NODE.JS

Server Side Javascript

Node.js – an intro

- In 2009 Ryan Dahl created Node.js or Node, a framework primarily used to create highly scalable servers for web applications.
  - Node.js is an open source, cross-platform runtime environment for server-side JavaScript.
  - Node.js is required to run JavaScript without a browser support. It uses Google V8 JavaScript engine to execute code.
  - It is written in C++ and JavaScript.
  - Node.js is a development framework that is based on Google's V8 JavaScript engine that powers Google's Chrome web browser.
  - You write Node.js code in JavaScript, and then V8 compiles it into machine code to be executed.

  - It's a highly scalable system that uses asynchronous, non-blocking I/O model (input/output), rather than threads or separate processes
  - It is not a framework like jQuery nor a programming language like C# or JAVA; Its primarily a Javascript engine

Node.js is really two things: a runtime environment and a library

Traditional Programming vs Event-driven programming

- In traditional programming I/O is performed in the same way as it does local function calls. i.e. Processing cannot continue until the operation is completed.
  - When the operation like executing a query against database is being executed, the whole process/thread idles, waiting for the response. This is termed as "Blocking"
  - Due to this blocking behavior we cannot perform another I/O operation, the call stack becomes frozen waiting for the response.

- Event-driven programming or Asynchronous programming is a programming style where the flow of execution is determined by events.
- Events are handled by event handlers or event callbacks
  - An event callback is a function that is invoked when something significant happens like when the user clicks on a button or when the result of a database query is available.

- This style of programming — whereby instead of using a return value you define functions that are called by the system when interesting events occur — is called event-driven or asynchronous programming.

Typical blocking I/O programming

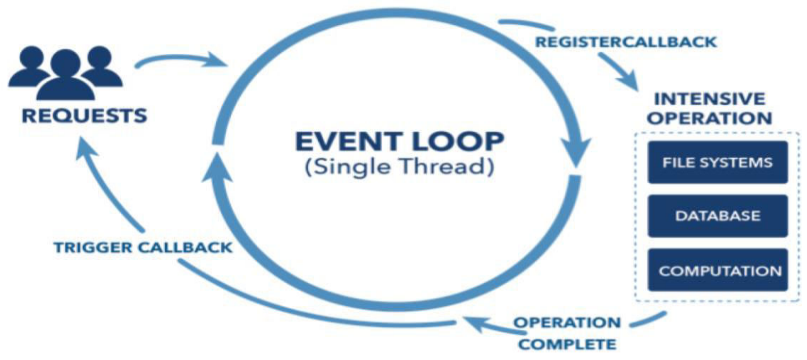
```
result = query('SELECT * FROM posts WHERE id = 1');
do_something_with(result);
```

```
query_finished = function(result) {
     do_something_with(result);
}
query('SELECT * FROM posts WHERE id = 1', query_finished);
```

Event loop

- An event loop is a construct that mainly performs two functions in a continuous loop — **event detection and event handler triggering**.
  - In any run of the loop, it has to detect which events just happened.
  - Then, when an event happens, the event loop must determine the event callback and invoke it.
- This event loop is just one thread running inside one process, which means that, when an event happens, the event handler can run without interruption. This means the following:
  - There is at most one event handler running at any given time.
  - Any event handler will run to completion without being interrupted.

  - Node.js uses the "Single Threaded Event Loop" architecture to handle multiple concurrent clients.



Asynchronous and Event Driven

- All APIs of Node.js library are asynchronous that is, non-blocking.
  - It essentially means a Node.js based server never waits for an API to return data.
  - The server moves to the next API after calling it and a notification mechanism of Node.js helps the server to get a response from the previous API call.
  - It is non-blocking, so it doesn't make the program wait, but instead it registers a callback and lets the program continue.

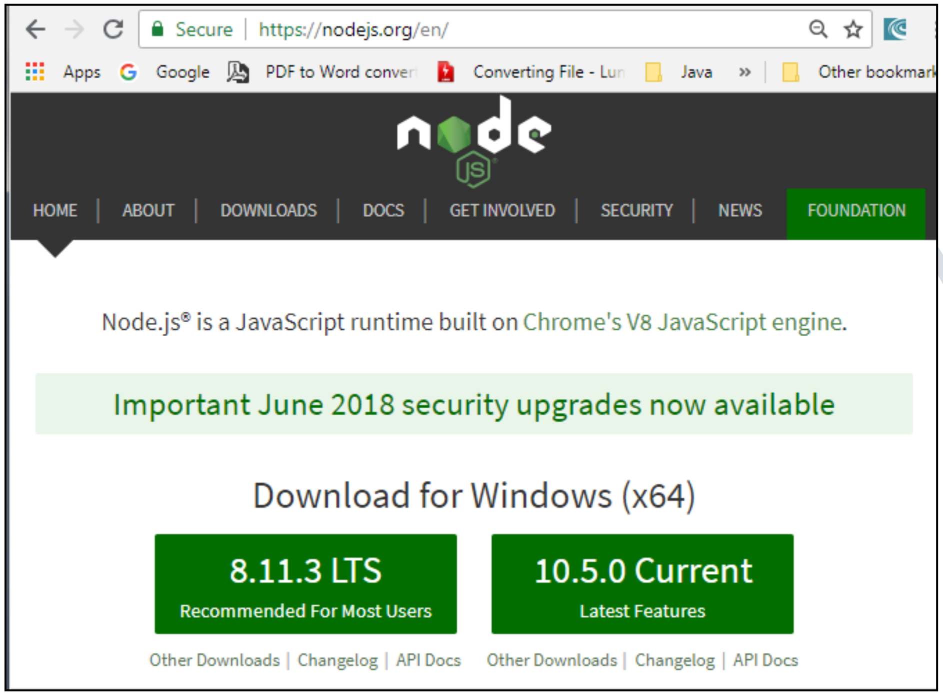
- Node.js is not fit for an application which performs CPU-intensive operations like image processing or other heavy computation work because it takes time to process a request and thereby blocks the single thread.
- Node.js is great for data-intensive applications.
  - Using a single thread means that Node.js has an extremely low-memory footprint when used as a web server and can potentially serve a lot more requests.
  - Eg, a data intensive application that serves a dataset from a database to clients via HTTP

