NODE JS

Prof. Vinay Joshi and Dr. Sarasvathi V
Department of Computer Science and Engineering

Acknowledgement
The slides are created from various internet resources with valuable contributions from multiple professors.

NODE JS

Module System

Department of Computer Science and Engineering

Node JS Basics

Datatypes: Node.js contains various types of data types similar to JavaScript.

Boolean
Undefined
Null
String
Number

```
var a = 35;
console.log(typeof a);

// Variable store string data type
a = “PESU";
console.log(typeof a);
```

NODE JS

Node JS basics

```
// Variable store Boolean data type
a = true;
console.log(typeof a);

// Variable store undefined (no value) data type
a = undefined;
console.log(typeof a);
```

NODE JS

Node JS basics

Objects & Functions: Node.js objects are same as JavaScript objects i.e. the objects are similar to variable and it contains many values which are written as name : value pairs. Name and value are separated by colon and every pair is separated by comma.

Example:

```
var company = {
   Name: “PESU",
   Address: "Noida",
   Contact: "+919876543210",
   Email: “abc@gmail.com"
};

// Display the object information
console.log("Information of variable company:", company);

// Display the type of variable
console.log("Type of variable company:", typeof company);
```

Output:

Node JS basics

Functions:

Node.js functions are defined using function keyword then the name of the function and parameters which are passed in the function.

In Node.js, we don’t have to specify datatypes for the parameters and check the number of arguments received.

Node.js functions follow every rule which is there while writing JavaScript functions.

NODE JS

Node JS basics

Example:

```
function multiply(num1, num2) {

    // It returns the multiplication
    // of num1 and num2
    return num1 * num2;
}

// Declare variable
var x = 2;
var y = 3;

// Display the answer returned by
// multiply function
console.log("Multiplication of", x, "and", y, "is", multiply(x, y));
```

NODE JS

Node JS basics-REPL

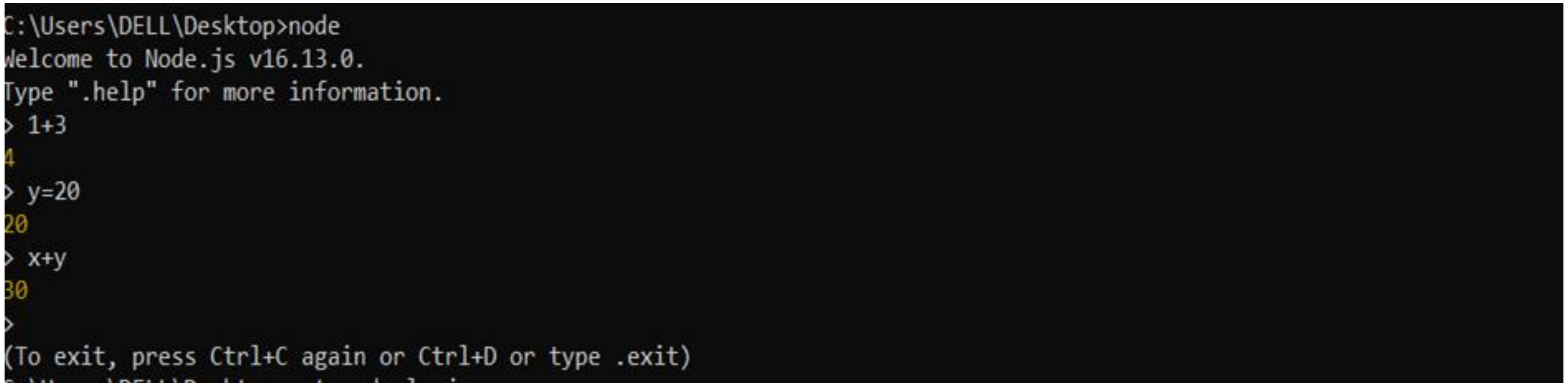
REPL (READ, EVAL, PRINT, LOOP) is a computer environment similar to Shell (Unix/Linux) and command prompt.

- Node comes with the REPL environment when it is installed.
- System interacts with the user through outputs of commands/expressions used.
- It is useful in writing and debugging the codes.

