# WEB COMPONENT DEV.

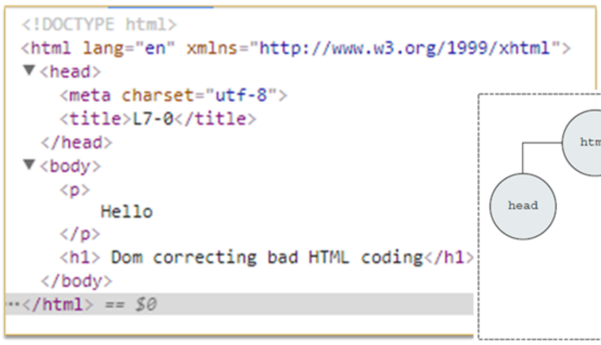
Lect2 JS6 commands recap

## Objectives

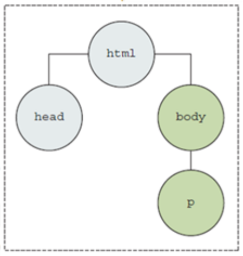
- Recap on required prerequisites.
- Look at DOM Vs Virtual DOM.
- Review High Level functions.
- Reinforce our understanding of async programming.
- Loo at the Node Package Manager

## Parsing HTML and building the DOM

- The Browser parses the HTML code, one HTML element at a time, and builds the Document Object Model (**DOM**).
- The DOM is a structured representation of the HTML page in which every HTML element is represented as a node.



```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
▼<head>
    <meta charset="utf-8">
    <title>L7-0</title>
  </head>
▼<body>
    <p>
        Hello
    </p>
    <h1> Dom correcting bad HTML coding</h1>
  </body>
··</html> == $0
```

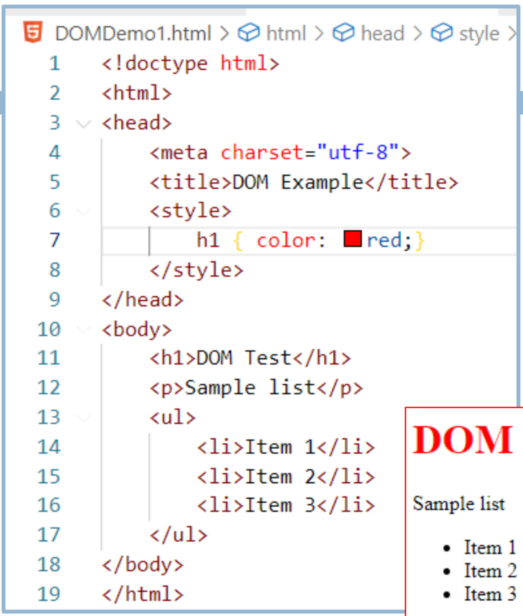


## DOM

- The Document Object Model (**DOM**) is an object-oriented representation of an HTML or XML document.
- The structure of an HTML / XML document is hierarchical, so the DOM structure resembles that of a tree.
- DOM provides an API to access and modify this tree of objects.
- The DOM API is specified in a language-independent manner by W3C

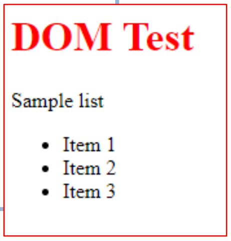
## Manipulating DOM

- One of the primary means for achieving highly dynamic web applications that respond to user actions is by modifying the DOM.
- It is important to have a good understanding of HTML and CSS before starting DOM manipulation.



DOMDemo1.html > html > head > style >

```html
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>DOM Example</title>
    <style>
        h1 { color: red;}
    </style>
</head>
<body>
    <h1>DOM Test</h1>
    <p>Sample list</p>
    <ul>
        <li>Item 1</li>
        <li>Item 2</li>
        <li>Item 3</li>
    </ul>
</body>
</html>
```

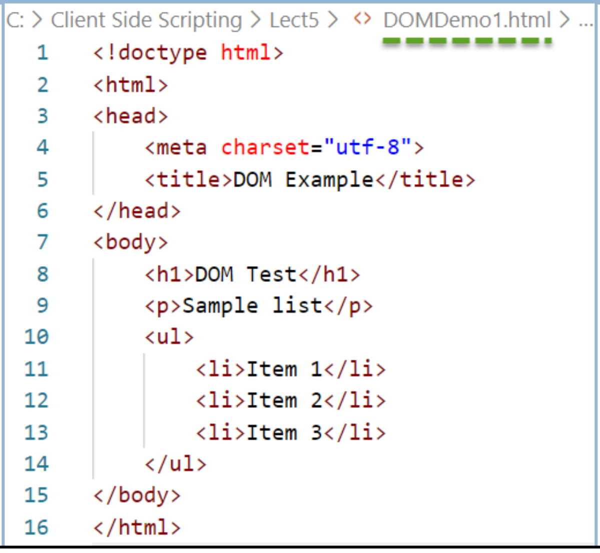


# DOM Test

Sample list

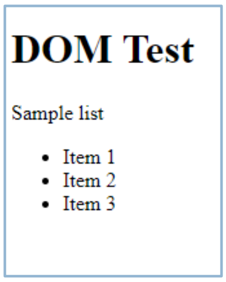
- Item 1
- Item 2
- Item 3

## HTML example



C: > Client Side Scripting > Lect5 > DOMDemo1.html > ...

```html
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>DOM Example</title>
</head>
<body>
    <h1>DOM Test</h1>
    <p>Sample list</p>
    <ul>
        <li>Item 1</li>
        <li>Item 2</li>
        <li>Item 3</li>
    </ul>
</body>
</html>
```