- **What Node is NOT!**

Node is **not** a webserver. By itself it doesn't do anything. It doesn't work like Apache. There is no config file where you point it to you HTML files. If you want it to be a HTTP server, you have to write an HTTP server (with the help of its built-in libraries). Node.js is just another way to execute code on your computer.
It is simply a JavaScript runtime.

Setting up Node

- To install and setup an environment for Node.js :
  - Download the latest version of Node.js installable archive file from https://nodejs.org/en/
  - Double click to run the msi file
  - Verify if the installation was successful : node –v in command window.



Secure | https://nodejs.org/en/

Apps Google PDF to Word conver Converting File - Lur Java » Other bookmark

HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | NEWS | FOUNDATION

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

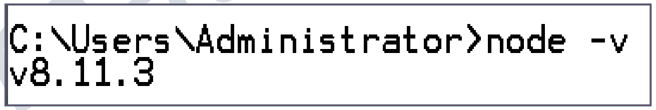
Important June 2018 security upgrades now available

Download for Windows (x64)

8.11.3 LTS
Recommended For Most Users

10.5.0 Current
Latest Features

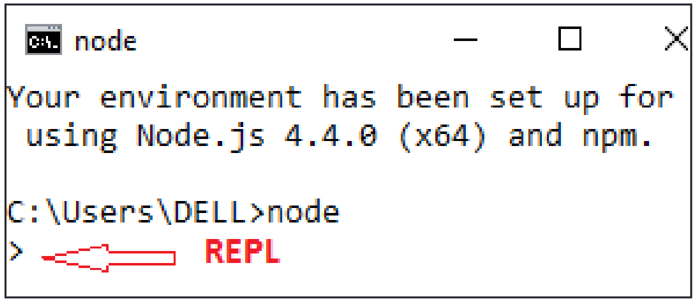
Other Downloads | Changelog | API Docs Other Downloads | Changelog | API Docs



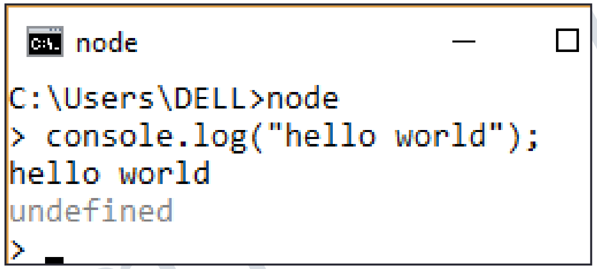
```
C:\Users\Administrator>node -v
v8.11.3
```

Using the Node CLI : REPL (Read-Eval-Print-Loop)

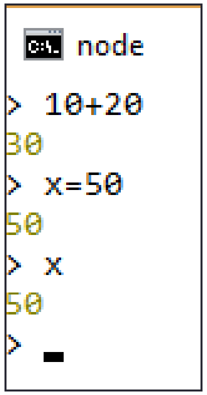
- There are two primary ways to use Node.js on your machines: by using the Node Shell or by saving JavaScript to files and running those.
  - Node shell is also called the Node REPL; a great way to quickly test things in Node.
  - When you run "node" without any command line arguments, it puts you in REPL



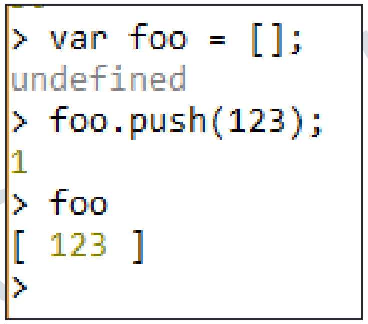
```
Your environment has been set up for
 using Node.js 4.4.0 (x64) and npm.

C:\Users\DELL>node
>
```



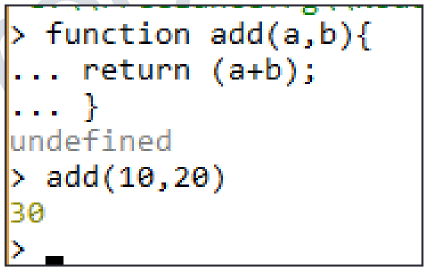
```
C:\Users\DELL>node
> console.log("hello world");
hello world
undefined
>
```



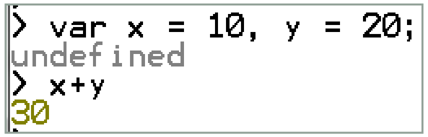
```
> 10+20
30
> x=50
50
> x
50
>
```



```
> var foo = [];
undefined
> foo.push(123);
1
> foo
[ 123 ]
>
```

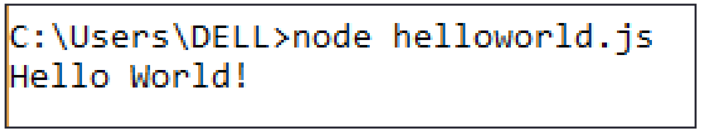


```
> function add(a,b){
... return (a+b);
... }
undefined
> add(10,20)
30
>
```



```
> var x = 10, y = 20;
undefined
> x+y
30
```

- You can also create a js file and type in some javascript.

