COMS/SE 319: Construction of User Interfaces
Spring 2022
LAB Activity 2 –Node.js

Download the sample code: Spring2022-Lab03-SampleCode.zip on Canvas.

Task 1: Node.js

In this lab activity, we recommend you to use text or code editors to edit javascript files.

Learning Objectives:

- **Get started with Node.js**
- **run simple js programs on desktop**

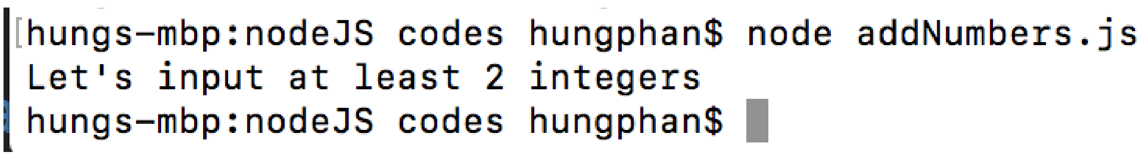
What is Node.js?

- Javascript became very popular on browsers.
- node.js developers wanted to make javascript run on the desktop.
- bundled javascript VM (google's V8) to allow one to create desktop programs in js.
- so now one can run js on the desktop!
- Also, a huge number of libraries exist.
- now one can easily create a web server using some of these libraries

Step 1: Install Node.js

You can download and install Node.js at https://nodejs.org/en/download/ . Besides, the portable version in http://nodejsportable.sourceforge.net/ or https://github.com/yjwong/nodejs-portable-runtime should work. They are basically downloading node.exe, node.lib and npm.zip from the node.js website. If the above links do not work, navigate to http://nodejs.org/dist/latest/win-x64/and download both node.exe and node.lib.

If you finish the installation successfully, you should be able to run Javascript files by node.js by typing node or node.exe in your computer:



```
[hungs-mbp:nodeJS codes hungphan$ node addNumbers.js
Let's input at least 2 integers
hungs-mbp:nodeJS codes hungphan$ 
```

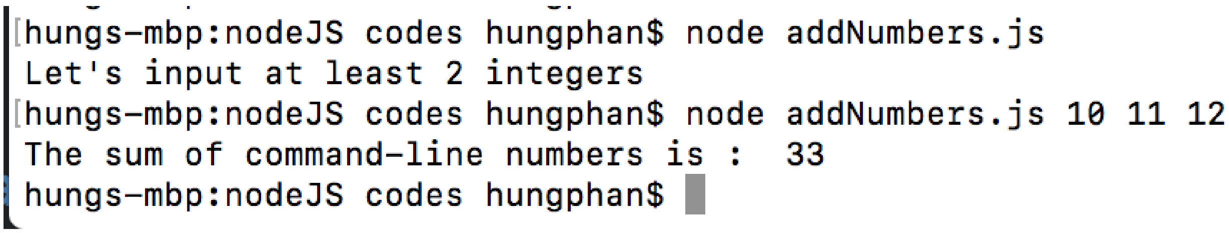
You will see the expected code after editing JS files in the end of each tasks.

Step 2: Run simple js code

- Open the file **addNumbers.js**
- **Write code to do some functions as follows:**
  A. Print a usage statement if arguments are less than two.
  B. Assume that the arguments are a variable number of numbers. Print their sum.
- Use the following statements:
  - **console.log ---** used to print to the terminal
  - **process.argv** --- an array of command-line arguments
    + The process.argv contains whole command-line invocations, including the **"node"** and **"addNumbers.js"**. So that, the input numbers (like 10 11 12 in the following screenshot) are passed as the 3rd element of process.argv. See the sample code and understand why the loop on line 7 start with **i=2.**
  - Example usage: Type these commands into terminal/ command prompt and you should see these output