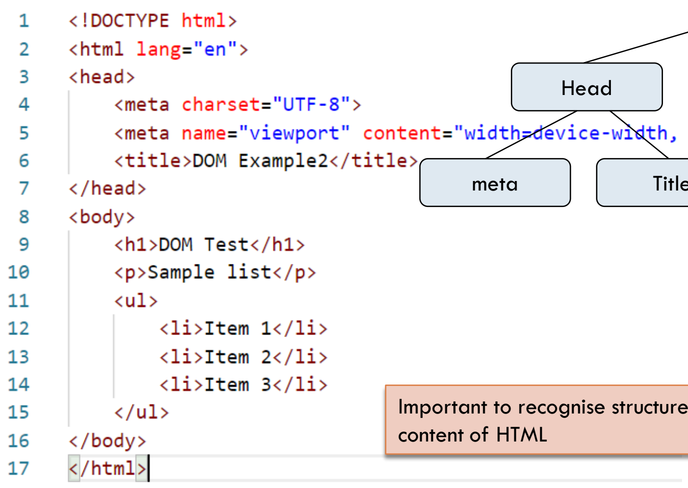
# DOM Test

Sample list

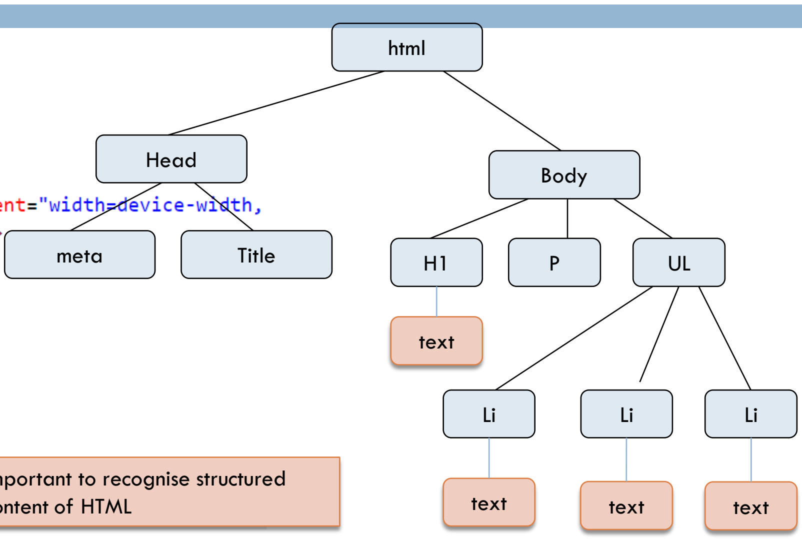
- Item 1
- Item 2
- Item 3

static web page
We want to start interacting with the page

## *HTML document and the DOM*



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
    <title>DOM Example2</title>
</head>
<body>
    <h1>DOM Test</h1>
    <p>Sample list</p>
    <ul>
        <li>Item 1</li>
        <li>Item 2</li>
        <li>Item 3</li>
    </ul>
</body>
</html>
```

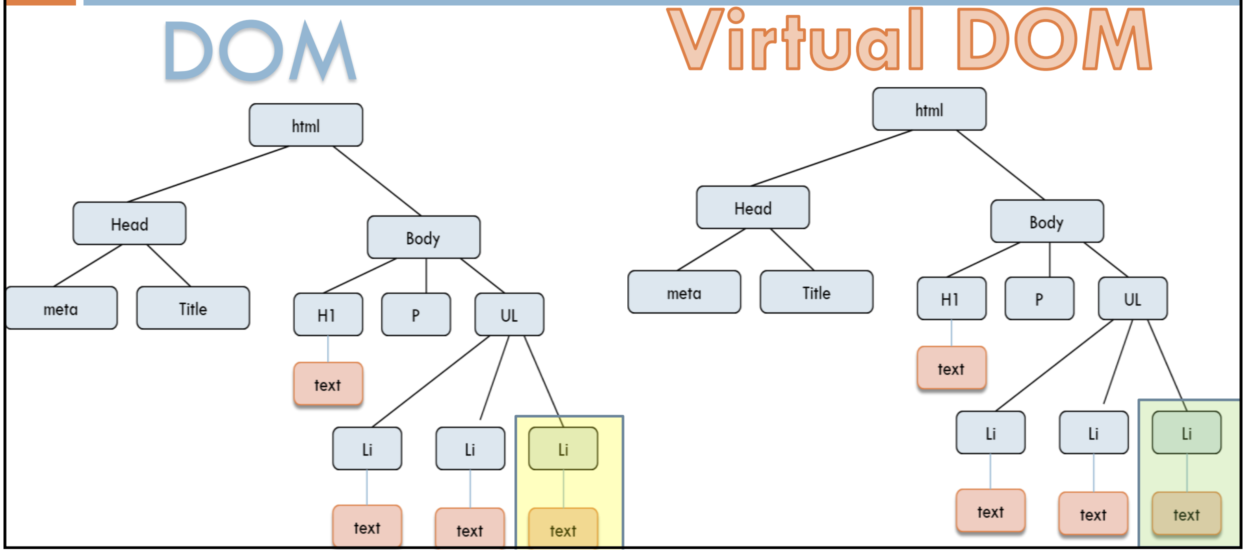


Important to recognise structured content of HTML

## Virtual DOM

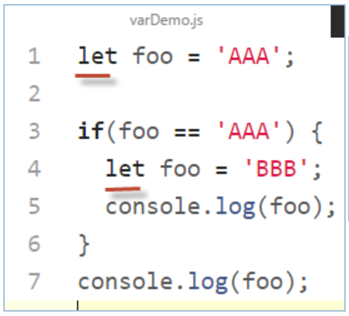
- Lightweight, in-memory representation of the real DOM. React uses it to decide how to update the UI efficiently.
- To manipulate the actual DOM directly is slow. The Virtual DOM optimises and batches modifications to reduce these updates.
  - When state or property change, React creates a new Virtual DOM tree.
  - The new tree is compared with the previous tree(diffing).
  - Calculates the minimal changes needed.
  - Only those changes are applied to the real DOM.

Reacts Virtual DOM only changes updated node from the DOM. This greatly enhances processing speed of React

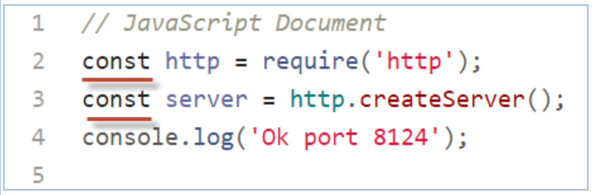


## JavaScript Extensions

- **Variables**
  - **`let and const`**
  - let is *block scoped*
  - const for variables that never change

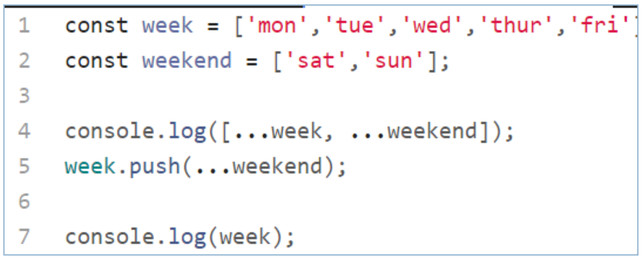

varDemo.js

```
let foo = 'AAA';

if(foo == 'AAA') {
  let foo = 'BBB';
  console.log(foo);
}
console.log(foo);
```

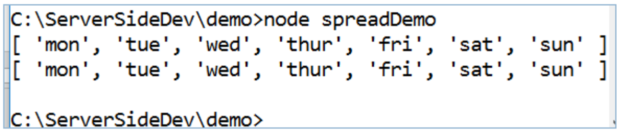


```
// JavaScript Document
const http = require('http');
const server = http.createServer();
console.log('Ok port 8124');

```

## spread pattern



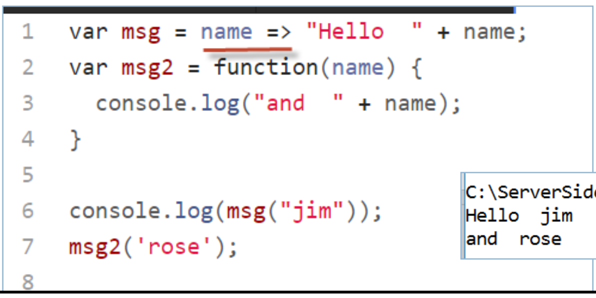
```javascript
const week = ['mon','tue','wed','thur','fri'
const weekend = ['sat','sun'];

console.log([...week, ...weekend]);
week.push(...weekend);

console.log(week);
```



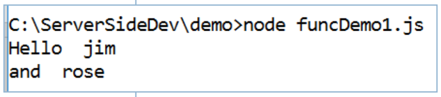
```
C:\ServerSideDev\demo>node spreadDemo
[ 'mon', 'tue', 'wed', 'thur', 'fri', 'sat', 'sun' ]
[ 'mon', 'tue', 'wed', 'thur', 'fri', 'sat', 'sun' ]

C:\ServerSideDev\demo>
```

## Arrow functions

- shorten function declarations
- function() {}
- () => {}
- param => expression

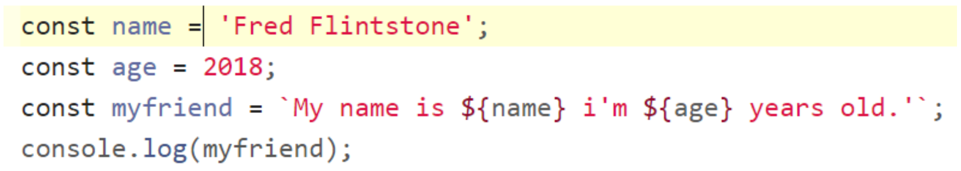


```javascript
var msg = name => "Hello  " + name;
var msg2 = function(name) {
  console.log("and  " + name);
}

console.log(msg("jim"));
msg2('rose');

```

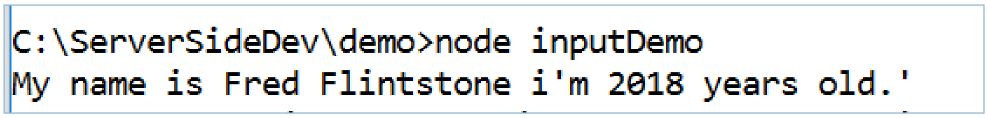


```
C:\ServerSideDev\demo>node funcDemo1.js
Hello  jim
and  rose
```

## String interpolation



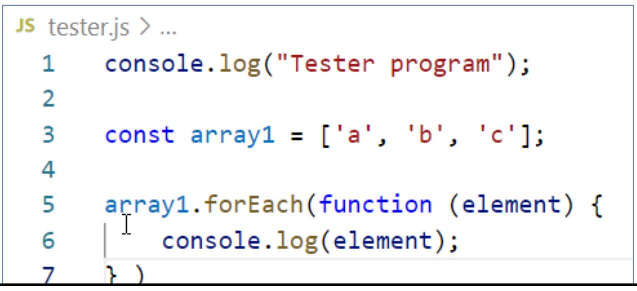
```
const name = 'Fred Flintstone';
const age = 2018;
const myfriend = `My name is ${name} i'm ${age} years old.'`;
console.log(myfriend);
```



```
C:\ServerSideDev\demo>node inputDemo
My name is Fred Flintstone i'm 2018 years old.'
```

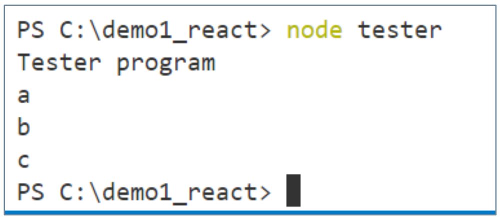
## forEach method

- The `forEach()` method executes a provided function once for each array element.
- The `foreach` takes whatever function you give it and it calls it on each element.



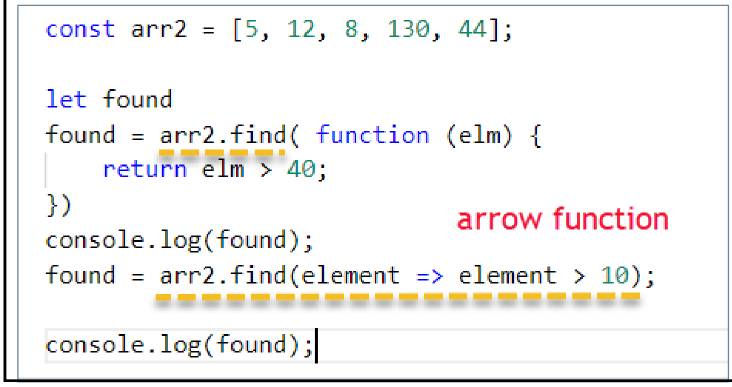
tester.js > ...

```
console.log("Tester program");

const array1 = ['a', 'b', 'c'];

array1.forEach(function (element) {
    console.log(element);
})
```

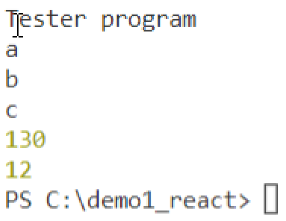


```
PS C:\demo1_react> node tester
Tester program
a
b
c
PS C:\demo1_react>
```

- The `find()` method returns the first element in the provided array that satisfies the provided testing function. If no values satisfy the testing function, undefined is returned.

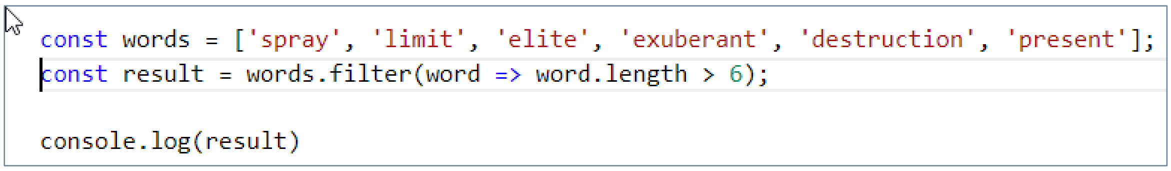


```
const arr2 = [5, 12, 8, 130, 44];

let found
found = arr2.find( function (elm) {
    return elm > 40;
})
console.log(found);
found = arr2.find(element => element > 10);

console.log(found);
```

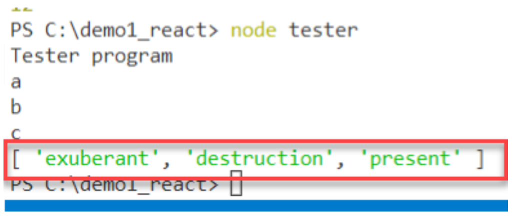


```
Tester program
a
b
c
130
12
PS C:\demo1_react> 
```

- Filters out subsets of an array which returns true or false’
- Elements that pass are added into the returned array.



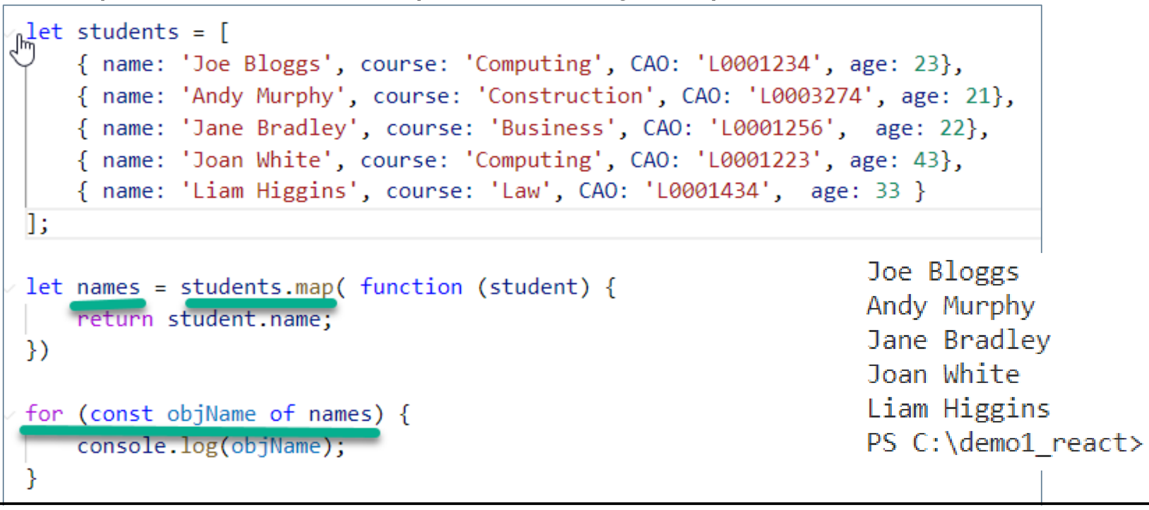
```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];
const result = words.filter(word => word.length > 6);

console.log(result)
```



```
PS C:\demo1_react> node tester
Tester program
a
b
c
[ 'exuberant', 'destruction', 'present' ]
PS C:\demo1_react> 
```

## Map method

- Map creates a new array from existing array.



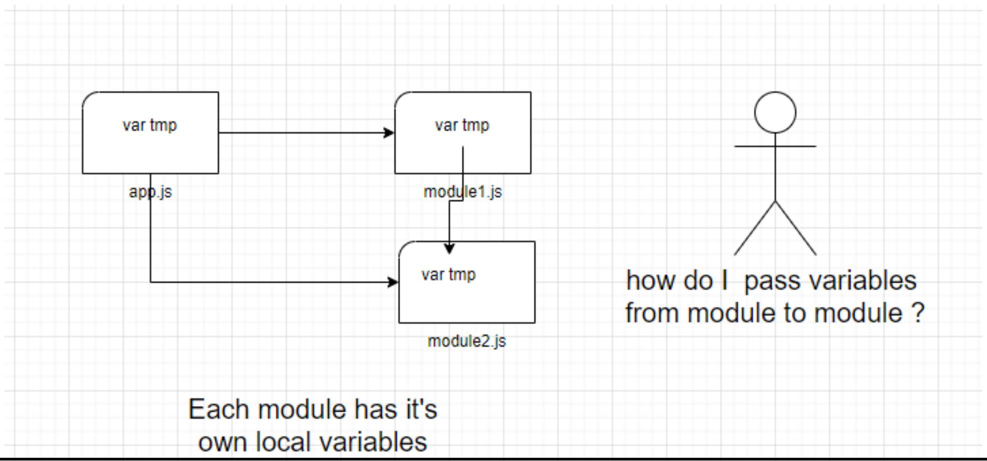
```
let students = [
    { name: 'Joe Bloggs', course: 'Computing', CAO: 'L0001234', age: 23},
    { name: 'Andy Murphy', course: 'Construction', CAO: 'L0003274', age: 21},
    { name: 'Jane Bradley', course: 'Business', CAO: 'L0001256',  age: 22},
    { name: 'Joan White', course: 'Computing', CAO: 'L0001223', age: 43},
    { name: 'Liam Higgins', course: 'Law', CAO: 'L0001434',  age: 33 }
];

let names = students.map( function (student) {
    return student.name;
})

for (const objName of names) {
    console.log(objName);
}
```

```
Joe Bloggs
Andy Murphy
Jane Bradley
Joan White
Liam Higgins
PS C:\demo1_react>
```

## Modules

- In **Node.js** a module can be a single or multiple JavaScript files which can be reused throughout the application.
- Each module in **Node.js** contains variables within the scope of the module, therefore it cannot interfere with other modules or pollute global scope.
- Also, each module can be placed in a separate **.js** file under a separate folder.

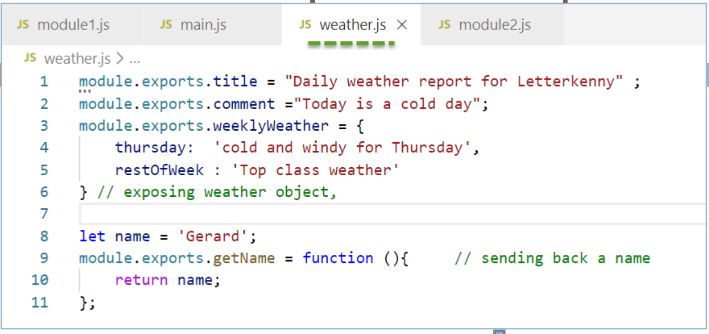
- Modules are self-contained units of functionality


var tmp
app.js
var tmp
module1.js
var tmp
module2.js
how do I pass variables from module to module ?
Each module has it's own local variables

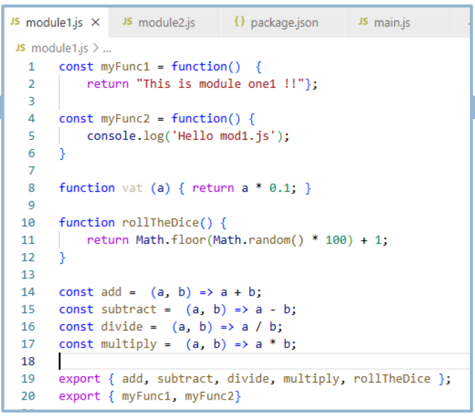
## Node Module Types

- Core Modules
  - Build-in to node.js
    - Used **http** module to build our server
- Local Modules
- Third Party Modules

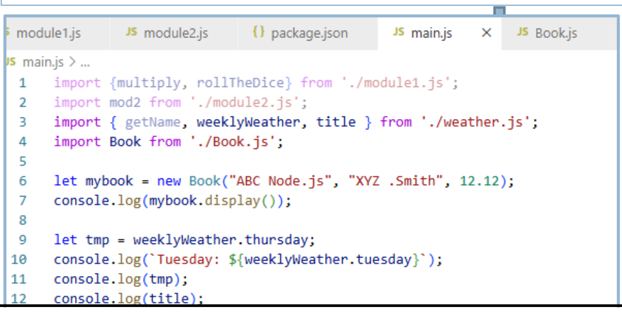
## Module Exports Example



```
module.exports.title = "Daily weather report for Letterkenny" ;
module.exports.comment ="Today is a cold day";
module.exports.weeklyWeather = {
    thursday:  'cold and windy for Thursday',
    restOfWeek : 'Top class weather'
} // exposing weather object,

let name = 'Gerard';
module.exports.getName = function (){      // sending back a name
    return name;
};
```

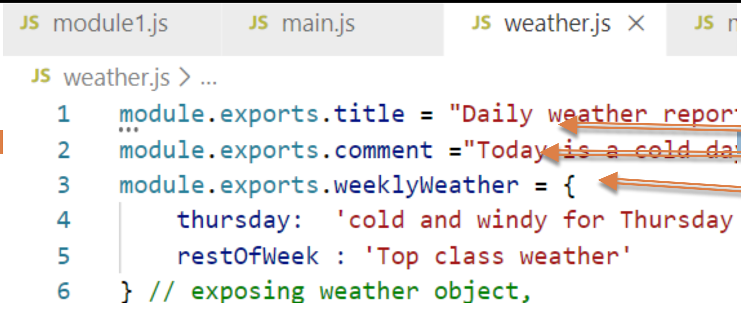


```
const myFunc1 = function()  {
    return "This is module one1 !!"};

const myFunc2 = function() {
    console.log('Hello mod1.js');
}

function vat (a) { return a * 0.1; }

function rollTheDice() {
    return Math.floor(Math.random() * 100) + 1;
}

const add =  (a, b) => a + b;
const subtract =  (a, b) => a - b;
const divide =  (a, b) => a / b;
const multiply =  (a, b) => a * b;

export { add, subtract, divide, multiply, rollTheDice };
export { myFunc1, myFunc2}
```



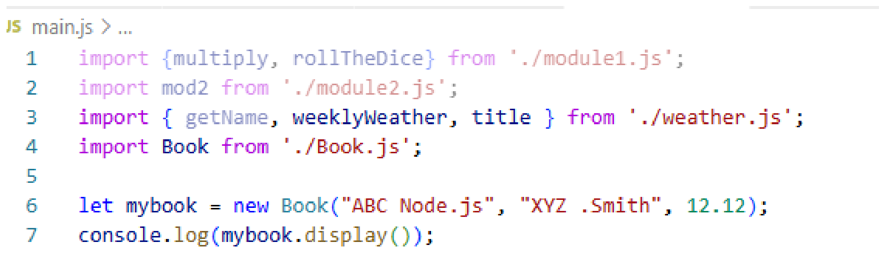
```
import {multiply, rollTheDice} from './module1.js';
import mod2 from './module2.js';
import { getName, weeklyWeather, title } from './weather.js';
import Book from './Book.js';

let mybook = new Book("ABC Node.js", "XYZ .Smith", 12.12);
console.log(mybook.display());

let tmp = weeklyWeather.thursday;
console.log(`Tuesday: ${weeklyWeather.tuesday}`);
console.log(tmp);
console.log(title);
```

```
PS C:\ServerSideDev\Lect3> node main
And this is module two
Gerard
cold and windy for Thursday
PS C:\ServerSideDev\Lect3>
```

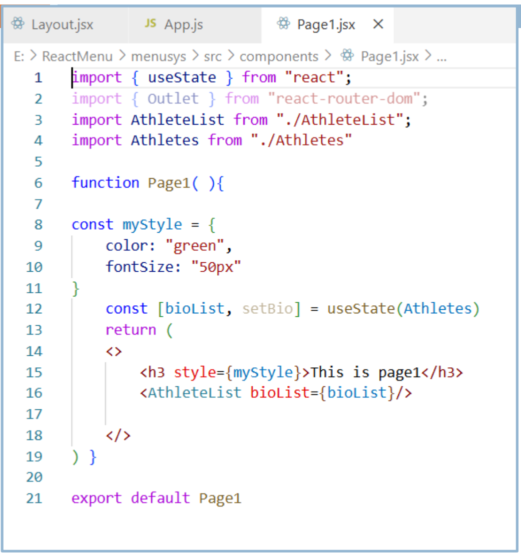


```
module.exports.title = "Daily weather repor
module.exports.comment ="Today is a cold da
module.exports.weeklyWeather = {
    thursday:  'cold and windy for Thursday
    restOfWeek : 'Top class weather'
} // exposing weather object,
```

Notice the .title, .comment and the .object



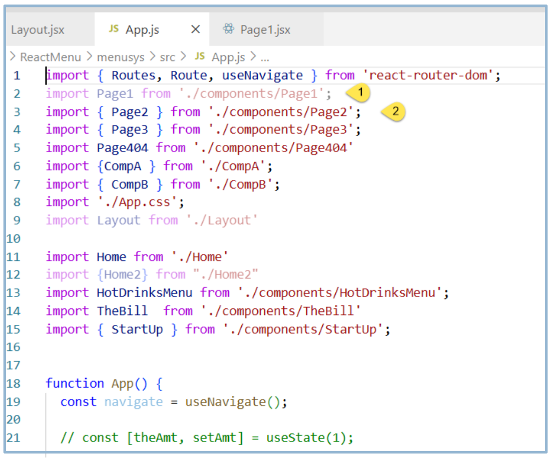
```
import {multiply, rollTheDice} from './module1.js';
import mod2 from './module2.js';
import { getName, weeklyWeather, title } from './weather.js';
import Book from './Book.js';

let mybook = new Book("ABC Node.js", "XYZ .Smith", 12.12);
console.log(mybook.display());
```



Layout.jsx | App.js | Page1.jsx

E: > ReactMenu > menusys > src > components > Page1.jsx > ...

```jsx
import { useState } from "react";
import { Outlet } from "react-router-dom";
import AthleteList from "./AthleteList";
import Athletes from "./Athletes"

function Page1( ){

const myStyle = {
    color: "green",
    fontSize: "50px"
}
    const [bioList, setBio] = useState(Athletes)
    return (
    <>
        <h3 style={myStyle}>This is page1</h3>
        <AthleteList bioList={bioList}/>

    </>
) }

export default Page1
```



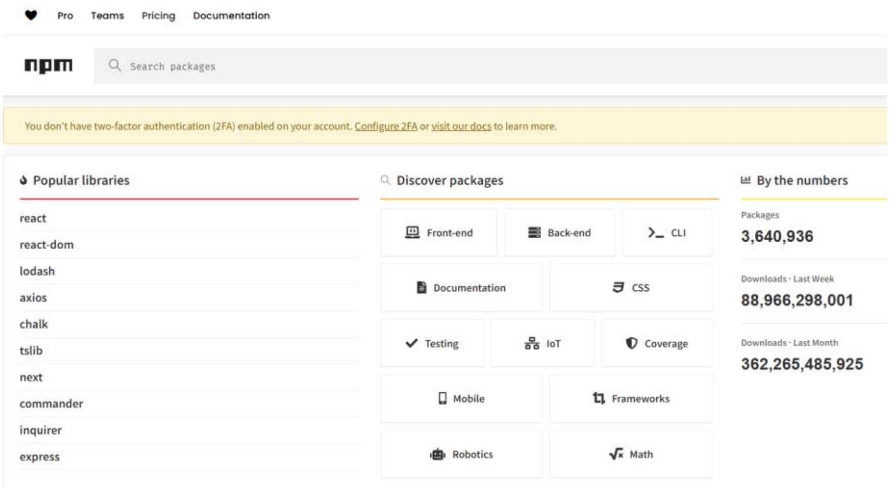
Layout.jsx | App.js | Page1.jsx

> ReactMenu > menusys > src > App.js > ...

```jsx
import { Routes, Route, useNavigate } from 'react-router-dom';
import Page1 from './components/Page1';   1
import { Page2 } from './components/Page2';   2
import { Page3 } from './components/Page3';
import Page404 from './components/Page404'
import {CompA } from './CompA';
import { CompB } from './CompB';
import './App.css';
import Layout from './Layout'

import Home from './Home'
import {Home2} from "./Home2"
import HotDrinksMenu from './components/HotDrinksMenu';
import TheBill  from './components/TheBill'
import { StartUp } from './components/StartUp';


function App() {
  const navigate = useNavigate();

  // const [theAmt, setAmt] = useState(1);
```

## NPM

- npm website https://www.npmjs.com/



You don't have two-factor authentication (2FA) enabled on your account. Configure 2FA or visit our docs to learn more.

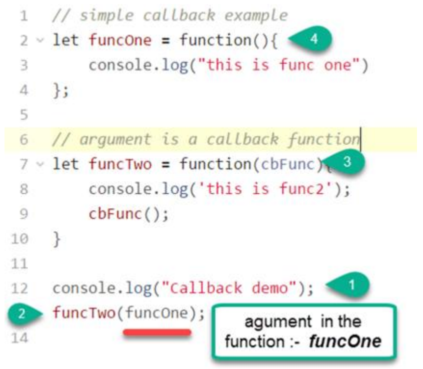
Packages
3,640,936

Downloads · Last Week
88,966,298,001

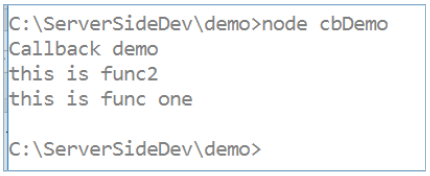
Downloads · Last Month
362,265,485,925

## High Order functions

- Do we remember what a callback function is?
- callback function that is a argument that is invoked when the higher level function is called.

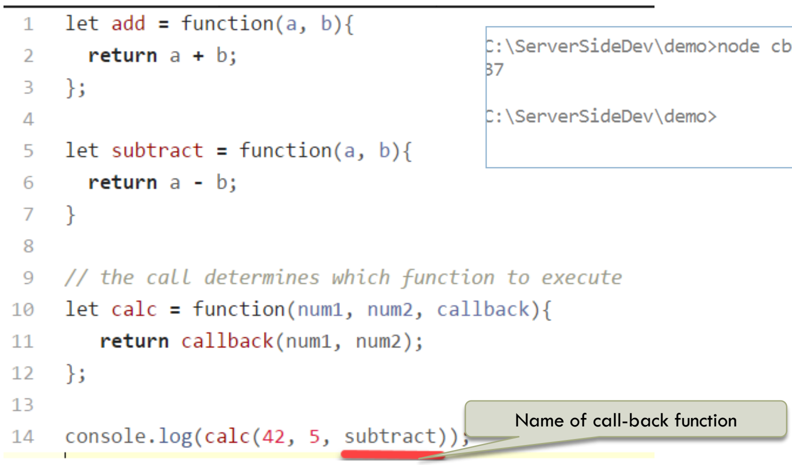


```
// simple callback example
let funcOne = function(){
    console.log("this is func one")
};

// argument is a callback function
let funcTwo = function(cbFunc){
    console.log('this is func2');
    cbFunc();
}

console.log("Callback demo");
funcTwo(funcOne);
```

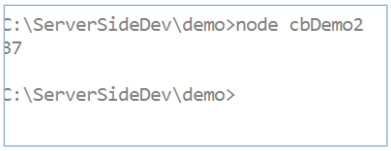


```
C:\ServerSideDev\demo>node cbDemo
Callback demo
this is func2
this is func one

C:\ServerSideDev\demo>
```

## Call back example 2



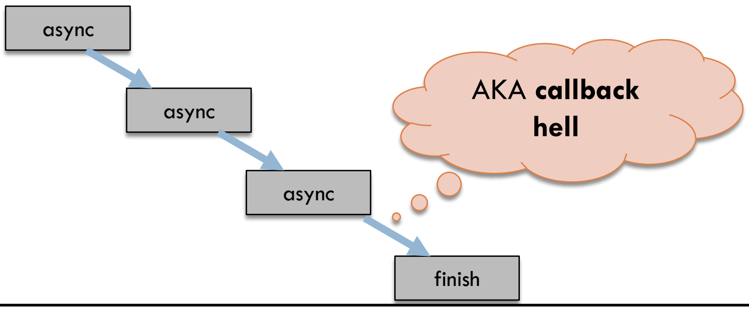
```
let add = function(a, b){
  return a + b;
};

let subtract = function(a, b){
  return a - b;
}

// the call determines which function to execute
let calc = function(num1, num2, callback){
   return callback(num1, num2);
};

console.log(calc(42, 5, subtract));
```



```
C:\ServerSideDev\demo>node cbDemo2
37

C:\ServerSideDev\demo>
```

## promises

- The previous slide shows one of the drawbacks working with files and web servers in node.
- Because of asynchronous non-blocking approach our code often contains numerous nested **callback** functions.
- This makes the code a headache to write and maintain.



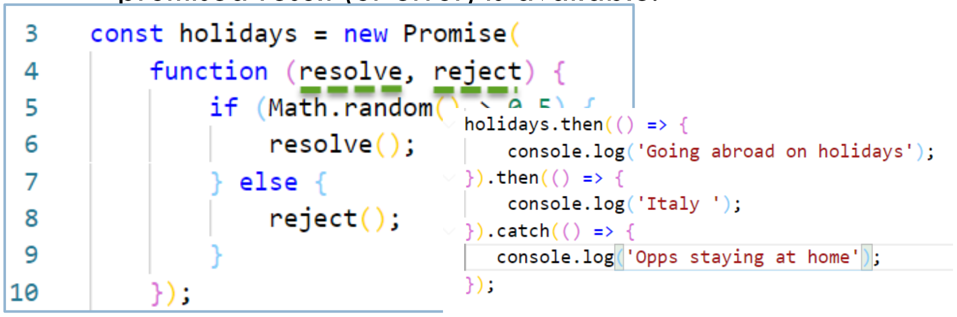
## Promises

- Code from the previous slide contains a lot of repetition.
- Ugly and difficult to understand/maintain.

- We need a better approach
  - Use JavaScript **Promise** object
  - This Object return the data or an error
  - Works well with the **async** code because it make it look synchronous.
  - The Promise object has a very important **`then()`** method which will be very beneficial when working with browsers, database, files…

## Promise example

- A Promise represents an operation that hasn't completed yet but is expected to be completed in the future. The promise has three states **pending, resolved** or **rejected.**
- The **.then** or **.catch** functions are invoked when the promised result (or error) is available.



```
const holidays = new Promise(
    function (resolve, reject) {
        if (Math.random() > 0.5) {
            resolve();
        } else {
            reject();
        }
    });
```

```
holidays.then(() => {
    console.log('Going abroad on holidays');
}).then(() => {
    console.log('Italy ');
}).catch(() => {
   console.log('Opps staying at home');
});
```

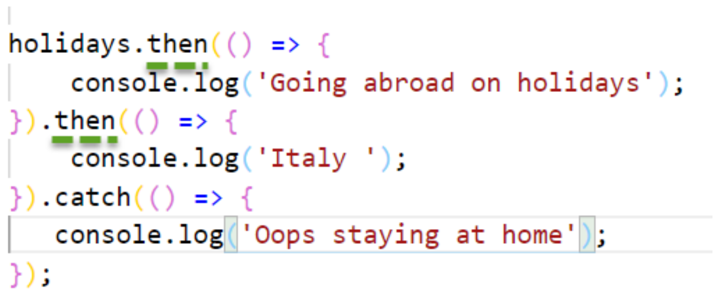
- // promise syntax
- new Promise(function (resolve, reject) {
  // code goes here } );

```
const holidays = new Promise(
    function (resolve, reject) {
        if (Math.random() > 0.5) {
            resolve();
        } else {
            reject();
        }
    });
```

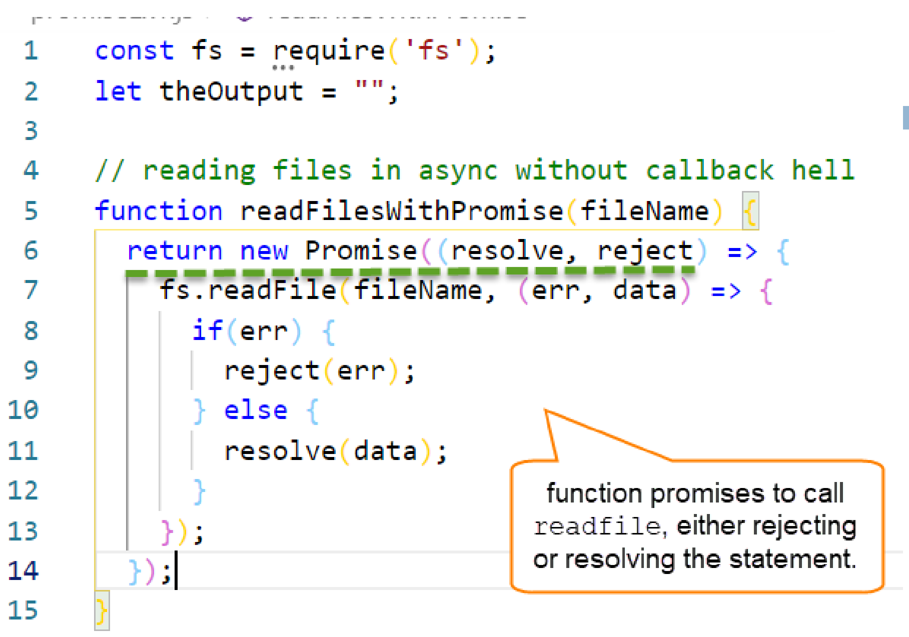
- A `promise` select one of two params: **resolve(success)** or **reject(error)**. If the operation is successful, you can pass data to the code block that uses that promise.
- Rejected promises (error state) can be handled in a catch.

## **then** methods

- **then** methods can be chained on promises, for example we can say once you have complete one task than go on to the next and the next …
- **catch** method is invoked if an error occurs anywhere in the chain. Note only one catch method is required.



```
holidays.then(() => {
    console.log('Going abroad on holidays');
}).then(() => {
    console.log('Italy ');
}).catch(() => {
   console.log('Oops staying at home');
});
```



```
const fs = require('fs');
let theOutput = "";

// reading files in async without callback hell
function readFilesWithPromise(fileName) {
  return new Promise((resolve, reject) => {
    fs.readFile(fileName, (err, data) => {
      if(err) {
        reject(err);
      } else {
        resolve(data);
      }
    });
  });
}
```

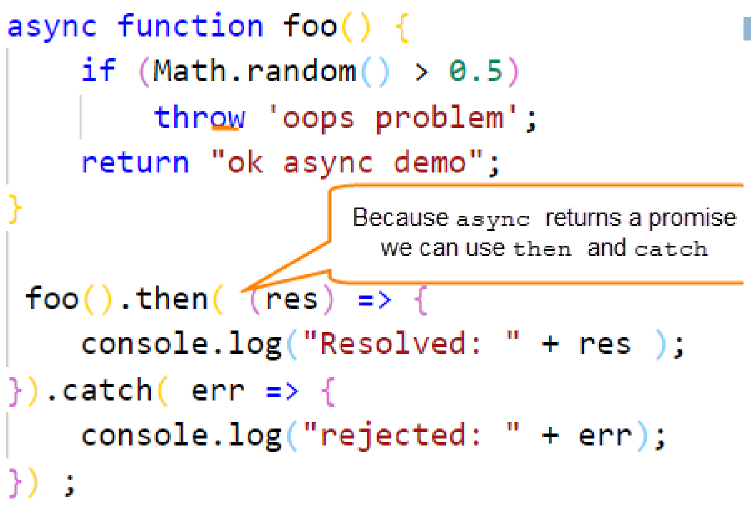
function promises to call `readfile`, either rejecting or resolving the statement.

## aSync calls

- **callbacks** are needed, not just for performance. But also, stop blockage or any holdups form other programs.
- Often our apps begin by loading files at the initialising stage.
- These file and may only take fractions of seconds.
- The use callbacks with one or more files can make our programs difficult to read and maintain.
- The developers of Node have given us various **'sync'** methods to help simplify these tasks.

## async / await

- **async / await**
  - **async** and **await** help to make the code syntax look prettier.
  - The beauty of **async** functions is that you can write asynchronous code as if it's synchronous code.
  - Putting the keyword **async** in front of a function, ensures that the function will return a **Promise**. If an exception occurs the promise get rejected.
  - Available since Node Version 8+

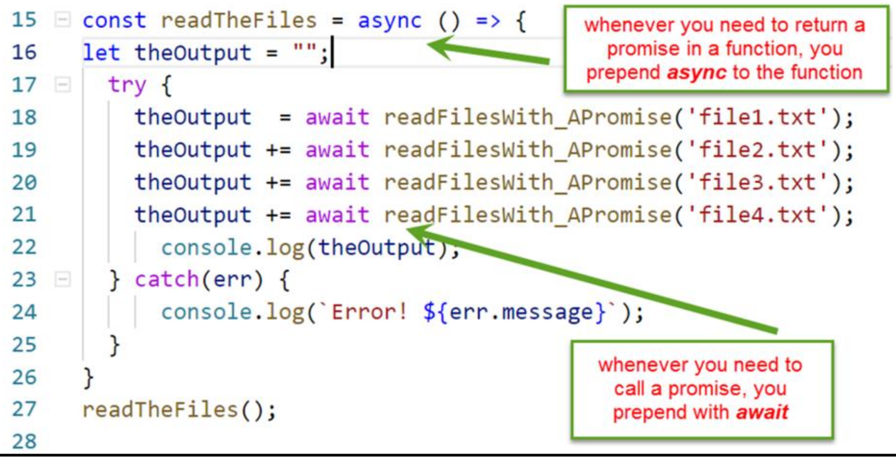


```
async function foo() {
    if (Math.random() > 0.5)
        throw 'oops problem';
    return "ok async demo";
}

 foo().then( (res) => {
    console.log("Resolved: " + res );
}).catch( err => {
    console.log("rejected: " + err);
}) ;
```

Because `async` returns a promise we can use `then` and `catch`

## await

- **await** only used inside **async** functions.
- Pauses the execution until the promise is resolved.



```
const readTheFiles = async () => {
let theOutput = "";
  try {
    theOutput  = await readFilesWith_APromise('file1.txt');
    theOutput += await readFilesWith_APromise('file2.txt');
    theOutput += await readFilesWith_APromise('file3.txt');
    theOutput += await readFilesWith_APromise('file4.txt');
      console.log(theOutput);
  } catch(err) {
      console.log(`Error! ${err.message}`);
  }
}
readTheFiles();
```

whenever you need to return a promise in a function, you prepend ***async*** to the function

whenever you need to call a promise, you prepend with ***await***

## Summary

- Investigated how the different elements on a HTML page are constructed to build the DOM
- Looked at JavaScript techniques for navigating  DOM
- Demonstrated newer  JavaScript functions.
- Develop understand of how sync and async applications differ and the use of Promise objects.

## Webography

- *http://www.w3.org/DOM/*
- http://www.w3.org/TR/#tr_DOM
- http://www.w3schools.com/jsref/dom_obj_document.asp
- http://www.htmlgoodies.com/primers/jsp/article.php/3594621/Javascript-Basics-Part-6.htm