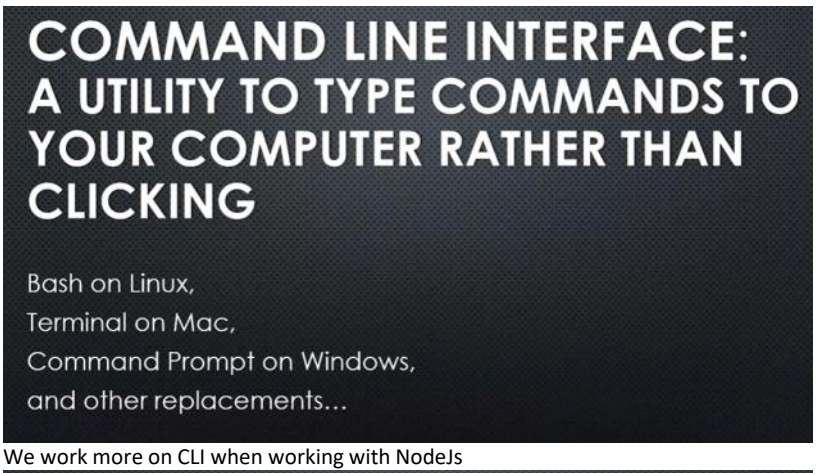
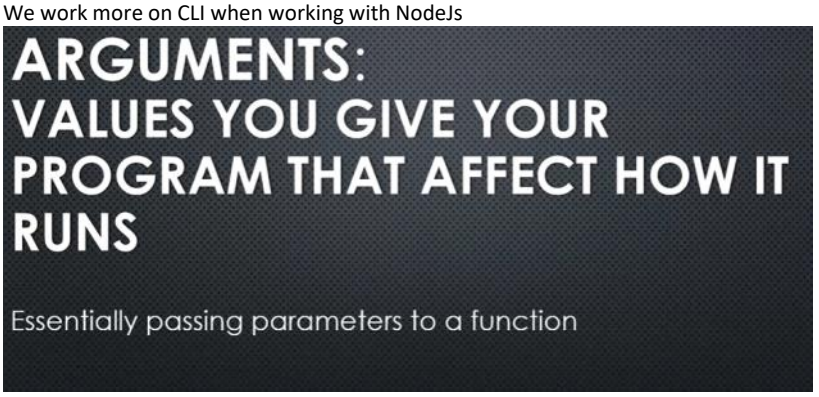
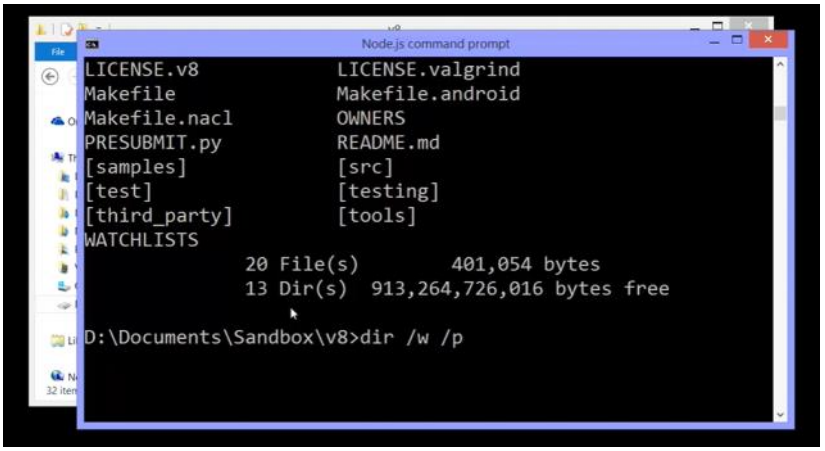
NodeJS

Wednesday, May 6, 2020 11:32 AM

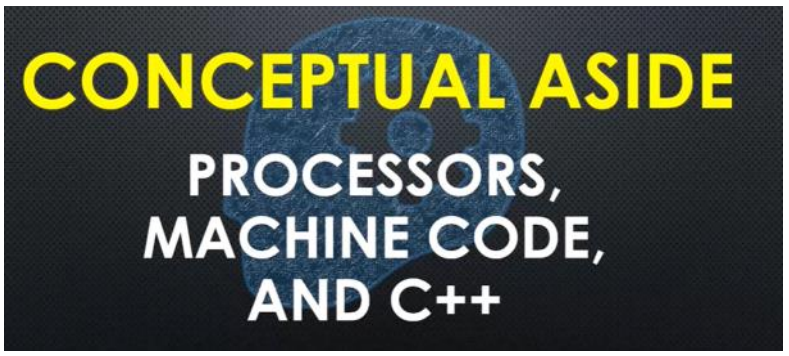
Introduction

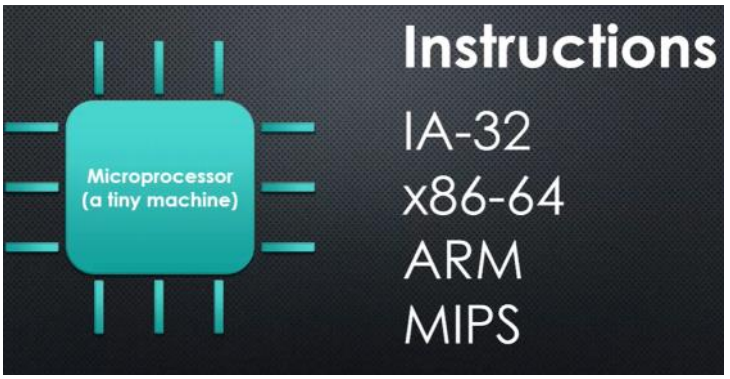
- 

- We work more on CLI when working with NodeJs
- 

  - Ex.
  - 

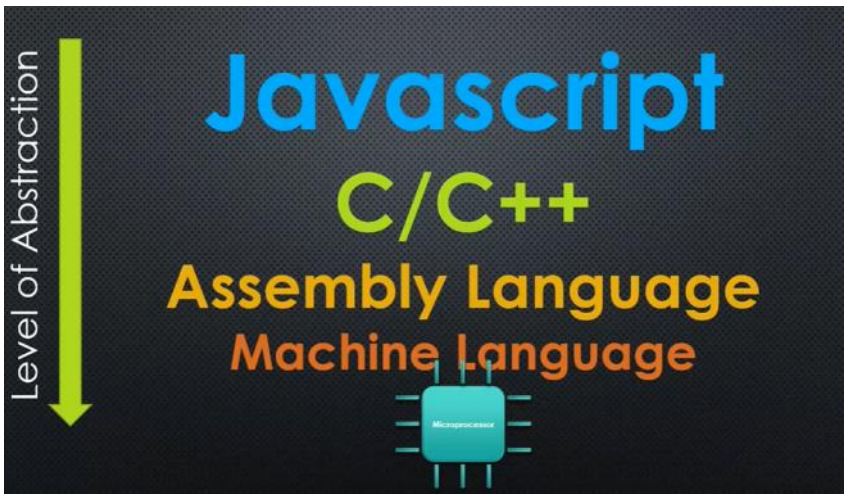
V8 - The Javascript Engine

- 

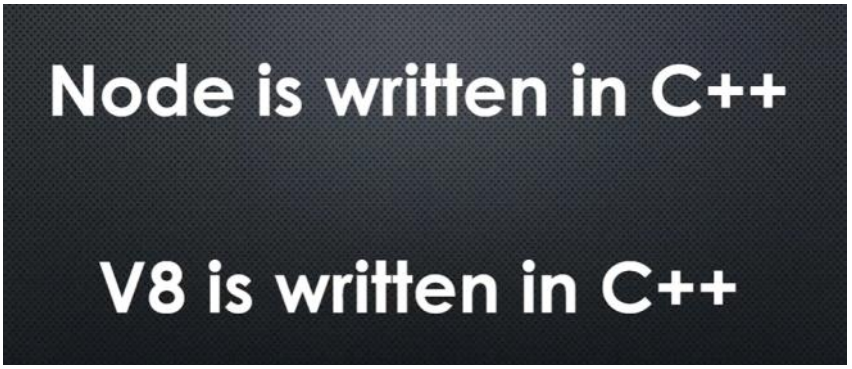
- Processors
  - 

  - 

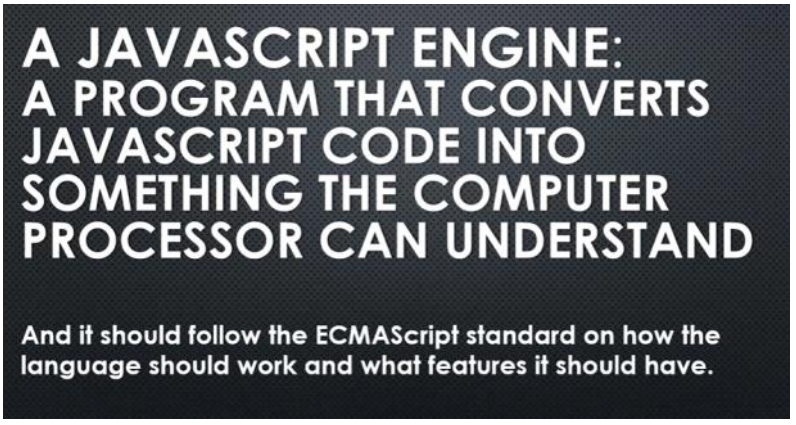
  - Instead of directly writing machine code/machine languages we write in languages that are converted/compiled into machine code.

- 

- We have an engine between JS and the microprocessor that takes care of the conversion to machine code.

- 

- Javascript Engines and the ECMAScript Specification
  - 

  - 

  - There are so many JS engines other than V8

- V8
  - **V8 JavaScript Engine**

    V8 is Google's open source JavaScript engine.

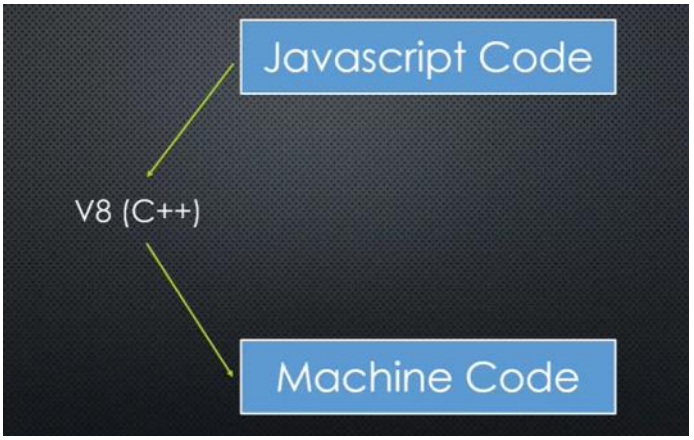
    V8 is written in C++ and is used in Google Chrome, the open source browser from Google.

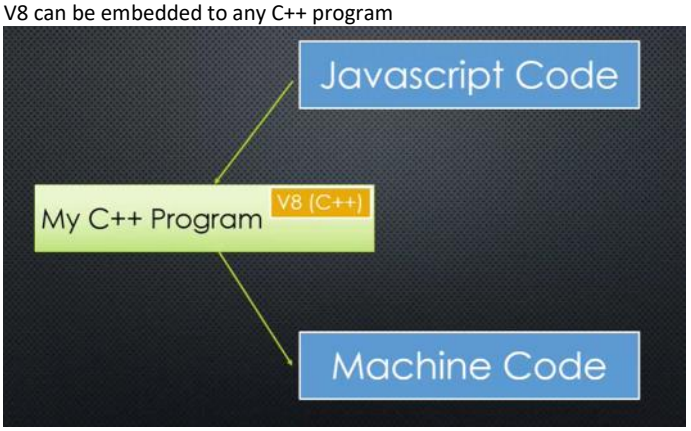
    V8 implements ECMAScript as specified in ECMA-262, 5th edition, and runs on Windows (XP or newer), Mac OS X (10.5 or newer), and Linux systems that use IA-32 (SSE2 required), x64, ARM (ARMv6 + VFP2 required) or MIPS processors.

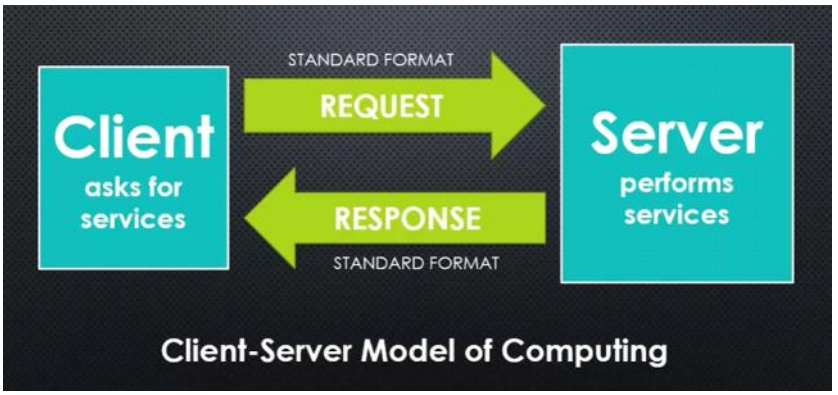
    V8 can run standalone, or can be embedded into any C++ application.

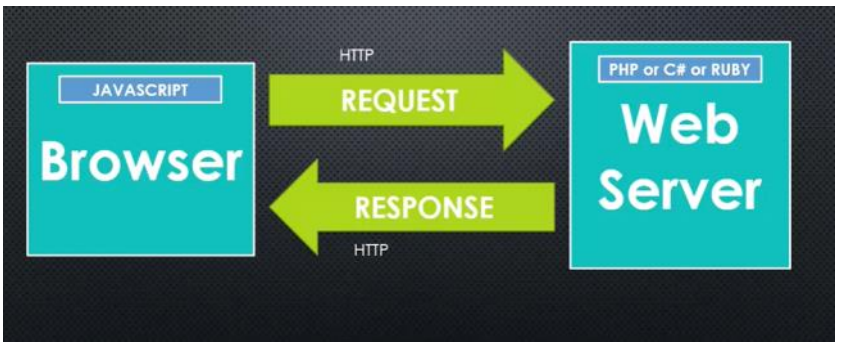
    You can find more information here:

    - The **V8 documentation** page which includes instructions on downloading and building V8.
    - **Performance documentation** covering the performance goals of V8, and instructions on how to run the Octane benchmark suite (evolution of the V8 benchmark suite).
    - **User mailing list**: http://groups.google.com/group/v8-users
    - The **V8 contributor** wiki page.
    - The **V8 blog** http://v8project.blogspot.com/
  - V8 can run standalone, or can be embedded into any C++ application.

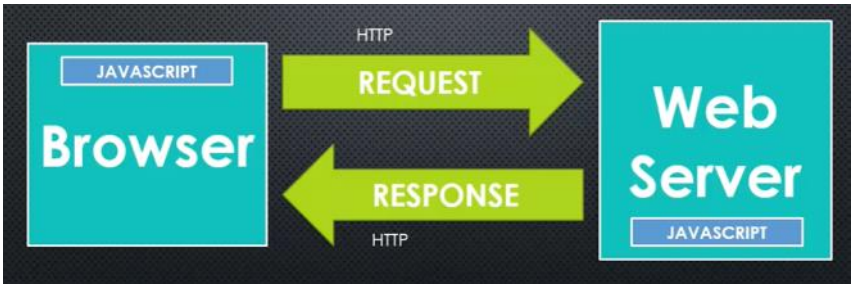
- 

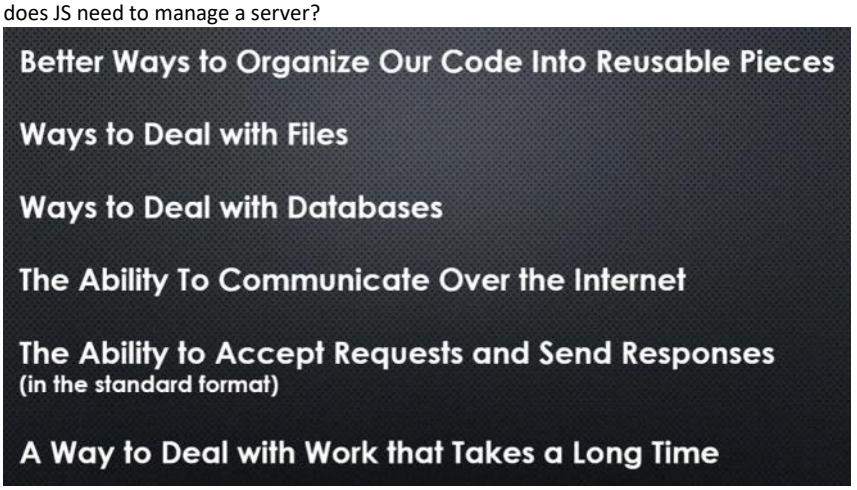
- V8 can be embedded to any C++ program
- 

- The Node Core
  - 

  - 

  - Generally, we use PHP/C#/Ruby etc.. for server side coding, but one of the purpose of nodejs is to write this in js

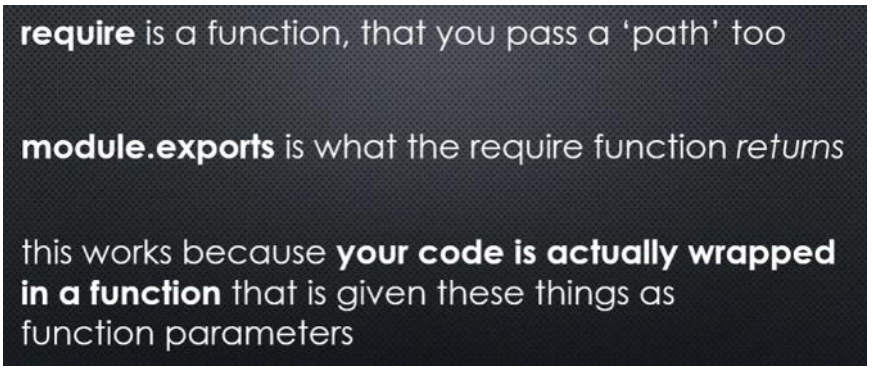
- 

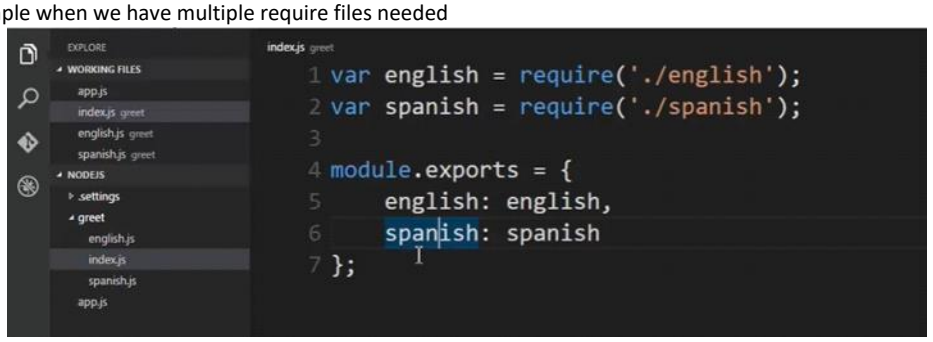
    - What does JS need to manage a server?
        - 

        - node folder structure
            - deps - dependencies - includes libraries which are built outside of nodejs
                - v8 - related to chrome engine
                - zlib - for zip files
                - http-parser
                - npm
            - src - c++ source code

- The JS Core
    - In the node folder structure, we also have
        - lib - consists of js files which are wrappers for c++ code/features to be used in js
            - util.js - generic functions for us to reuse

- Require
    - require() is not part of the standard JavaScript API. But in Node.js, it's a built-in function with a special purpose: to load modules.
    - Modules are a way to split an application into separate files instead of having all of your application in one file. This concept is also present in other languages with minor differences in syntax and behavior, like C's include, Python's import, and so on.
    - One big difference between Node.js modules and browser JavaScript is how one script's code is accessed from another script's code.
        - In browser JavaScript, scripts are added via the <script> element. When they execute, they all have direct access to the global scope, a "shared space" among all scripts. Any script can freely define/modify/remove/call anything on the global scope.
        - In Node.js, each module has its own scope. A module cannot directly access things defined in another module unless it choosesto expose them. To expose things from a module, they must be assigned to exports or module.exports. For a module to access another module's exports or module.exports, it must use require().
    - In your code, var pg = require('pg'); loads the pg module, a PostgreSQL client for Node.js. This allows your code to access functionality of the PostgreSQL client's APIs via the pg variable.
    - Why does it work in node but not in a webpage?
    - require(), module.exports and exports are APIs of a module system that is specific to Node.js. Browsers do not implement this module system.
    - Also, before I got it to work in node, I had to do npm install pg. What's that about?
    - NPM is a package repository service that hosts published JavaScript modules. npm install is a command that lets you download packages from their repository.
    - Where did it put it, and how does Javascript find it?
    - The npm cli puts all the downloaded modules in a node_modules directory where you ran npm install. Node.js has very detailed documentation on how modules find other modules which includes finding a node_modules directory.
    - When require is used in js file to load another js module, it gets wrapped in a function expression and applies it.
        - 

-
```
787  NativeModule.getSource = function(id) {
788    return NativeModule._source[id];
789  }
790
791  NativeModule.wrap = function(script) {
792    return NativeModule.wrapper[0] + script +
       NativeModule.wrapper[1];
793  };
794
795  NativeModule.wrapper = [
796    '(function (exports, require, module, __filename,
       __dirname) {',
797    '\n});'
798  ];
```
-
```
(function (exports, require, module, __filename, __dirname) {

      var greet = function() {
        console.log('Hello!');
      };

      module.exports = greet;

});

fn(module.exports, require, module, filename, dirname);

return module.exports;
```
-


- Example when we have multiple require files needed
-


- 

- Module Patterns
  - greet1.js
```
1 module.exports = function() {
2     console.log('Hello world');
3 };
```
  - ```
1 var greet = require('./greet1');
2 greet();
```
  - greet2.js
```
1 module.exports.greet = function() {
2   console.log('Hello world!')
3 }
```
  - ```
4 var greet2 = require('./greet2').greet;
5 greet2();
```
  - greet3.js
```
1 function Greetr() {
2     this.greeting = 'Hello world!!';
3     this.greet = function() {
4         console.log(this.greeting);
5     }
6 }
7
8 module.exports = new Greetr();
```
  - ```
7 var greet3 = require('./greet3');
8 greet3.greet();
```
  - ```
7 var greet3 = require('./greet3');
8 greet3.greet();
9 greet3.greeting = 'Changed hello world!';
10
11 var greet3b = require('./greet3');
12 greet3b.greet();
```
  - If we assign the same require file to 2 different variables, will we see the greeting as 'Hello World!!' as in greet3.js or 'Changed hello world!' as changed above greet3b
    - Ans: 'Changed hello world!'
    - Reason: Cached Module stores the module which is already loaded and returns the same module in greet3 and passes the object by references as all objects do
    - To overcome this, instead of returning a new function constructor we have to give ability to the greet function to create a new function constructor as in the below pattern

greet4.js

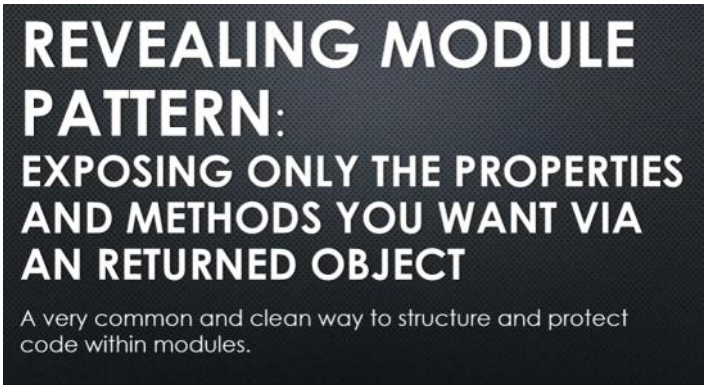
- greet4.js

```
function Greetr() {
    this.greeting = 'Hello world!!';
    this.greet = function() {
        console.log(this.greeting);
    }
}

module.exports = Greetr;
```

```
var Greet4 = require('./greet4');
var grtr = new Greet4();
grtr.greet();
```

- greet5.js

```
var greeting = 'Hello world!!!!';

function greet() {
    console.log(greeting);
}

module.exports = {
    greet: greet
}
```

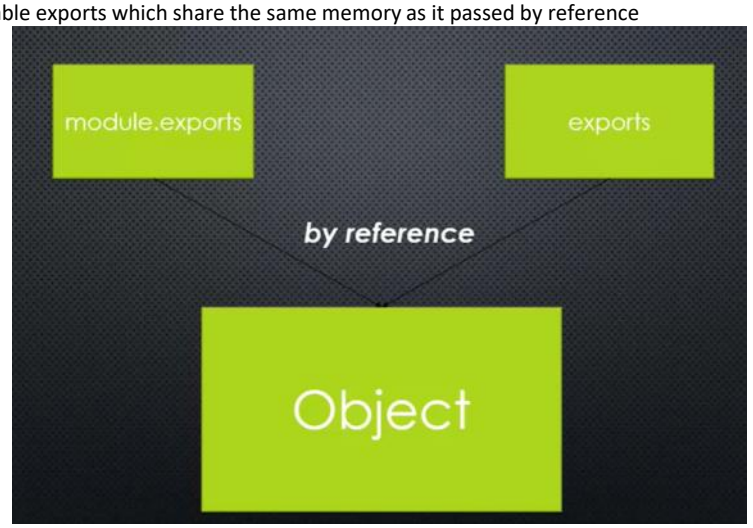
- 


```
var greet5 = require('./greet5').greet;
greet5();
```

- exports vs module.exports

```
(function (exports, require, module, __filename, __dirname) {

      var greet = function() {
        console.log('Hello!');
      };

      module.exports = greet;

});

fn(module.exports, require, module, filename, dirname);
```

  - exports and module.exports are two different variables sharing the same memory space.

- We pass module.exports to the function in which we use short hand to represent it as exports and return module.exports.
-
```
greet.js
1 exports = function() {
2     console.log('Hello');
3 }
4
5 console.log(exports);
6 console.log(module.exports);
```
-
```
D:\Documents\Sandbox\nodejs>node app.js
[Function]
{}
```
  - exports - function
  - module.exports - empty obj
- Initially, module.exports will be assigned an empty object({}) until function expression has been invoked and module.exports is passed to the variable exports which share the same memory as it passed by reference
- 

- But the reference is broken when an function is assigned a function using an equal to and they no longer share the same memory
```
greet.js
1 exports = function() {
2     console.log('Hello');
3 }
4
5 console.log(exports);
6 console.log(module.exports);
```
- 

  - We get an error, when executed

    - app.js

```
1 var greet = require('./greet');
2 greet();
```

    - Node.js command prompt

```
D:\Documents\Sandbox\nodejs\app.js:2
greet();
^
TypeError: object is not a function
    at Object.<anonymous> (D:\Documents\Sandbox\nodejs\app.j
s:2:1)
    at Module._compile (module.js:460:26)
    at Object.Module._extensions..js (module.js:478:10)
    at Module.load (module.js:355:32)
    at Function.Module._load (module.js:310:12)
    at Function.Module.runMain (module.js:501:10)
    at startup (node.js:129:16)
    at node.js:814:3

D:\Documents\Sandbox\nodejs>
```

- So, instead of using equal to we mutate the object, then we get the function assigned
    - greet2.js

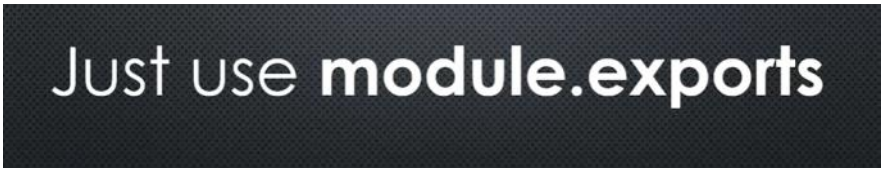
```
1 exports.greet = function() {
2     console.log('Hello');
3 }
4
5 console.log(exports);
6 console.log(module.exports);
```

    - app.js

```
1 var greet = require('./greet');
2 var greet2 = require('./greet2');
```

    -

```
D:\Documents\Sandbox\nodejs>node app.js
[Function]
{}
{ greet: [Function] }
{ greet: [Function] }

D:\Documents\Sandbox\nodejs>
```

    - 

- Requiring native(core) modules
    - We use core modules as in the below example (without a './')

app.js

```
1 var util = require('util');
2
3 var name = 'Tony';
4 var greeting = util.format('Hello, %s', name);
5 util.log(greeting);
```

```
D:\Documents\Sandbox\nodejs>node app.js
2 Sep 21:07:03 - Hello, Tony
```

- Modules and ES6

```
export function greet() {
  console.log('Hello');
}
```

greet.js

```
import * as greetr from 'greet';
greetr.greet();
```

app.js

- Web server checklist

Better Ways to Organize Our Code Into Reusable Pieces

Ways to Deal with Files

Ways to Deal with Databases

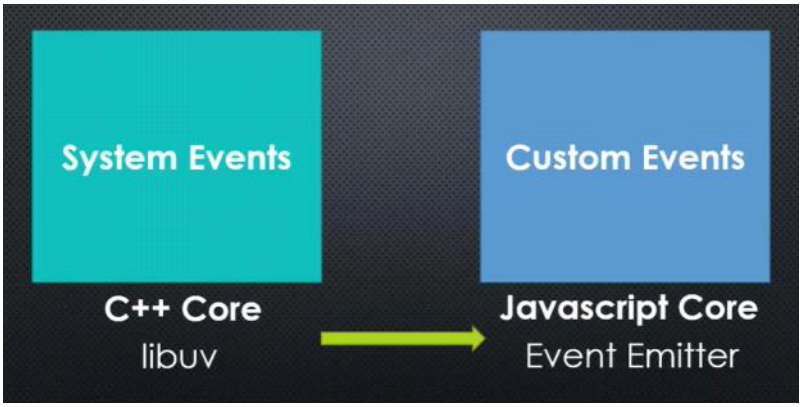
The Ability To Communicate Over the Internet

The Ability to Accept Requests and Send Responses
(in the standard format)

A Way to Deal with Work that Takes a Long Time

- Events and Event Emitter

- 

- 

  - System Events - Handled by C++ core using libuv
  - Custom Events - Handled by javascript core using Event Emitter
  - There is no eventing concept in JS, there is no event object but we can fake it and create our own

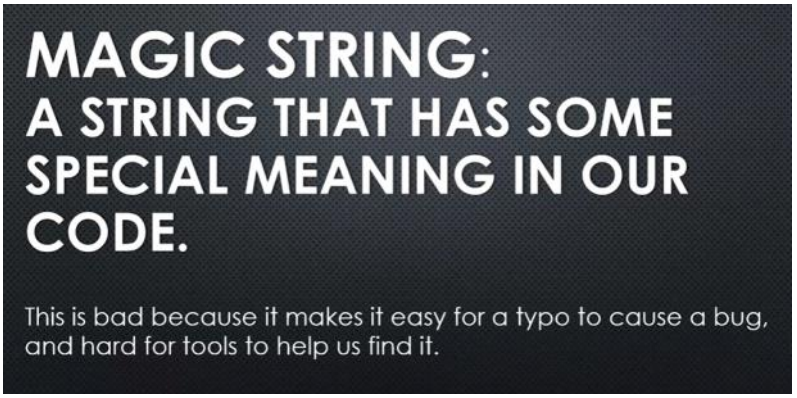
- Event Emitter
  - 

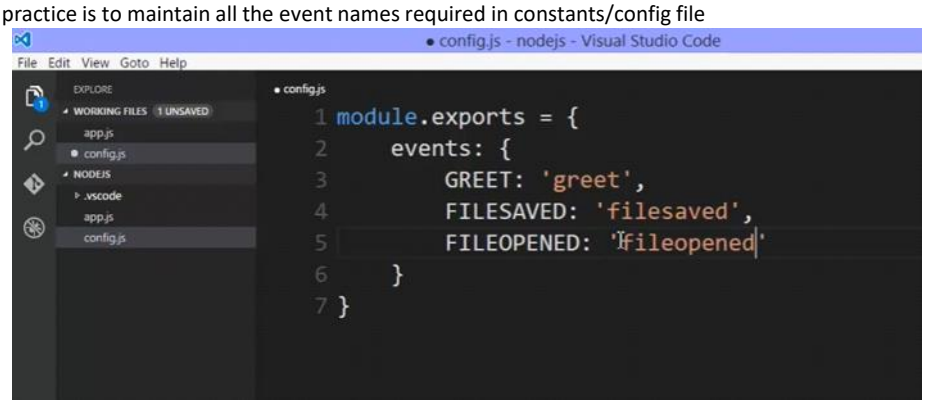
  - We can have n number of listeners listening to the same event
  - 

- Node Event Emitter

app.js

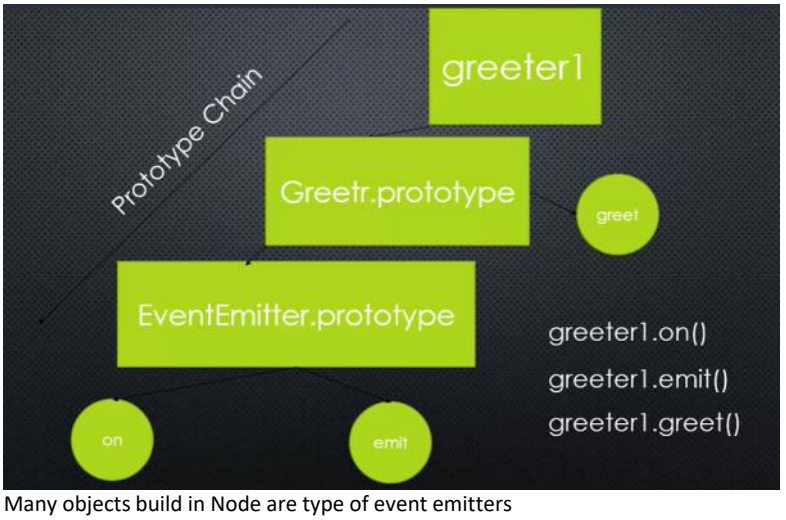
```
1 var Emitter = require('events');
2
3 var emtr = new Emitter();
4
5 emtr.on('greet', function() {
6     console.log('Somewhere, someone said hello.');
7 });
8
9 emtr.on('greet', function() {
10     console.log('A greeting occurred!');
11 });
12
13 console.log('Hello!');
14 emtr.emit('greet');
```

- 

- Best practice is to maintain all the event names required in constants/config file
  - 

  - app.js

```
1 var Emitter = require('events');
2 var eventConfig = require('./config').events;
3
4 var emtr = new Emitter();
5
6 emtr.on(eventConfig.GREET, function() {
7     console.log('Somewhere, someone said hello.');
8 });
9
10 emtr.on(eventConfig.GREET, function() {
11     console.log('A greeting occurred!');
12 });
13
14 console.log('Hello!');
15 emtr.emit(eventConfig.GREET);
```

- Inheriting from event emitter
  - Node provides inherit function for its exports in util.js
  - This is to inherit the prototype methods from one constructor into another
  - ```
    app.js
    var EventEmitter = require('events');
    var util = require('util');

    function Greetr() {
        this.greeting = 'Hello world!';
    }

    util.inherits(Greetr, EventEmitter);

    Greetr.prototype.greet = function() {
        console.log(this.greeting);
        this.emit('greet');
    }
    ```
  - ```
    var greeter1 = new Greetr();

    greeter1.on('greet', function() {
        console.log('Someone greeted!');
    });

    greeter1.greet();
    ```
  - Passing data to the event
    - ```
      Greetr.prototype.greet = function(data) {
          console.log(this.greeting + ': ' + data);
          this.emit('greet', data);
      }

      var greeter1 = new Greetr();

      greeter1.on('greet', function(data) {
          console.log('Someone greeted!: ' + data);
      });

      greeter1.greet('Tony');
      ```
  - 

  - Many objects build in Node are type of event emitters
  - Using call as super constructor while using inherits
      ```

- The below passes in the current object context to the parent constructor where we inherit the methods from
- Equal to calling super()

app.js
```
var EventEmitter = require('events');
var util = require('util');

function Greetr() {
    EventEmitter.call(this);
    this.greeting = 'Hello world!';
}

util.inherits(Greetr, EventEmitter);

Greetr.prototype.greet = function(data) {
    console.log(this.greeting + ': ' + data);
    this.emit('greet', data);
}

var greeter1 = new Greetr();
```

- Example

app2.js
```
var util = require('util');

function Person() {
    this.firstname = 'John';
    this.lastname = 'Doe';
}

Person.prototype.greet = function() {
    console.log('Hello ' + this.firstname + ' ' +
    this.lastname);
}
```

```
function Policeman() {
    Person.call(this);
    this.badgenumber = '1234';
}

util.inherits(Policeman, Person);
var officer = new Policeman();
officer.greet();
```

  - If Person.call(this) is removed the output would be "Hello undefined undefined"
    - Inherits just connects the prototypes

- To use ES6 in VSCode

jsconfig.json
```
{
    "compilerOptions": {
        "target": "ES6"
    }
}
```

- Using ES6 classes in nodejs
  - We need to mention 'use strict' at the top in node js to use ES6 classes
  - Examples

- app.js

```
'use strict';

class Person {
    constructor(firstname, lastname) {
        this.firstname = firstname;
        this.lastname = lastname;
    }

    greet() {
        console.log('Hello, ' + this.firstname + ' ' +
        this.lastname);
    }
}

var john = new Person('John', 'Doe');
john.greet();
```

- app.js

```
class Greetr extends EventEmitter {
    constructor() {
        super();
        this.greeting = 'Hello world!';
    }

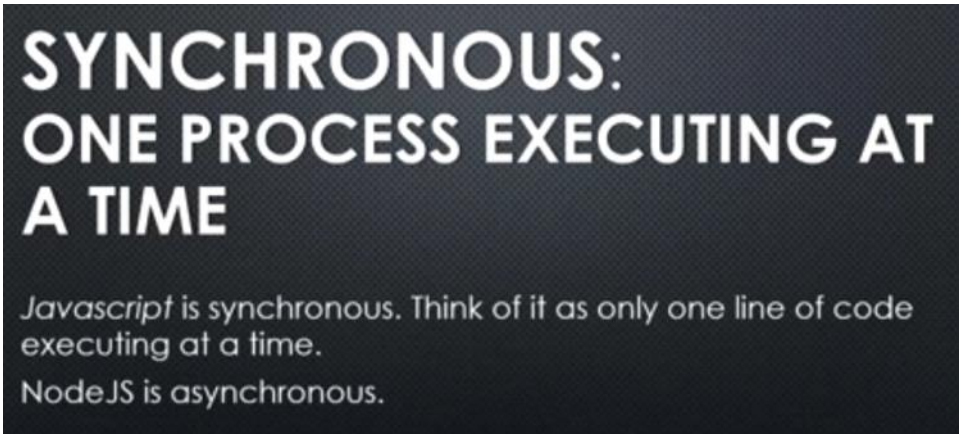
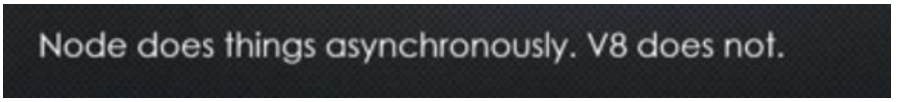
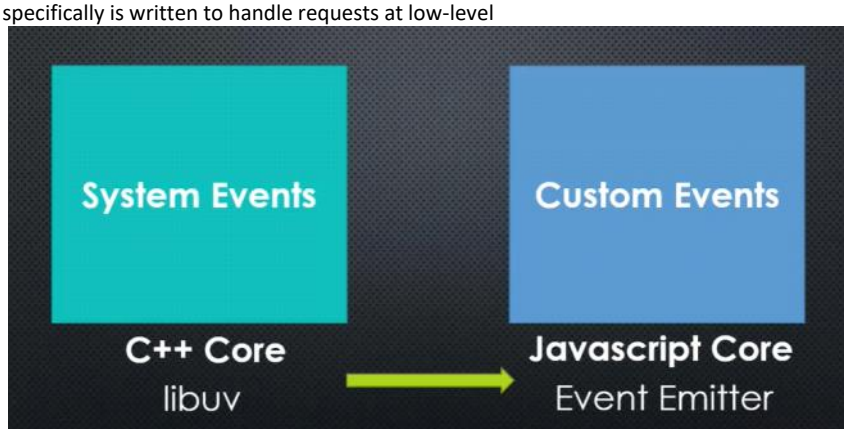
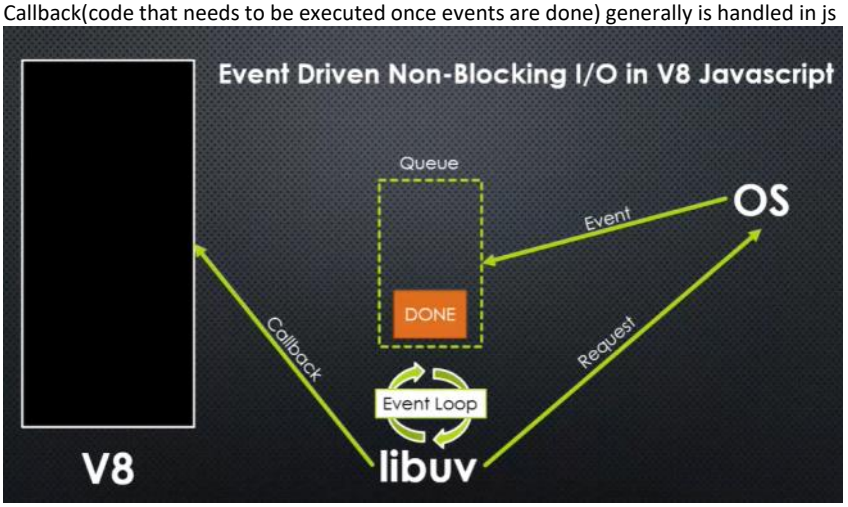
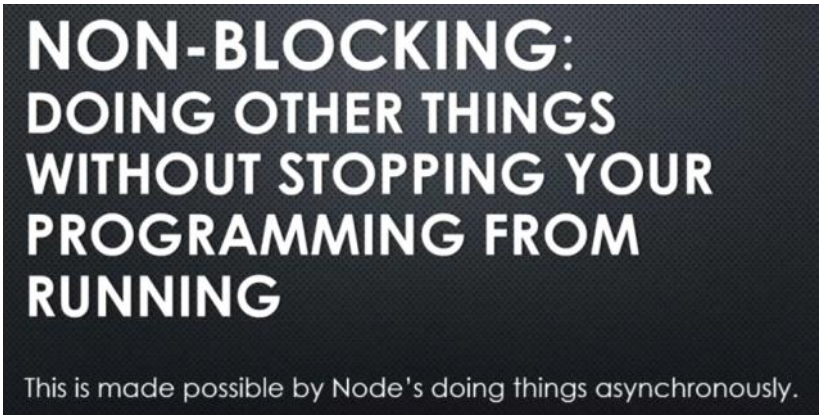
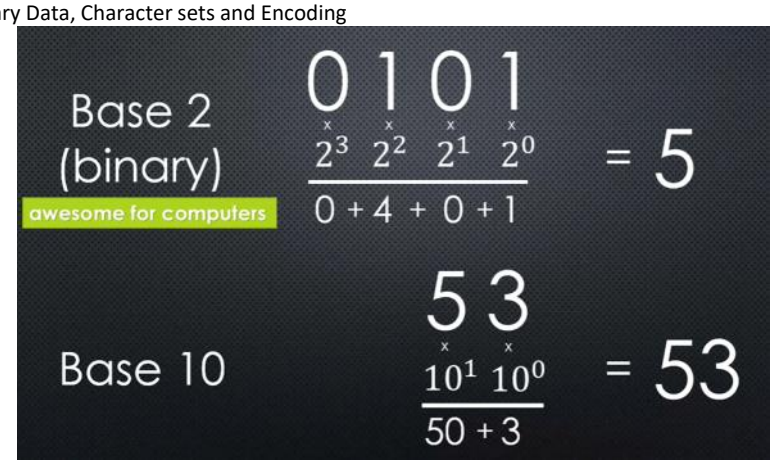
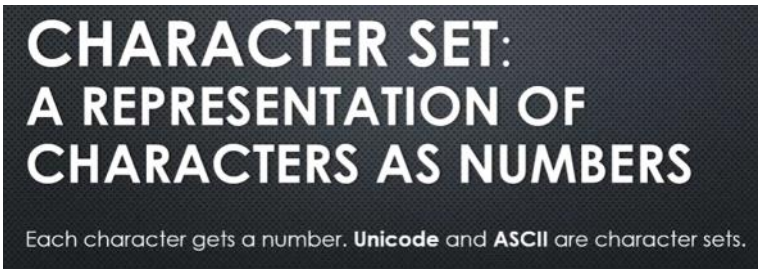
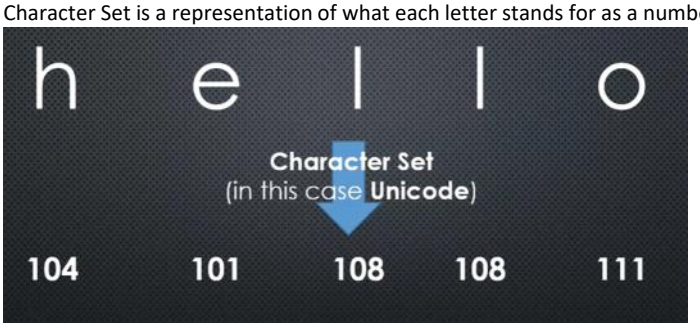
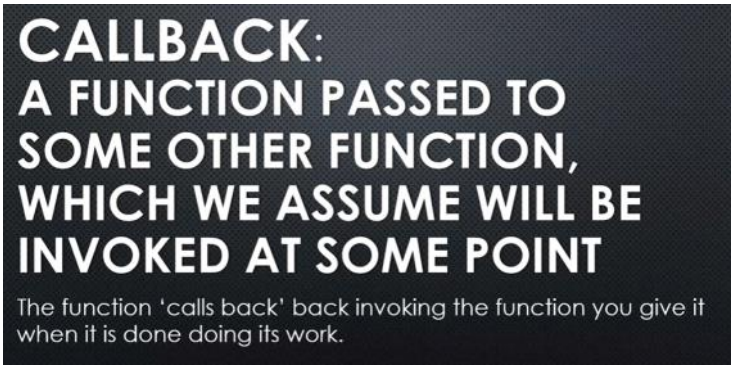
    greet(data) {
        console.log(`${ this.greeting }: ${ data }`);
        this.emit('greet', data);
    }
}

var greeter1 = new Greetr();

greeter1.on('greet', function(data) {
    console.log('Someone greeted!: ' + data);
```

- greetr.js

```
'use strict';

var EventEmitter = require('events');

module.exports = class Greetr extends EventEmitter {
    constructor() {
        super();
        this.greeting = 'Hello world!';
    }

    greet(data) {
        console.log(`${ this.greeting }: ${ data }`);
        this.emit('greet', data);
    }
}
```

- 

- 

- libuv
  - System events in C++ core are handled by libuv which is embedded inside node
  - Inside Node we also have V8
  - libuv specifically is written to handle requests at low-level
    - 

  - libuv connects requests from OS to open files/connect to internet
    - Once these requests are done events are emitted and placed in the queue
    - libuv checks the queue and sees if something is complete ii fires the callback
    - Callback(code that needs to be executed once events are done) generally is handled in js
    - 

  - Things in NodeJs(v* and libuv) as shown run simultaneously as shown above
    - Hence NodeJS is often called Event Driven Non-Blocking I/O in V8 Javascript
    - libuv consists of all .c and .h files
    - In nodejs js runs synchronously and at the same time listens to the event callbacks from V8

- 

- Binary Data, Character sets and Encoding
  - 

  - 

    - Character Set is a representation of what each letter stands for as a number
  - 

  - 

    - Character encoding is the representation of the characters in binary (bits)

- Numbers in base 10 converted to base 2.
- 

- Buffers
  - Holds raw binary data
  - To convert between buffers and string we need some encoding methods like
    - utf-8, base64 etc…
    - default is utf-8
    - Ex.
      - app.js
```
1 var buf = new Buffer('Hello', 'utf8');
2 console.log(buf);
3 console.log(buf.toString());
4 console.log(buf.toJSON());
```
      -
```
D:\Documents\Sandbox\nodejs>node app.js
<Buffer 48 65 6c 6c 6f>
Hello
{ type: 'Buffer', data: [ 72, 101, 108, 108, 111 ] }
```

- ES6 Typed Arrays
  - app.js
```
1 var buffer = new ArrayBuffer(8);
```
    - Based on the above statement, buffer can store 8 bytes of data (64 bits of data)
  - app.js
```
1 var buffer = new ArrayBuffer(8);
2 var view = new Int32Array(buffer);
```
    - As buffer is 64-bit, Int32Array can store 2 32-bit numbers
  - app.js
```
1 var buffer = new ArrayBuffer(8);
2 var view = new Int32Array(buffer);
3 view[0] = 5;
4 view[1] = 15;
5 console.log(view);
```

- Callbacks

- 

- app.js
```
function greet(callback) {
    console.log('Hello!');
    callback();
}

greet(function() {
    console.log('The callback was invoked!');
});
```
- app.js
```
function greet(callback) {
    console.log('Hello!');
    callback();
}

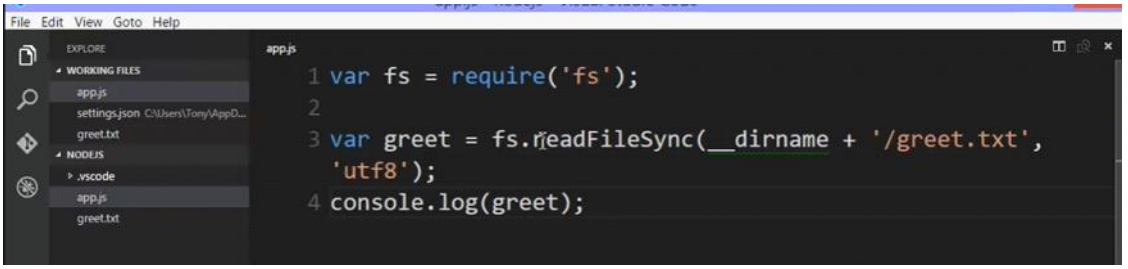
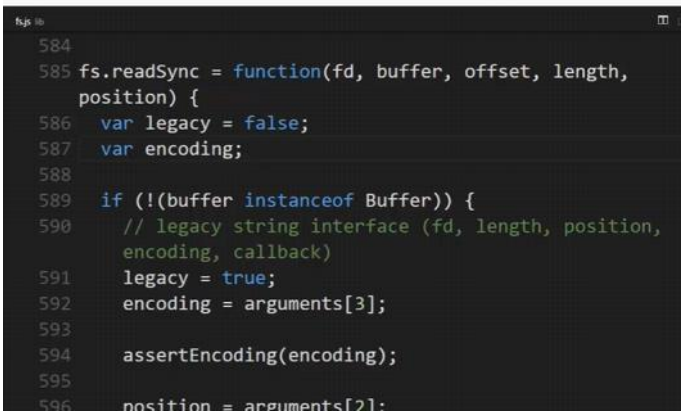
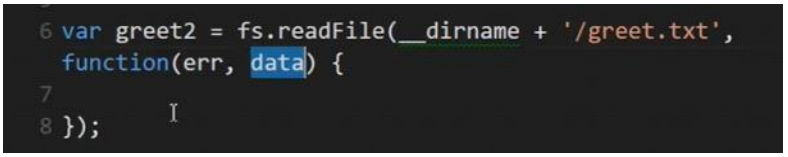
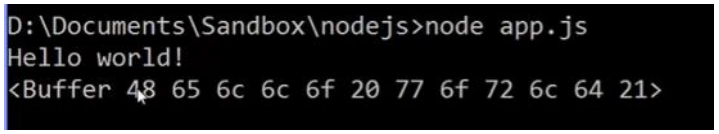
greet(function() {
    console.log('The callback was invoked!');
});

greet(function() {
    console.log('A different callback was invoked!');
});
```
- app.js
```
function greet(callback) {
    console.log('Hello!');
    var data = {
        name: 'John Doe'
    };

    callback(data);
}

```
- ```
greet(function(data) {
    console.log('The callback was invoked!');
    console.log(data);
});

greet(function(data) {
    console.log('A different callback was invoked!');
    console.log(data.name);
});
```

- Files and fs
```

- 
```
File Edit View Goto Help
EXPLORE
WORKING FILES
app.js
settings.json C:\Users\Tony\AppD...
greet.txt
NODEJS
.vscode
app.js
greet.txt
app.js
1 var fs = require('fs');
2
3 var greet = fs.readFileSync(__dirname + '/greet.txt',
  'utf8');
4 console.log(greet);
```
  - In the background of Nodejs, the readsync method in fs.js, when the file is chosen to load, the contents of the file is loaded into the buffer as the buffer can deal with the binary data as shown in the below code
  - 
```
fs.js lib
584
585 fs.readSync = function(fd, buffer, offset, length,
    position) {
586   var legacy = false;
587   var encoding;
588
589   if (!(buffer instanceof Buffer)) {
590     // legacy string interface (fd, length, position,
      encoding, callback)
591     legacy = true;
592     encoding = arguments[3];
593
594     assertEncoding(encoding);
595
596     position = arguments[2];
```
  - readFileSync vs readSync
    - readFileSync will make the code synchronous and waits for the file to be read and till that it is loaded it won`t execute thenext line of code.
    - readFile is asynchronous and other code can be run parallelly. It takes a callback with err and data as arguments to run thecallback function once the file is loaded.
      - 
```
6 var greet2 = fs.readFile(__dirname + '/greet.txt',
  function(err, data) {
7
8 });
```
      - 
```
app.js
1 var fs = require('fs');
2
3 var greet = fs.readFileSync(__dirname + '/greet.txt',
  'utf8');
4 console.log(greet);
5
6 var greet2 = fs.readFile(__dirname + '/greet.txt',
  function(err, data) {
7     console.log(data)
8 });
```
        - 
```
D:\Documents\Sandbox\nodejs>node app.js
Hello world!
<Buffer 48 65 6c 6c 6f 20 77 6f 72 6c 64 21>
```
    - We can include the encoding method in the readFile method as an argument to convert the data as per that encoding mentioned

```
app.js
1 var fs = require('fs');
2
3 var greet = fs.readFileSync(__dirname + '/greet.txt',
  'utf8');
4 console.log(greet);
5
6 var greet2 = fs.readFile(__dirname + '/greet.txt', 'utf8',
   function(err, data) {
7     console.log(data);
8 });
```

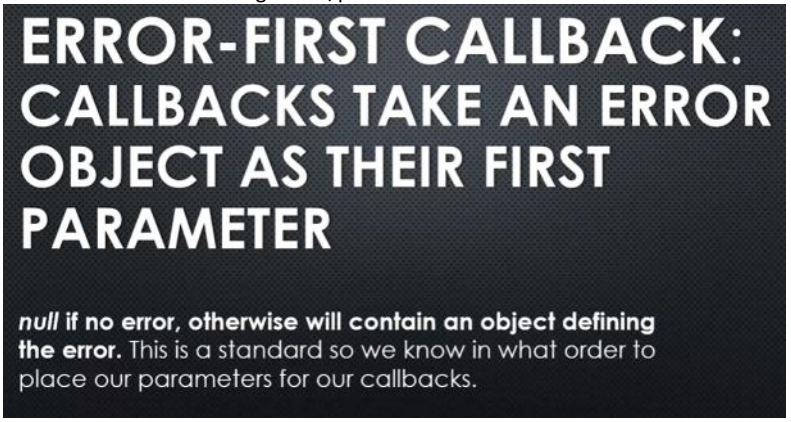
```
D:\Documents\Sandbox\nodejs>node app.js
Hello world!
Hello world!
```

```
app.js
1 var fs = require('fs');
2
3 var greet = fs.readFileSync(__dirname + '/greet.txt',
  'utf8');
4 console.log(greet);
5
6 var greet2 = fs.readFile(__dirname + '/greet.txt', 'utf8',
   function(err, data) {
7     console.log(data);
8 });
9
10 console.log('Done!');
```

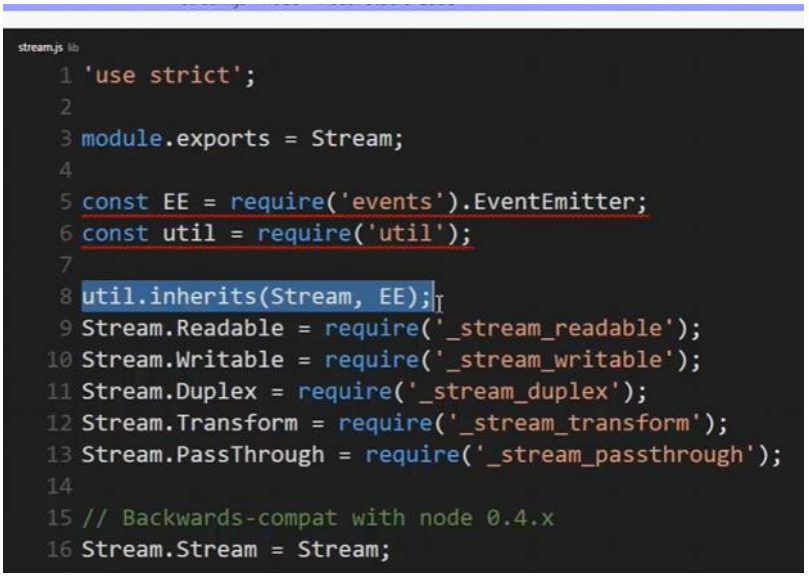
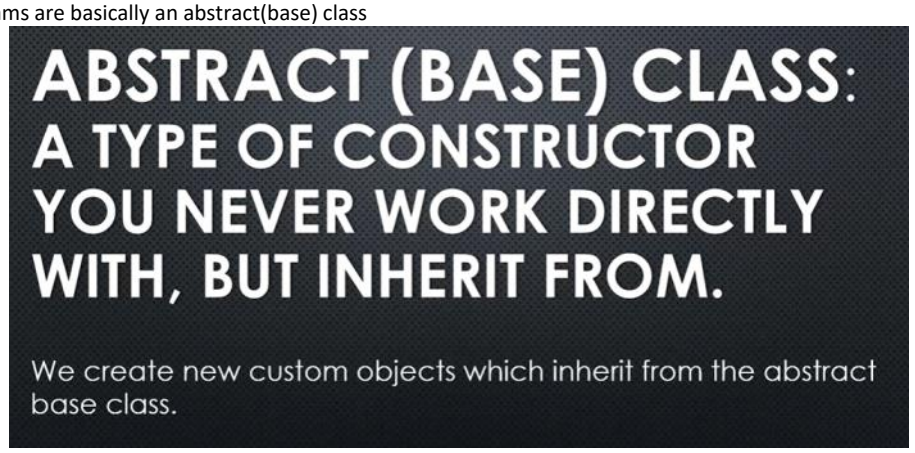
```
D:\Documents\Sandbox\nodejs>node app.js
Hello world!
Done!
Hello world!
```

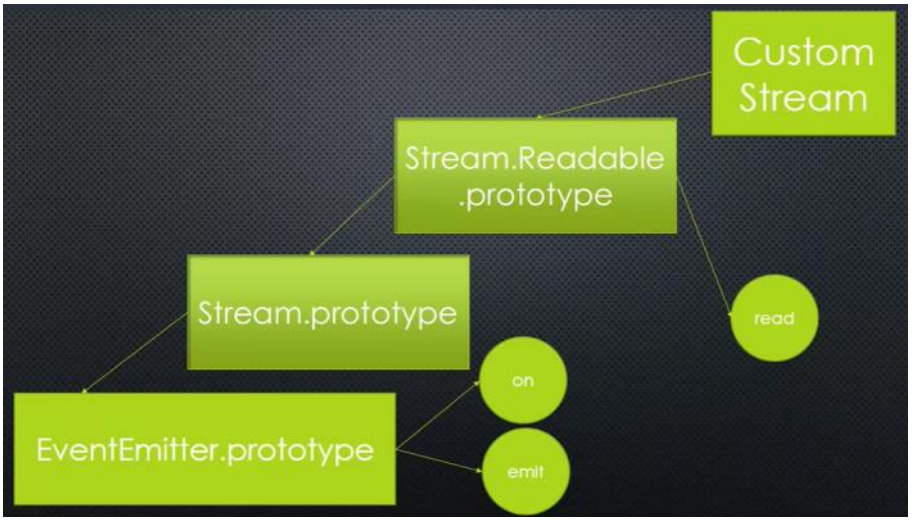
- Buffer content is not stored in memory, it is stored in heap memory of V8
  - Heap memory is where the data the application is using exists
  - So if we are loading/reading too many files, buffer is created for all the files and is stored in heap memory and as it gets overloaded it may create issues for the application
  - So we need to overcome this issue

- Note: NodeJs follows error first callback
  - This means that the first argument/parameter in the callback is related to error or contains the error object



- Streams
  - Streams is sequence of pieces of data that is broken up into what is called as chunks
  - 

  - 
```
stream.js lib
1 'use strict';
2
3 module.exports = Stream;
4
5 const EE = require('events').EventEmitter;
6 const util = require('util');
7
8 util.inherits(Stream, EE);
9 Stream.Readable = require('_stream_readable');
10 Stream.Writable = require('_stream_writable');
11 Stream.Duplex = require('_stream_duplex');
12 Stream.Transform = require('_stream_transform');
13 Stream.PassThrough = require('_stream_passthrough');
14
15 // Backwards-compat with node 0.4.x
16 Stream.Stream = Stream;
```
  - Streams are inherited from event emitters
  - Types of Streams
    - Readable - Only read the data from the stream
    - Writable - Only send the data to the stream
    - Duplex - Read and write data
    - Transform - change the data as it moves through the stream
    - PassThrough
  - Streams are basically an abstract(base) class
  - 

- 

- 

  - In node perspective, the request above sent from browser is readable and the response is writable
- 

```
fs.js lib
1605
1606 fs.createReadStream = function(path, options) {
1607   return new ReadStream(path, options);
1608 };
1609
1610 util.inherits(ReadStream, Readable);
1611 fs.ReadStream = ReadStream;
1612
1613 function ReadStream(path, options) {
1614   if (!(this instanceof ReadStream))
1615     return new ReadStream(path, options);
1616
1617   if (options === undefined)
1618     options = {};
1619   else if (typeof options === 'string')
1620     options = { encoding: options };
1621   else if (options === null || typeof options !==
```

  - In the above code we can see that ReadStream is inherited from Readable, Readable is inherited from Streams and Streams is inherited from Event Emitters. So, we can say that ReadStream is a kind of Event Emitter

- Readable stream example

app.js

```
1 var fs = require('fs');
2
3 var readable = fs.createReadStream(__dirname +
  '/greet.txt', { encoding: 'utf8', highWaterMark: 16 *
  1024 });
4
5 readable.on('data', function(chunk) {
6     console.log(chunk.length);
7 });
```

- By default the buffer size is 64kb
- We can specify the buffer size by using highWaterMark

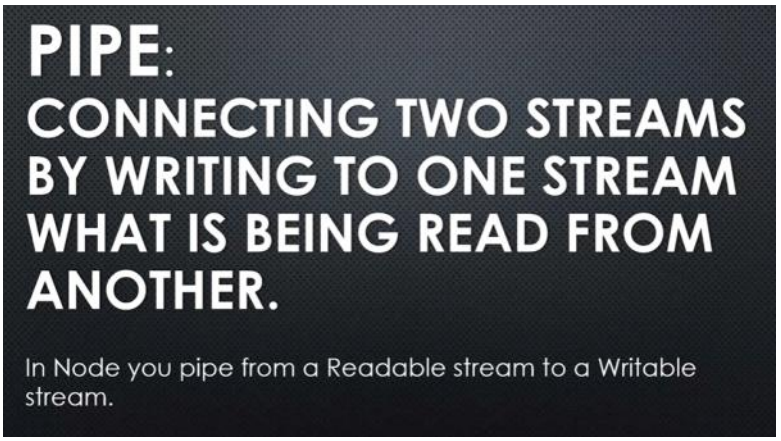
```
D:\Documents\Sandbox\nodejs>node app.js
16384
16384
16384
12456
```

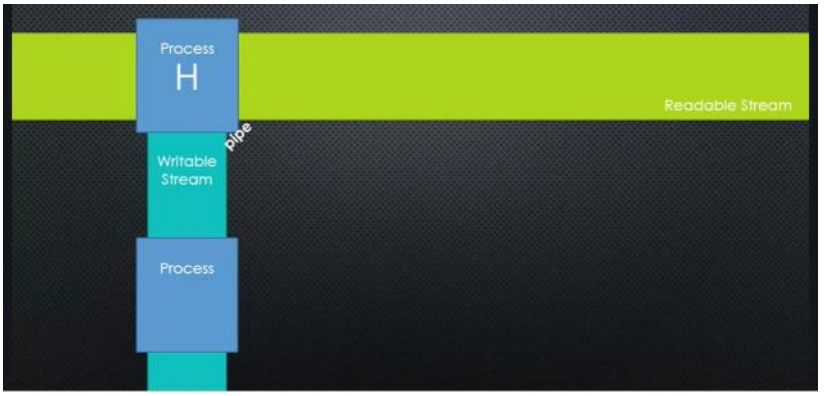
- Writable stream example

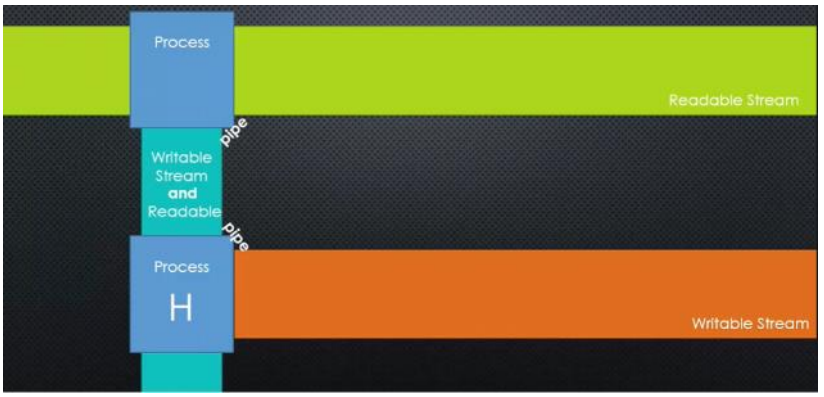
app.js

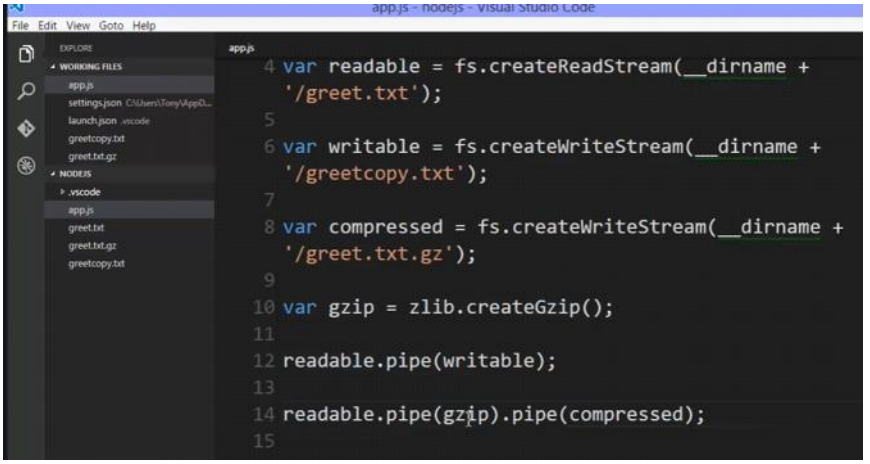
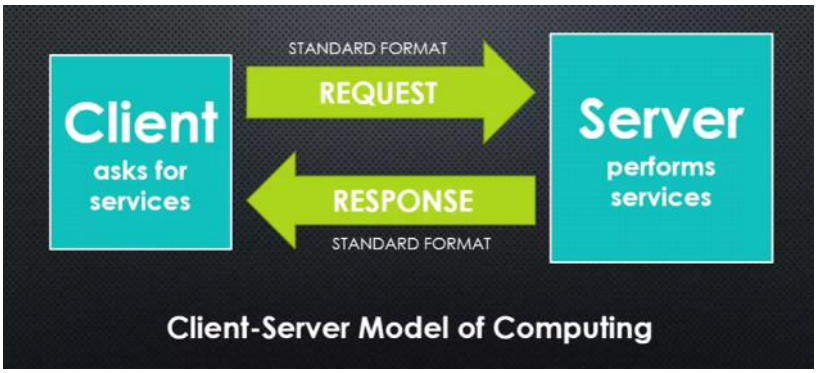
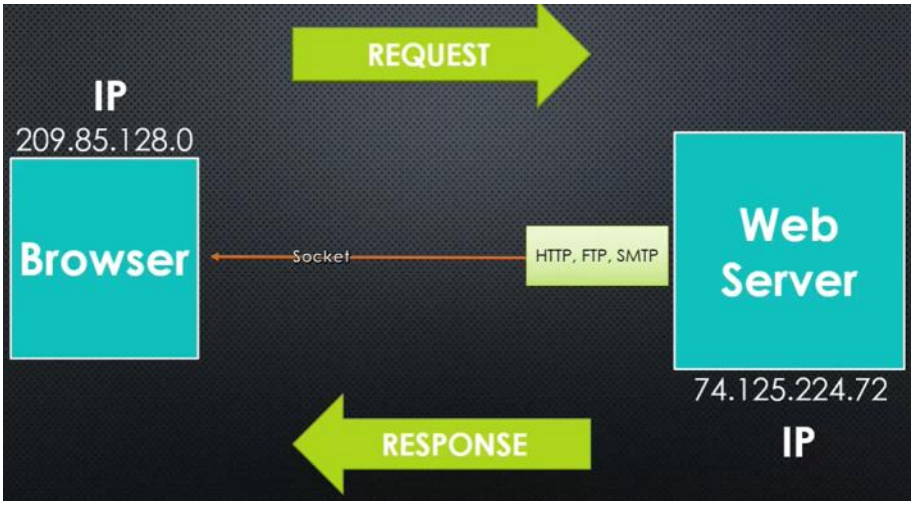
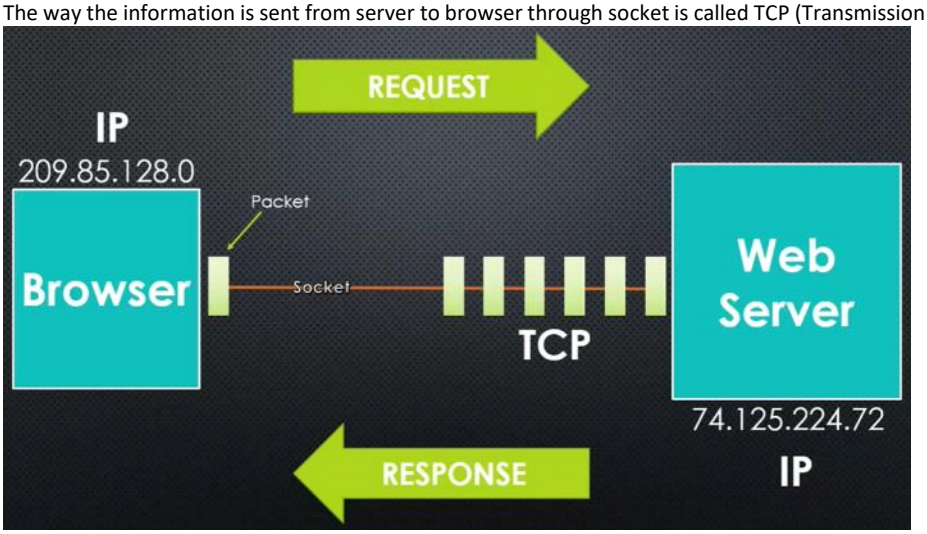
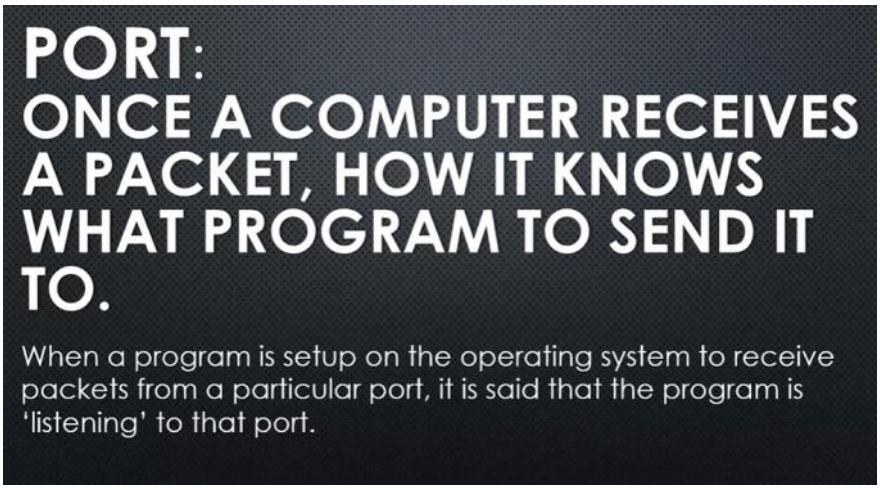
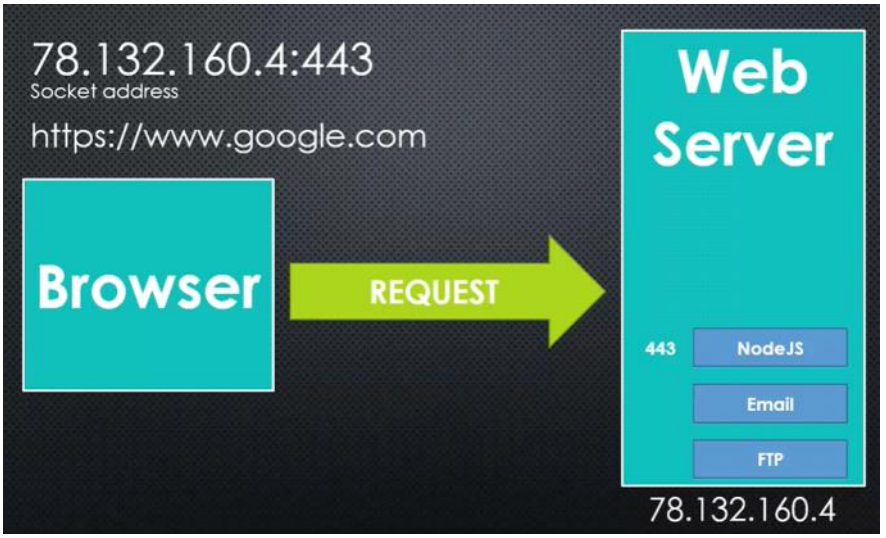
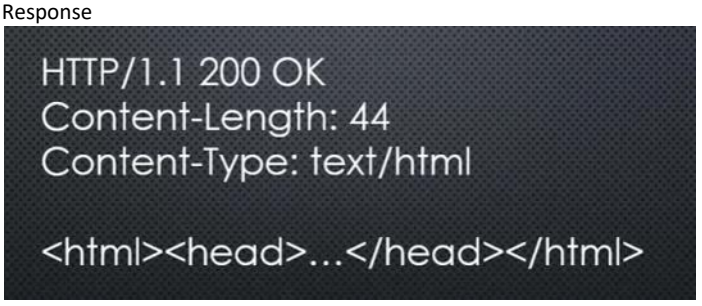
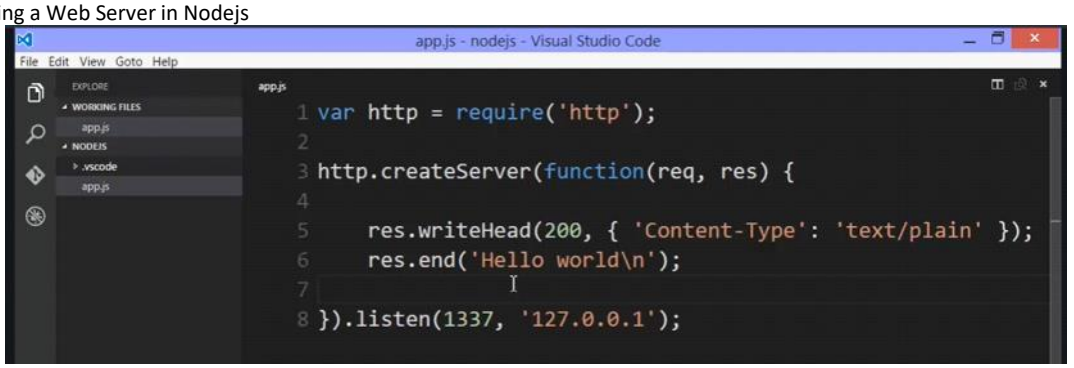
```
1 var fs = require('fs');
2
3 var readable = fs.createReadStream(__dirname +
  '/greet.txt', { encoding: 'utf8', highWaterMark: 16 *
  1024 });
4
5 var writable = fs.createWriteStream(__dirname +
  '/greetcopy.txt');
6
7 readable.on('data', function(chunk) {
8     console.log(chunk);
9     writable.write(chunk);
10 });
```

- Pipes



- 

- 

- ```
  Readable.prototype.pipe = function(dest, pipeOpts) {
    var src = this;
    var state = this._readableState;

    switch (state.pipesCount) {
      case 0:
        state.pipes = dest;
        break;
      case 1:
        state.pipes = [state.pipes, dest];
  ```
- It accepts a param dest where the read data is stored into and returned
- app.js
  ```
  var fs = require('fs');

  var readable = fs.createReadStream(__dirname +
  '/greet.txt');

  var writable = fs.createWriteStream(__dirname +
  '/greetcopy.txt');

  var readable: any
  readable.pipe(writable);

  ```
  - In the above example it reads from greet.txt and writes into greetcopy using pipe
- zlib
  - This is part of node core.
  - Allows us to implement a z-zip file
  - z-zip is a particular and common algorithm for compressing files and node has built in component zlib to do that
  ```

- 

  - As written in the above example
    - We store the compressed version of the greet.txt file in greet.txt.gz
    - readable.pipe(gzip) -> pipe will read from greet.txt and return writable stream and gzip will transform that chunk and return the readable stream
    - readable.pipe(gzip).pipe(compressed) -> the output of first pipe then returns the transformed stream which is readable and writes to compressed (greet.txt.gz)
  - Asynchronous and streams implementation will improve the performance

- Building a web server
  - TCP/IP
  - 

  - 

  - How do client and server identify each other?
    - IP (Internet Protocol)
    - Socket (Line through which internet flows)

- 

- The way the information is sent from server to browser through socket is called TCP (Transmission Control Protocol)
- 

- Information that we are sending in whatever format/protocol will be taken by TCP and split in the form of packets and send it to the other side along the line.
- Node gives us the ability to talk to the OS (OS will have the capability of TCP/IP) to use the features of the OS.
  - Ability to create a socket to make the network connection to send the information in the form of packets through TCP/IP
  - So, In Node we can define the information that we would like to send
- With internet we open and close sockets constantly
  - We ask for information, receive it and the socket is closed
  - Then we make another request, click a link, we ask to download an image new sockets are opened.
  - The concept of web sockets is to keep the socket open constantly so that the client and the server can keep sending information to each other whenever they need.

- Addresses and Ports
  - 

  - Port is to map the program running on the server when there are multiple programs running in the server.

- 

- HTTP
  - 

  - HTTP Headers
    - 

  - HTTP Response
    - 

    - 

- 

    - HTTP Parser
        - Used in _http_server.js in the node library where node handles the status codes, headers, messages for the response and request

- Building a Web Server in Nodejs
    - 

    - http server response is a writable stream when we can write our data to be passes to the client

- Outputting HTML and Templates
    - 

- 

- app.js
```
var http = require('http');
var fs = require('fs');

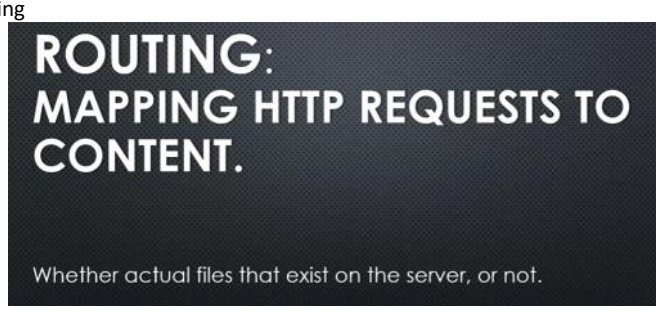
http.createServer(function(req, res) {

    res.writeHead(200, { 'Content-Type': 'text/html' });
    var html = fs.readFileSync(__dirname + '/index.htm',
    'utf8');
    var message = 'Hello world...';
    html = html.replace('{Message}', message);
    res.end(html);

}).listen(1337, '127.0.0.1');
```
- index.htm
```
<html>
    <head></head>
    <body>
        <h1>{Message}</h1>
    </body>
</html>
```

- Streams and Performance
  - createReadStream is to read the data in parts known as chunks
  - Every chunk of data read from the file can be piped to a writeable stream, this increases the performance and application will be more responsive
  - Streams are the fundamentals on how the internet works
- app.js
```
var http = require('http');
var fs = require('fs');

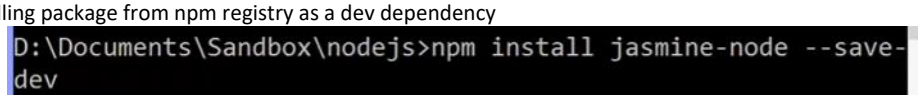
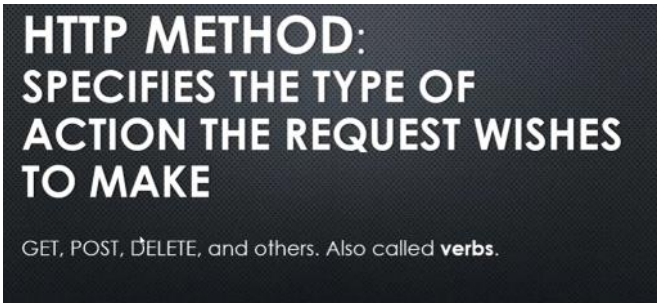
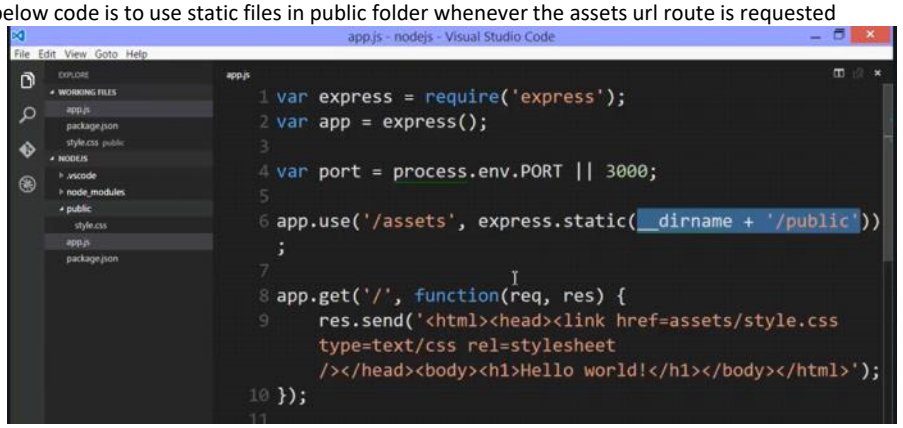
http.createServer(function(req, res) {

    res.writeHead(200, { 'Content-Type': 'text/html'
    fs.createReadStream(__dirname + '/index.htm').pipe(res);

}).listen(1337, '127.0.0.1');
```

- API's and Endpoints
  - 

  - 

- Outputting JSON
  - 

- Routing
  - 

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

http.createServer(function(req, res) {

    if (req.url === '/') {
        fs.createReadStream(__dirname + '/index.htm').pipe
        (res);
    }
```
-
```
    else if (req.url === '/api') {
        res.writeHead(200, { 'Content-Type':
        'application/json' });
        var obj = {
            firstname: 'John',
            lastname: 'Doe'
        };
        res.end(JSON.stringify(obj));
    }
    else {
        res.writeHead(404);
        res.end();
    }

}).listen(1337, '127.0.0.1');
```

- NPM - Node Package Manager
  - Semantic Versioning
    - 

    - 

  - NPM and NPM Registry
    - To check if npm is installed -> npm -v
    - npm registry -> www.npmjs.com
    - Details of an package in npm registry
      - 

      - 

- init, nodemon and package.json
  - npm init -> to initialize an project with package.json
    - 

  - Installing a library as a dependency from npm registry into your project
    - npm install moment --save
    - 

    - ^2.10.6 - Use any latest version from 2.10.6 onwards
    - ~2.10.6 - Use exact version 2.10.6
  - 

  - Installing package from npm registry as a dev dependency
    - 

  - Installing package globally from npm registry so that it can be used in all projects
    - npm install -g nodemon
    - nodemon -> watch tool which will look for changes in code and build and run the code again

- Express
  - npm i --save express
  - 

-
```
app.js
1 var express = require('express');
2 var app = express();
3
4 var port = process.env.PORT || 3000;
5
6 app.get('/', function(req, res) {
7     res.send('<html><head></head><body><h1>Hello
      world!</h1></body></html>');
8 });
9
10 app.get('/api', function(req, res) {
11     res.json({ firstname: 'John', lastname: 'Doe' });
12 });
13
14 app.listen(port);
```
- 

- Routing
  - Adding params to routing
    -
```
10 app.get('/person/:id', function(req, res) {
11     res.send('<html><head></head><body><h1>Person: ' +
       req.params.id + '</h1></body></html>');
12 });
```
- Static Files and Middleware
  - 

  - The below code is to use static files in public folder whenever the assets url route is requested
    - 

  - we can write our custom code using middleware for a particular route as shown below

- ```
  var express = require('express');
  var app = express();

  var port = process.env.PORT || 3000;

  app.use('/assets', express.static(__dirname + '/public'))
  ;

  app.use('/', function(req, res, next) {
      console.log('Request Url: ' + req.url);
      next();
  });
  ```
- next() is used invoke the next middleware
- Middleware examples are app.get(), app.use()

○ Third-party middleware example

- ```
  var express      = require('express')
  var cookieParser = require('cookie-parser')

  var app = express()
  app.use(cookieParser())
  ```

- Templates and Template Engines
  - View Engines
    - jade

- ```
  app.set('view engine', 'jade');
  ```

  Create a Jade template file named "index.jade" in the

  ```
  html
    head
      title!= title
    body
      h1!= message
  ```

    - EJS

- Example

  ```
  <% if (user) { %>
    <h2><%= user.name %></h2>
  <% } %>
  ```

  Usage

  ```
  var template = ejs.compile(str, options);
  template(data);
  // => Rendered HTML string

  ejs.render(str, data, options);
  // => Rendered HTML string
  ```
  ```

-
```
<html>
    <head>
        <link href="assets/style.css" type="text/css"
        rel="stylesheet" />
    </head>
    <body>
        Hello world!
    </body>
</html>
```
-
```
app.set('view engine', 'ejs');

app.use('/', function (req, res, next) {
    console.log('Request Url:' + req.url);
    next();
});

app.get('/', function(req, res) {
    res.render('index');
});
```
- We can display an output in ejs by using <%= <property-name> %>
  -
```
<html>
    <head>
        <link href="/assets/style.css" type="text/css"
        rel="stylesheet" />
    </head>
    <body>
        <h1>Person: <%= ID %></h1>
    </body>
</html>
```
  -
```
app.set('view engine', 'ejs');

app.use('/', function (req, res, next) {
    console.log('Request Url:' + req.url);
    next();
});

app.get('/', function(req, res) {
    res.render('index');
});

app.get('/person/:id', function(req, res) {
    res.render('person', { ID: req.params.id });
});
```

- Query String and Post params
  -
```
GET /?id=4&page=3 HTTP/1.1
Host: www.learnwebdev.net
Cookie: username=abc;name=Tony
```

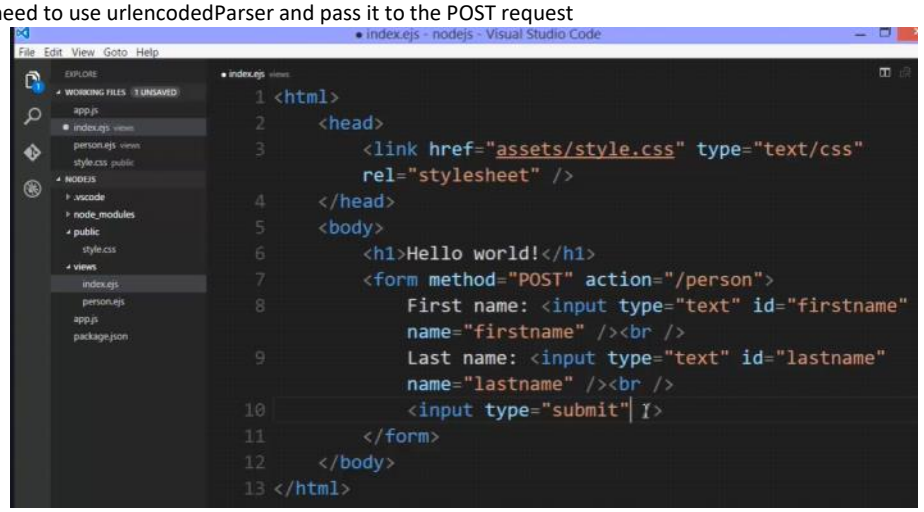
-
```
POST / HTTP/1.1
Host: www.learnwebdev.net
Content-Type: application/x-www-form-urlencoded
Cookie: num=4;page=2

username=Tony&password=pwd
```
-
```
POST / HTTP/1.1
Host: www.learnwebdev.net
Content-Type: application/json
Cookie: num=4;page=2

{
   "username": "Tony",
   "password": "pwd"
}
```

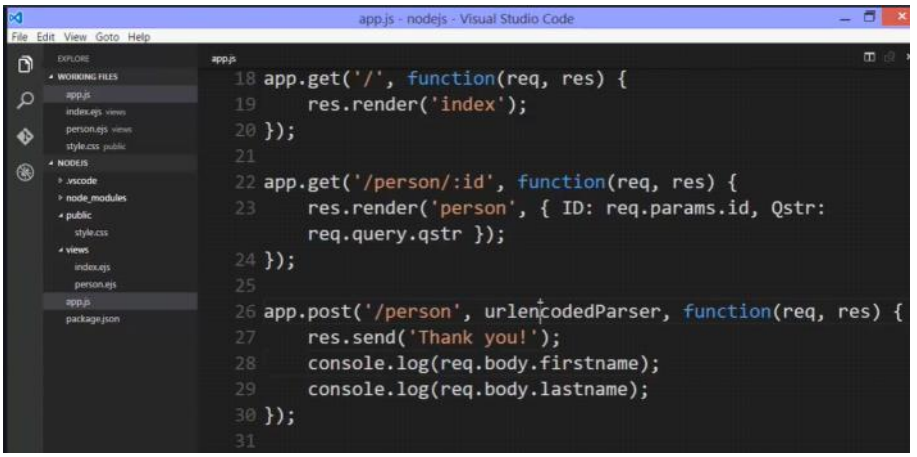
- Query String
  -
```
app.get('/', function(req, res) {
    res.render('index');
});

app.get('/person/:id', function(req, res) {
    res.render('person', { ID: req.params.id, Qstr:
    req.query.qstr });
});
```
  -
```
<html>
    <head>
        <link href="/assets/style.css" type="text/css"
        rel="stylesheet" />
    </head>
    <body>
        <h1>Person: <%= ID %></h1>
        <h2>Querystring Value: <%= Qstr %></h2>
    </body>
</html>
```

- Post parameters
  - We need third-party middleware for body parsing
  - npm i body-parser --save

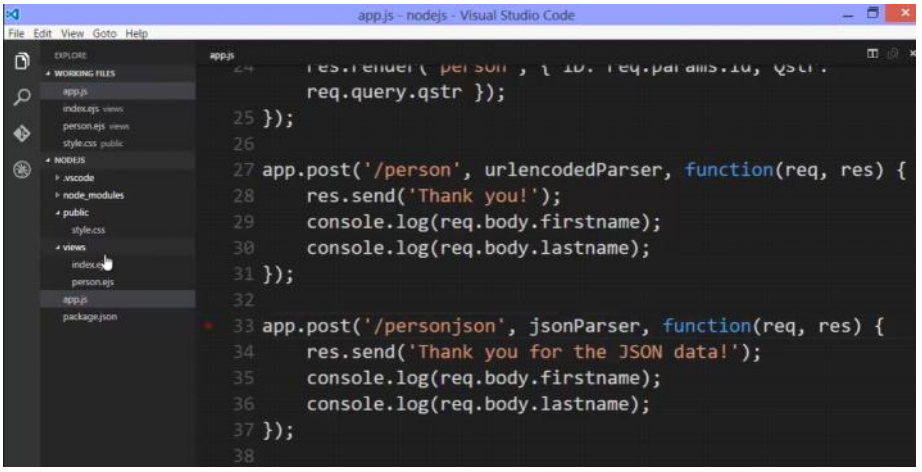
```
var express = require('express')
var bodyParser = require('body-parser')

var app = express()

// create application/json parser
var jsonParser = bodyParser.json()

// create application/x-www-form-urlencoded parser
var urlencodedParser = bodyParser.urlencoded({ extended: false })

// POST /login gets urlencoded bodies
app.post('/login', urlencodedParser, function (req, res) {
  if (!req.body) return res.sendStatus(400)
  res.send('welcome, ' + req.body.username)
})

// POST /api/users gets JSON bodies
app.post('/api/users', jsonParser, function (req, res) {
  if (!req.body) return res.sendStatus(400)
  // create user in req.body
})
```

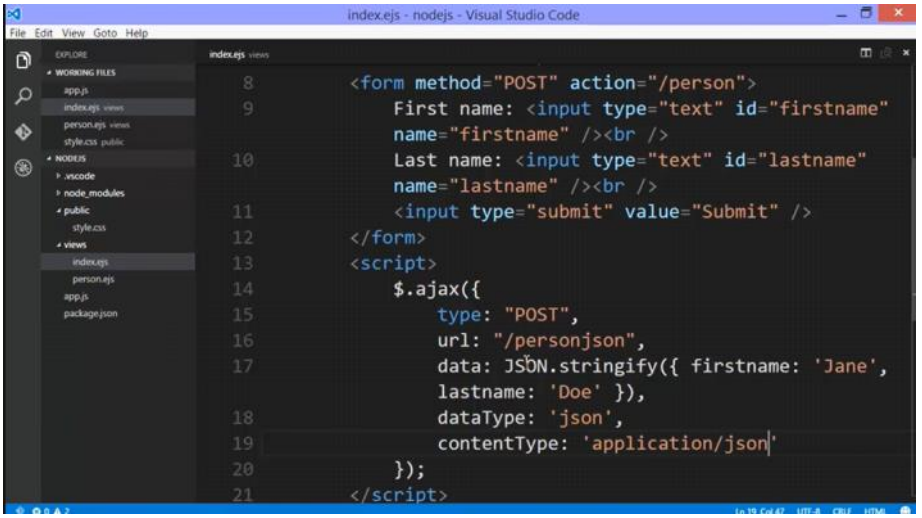
- We need to use urlencodedParser and pass it to the POST request





- We need to use jsonParser for parsing json data
  - var jsonParser = bodyParser.json();

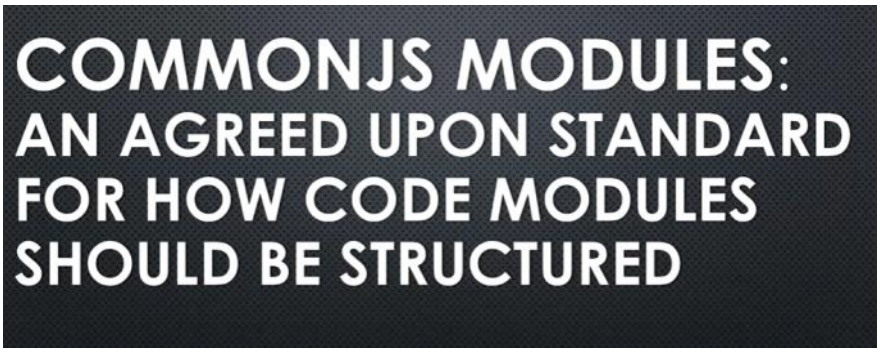

app.js - nodejs - Visual Studio Code
File Edit View Goto Help
app.js
req.query.qstr });
});
app.post('/person', urlencodedParser, function(req, res) {
res.send('Thank you!');
console.log(req.body.firstname);
console.log(req.body.lastname);
});
app.post('/personjson', jsonParser, function(req, res) {
res.send('Thank you for the JSON data!');
console.log(req.body.firstname);
console.log(req.body.lastname);
});

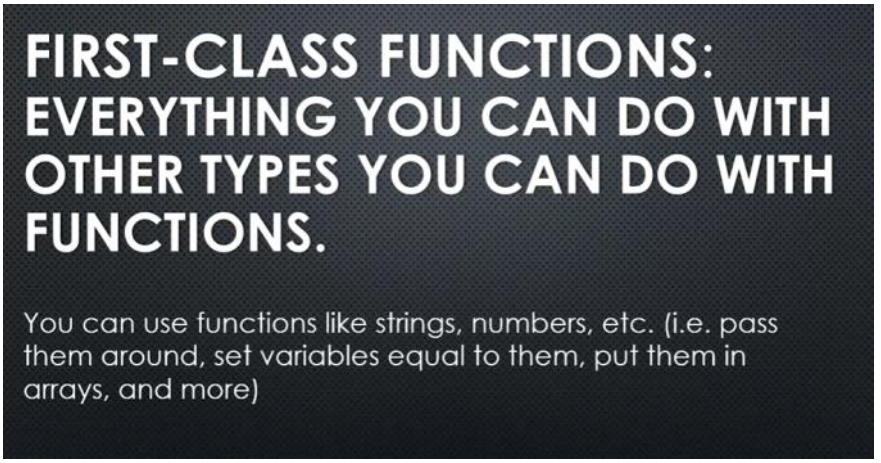

index.ejs - nodejs - Visual Studio Code
File Edit View Goto Help
index.ejs
<form method="POST" action="/person">
First name: <input type="text" id="firstname"
name="firstname" />
Last name: <input type="text" id="lastname"
name="lastname" />
<input type="submit" value="Submit" />
</form>
<script>
$.ajax({
type: "POST",
url: "/personjson",
data: JSON.stringify({ firstname: 'Jane',
lastname: 'Doe' }),
dataType: 'json',
contentType: 'application/json'
});
</script>

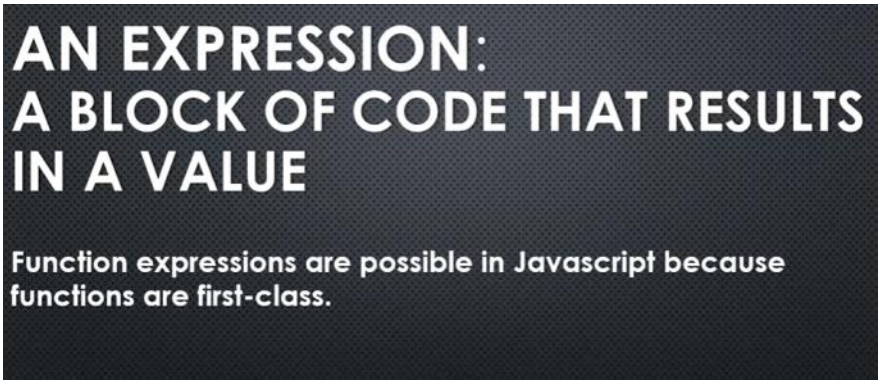
Javascript aside

Thursday, June 25, 2020 5:05 PM

- Modules, Exports and Require
  - 

  - 

  - 

  - 

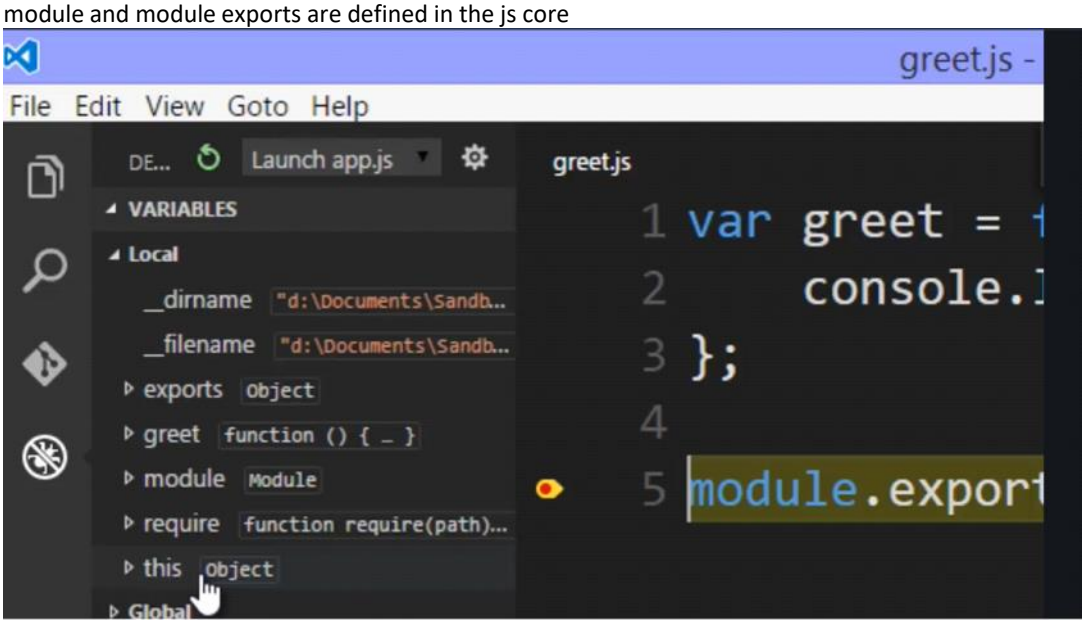
-
```
// function statement
function greet() {
    console.log('hi');
}
greet();

// functions are first-class
function logGreeting(fn) {
    fn();
}
logGreeting(greet);
```
-
```
// functions are first-class
function logGreeting(fn) {
    fn();
}
logGreeting(greet);

// function expression
var greetMe = function() {
    console.log('Hi Tony');
}
greetMe();

// it's first-class
logGreeting(greetMe);
```
-
```
// use a function expression on the fly
logGreeting(function() {
    console.log('Hello Tony!');
});
```
- Modules
  - Modules(greet.js) referred as require in app.js can be compiled from app.js but we cannot use the functions in the greet.js
  - greet.js
```
1 var greet = function() {
2     console.log('Hello!');
3 };
```
  - app.js
```
1 require('./greet.js');
2 greet();
```
  - To use the methods of the greet module in app.js, we can export the methods that can be used in greet.js

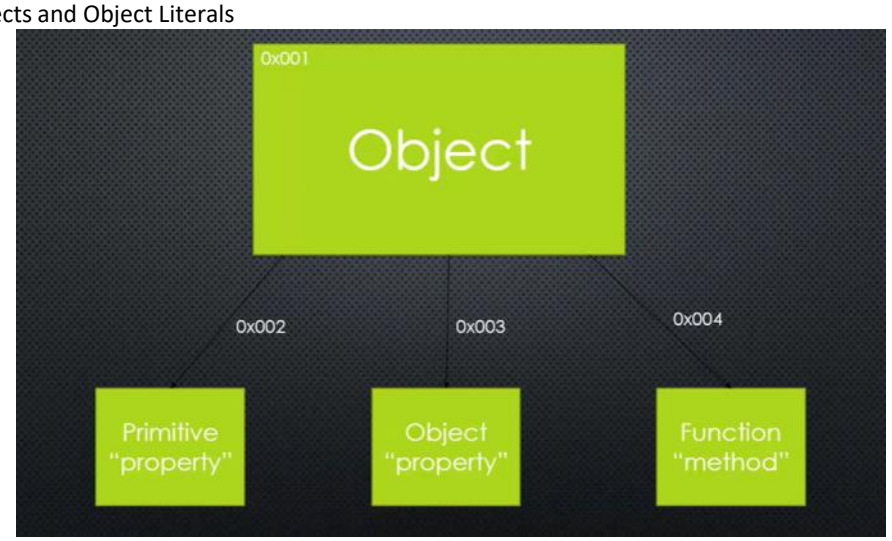
- greet.js

```
1 var greet = function() {
2     console.log('Hello!');
3 };
4
5 module.exports = greet;
```

- app.js

```
1 var greet = require('./greet.js');
2 greet();
```

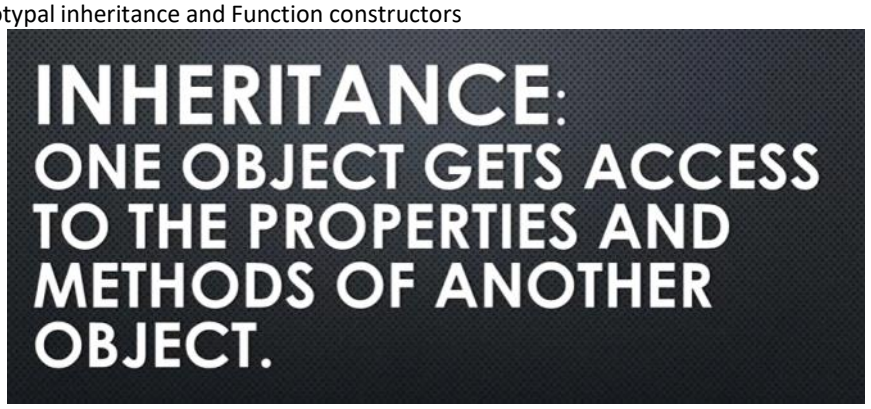
- module and module exports are defined in the js core
- 

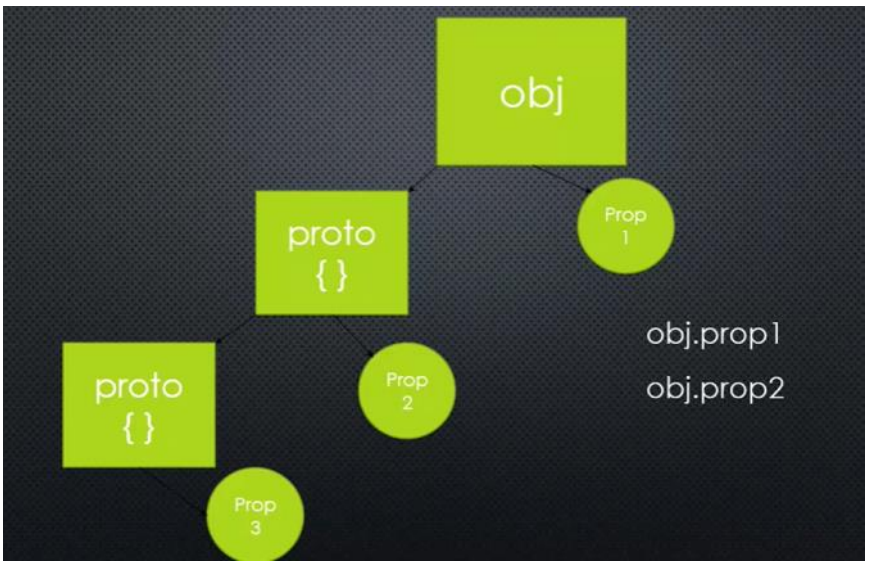
- Objects and Object Literals
  - 

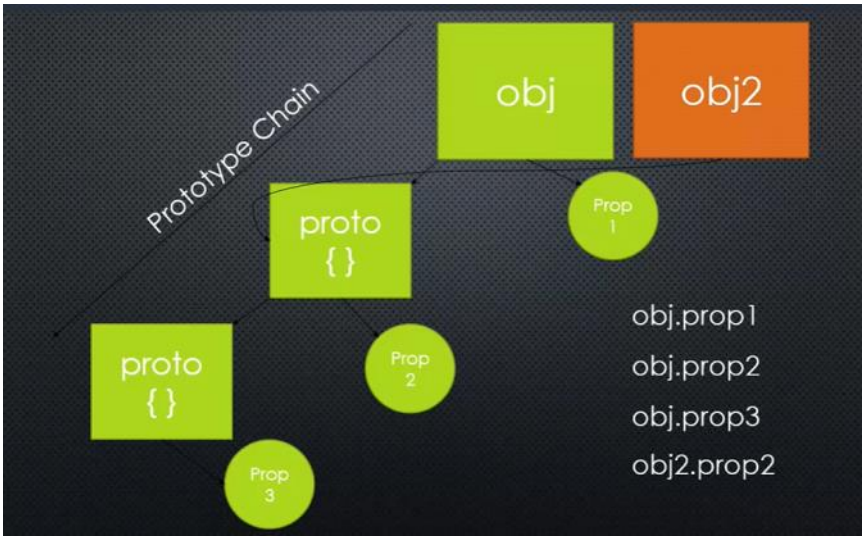
  - Object Literals

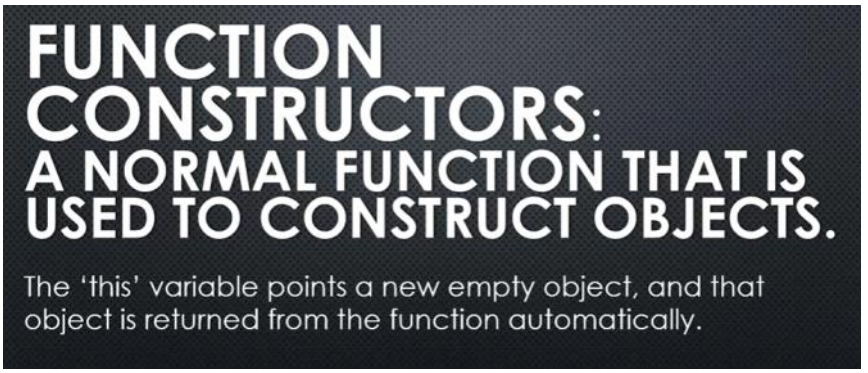
-

```
app.js
1 var person = {
2     firstname: 'John',
3     lastname: 'Doe',
4     greet: function() {
5         console.log('Hello, ' + this.firstname + ' ' +
          this.lastname);
6     }
7 };
8
9 person.greet();
10
11 console.log(person['firstname']);
```

- Prototypal inheritance and Function constructors
  - 

  - 

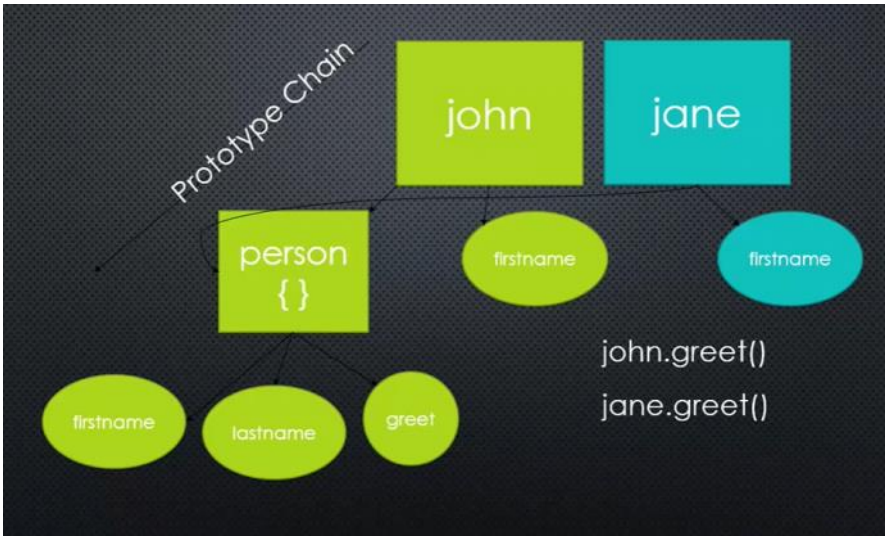




app.js

```
1 function Person(firstname, lastname) {
2
3     this.firstname = firstname;
4     this.lastname = lastname;
5
6 }
7
8 Person.prototype.greet = function() {
9     console.log('Hello, ' + this.firstname + ' ' +
      this.lastname);
10 };
11
12 var john = new Person('John', 'Doe');
13 john.greet();
```

```
12 var john = new Person('John', 'Doe');
13 john.greet();
14
15 var jane = new Person('Jane', 'Doe');
16 jane.greet();
17
18 console.log(john.__proto__);
19 console.log(jane.__proto__);
20 console.log(john.__proto__ === jane.__proto__);
```

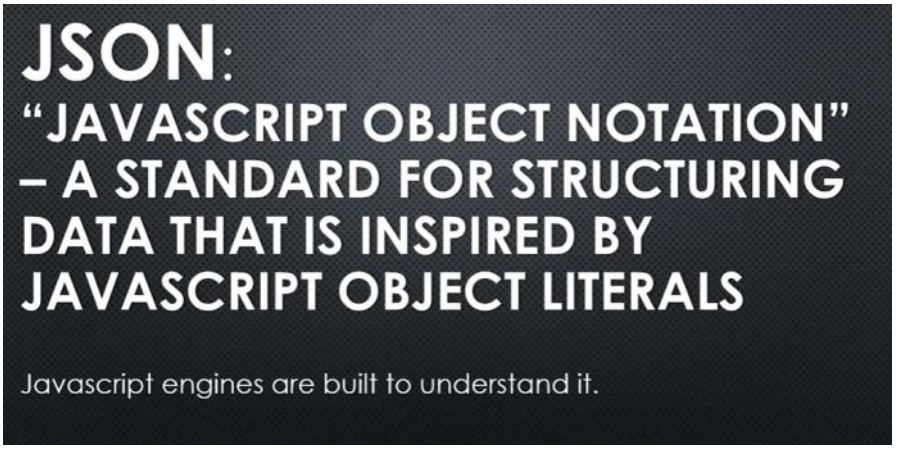


- IIFE

app.js

```
1 var firstname = 'Jane';
2
3 (function (lastname) {
4
5     var firstname = 'John';
6     console.log(firstname);
7     console.log(lastname);
8
9 }('Doe'));
10
11 console.log(firstname);
```

- JSON



-
```
{
   "firstname": "John",
   "lastname": "Doe",
   "address": {
      "street": "101 Main St.",
      "city": "New York",
      "state": "NY"
   }
}
```

- Object Properties and Methods
  -
```
app.js
1 // object properties and methods
2 var obj = {
3     greet: 'Hello'
4 }
5
6 console.log(obj.greet);
7 console.log(obj['greet']);
8 var prop = 'greet';
9 console.log(obj[prop]);
```

- Functions and Arrays
  - Functions are first class in js
  -
```
11 // functions and arrays
12 var arr = [];
13
14 arr.push(function() {
15     console.log('Hello world 1');
16 });
17 arr.push(function() {
18     console.log('Hello world 2');
19 });
20 arr.push(function() {
21     console.log('Hello world 3');
22 });
23
24 arr.forEach(function(item) {
25     item();
26 });
```

- ES6
  - 

- We can write code in ES6 and convert to older js using babeljs.io

- 

- app.js
```
1 var name = 'John Doe';
2
3 var greet = 'Hello ' + name;
4 var greet2 = `Hello ${ name }`;
5
6 console.log(greet);
7 console.log(greet2);
```

- Call and Apply
  - app.js
```
1 var obj = {
2     name: 'John Doe',
3     greet: function(param) {
4         console.log(`Hello ${ this.name }`);
5     }
6 }
7
8 obj.greet();
9 obj.greet.call({ name: 'Jane Doe'});
10 obj.greet.apply({ name: 'Jane Doe'});
```

  - same as function calls but we apply the context this when using call and apply
  - Only difference between call and apply is call accepts arguments separated by comma and apply takes arguments as an array

VSCode Extensions

Monday, August 3, 2020 3:38 PM

VS Code Plugins
Click on extension marketplace on the left tab in VS Code and search for following Plugins and install

1. **Bracket Pair Colorizer**
2. **Code Coverage**
3. **Coverage Gutters**
4. **ESlint**
5. **Jest Test Explorer**
6. **Path Intellisense**
7. **Test Explorer UI**
8. **Visual Studio IntelliCode**

nodemon - For live tracking changes in code

Training

Tuesday, August 4, 2020 7:53 PM

Day 1

jest for unit testing
istanbul for code coverage statistics

for code standardization -> npx eslint --init
enable rules to standardize your application
for rules go through eslint website

Day 2
Modules - Modularizing our code, Encapsulating the code
Tree shaking - removing dead code from our application, use de-structuring of exported module objects wherever possible
Events and Event Emitter
file system -

- fs.readFile
- fs.writeFile
- fs.mkdir
- fs.exists
- __dirname - to get path of the folder where the current file resides
- ./ - gives current working directory

util module

- we can use promisify for using promises in nide
- we can also use async .. await

Day 3
Callbacks
Promises - Node offers util.promisify to use promises, bluebird
Async/Await
Worker threads - Node provides worker_threads module for multi-threading, Used for mostly CPU extensive operations

Command Line Arguments
readLine -

```
var readInterface = readLine.createInterface({
        input: process.stdin,
        output: process.stdout
)};
```

```
readInterface.question('How are you?', function(ans) {
        console.log(ans);
        readInterface.close();
});
```

PromptSync -

yargs -

commander -

Colorful output-

Recommended -> yargs, commander

Day 4

nodemon - for live changes tracking without starting and stopping node server
express js -

```
const express = require('express');
let myApp = express();
myApp.get('/').
```

```
nodemon --exec babel-node app
```

js - ES6 features
- async/await
- destructuring
- spread … rest

using es6 in nodejs by installing babel

Day 5
Server
- body-parser
- app.use(bodyParser.urlencoded(extended: this))

Router
- app.all('/', (req, res, next) => {
  req.send('All things');
  })
- String patterns -> Route(/) can be anything, we can also serve /random.txt / /ab?cd
  - Regex can also be used
- Route params ->

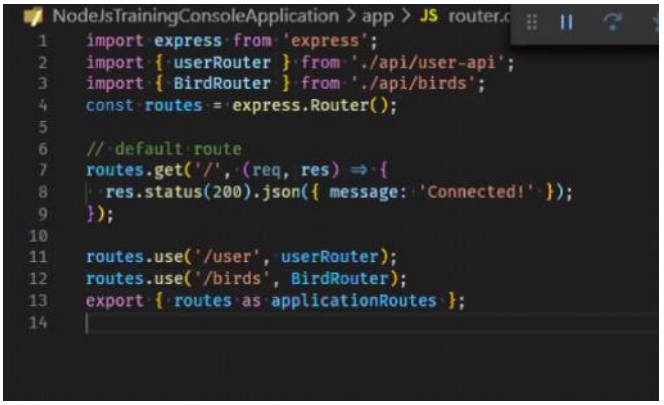
-
```
Request URL http://localhost:4100/users/34/books/2222

Route path : /users/:userId/books/:bookID

req.params :{"userId":"34","bookID":"2222"}
```

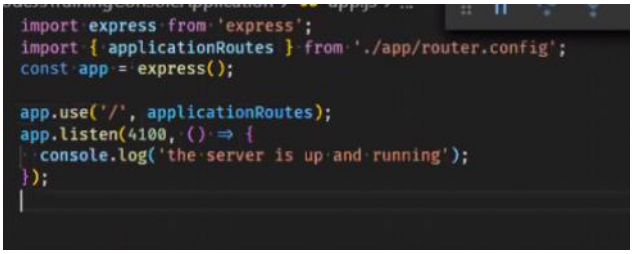
-
```
import express from 'express';
import bodyParser from 'body-parser';
const app = express();

app.use(bodyParser.urlencoded({ extended: this }));

app.all('/shivansh/', (req, res, next) => {
  res.send('ab?cd');
});

app.get('/users/:userId/books/:bookId', (req, res) => {
  res.status(200).send(req.params);
});
app.listen(4100, () => {
  console.log('The server is up and running');
});

// Find out Josn parser with req.body using body parser

//
```

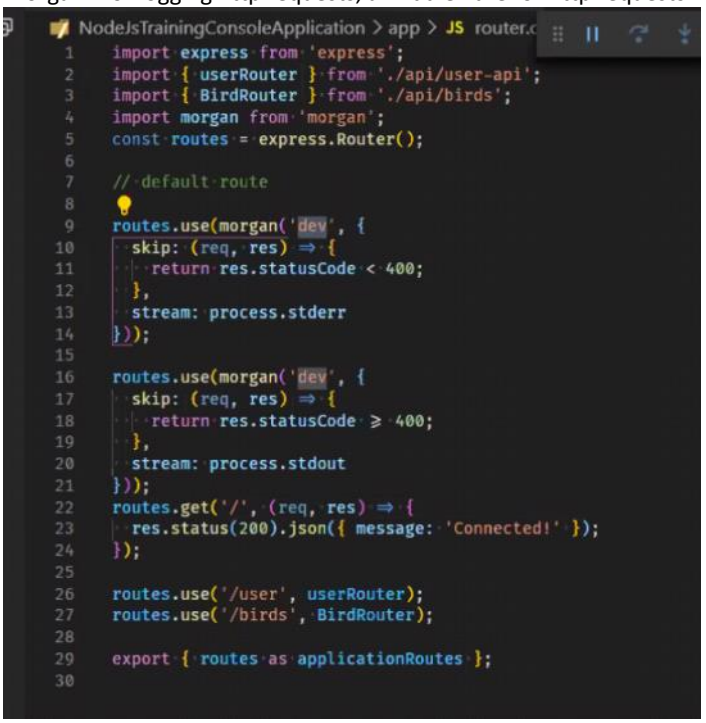
- Route Handlers
  - we can handle by using next param
  -
```
app.get('/users/:userId/books/:bookId', (req, res, next) => {
  console.log(req.params);
  req.params.addValue = 'newOne';
  next();
  // res.status(200).send(req.params);
}, (req, res) => {
  res.status(200).send(req.params);
});
```

  - we can modify the request/response using next
  - using next we can modify request headers also
-
```
import express from 'express';

const router = express.Router();

router.use((req, res, next) => {
  console.log('Specific middleware to birds route invoked');
  next();
});

router.get('/', (req, res) => {
  res.status(200).send('Hello Birdy');
});

router.get('/bird-about', (req, res) => {
  res.status(200).send('Penguin ');
});

export { router as BirdRouter };
```

- router.config

-


-


- Logging
  - Morgan with Winston
  - Morgan - for logging http requests, a middleware for http requests
  -


  - Winston is used for log level features like warn, error

- 
```
import { transports, createLogger } from 'w

const logLevel = process.env.LOG_LEVEL || 'debug';

const logger = createLogger({
  transports: [
    new transports.Console({
      level: logLevel,
      timestamp: function () {
        return (new Date()).toISOString();
      }
    })
  ]
});

export { logger as Logger };
```
- 
```
NodeJsTrainingConsoleApplication > app > api > JS birds.js
import express from 'express';
import { Logger } from '../../logger/logger';
const router = express.Router();

router.use((req, res, next) => {
  Logger.debug('Logging the debug level logs');
  next();
});

router.get('/', (req, res) => {
  res.status(200).send('Hello Birdy');
});

router.get('/bird-about', (req, res) => {
  res.status(200).send('Penguin ');
});

export { router as BirdRouter };
```

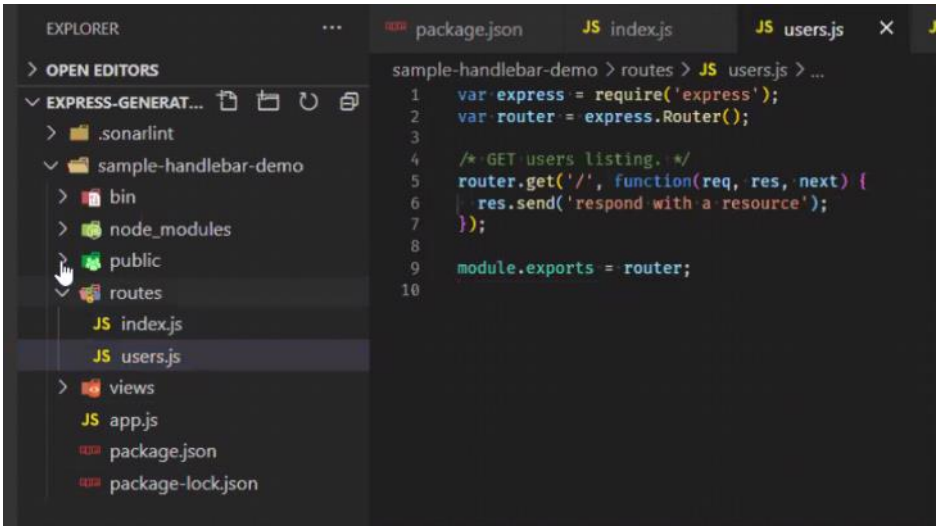
Day 6

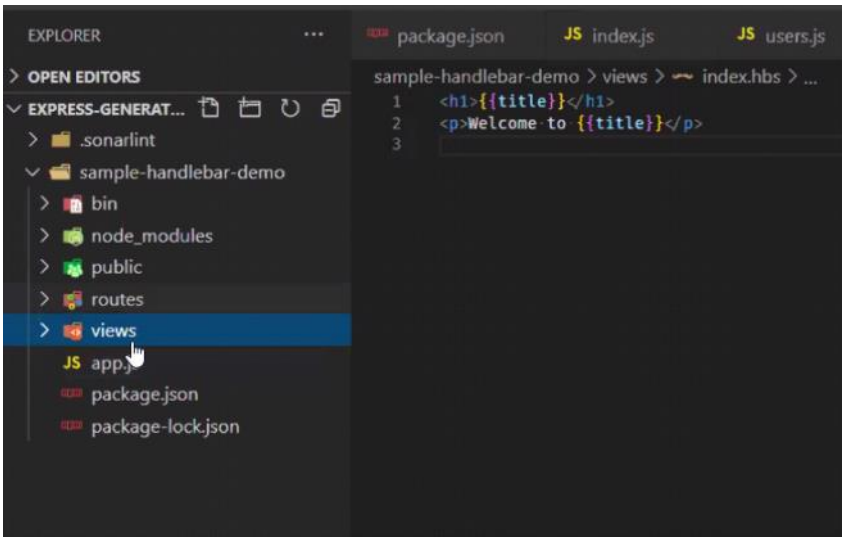
- Real time web socket
  - We use web sockets in Instant messenger / wherever we need real time data to be in sync
  - Socket.io
    - to use web sockets
    - npm i --save socket.io

- ```
  import express from 'express';
  import { applicationRoutes } from './app/router.config';
  import SocketIO from 'socket.io';
  import http from 'http';
  const app = express();
  const server = http.Server(app);
  const io = new SocketIO(server);
  // app.use('/', applicationRoutes);

  app.get('/', (req, res) => {
    res.send(`<!DOCTYPE html>
  <html>
    <head>
      <title>Hello world</title>
    </head>
    <body>Hello world</body>
  </html>`);
  });

  // setting up morgan logging middleware for HTTP Requets
  // setting up static files
  // app.use('/index', express.static(__dirname + 'public'));
  app.listen(4100, () => {
    console.log('the server is up and running');
  });

  io.on('connection', (socket) => {
    console.log('live connection is made');
    socket.on('disconnect', () => {
      console.log('user is disconencted');
    });
  });
  ```

- express generator
  - to create an app skeleton
  - npx express-generator -g sample-handlebar-demo --view=hbs
    - hbs -> handle bar -> templating engine
    - Angular internally uses handle bar
    - ```
      package.json ×  JS app.js
      sample-handlebar-demo > package.json > ...
      {
        "name": "sample-handlebar-demo",
        "version": "0.0.0",
        "private": true,
        ▷ Debug
        "scripts": {
          "start": "node ./bin/www"
        },
        "dependencies": {
          "cookie-parser": "~1.4.4",
          "debug": "~2.6.9",
          "express": "~4.16.1",
          "hbs": "~4.0.4",
          "http-errors": "~1.6.3",
          "morgan": "~1.9.1"
        }
      }
      ```

- 

- 

- 

- templating engines

Exercise

Tuesday, August 4, 2020 7:54 PM

Day 1

Creating Project

- md Bot
- cd Bot
- Bot/npm init
  - Enter all the required details about repo, author, name, description, license
- npm i --save jest // install jest in project
- npm i --save istanbul // for code coverage
- npm i -g jest // run as admin and globally install jest
- jest --init // to generate basic jest config
  - Select node
  - coverage - yes
  - instrument for coverage - v8
  - Automatically clear mock calls and instances between every test? - yes
- npm i --save eslint
- npm i -g eslint // run as admin and globally install eslint
- eslint --init
  - How would you like to use ESLint? · style
  - What type of modules does your project use? · commonjs
  - Which framework does your project use? · none
  - Does your project use TypeScript? · No / Yes
  - Where does your code run? · node
  - How would you like to define a style for your project? · guide
  - Which style guide do you want to follow? · airbnb
  - What format do you want your config file to be in? · YAML

Node app commands

Thursday, August 19, 2021 12:55 PM

- npm init --yes
- npm i express --save
- npm i mongoose --save
- npm i cors --save