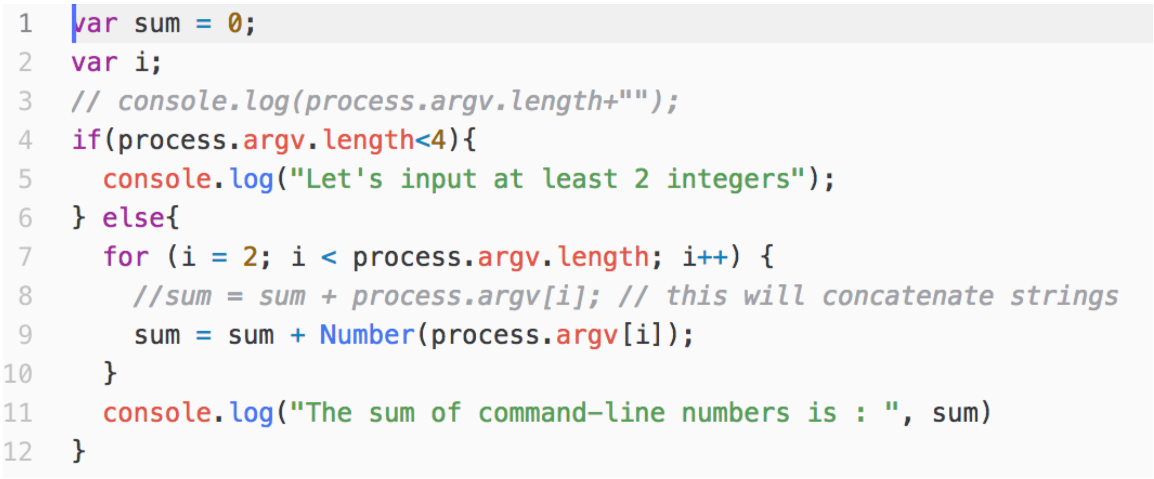
– **node addNumbers.js**

– **node addNumbers.js 10 11 12 (prints "sum is 33")**



```
[hungs-mbp:nodeJS codes hungphan$ node addNumbers.js
Let's input at least 2 integers
[hungs-mbp:nodeJS codes hungphan$ node addNumbers.js 10 11 12
The sum of command-line numbers is :  33
hungs-mbp:nodeJS codes hungphan$ 
```

*Sample code for implementing **addNumber.js***



```
var sum = 0;
var i;
// console.log(process.argv.length+"");
if(process.argv.length<4){
  console.log("Let's input at least 2 integers");
} else{
  for (i = 2; i < process.argv.length; i++) {
    //sum = sum + process.argv[i]; // this will concatenate strings
    sum = sum + Number(process.argv[i]);
  }
  console.log("The sum of command-line numbers is : ", sum)
}
```

<u>Step 3</u>: Play with arrays

- Open playWithArrays.js.

- Editing the playWithArrays.js:

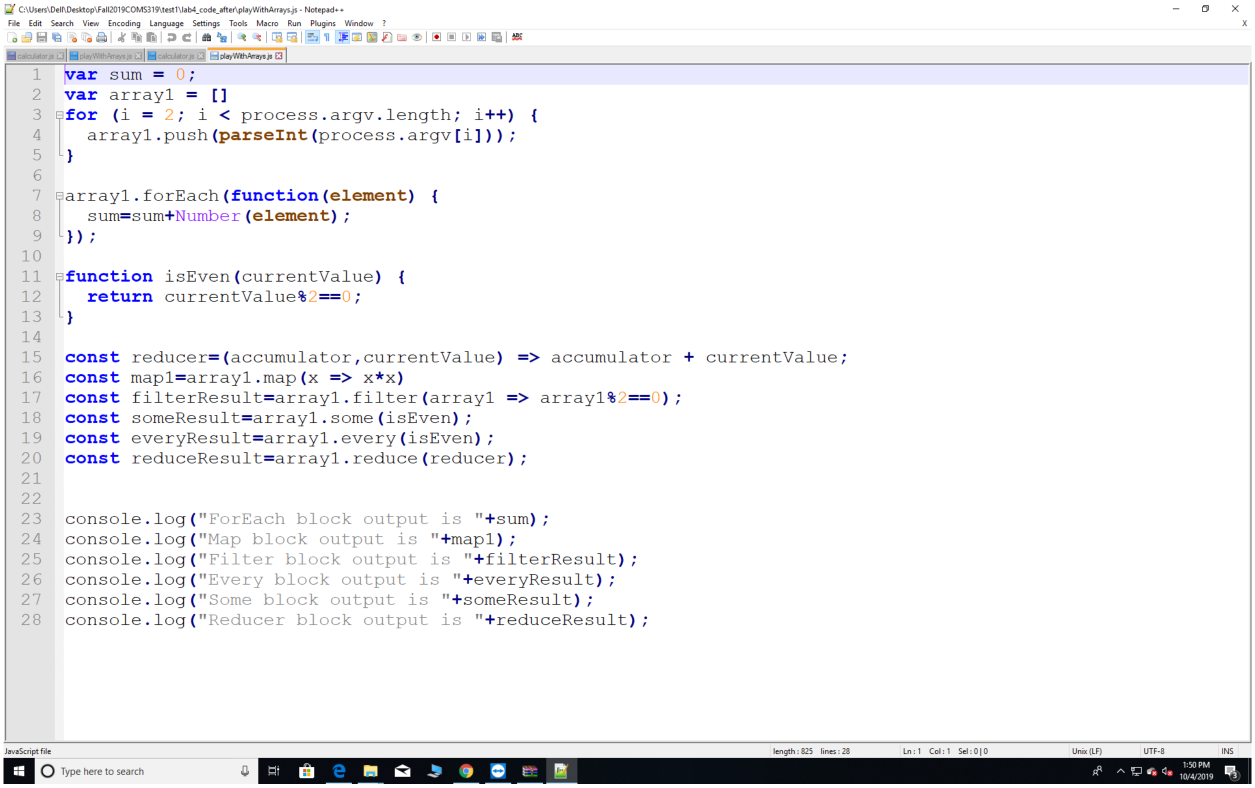
Write code (in Javascript) to do these functions:

A) Take a series of numbers as a command-line argument.

B) Use the following array functions:

❖ forEach (print the sum of numbers)
❖ map (return an array with each number squared)
❖ filter (return an array with only even numbers)
❖ every (return true if all the numbers are even)
❖ some (return true if some numbers are even)
❖ reduce (return the sum of the numbers by **reducer**)

Expected code:


C:\Users\Dell\Desktop\Fall2019COMS319\test1\lab4_code_after\playWithArrays.js - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
calculator.js
playWithArrays.js
calculator.js
playWithArrays.js
JavaScript file
length : 825 lines : 28
Ln : 1 Col : 1 Sel : 0 | 0
Unix (LF)
UTF-8
INS
Type here to search
1:50 PM
10/4/2019

```javascript
var sum = 0;
var array1 = []
for (i = 2; i < process.argv.length; i++) {
  array1.push(parseInt(process.argv[i]));
}

array1.forEach(function(element) {
  sum=sum+Number(element);
});

function isEven(currentValue) {
  return currentValue%2==0;
}

const reducer=(accumulator,currentValue) => accumulator + currentValue;
const map1=array1.map(x => x*x)
const filterResult=array1.filter(array1 => array1%2==0);
const someResult=array1.some(isEven);
const everyResult=array1.every(isEven);
const reduceResult=array1.reduce(reducer);


console.log("ForEach block output is "+sum);
console.log("Map block output is "+map1);
console.log("Filter block output is "+filterResult);
console.log("Every block output is "+everyResult);
console.log("Some block output is "+someResult);
console.log("Reducer block output is "+reduceResult);
```

Task 2:Node.JS Callbacks

Learning Objectives:

- **Learn how to create our own modules --- that accept callbacks**

Definition of callback function:

- https://www.javascripttutorial.net/javascript-callback/#:~:text=In%20JavaScript%2C%20a%20callback%20is,method%20of%20the%20Array%20object.

Step 1: Create a reusable module (library)

- Go into the **callback1** folder
- Don’t need to change the code in myLib.js and call1.js, Try to understand the code.
- On the **library side (myLib.js)** you will need:

  module.exports = { <<properties you want to export>>};

  You can export as many properties as you want.

  These can be an object, function, array, string etc.

  Example: module.exports = {'name': "Tom", … }; // in myLib.js

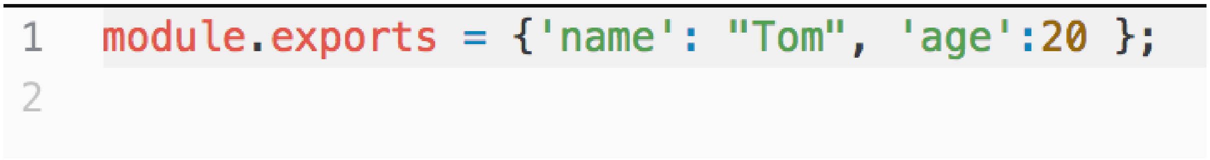
- On the consumer of the library side, you will need

  let var_name = require('./filename');

  var_name.<<property>> will give you access to the desired property.

