textbook site we might want the same headers and nav bars on all our pages, but have subsites for buying or renting books.

**Read one of the following short introductions to Express:**

* **Chapter 7 Introductionto Express in Syed’s book Beginning Node** [**https://acm.skillport.com/skillportfe/main.action#summary/BOOKS/RW$39899:\_ss\_book:77629**](https://acm.skillport.com/skillportfe/main.action#summary/BOOKS/RW$39899:_ss_book:77629)
* **Chapter 8 Setting up an app with Express.js in Wexler’s book Get Programming with Node.js** [**https://acm.skillport.com/skillportfe/main.action#summary/BOOKS/RW$275850:\_ss\_book:147200**](https://acm.skillport.com/skillportfe/main.action#summary/BOOKS/RW$275850:_ss_book:147200)
* **Chapter 18 of the Brad Dayley book Node.js MongoDB, etc. in the Simmons Library.**

**Read about template engines – browse very quickly** [**https://blog.logrocket.com/top-express-js-template-engines-for-dynamic-html-pages/**](https://blog.logrocket.com/top-express-js-template-engines-for-dynamic-html-pages/) **We will be using handlebars (which was insprired by mustache.)**

**Go to the Links for Handlebars and read/watch one of the tutorials.** It’s okay if you don’t understand all of it--- I want you to get the lay of the land. **NOTE:** In Express we will have a folder **layouts** which will hold complete HTML pages (although they may have pieces to replace/fill in) and a folder **partials** which will hold frequently used elements of a web page, such as nav bars, headers, etc.  
Note: {{ }} are used to enclose expressions in handlebars – see for instance <https://www.geeksforgeeks.org/handlebars-templating-in-expressjs/>   
It also uses {{{ something }}} where something is a piece of HTML which will be placed there.

There are many books and tutorials on Express (please see the doc on Links for Express and Handlebars.) Two of my favorite videos are”  
 Brad Traversy at <https://www.youtube.com/watch?v=L72fhGm1tfE&t=1828s>

Mosh Hamedi at <https://www.youtube.com/watch?v=pKd0Rpw7O48>

We are going to work thru the Brad Traversy video. Please **watch the first 18 minutes of the Brad Traversy video, implementing what he does at each step. 24 hours before the synchronous session please email your instructor with info about anything that didn’t work or that you got stuck at. If you had trouble installing Express, please check out** <https://stackoverflow.com/questions/62207786/i-am-facing-problem-in-npm-install-express-save>

**Before class** please read the file 11. Introduction to Handlebars in the section of “Files to go with the assignments.

**Assignment 4 – on connecting to a MongoDB database.**

.

**Have then read 6\_ Module part 2 and also watch the rest of the Mosh video from Assignment 0 and also look at the tutorialspoint pages**

**Reference the Syed book**

**Ctl+c to exit the REPL loop**