# JavaScript NodeJS NodeJS MongoDB Express Beginner to Intermediate JavaScript

# #SECTION 1 Introduction to Installing and Setup of Node

This section is a fast paced quick intro to NodeJS, and how you start using NodeJS.   We discuss what NodeJS is and that it is not a programming language like JavaScript is but an interpreter and Environment for JavaScript.
Node is a lightweight framework developed on Chrome's V8 engine that can be used for largeScale application development.   It's scalable because the server can respond in a non-blocking way.

Process object which is a global that provides information, and control over, the current Node.js process. Because its global it is always available to Node.js applications without using require()

**npm** registry contains packages, many of which are also Node modules, or contain Node module

1. Demo how to setup your development environment setting up Editor
2. Visual Studio Code as an Editor with ready terminal
3. Creating your first Node File and running the code
4. How Node Works and examples of simple node syntax
5. Where you can find out more about Node with more examples.
6. How to debug and output content into the terminal various command line and console outputs
7. NodeJS programming language environment **REPL** Read Evaluate Print Loop
8. Examples of Node
9. Process Object - with examples
10. What Node Packages and NPM

## 1 What is Node Setup Dev Environment

- Setup and introduction  to getting started with Node Node website https://nodejs.org/en/download/
- Editor Visual Studio Code https://code.visualstudio.com/
- NPM - setup https://www.npmjs.com/
- Terminal Command+Space or Windows https://www.microsoft.com/en-ca/p/windows-terminal-preview/
- Check Node Version and NPM version `node -v` and `npm -v`

## 2 Visual Studio Code Terminal Ready

- How to use Visual Studio Code
- Open and access Terminal in Visual Studio code
- Editor VIsual Studio Code https://code.visualstudio.com/
- Setup JS file for Coding

## 3 Create a Node file and Run it

- Create a first js file
- How to run a node js file
- Terminal setup
- Console.log in browser and in terminal
- Use of console.log to output content into console
- ls or dir to list files within the current directory using the terminal
- node app.js running a node file.

## 4 What is Node and how it Works

- Node code and how it runs on your machine
- V8 engine
- Asynchronous and non Blocking
- Benefits of Node
- Why node is a good choice
- Node applications.

## 5 NodeJS resources

- V8 Chrome and Node
- History of Node
- Browser document object DOM
- Selecting page element
- Window Object in browser
- No window object in Node
- Node global object
- Non Blocking Asynchronous
- Speed difference

## 6 Working with Node JS

- CTRL+C to exit from node application
- Run Node Application node appNAME.js

- Create console.log message
- Create Loop - stop loop with CTRL+C
- Process object within node - process.exit()
- for loop and using exit to break loop

```
console.log(process);
for(let x=0;x<1;x++){
   console.log(x);
   if(x==30000){
       process.exit();
   }
}
```

## 7 Node JS REPL

- Type node in the terminal
- To get help .help in REPL mode
- REPL also known as Read Evaluate Print Loop is a programming language environment
- console. Press tab to get a list of available functions.
- global. +tab to get list of available functions within global object
- console option in REPL
- Global Object in Node
- Math in the terminal

## 8 Common Node Examples

- Filename and  dirname
- Locate file location to use within node application
- Use of node documentation for examples and code samples
- Node global object
- __dirname get directory of current file
- __filename get filename of current file

## 9 Node Process Object

- The process object is a global that provides information about, and control over, the current Node.js process
- Process argv - output and  use process values within node application

- process.pid
- Logging process argv by index using console.log
- template literal for strings ${} backticks `

```
console.log(__dirname);
console.log(__filename);

console.log(process.argv);
console.log(process.argv[1]);

const first = process.argv[2];
const second = process.argv[3];

let message = `Hi, welcome ${first} ${second}`;
console.log(message);
```

## 10. Use Node Packages NodeMon

- nodemon is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected.
- Search at https://www.npmjs.com/
- How to install packages
- npm install -g nodemon
- nodemon app.js
- Use of sudo prefix for admin on Mac
- -g global flag

## 11. Command Line and Console

- Type code directly within REPL mode
- Use of Console.log to debug and output to terminal console
- Console API options https://nodejs.org/dist/latest/docs/api/console.html
- Log, warn, info, table and count

- console.log %s use of string placeholder
- console.count()
- console.clear()
- console.table()
- console.warn()
- console.info()

## Assignment Section #1

1. Open the REPL mode and list the process object.  Return the process.ppid and the pid into the terminal.
2. Run the node script with 2 arguments: your first name and lastname.  Create a node application and output process.argv[2] process.argv[3] into the terminal like the example below.

```
const a = 4;
const b = 6;
console.log(a,b,a+b);
console.log('%s',a);
console.log('Hi Laurence');
console.count('test'+a);
console.count('test'+a);
console.count('test'+a);
console.clear();
console.log('Hi Laurence');
console.warn('warn');
console.info('info');
console.table([{a:1,b:2},{a:5,b:10}]);
```

# #SECTION 2 JavaScript Fundamentals quick review of JavaScript Coding

Explore the fundamentals of JavaScript and writing JavaScript code within the Node.   Loaded with examples about the basics of coding JavaScript.   Back to the basics of JavaScript code . Because node uses JavaScript code this section will provide a refresher and quick overview of JavaScript.   Designed to cover the core JavaScript required for the upcoming sections and writing Node.

1. Variables and the Data Types.  How to declare variables and examples of different Data Types.
2. Adding Comments to code
3. How to create Objects and Array
4. What JavaScript Functions are and how they work
5. Advanced Functions Immediately invoked Function Expression IIFE, Arrow format for functions, JavaScript Closure example
6. Different JavaScript operators and examples
7. Use of conditions for logic, turthy and falsey.  How to use the  Ternary Operator.
8. Examples of logic switch statement, and condition statement
9. JavaScript loop Structures For Loop, While Loop, DO-While Loop with code examples
10. Use of Objects and objects with methods.
11. JavaScript Arrays, adding, updating and removing items from an array. lots of commonly used useful array methods for array manipulation.
12. Array and Object Values, iterating through the contents of each.
13. Use of Template literals to set string values
14. Math and Data Method examples, generating random numbers and dates in JavaScript
15. Global object in Node with the use of timeout
16. Event Queue and Asynchronous operations with JavaScript
17. Examples of JavaScript Node Timer module
18. How to Construct promises and Resolve and Reject Promises
19. Debugging - use of Try, Catch, Finally which are used to handle exceptions in the JavaScript
20. JavaScript Async and Await examples and how the order of output on promise completion.

## 1. Variables and Data Types

- How to declare a variable
- Assign New Values to a Variable
- What Variables are and how to use them

- Comments and multiline comments
- Quotes and Strings
- const, let, var
- typeof operator
- Data Types
- declare variables

## 2. Variables and Data Types

- Strings, numbers, booleans
- Variables let const - var - assign values
- Rules for Declaring Variables
- Data type typeof ""
- ReAssign new values = assign values

## 3. Arrays Objects Data Types

- Strings, numbers, booleans, arrays, objects
- Intro to arrays and objects
- Dynamic type conversion
- Blocks of code with {} brackets
- Scope within blocks
- Create an object
- Create and array
- Add string with number

```
let a = 5;
let b = "String\"'s";
let c = 'String"s"';

console.log(a,b,c);

let myNewValue = "test";//no spaces
let one1 = "one"; //can't start with number
let $_val = "test"; //can use $ or _
let boo = true; //boolean

console.log(typeof a);
console.log(typeof b);
console.log(typeof boo);


const newVariable = "hello world";
console.log(newVariable);
//newVariable = "world";
//console.log(newVariable);
/*
{
   let newVariable = "hello";
   console.log(newVariable);
   newVariable = "world 2";
   console.log(newVariable);
}
*/
{
   const newVariable = "world";
}
console.log(newVariable);

const arr = ['hello','world',123,true];
console.log(arr);
const myobj = {"first":"Laurence",last:"svekis",age:'none'};
```

```
console.log(myobj);

let myStr = "hello";
console.log(typeof myStr);
myStr = 500;
myStr = 500 + "Words";
console.log(typeof myStr);
myStr = "5" + 5;
console.log(myStr);
console.log(typeof myStr);
```

## 4. Introduction to Javascript Functions

- Functions - invoke blocks of code
- Function scope
- Function arguments
- Invoking functions with ()
- Passing parameters to functions
- Function Declarations vs. Function Expressions
- Return response callback on functions

```
const myFun4 = function(a,b,c){
   const val1 = a * b -c;
   return val1;
}

const a4 = myFun4(4,6,2);
const a5 = myFun4(24,33,4);
const a6 = myFun4(72,25,6);
console.log(a4,a5,a6);
```

```
myFun2('Hello','Laurence',100);
myFun2('Hi','John',500);

const val1 = myFun2('Welcome','Jane',1000);
console.log(val1);

const a1 = myFun3(4,6,2);
const a2 = myFun3(24,33,4);
const a3 = myFun3(72,25,6);
console.log(a1,a2,a3);

function myFun1() {
   console.log('Hello');
   console.log('World');
};
function myFun2(message,name,val) {
   console.log('1.'+message,name,val);
   const val1 = name + ' ' + message;
   console.log('2.'+val1);
   return val1;
};
function myFun3(a,b,c){
   const val1 = a * b /c;
   console.log('===='+val1);
   return val1;
}
```

## 5-1. Javascript Functions Advanced Features

- Function expressions
- Function declarations
- Immediately invoked Function Expression IIFE example
- (function()){})() IIFE example

## 5-2. Modern Function Examples

- Arrow format for function ES6+  (e)=>{}
- Function recursion
- Function invoking within itself
- Use of return
- Loop of function

```
const fun1 = function(a,b,c){
   const val = a * b * c;
   return val;
} //expression

const a =5;
const b = 6;
const c = 7;

(function(a,b,c) {
   const val = a * b * c;
   console.log(val);
})(a,b,c); //IIFE

const fun3 = (function(a,b,c) {
   const val = a * b * c;
   return val;
})(a,b,c); //IIFE
console.log(fun3);

const fun4 = (a,b,c)=>{
   const val = a * b * c;
   return val;
} //expression

console.log(fun4(7,7,7));

(()=>{
   console.log('test');
})() //IFEE arrow

const val1 = fun1(5,6,7);
const val2 = fun2(5,6,7);

console.log(val1);
console.log(val2);

function fun2(a,b,c){
   const val = a * b * c;
   return val;
}

let counter = 100;
function loop(val){
   console.log(val);
   if(val<88){
       return
   }
   const temp = val -1;
   loop(temp);
}

function loop2(val){
   console.log(val);
   if(val > 0 ){
       val=val-1;
       return loop2(val);
   }
   loop(counter);
   return 'end';
}

const looper =loop2(25);
console.log(looper);
```

## 6. Javascript Closure Example

- How to create a JavaScript closure
- Anonymous Functions
- JavaScript Function Arrow format
- Use of JavaScript Closures
- Closure gives you access to an outer function’s scope from an inner function
- Closures are created every time a function is created
- Associate data with a function that operates the data

```
function maker(val1){
   return (val2)=>{
       console.log(val1,val2);
       return val1 + val2;
   }
}
const fun1 = maker(5);
const fun2 = maker(25);

console.log(fun1(4));
console.log(fun1(10));
console.log(fun2(75));
```