Expected code:

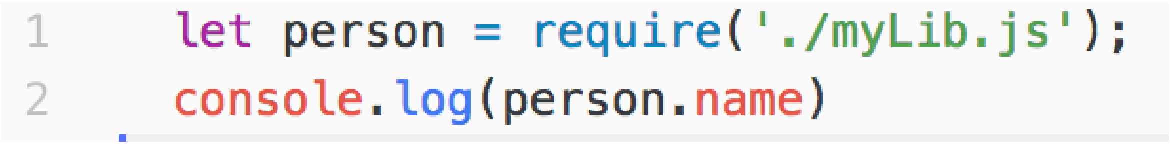
- On myLib.js



```
module.exports = {'name': "Tom", 'age':20 };

```

- On call1.js

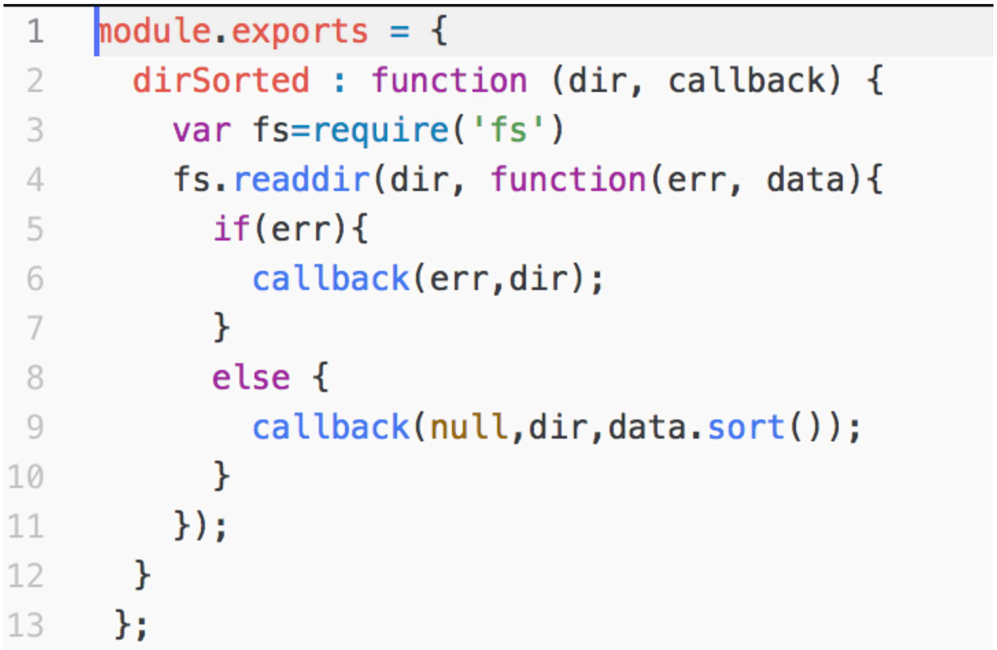


```
let person = require('./myLib.js');
console.log(person.name)
```

The output of call1.js should show the person's name. Using the sample code above, what’s the output?