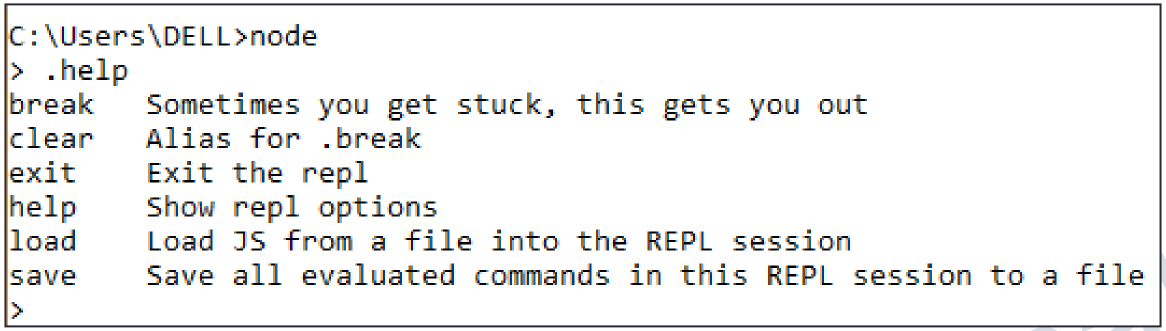


```
C:\Users\DELL>node helloworld.js
Hello World!
```

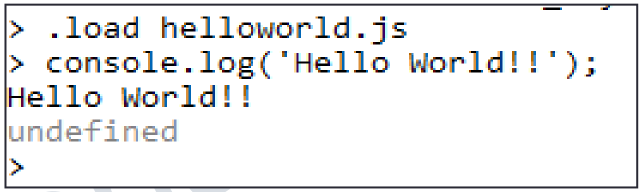
```
//helloworld.js
console.log("Hello World!");
```

Using the REPL

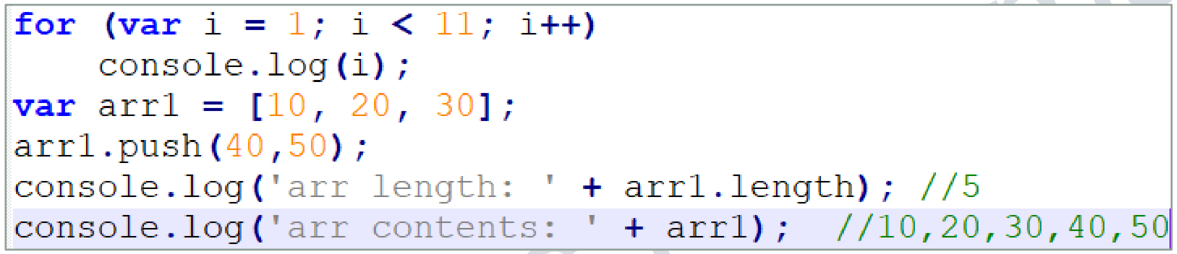
- To view the options available to you in REPL type .help and press Enter.



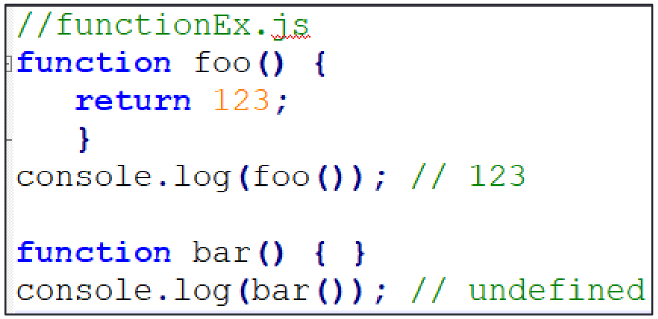
```
C:\Users\DELL>node
> .help
break   Sometimes you get stuck, this gets you out
clear   Alias for .break
exit    Exit the repl
help    Show repl options
load    Load JS from a file into the REPL session
save    Save all evaluated commands in this REPL session to a file
>
```



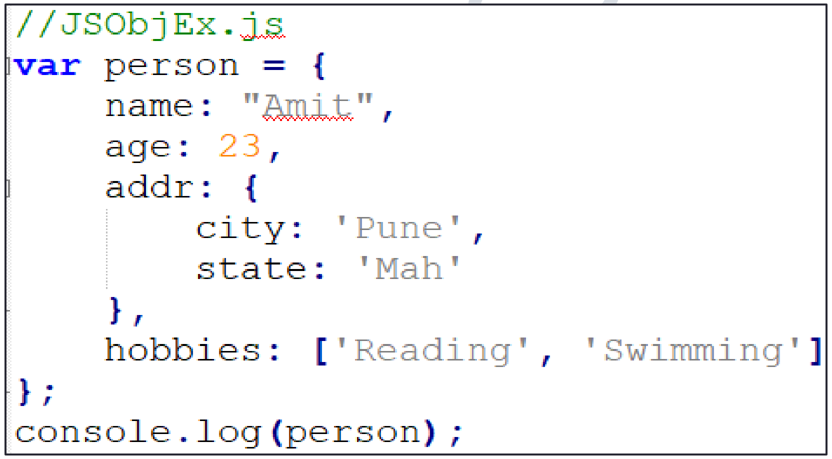
```
> .load helloworld.js
> console.log('Hello World!!');
Hello World!!
undefined
>
```



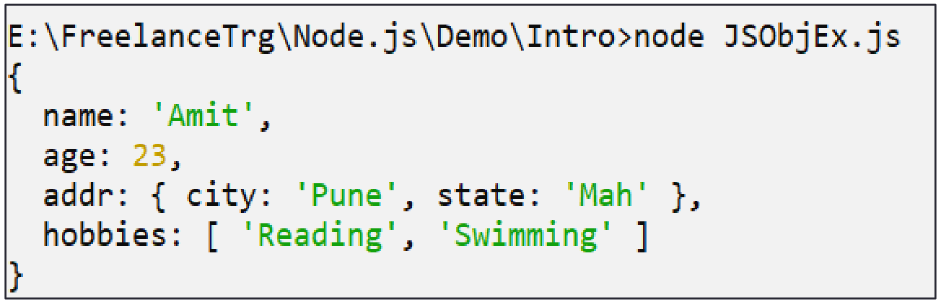
```
for (var i = 1; i < 11; i++)
    console.log(i);
var arr1 = [10, 20, 30];
arr1.push(40,50);
console.log('arr length: ' + arr1.length); //5
console.log('arr contents: ' + arr1);  //10,20,30,40,50
```



```
//functionEx.js
function foo() {
    return 123;
    }
console.log(foo()); // 123

function bar() { }
console.log(bar()); // undefined
```



```
//JSObjEx.js
var person = {
    name: "Amit",
    age: 23,
    addr: {
        city: 'Pune',
        state: 'Mah'
    },
    hobbies: ['Reading', 'Swimming']
};
console.log(person);
```



```
E:\FreelanceTrg\Node.js\Demo\Intro>node JSObjEx.js
{
  name: 'Amit',
  age: 23,
  addr: { city: 'Pune', state: 'Mah' },
  hobbies: [ 'Reading', 'Swimming' ]
}
```

Node js Modules

- A module in Node.js is a logical encapsulation of code in a single unit.
  - Since each module is an independent entity with its own encapsulated functionality, it can be managed as a separate unit of work.

- Consider modules to be the same as JavaScript libraries.
  - A set of functions you want to include in your application.
  - Module in Node.js is a simple or complex functionality organized in JavaScript files which can be reused throughout a Node.js application.
  - A module is a discrete program, contained in a single file in Node.js. Modules are therefore tied to files, with one module per file.

- Node.js has a set of built-in modules which you can use without any further installation.
  - Built-in modules provide a core set of features we can build upon.
  - Also, each module can be placed in a separate .js file under a separate folder.
  - To include a module, use the **require()** function with the name of the module.

- In Node, modules are referenced either by file path or by name
  - For example, we can require some native modules:

```
var http = require('http');
var dns = require('dns');
```

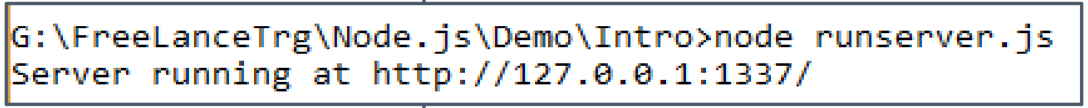
```
var myFile = require('./myFile'); // loads myFile.js
```

Node.js Web App

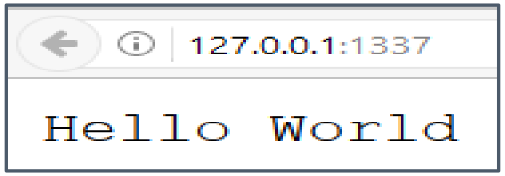
```
//RunServer.js
var http = require("http");

function process_request(req, res) {
   var body = 'Hello World\n';
   var content_length = body.length ;
   res.writeHead(200, {
              'Content-Length': content_length,
              'Content-Type': 'text/plain'   });

    res.end(body);
}

var srv = http.createServer(process_request);
srv.listen(1337, '127.0.0.1');

console.log('Server running at http://127.0.0.1:1337/');
```

Import required module using **require**; load http module and store returned HTTP instance into http variable

createServer() : turns your computer into an HTTP server
Creates an HTTP server which listens for request over 1337 port on local machine



```
G:\FreeLanceTrg\Node.js\Demo\Intro>node runserver.js
Server running at http://127.0.0.1:1337/
```



127.0.0.1:1337

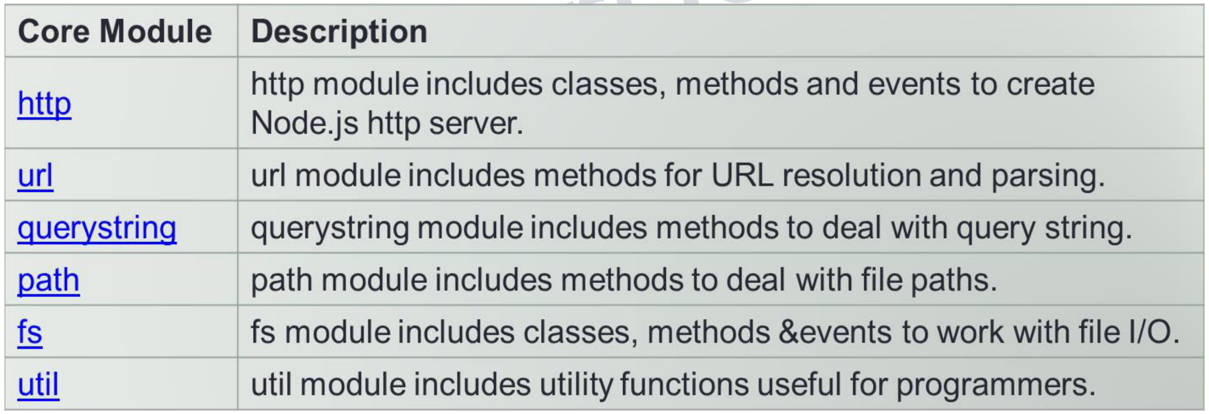
Hello World

```
var http = require('http');
http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello World\n');
   })
  .listen(1337, '127.0.0.1');
```