The work of REPL can be understood from its full form:

Read : It reads the inputs from users and parses it into JavaScript data structure. It is then stored to memory.
Eval : The parsed JavaScript data structure is evaluated for the results.
Print : The result is printed after the evaluation.
Loop : Loops the input command. To come out of NODE REPL, press ctrl+c twice

Node JS basics-REPL



```
C:\Users\DELL\Desktop>node
Welcome to Node.js v16.13.0.
Type ".help" for more information.
> 1+3
4
> y=20
20
> x+y
30
>
(To exit, press Ctrl+C again or Ctrl+D or type .exit)
```

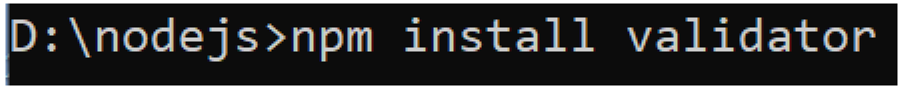
NPM and installing modules

- NPM is a package manager for Node.js packages, or modules if you like.
- www.npmjs.com hosts thousands of free packages to download and use.
- The NPM program is installed on your computer when you install Node.js

  A package in Node.js contains all the files you need for a module.

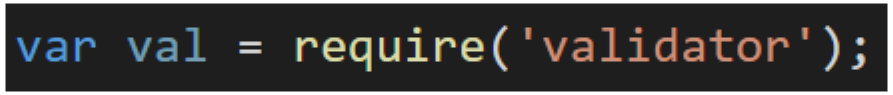
  Modules are JavaScript libraries you can include in your project.

Example:



```
D:\nodejs>npm install validator
```

- validator package is downloaded and installed. NPM creates a folder named "node_modules", where the package will be placed.
- To include a module, use the **require()** function with the name of the module



```
var val = require('validator');
```

Node JS Modules

What is a Module in Node.js?

- Modules are the blocks of encapsulated code that communicates with an external application on the basis of their related functionality.
- Modules can be a single file or a collection of multiples files/folders.
- The reason programmers are heavily reliant on modules is because of their re-usability as well as the ability to break down a complex piece of code into manageable chunks

Node JS Modules

There are 2 types of Modules:

Built in Modules

Local Modules

Node JS Modules

- https://nodejs.org/dist/latest-v12.x/docs/api/

  To see all the available modules

- Few modules are inbuilt globally available.

  Ex: Console module, Timer Module

- Many modules need to be explicitly included in our application

  Ex: File System module

Such modules need to be required at first in the application

NODE JS

Node JS built-in Modules

Node.js has many built-in modules that are part of the platform and comes with Node.js installation. **These modules can be loaded into the program by using the require function.**

Syntax:

var module = require('module_name’);

The require() function will return a JavaScript type depending on what the particular module returns. The following example demonstrates how to use the Node.js Http module to create a web server.

Importing your own modules

- The module.exports is a special object which is included in every JavaScript file in the Node.js application by default.
- The module is a variable that represents the current module, and exports is an object that will be exposed as a module.
- So, whatever you assign to module. exports will be exposed as a module. It can be
  - Export Literals
  - Export Objects
  - Export Functions
  - Export Function as a class

Node JS Global objects

Global Objects are the objects that are available in all modules.

Global Objects are built-in objects that are part of the JavaScript and can be used directly in the application without importing any particular module.

1.Class: Buffer

The Buffer class is an inbuilt globally accessible class that means it can be used without importing any module. The Buffer class is used to deal with binary data. Buffer class objects are used to represent binary data as a sequence of bytes.

2. **console:** It is an inbuilt global object used to print to stdout and stderr.

3. global: It is a global namespace. Defining a variable within this namespace makes it globally accessible.

var myvar

Node JS Timer Module

- This module provides a way for functions to be called later at a given time.
- The Timer object is a global object in Node.js, and it is not necessary to import it

Method	Description
clearImmediate()	Cancels an Immediate object
clearInterval()	Cancels an Interval object
clearTimeout()	Cancels a Timeout object
ref()	Makes the Timeout object active. Will only have an effect if the Timeout.unref() method has been called to make the Timeout object inactive.
setImmediate()	Executes a given function immediately.
setInterval()	Executes a given function at every given milliseconds
setTimeout()	Executes a given function after a given time (in milliseconds)
unref()	Stops the Timeout object from remaining active