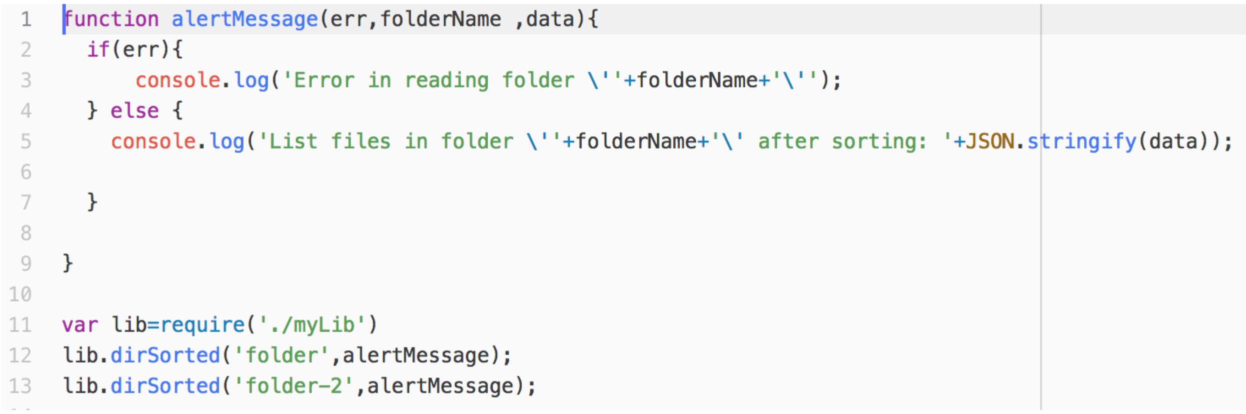
Step 2: Accept a callback

- Go into the **callback2** folder, don’t **create/remove** any sub-folders in this folder.
- A callback is just a function. For your module to accept a callback, all it has to do is to accept a function as a parameter. Inside your library code, you will call that function.
- Read through the **myLib.js** and **call2.js** to get the ideas of call back.
- Now, editing the myLib.js to add these lines of code. They will make your program accept a callback function.



```
module.exports = {
  dirSorted : function (dir, callback) {
    var fs=require('fs')
    fs.readdir(dir, function(err, data){
      if(err){
        callback(err,dir);
      }
      else {
        callback(null,dir,data.sort());
      }
    });
  }
};
```

- Editing the **call2.js** to update the alertMessage function as follows;



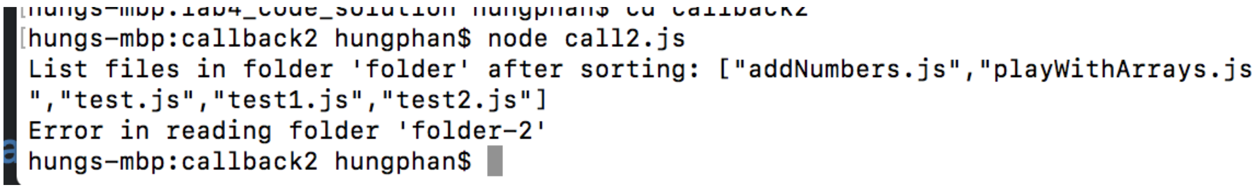
```
function alertMessage(err,folderName ,data){
  if(err){
      console.log('Error in reading folder \''+folderName+'\'');
  } else {
    console.log('List files in folder \''+folderName+'\' after sorting: '+JSON.stringify(data));

  }

}

var lib=require('./myLib')
lib.dirSorted('folder',alertMessage);
lib.dirSorted('folder-2',alertMessage);
```

- Run **call2.js.** You should see the output as follows:



```
hungs-mbp:lab4_code_solution hungphan$ cd callback2
hungs-mbp:callback2 hungphan$ node call2.js
List files in folder 'folder' after sorting: ["addNumbers.js","playWithArrays.js
","test.js","test1.js","test2.js"]
Error in reading folder 'folder-2'
hungs-mbp:callback2 hungphan$ 
```

- Why the program return list of files in ‘folder’ but it shows an error message when reading ‘folder-2’?

Task 3:Node.js programming

Objectives:

Learn to use node.js programming.

Warm-up:

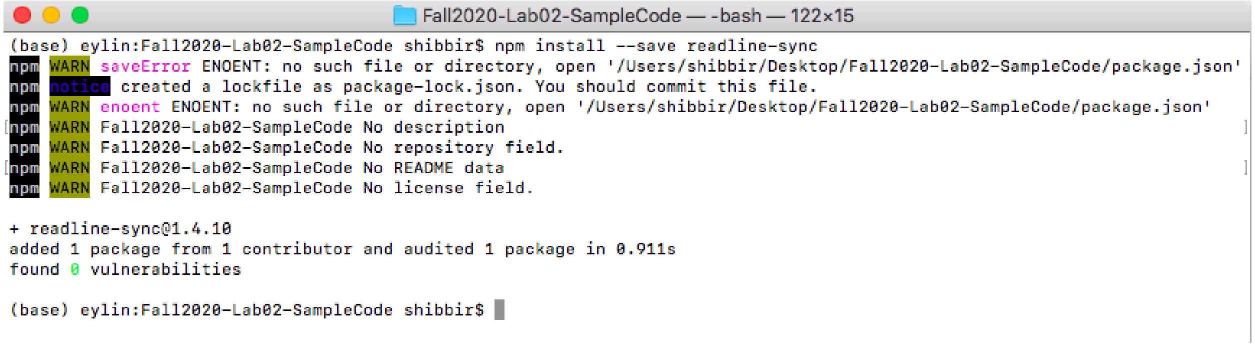
- Go to the parent folder (get out of callback2 subfolder by **“cd ..”**)
- Play with the given example -*example.js*. Open using a text editor of your choice and modify to learn how the different instructions work.

