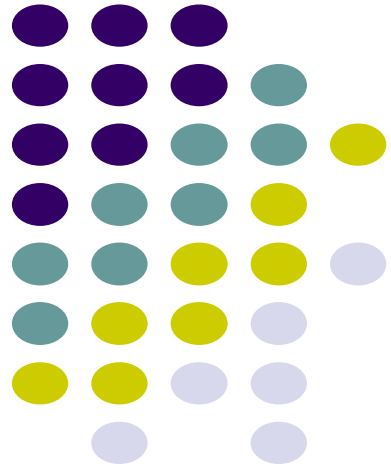
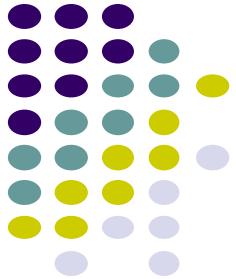
Node JS

Dr. Arul Xavier V M
Assistant Professor



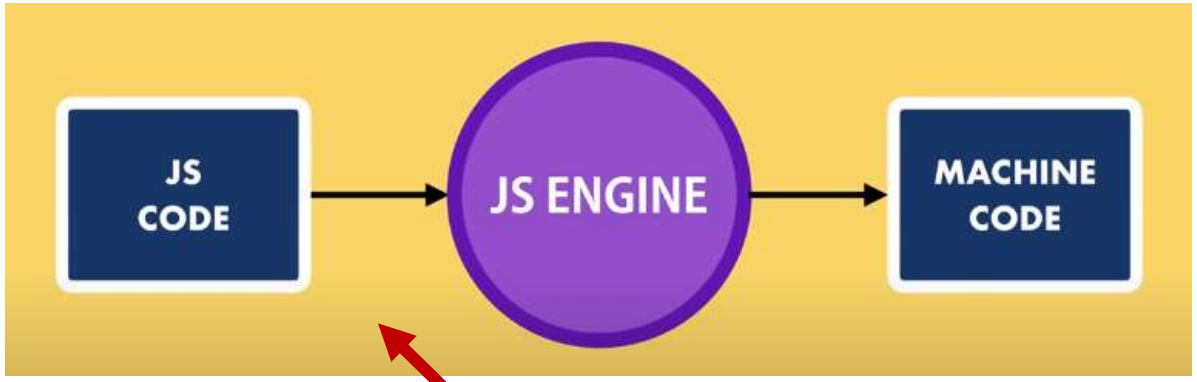
Introduction



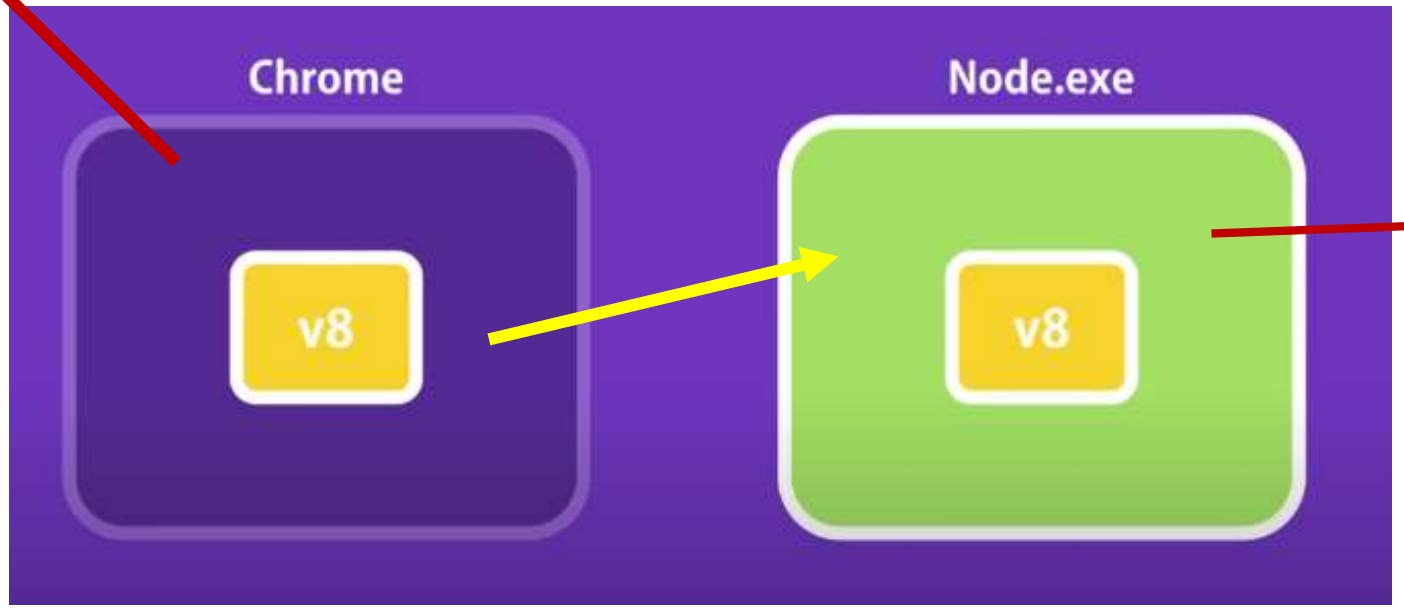
- Node.js allows you to run JavaScript outside the web browser.
- Used to create both client and server side applications(Full Stack Web Application Development)
- In 2009, Ryan Dahl created Node.js, used primarily to create scalable network applications.
- We often use Node JS to build back-end services such as API(Application Programming Interface.

Node JS – Run Time Environment

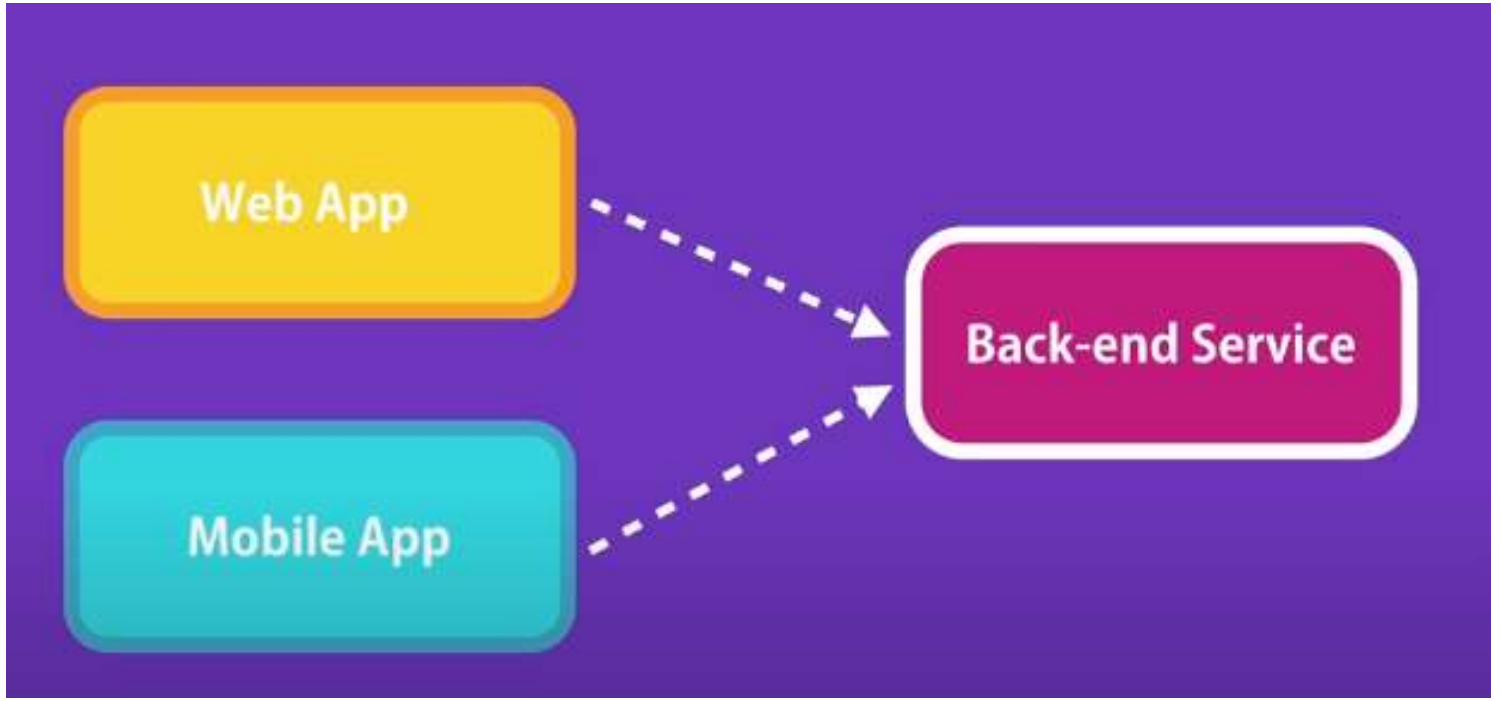
- Node.js is an open source run time environment for executing JavaScript.



Ryan Dahl

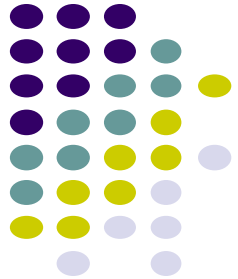


Node JS Back End Service



Node JS is a Highly Scalable, data-intensive and real-time apps.

Special about Node JS

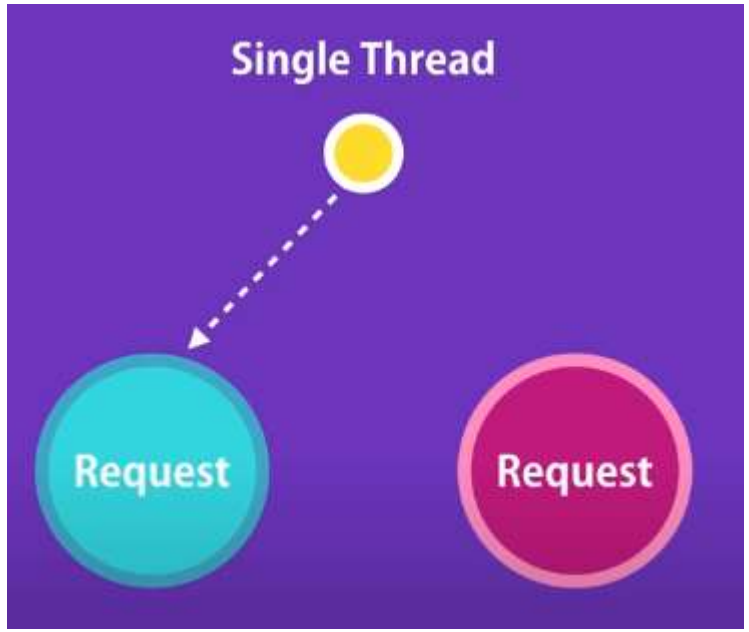


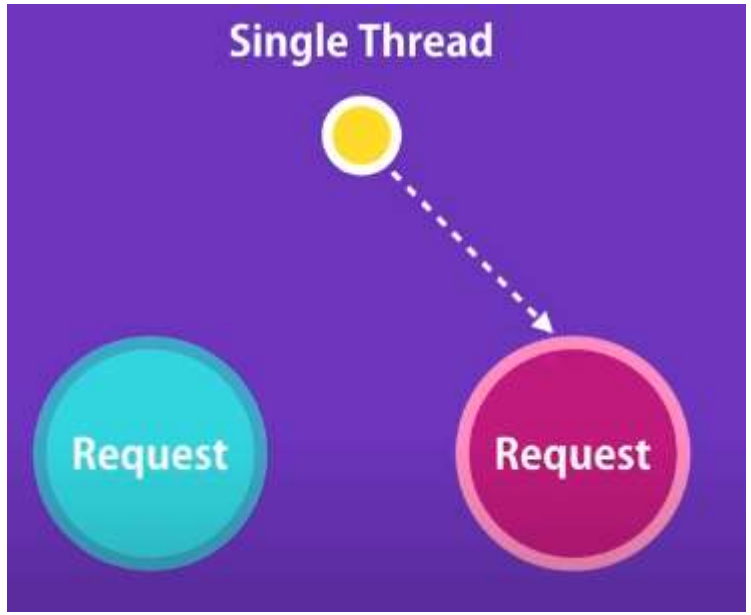
- Suitable for super fast highly scalable back end services.
- Great for prototyping and agile development.
- Purely based on JavaScript

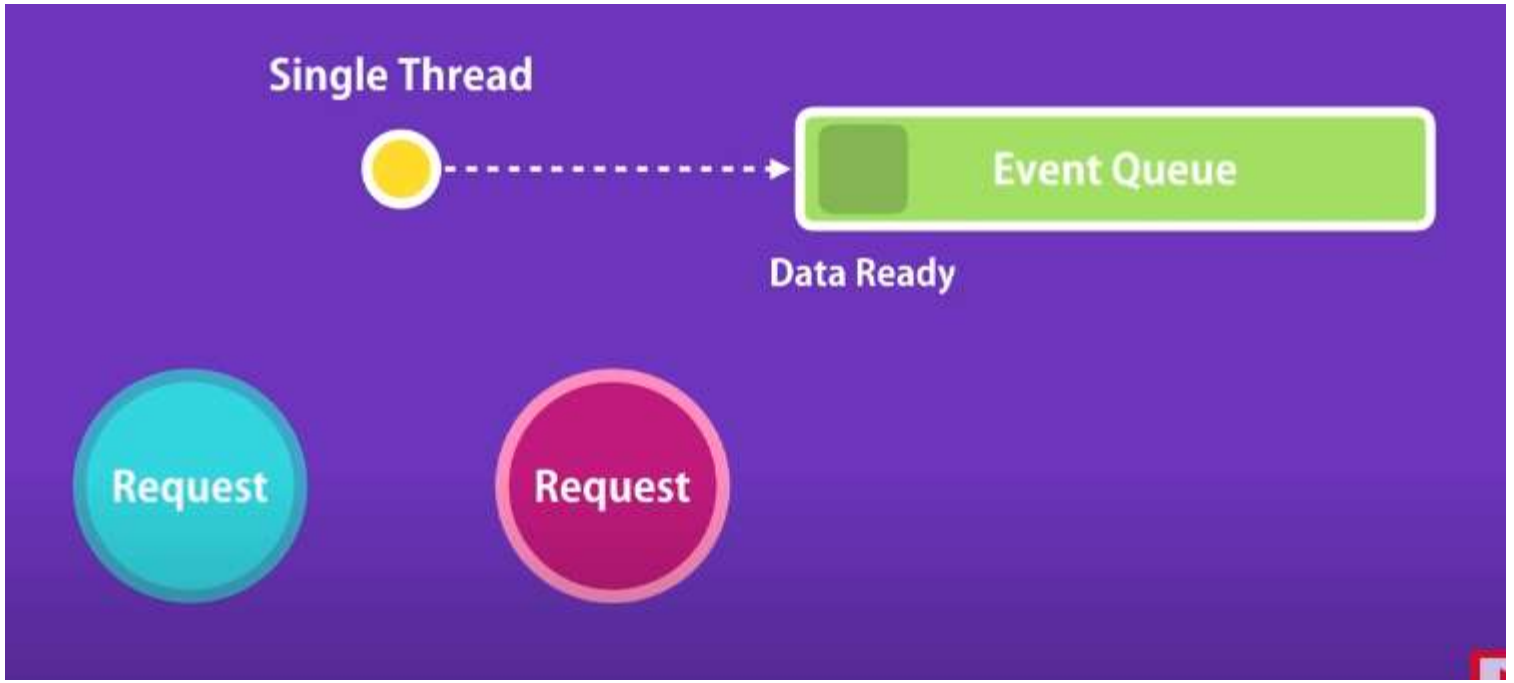
Node JS

- Highly scalable
  - Non – Blocking Asychronous
  - Node JS is single threaded application environment.
  - Node JS best suitable for creating websites and web application which contain less CPU intensive tasks.
  - Example
    - E-Commerce, E-Governance, Websites, etc..
  - Drawbacks
    - Due to single threaded model, node js is not suitable for CPU intensive tasks such as image processing, machine learning, deep learning and etc.

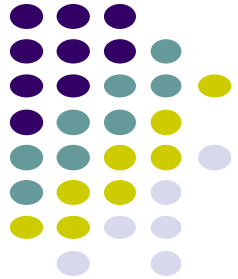
Non-Blocking Asynchronous







How Node JS works?



- Install Node JS Software
  - https://nodejs.org/en/
- Create javascript program with an extension .js
  - Example: app.js
- Execute the javascript program using the command ‘node’
  - node app.js

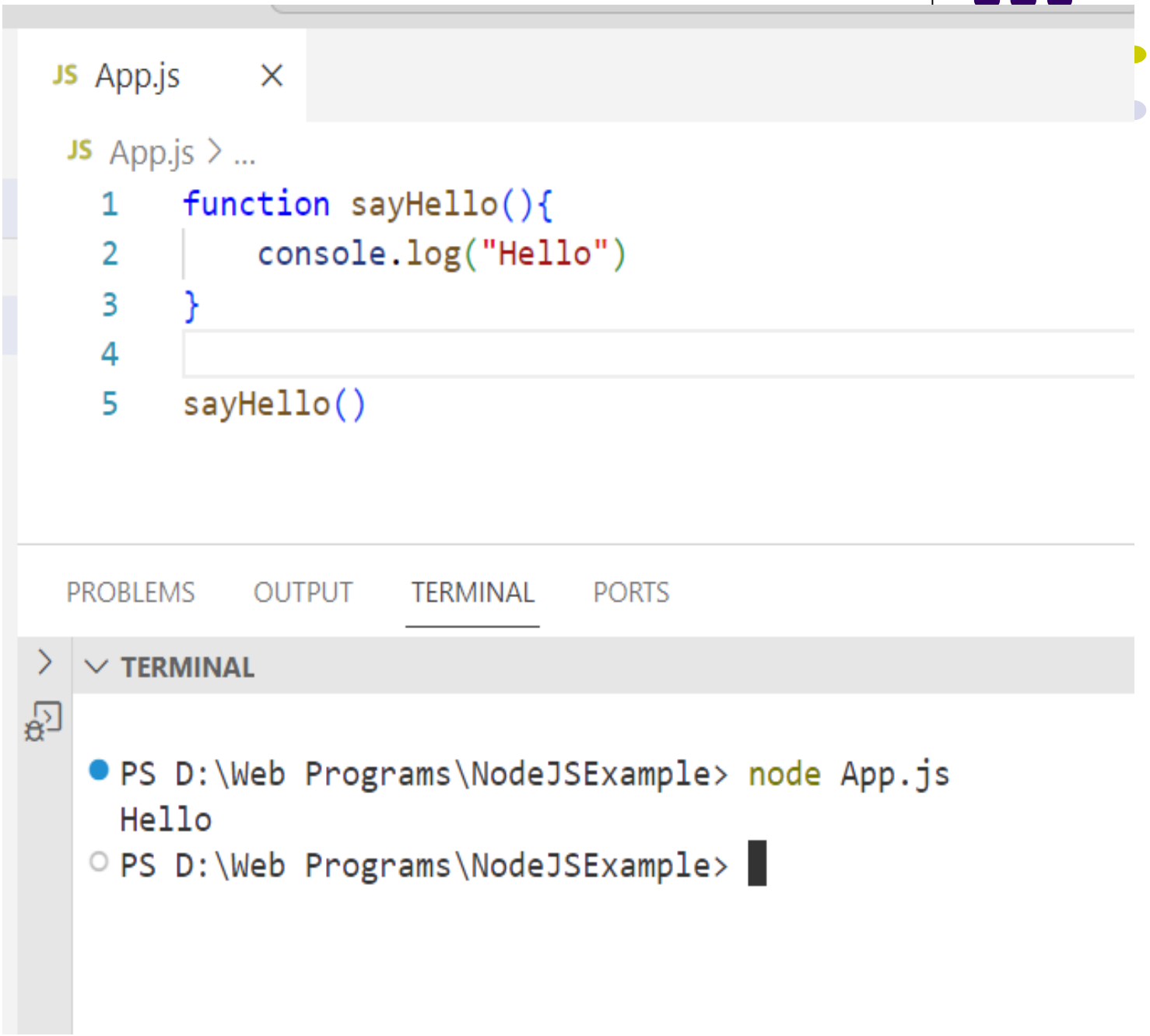
Node JS is similar to Java, Python, C++ environments

Example Node JS

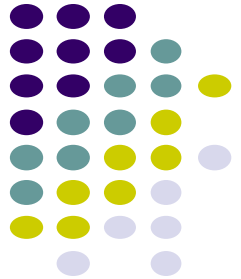
```
function sayHello(){
    console.log("Hello")
}

sayHello()
```

To Execute:

node App.js

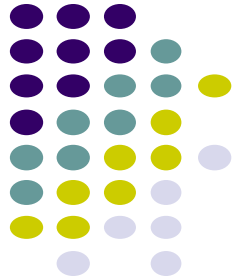


Node Module System



- Module is a javascript file, which contains data and functionalities.
- Types
  - Built-in modules
    - http - HTTP Protocol Services(request and response)
      - used to create server.
    - fs - File System module
    - os – Operating System module
  - User defined modules

Creating User Defined Module



Module 1

app.js

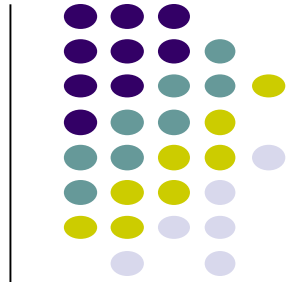
```
const calc = require('./test');
console.log(calc.add(10,20))
console.log(calc.multiply(3,5))
```

Module 2

test.js

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

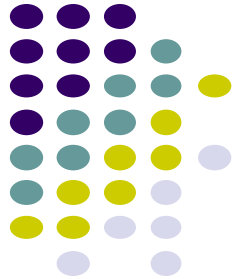
http module



- Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).
- To include the HTTP module, use the require() method:

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

Node.js as a Web Server



- The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.
- Use the createServer() method to create an HTTP server:

```
const http = require('http');
const server=http.createServer(function (req, res) {
  res.writeHead('200',{'Content-Type':'text/html'});
  res.write('Hello World!'); //write a response to the client
  res.end();                 //end the response
});
server.listen(5000,function(){
    console.log('Server is listening at 5000');
});
```

req and res object

- req – user defined object parameter used to read incoming messages in http request.
  - Access the url of HTTP Request

```
res.write(req.url);
```

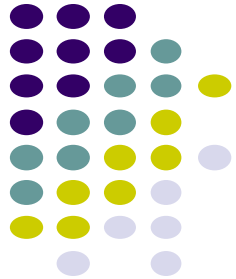
  - Access the request method type (POST or GET)

```
res.write(req.method);
```

- res – user defined object parameter used to send response data from server to browser(client)
  - Send response content in text or html to client browser

```
res.write("<h1>Welcome to Node JS</h1>");
```

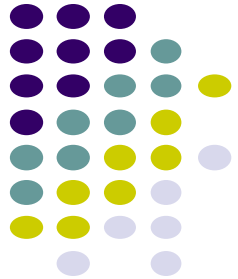
Add an HTTP Header



- If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type:

```
response.writeHead(200, {'Content-Type': 'text/html'});
```

Display Output of HTML file using HTTP res object

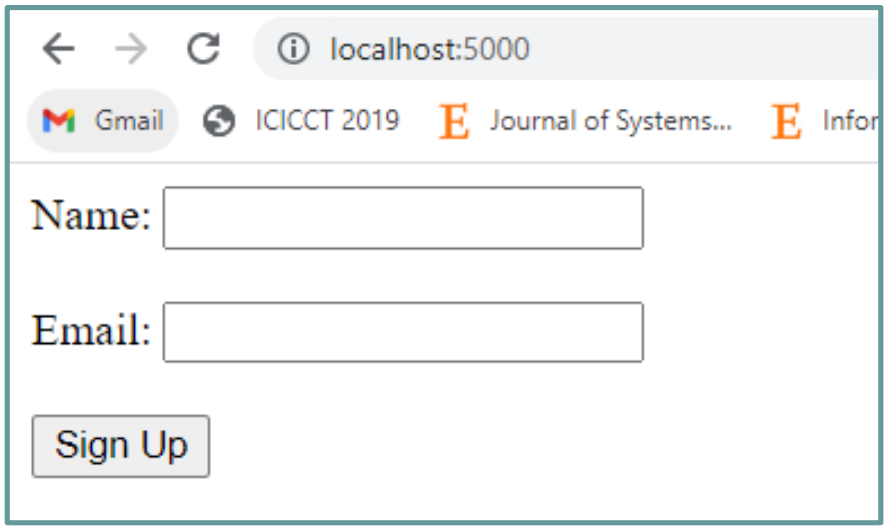


- res object can be used to send HTML page content as response to the HTTP Response

```
const fs = require('fs');
fs.createReadStream('signup.html').pipe(res);
```

```
const http = require('http');
const fs = require('fs');
const server=http.createServer(function (req, res) {
  res.writeHead('200',{'Content-Type':'text/html'});
  fs.createReadStream('signup.html').pipe(res);
});
server.listen(5000,function(){
    console.log('Server is listening at 5000');
});
```

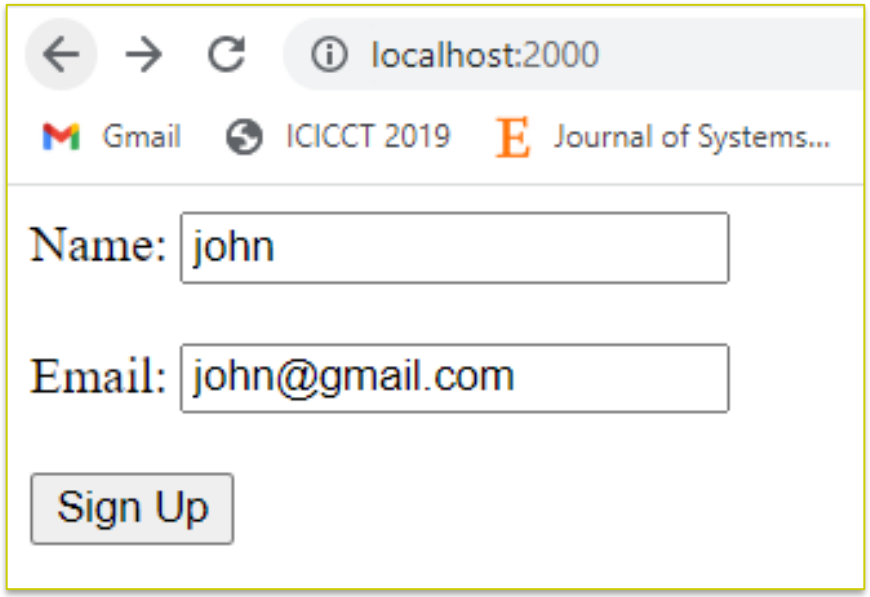
app.js



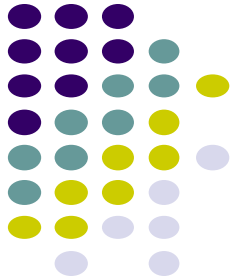
signup.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta name="viewport" content="width=device-
width, initial-scale=1.0">
    <title>Sign Up</title>
</head>
<body>
    <form action="/signup" method="post">
        Name: <input type="text"><br><br>
        Email: <input type="email"><br><br>
        <button type="submit">Sign Up</button>
    </form>
</body>
</html>
```

Reading HTML Form Data and Display



Create Sign Up Page



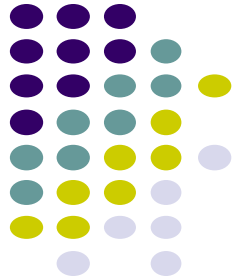
signup.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Sign Up</title>
</head>
<body>
    <form action="/signup" method="post">
        Name: <input type="text" name="username"><br><br>
        Email: <input type="email" name="useremail"><br><br>
        <button type="submit">Sign Up</button>
    </form>
</body>
</html>
```

app.js

```
const http = require('http');
const fs = require('fs');
const server = http.createServer(function (req, res) {
    if(req.url==='/'){
      res.writeHead(200,{"Content-Type": "text/html"});
      fs.createReadStream('signup.html').pipe(res);
    }
    else if(req.url ==='/signup' && req.method == 'POST'){
      var rawData = '';
      req.on('data',function(data){
         rawData += data;
      })
      req.on('end',function(){
          var inputdata = new URLSearchParams(rawData);
          res.writeHead(200,{"Content-Type": "text/html"});
          res.write('User Name: ' +inputdata.get('username') + '<br>');
          res.write('User Email: ' +inputdata.get('useremail') + '<br>');
          res.end();
      });
    }
  });
  server.listen(2000,function(){
    console.log('listening at 2000')
  });
```

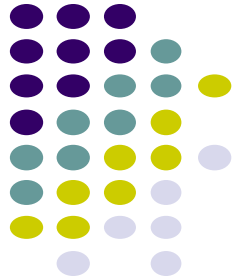
req.on method



- When receiving a POST or GET request, body data can be accessed via req object through ReadableStream.
- We can grab the data right out of the stream by listening to the stream's 'data' and 'end' events.
- The ‘data’ and ‘end’ event are handled by req.on method.

```
else if(req.url ==='/signup' && req.method == 'POST'){
      var rawData = '';
      req.on('data',function(data){
         rawData += data;
      })
      req.on('end',function(){
          var inputdata = new URLSearchParams(rawData);
          res.writeHead(200,{"Content-Type": "text/html"});
          res.write('User Name: ' +inputdata.get('username') + '<br>');
          res.write('User Email: ' +inputdata.get('useremail') + '<br>');
          res.end();
      });
   }
```

Node JS File Handling

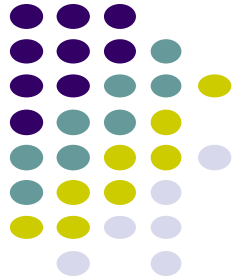


- The Node.js file system module allows you to work with the file system on your computer.
- To include the File System module, use the require() method:

```
var fs = require('fs');
```

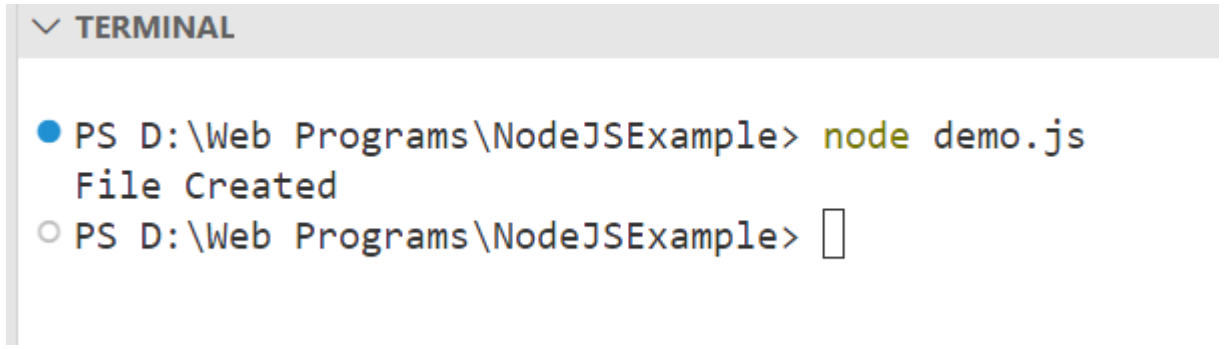
- Common use for the File System module:
  - Create File
  - Read files
  - Write files
  - Update files
  - Delete files
  - Rename files

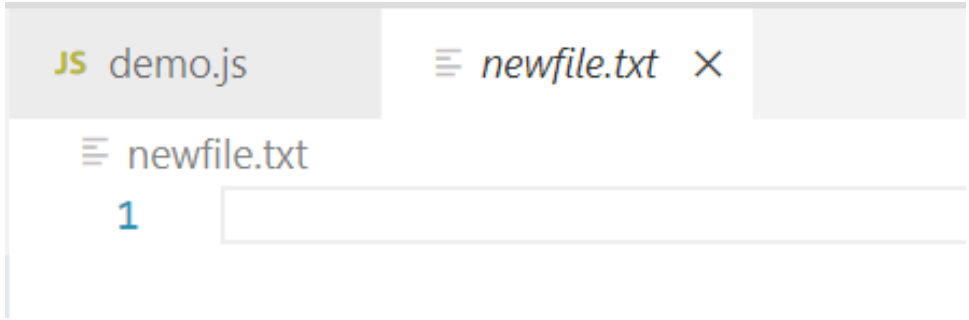
Node JS File Handling



- Create a New File
  - fs.open(filename, mode, callback) used to create a new file with mode of reading, writing or appending

```
const fs = require('fs')
fs.open('newfile.txt','w', function(err){
   if(err) throw err;
   console.log('File Created')
})
```





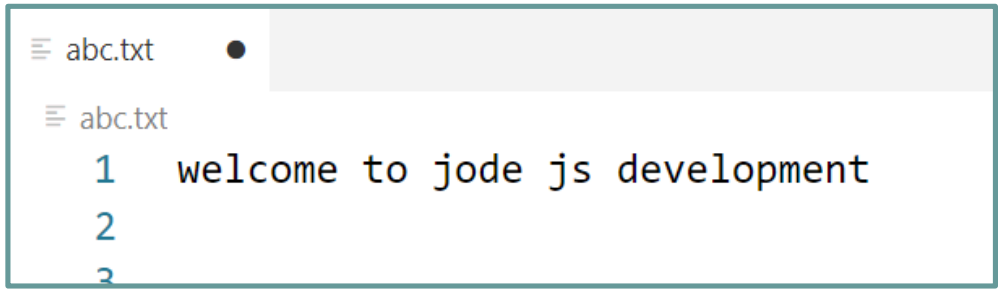
Node JS File Handling

- Read Files
  - readFile() used for reading file contents

To read the file and display its contents

```
const fs = require('fs');
fs.readFile('abc.txt',function(error, data){
    console.log(data.toString());
})
```

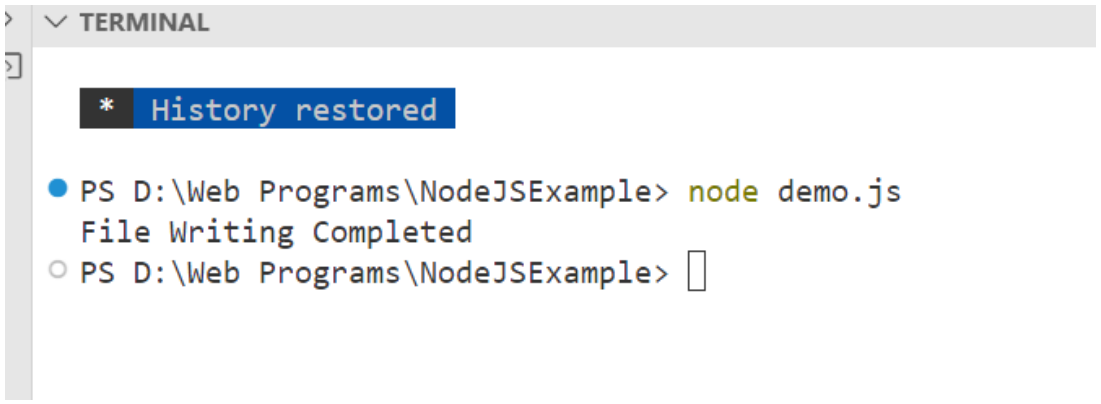
Consider this file in current directory

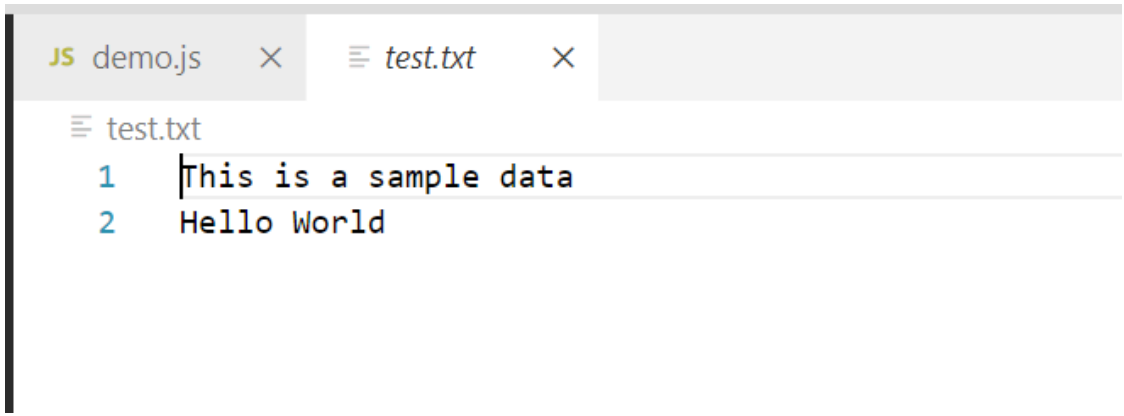


Node JS File Handling

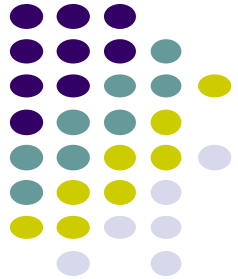
- Write data to File
  - writeFile(filename, data, callback function) used to write new data in an existing file. If the file is not exists, this method creates a new file. However, if the file is exists it overwrites the existing file content.

```
const fs = require('fs')
fs.writeFile('test.txt','This is a sample data\nHello World', function(err){
   if(err) throw err;
   console.log('File Writing Completed')
})
```





Node JS File Handling



- Update data in the existing File
  - appendFile(filename, data, callback) is used to asynchronously append the given data to a file. A new file is created if it does not exist.

```
const fs = require('fs')
fs.appendFile('test.txt','New Data \n This is new', function(err){
    if(err) throw err;
    console.log('File Appending Completed')
})
```

```
TERMINAL

PS D:\Web Programs\NodeJSExample> node demo.js
File Appending Completed
PS D:\Web Programs\NodeJSExample>
```

JS demo.js | test.txt

test.txt

```
1   This is a sample data
2   Hello WorldNew Data
3    This is new
```

Node JS File Handling

- Rename the file
  - rename(oldfilename, newfilename, callback) used to rename the existing file name.

```
const fs = require('fs')
fs.rename('test.txt','newfile.txt', function(err){
    if(err) throw err;
    console.log('Filename Changed')
})
```

TERMINAL

```
PS D:\Web Programs\NodeJSExample> node demo.js
Filename Changed
PS D:\Web Programs\NodeJSExample>
```

newfile.txt

```
1   This is a sample data
2   Hello WorldNew Data
3    This is new
4
5
```

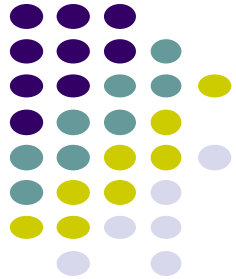
Node JS File Handling

- Delete Files
  - unlink(filename, callback) used to delete the existing file.

```
const fs = require('fs')
fs.unlink('newfile.txt', function(err){
    if(err) throw err;
    console.log('File Deleted')
})
```

```
TERMINAL

PS D:\Web Programs\NodeJSExample> node demo.js
File Deleted
PS D:\Web Programs\NodeJSExample>
```

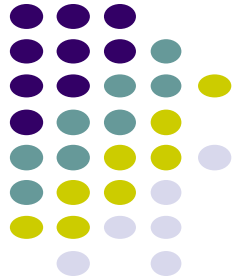
Create a new folder



- Use fs.mkdir() to create a new folder.

```
const fs = require('fs')

fs.mkdir('songs',function(err){
    console.log('Folder Created')
})
```

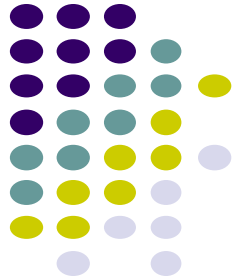
Delete a folder



- Use fs.rmdir() function to delete the given folder.

```
const fs = require('fs')

fs.rmdir('songs',function(err){
    console.log('deleted')
})
```

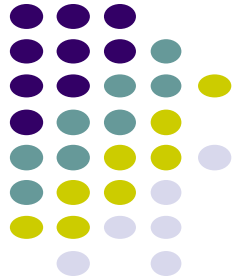
Read the content of a directory



- Use fs.readdir() to read the contents of a directory.
  - fs.readdir(path, options, callback)

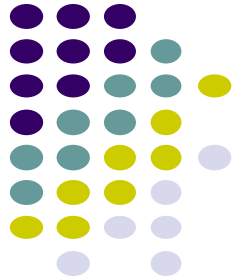
```
const fs = require('fs')

fs.readdir('D:/Web Example',function(err,files){
    if(err) throw err
    files.forEach((filename)=>{
        console.log(filename)
    })
})
```

Event Loop



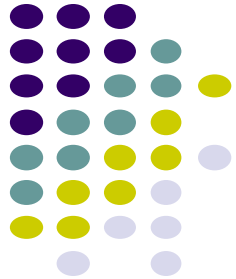
- Node.js is a single-threaded event-loop platform that is capable of running non-blocking, asynchronous programming.
- These functionalities of Node.js make it memory efficient.

Features of Event Loop



- An event loop is an endless loop, which waits for tasks, executes them, and then sleeps until it receives more tasks.
- The event loop executes tasks from the event queue only when the call stack is empty i.e. there is no ongoing task.
- The event loop allows us to use callbacks and promises.
- The event loop executes the tasks starting from the oldest first.

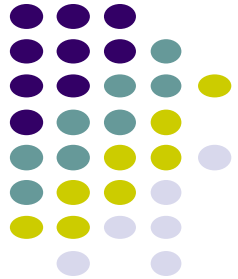
Example



```
console.log("This is the first statement");

setTimeout(function(){
    console.log("This is the second statement");
}, 1000);

console.log("This is the third statement");
```

The first console log statement is pushed to the call stack, and “This is the first statement” is logged on the console, and the task is popped from the stack. Next, the setTimeout is pushed to the queue and the task is sent to the Operating system and the timer is set for the task. This task is then popped from the stack. Next, the third console log statement is pushed to the call stack, and “This is the third statement” is logged on the console and the task is popped from the stack.
When the timer set by the setTimeout function (in this case 1000 ms) runs out, the callback is sent to the event queue. The event loop on finding the call stack empty takes the task at the top of the event queue and sends it to the call stack. The callback function for the setTimeout function runs the instruction and “This is the second statement” is logged on the console and the task is popped from the stack.

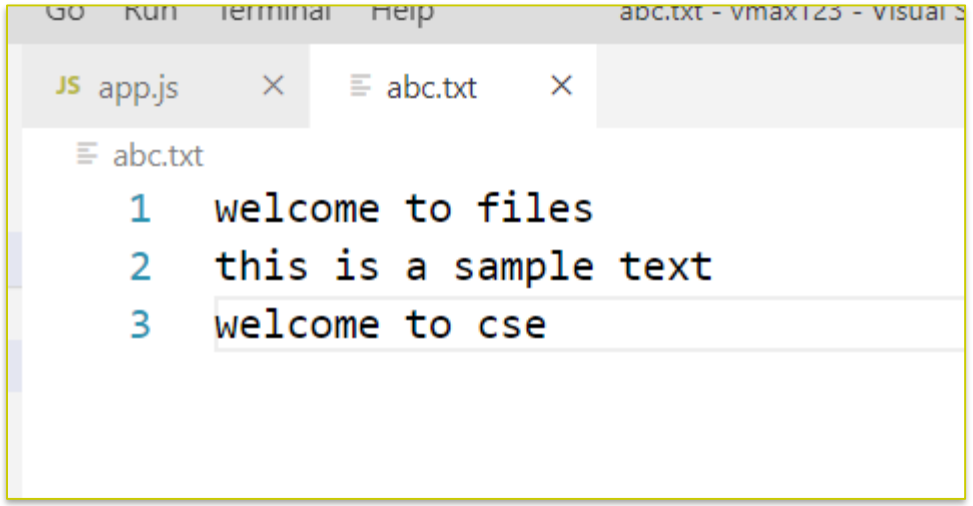
Asynchronous Code

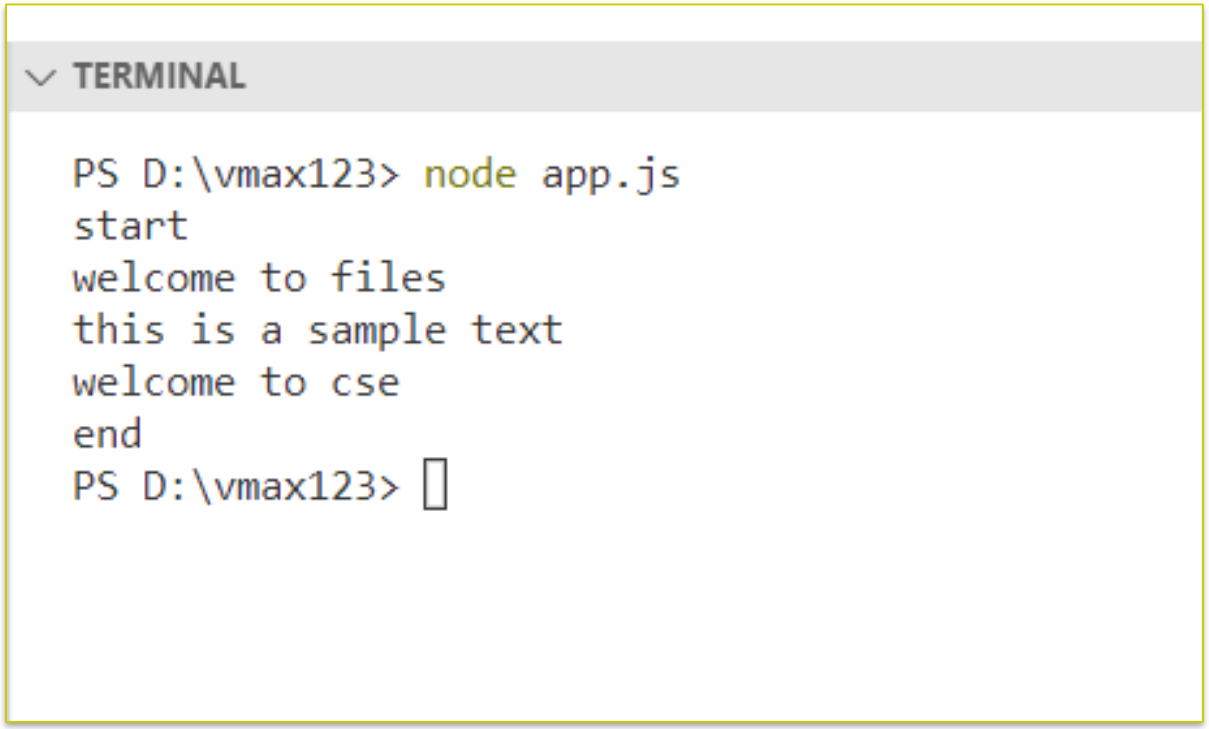


- Asynchronous means that things can happen independently of the main program flow.
- Blocking methods execute synchronously and non-blocking methods execute asynchronously.
- Example
  - File Reading functions
    - readFileSync() – synchrounous - blocking call
    - readFile() – asynchronous – non-blocking call

Synchronous Code Example

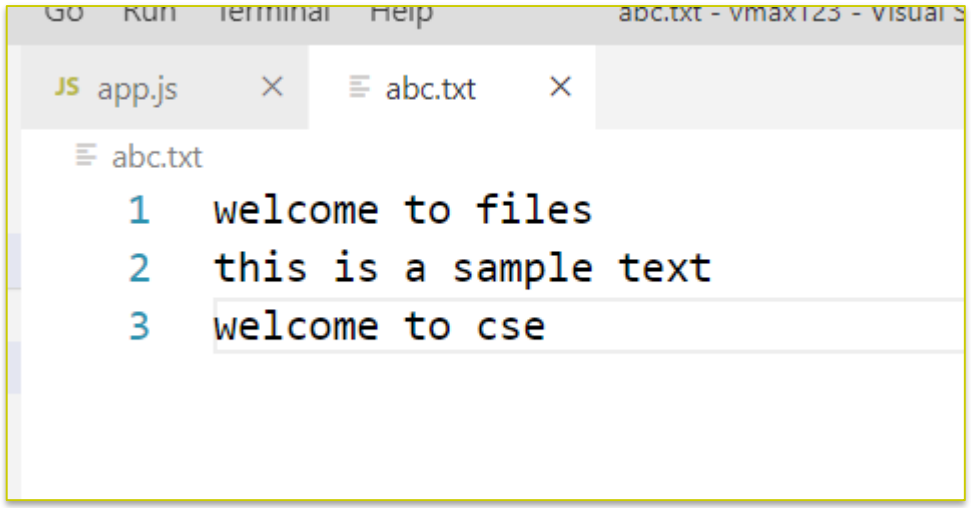
```
const fs = require('fs')
console.log('start')
const data = fs.readFileSync('abc.txt');
console.log(data.toString());
console.log('end')
```

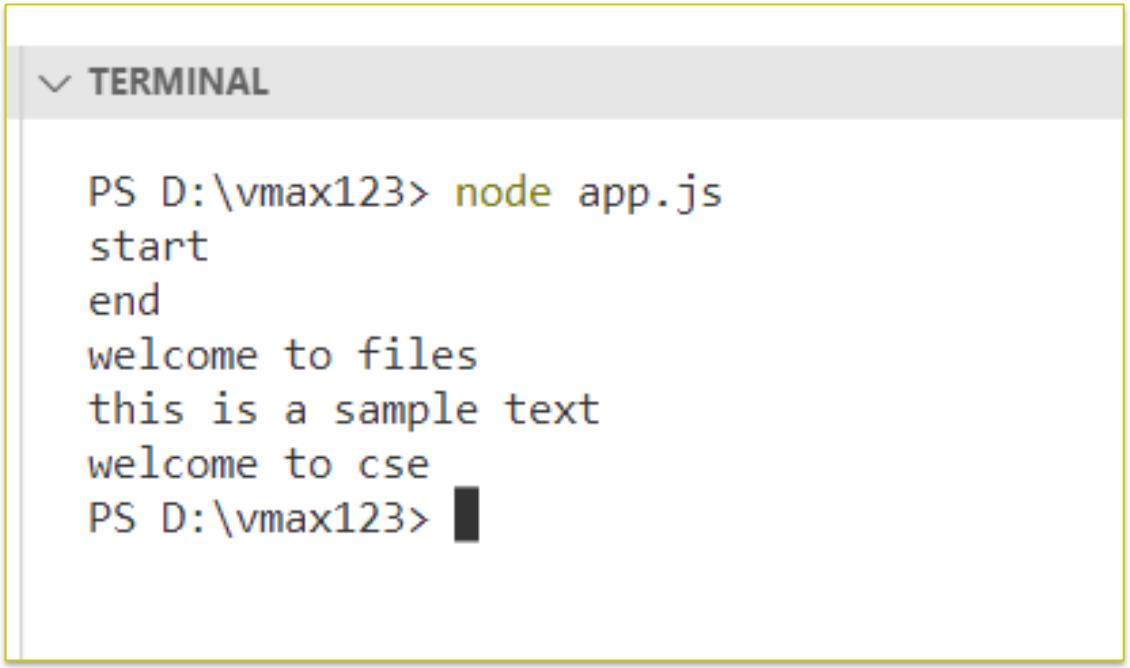




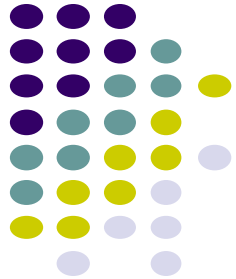
Asynchronous Code Example

```
const fs = require('fs')
console.log('start')
fs.readFile('abc.txt', function(err, data){
    if (err) throw err;
    console.log(data.toString());
  });
console.log('end')
```





Callback Functions



- A callback function is called at the completion of a given task.
- Node makes heavy use of callbacks.
- All the APIs of Node are written in such a way that they support callbacks.

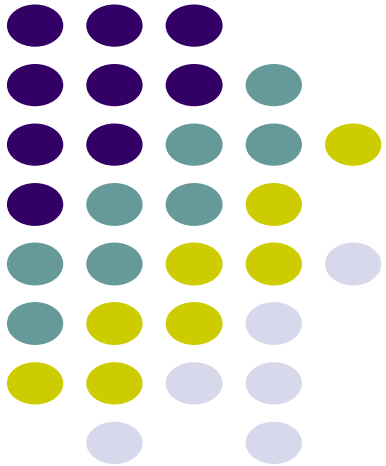
For example, a function to read a file may start reading file and return the control to the execution environment immediately so that the next instruction can be executed. Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as a parameter. So there is no blocking or wait for File I/O. This makes Node.js highly scalable, as it can process a high number of requests without waiting for any function to return results.

Callback Function example

```
const fs = require('fs')
console.log('start')
fs.readFile('abc.txt', function(err, data){
    if (err) throw err;
    console.log(data.toString());
  });
console.log('end')
```

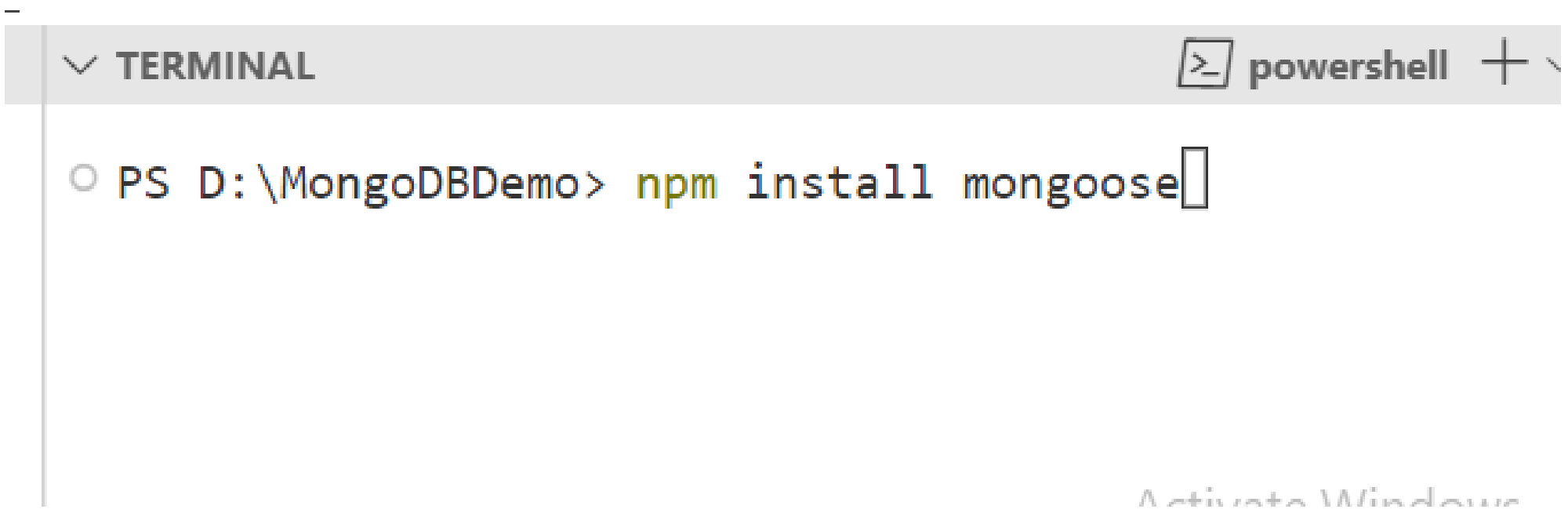
Here, the function(err,data){} is said to be callback function which will be executed after reading data from the file.

Node JS Program to save data in Database

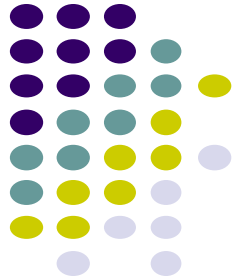


Install the mongoose module

PS D:\MongoDBDemo> npm install mongoose



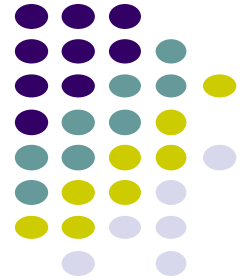
Create a HTML Form



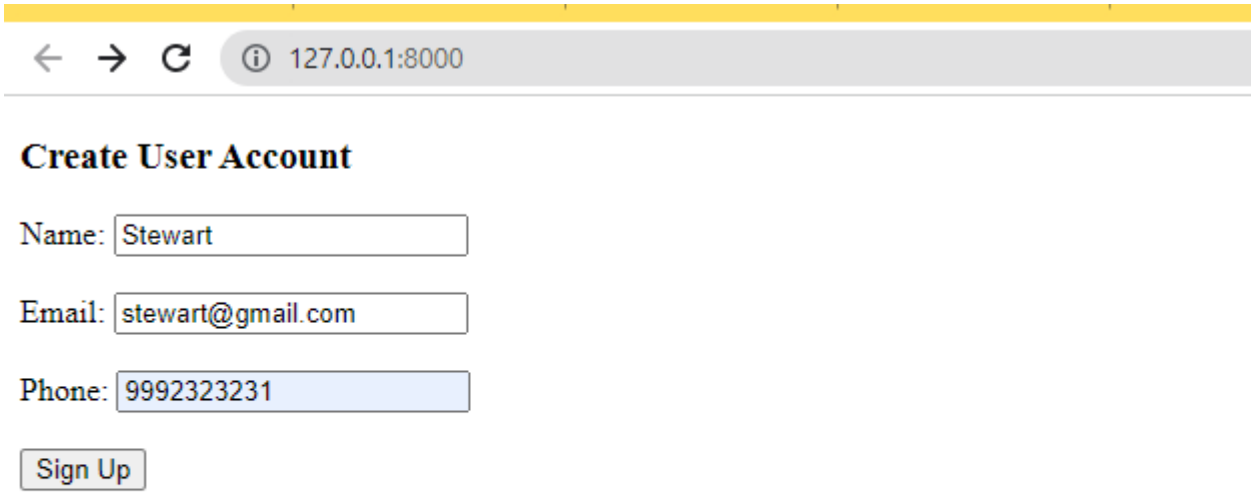
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sign Up Page</title>
</head>
<body>
    <form action="/signup" method="POST">
        <h3>Create User Account</h3>
        Name: <input type="text" name="name" required><br><br>
        Email: <input type="email" name="email" required><br><br>
        Phone: <input type="text" name="phone" required><br><br>
        <button type="submit">Sign Up</button>
    </form>
    <br><br>
    </body>
</html>
```

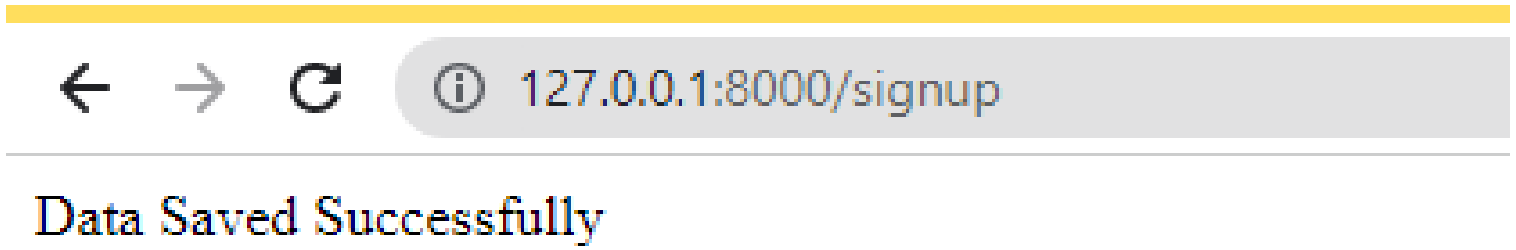
```
const http = require('http');
const fs = require('fs');

//To include mongoose module in node js program
const mongoose = require('mongoose');

//Connecting to the mongodb database
mongoose.connect('mongodb://127.0.0.1:27017/college')
        .then(function(){
            console.log('DB Connected')
        })

//Defining the Structure of mongodb document
const studentSchema = new mongoose.Schema({name:String, email:String,phone:String});

//Create collection model
const studentmodel = mongoose.model('students',studentSchema);
```

```
const server = http.createServer(function(req,res){
      if(req.url === '/'){
         res.writeHead('200',{'Content-Type':'text/html'});
         fs.createReadStream('signup.html').pipe(res);
      }
      else if(req.url=== '/signup' && req.method === 'POST'){
           var rawdata = '';
           req.on('data',function(data){
               rawdata += data;
           })
           req.on('end',function(){
               var formdata = new URLSearchParams(rawdata);
               res.writeHead('200',{'Content-Type':'text/html'});
               studentmodel.create({name:formdata.get('name'),
                                    email:formdata.get('email'),
                                    phone:formdata.get('phone')
                                   })
               res.write('Data Saved Successfully');
               res.end();
           })
      }
);
```

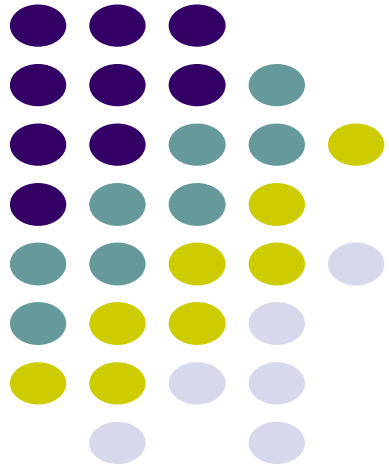


```
server.listen('8000',function(){
    console.log('Server started at port
http://127.0.0.1:8000');
})
```

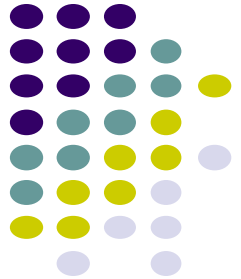

127.0.0.1:8000
Create User Account
Name: Stewart
Email: stewart@gmail.com
Phone: 9992323231
Sign Up


127.0.0.1:8000/signup
Data Saved Successfully

Node JS Program to Display all data from Database

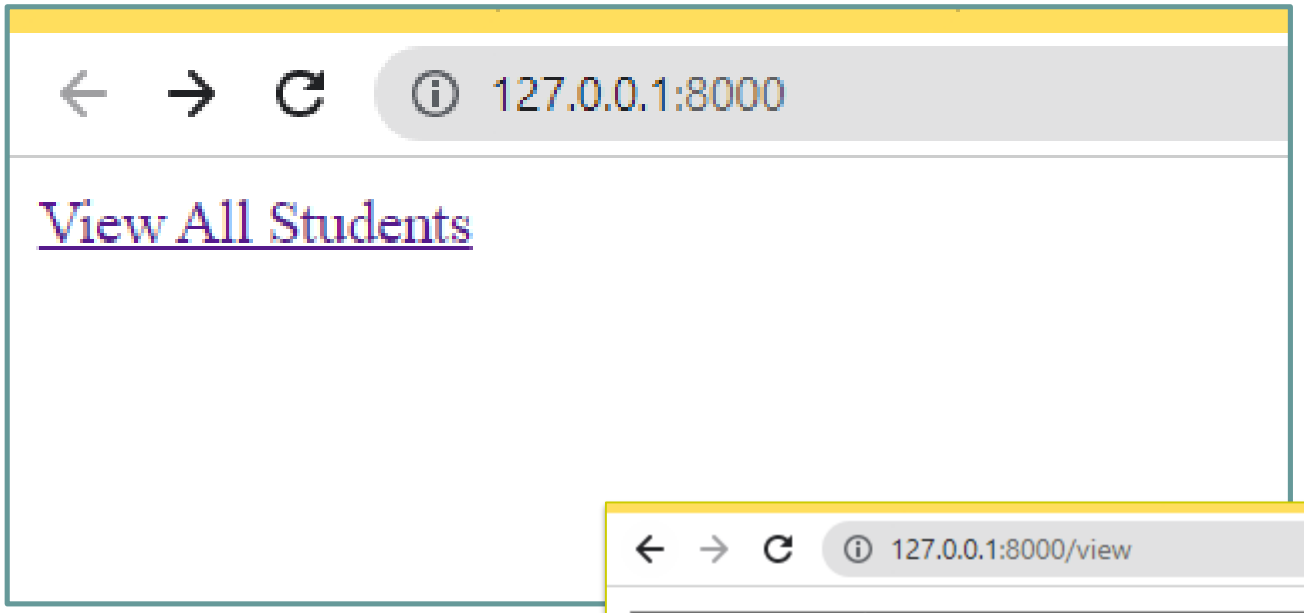


Node JS program display all data



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

//To include mongoose module in node js program
const mongoose = require('mongoose');

//Connecting to the mongodb database
mongoose.connect('mongodb://127.0.0.1:27017/college')
        .then(function(){
            console.log('DB Connected')
        })

//Defining the Structure of mongodb document
const studentSchema = new mongoose.Schema({name:String, email:String,phone:String});

//Create collection model
const studentmodel = mongoose.model('students',studentSchema);
```

```
const server = http.createServer(function(req,res){
        res.writeHead('200',{'Content-Type':'text/html'});
          //To fetch all data from mongodb database collection
        studentmodel.find().then(function(students){
                res.write("<table border=1 cellspacing=0 width=400>");
                res.write("<tr><th>Name</th><th>Email</th><th>Phone</th></tr>");
                students.forEach(student=>{
                    res.write("<tr>");
                    res.write("<td>"+student.name+"</td>");
                    res.write("<td>"+student.email+"</td>");
                    res.write("<td>"+student.phone+"</td>");
                    res.write("</tr>");
                })
                res.end();
        })
})
server.listen('8000',function(){
    console.log('Server started at port http://127.0.0.1:8000');
})
```



Name	Email	Phone
John	john@gmail.com	8239232323
Raj	raj@gmail.com	9923239232
vmax	vmax@gmail.com	892329323
reban	reban@gmail.com	8923023232
Stewart	stewart@gmail.com	9992323231