NODE JS

Node JS Modules

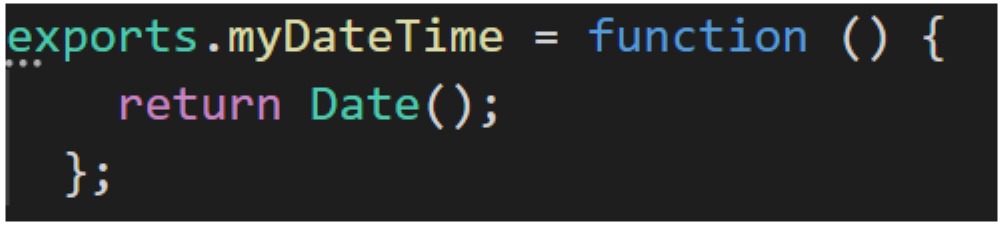
```
function printHello() {
   console.log( "Hello, World!");
}

// Now call above function after 2 seconds
var timeoutObj = setTimeout(printHello, 2000);
```

NODE JS

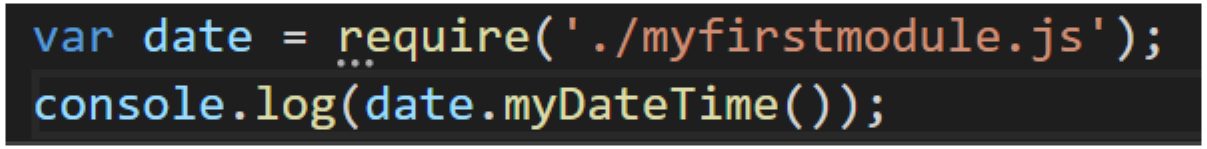
Importing your own modules

You can create your own modules, and easily include them in your applications. The following example creates a module that returns a date and

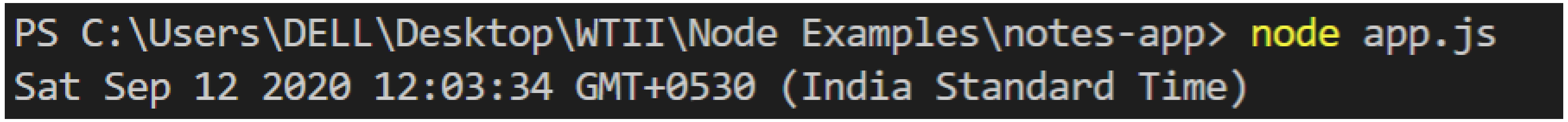


```
exports.myDateTime = function () {
    return Date();
  };
```

Use the exports keyword to make properties and methods available outside the module file.



```
var date = require('./myfirstmodule.js');
console.log(date.myDateTime());
```



```
PS C:\Users\DELL\Desktop\WTII\Node Examples\notes-app> node app.js
Sat Sep 12 2020 12:03:34 GMT+0530 (India Standard Time)
```

Node JS Local-Modules

Local in modules example:

local modules are created locally in your Node.js application. Let’s create a simple calculating module that calculates various operations. Create a calc.js file that has the following code:

Filename: calc.js

```
exports.add = function (x, y) {
   return x + y;
};

exports.sub = function (x, y) {
   return x - y;
};

exports.mult = function (x, y) {
   return x * y;
};
```

Node JS Modules

Since this file provides attributes to the outer world via exports, another file can use its exported functionality using the require() function.

Filename: index.js

```
var calculator = require('./calc');

var x = 50, y = 20;

console.log("Addition of 50 and 10 is "
            + calculator.add(x, y));

console.log("Subtraction of 50 and 10 is "
            + calculator.sub(x, y));

console.log("Multiplication of 50 and 10 is "
            + calculator.mult(x, y));
```

Step to run this program: Run index.js file using the following command:

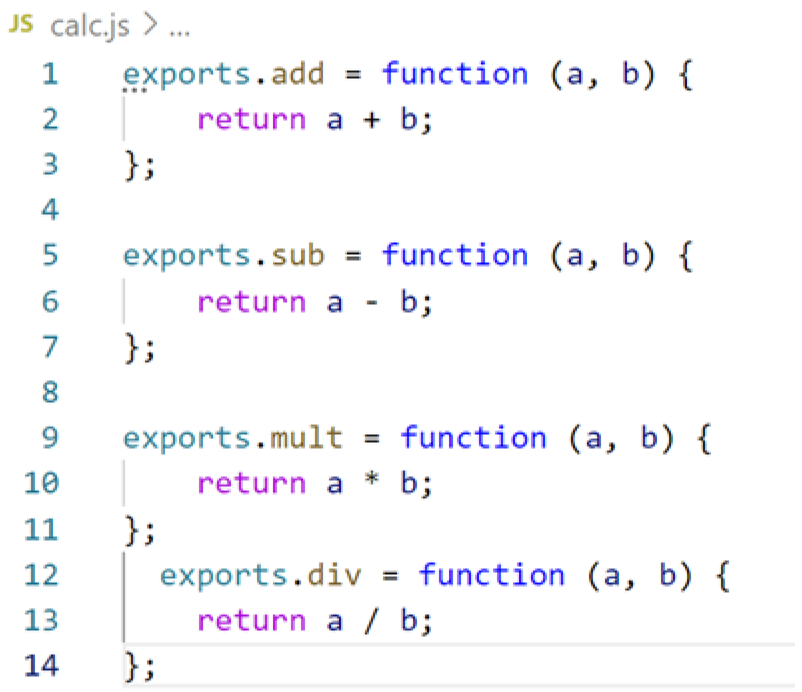
```
node index.js
```

NODE JS

Demo for Modules

Create Modules in Node.js

- To create a module in Node.js, use **exports** keyword tells Node.js that the function can be used outside the module.
- **Create a file that you want to export**



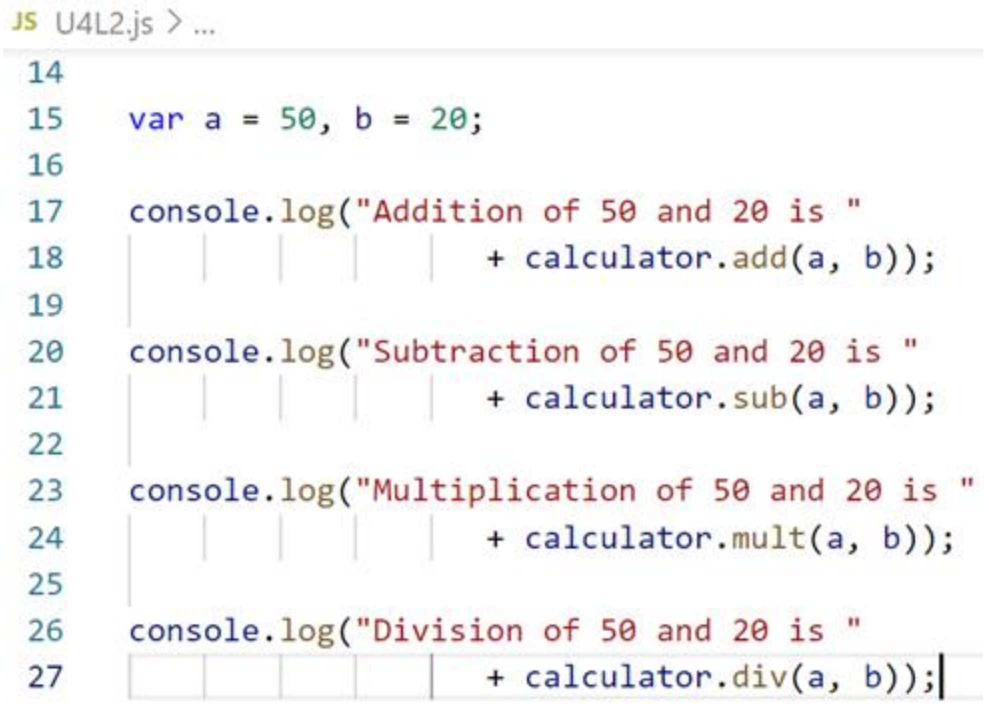
calc.js > ...

```
exports.add = function (a, b) {
    return a + b;
};

exports.sub = function (a, b) {
    return a - b;
};

exports.mult = function (a, b) {
    return a * b;
};
  exports.div = function (a, b) {
    return a / b;
};
```