Node.js Module

- Node.js includes three types of modules:
  - Core Modules
  - Local Modules
  - Third Party Modules

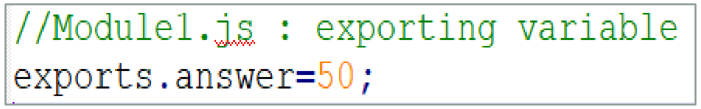
- Loading a core module
  - Node has several modules compiled into its binary distribution called core modules.
  - It is referred solely by the module name, not by the path and are preferentially loaded even if a third-party module exists with the same name.
  - var http = require('http');
- Some of the important core modules in Node.js



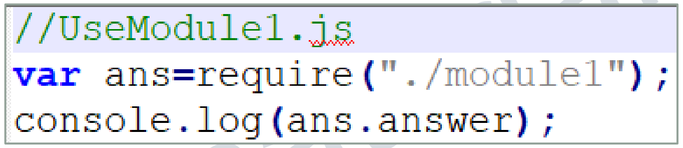
Core Module	Description
http	http module includes classes, methods and events to create Node.js http server.
url	url module includes methods for URL resolution and parsing.
querystring	querystring module includes methods to deal with query string.
path	path module includes methods to deal with file paths.
fs	fs module includes classes, methods &events to work with file I/O.
util	util module includes utility functions useful for programmers.

Node.js Local Module

- The local modules are custom modules that are created locally by developer in the app
  - These modules can include various functionalities bundled into distinct files and folders
  - You can also package it and distribute it via NPM, so that Node.js community can use it.
  - For example, if you need to connect to MongoDB and fetch data then you can create a module for it, which can be reused in your application.

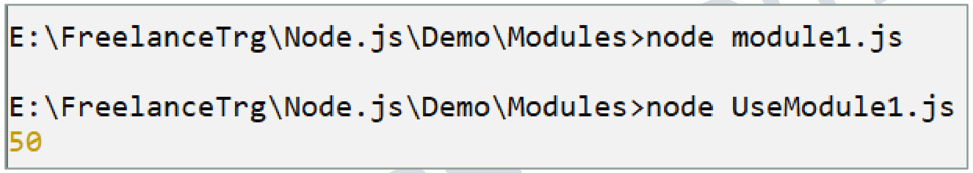


```
//Module1.js : exporting variable
exports.answer=50;
```

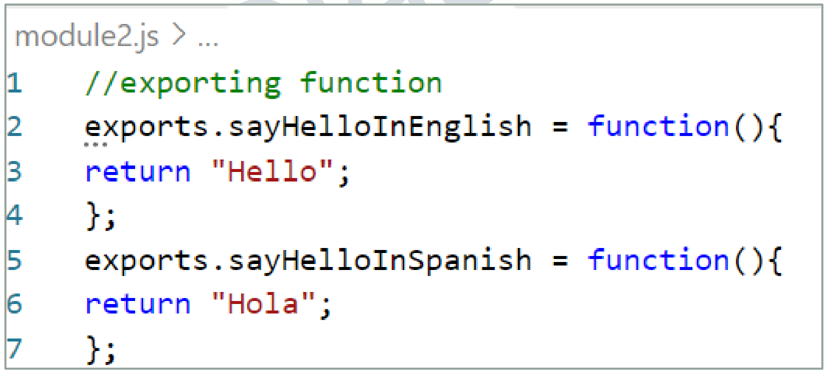


```
//UseModule1.js
var ans=require("./module1");
console.log(ans.answer);
```

exports object is a special ***object*** created by the Node module system which is returned as the value of the require function when you include that module.

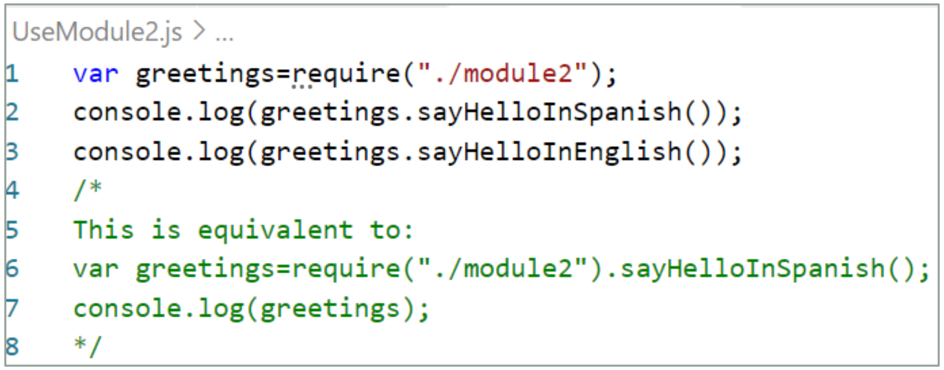


```
E:\FreelanceTrg\Node.js\Demo\Modules>node module1.js

E:\FreelanceTrg\Node.js\Demo\Modules>node UseModule1.js
50
```



module2.js > ...