## 7. Javascript Operators

- Assignment operators
- Short format for adding subtracting
- Remainder (%) returns the remainder left over when one operand is divided by a second operand
- Modulus %  Operator: var1 % var2
- Examples of various assignment operators
- Adding/Subtracting  with shorthand methods ++ += -- -=
- Multiplication * division /

## 8. Javascript More Operators Examples

- Joining strings together
- String concatenation
- Comparison operators with data type absolute and regular. Comparison operator compares its operands and returns a Boolean value based on whether the comparison is true.
- Logical operators and and or and logic not.  Logical AND && Logical OR ||
- Equality operator Equality operator == Inequality operator != Identity operator === Nonidentity operator !==

```
/*
let val = 100;
val = val + 1000;
val += 1;
val++;
val--;
val -= 500;
val *= 2;
val /= 3;
console.log(401/2);
console.log(401%2);
console.log(400/2);
console.log(400%2);
console.log(3**4);
let val2 = 50;
val2 %= 10;
console.log(val2);
console.log(val*10);
//console.log(val);
const first = "hello";
const second = "world";
//console.log(first + ' ' + second);
*/
let a = "5";
let b = 5;
//comparison operators
//const val1 = (a == b);
```

```
let val = (a === b);
val = (a >= b);
val = (a !== b);
//console.log(val);
// logical operators
val = (true)&&(false); //both true to be true
val = (false)||(false); //one has to be true to be true
val = !(false);
console.log(val);
```

## 9. Conditions Truthy and Falsey

- Truthy and Falsey
- The if statement executes a statement if a specified condition is truthy. If the condition is falsy, another statement can be executed.
- Condition statements if else else if conditions
- Check condition within function return results

## 10. Ternary Operator

- condition ? exprIfTrue : exprIfFalse
- Ternary operator
- JavaScript operator that takes three operands
- Executes express for True or if condition is false.
- Used commonly as shortcut for if statement
- 3 operands: condition followed by a question mark (?), expression to execute if the condition is truthy followed by a colon (:), lastly the expression to execute if the condition is falsy
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_Operator

```
/*
const num1 = 12398909;
const rem = num1 % 2;
console.log(rem);
```

```
if(rem){
   console.log(num1 + ' : was odd');
}else{
   console.log(num1 + ' : was even');
}

if(1){
   console.log('was true');
}else{
   console.log('was false');
}

function checkOutput(val){
   let message;
   let checkNum = 10;
   if(val > checkNum){
       message = val + ' was larger than ' + checkNum;
   }else if(val == checkNum ){
       message = 'Both are equal to ' + val;
   }
   else {
       message = val + ' was less than ' + checkNum;
   }
   return message
}

let counter = 3;
let temp = (counter > 10) ? 'True' : 'False not greater';
console.log(temp);
console.log(checkOutput(counter));
counter++;
counter+=8;
console.log(checkOutput(counter));
counter-=2;
console.log(checkOutput(counter));
*/

const age =  15;
let message;
let message1 = (age >= 21) ? 'Allow in' : 'Not allowed';
//Ternary Operator
console.log(message1);

if(age >= 21 ){
   message = 'Allow in';
}else{
   message = 'Not allowed';
}
console.log(message);
```

## 11. JavaScript Switch Statement

- Create a switch statement
- Test results in case
- Check for a match result run code
- Use of break to separate the case response
- Use of Default in case not match was found

- switch statement evaluates an expression
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/switch

```
const val = 'test';

switch (val) {
   case 'yes' :
       console.log('it was yes');
       console.log('test');
       break;
   case 'none' :
   case 'zero' :
       console.log('was not yes or no');
       break;
   case 'no' :
       console.log('it was no');
       break;
   default:
       console.log('none was found');
}
```

## 12. JavaScript Loop For Do While

- Loops run blocks of code over multiple iterations
- For loop
- Decreasing loop
- Do while loop vs While Loop what is the difference
- Quick and easy way to do something repeatedly
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration

```
let counter = 10;
```

```
for(let i=0;i<counter;i++){
   console.log('value of i '+i);
}

for(let x=10;x>0;x--){
   console.log('value of x '+x);
}
console.clear();
let y = 0;
while (y<10) {
   console.log('value of y '+y);
   y++;
}

do{
   console.log('DO WHILE value of y '+y);
   y++;
}
while (y<10);
```

## 13. JavaScript Objects

- Add values into JavaScript object
- Named pair values name:value
- Create object
- Update object data
- Copy object
- Bracket notation vs Dot Notation
- https://jsonlint.com/
- Dynamic Object property names

## 14. JavaScript Object Methods

- objectName.methodname = functionName;
- Object Functions
- Use of ‘this’ JavaScript keyword within an object
- Create an object with object nested inside
- Return values from Object
- Add a function within the object
- Return values from object using object values

```javascript
const myObj1 = {
   first : "Laurence",
   last : "Svekis",
   age : 40,
   a1 : 'test',
   a2 : 'test 2',
   a3 : 'test 3',
   "test 1" : "wow"
};
console.log(myObj1);

console.log(myObj1.first + ' ' + myObj1.last); //dot notation

console.log(myObj1['first'] + ' ' + myObj1['last']);
console.clear();
for(let x=1;x<4;x++){
   console.log('a'+x);
   console.log(myObj1['a'+x])
}

console.log(myObj1['test 1']);
console.clear();

const myObj2 = {
   "first": "Laurence",
   "last": "Svekis",
   "age": 40,
   "test 1": "wow"
}
myObj2.last = 'NewName';
myObj2.newOne = 'Test 1000';
console.log(myObj2.last);
console.log(myObj2);
console.clear();
const myObj3 = myObj2;
myObj3.last = "Changed Last Name";
console.log(myObj3);
console.log(myObj2);

console.clear();

const myObj4 = {
   val : 100,
   val1 : true,
   val2 : 'String',
   val3 : {
       first : 'Laurence',
       last : 'Svekis'
   },
   val4 : {
       first : 'Laurence',
       last : 'Smith'
   },
   fullName : function(val5){
       console.log(this);
       console.log(val5);
       return this.val4.first + ' ' + this.val4.last
   }}
//console.log(myObj4.val3.first + ' ' + myObj4.val3.last);

console.log(myObj4.fullName('test'));
```

## 15. JavaScript Array Introduction

- How to create a JavaScript Array
- Update Values of Array
- Use of index value to select from the array
- Array holds number strings objects booleans and other data types
- Typeof Array is object
- Select item from array with index value
- Single objects that contain multiple values stored in a list
- Arrays consist of square brackets and elements that are separated by commas

## 16. JavaScript Update Array

- Add to the end of the array push method
- Remove from the end of an array with response value pop method
- Add to the beginning of an array unshift  method
- Remove from the beginning of an array shift method
- Add/Remove from array starting at index value with splice
- pop() method removes the last element from an array and returns the value
- push() method adds a new element to the end of an array
- shift() method removes the first array element
- unshift() method adds a new element to the start of an array

## 17. JavaScript Delete and Add to an Array

- How to remove contents from an array
- Assigning values to array item using index
- Array splice method
- Update items and remove items from array with splice

```
console.clear();
const myArray = ['test',10,true,{one:'one',two:'two'}];
const myObj = {one:'one',two:'two'};
console.log(myArray);

console.log(myArray.length);
console.log(typeof myObj);
console.log(typeof myArray);

const myArr1 = myArray;

myArr1[0] = 'Tested Okay';
console.log(myArray);
console.log(myArr1[0]);
console.log(myArr1[1]);
console.log(myArr1[2]);
console.log(myArr1[3]);
console.log(myArray[3]['two']);
console.clear();
for(let x=0;x<5;x++){
   //myArr1[myArray.length] = 'Counter ' + x;
   const temp = 'Counter ' + x;
   myArr1.push(temp); //add to end
}
for(let x=0;x<3;x++){
   const temp = myArr1.pop(); //remove from end with response.
   console.log('REMOVED '+temp);
}

for(let x=0;x<3;x++){
   const temp = myArr1.shift();
   console.log('REMOVED from Start '+temp);
}

for(let x=3;x>0;x--){
   const temp = 'Add ' + x;
   myArr1.unshift(temp);

}//delete myArr1[4];
//myArr1[100] = "100";

console.log(myArr1);
//let tempArr = [];
for(let x=5;x>0;x--){
  const temp = 'Value ' + x;
  myArr1.splice(3,0,temp);
  //tempArr.push(temp);
}
//
```

```
//console.log(tempArr);
//myArr1.splice(3,2,tempArr)

console.log(myArr1);
```

## 18. Common JavaScript Array Methods

- Use of Array Methods
- Several of the built-in array methods
- JSON parse and JSON stringify to convert to object and back to string
- Array toString methods toString()
- Array join()
- Array concat()
- Array methods join to create a string with separator
- Concat method to merge 2 arrays together or duplicate contents of one array

## 19. More JavaScript Array Methods

- includes() return boolean
- indexOf() get index value of item in array
- Check if object is an array isArray()
- Useful methods for array

## 20. Array Method Update Array

- Copy and create new array
- Update array order with sort() and reverse() methods
- Get index value of item get lastIndexOf and IndexOf item in the array
- Check to see if item is within an array with includes and indexOf methods
- Array examples

## 21. JavaScript Empty Array Map Method

- How to remove contents from an array
- Use of map to iterate array contents and create a new array
- Use of Array Method Map Method
- map() method creates a new array populated with the results of original array
- Array and Loops to create array data
- Assigning values to array item using index

```
console.clear();
const myArr1 = ['one','two',55,true,{one:"one",two:"two"}];

let val = myArr1.toString();
val = JSON.stringify(myArr1);
val = JSON.parse(val);
val[0] = "UPDATED";

console.log(val);
console.log(myArr1);

const myArr2 =
['one','z','two','a','three','two','three','two','three','two','
three'];
val = myArr2.join('{{||}}');

val = myArr1.concat(myArr2);

val = myArr2.includes('two');
val = myArr2.indexOf('two');
val = myArr2.lastIndexOf('two');
console.log(myArr2);
myArr2.sort();
myArr2.reverse();
console.log(myArr2);
console.clear();
//myArr2.length = 0;
console.log(myArr2);
```

```
const myArr3 = myArr2.map(val=>val);
myArr2[2] = 'HELLO';
console.log(myArr3);
console.log(myArr2);
const myArr4 = myArr2.map(val =>{
   val += ' TESTED';
   return val;
})
console.log(myArr4);
//val = Array.isArray(myArr2);
console.log(val);
```

## 22. Array Iteration and Object Values

- Loop through and get contents of an array
- For loop with Array contents
- array.forEach get array contents
- JavaScript Objects for in loop for objects
- Use of JavaScript Object.entries(myObj)
- Object to Array with Keys and Values from Object Object.keys(myObj)
- Other values from array forEach((val,index,arr)

```
const myArr = [1,2,3,4,'five','six'];
const myObj = {
   first : "Laurence",
   last : "Svekis",
   city : "Toronto"
}
```

```
console.log(myArr.length);

for(let i=0;i<myArr.length;i++){
   console.log(myArr[i]);
}

myArr.forEach((val)=>{
   console.log(val);
})

for ( key in myObj){
   console.log(key,myObj[key]);
}

console.log(Object.entries(myObj));

Object.entries(myObj).forEach(([val,key])=>{
   console.log(val,key);
})

console.log(Object.keys(myObj));
const myArr2 = (Object.values(myObj));
myArr2.forEach((val,index,arr)=>{
   console.log(val,index,arr);
})
```

## 23. JavaScript Template Literals

- Use of expressions within template literal

- Template literals (Template strings)
- Template literals are string literals allowing embedded expressions.
- You can use multi-line strings and string interpolation features with them.
- Use of multiple lines
- Output content  in string
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals

```
const a = `Hello`;
const b = `World`;
function add(x,y){
   return x + y;
}
const val = `
${a} ${b} ${5+5} ${add(10,50)}
single quotes '
double quotes "
New \n Line

`;
console.log(val)
```

## 24. JavaScript Math Method

- Use of Math random to generate random number
- Math floor ceil to round up or down from random  result
- Create a random number generator function
- Math.floor(x)
- Math.ceil(x)
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/floor
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/ceil

```
console.log(Math.random());
```

```
const ran1 = Math.random()*100;
console.log(Math.floor(ran1));
console.log(Math.ceil(ran1));

for(let i=0;i<1000;i++){
   console.log(ranNum(1,10));
}

function ranNum(min,max){
   return Math.floor(Math.random()*max)+min;
}
```

## 25. JavaScript Date Method

- JavaScript Date objects represent a single moment in time
- Create date object
- Get time from date object
- Set date object values
- Date objects contain a Number that represents milliseconds since 1 January 1970 UTC
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

```
const newD = new Date();
console.log(newD);
const oldD = new Date(2020, 0, 1,9,30,50);
const oldD1 = new Date(2020, 0);
console.log(oldD1);
const zeroDay = new Date(10000000000000); //Jan 1,1970
console.log(zeroDay);
const isoDate = new Date("1980-01-15");
console.log(isoDate);
```

```
zeroDay.setFullYear(2020);
console.log(zeroDay.getTime());
console.log(zeroDay.getFullYear());
console.log(zeroDay.getDay());
console.log(zeroDay);
```

## 26. JavaScript Asynchronous Callback Examples

- Global object in Node
- Setup of setTimeout()
- Run function after timer
- Dynamic values in functions
- Terminal output order
- Variable update effects
- JavaScript runs tasks on a single thread, called the main thread
- Asynchronous operations like promises are put into an event queue, this runs after the main thread has finished processing.
- Event queue only runs after the main thread so they do not block subsequent JavaScript code from running.

```
//console.log(global);
let a = 'hello';
let counter = 0;
function timer(val){
   message(`timer ran ${val} ${a}`);
}

function message(val){
   counter++;
   console.log(`${counter}. - ${val}`);
}
```

```
for(let i=5;i>0;i--){
   setTimeout(()=>{message(`i=${i}`)},i*1000,`${counter} Loop :
${i*1000}`);
}

console.log('new val');

a = 'World';
```

## 27. JavaScript Node Timers Global

- The timer module exposes a global API for scheduling functions to be called at some future period of time.
- Timer functions use internal implementation that is built around the Node.js and function similar to what you might see ina browser.
- Global object in Node setImmediate()
- Set interval and clearInterval
- JavaScript Node Set Timeout clear timeout examples
- Global timers with Node

```
//console.log(global);
let counter = 0;
console.log('first');

setImmediate((val)=>{
   console.log(`immediate ${val}`)
},'Hello World');

console.log('last');

for(let i=0;i<10;i++){
   console.log(i);
}

function keepRunning(){
   counter++;
   if(counter>5){
       clearInterval(int1);
   }
   console.log(`running ${counter}`);
}

const int1 = setInterval(keepRunning, 1000);

const int2 = setTimeout(() => {
   console.log('5 second timeout');
}, 5000);

const int3 = setTimeout(() => {
   console.log('3 second timeout');
   clearTimeout(int2);
}, 3000);
```

## 28. JavaScript Promises

- Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.
- Construct promises
- Resolve and Reject Promises
- JavaScript Promises new Promise

```
const pro1 = new Promise((resolve,reject)=>{
   const temp = 3;
   setTimeout(() => {
       reject('TIMED out');
   }, 1000);
   setTimeout(() => {
       resolve('Hello');
   }, 500);
   /*
   if(temp > 5){
       resolve('success');
   }else{
       reject('reject');
   }
   */
})

pro1.then(
   (val)=>{
       console.log(`TRUE : ${val}`);
   },
   (err)=>{
       console.log(`Error!~!! : ${err}`)
   }
);
```

## 29. JavaScript Try and Catch Debugging

- The try...catch statement marks a block of statements to try and specifies a response should an exception be thrown.
- Code debugging
- Catching and throwing errors
- Try and Catch statements
- try lets you test a block of code for errors.
- catch ets you handle the error.
- throw lets you create custom errors.

```
function tester(val){
   try{
       if(val == undefined) throw "no value";
       if(isNaN(val)) throw "not a number";
       val = Number(val);
   }
   catch(err){
       console.log(`There was an error ${err} = ${val}`);
   }
   finally {
       console.log(`Done ${val}`);
   }
}
tester();
tester(100);
tester('Hello');
```

## 30. JavaScript Async and Await

- Adding async to add promises to functions
- Set Timeout within  function

- Add promise to resolve
- Return on completion
- Order of output to console
- Use of them once a promise is completed.

```
let counter = 0;

async function hello(mes){
   console.log(mes);
   counter++;
   let pro1 = new Promise((res,rej)=>{
   setTimeout(res("works"), 2000);
   });
   let result = await pro1;
   return `${mes} ${counter} ${result}`;
}

function output(mes){
   counter++;
   console.log(`${mes} ${counter}`)
}

hello('Hello World').then((val)=>{
   console.log(val);
})

for(let i=0;i<5;i++){
   output(`Loop ${i}`);
}
```

## Section 2 Content Review

#1 What are JavaScript Data Types?
JavaScript Data types:

- Number
- String
- Boolean
- Object
- Undefined

#2 What is === operator?

- Both operands must have the same value without conversion of data type
- strict equality operator

#3 What are 3 loop structures in JavaScript.

- for loop
- while loop
- Do-While Loops

#4 What is the result of this statement 5 * 2 + "5" + 10

- "10510"

#5 What would you see as the value when a variable used in the code doesn't exist?

- undefined

#6 List 2 ways to return an object value.  const myObj = {name:"Laurence"}

- myObj.name
- myObj["name"]

#7 List out 4 methods to add/remove to start of an array, and add/remove to the end of the array.

- pop() method removes the last element from an array and returns the value
- push() method adds a new element to the end of an array
- shift() method removes the first array element
- unshift() method adds a new element to the start of an array

#8 How would you get each key next to each value from this object in the console?

```
const myObj = {
   first : "Laurence",
   last : "Svekis",
   city : "Toronto"
}
```

- `for ( key in myObj){`
- `   console.log(key,myObj[key]);`

- `}`

#9 Create a function to return a random number with parameters of min and max as arguments in the function.

- `function ranNum(min,max){`
- `   return Math.floor(Math.random()*max)+min;`
- `}`

#10 What is a promise in JavaScript
Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

#11 Which keywords are used to handle exceptions in JavaScript?
Try, Catch, Finally

#12 Which loop structure will fun the block at least once?
do…while loop, block of code gets executed once before the condition is checked.

## Assignment Section #2

1. *Create an array that can hold a list of your friends.*
2. *Use a JavaScript function to add a new friend as an object including first name and last name to the friends array.*
3. *Add more friends to your array so that you have several*
4. *Create a loop to output all the friends into the console listing their first and last name.*

# #SECTION 3 JavaScript Asynchronous programming

Asynchrony, in computer programming, refers to the occurrence of events independent of the main program flow and ways to deal with such events. How to add callbacks that run once a stack is complete.

Covering Input Output which accesses anything outside the application.  Once the application is started it will be loaded into the machine memory.

Coding Examples, and practice code outputting into the console when ready

Event Driven programming which makes Node.js fairly fast compared to other similar technologies. Once the application starts it initiates its variables, declares its functions and then waits for the event to occur.

The two types of API functions in Node.js, Asynchronous, non-blocking functions and Synchronous, blocking functions.

The differences between blocking and non-blocking calls in Node.js.

Blocking occurs when the execution of additional JavaScript in the Node.js process must wait until a non-JavaScript operation completes. The event loop is unable to continue running JavaScript while a blocking operation is occurring. All the IO methods in node.js provide asynchronous versions, which are non-blocking.

Node keeps an event loop, which monitors when a task gets completed, which would then fire the corresponding events. Node.js overcomes the problem of blocking of I/O operations by using the event loop instead of threads.

Recursive functions, Function that calls itself is called a recursive function - must have a condition to break the loop.

The EventEmitter module makes possible the communication and interaction between objects in Node. EventEmitter provides properties like on and emit. The on property is used to bind a function with the event and emit property is used to fire an event.

1. NodeJS works on the Google v8 environment utilizing JavaScript as its scripting language.
2. How a callback works, Callback added to the background will run once stack is complete
3. Use of setTimeout and how it works in the queue.
4. Callback example Create a function that returns the results from another function
5. Event Loop with an Example of a Recursive Function
6. The three states of promises
7. How to create promises and more promises examples
8. Node process object
9. Node Event EventEmitter
10. how to setup events and trigger emit of events

## 1. Callbacks Stack and Queue

- Asynchrony, in computer programming, refers to the occurrence of events independent of the main program flow and ways to deal with such events.
- Use of setTimeout to delay processing way for callback

- SetTimeout to zero still adds to the queue and will get run after the initial stack is completed
- Async and sync examples with array data to output order of contents to console.
- Adding to the queue vs stack
- Single  thread - process one at a time
- Callback added to the background will run once stack  is  complete
- SetTimeout is not part of V8 - its part of Node which allows us to use the runtime to sent requests to the stack and handle the response callback
- if stack is empty then goes to task queue and runs it.
- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Concepts

## 2. Stack and Queue Examples

- MDN documentation
- JavaScript Single Thread (main thread)
- SetTimeout example into the console moving from queue to stack
- Examples and practice code output to console when ready

```
const arr = [1,2,3,4,5];
console.clear();
function aFun(myArr,callback){
   myArr.forEach((el)=>{
       setTimeout(callback,1000,el);
   })
   myArr.forEach((el)=>{
       setTimeout(callback,0,`${el} -`);
   })
}

aFun(arr,(val)=>{
   //console.log(val);
   //console.log('async');
})
```

```
console.log('test');

arr.forEach((el)=>{
   console.log(el);
   console.log('sync');
})
console.clear();
console.log('one');
setTimeout(()=>{
   console.log('five');
},2000);
console.log('two');
setTimeout(()=>{
   console.log('four and a half');
},500);
console.log('three');
setTimeout(()=>{
   console.log('four');
},0);
console.log('pre four');
```

## 3 Callback example in JavaScript

- Create a function that returns the results from another function
- Sequence and order of results from function callbacks.
- Simple Callback model to illustrate Node Process model off single thread

```
function adder(a,b){
   console.log('adder');//3
   return a + b;
}

function doubler(a,b){
   console.log('doubler');//2
   return adder(a,b) * 2;
}

function output(a,b){
   console.log(a,b); //1
   const dub = doubler(a,b);
   console.log(dub);//4
}

output(2,5);
```

## 4. Calls Event Loops Job Function

- Event loop checks call stack
- Event loop give priority  to call stack - only once the stack is completed and nothing is there then it checks the queue.

## 5. Event Loop Example Recursive Function

- The event loop is what allows Node. js to perform non-blocking I/O operations
- Event loop can be asynchronous and have non-blocking I/O
- Event loop continuously checks the call stack to see if there are any functions that need to run.
- Recursive function max stack errors
- Function that calls itself is called a recursive function - must have a condition to break the loop.
- Condition to return and break recursive loop

```
//JS single Thread
function test1(){
   return test1(); //recrusive function
}

var test4 = function test5(i){
   console.log(i);
   if(i<5){
       test5(i+1);
   }
}
test4(0);
console.log('one');
//test1();

//Event loop checks call stack -

//console.clear();
const one  = ()=> console.log('one'); //4 //6
const two  = ()=> console.log('two'); //6 //5
const test2 = () =>{
   console.log('three'); //2
   one(); //3
   two(); //5
}
const test3 = () =>{
   console.log('three');  //2
   setTimeout(one,0); //3 starts timer
   two(); //4
}

test3(); //1

//event loop give priority  to call stack - only once the stack
is compeleted and nothing is there then it checks the queue.

console.clear();

(()=>{
   console.log('run right away');
})(); //iife

setTimeout(()=>{
   console.log('I ran after 1 second')
},1000);
```

## 6-1. Job Queue Event Loop Function Execution

- Intro to event loop and function execution examples
- setTimeout
- SetImmediate
- nextTick process methods
- Promise and Resolve and Reject.

**A Promise is in one of these states:**
pending: initial state, neither fulfilled nor rejected.
fulfilled: meaning that the operation was completed successfully.
rejected: meaning that the operation failed.
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

**The Node.js Event Loop, Timers, and process.nextTick()**
The event loop is what allows Node.js to perform non-blocking I/O operations — despite the fact that JavaScript is single-threaded
https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

## 6-2. Call Stack Output Order Examples

- Recursive function testing with Job Queue
- Immediately Invoked Function Expression IIFE Example
- Setting Timeout to output to terminal
- Output to terminal testing examples

```
console.clear();
const one  = ()=> console.log('one');
const two  = ()=> console.log('two');

const test1 = () =>{
  console.log('three');  //1

  new Promise((resolve,reject)=>{
      resolve('now')
  }).then(resolve => {
      console.log(resolve); //5
  })
  process.nextTick(()=>{
   console.log('WOW'); //4 //current event loop
  })
  setImmediate(()=>{
   console.log('Hello'); //next iteration event loop
  })
  setTimeout(one,10); //6
  two(); //2


}
test1(); //1
console.log('last'); //3
```

## 7 JavaScript Promises

- How to create a promise
- Resolve and Reject on Promise
- Use of Promise Callback
- Use of  then and catch to handle errors and responses for promise
- Promise object represents the eventual completion or failure of an asynchronous operation and its resulting value

```
let ready = false;

const checker = new Promise((resolve,reject)=>{
   console.log(ready);
   if(ready){
       const markReady = 'This is the result';
       resolve(markReady);
   }else{
       const markNot = 'Still working  on it.';
       reject(markNot);
   }
})

//console.log(checker);
ready = false;
//console.log(checker);
```

```
const check1 = () =>{
   checker.then(val=>{
       console.log(val);
   })
   .catch(err =>{
       console.log('error');
       console.log(err);
   })
}
check1();
```

## 8 Node Process Object Terminal Input Output

- The process object is a global that provides information about, and control over, the current Node.js process. As a global, it is always available to Node.js applications without using require().
- Use of readline to create terminal input and output
- Create readline questions and response
- Process on exit and before  exit
- Process env and process properties
- https://nodejs.org/dist/latest/docs/api/process.html

```
//console.log(process.env);

const rLine = require('readline').createInterface({
   input: process.stdin,
   output: process.stdout
 })
```

```
rLine.question(`What is your name?`, name => {
   console.log(name);
   //console.log(process);
   console.log(`Welcome ${name}`);
   rLine.close();
})

process.on('exit',(val)=>{
   console.log(`Process Exiting `);
   console.log(val);
})
process.on('beforeExit',(val)=>{
   console.log(`Before Exit`);
   console.log(val);
})
console.log(`Last Message`);
```

## 9 Node Event EventEmitter

- The EventEmitter is a module that facilitates communication and interaction between objects in Node
- Add new EventEmitter to code
- Require events
- Emit events
- Send Arguments to events
- Use of SetTimeout to setup events

## 10 Setup Node Events

- Any function passed as the setImmediate() argument is a callback that's executed in the next iteration of the event loop

- Loop of Events
- After stack event loop is complete run event with  setImmediate
- Example of Node event  setup
- Looping event code
- Events module require
- EventEmitter()
- on event trigger
- Emit on event trigger

```
const events = require('events');
const eEmitter = new events.EventEmitter();

let counter = 1;

const getParty = function (){
   counter++;
   console.log('Time to Party');
   const ran = Math.ceil(Math.random()*3);
   console.log(`party starts in ${ran}`);
   tester(ran);
}

eEmitter.on('music',getParty);

//eEmitter.emit('music');

function tester(val){
   if(counter<5){
   setTimeout((e)=>{
       eEmitter.emit('music');
   },val*1000);
}
}

eEmitter.on('event',(a,b)=>{
   console.log(a,b,this);
})

eEmitter.emit('event','a','b');


let m = 0;
eEmitter.on('event4',()=>{
   console.log(++m);
})

eEmitter.emit('event4');

eEmitter.on('event1',(a,b)=>{
   setImmediate(()=>{
       console.log(`1 Async ${a} ${b}`)
   })
})
eEmitter.on('event2',(a,b)=>{
   console.log(`2 Sync ${a} ${b}`)
})
eEmitter.once('event3',()=>{
   console.log('Once Run');
})
```

```
for(let x=0;x<4;x++){
   eEmitter.emit('event1',x,'b1');
   eEmitter.emit('event2',x,'b2');
   eEmitter.emit('event3');
}

eEmitter.emit('event4');
eEmitter.emit('event4');
eEmitter.emit('event4');
```

## Section #3 Review

#1 How does NodeJS work?
Node.js works on the Google v8 environment utilizing JavaScript as its scripting language.

#2 What is I/O mean?
Input Output which accesses anything outside the application. Once the application is started it will be loaded into the machine memory.

#3 What is the difference between blocking and non-blocking calls in Node.js
Blocking occurs when the execution of additional JavaScript in the Node.js process must wait until a non-JavaScript operation completes. The event loop is unable to continue running JavaScript while a blocking operation is occurring. All the IO methods in node.js provide asynchronous versions, which are non-blocking.

#4 What is an event loop in Node.js ?
Node keeps an event loop, which monitors when a task gets completed, which would then fire the corresponding events. The event loop is what allows Node.js to perform non-blocking I/O operations — despite the fact that JavaScript is single-threaded

#5 What does the event loop solve?
Node.js overcomes the problem of blocking of I/O operations by using the event loop instead of threads.

#6 What are the 3 states of Promises
pending: initial state, neither fulfilled nor rejected.
fulfilled: meaning that the operation was completed successfully.

rejected: meaning that the operation failed.

#7 What is the process object
The process object is a global that provides information about, and control over, the current Node.js process. As a global, it is always available to Node.js applications without using require().

#8 What is the eventEmitter
The EventEmitter module makes possible the communication and interaction between objects in Node.  EventEmitter provides properties like on and emit. The on property is used to bind a function with the event and emit property is used to fire an event.

## Assignment Section #3

*Setup a custom event and emit the event into the terminal*

1. *require the events module*
2. *setup a new events emitter*
3. *create a function the outputs a value into the terminal*
4. *Setup custom event emitter to listen for value.  Add the function which will run the block of code.  eventEmitter.on(value,function);*
5. *using emit select the value.  eventEmitter.emit(value);*

# #SECTION 4 Node Modules NPM

Each module in NodeJS has its own context and cannot interfere with other modules.  Modules are like libraries that you can include into your application.  Node.js has a set of built-in modules which you can use without any further installation like http, fs, path, and url.

You can create your own modules, and easily include them in your applications. Save them locally and use them within your node application as needed.  Perfect to separate functionality for your application. To make properties and methods available outside the module file use the keyword exports.

NPM hosts thousands of third party modules, free packages to download and use.  A package in Node.js contains all the files you need for a module.

Debugging - The try...catch statement marks a block of statements to try and specifies a response should an exception be thrown.

1. How you can create your own module locally
2. Commonly used builtin modules like http, fs, path and url
3. Using FS module to update file contents, delete files, create new files
4. Getting file information including file size with fs.stat()
5. Use of node package manager NPM explore some popular node packages.
6. Use of package.json - adding and removing packages
7. Node Package Manager Module lodash
8. Node Buffers how to copy and slice buffer contents
9. Debugging node use of Throw, try and catch

## 1. Node Modules

- Modules are the building block of any node application and are loaded by using require statement
- How to create your own Node. js module
- Writing simple modules
- Modules and Objects
- Local Module example

## 2. Node Modules Types

- Local Module simple setup create your own modules
- Core Built-in Module use of modules (http, url, path, fs, util) You need to import the core module first in order to use it in your application
- Third Party Modules (express, gulp, lodash, async, socket.io, mongoose) Third party modules can be install inside the project folder or globally
- Exports in modules
- Module object
- Use function from local module
- Use Data object from local module

```
const test1 =  function(){
   console.log('test500');
}
const person = {
   first : 'John',
   last : 'Smith'
}

//test1();
//const test1 = "hello";
//console.log(module.filename);

exports.test1 = test1;
exports.person = person;
```

```
const test1 = require('./test1');
const test2 = require('./test2');
//console.log(module.filename);

test1.test1();
console.log(test2);
```

```
const myName = `${test1.person.first} ${test1.person.last}`;
const myName2 = `${test2.person.first} ${test2.person.last}`;
console.log(myName);
console.log(myName2);
```

## 3. Local Node Modules

- Module exports
- Export single module
- Use of local module function
- Use of local module variables
- Require of module
- Same function name modules

```
const output1 = require('./test3');
const output2 = require('./test4');

console.log(output1);
console.log(output1('Hello'));

console.log(output2);
console.log(output2.output('TEST'));
console.log(output2.val1);
console.log(output2.val3);
console.log(output2.val2);
```

```
const output = function(message){
   console.log(`This is your message ${message}`);
   return message;
}

module.exports = output;
```

```
const val1 = 'Hello World';
const val2 = 'Test Private 2';
const val3 = 'New Message';

function output(message){
   return `This is your message ${message}. Private value
${val2}`;
}


module.exports.output = output;
module.exports.val1 = val1;
exports.val3 = val3;
```

## 4 Local Node Modules Exports Class

- Create a class
- Construct and object to use from module
- Export single module anonymous function
- Module exports are the instruction that tells Node. js which bits of code are exported

```
const person = require('./mods/test5');

const friend1 = new person('Laurence','Svekis');
const friend2 = new person('Jane','Doe');
const friend3 = new person('Jack','Jillman');

console.log(friend1.first);
console.log(friend1.last);
console.log(friend1.myName());
console.log(friend2.myName());
console.log(friend3.myName());
```

```
module.exports = function(first,last){
   this.first = first;
   this.last = last;
   this.newMod = 'Hello World';
   this.myName = function(){
       return `${this.newMod} ${this.first}  ${this.last} `;
   }
}
```

## 5. Core Modules http web Server Setup

- Use of http module
- Built in modules core node modules
- Create web server with node
- Route of files with conditions
- Change of ports
- Use of request module for json
- Output of json to web server from  JavaScript object

```
const http = require('http');

//console.log(http);

http.createServer((req,res)=>{
   res.writeHead(200, {'Content-Type':'text/html'});
   if(req.url == '/'){
       res.write('Main landing page');
   }
   else if(req.url == '/test'){
       res.write('Test page');
   }
   else{
       res.write('Page not FOUND');
   }
   res.end();
}).listen(8080);

console.log('Server Running...');
```

```
const http = require('http');

//console.log(http);
const myJson = require('./json1');
console.log(myJson);

http.createServer((req,res)=>{
    res.writeHead(200, {'Content-Type':'application/json'});
    res.write(JSON.stringify(myJson));
    res.end();
}).listen(8080);

console.log('Server Running...');
```

```
exports.output =  {
   first:"Laurence",
   last:"Svekis"
}
```

## 6. Make a Folder File System

- fs module provides functionality to access and interact with the file system
- file system on your computer
- mkdirSync() create folder
- Check if the folder exists.
- Get folder files names in array
- Folder directory contents
- const fs = require('fs')

```
const fs = require('fs');
const folderName = 'public3';
console.clear();
try {
   if(!fs.existsSync(folderName)){
       fs.mkdirSync(folderName);
       console.log('folder Made');
   }else{
       console.log('folder exists');
   }
}
catch (err){
   console.log(err);
}
```

```
const folderPath = `./${folderName}`;

fs.writeFile(folderPath+'/index2.html','<h1>Hello
World</h1>',(err)=>{
   if(err) throw err;
   console.log('file created');
})


let val1 = fs.readdirSync(folderPath);
console.log(val1);
```

## 7. Node FileSystem Examples

- file system on your computer
- Check Folder Directory fs.readdir()
- Read files fs.readFile()
- Create files fs.appendFile() - will create if it does not exist
- Create multiple files using a loop
- File maker function example
- fs module is responsible for all the asynchronous or synchronous file I/O operations

## 8. Update the contents of a file

- Update files fs.appendFile() add to content
- Replace Content fs.writeFile()
- Create a logging function that writes into a file
- Create files fs.appendFile()
- Overwrite file and create it
- Adding new files
- Create logging file

## 9. Delete files using  the file system

- Delete files fs.unlink()
- delete file function code example
- readdir get files from the directory
- loop through contents of the directory get all the files
- Create a function to remove files in the folder.

## 10. Rename files using Node

- Content fs.writeFile()
- Rename files fs.rename()
- Other IO Operations
- Example code creates a function to rename all files in the folder.

```
const fs = require('fs');
const dir = './public';
let tempCounter = 1;
let newFileName = `new${tempCounter}.html`;
console.clear();
maker();

function renameFiler(fileName, newName) {
   console.log(fileName);
   fs.rename(fileName, newName, (err) => {
       if (err) throw err;
       console.log(`File Renamed - ${newName} from
${fileName}`);
   })
}
renameAll()

function renameAll() {
   fs.readdir(dir, (err, files) => {
       files.forEach((myFile, ind) => {
           renameFiler(dir + '/' + myFile,
`${dir}/new${ind+1}.html`);
       })
   })
}

function maker() {
   for (let i = 0; i < 5; i++) {
       fs.readdir(dir, (err, files) => {
           console.log(files);
           console.log(files.length);
           console.log(err);
           tempCounter = files.length + i + 1;
           newFileName = `new${tempCounter}.html`;
           createNewFile(newFileName);
       })
   }
} //removeAll()
function removeAll() {
   fs.readdir(dir, (err, files) => {
       files.forEach((myFile) => {
           delFile(dir + '/' + myFile);
       })
   })
} //delFile(dir + '/log.html');
function delFile(val) {
   fs.unlink(val, (err) => {
       if (err) throw err;
       console.log(`file Deleted ${val}`);
   })
}
//myLog('test2');
function overmyLog(val) { //overwriting file /create
   const html = ` - ${val}`;
   fs.writeFile(dir + '/log.html', html, (err) => {
       if (err) throw err;
       console.log(`Updated writeFile ${val}`);
   })
}

function myLog(val) { //adding /create
   const html = ` <br> * ${val}`;
   fs.appendFile(dir + '/log.html', html, (err) => {
       if (err) throw err;
       console.log(`*Appended ${val}`);
   })
}

function createNewFile(myfileName) {
   const html = `<h1>Hello ${tempCounter} </h1>`;
   fs.appendFile(dir + '/' + myfileName, html, (err) => {
       if (err) throw err;
       console.log(`Save File ${myfileName}`)
   })
```

```
}
```

## 11 File System Stats

- Output html file in http web browser
- fs.stat()
- Get file from directory
- createServer http
- Fs read files and create output
- require('fs')
- Create HTTP output from file
- stats.isFile()
- stats.isDirectory()
- stats.size

```
const fs = require('fs');
const http = require('http');
const dir = './public';
const mainFile = 'new1.html';
const url = `${dir}/${mainFile}`;
console.log(url);

http.createServer((req,res)=>{
   fs.readFile(url,(err,html)=>{
       res.writeHead(200,{
           'Content-Type':'text/html'
       });
       ///console.log(html);
       res.write(html);
       return res.end();
   })
   console.log('server running');
}).listen(8080);


const rs = fs.createReadStream(url);
rs.on('open',(e)=>{
   console.log('File is open');
   let val = '<br>Opened';
   updateFile(val);
})

function updateFile(val){
   fs.appendFile(url,val,(err)=>{
       if(err) throw err;
       console.log('updated file');
   })
}

fs.stat(url,(err,stats)=>{
   console.log(stats.isFile());
   console.log(stats.isDirectory());
   console.log(stats.size);
})
```

## 12. Reading Files FS module

- Read file contents readFileSync
- fs.readFile() and fs.readFileSync() read contents of the file in memory
- Reading a file
- Create add to File contents with array items
- Read contents of new files

```
const fs = require('fs');
const fileName = './test.txt';
const data = fs.readFileSync(fileName,'utf8');
console.log(data);

const arr = ['red','blue','green'];

function addMe(data){
   fs.appendFile(fileName,`\n${data}`,(err)=>{
       if(err) throw err;
       console.log('DONE');
   })
}

arr.forEach((ele)=>{
   addMe(ele);
})
```

## 13. File and Folder Paths

- Use of Path module
- Get dirname() basename() extname()
- Path Resolve
- require('path')
- Files on File System

```
const path = require('path');
const fileName = 'new1.html';
const dir = 'public';
console.clear();
const url = path.join('/','.',dir,fileName);
console.log(url);

console.log(path.dirname(url));
console.log(path.basename(url));
console.log(path.extname(url));

const info = path.resolve(fileName);
console.log(info);
```

## 14. Node Package Package.json Fetch

- Adding dependency in package.json
- npm help init
- npm install <pkg>
- `npm install node-fetch`
- `npm init -y - creates create a package.json file in the director`
- npm init - to setup package.json

- Package.json is actual JSON file
- Installing packages globally
- Create JSON
- --save at the end of the install command to add dependency entry into package.json

```
const fetch = require('node-fetch');
console.clear();

fetch('https://google.com')
.then(res => res.text())
.then(data => console.log(data));


fetch('https://randomuser.me/api/?results=5')
.then(res => res.json())
.then(json => {
   //console.log(json.results);
   json.results.forEach(person => {
       const temp = person.name;
       console.log(`${temp.first} ${temp.last}`);
   });
});
```

## 15. Node Package Manager Module

- The Node Package Manager
- Once the package is installed, it is ready to use
- Include the package into your code with require()
- Searching for modules
- Package.json
- Installing Packages Locally
- npm install <package name>

- NPM are installed under node_modules
- npm install -g global install flag
- npm update <package name> update package
- Remove package npm uninstall <package name>
- Create package.json npm init - to setup package.json
- $ sudo npm i -g npm
- $ sudo npm i --save lodash
- https://lodash.com/
- https://www.npmjs.com/package/lodash
- https://lodash.com/docs/4.17.15#times

```
const lodash = require('lodash');

//console.log(lodash);

let ran1 = lodash.random(100);
console.log(ran1);

let ran2 = lodash.random(100,1000);
console.log(ran2);

const arr = [3,5,77,1,3,2323,3,3223324,'test'];

console.log(lodash.shuffle(arr));

lodash.times(10,()=>{
   console.log(lodash.random(100));
})
```

```json
{
 "name": "lsvekis",
 "version": "1.0.0",
 "main": "apps.js",
 "scripts": {
   "test": "echo \"Error: no test specified\" && exit 1"
 },
 "keywords": [
   "test"
 ],
 "author": "",
 "license": "ISC",
 "description": "",
 "dependencies": {
   "lodash": "^4.17.21"
 }
}
```

## 16. Node Buffers

- A buffer is an area of memory
- Buffer.from()
- Output Buffer memory
- Copy Buffer contents
- Slice Buffer Contents

```javascript
const str = "Hello World 22";
const buffer = Buffer.from(str,"utf-8");
console.log(buffer);
console.log(buffer.length);

for(const val of buffer){
   console.log(val);
}

const val2 = buffer.toString();
console.log(val2);
buffer[2] = 50;
const val3 = buffer.slice(0,5);
console.log(val3.toString());
```

## 17. Errors Throw try and Catch

- Try and catch
- Custom errors
- Extend error
- Debugging in node
- How to throw errors
- Extending errors to create custom errors

```
const fs = require('fs');
const dir = 'public';
fs.readdir(dir,(err,files)=>{
   if(err) throw err;
   //console.log(files);
})
```

```
try {
   console.lg('hello');
}
catch (err){
   console.log(err);
   //console.log('ERROR');
}

class myError extends Error {}

throw new myError('new error');
```

## Assignment Section #4

# #SECTION 5 Node Web Application with Express

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.  Explore how to setup a web application quickly with Express.

The arguments which are available to an Express JS route handler function are request (request Object), response (response Object) and optional Next which can pass control to one of the subsequent route handlers.

1. How to setup express and output html into a localhost webpage
2. How to Serve files from  a static directory using Express
3. Values that can be used in route filtering, pattern match and reg exp
4. Route parameters with req.params
5. Express Route with full CRUD Create Read Update Delete Methods
6. Returning JSON response
7. Use of bodyparser
8. Sending data from a frontend form to node express to output into the terminal
9. Returning a JSON response to the frontend from a frontend request

## 1. Express Setup

- Installing express https://expressjs.com/
- npm install express --save
- Fast minimalist web framework for Node.js
- Easy to configure and customize
- npm install -g express
- https://www.npmjs.com/package/express

```
const express = require('express');
const app = express();
const port = process.env.PORT || 8080;

app.get('/',(req,res)=>{
   res.send('<h1>Hello</h1>');
})
app.get('/test',(req,res)=>{
   res.send('<h1>Test</h1>');
})

app.listen(port,()=>{
   console.log(`Ready listening on ${port}`);
})
```

## 2. Serving Files with Express Framework

- Creating a static route
- __dirname to get current file directory
- Path join to create file path
- res.sendFile response send file from folder
- Create and return page output

```
const express = require('express');
const path = require('path');
const app = express();
const port = process.env.PORT || 8080;

const fileName = 'index.html';
const url = path.join(__dirname,'public','index.html');
console.log(url);

app.use(express.static(__dirname+'/public')); //static folder
app.get('/',(req,res)=>{
   //res.sendFile(url);
   res.sendFile(fileName);
})

app.get('/test',(req,res)=>{
   res.send('<h1>Test</h1>');
})

app.listen(port,()=>{
   console.log(`Ready listening on ${port}`);
})
```

## 3 Express Routing

- Routing and layers in express
- Routing basics
- Pattern matching in routes
- Route parameters
- Reg expression in URI
- app express() get routes
- Return response send html back

## 4 Multiple CallBack Functions

- Next() adding callbacks
- use of req.params
- get the id
- request and response within callback
- passing req.params into callback

```
const express = require('express');
const app = express();
const port = process.env.PORT || 8080;

app.get('/',(req,res)=>{
   res.send('<h1>Test</h1>');
})

app.get('/info',(req,res)=>{
   res.send('<h1>Info</h1>');
})
//pattern match
app.get('/ab?cd',(req,res)=>{
   res.send('<h1>ABC</h1>');
})

app.get('/ab*de',(req,res)=>{
   res.send('<h1>AB__________DE</h1>');
})

//reg exp
app.get(/z/,(req,res)=>{
   res.send('<h1>ZZZZZZZZ</h1>');
})
//route parameters
app.get('/users/:id/:val',(req,res,next)=>{
   res.send(req.params)
   console.log(req.params.id)
   next()},
   (req,res,next)=>{
       console.log(`Second callback ${req.params.val}`)
       next()
   },(req,res,next)=>{
   console.log(`third callback ${req.params.val}`)
})


app.listen(port,()=>{
   console.log(`Ready listening on ${port}`);
})
```

## 5 Express Route Handler GET

- CRUD put post delete get
- Create (POST) - Make something
- Read (GET)- Get something
- use of __dirname
- response sendFile() on get method
- create HTML file
- user and users routes send response
- Test webapp in browser

## 6 Express Route POST

- CRUD Create (POST) - Make something Read (GET)- Get something
- Update (PUT) - Change something Delete (DELETE)- Remove something
- BodyParser
- POST from web page
- Bodyparser to get POST form data
- Response with URI params

## 7 Express Route Update Delete

- Update (PUT) - Change something
- Delete (DELETE)- Remove something

- Frontend Document add event listener
- Frontend fetch method to connect to endpoint route
- fetch for send update and delete method data
- Create routes for user with request params
- Client side index AJAX fetch
- Return JSON response to  AJAX request

```
const express = require('express');
const app = express();
const bodyparser = require('body-parser');
const port = process.env.PORT || 8080;
app.use(bodyparser.urlencoded({
   extended: true
}))

app.get('/user', (req, res) => {
   res.sendFile(__dirname + '/index.html');
})

app.post('/users', (req, res) => {
   console.clear();
   console.log(req.body);
   res.send(`DONE:User ${req.body.user} ID  ${req.body.id}`);
})

app.listen(port, () => {
   console.log(`Ready listening on ${port}`);
})

app.get('/users/:id', (req, res) => {
   const id = req.params.id;
   console.log('found ' + id);
   res.send('found ' + id);
})

app.delete('/users/:id', (req, res) => {
   const id = req.params.id;
   console.log('deleted ' + id);
   res.send({
       status: 'delete',
       id: id
   });
})

app.put('/users/:id', (req, res) => {
   const id = req.params.id;
   console.log('Updated ' + id);
   res.send({
       status: 'updated',
       id: id
   });
})
```

```
<html>

<head>
   <title>Form</title>
</head>

<body>
```

```
   <h1>Test Methods 3</h1>

   <form action="/users" method="POST">
       <input type="text" name="user" value="Laurence">
       <input type="text" name="id" value="100">
       <button type="submit">Submit</button>
   </form>
   <button class="btn">Click</button>
   <script>
       const btn = document.querySelector('.btn');
       const myId = document.querySelector('input[name="id"]');
       btn.addEventListener('click', () => {
           console.log('clicked');
           const endPoint = '/users';
           const id = myId.value;
           fetch(endPoint + '/' + id, {
                   method: 'delete'
               }).then(res => res.json())
               .then(data => console.log(data));
           fetch(endPoint + '/' + id, {
                   method: 'put'
               }).then(res => res.json())
               .then(data => console.log(data));
       })
   </script>
</body>

</html>
```

## 8 JSON Data POST Values

- Array Methods filter and some
- Response json() method
- Return json data
- Request endpoint with params
- Response json data
- Filter some array functions
- Return matching results from javascript object
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/some
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

## 9 Output JSON URL Params

- Loop through data from the response JSON
- Output Server side response to web page
- Get route with request params
- Frontend fetch request for AJAX
- Adding frontend browser event listeners
- Custom module with data for app

```
const express = require('express');
const app = express();
const bodyparser = require('body-parser');
const myObj = require('./users');
const port = process.env.PORT || 8080;
app.use(bodyparser.urlencoded({
   extended: true
}))

console.log(myObj);
console.log(myObj.users.some((ele) => ele.id == 1));
app.get('/users/:id', (req, res) => {
   console.log(req.params)
   let myJson = [];
   if (myObj.users.some((ele) => ele.id ===
parseInt(req.params.id))) {
       myJson = myObj.users.filter((ele) => ele.id ===
parseInt(req.params.id))
   }
   res.json(myJson)
})


app.get('/user', (req, res) => {
   res.sendFile(__dirname + '/index.html');
})


app.listen(port, () => {
   console.log(`Ready listening on ${port}`);
})
```

```
<html>

<head>
   <title>Form</title>
</head>

<body>
   <h1>Test Methods 3</h1>
```

```
<form action="/users" method="POST">
    <input type="text" name="user" value="Laurence">
    <input type="text" name="id" value="100">
    <button type="submit">Submit</button>
</form>
<button class="btn">Click</button>
<button class="btn1">Get JSON</button>
<button class="btn2">Get User</button>
<script>
    const btn2 = document.querySelector('.btn2');
    const btn = document.querySelector('.btn');
    const myId = document.querySelector('input[name="id"]');
    const btn1 = document.querySelector('.btn1');


    btn2.addEventListener('click', () => {
        fetch(`/users/${myId.value}`).then(res => res.json())
            .then((data) => {
                console.log(data)
                if (data.length != 0) {
                    data.forEach(element => {
                        console.log(element);
                    });
                } else {
                    console.log('not found');
                }
            });
    })



    btn1.addEventListener('click', () => {
        fetch('/users').then(res => res.json())
            .then((data) => {
                console.log(data)
                data.users.forEach(element => {
                    console.log(element);
                });
            });
    })


    btn.addEventListener('click', () => {
        console.log('clicked');
        const endPoint = '/users';
        const id = myId.value;
        fetch(endPoint + '/' + id, {
                method: 'delete'
            }).then(res => res.json())
            .then(data => console.log(data));
        fetch(endPoint + '/' + id, {
                method: 'put'
            }).then(res => res.json())
            .then(data => console.log(data));
    })
```

```
   </script>
</body>

</html>
```

```
const myObj = {
   users: [{
           id: 1,
           name: "Laurence"
       },
       {
           id: 2,
           name: "Mike"
       },
       {
           id: 3,
           name: "Jane"
       },
   ]
}

module.exports = myObj;
```

## 10. MiddleWare POST with bodyParser

- Additional middleware functionality
- Ordering of middleware
- var bodyParser = require("body-parser");
- npm install body-parser --save
- Create your own middleware

```
const express = require('express');
const app = express();
const bodyparser = require('body-parser');
const myObj = require('./users');
const port = process.env.PORT || 8080;
const logOutput = (req,res,next) =>{
   console.log('Middleware in action');
   console.log(req.params);
   next();
}
app.use(logOutput);
app.use(bodyparser.urlencoded({
   extended: true
}))

console.log(myObj);
console.log(myObj.users.some((ele) => ele.id == 1));
app.get('/users/:id', (req, res) => {
   console.log(req.params)
   let myJson = [];
   if (myObj.users.some((ele) => ele.id ===
parseInt(req.params.id))) {
       myJson = myObj.users.filter((ele) => ele.id ===
parseInt(req.params.id))
   }
   res.json(myJson)
})
```

```
app.post('/users',(req,res) =>{
   console.log(req.body);
   res.sendFile(__dirname + '/index.html');
})

app.get('/user', (req, res) => {
   res.sendFile(__dirname + '/index.html');
})



app.listen(port, () => {
   console.log(`Ready listening on ${port}`);
})
```

## 11 Express routes Router

- chained route handlers
- using app.route()
- express.Router
- express.Router()
- Middleware express.Router()

```
const express = require('express');
const app = express();
const router = express.Router()
const bodyparser = require('body-parser');
const myObj = require('./users');
const port = process.env.PORT || 8080;
const users = require('./user');

app.use('/users', users);
```

```
/*
app.route('/users')
.get((req,res) =>  res.send('Test GET'))
.post((req,res) => res.send('Test 2'))
.put((req,res) => res.send('Test 3'))
*/

app.listen(port, () => {
   console.log(`Ready listening on ${port}`);
})
```

```
const express = require('express');
const router = express.Router()

router.use((req,res,next)=>{
   console.log('Users middleware');
   next()
})

router.get('/',(req,res)=>{
   res.send('Main User');
})
router.get('/info',(req,res)=>{
   res.send('User info');
})

module.exports = router;
```

## 12 Express application generator

- npm install -g express-generator
- express -h or express to run
- express --view=pug testApp
- quick start and way to develop Express JS applications
- Express Generator is also a Node JS Module
- Quickly create an application skeleton
- npm install -g express-generator

# Assignment Section #5

Setup a frontend application that can send data to the backend. Use of both Get and Post methods to different routes in Express.

1. Create your express server run the express server
2. Add body-parser module middleware
3. Setup app get route to return the value in the request query. Try within your browser to ensure you can get the value of id and send it as output to the page.
4. Setup app.post use the request body object req.body and return as JSON back to the client side
5. Setup the frontend code with a web page that contains a form that can Post to the node route for post. Add 2 input fields and a submit button sending the data to node express.

# #SECTION 6 MongoDB with Frontend web page interaction using Node and Express

The official MongoDB Node. js driver allows Node. js applications to connect to MongoDB and work with data. The driver features an asynchronous API which allows you to access method return values through Promises or specify callbacks to access them when communicating with MongoDB.

1. How to setup a cloud base mongo Database to use in practice development
2. Connecting to a mongoDB
3. Use of mongoDB client in Node
4. Checking a connection to the database
5. How to connect to a MongoDB and select the collections, create collections
6. How to output database collections to the frontend using Express and Node.

Creating a fully functional node application that uses Express to run a web application. Connect to a MongoDB and using Node allow the frontend web application to access the Database items, full CRUD interaction.

## 1. Setup Get Started with MongoDB

- Setting up Mongo DB
- Installing MongoDB
- Using MongoDB in Node.js
- MongoClient = require('mongodb').MongoClient;
- It's all JavaScript
- Understanding the Basic Operations
- sudo npm i express
- sudo npm i body-parser
- sudo npm i mongodb
- Setup Express Server
- Ready for Database

## 2. Create MongoDB in the Cloud

- Access cloud database https://www.mongodb.com
- MongoDB Atlas is the global cloud database service for modern applications.
- Deploy fully managed MongoDB
- Free MongoDB online

```
const express = require('express');
const bodyParser = require('body-parser');
const MongoClient = require('mongodb').MongoClient;
const router = express.Router();
const app = express();

app.get('/',(req,res)=>{
   res.sendFile(__dirname+'/public/index.html');
})

app.listen(8080,()=>{
   console.log('server ready');
})
```

```
<!DOCTYPE html>
<html><head><title>Mongo Tester</title></head>
<body>
   <h1>Hello Mongo</h1>
</body>
</html>
```

```
{
 "name": "mongo",
 "version": "1.0.0",
 "description": "",
 "main": "app.js",
 "scripts": {
   "test": "echo \"Error: no test specified\" && exit 1"
 },
 "author": "",
 "license": "ISC",
 "dependencies": {
   "body-parser": "^1.19.0",
   "express": "^4.17.1",
   "mongodb": "^3.6.6"
 }
}
```

## 3. Connect and Create MongoDB

- MongoClient.connect()
- Connecting and creating a database
- const mongo = require('mongodb');
- Setup of Database user and password
- Key and Path to connect to cloud database

```
const express = require('express');
const bodyParser = require('body-parser');
const MongoClient = require('mongodb').MongoClient;
const router = express.Router();
const app = express();
const url = require('./secret.js');

MongoClient.connect(url,(err,db)=>{
   if(err) throw err;
   console.log('Connected');
   db.close();
})
```

```
app.get('/',(req,res)=>{
   res.sendFile(__dirname+'/public/index.html');
})

app.listen(8080,()=>{
   console.log('server ready');
})
```

## 4. Create collection of Data

- MongoClient = require('mongodb').MongoClient;
- Structuring your data for MongoDB
- Creating Collections
- Inserting documents into collections
- Accessing collection
- insert to cloud Database collection
- client.connect using Mongo

```
const express = require('express');
const bodyParser = require('body-parser');
const MongoClient = require('mongodb').MongoClient;
const router = express.Router();
const app = express();
const url = require('./secret.js');
const client = new MongoClient(url,{
   useNewUrlParser : true,
   useUnifiedTopology : true
})

client.connect(err =>{
   const coll = client.db('people').collection('friends');
   console.log('ready');
   const myObj = {name:"Laurence"};
   coll.insertOne(myObj,(err,res)=>{
       console.log('inserted');
       client.close();
   })

})



app.get('/',(req,res)=>{
   res.sendFile(__dirname+'/public/index.html');
})

app.listen(8080,()=>{
   console.log('server ready');
})
```

## 5. WebPage App to Database Setup

- setup express server
- Output index page to browser
- Setup users route
- Database connect to client table and collection
- Setup CRUD routes
- Setup BodyParser middleware
- Database insertOne into collection MongoDB
- JSON.stringify results into console
- Send JSON results back to client side
- Setup HTML web page to request data
- AJAX with eventListener to trigger request

## 6 WebPage AJAX to Endpoint

- insertOne JS object
- app.use(bodyParser.json());
- Document Object DOM
- JavaScript web page AJAX request
- addEventListener to page element
- Fetch request to node route users
- Send headers as json data
- Setup of data from webpage
- Check Cloud Database insert of item

## 7. AJAX Post setup to Database

- AJAX Post
- test headers
- Test request to Node and Database insert
- JSON. stringify

```
const express = require('express');
const bodyParser = require('body-parser');
const MongoClient = require('mongodb').MongoClient;
const router = express.Router();
const app = express();
const url = require('./secret.js');
app.use(bodyParser.json());
const client = new MongoClient(url, {
   useNewUrlParser: true,
   useUnifiedTopology: true
})

client.connect(err => {
   const myDB = client.db('people').collection('friends');
   app.route('/users')
       .get((req, res) => {

       })
       .post((req, res) => {
           console.log(req.body);
           myDB.insertOne(req.body).then(results => {
               console.log(req.body);
               res.contentType('application/json');
               res.send(JSON.stringify(req.body))
           })
       })
       .put((req, res) => {

       })
       .delete((req, res) => {

       })
})



app.get('/', (req, res) => {
   res.sendFile(__dirname + '/public/index.html');
})

app.listen(8080, () => {
   console.log('server ready');
})

```

```
<!DOCTYPE html>
<html>

<head>
   <title>Mongo Tester</title>
</head>

<body>
   <h1>Hello Mongo</h1>
   <div>
       <input type="text" name="user" value="Laurence">
       <button class="btn1">Add User</button>
   </div>
   <script>
       const input1 =
document.querySelector('input[name="user"]');
```

```
       const btn1 = document.querySelector('.btn1');
       const endPoint = '/users';

       btn1.addEventListener('click', () => {
           const data = {
               name: input1.value
           };
           fetch(endPoint, {
               method: 'post',
               body: JSON.stringify(data),
               headers: {
                   'Content-Type': 'application/json'
               }
           }).then(res => res.json()).then(data => {
               console.log(data);
           })
       })
   </script>
</body>

</html>
```

## 8. Ajax get Mongo Data

- Frontend AJAX setup Fetch
- Database results
- toArray() return results as array
- Use find()
- Send as JSON
- Web Page document createElements for output of response results

- Add eventlisteners to page elements

## 9 Dynamic Page Elements AJAX

- Create output using the DOM
- Making Page elements interactive with event listeners
- Setup for DELETE method and PUT to node

```
const express = require('express');
const bodyParser = require('body-parser');
const MongoClient = require('mongodb').MongoClient;
const router = express.Router();
const app = express();
const url = require('./secret.js');
app.use(bodyParser.json());
const client = new MongoClient(url, {
   useNewUrlParser: true,
   useUnifiedTopology: true
})

client.connect(err => {
   const myDB = client.db('people').collection('friends');
   app.route('/users')
       .get((req, res) => {
           myDB.find().toArray().then(results => {
               console.log(results);
               res.contentType('application/json');
               res.send(JSON.stringify(results))
           })
       })
       .post((req, res) => {
           console.log(req.body);
           myDB.insertOne(req.body).then(results => {
               console.log(req.body);
               res.contentType('application/json');
               res.send(JSON.stringify(req.body))
           })
       })
       .put((req, res) => {

       })
       .delete((req, res) => {

       })

})



app.get('/', (req, res) => {
   res.sendFile(__dirname + '/public/index.html');
})

app.listen(8080, () => {
   console.log('server ready');
})
```

```
<!DOCTYPE html>
```

```
<html>

<head>
   <title>Mongo Tester</title>
</head>

<body>
   <h1>Hello Mongo</h1>
   <div>
       <input type="text" name="user" value="Laurence">
       <button class="btn1">Add User</button>
   </div>
   <button class="btn2">List Data</button>
   <div class="output"></div>
   <script>
       const input1 =
document.querySelector('input[name="user"]');
       const btn1 = document.querySelector('.btn1');
       const btn2 = document.querySelector('.btn2');
       const output = document.querySelector('.output');
       const endPoint = '/users';

       function createOutput(data){
           output.innerHTML = '<div>Users</div>';
           data.forEach(element => {
               console.log(element);
               const main = document.createElement('div');
               const myInput = document.createElement('input');
               myInput.setAttribute('type','text');
               myInput.value = element.name;
               main.append(myInput);

               const span1 =  document.createElement('span');
               span1.textContent = element._id;
               main.append(span1);

               const button1 =
document.createElement('button');
               button1.textContent = 'Delete';
               button1.style.color = 'red';
               button1.addEventListener('click',()=>{
                   console.log('DELETE');
               })
               main.append(button1);

               const button2 =
document.createElement('button');
               button2.textContent = 'Update';
               button2.style.color = 'green';
               button2.addEventListener('click',()=>{
                   console.log('PUT');
               })
               main.append(button2);

               output.append(main);
           });
```

```
        }



        btn2.addEventListener('click',()=>{
            fetch(endPoint).then(res => res.json())
            .then(data => {
                createOutput(data);
            })
        })

        btn1.addEventListener('click', () => {
            const data = {
                name: input1.value
            };
            fetch(endPoint, {
                method: 'post',
                body: JSON.stringify(data),
                headers: {
                    'Content-Type': 'application/json'
                }
            }).then(res => res.json()).then(data => {
                console.log(data);
            })
        })
    </script>
</body>

</html>
```

## 10. Find matches from MongoDB Get

- Find using req.params
- JSON results
- Return matches to webpage
- Output content to webpage

```
const express = require('express');
const bodyParser = require('body-parser');
const MongoClient = require('mongodb').MongoClient;
const router = express.Router();
const app = express();
const url = require('./secret.js');
app.use(bodyParser.json());
const client = new MongoClient(url, {
   useNewUrlParser: true,
   useUnifiedTopology: true
})

client.connect(err => {
   const myDB = client.db('people').collection('friends');
   app.get('/user/:name',(req,res)=>{
       console.log(req.params);
       myDB.find(req.params).toArray().then(results => {
           console.log(results);
           res.contentType('application/json');
           res.send(JSON.stringify(results))
       })
   })

   app.route('/users')
```

```
       .get((req, res) => {
           myDB.find().toArray().then(results => {
               console.log(results);
               res.contentType('application/json');
               res.send(JSON.stringify(results))
           })
       })
       .post((req, res) => {
           console.log(req.body);
           myDB.insertOne(req.body).then(results => {
               console.log(req.body);
               res.contentType('application/json');
               res.send(JSON.stringify(req.body))
           })
       })
       .put((req, res) => {

       })
       .delete((req, res) => {

       })

})



app.get('/', (req, res) => {
   res.sendFile(__dirname + '/public/index.html');
```

```
})

app.listen(8080, () => {
   console.log('server ready');
})
```

```
<!DOCTYPE html>
<html>

<head>
   <title>Mongo Tester</title>
</head>

<body>
   <h1>Hello Mongo</h1>
   <div>
       Add new User
       <input type="text" name="user" value="Laurence">
       <button class="btn1">Add User</button>
   </div>
   <div>
       Find in Database Filter
       <input type="text" name="finder" value="Laurence">
       <button class="btn3">Find Match</button>
   </div>
   <button class="btn2">List Data</button>
   <div class="output"></div>
   <script>
```

```
       const input1 =
document.querySelector('input[name="user"]');
       const input2 =
document.querySelector('input[name="finder"]');
       const btn1 = document.querySelector('.btn1');
       const btn2 = document.querySelector('.btn2');
       const btn3 = document.querySelector('.btn3');
       const output = document.querySelector('.output');
       const endPoint = '/users';



       function createOutput(data){
           output.innerHTML = '<div>Users</div>';
           data.forEach(element => {
               console.log(element);
               const main = document.createElement('div');
               const myInput = document.createElement('input');
               myInput.setAttribute('type','text');
               myInput.value = element.name;
               main.append(myInput);

               const span1 =  document.createElement('span');
               span1.textContent = element._id;
               main.append(span1);

               const button1 =
document.createElement('button');
```

```
               button1.textContent = 'Delete';
               button1.style.color = 'red';
               button1.addEventListener('click',()=>{
                   console.log('DELETE');
               })
               main.append(button1);

               const button2 =
document.createElement('button');
               button2.textContent = 'Update';
               button2.style.color = 'green';
               button2.addEventListener('click',()=>{
                   console.log('PUT');
               })
               main.append(button2);

               output.append(main);
           });
       }

       btn3.addEventListener('click',()=>{
           fetch('/user/'+input2.value).then(res => res.json())
           .then(data => {
               console.log(data);
               createOutput(data);
           })
       })
       btn2.addEventListener('click',()=>{
           fetch(endPoint).then(res => res.json())
           .then(data => {
               createOutput(data);
           })
       })

       btn1.addEventListener('click', () => {
           const data = {
               name: input1.value
           };
           fetch(endPoint, {
               method: 'post',
               body: JSON.stringify(data),
               headers: {
                   'Content-Type': 'application/json'
               }
           }).then(res => res.json()).then(data => {
               console.log(data);
           })
       })
   </script>
</body>

</html>
```