Task:

You need to create a simple binary calculator program by modifying the given *example.js*.You do not need to submit this file into Canvas just answer the Quiz question in Canvas. You can start with the given warm-up example “*example.js*” and follow lab activity 3.

- You need to install **'readline-sync'** like here (https://stackoverflow.com/questions/52675705/error-cannot-find-module-readline-sync-node-js). The two command line for the install could be

```
npm init
npm install --save readline-sync
```

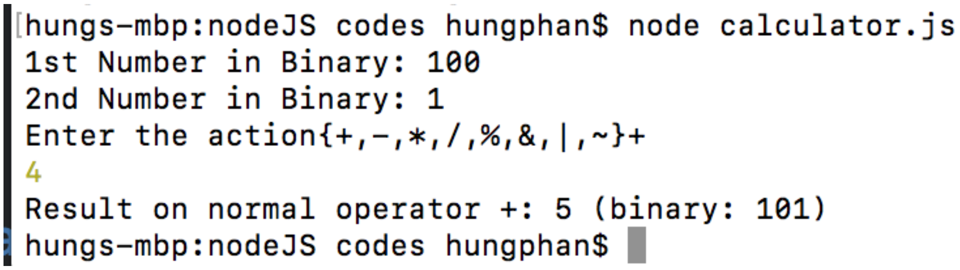


```
(base) eylin:Fall2020-Lab02-SampleCode shibbir$ npm install --save readline-sync
npm WARN saveError ENOENT: no such file or directory, open '/Users/shibbir/Desktop/Fall2020-Lab02-SampleCode/package.json'
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN enoent ENOENT: no such file or directory, open '/Users/shibbir/Desktop/Fall2020-Lab02-SampleCode/package.json'
npm WARN Fall2020-Lab02-SampleCode No description
npm WARN Fall2020-Lab02-SampleCode No repository field.
npm WARN Fall2020-Lab02-SampleCode No README data
npm WARN Fall2020-Lab02-SampleCode No license field.

+ readline-sync@1.4.10
added 1 package from 1 contributor and audited 1 package in 0.911s
found 0 vulnerabilities

(base) eylin:Fall2020-Lab02-SampleCode shibbir$
```

- Open the file **calculator.js**. From now you need to modify and add the code for handling input by users.
- **Input** and **Output:** Users can input two **binary integers** and **operator** and get the output as follows:

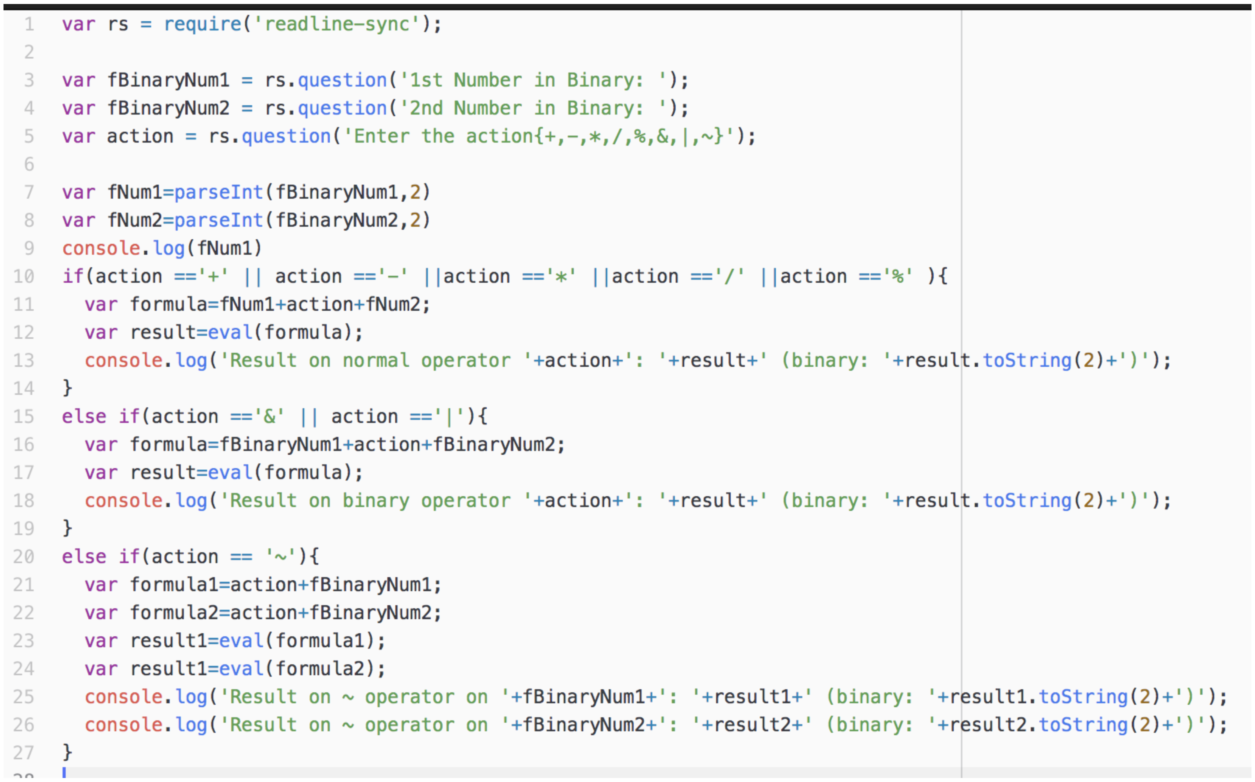


```
hungs-mbp:nodeJS codes hungphan$ node calculator.js
1st Number in Binary: 100
2nd Number in Binary: 1
Enter the action{+,-,*,/,%,&,|,~}+
4
Result on normal operator +: 5 (binary: 101)
hungs-mbp:nodeJS codes hungphan$
```

For a binary calculator, it should have these functions:

1. Note that for some operations on the binary calculator, it may be more convenient to convert the binary numbers to integers and then do the operation. (It is a suggestion, you can implement your own logic).
2. You can assume that only positive binary numbers are represented and used. For example, positive 9 is represented as 1001.
3. Binary operator "+" represents addition operation.
4. Binary operator "*" represents multiplication.
5. Binary operator "/" represents division.
6. Binary operator "%" represents mod or remainder (i.e. divide the first value by the second, what is remaining, only works on positive numbers).
7. Binary operator "&" represents AND (only works on positive numbers) e.g. (101 & 1011 gives 0001)
8. Binary operator "|" represents OR (only works on positive numbers) e.g: 101 | 1010 gives 1111.

9. Unary operator “~” represents not (i.e. invert each bit of the binary value).
The sample code for this function is shown below



```
var rs = require('readline-sync');

var fBinaryNum1 = rs.question('1st Number in Binary: ');
var fBinaryNum2 = rs.question('2nd Number in Binary: ');
var action = rs.question('Enter the action{+,-,*,/,%,&,|,~}');

var fNum1=parseInt(fBinaryNum1,2)
var fNum2=parseInt(fBinaryNum2,2)
console.log(fNum1)
if(action =='+' || action =='-' ||action =='*' ||action =='/' ||action =='%' ){
  var formula=fNum1+action+fNum2;
  var result=eval(formula);
  console.log('Result on normal operator '+action+': '+result+' (binary: '+result.toString(2)+')');
}
else if(action =='&' || action =='|'){
  var formula=fBinaryNum1+action+fBinaryNum2;
  var result=eval(formula);
  console.log('Result on binary operator '+action+': '+result+' (binary: '+result.toString(2)+')');
}
else if(action == '~'){
  var formula1=action+fBinaryNum1;
  var formula2=action+fBinaryNum2;
  var result1=eval(formula1);
  var result1=eval(formula2);
  console.log('Result on ~ operator on '+fBinaryNum1+': '+result1+' (binary: '+result1.toString(2)+')');
  console.log('Result on ~ operator on '+fBinaryNum2+': '+result2+' (binary: '+result2.toString(2)+')');
}
```

Please only submit the <u>Quiz questions</u> on Canvas. No need to submit other questions mentioned in this pdf.