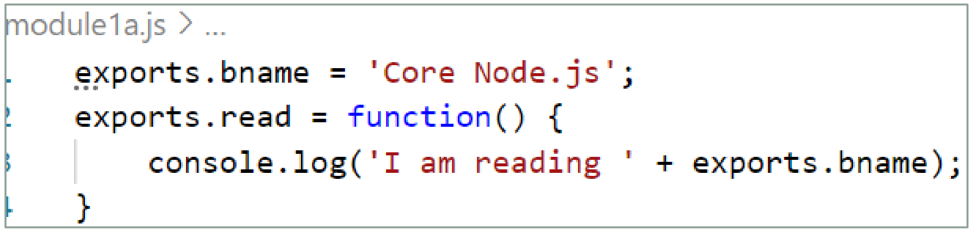
```
//exporting function
exports.sayHelloInEnglish = function(){
return "Hello";
};
exports.sayHelloInSpanish = function(){
return "Hola";
};
```



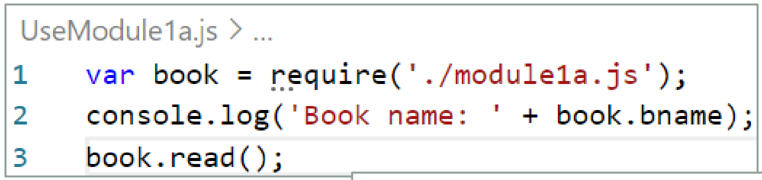
UseModule2.js > ...

```
var greetings=require("./module2");
console.log(greetings.sayHelloInSpanish());
console.log(greetings.sayHelloInEnglish());
/*
This is equivalent to:
var greetings=require("./module2").sayHelloInSpanish();
console.log(greetings);
*/
```

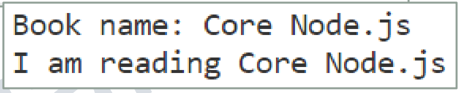
Create Your Own Modules


module1a.js > ...

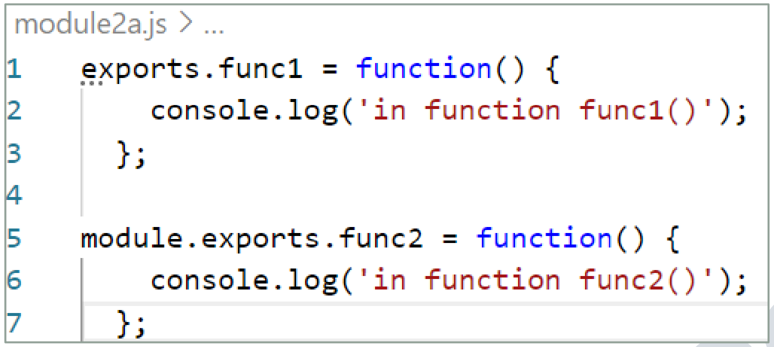
```
exports.bname = 'Core Node.js';
exports.read = function() {
    console.log('I am reading ' + exports.bname);
}
```


UseModule1a.js > ...

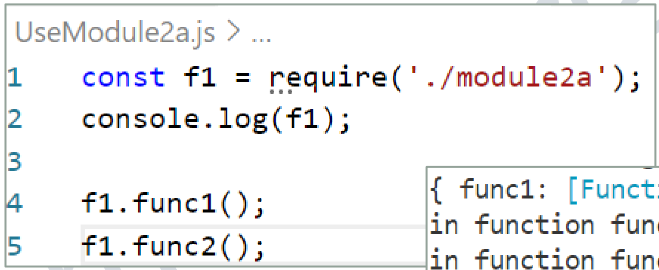
```
var book = require('./module1a.js');
console.log('Book name: ' + book.bname);
book.read();
```



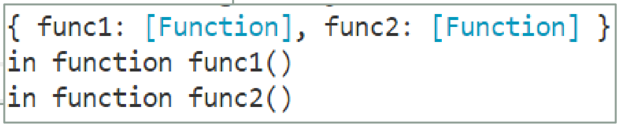
```
Book name: Core Node.js
I am reading Core Node.js
```


module2a.js > ...

```
exports.func1 = function() {
    console.log('in function func1()');
};

module.exports.func2 = function() {
    console.log('in function func2()');
};
```

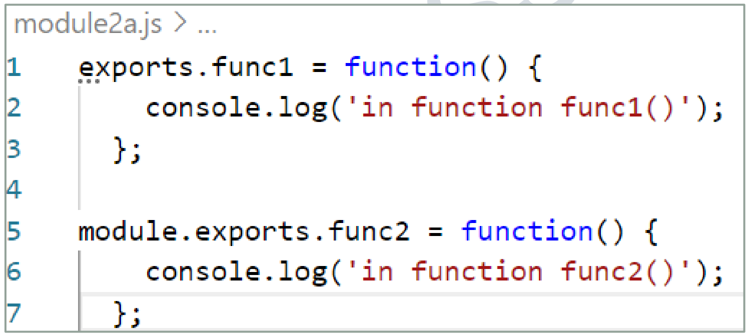

UseModule2a.js > ...

```
const f1 = require('./module2a');
console.log(f1);

f1.func1();
f1.func2();
```

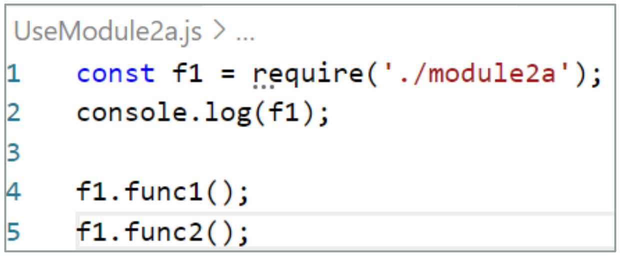


```
{ func1: [Function], func2: [Function] }
in function func1()
in function func2()
```

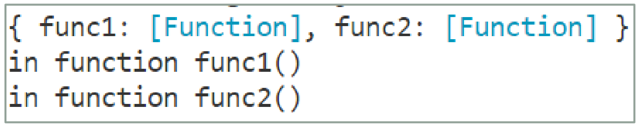
- Exports is just module.exports's little helper. Your module returns module.exports to the caller ultimately, not exports. All exports does is collect properties and attach them to module.exports


module2a.js > ...

```
exports.func1 = function() {
    console.log('in function func1()');
};

module.exports.func2 = function() {
    console.log('in function func2()');
};
```


UseModule2a.js > ...

```
const f1 = require('./module2a');
console.log(f1);

f1.func1();
f1.func2();
```

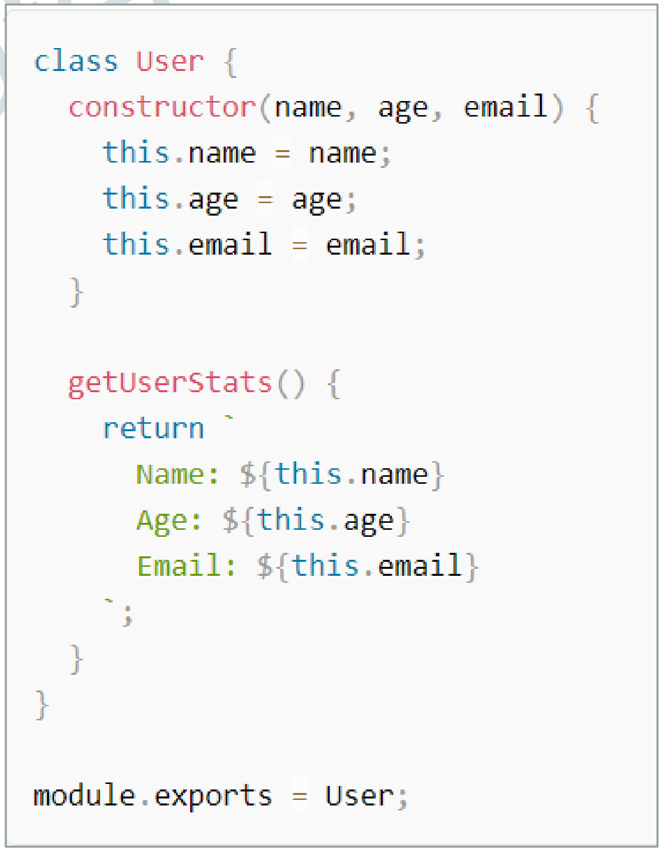


```
{ func1: [Function], func2: [Function] }
in function func1()
in function func2()
```

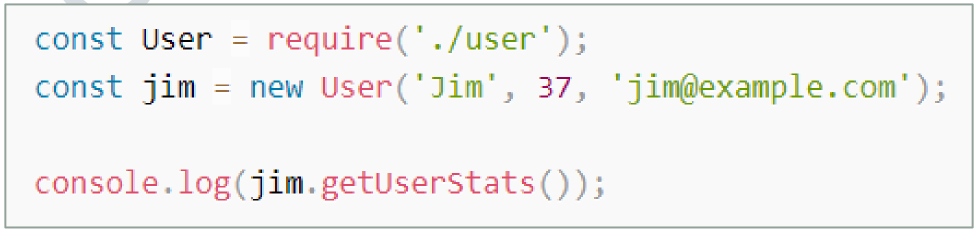
```
//module7.js
function sayHello(){
  console.log("Hello World!")
}

const username = 'Joan'

module.exports = {username, sayHello}

//or, using dot notation:
module.exports.username = username
module.exports.sayHello = sayHello

//alternatively
exports.username = username
exports.sayHello = sayHello
```

```
//import the assigned properties with a destructuring assignment:
const { username, sayHello } = require('./module7')

//Or with a regular assignment and dot notation:

const helloModule = require('./module7')

helloModule.sayHello()
console.log(helloModule.username)
```

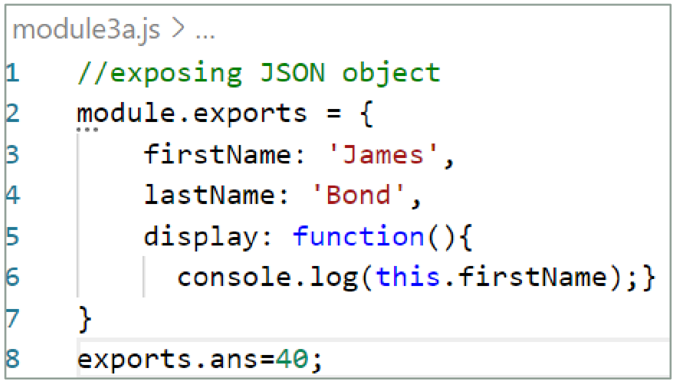
- To Expose properties and functions we use exports
- On other hand, to expose user defined objects (class) and JSON objects, we use module.exports



```
class User {
  constructor(name, age, email) {
    this.name = name;
    this.age = age;
    this.email = email;
  }

  getUserStats() {
    return `
      Name: ${this.name}
      Age: ${this.age}
      Email: ${this.email}
    `;
  }
}

module.exports = User;
```

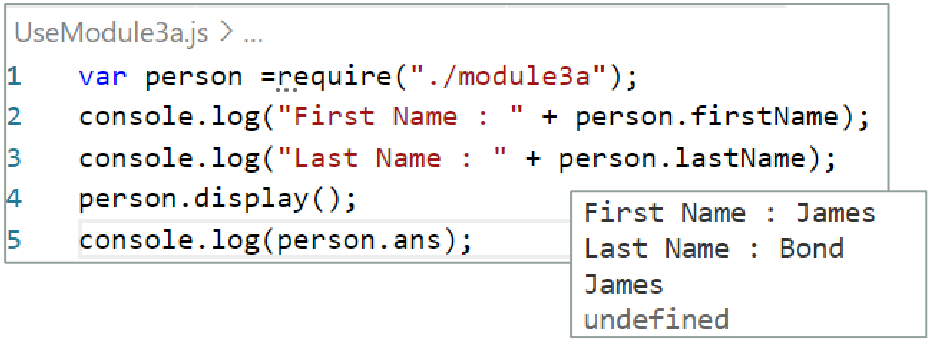


```
const User = require('./user');
const jim = new User('Jim', 37, 'jim@example.com');

console.log(jim.getUserStats());
```



module3a.js > ...

