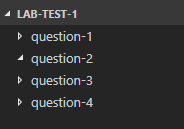
**COMP 3123 – Lab 5 – Node Test**

**Developer Note:**When working on your questions, please create separate folder for your work. This way you won’t putting all your code in the same file, which can pollute the global name space. In short, it will prevent you from overwriting your own work and causing your code to compile incorrectly.

Organize your folder structure in this way.



**Lab Test Submission**

Please submit your test at the end of the lab, using either GBLearn web account or GitHub. If you are submitting via pull request in Git, let your instructor know your submission will be there.

**Question 1: Modules and Unit Testing**

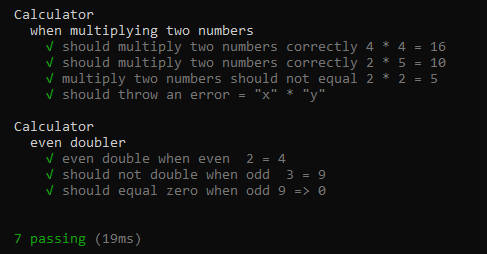
**Part 1**

1. Open a command prompt create a directory for **question-1.**
2. Open Visual Studio Code and open the folder **question-1**
3. Create a file named **calculator.js**.
4. In the **calculator.js** file create two methods named **multiplyTwoNumbers** and **evenDoubler** with the following requirements.
   * **multiplyTwoNumbers**
     + will take two parameters x and y
     + will multiply the two parameters and return the result
     + will throw an error if either x or y is not an integer.
   * **evenDoubler**
     + will take one parameter x
     + will multiple x by x, if x is an even number
     + will return zero, if x is odd
     + will throw an error x is not an integer
5. Export from you calculator module, the two methods **multiplyTwoNumbers** and **evenDoubler.**
6. Create an **app.js** file and import the calculator module.
7. Use the calculator module to output the following when running app.js on the command line.



**Part 2**

1. Install third party modules, **should.js** and **mocha** using the node package manager (npm)
2. Create a new file named **tests.js**
3. In **tests.js**, require the **calculator** module and **should.js** module
4. Write the following unit test cases for the calculator methods using should.js. When running **mocha tests.js** at the command line the following tests should be executed.



**Question 2: Buffers and Streams**

**Part 1**

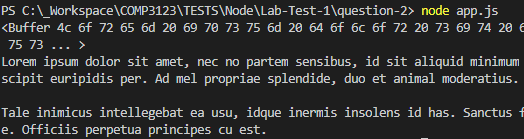
1. Open a command prompt create a directory for **question-2.**
2. Open Visual Studio Code and open the folder **question-2.**
3. Create a file named **apps.js** and require the built-in node **file system** module.
4. Create a **readable stream** to a file named **data.txt** (contents found in link below)

<https://drive.google.com/file/d/1uFTgg1aEf6nFoO4HGELqOY6Uj3bzRHO_/view?usp=sharing>

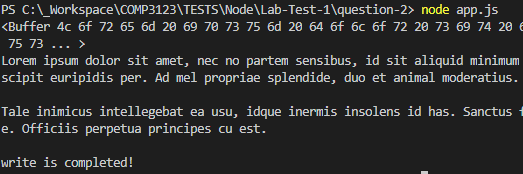
1. Subscribe the **readable stream** to the **‘on’ data event**, that will output the contents of the **buffer** as the data is read through the stream. The following output is expected.



1. In the same callback function of the **‘on’ data event**, convert the data stream in the buffer to string and output as data is read. The following output is expected:



1. Create a **writeable stream** to a file named **output.txt.** Run **node app.js** at the command line. The expected result is all the data from file **data.txt** will be written to the file **output.txt**
2. Subscribe the **readable stream** to the **on ‘end’ event** with a callback function. In the function **close the writeable stream** and output the message the write is completed. The following output is expected.



**Part 2**

1. Create a new file named **pipe.js**. Require the node built-in module for file system and zlib
2. Create a readable stream for **output.txt.**
3. Using the pipe command as the data is read from the file, pipe the data into a compression stream that will create a file named **output.txt.gz**

**Question 3: Web Requests and Routing**

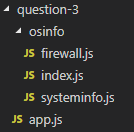
**Part 1**

1. Open a command prompt create a directory for **question-3.**
2. Open Visual Studio Code and open the folder **question-3**
3. Create a file named **app.js** and use the starter code below to create a web server.

<https://nodejs.org/en/about/>

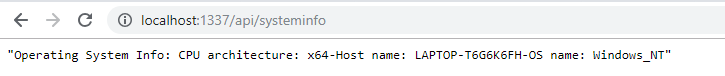


1. Create a **osinfo** folder with the following file structure:

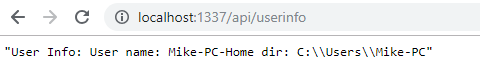


1. In the **systeminfo.js** module, create the following methods and return the following data using the built-in os module <https://nodejs.org/api/os.html>
   * **getSysteminfo()**
     + CPU Architecture
     + Hostname
     + OS Name
     + Combine the above information and return in string format
   * **getUserInfo()**
     + UserInfo
     + Username
     + Home Directory
     + Combine the above information and return in string format
2. In the **firewall.js** module, create a following method:
   * **getStatuses()**
     + returns an array of string statues ie. OK, ALLOW, DENY, BLOCK
3. Use two requires to require both the **systeminfo.js** and **firewall.js** module in the **index.js** file and then export them both to make them available to the **app.js** file
4. Require the **os module** in the **app.js**. *(\*\* if you cannot figure how to use index.js to export multiple modules, then use two requires for both firewall.js and osinfo.js directly in the app.js file)*
5. By inspecting the URL in the request handle the different routes and take the following actions:

* **The default route ‘/’**
  + Header response will be **200 OK**, with **Content-type application/json**
  + Return ‘No data found’ to the browser
* **The route ‘/api/systeminfo’**
  + Header response will be **200 OK**, with **Content-type application/json**
  + Use the **os module getSystemInfo()** to return the following
  + (Note: Your system will display difference specifications here).



* **The route** **‘/api/userinfo’**
  + Header response will be **200 OK**, with **Content-type application/json**
  + User the **os module getUserInfo()** to return the following.
  + (Note: Your system will display difference specifications here).



* **The route** **‘/api/firewall’**
  + Header response will be **200 OK**, with **Content-type application/json**
  + User the **os module getStatuses()** to return the following.



* **Unknown routes** that do not match any named routes
  + Header response will be **404 Not Found**, with **Content-type application/json**

**Question 4: Event Emitter and Callbacks**

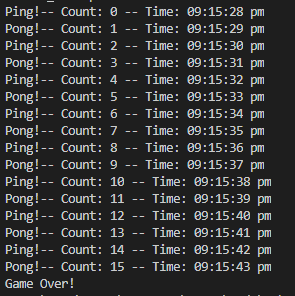
1. Open a command prompt create a directory for **question-4.**
2. Open Visual Studio Code and open the folder **question-4**
3. Create a file named **ping-pong.js** with the following folder structure



1. Use **npm** to install **moment** to handle date and time.  
   <http://momentjs.com/docs/>
2. The application should have the following requirements:

* + Set up two events ‘Ping’ and ‘Pong’ with a 1 second delay between responses.
  + Display the count and the Timestamp for every response.
  + Stop emitting after a count of 15 and return the message ‘Game Over’

1. Using the **events and the event emitter, callbacks** and **setTimeOut** functions create the following response.



1. To start the event emitter chain use the following:

