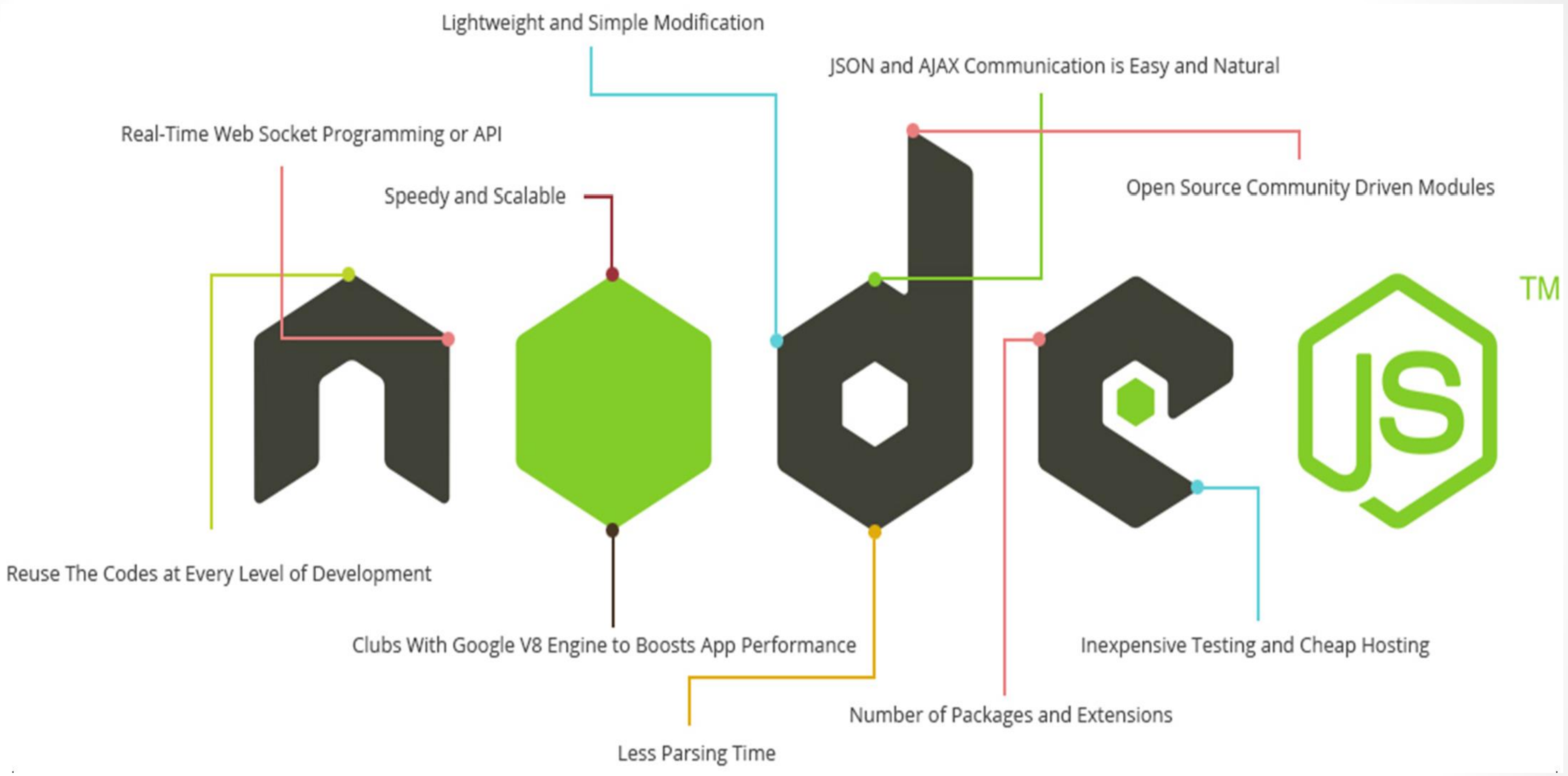
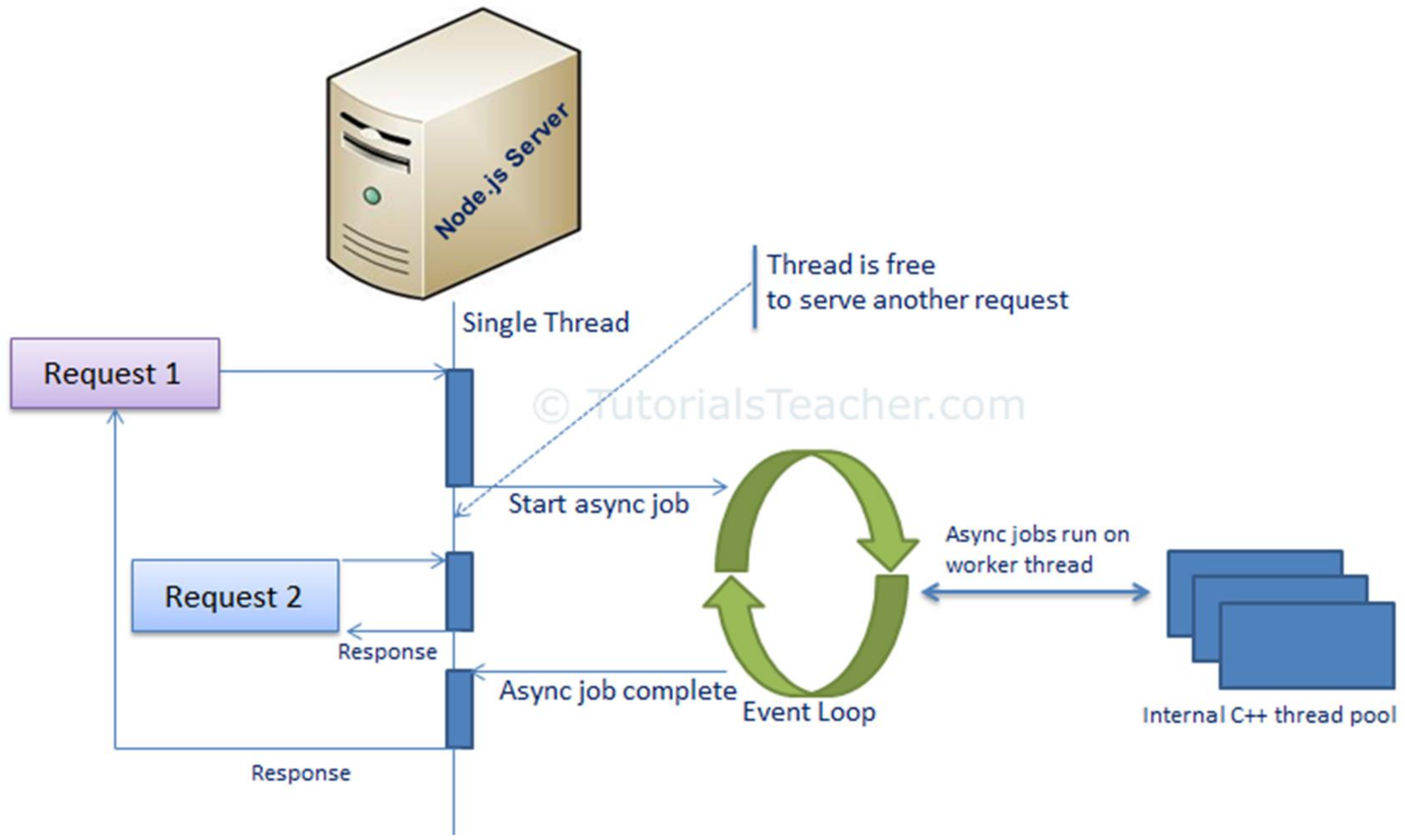
Node JS


Lightweight and Simple Modification
JSON and AJAX Communication is Easy and Natural
Real-Time Web Socket Programming or API
Speedy and Scalable
Open Source Community Driven Modules
Reuse The Codes at Every Level of Development
Clubs With Google V8 Engine to Boosts App Performance
Inexpensive Testing and Cheap Hosting
Number of Packages and Extensions
Less Parsing Time


Node.js Server
Single Thread
Thread is free
to serve another request
Request 1
© TutorialsTeacher.com
Start async job
Async jobs run on
worker thread
Request 2
Response
Async job complete
Event Loop
Internal C++ thread pool
Response

Introduction

- Node.js is a very powerful JavaScript-based framework/platform built on Google Chrome's JavaScript V8 Engine.
- Node.js allows you to run JavaScript on the server.
- Node.js was developed by Ryan Dahl in 2009 and its latest recommended version is 12.18.4
- It is used to develop I/O intensive web applications like video streaming sites, single-page applications, networking applications, and other web applications.
- Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.
- Node.js is open source, completely free, and used by thousands of developers around the world.

Node.js = Runtime Environment + JavaScript Library

Why Node.js?

- **Node.js uses asynchronous programming!**
- A common task for a web server can be to open a file on the server and return the content to the client.

Here is how PHP or ASP handles a file request:

- Sends the task to the computer's file system.
- Waits while the file system opens and reads the file.
- Returns the content to the client.
- Ready to handle the next request.

Here is how Node.js handles a file request:

- Sends the task to the computer's file system.
- Ready to handle the next request.
- When the file system has opened and read the file, the server returns the content to the client.
- Node.js eliminates the waiting, and simply continues with the next request.
- Node.js runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.

Who uses Node.js?

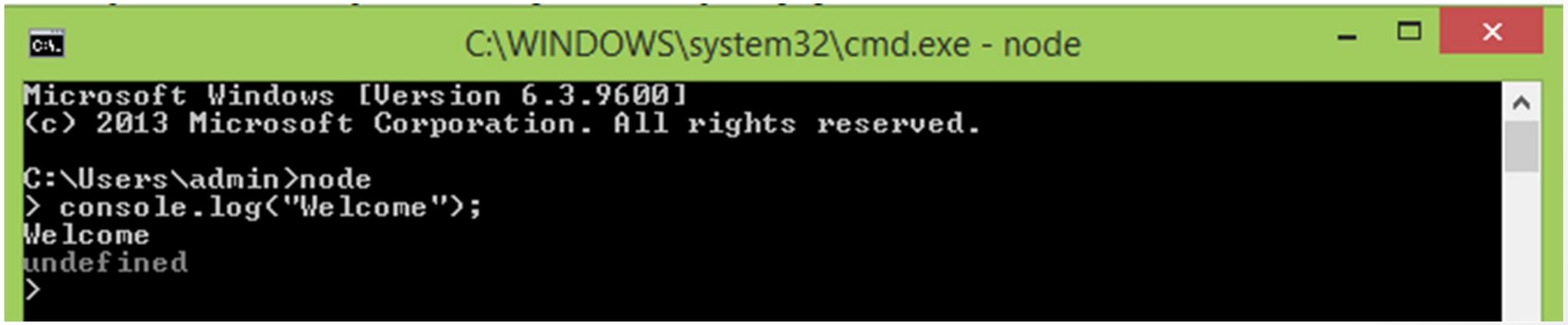
- Following is the link on github wiki containing an exhaustive list of projects, application and companies which are using Node.js. This list includes eBay, General Electric, GoDaddy, Microsoft, PayPal, Uber, Wikipins, Yahoo!, and Yammer to name a few.

What Can Node.js Do?

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database

What is a Node.js File?

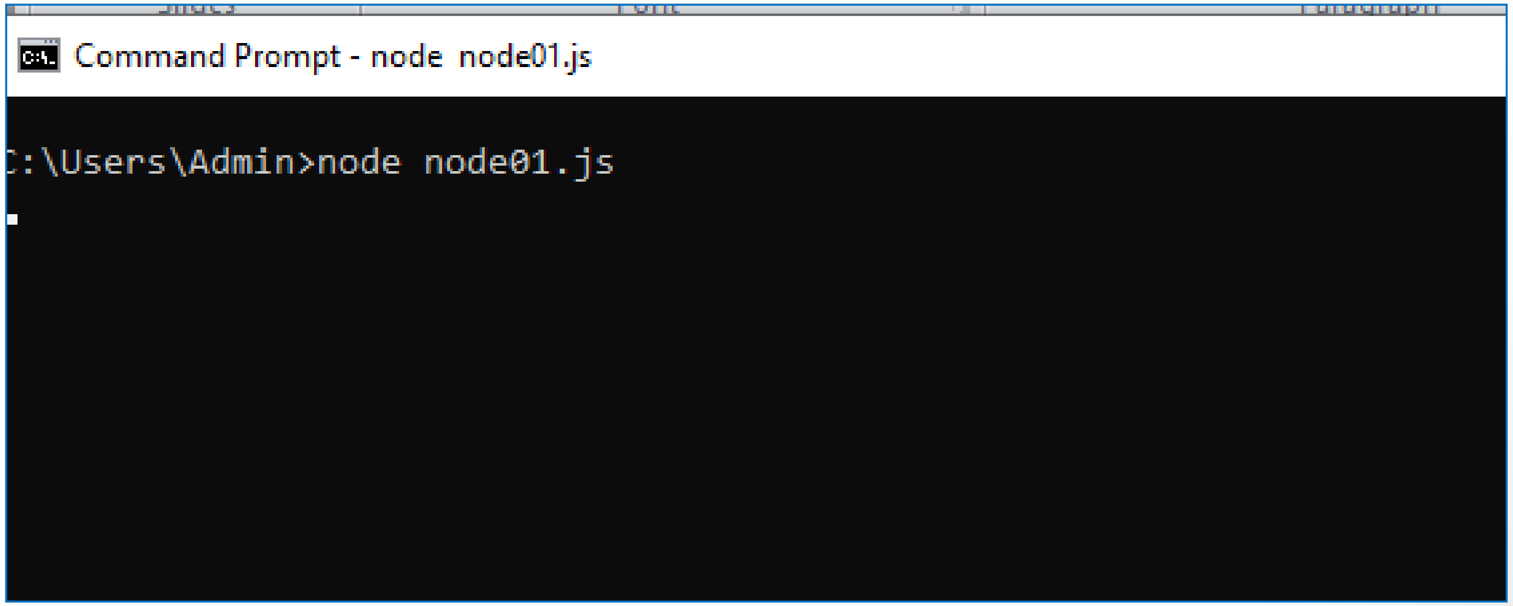
- Node.js files contain tasks that will be executed on certain events
- A typical event is someone trying to access a port on the server
- Node.js files must be initiated on the server before having any effect
- Node.js files have extension ".js"
- **Download Node.js**
- The official Node.js website has installation instructions for Node.js: https://nodejs.org
- Check the installation by



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\admin>node
> console.log("Welcome");
Welcome
undefined
>
```

C:\Users\admin\node01.js

```
var http = require('http');
http.createServer(function (req, res)
{
   res.writeHead(200,{'Content-Type':'text/html'});
   res.end('Hello,Your Node JS Server is ready!');
}).listen(8080);
```



```
:\Users\Admin>node node01.js
```

node01.js - explanation

- The basic functionality of the "require" function is that it reads a JavaScript file, executes the file, and then proceeds to return an object. Using this object, one can then use the various functionalities available in the module called by the require function. So in our case, since we want to use the functionality of http and we are using the require(http) command.
- In this 2nd line of code, we are creating a server application which is based on a simple function. This function is called, whenever a request is made to our server application.
- When a request is received, we are asking our function to return a 'Hello, Your Node JS Server is ready!' response to the client. The writeHead function is used to send header data to the client and while the end function will close the connection to the client.
- We are then using the server.listen function to make our server application listen to client requests on port no 8080. You can specify any available port over here.

Node.js application

- A Node.js application consists of the following three important components :
  - **Import required modules** − We use the **require** directive to load Node.js modules.
  - **Create server** − A server which will listen to client's requests similar to Apache HTTP Server.
  - **Read request and return response** − The server created in an earlier step will read the HTTP request made by the client which can be a browser or a console and return the response.

Creating Node.js Application

- Step 1 - Import Required Module
  - We use the **require** directive to load the http module and store the returned HTTP instance into an http variable as follows −

```
var http = require("http");
```

- Step 2 - Create Server
  - We use the created http instance and call **http.createServer()** method to create a server instance (web server object)and then we bind it at port 8080 using the **listen** method associated with the server instance.
  - Pass it a function with parameters request and response

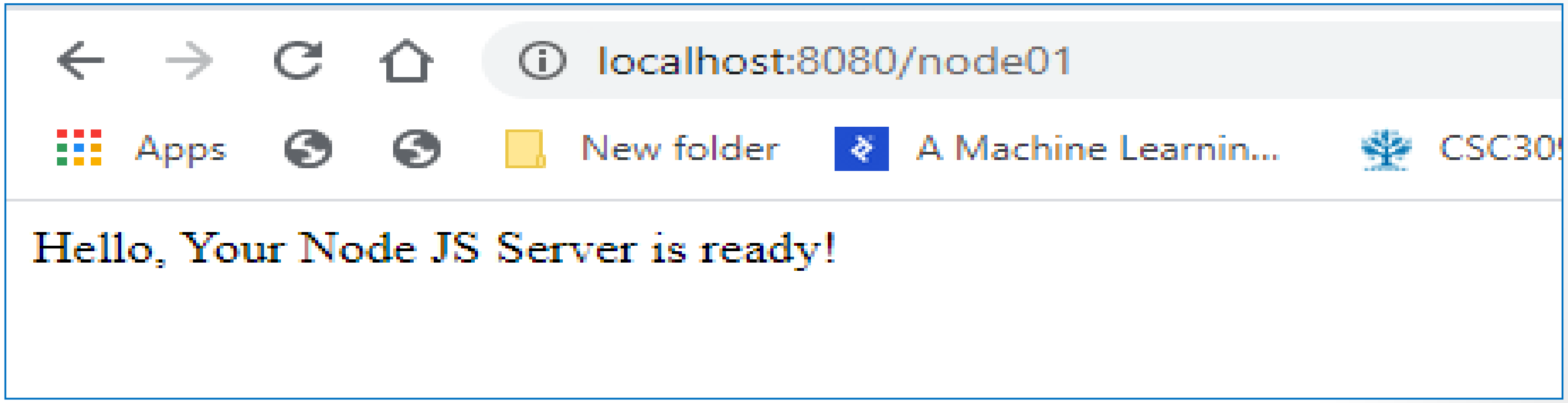
```
http.createServer(function (request, response)  {
………                       }).listen(8080);
```

- Step 3 - Testing Request & Response

```
{
res.writeHead(200, {'Content-Type':'text/html'});
res.end('Hello, Your Node JS Server is  ready!');
}
```

Executing the code

- Save the file on your computer: C:\Users\admin\node01.js
- In the command prompt, navigate to the folder where the file is stored. Enter the command node node01.js
- Now, your computer works as a server! If anyone tries to access your computer on port 8080, they will get a "Hello, Your Node JS Server is ready!" message in return!
- Start your internet browser, and type in the address: http://localhost:8080



```
Hello, Your Node JS Server is ready!
```

What are modules in Node.js?

- Consider modules to be the same as JavaScript libraries. A set of functions you want to include in your application.
- Modules in Node js are a way of encapsulating code in a separate logical unit. There are many readymade modules available in the market which can be used within Node js.
- Below are some of the popular modules which are used in a Node js application
  - **Express framework** – Express is a minimal and flexible Node js web application framework that provides a robust set of features for the web and Mobile applications.
  - **Socket.io** - Socket.IO enables real-time bidirectional event-based communication. This module is good for creation of chatting based applications.
  - **Jade** - Jade is a high-performance template engine and implemented with JavaScript for node and browsers.
  - **MongoDB** - The MongoDB Node.js driver is the officially supported node.js driver for MongoDB.
  - **Restify** - restify is a lightweight framework, similar to express for building REST APIs
  - **Bluebird** - Bluebird is a fully featured promise library with focus on innovative features and performance

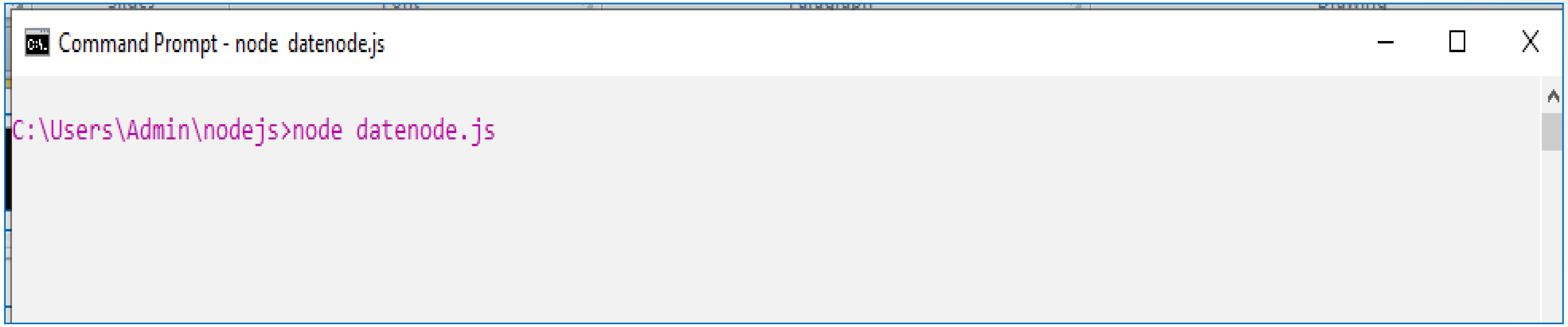
C:\Users\admin\getdate.js

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

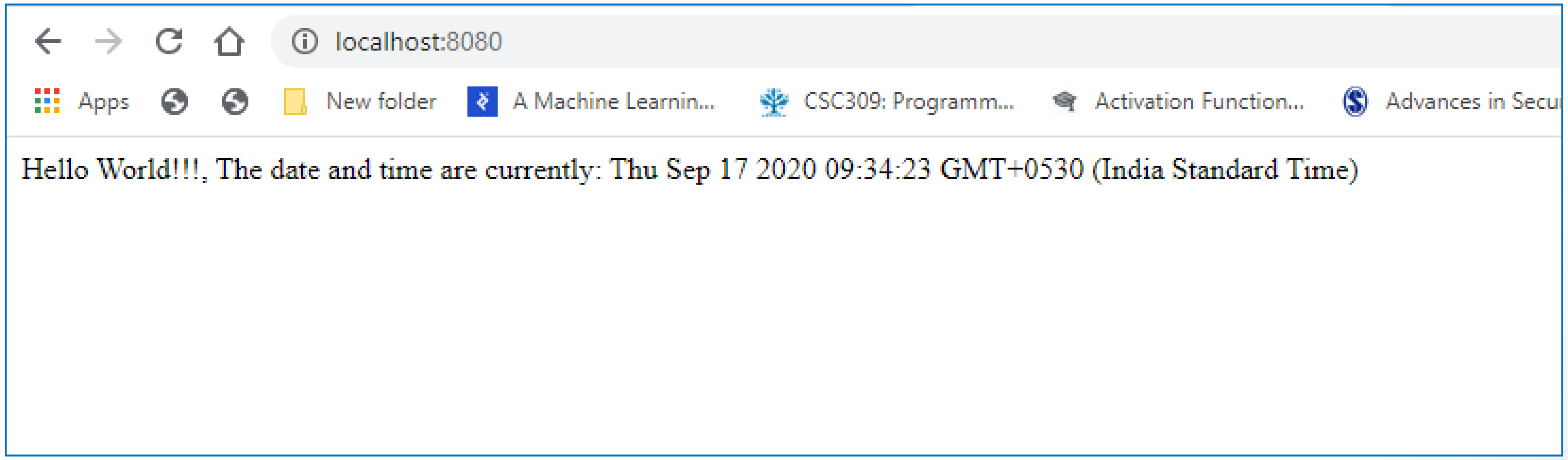
Use the exports keyword to make properties and methods available outside the module file.

C:\Users\admin\datenode.js

```
var http = require('http');
var dt = require('./getdate'); // import
http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type':'text/html'});
    res.write("Hello World !!!, The date and time are
    currently: " +   dt.myDate ());
    res.end();}).listen(8080);
```


Command Prompt - node datenode.js

```
C:\Users\Admin\nodejs>node datenode.js
```



```
Hello World!!!, The date and time are currently: Thu Sep 17 2020 09:34:23 GMT+0530 (India Standard Time)
```

Node.js HTTP Module

The Built-in HTTP Module

- Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the HyperTextTransfer Protocol (HTTP).
- To include the HTTP module, use the require() method:
- var http = require('http');

Node.js as a WebServer

- 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.
- The function passed into the http.createServer() method, will be executed when someone tries to access the computer on port 8080.

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.
- The first argument of the res.writeHead() method is the status code, 200 means that all is OK, the second argument is an object containing the response headers.

Read the Query String

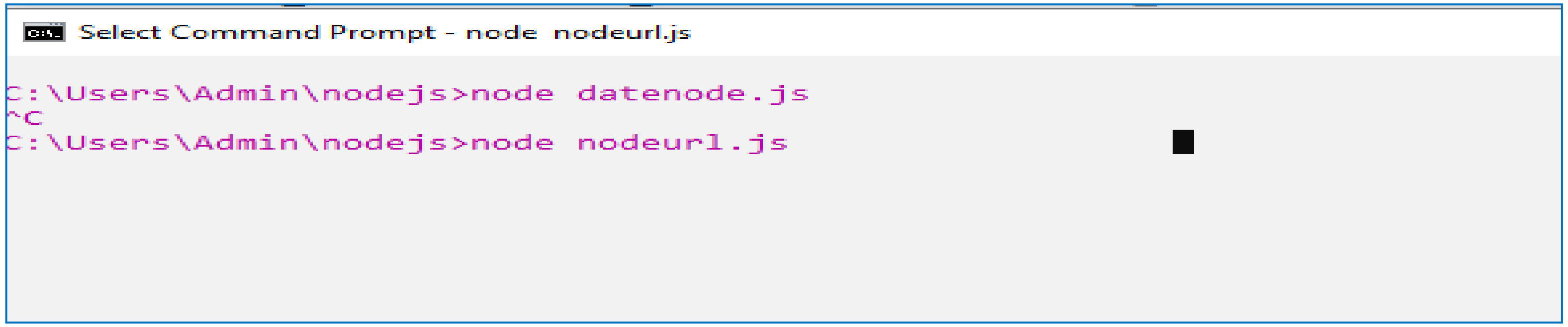
- The function passed into the http.createServer() has a req argument that represents the request from the client, as an object (http.IncomingMessage object).
- This object has a property called "url" which holds the part of the url that comes after the domain name:

C:\Users\admin\nodeurl.js

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

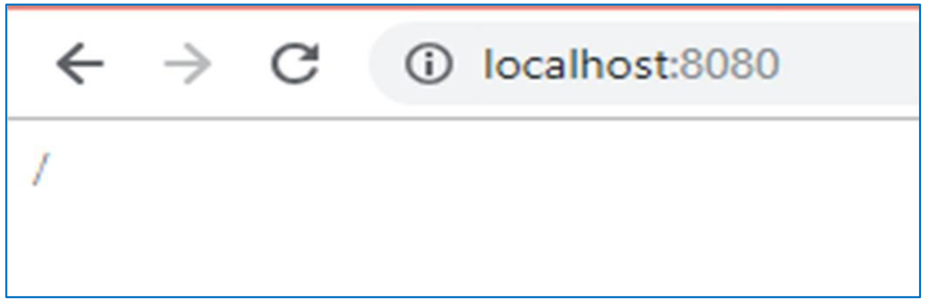
It just refers the value of URL

Initiate the node js server

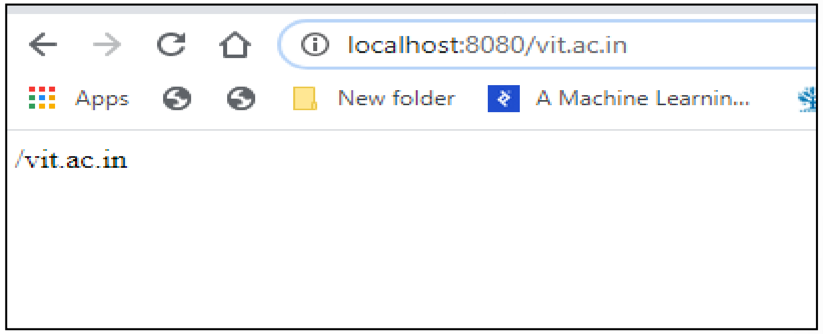


```
Select Command Prompt - node  nodeurl.js

C:\Users\Admin\nodejs>node datenode.js
^C
C:\Users\Admin\nodejs>node nodeurl.js
```

Run the browser

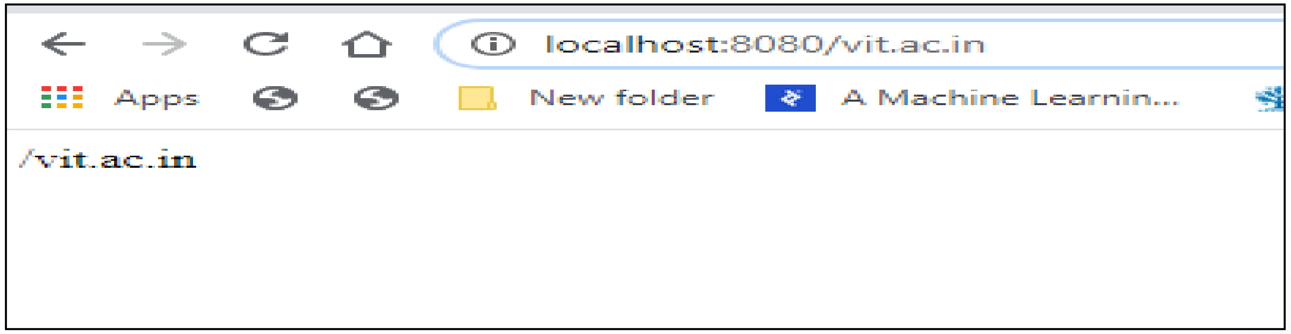


```
localhost:8080

/
```



```
localhost:8080/vit.ac.in

/vit.ac.in
```

Run the browser with adding contents in url



```
localhost:8080/vit.ac.in

/vit.ac.in
```

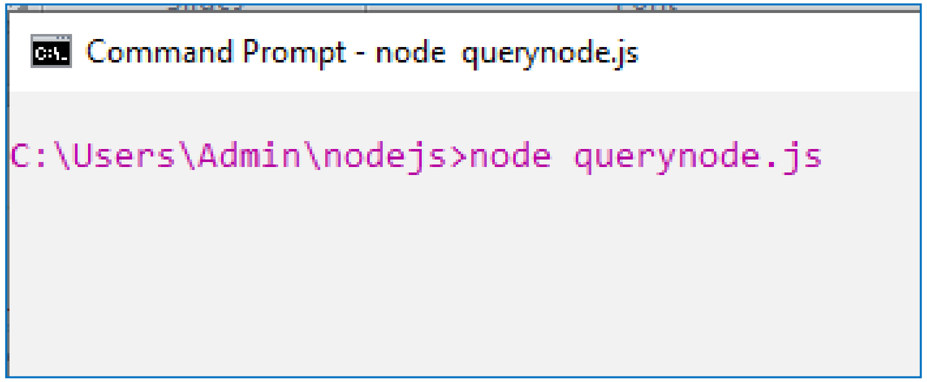
C:\Users\admin\querynode.js

Split the Query String

```
var http = require('http');
var url = require('url');
http.createServer(function (req, res) {


  res.writeHead(200, {'Content-Type':'text/html'});
  var q = url.parse(req.url, true).query;
  var txt = q.year + " " + q.month;
  res.end(txt);
}).listen(8080);
```

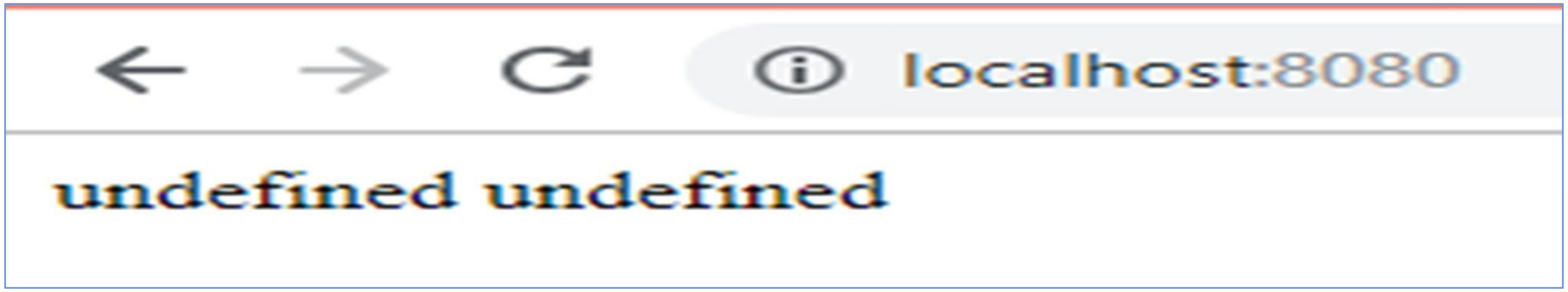
There are built-in modules to easily split the query string into readable parts, such as the URL module.

Initiate the node js server



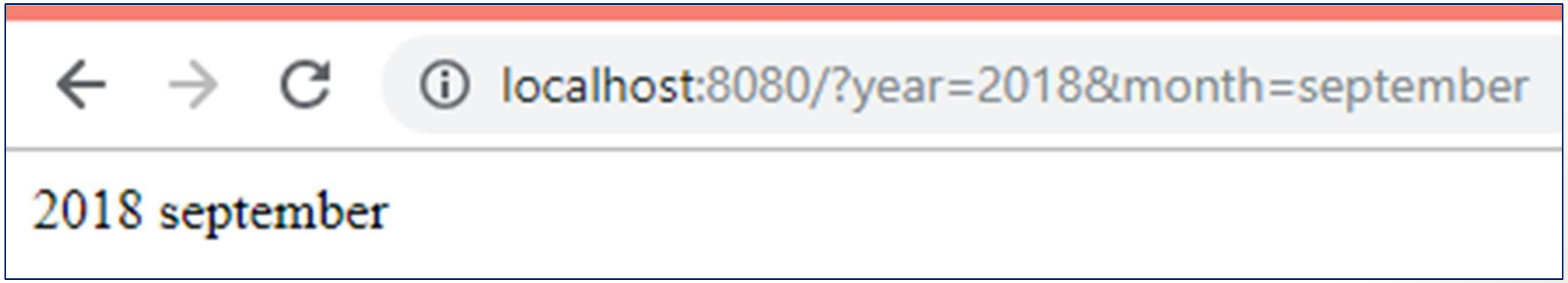
```
Command Prompt - node querynode.js

C:\Users\Admin\nodejs>node querynode.js
```

Run the browser



```
localhost:8080

undefined undefined
```

Run the browser with adding query string by ? in url



```
localhost:8080/?year=2018&month=september

2018 september
```

Node.js File System Module

- 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: Read files, Create files, Update files, Delete files and Rename files.
- The fs.readFile() method is used to read files on your computer.

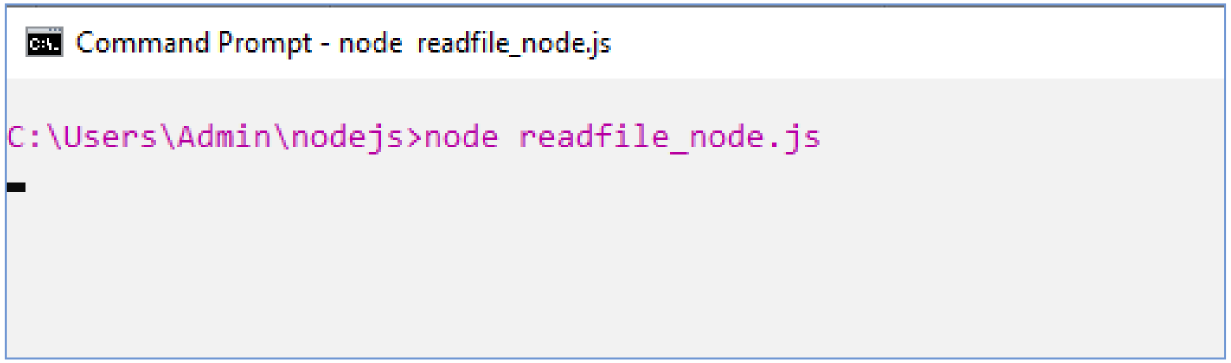
C:\Users\admin\readfile1.html

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9
/angular.min.js"></script>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9
/angular-route.js"></script>

<body ng-app="myApp">

<p><a href="#/!">Main</a></p>

<a href="#!banana">Banana</a>
<a href="#!tomato">Tomato</a>
<a href="#!tomato">Lemon</a>

<p>Click on the links to change the content.</p>

<p>The HTML shown in the ng-view directive are written
in the template property of the $routeProvider.when
method.</p>

<div ng-view></div>

<script>
var app = angular.module("myApp", ["ngRoute"]);
app.config(function($routeProvider) {
   $routeProvider
   .when("/", {
      template : "<h1>Main</h1><p>Click on the links to
change this content</p>"
   })
   .when("/banana", {
      template : "<h1>Banana</h1><p>Bananas contain
around 75% water.</p>"
   })
   .when("/tomato", {
      template : "<h1>Tomato</h1><p>Tomatoes contain
around 95% water.</p>"
   });
                    when("/lemon", {
      template : "<h1>Tomato</h1><p>Lemon is rich in
vitamine C and around 95% water.</p>"
   });
});
</script>

</body>
</html>
```

C:\Users\admin\node_readfile.js

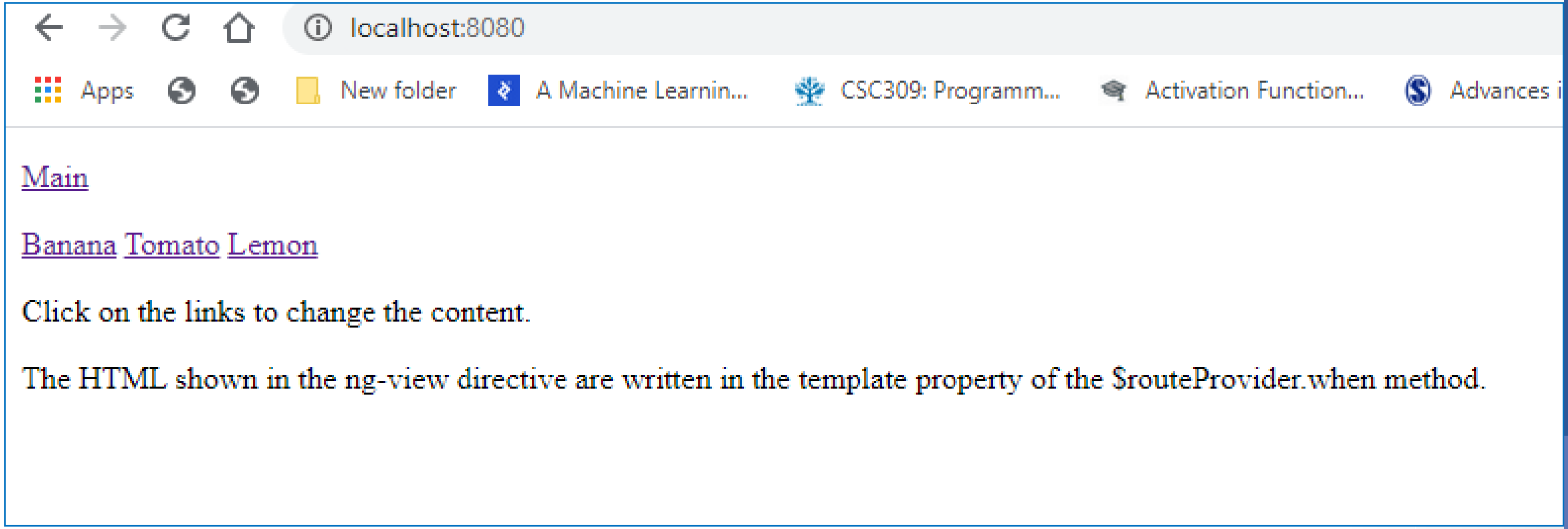
```
var http = require('http');
var fs = require('fs');
http.createServer(function (req, res)
{
 fs.readFile('readfile1.html', function(err, data) {
   res.writeHead(200, {'Content-Type': 'text/html'});
   res.write(data);
   res.end();
 });}).listen(8080);
```

Initiate readfile_node.js


Command Prompt - node readfile_node.js

```
C:\Users\Admin\nodejs>node readfile_node.js
```

Run in browser



Main

Banana Tomato Lemon

Click on the links to change the content.

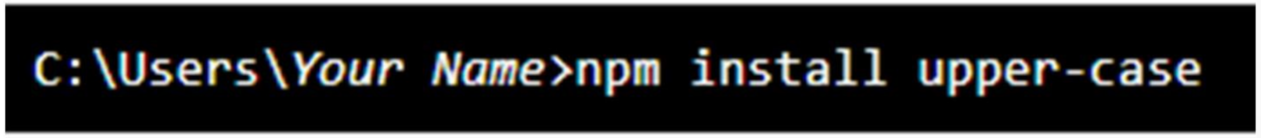
The HTML shown in the ng-view directive are written in the template property of the $routeProvider.when method.

Node.js NPM

- NPM is a package manager for Node.js packages, or modules if you like. Modules are JavaScript libraries you can include in your project.
- 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

Download a Package

- Open the command line interface and tell NPM to download the package you want.

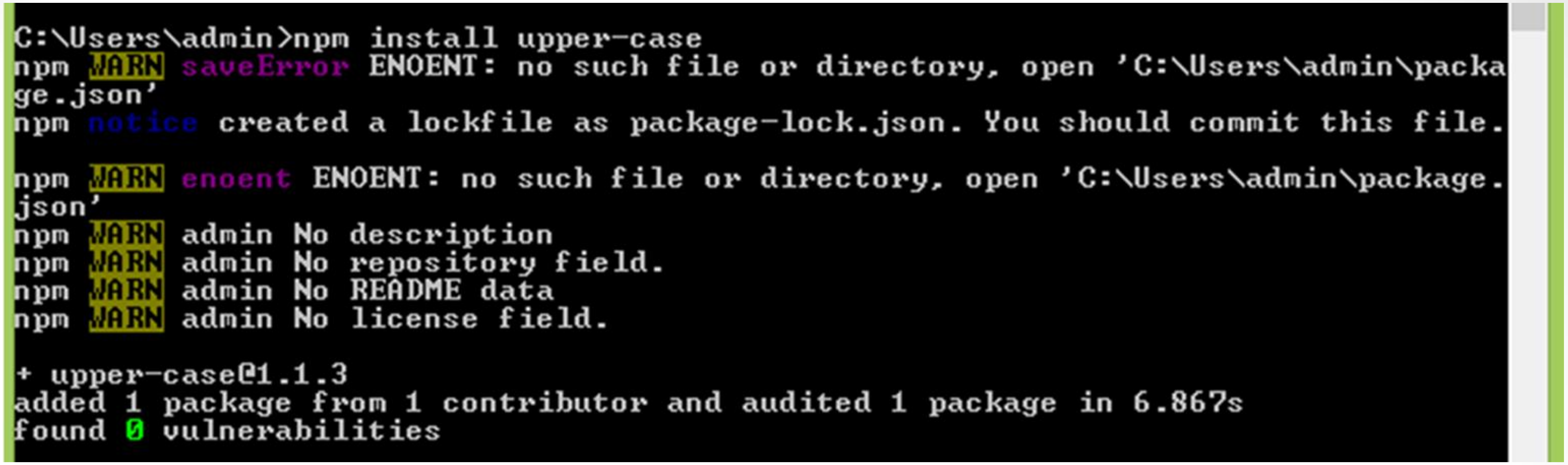


```
C:\Users\Your Name>npm install upper-case
```

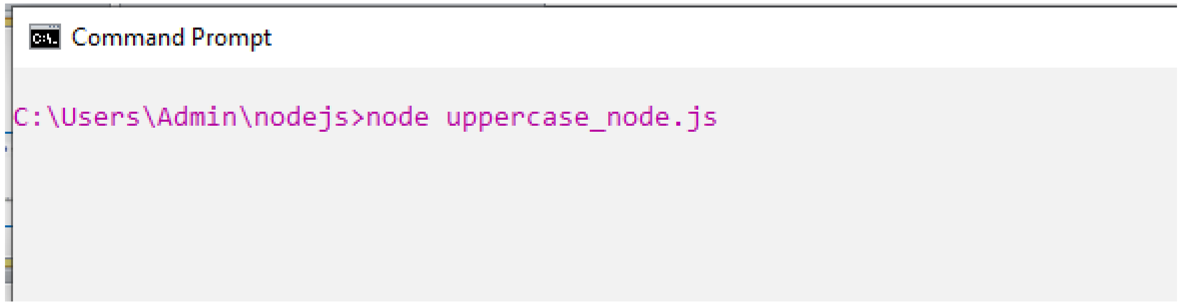
- NPM creates a folder named "node_modules", where the package will be placed. All packages you install in the future will be placed in this folder.

Using a Package

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

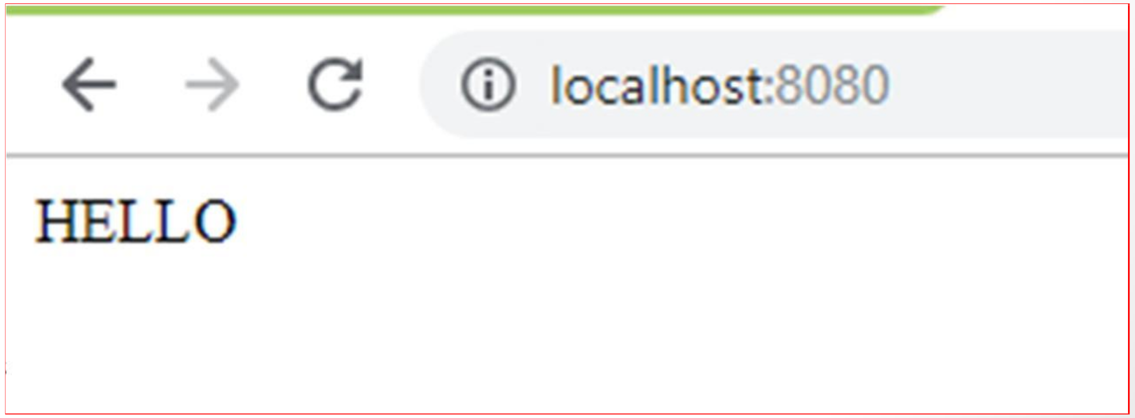


```
C:\Users\admin>npm install upper-case
npm WARN saveError ENOENT: no such file or directory, open 'C:\Users\admin\packa
ge.json'
npm notice created a lockfile as package-lock.json. You should commit this file.

npm WARN enoent ENOENT: no such file or directory, open 'C:\Users\admin\package.
json'
npm WARN admin No description
npm WARN admin No repository field.
npm WARN admin No README data
npm WARN admin No license field.

+ upper-case@1.1.3
added 1 package from 1 contributor and audited 1 package in 6.867s
found 0 vulnerabilities
```



Command Prompt

```
C:\Users\Admin\nodejs>node uppercase_node.js
```



localhost:8080

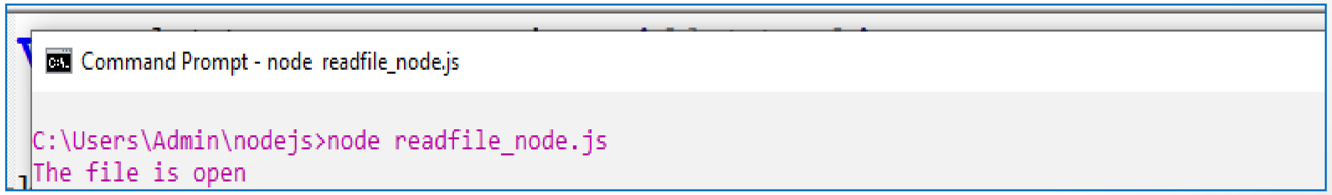
HELLO

Node JS - Events

- Node.js is perfect for event-driven applications.
- Every action on a computer is an event. Like when a connection is made or a file is opened.
- Objects in Node.js can fire events, like the readStream object fires events when opening and closing a file.

C:\Users\admin\readfile_node.js

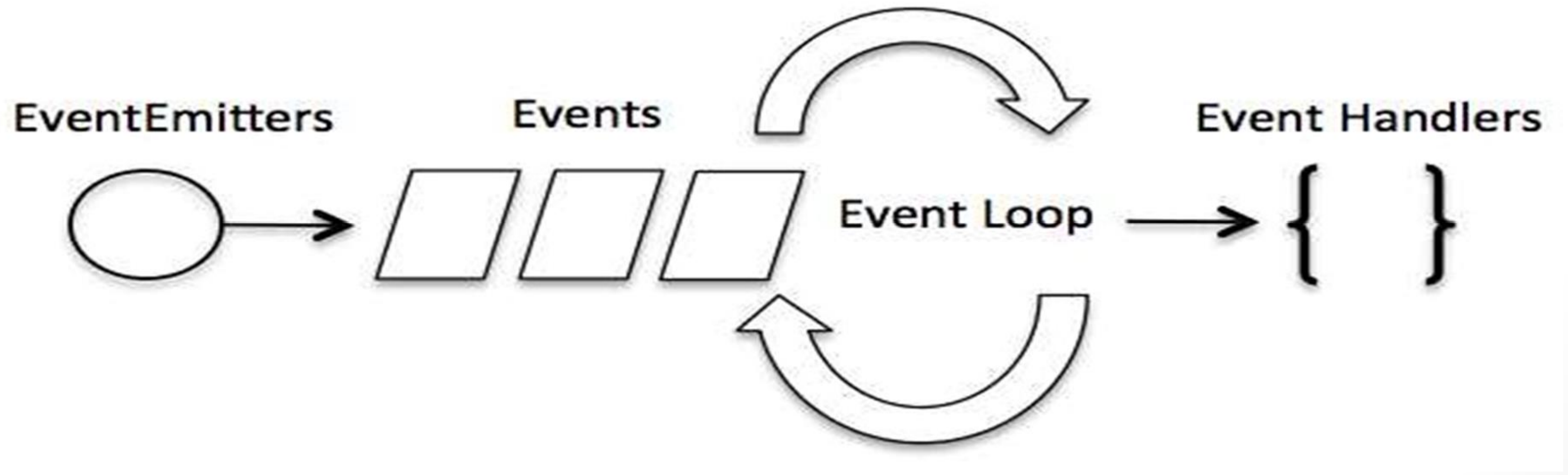
```
var fs = require('fs');
var readStream = fs.createReadStream('./ex-1.html');
readStream.on('open', function () {
  console.log('The file is open');});
```


Command Prompt - node readfile_node.js

```
C:\Users\Admin\nodejs>node readfile_node.js
The file is open
```

Event-Driven Programming

- Node.js uses events heavily and it is also one of the reasons why Node.js is pretty **fast** compared to other similar technologies. As soon as Node starts its server, it simply initiates its variables, declares functions and then simply waits for the event to occur.
- In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected.

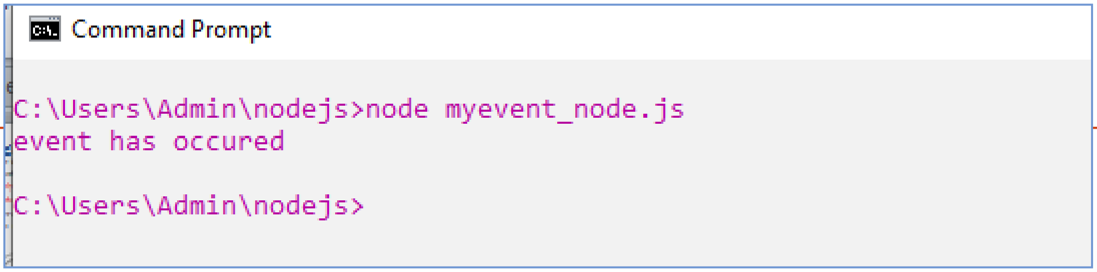


What and Why eventemitter?

- Put simply, it allows you to listen for "events" and assign actions to run when those events occur. If you're familiar with front-end JavaScript, you'll know about mouse and keyboard events that occur on certain user interactions. These are very similar, except that we can emit events on our own, when we want to, and not necessary based on user interaction.
- The principles EventEmitter is based on have been called the publish/subscribe model, because we can subscribe to events and then publish them. There are many front-end libraries built with pub/sub support, but Node has it buildin.

- Node.js has a built-in module, called "Events", where *you can create-, fire-, and listen for- your own events.*
- All event properties and methods are an instance of an **EventEmitter** object.
- You can assign event handlers to your own events with the EventEmitter object. To fire an event, use the emit() method.

```
var events = require("events");
var myEmitter=new events.EventEmitter();
myEmitter.on("someEvent", function () {
      console.log("event has occured");   });
myEmitter.emit("someEvent");
```



```
C:\Users\Admin\nodejs>node myevent_node.js
event has occured

C:\Users\Admin\nodejs>
```

- // Import events module

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

- Create an eventEmitter object

```
var eventEmitter = new events.EventEmitter();
```

- Bind event and event handler as follows
  ```
  eventEmitter.on('eventName', eventHandler);
  ```

- // Fire an event

```
eventEmitter.emit('eventName');
```

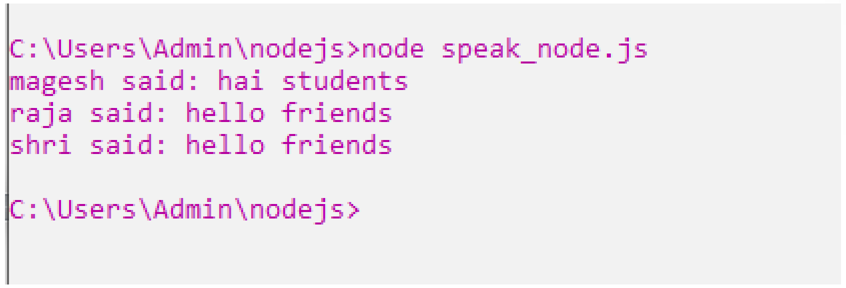
Example

```
const EventEmitter = require('events').EventEmitter;
const chatRoomEvents = new EventEmitter; → Step 1
function userJoined(username){ → Step 4
// Assuming we already have a function to alert all users.
alertAllUsers('User ' + username + ' has joined the chat.'); }
// Run the userJoined function when a 'userJoined' event is triggered.
chatRoomEvents.on('userJoined', userJoined); → Step 3


function login(username){
        chatRoomEvents.emit('userJoined', username); } → Step 2
```

util module

- The util module is primarily designed to support the needs of Node.js' own internal APIs.
- However, many of the utilities are useful for application and module developers as well.
- **util.inherits(constructor, superConstructor)**
- Inherit the prototype methods from one constructor into another. The prototype of constructor will be set to a new object created from superConstructor.

```
var events=require('events');
 var util=require('util');
var person=function(name){this.name=name;};
util.inherits(person,events.EventEmitter);
var magesh=new person('magesh');
var raja=new person('raja');
var shri=new person('shri');
var people=[magesh,raja,shri];
people.forEach(function(person){
            person.on('speak',function(msg){
            console.log(person.name+' said: '+msg);
            });
});
magesh.emit('speak','hai students');
```



```
C:\Users\Admin\nodejs>node speak_node.js
magesh said: hai students
raja said: hello friends
shri said: hello friends

C:\Users\Admin\nodejs>
```

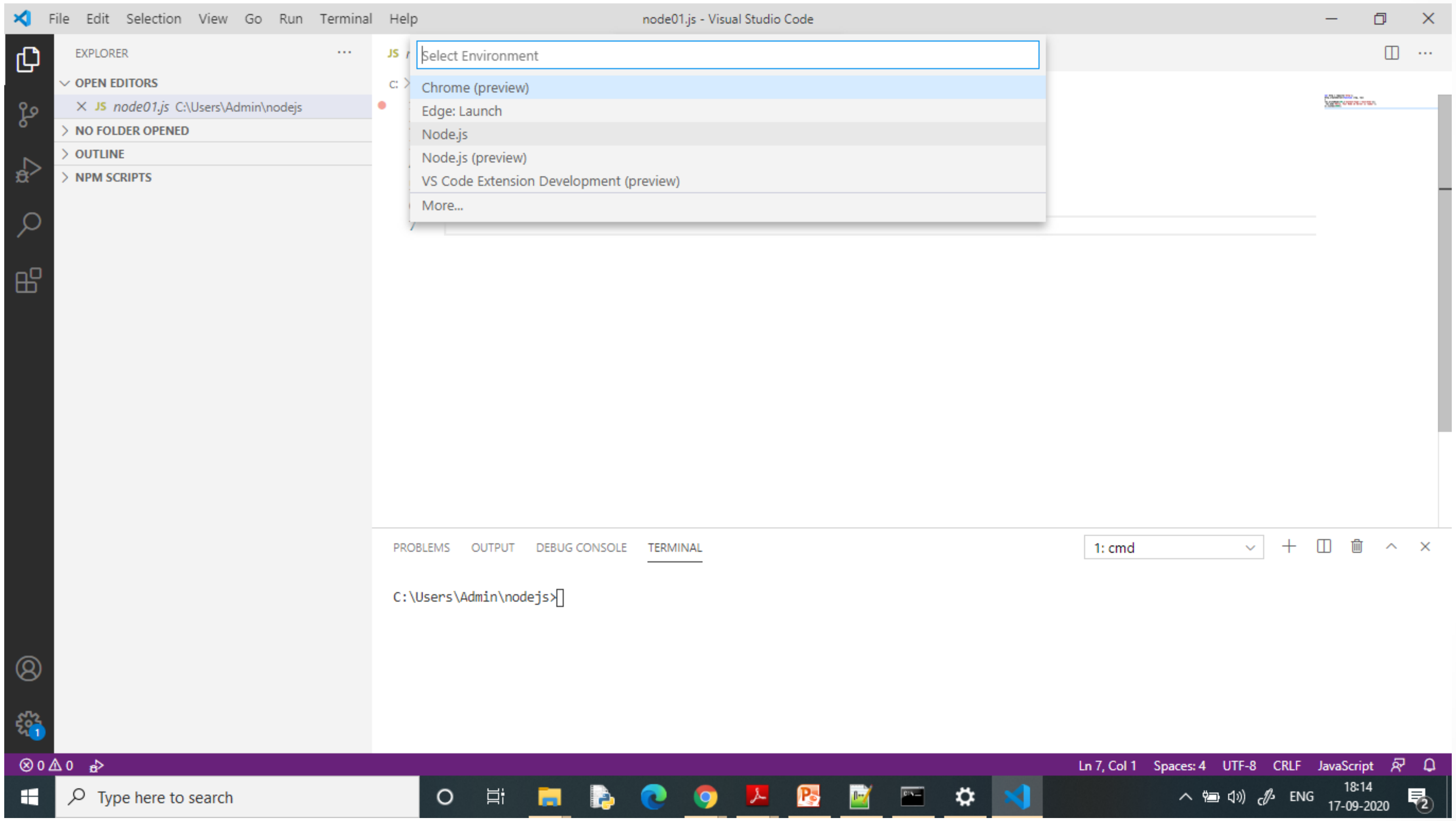
EventEmitter properties and methods

Method	Description
addListener()	Adds the specified listener
defaultMaxListeners	Sets the maximum number of listeners allowed for one event. Default is 10
emit()	Call all the listeners registered with the specified name
eventNames()	Returns an array containing all registered events
getMaxListeners()	Returns the maximum number of listeners allowed for one event
listenerCount()	Returns the number of listeners with the specified name
listeners()	Returns an array of listeners with the specified name
on()	Adds the specified listener
once()	Adds the specified listener once. When the specified listener has been executed, the listener is removed

EventEmitter properties and methods

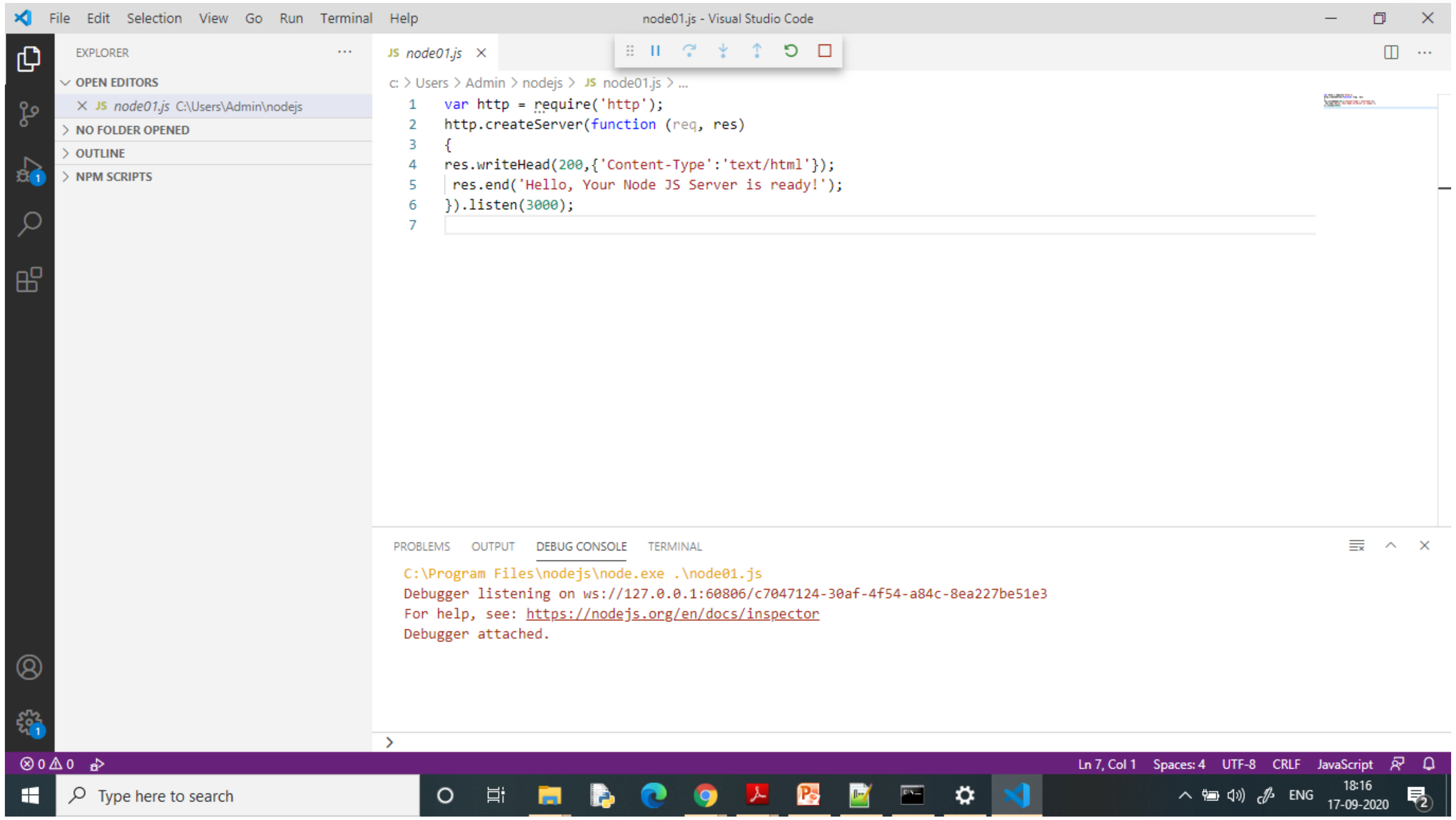
prependListener()	Adds the specified listener as the first event with the specified name
prependOnceListener()	Adds the specified listener as the first event with the specified name, once. When the specified listener has been executed, the listener is removed
removeAllListeners()	Removes all listeners with the specified name, or ALL listeners if no name is specified
removeListener()	Removes the specified listener with the specified name
setMaxListeners()	Sets the maximum number of listeners allowed for one event. Default is 10

Visual Studio Code - Editor
Install - https://code.visualstudio.com/


File Edit Selection View Go Run Terminal Help
node01.js - Visual Studio Code
EXPLORER
OPEN EDITORS
node01.js C:\Users\Admin\nodejs
NO FOLDER OPENED
OUTLINE
NPM SCRIPTS
Select Environment
Chrome (preview)
Edge: Launch
Node.js
Node.js (preview)
VS Code Extension Development (preview)
More...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: cmd
Ln 7, Col 1 Spaces: 4 UTF-8 CRLF JavaScript
Type here to search
18:14
17-09-2020

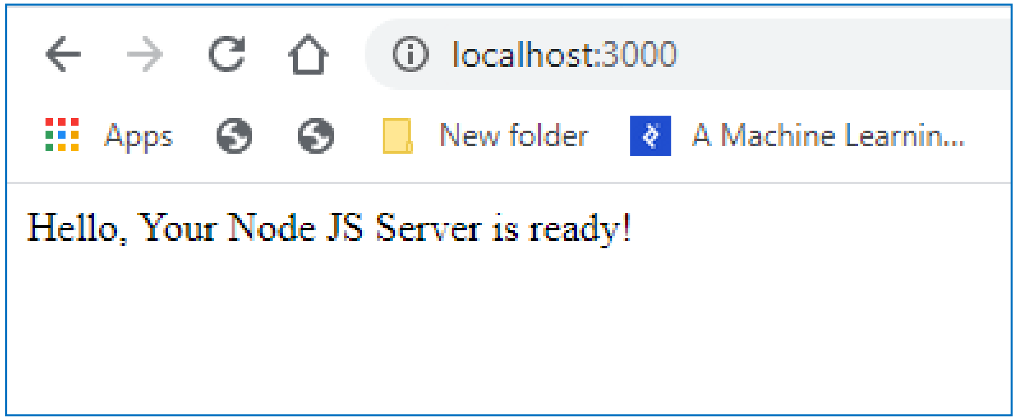
```
C:\Users\Admin\nodejs>
```

Run → without debugging → select node.js


File Edit Selection View Go Run Terminal Help
node01.js - Visual Studio Code
EXPLORER
OPEN EDITORS
node01.js C:\Users\Admin\nodejs
NO FOLDER OPENED
OUTLINE
NPM SCRIPTS
c: > Users > Admin > nodejs > node01.js > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Ln 7, Col 1 Spaces: 4 UTF-8 CRLF JavaScript
Type here to search
18:16
17-09-2020

```
var http = require('http');
http.createServer(function (req, res)
{
res.writeHead(200,{'Content-Type':'text/html'});
 res.end('Hello, Your Node JS Server is ready!');
}).listen(3000);
```

```
C:\Program Files\nodejs\node.exe .\node01.js
Debugger listening on ws://127.0.0.1:60806/c7047124-30af-4f54-a84c-8ea227be51e3
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
```



Hello, Your Node JS Server is ready!

Node.js VS Apache

1. It's fast
2. It can handle tons of concurrent requests
3. It's written in JavaScript (which means you can use the same code server side and client side)

Platform	Number of request per second
PHP (via Apache)	3187,27
Static (via Apache)	2966,51
Node.js	5569,30

Conclusion

- Node.js faster than apache but it more hungry system’s CPU and memory

- Node.js use event based programming, it make the server doesn’t wait for the IO operation to complete while it can handle other request at the same time
- Ref: https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js

Appendix

- step 1: npm install express --save
- Express allows us to set up middlewares to respond to HTTP Requests
- step 2: npm install body-parser --save
- If you want to read HTTP POST data , you have to use the �body-parser� node module.
- npm install mongoose --save
- Mongoose is an object document modeling (ODM) layer which sits on the top of Node�s MongoDB driver.
- ow to run node appln
- node app.js
- http://127.0.0.1:3000/

Callback

- Callback is an asynchronous equivalent for a function
- 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.

Callback

- 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.

Blocking Code Example

```
var fs = require("fs");
var data = fs.readFileSync('input.txt');
console.log(data.toString());
console.log("Program  Ended");
```

o/p:

```
Welcome to node.js
Program Ended
```

Non-Blocking Code Example

```
var fs = require("fs");
    fs.readFile('input.txt', function (err, data) {
if (err)  return console.error(err);
console.log(data.toString());
   });
   console.log("Program Ended");
```

o/p:

```
Program Ended
Welcome to node.js
```

Blocking and Non Blocking code

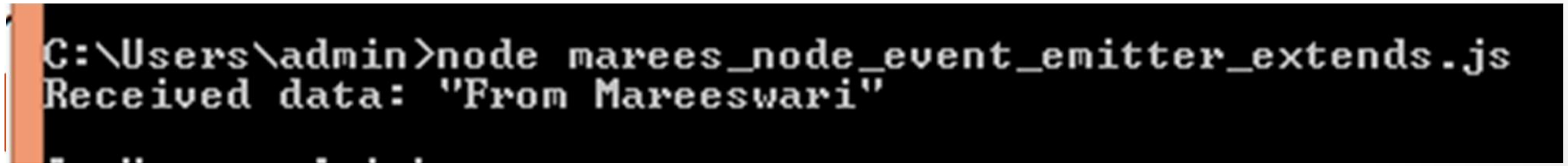
- These two examples explain the concept of blocking and non-blocking calls.
- The first example shows that the program blocks until it reads the file and then only it proceeds to end the program.
- The second example shows that the program does not wait for file reading and proceeds to print "Program Ended" and at the same time, the program without blocking continues reading the file.

Blocking and Non Blocking code

- Thus a blocking program executes very much in sequence.
- From the programming point of view, it is easier to implement the logic but non-blocking programs do not execute in sequence.
- In case a program needs to use any data to be processed, it should be kept within the same block to make it sequential execution.

extends keyword instead of inherits

```
var EventEmitter = require('events');
class MyStream extends EventEmitter {
write(data) {
   this.emit('data', data); }}
stream = new MyStream();
stream.on('data', (data) => {
 console.log(`Received data: "${data}"`);
});
stream.write('From marywari');
```



```
C:\Users\admin>node marees_node_event_emitter_extends.js
Received data: "From Mareeswari"
```

File System

- Synchronous vs Asynchronous
- Every method in the fs module has synchronous as well as asynchronous forms.
- Asynchronous methods take the last parameter as the completion function callback and the first parameter of the callback function as error.
- It is better to use an asynchronous method instead of a synchronous method, as the former never blocks a program during its execution, whereas the second one does.

- Syntax for writing to a file
- fs.writeFile(filename, data[, options], callback)This method will over-write the file if the file already exists

Parameters

- Here is the description of the parameters used −
- **path** − This is the string having the filename including path.
- **data** − This is the String or Buffer to be written into thefile.
- **options** −The third parameter is an object which will hold encoding.
- **callback** − This is the callback function which gets a single parameter err that returns an error in case of any writing error.

Reading from a file

- fs.read(fd, buffer, offset, length, position, callback) This method will use file descriptor to read the file. If you want to read the file directly using the file name, then you should use another method available.
- Parameters
- Here is the description of the parameters used −
- **fd** − This is the file descriptor returned by fs.open().
- **buffer** −This is the buffer that the data will be written to.
- **offset** −This is the offset in the buffer to start writing at.
- **length** − This is an integer specifying the number of bytes to read.
- **position** − This is an integer specifying where to begin reading from in the file. If position is null, data will be read from the current file position.
- **callback** − This is the callback function which gets the three arguments, (err, bytesRead, buffer).

Truncate a File

- The syntax of the method to truncate an opened file –
- fs.ftruncate(fd, len, callback)
- the description of the parameters used –
- **fd** – This is the file descriptor returned by fs.open().
- **len** – This is the length of the file after which the file will be truncated.
- **callback** – This is the callback function.

setTimeout(cb, ms)

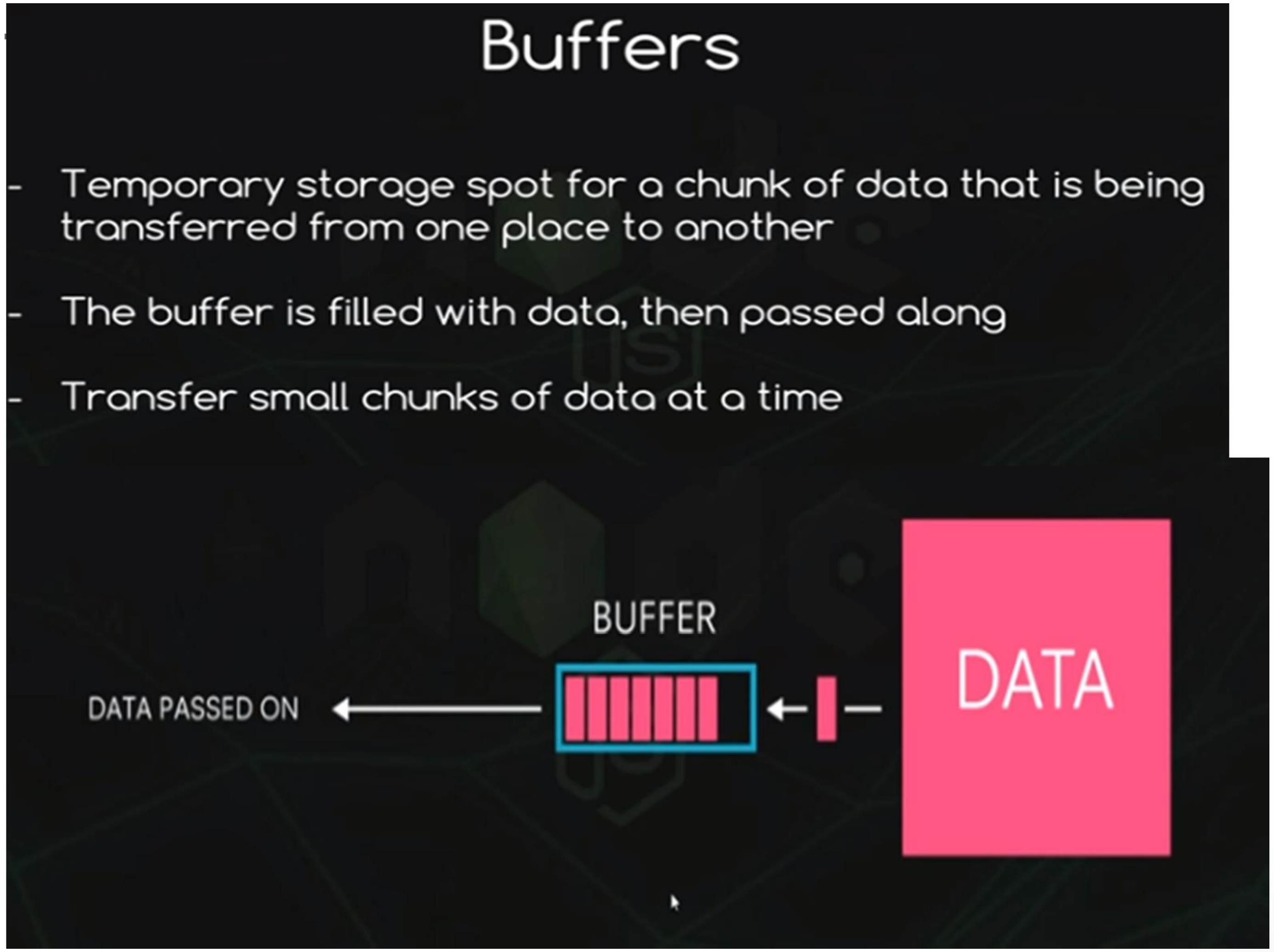
- The **setTimeout(cb, ms)** global function is used to run callback cb after at least ms milliseconds.
- The actual delay depends on external factors like OS timer granularity and system load.
- A timer cannot span more than 24.8 days.
- Ex: settimer.js

clearTimeout(t)

- The **clearTimeout(t)** global function is used to stop a timer that was previously created with setTimeout().
- t is a timer which is cleared.

- Ex: settime.js

Buffer



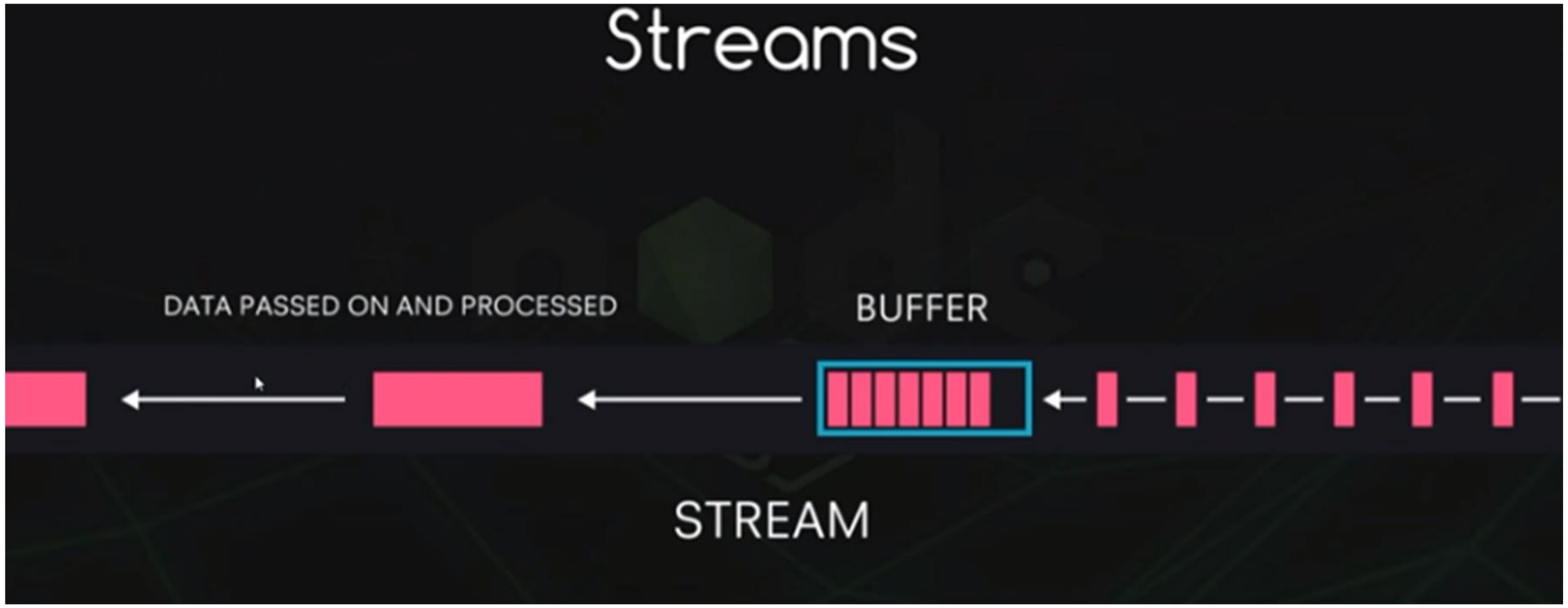
Buffers

- Temporary storage spot for a chunk of data that is being transferred from one place to another
- The buffer is filled with data, then passed along
- Transfer small chunks of data at a time

Stream

- In buffer,it collects all the chunks of data and when the buffer is full ,it passes the chunk of data down the stream to be processed and sends to client.

streams



- Writeable streams (when a response is received it is writing the data to the stream and client receives it)
- Readable streams(whenever we request data to the server, it will be reading the data coming from the stream)
- Duplex streams

Readstreams.js

Streams

- Each type of Stream is an **EventEmitter** instance and throws several events at different instance of times.
- For example, some of the commonly used events are −
- **data** − This event is fired when there is data is available to read.
- **end** − This event is fired when there is no more data to read.
- **error** − This event is fired when there is any error receiving or writing data.

Piping the Streams

- Piping is a mechanism where we provide the output of one stream as the input to another stream.
- It is normally used to get data from one stream and to pass the output of that stream to another stream.
- There is no limit on piping operations
- Ex: pipe1.js

Chaining the Streams

- Chaining is a mechanism to connect the output of one stream to another stream and create a chain of multiple stream operations.
- It is normally used with piping operations.
- Use piping and chaining to first compress a file and then decompress the same.
- Ex: chaining.js , decompress.js

Buffer

- Whenever we receive a file or data from some stream, that data cannot be used natively in javascript, so make it consumable, the raw data is converted into usable format and for that we use the buffer data.
- Node Buffer can be constructed in a variety of ways.
- Method 1
- Following is the syntax to create an uninitiated Buffer of size **10**
- var buf = new Buffer(10);
- Method 2
- Following is the syntax to create a Buffer from a given array –
- var buf = new Buffer([10, 20, 30, 40, 50]);

Buffer

- Following is the syntax to create a Buffer with optionally encoding type –

- var buf = new Buffer("Simply Easy Learning", "utf-8");

- Though "utf8" is the default encoding, you can use any of the following encodings "ascii", "utf8", "utf16le", "ucs2", "base64" or "hex".

Writing to Buffers

- Following is the syntax of the method to write into a Node Buffer −
- buf.write(string[, offset][, length][,encoding])
- the description of the parameters used −
- **string** −This is the string data to be written to buffer.
- **offset** −This is the index of the buffer to start writing at. Default value is 0.
- **length** −This is the number of bytes to write. Defaults to buffer.length.
- **encoding** − Encoding to use. 'utf8' is the default encoding.
- Return Value
- This method returns the number of octets written. If there is not enough space in the buffer to fit the entire string, it will write a part of the string.

Example of Buffer

- buf = new Buffer(256);
- len = buf.write("Simply Easy Learning");
- console.log("Octets written : "+ len);

Reading from Buffers

- buf.toString([encoding][, start][, end])
- the description of the parameters used −
- **encoding** − Encoding to use. 'utf8' is the default encoding.
- **start** − Beginning index to start reading, defaults to 0.
- **end** − End index to end reading, defaults is complete buffer.
- Return Value
- This method decodes and returns a string from buffer data encoded using the specified character set encoding.
- Reasbuf.js

Example of Buffer program

- buf = new Buffer(26);
- for (var i = 0 ; i < 26 ; i++)
- { buf[i] = i + 97; }
- console.log(buf.toString('ascii'));
- // outputs: abcdefghijklmnopqrstuvwxyz
    - console.log(buf.toString('ascii',0,5));
    - // outputs: abcde
    - console.log(buf.toString('utf8',0,5));
    - // outputs: abcde
    - console.log(buf.toString(undefined,0,5));
    - encoding defaults to'utf8', outputs abcde

Convert Buffer to JSON

- buf.toJSON()

- var buf = new Buffer('Simply Easy Learning');
- var json = buf.toJSON(buf);
- console.log(json);

- o/p: [83, 105, 109, 112, 108, 121, 32, 69, 97, 115, 121, 32, 76, 101, 97, 114, 110, 105, 110, 103]

Concatenate Buffers

- Buffer.concat(list[, totalLength])
- **list** – Array List of Buffer objects to be concatenated.
- **totalLength** – This is the total length of the buffers when concatenated.
- This method returns a Buffer instance.

Concatenate Buffers

- var buffer1 = new Buffer(‘Node js ');
- var buffer2 = new Buffer('Simply Easy Learning');
- var buffer3 = Buffer.concat([buffer1,buffer2]);

- console.log("buffer3 content: " + buffer3.toString());

- o/p:
- buffer3 content: Node js Simply Easy Learning

Copy Buffer

- buf.copy(targetBuffer[, targetStart][, sourceStart][, sourceEnd])

- Here is the description of the parameters used −

- **targetBuffer** − Buffer object where buffer will be copied.
- **targetStart** − Number, Optional, Default: 0
- **sourceStart** − Number, Optional, Default: 0
- **sourceEnd** − Number, Optional, Default: buffer.length

- No return value.
- Copies data from a region of this buffer to a region in the target buffer even if the target memory region overlaps with the source.

Copy Buffer

- var buffer1 = new Buffer('ABC');
- //copy a buffer
- var buffer2 = new Buffer(3);
  buffer1.copy(buffer2);
- console.log("buffer2 content: " +
  buffer2.toString());

- O/P: buffer2 content: ABC

Slice Buffer

- to get a sub-buffer of a node buffer −
- buf.slice([start][, end])
- the description of the parameters used −
- **start** − Number, Optional, Default: 0
- **end** − Number, Optional, Default: buffer.length
- Returns a new buffer which references the same memory as the old one, but offset and cropped by the start (defaults to 0) and end (defaults to buffer.length) indexes
- Concate.js

Example of Slicing Buffer

- var buffer1 = new Buffer(‘WelcometoNodejs'); //slicing a buffer
- var buffer2 = buffer1.slice(0,6);
- console.log("buffer2 content: " + buffer2.toString());

- o/p: Welcome

Buffer Length

- to get a size of a node buffer in bytes –
- buf.length;
- Returns the size of a buffer in bytes.
- Ex:
- var buffer = new Buffer(‘WelcometoNodejs');
- //length of the buffer
- console.log("buffer length: " + buffer.length);

- o/p: buffer length: 15