- **Use the ‘require’ keyword to import the file**



U4L2.js > ...

```

var a = 50, b = 20;

console.log("Addition of 50 and 20 is "
                + calculator.add(a, b));

console.log("Subtraction of 50 and 20 is "
                + calculator.sub(a, b));

console.log("Multiplication of 50 and 20 is "
                + calculator.mult(a, b));

console.log("Division of 50 and 20 is "
                + calculator.div(a, b));
```

Node JS Modules

```
// Default export
export default function add(a, b) {
    return a + b;
}
// Named export
export function sub(a, b) {
    return a - b;
}
```

Node JS Modules

export default:

- You are exporting the add function as the default export. This means that when the module is imported, this function can be imported without specifying its exact name.

export:

- You are also exporting the sub function as a **named export**. This requires that it be imported by its exact name in other modules.

- import addition from './yourModule.js'; // Default import
- import { sub } from './yourModule.js'; // Named import

Node JS Built-in Modules

- Node.js has many built-in modules that are part of the platform and comes with Node.js installation.
- These modules can be loaded into the program by using the require function.
- Syntax:
- **var module = require('module_name');**
- The require() function will return a JavaScript type depending on what the particular module returns.

Node JS Modules

```
console.clear()
const assert = require('assert');

let x = 4;
let y = 5;

try {

    // Checking condition
    assert(x == y);
}
catch {

    // Error output
    console.log(
        `${x} is not equal to ${y}`);
}
```

The assert module provides a set of assertion functions for verifying invariants. If the condition is true it will output nothing else an assertion error is given by the console.

Node JS Modules

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.end();
}).listen(8080);
```

This object has a property called "url" which holds the part of the url that comes after the domain name:

http://localhost:8080/summer

Node JS Modules

Given a URL, http://localhost:8080/pes.htm?city=Mangalore&year=2022
Write a code snippet to parse the URL and store the hostname, pathname, and the request parameters to a file named “requestlog.txt”.

Answer:

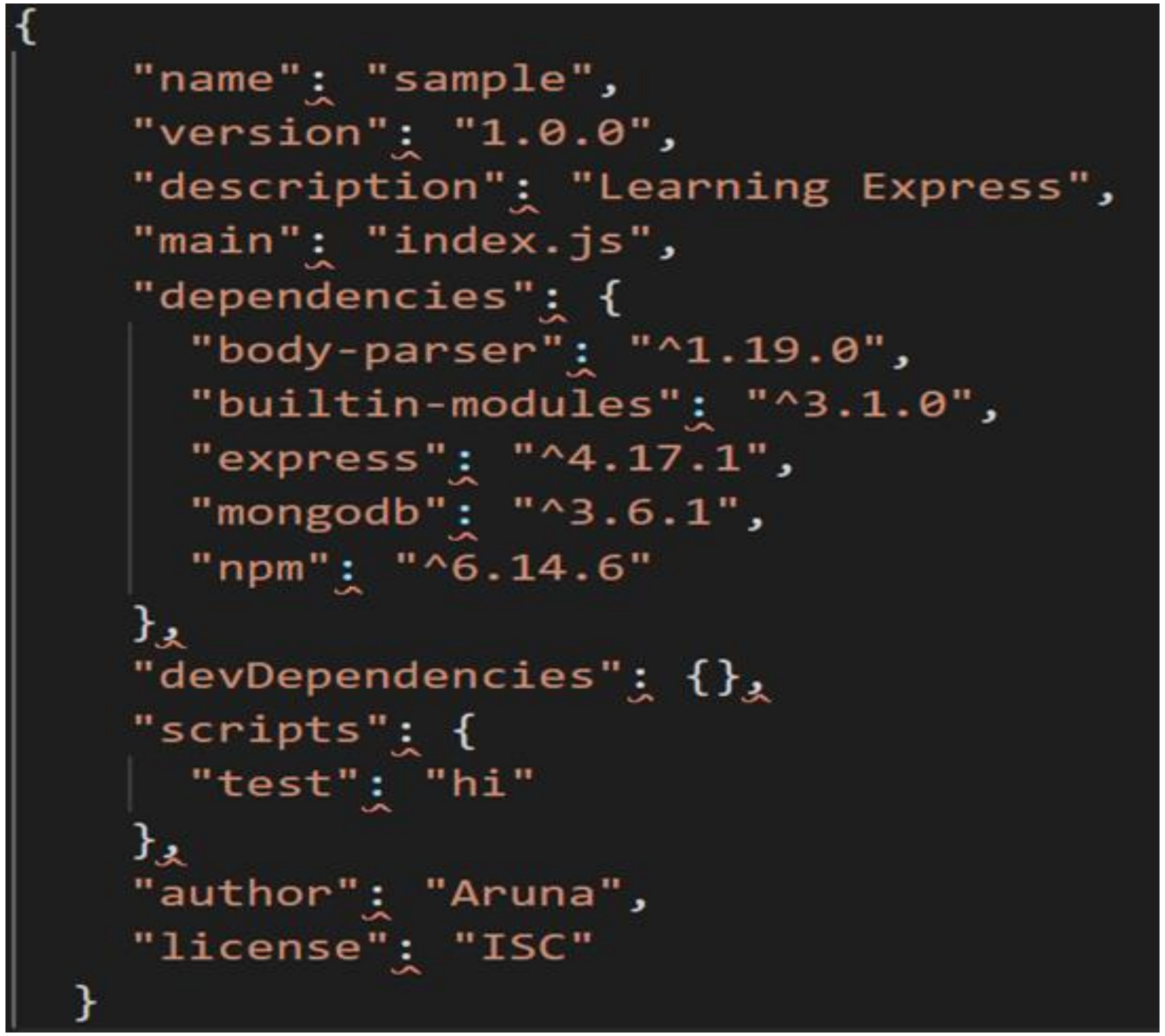
```
var myurl = url.parse(request.url)
var pathname = myurl.pathname;
var query = request.query;
var host = myurl.host;
fs.writeFile(“requestlog.txt”, host + pathname + query, function(err){
});
```

Package.json file

- All npm packages contain a file, usually in the project root, called package.json
- This file holds various metadata relevant to the project.
- It is used to give information to npm that allows it to identify the project as well as handle the project's dependencies.
- It can also contain other metadata such as a project description, the version of the project in a particular distribution, license information, even configuration data - all of which can be vital to both npm and to the end users of the package.
- The package.json file is normally located at the root directory of a Node.js project.

NODE JS

A Sample Package.json file



```
{
    "name": "sample",
    "version": "1.0.0",
    "description": "Learning Express",
    "main": "index.js",
    "dependencies": {
      "body-parser": "^1.19.0",
      "builtin-modules": "^3.1.0",
      "express": "^4.17.1",
      "mongodb": "^3.6.1",
      "npm": "^6.14.6"
    },
    "devDependencies": {},
    "scripts": {
      "test": "hi"
    },
    "author": "Aruna",
    "license": "ISC"
  }
```

Demo for Modules

- **Installation of validator module:**
- You can install this package by using this command.
  - *npm install validator*
- After installing validator module you can check your validator version in command prompt using the command.
  - *npm version validator*

NODE JS

Demo for Modules

```
const validator = require('validator')

// Check whether given email is valid or not
var email = 'testmail@gmail.com'
console.log(validator.isEmail(email))  // true
email = 'testmail@'
console.log(validator.isEmail(email))  // false

// Check whether string is in lowercase or not
var name = 'john'
console.log(validator.isLowercase(name))  // true
name = 'JOHN'
console.log(validator.isLowercase(name))  // false

// Check whether string is empty or not
var name = '    '
console.log(validator.isEmpty(name))  // true
name = 'Smith'
console.log(validator.isEmpty(name))  // false

// Other functions also available in  isBoolean(), isCurrency(), isDecimal(), isJSON(),
isJWT(), isFloat(), isCreditCard(), etc.
```

NODE JS

CHALK and nodemon Module

https://www.npmjs.com/package/chalk

- The Chalk Module Can Be Used With The Console's String Interpolation in Node. Js.
- The Chalk module allows you to add styles to your terminal output.

https://www.npmjs.com/package/nodemon

- **Nodemon** is a utility that will monitor for any changes in your source and automatically restart your server.
- Just use **nodemon** instead of node to run your code

THANK YOU

Prof. Vinay Joshi and Dr. Sarasvathi V

Department of
Computer Science and Engineering