## 11. Update MongoDB Data

- findOneAndUpdate()

- $set value
- Req.body._id
- AJAX fetch PUT method
- Send to node application put data
- Using _id as unique finder for items in MongoDB

```
const express = require('express');
const bodyParser = require('body-parser');
const MongoClient = require('mongodb').MongoClient;
const ObjectId = require('mongodb').ObjectId;
const router = express.Router();
const app = express();
const url = require('./secret.js');
app.use(bodyParser.json());
const client = new MongoClient(url, {
   useNewUrlParser: true,
   useUnifiedTopology: true
})

client.connect(err => {
   const myDB = client.db('people').collection('friends');
   app.get('/user/:name', (req, res) => {
       console.log(req.params);
       myDB.find(req.params).toArray().then(results => {
           console.log(results);
           res.contentType('application/json');
           res.send(JSON.stringify(results))
       })
   })

   app.route('/users')
```

```
    .get((req, res) => {
        myDB.find().toArray().then(results => {
            console.log(results);
            res.contentType('application/json');
            res.send(JSON.stringify(results))
        })
    })
    .post((req, res) => {
        console.log(req.body);
        myDB.insertOne(req.body).then(results => {
            console.log(req.body);
            res.contentType('application/json');
            res.send(JSON.stringify(req.body))
        })
    })
    .put((req, res) => {
        console.log(req.body);
        myDB.findOneAndUpdate({
            _id: ObjectId(req.body._id)
        }, {
            $set: {
                name: req.body.name
            }
        }, {
            upsert: false
        }).then(result => {
            res.contentType('application/json');
            res.send({
                   "status": true
               })
           })
       })
       .delete((req, res) => {

       })

})



app.get('/', (req, res) => {
   res.sendFile(__dirname + '/public/index.html');
})

app.listen(8080, () => {
   console.log('server ready');
})

```

```
<!DOCTYPE html>
<html>

<head>
   <title>Mongo Tester</title>
</head>

<body>
```

```
   <h1>Hello Mongo</h1>
   <div>
       Add new User
       <input type="text" name="user" value="Laurence">
       <button class="btn1">Add User</button>
   </div>
   <div>
       Find in Database Filter
       <input type="text" name="finder" value="Laurence">
       <button class="btn3">Find Match</button>
   </div>
   <button class="btn2">List Data</button>
   <div class="output"></div>
   <script>
       const input1 =
document.querySelector('input[name="user"]');
       const input2 =
document.querySelector('input[name="finder"]');
       const btn1 = document.querySelector('.btn1');
       const btn2 = document.querySelector('.btn2');
       const btn3 = document.querySelector('.btn3');
       const output = document.querySelector('.output');
       const endPoint = '/users';



       function createOutput(data){
           output.innerHTML = '<div>Users</div>';
           data.forEach(element => {
               console.log(element);
               const main = document.createElement('div');
               const myInput = document.createElement('input');
               myInput.setAttribute('type','text');
               myInput.value = element.name;
               main.append(myInput);

               const span1 =  document.createElement('span');
               span1.textContent = element._id;
               main.append(span1);

               const button1 =
document.createElement('button');
               button1.textContent = 'Delete';
               button1.style.color = 'red';
               button1.addEventListener('click',()=>{
                   console.log('DELETE');
               })
               main.append(button1);

               const button2 =
document.createElement('button');
               button2.textContent = 'Update';
               button2.style.color = 'green';
               button2.addEventListener('click',()=>{
                   element.name = myInput.value;
                   console.log(element);
```

```
                fetch(endPoint,{
                    method:'put',
                    body:JSON.stringify(element),
                    headers: {
                    'Content-Type': 'application/json'
                    }
                }).then(res => res.json()).then(data => {
                console.log(data);
                })
            })
            main.append(button2);

            output.append(main);
        });
    }

    btn3.addEventListener('click',()=>{
        fetch('/user/'+input2.value).then(res => res.json())
        .then(data => {
            console.log(data);
            createOutput(data);
        })
    })

    btn2.addEventListener('click',()=>{
        fetch(endPoint).then(res => res.json())
        .then(data => {
            createOutput(data);
            })
        })

        btn1.addEventListener('click', () => {
            const data = {
                name: input1.value
            };
            fetch(endPoint, {
                method: 'post',
                body: JSON.stringify(data),
                headers: {
                    'Content-Type': 'application/json'
                }
            }).then(res => res.json()).then(data => {
                console.log(data);
            })
        })
    </script>
</body>

</html>
```

## 12. Delete remove data from mongoDB with AJAX

- deleteOne()
- _id: ObjectId(req.body._id)
- Delete method with AJAX
- myDB.deleteOne()

```
const express = require('express');
const bodyParser = require('body-parser');
const MongoClient = require('mongodb').MongoClient;
const ObjectId = require('mongodb').ObjectId;
const router = express.Router();
const app = express();
const url = require('./secret.js');
app.use(bodyParser.json());
const client = new MongoClient(url, {
   useNewUrlParser: true,
   useUnifiedTopology: true
})

client.connect(err => {
   const myDB = client.db('people').collection('friends');
   app.get('/user/:name', (req, res) => {
       console.log(req.params);
       myDB.find(req.params).toArray().then(results => {
           console.log(results);
           res.contentType('application/json');
           res.send(JSON.stringify(results))
       })
   })

   app.route('/users')
       .get((req, res) => {
           myDB.find().toArray().then(results => {
               console.log(results);
               res.contentType('application/json');
               res.send(JSON.stringify(results))
```

```
        })
    })
    .post((req, res) => {
        console.log(req.body);
        myDB.insertOne(req.body).then(results => {
            console.log(req.body);
            res.contentType('application/json');
            res.send(JSON.stringify(req.body))
        })
    })
    .put((req, res) => {
        console.log(req.body);
        myDB.findOneAndUpdate({
            _id: ObjectId(req.body._id)
        }, {
            $set: {
                name: req.body.name
            }
        }, {
            upsert: false
        }).then(result => {
            res.contentType('application/json');
            res.send({
                "status": true
            })
        })
    })
    .delete((req, res) => {
           console.log(req.body);
           myDB.deleteOne({
                   _id: ObjectId(req.body._id)
               }).then(result => {
                   let boo = true;
                   if (result.deleteCount === 0) {
                       boo: false
                   }
                   res.send({
                       "status": boo
                   })
               })
               .catch(error => console.log(error))
       })

})



app.get('/', (req, res) => {
   res.sendFile(__dirname + '/public/index.html');
})

app.listen(8080, () => {
   console.log('server ready');
})
```

```
<!DOCTYPE html>
```

```
<html>

<head>
   <title>Mongo Tester</title>
</head>

<body>
   <h1>Hello Mongo</h1>
   <div>
       Add new User
       <input type="text" name="user" value="Laurence">
       <button class="btn1">Add User</button>
   </div>
   <div>
       Find in Database Filter
       <input type="text" name="finder" value="Laurence">
       <button class="btn3">Find Match</button>
   </div>
   <button class="btn2">List Data</button>
   <div class="output"></div>
   <script>
       const input1 =
document.querySelector('input[name="user"]');
       const input2 =
document.querySelector('input[name="finder"]');
       const btn1 = document.querySelector('.btn1');
       const btn2 = document.querySelector('.btn2');
       const btn3 = document.querySelector('.btn3');
       const output = document.querySelector('.output');
       const endPoint = '/users';



       function createOutput(data) {
           output.innerHTML = '<div>Users</div>';
           data.forEach(element => {
               console.log(element);
               const main = document.createElement('div');
               const myInput = document.createElement('input');
               myInput.setAttribute('type', 'text');
               myInput.value = element.name;
               main.append(myInput);

               const span1 = document.createElement('span');
               span1.textContent = element._id;
               main.append(span1);

               const button1 = document.createElement('button');
               button1.textContent = 'Delete';
               button1.style.color = 'red';
               button1.addEventListener('click', () => {
                   console.log('DELETE');
                   fetch(endPoint, {
                       method: 'delete',
                       body: JSON.stringify(element),
                       headers: {
                           'Content-Type': 'application/json'
                       }
                   }).then(res => res.json()).then(data => {
                       if (data.status) {
                           main.remove();
                       }
                   })
               })
               main.append(button1);

               const button2 = document.createElement('button');
               button2.textContent = 'Update';
               button2.style.color = 'green';
               button2.addEventListener('click', () => {
                   element.name = myInput.value;
                   console.log(element);
                   fetch(endPoint, {
                       method: 'put',
                       body: JSON.stringify(element),
                       headers: {
                           'Content-Type': 'application/json'
                       }
                   }).then(res => res.json()).then(data => {
                       console.log(data);
                   })
               })
               main.append(button2);
```

```
                output.append(main);
            });
        }

        btn3.addEventListener('click', () => {
            fetch('/user/' + input2.value).then(res =>
res.json())
                .then(data => {
                    console.log(data);
                    createOutput(data);
                })
        })

        btn2.addEventListener('click', () => {
            fetch(endPoint).then(res => res.json())
                .then(data => {
                    createOutput(data);
                })
        })

        btn1.addEventListener('click', () => {
            const data = {
                name: input1.value
            };
            fetch(endPoint, {
                method: 'post',
                body: JSON.stringify(data),
                headers: {
                    'Content-Type': 'application/json'
                }
            }).then(res => res.json()).then(data => {
                console.log(data);
            })
        })
    </script>
</body>

</html>
```

## 13. More with MongoDB documents and Coding

- https://docs.mongodb.com/manual/
- db.inventory.updateMany()
- Update multiple documents
- https://docs.mongodb.com/guides/

### Assignment Section #6

Creating a fully functional node application that uses Express to run a web application. Connect to a MongoDB and using Node allow the frontend web application to access the Database items, full CRUD interaction.

1. Create a web application using Express - output html file from public folder
2. Setup a MongoDB in the cloud prep to connect using NodeJS
3. Create and test insert into the database
4. Setup Frontend web application
5. Add DOM event listeners to trigger requests
6. Generate visual response as HTML from the Node response object
7. Update and create interactive list of database content CRUD
8. Add HTML updates to send requests to the database.