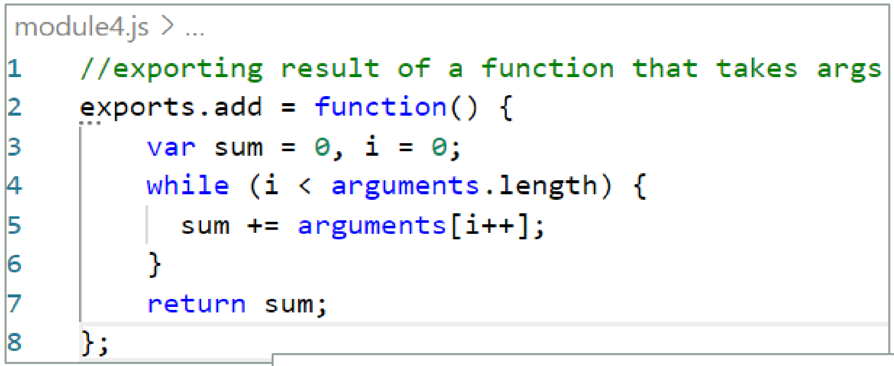
```javascript
//exposing JSON object
module.exports = {
    firstName: 'James',
    lastName: 'Bond',
    display: function(){
      console.log(this.firstName);}
}
exports.ans=40;
```



UseModule3a.js > ...

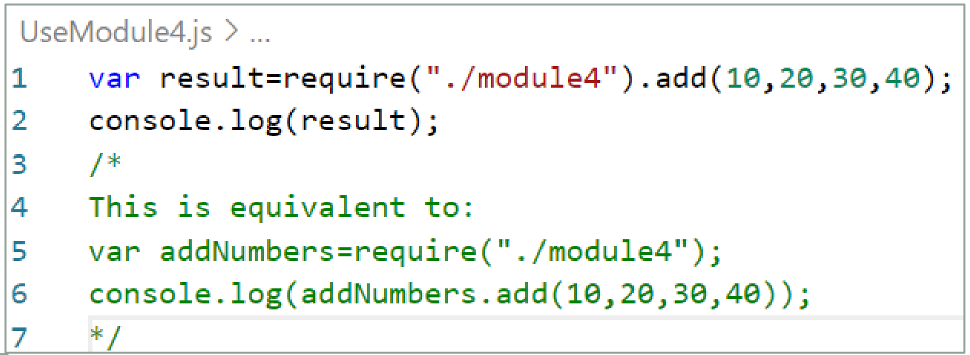
```javascript
var person =require("./module3a");
console.log("First Name : " + person.firstName);
console.log("Last Name : " + person.lastName);
person.display();
console.log(person.ans);
```

```
First Name : James
Last Name : Bond
James
undefined
```



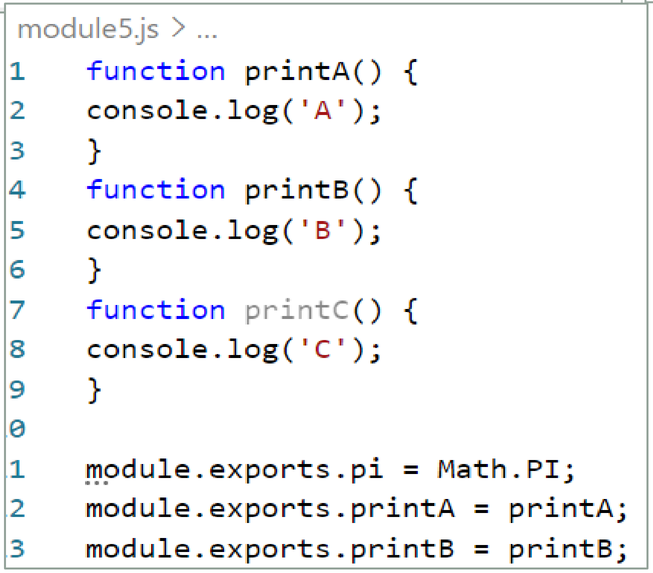
module4.js > ...

```javascript
//exporting result of a function that takes args
exports.add = function() {
    var sum = 0, i = 0;
    while (i < arguments.length) {
      sum += arguments[i++];
    }
    return sum;
};
```



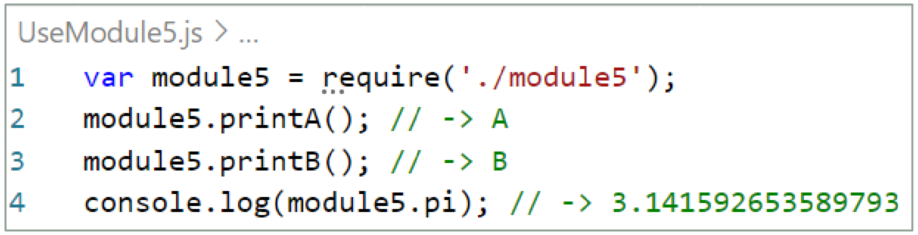
UseModule4.js > ...

```javascript
var result=require("./module4").add(10,20,30,40);
console.log(result);
/*
This is equivalent to:
var addNumbers=require("./module4");
console.log(addNumbers.add(10,20,30,40));
*/
```



module5.js > ...

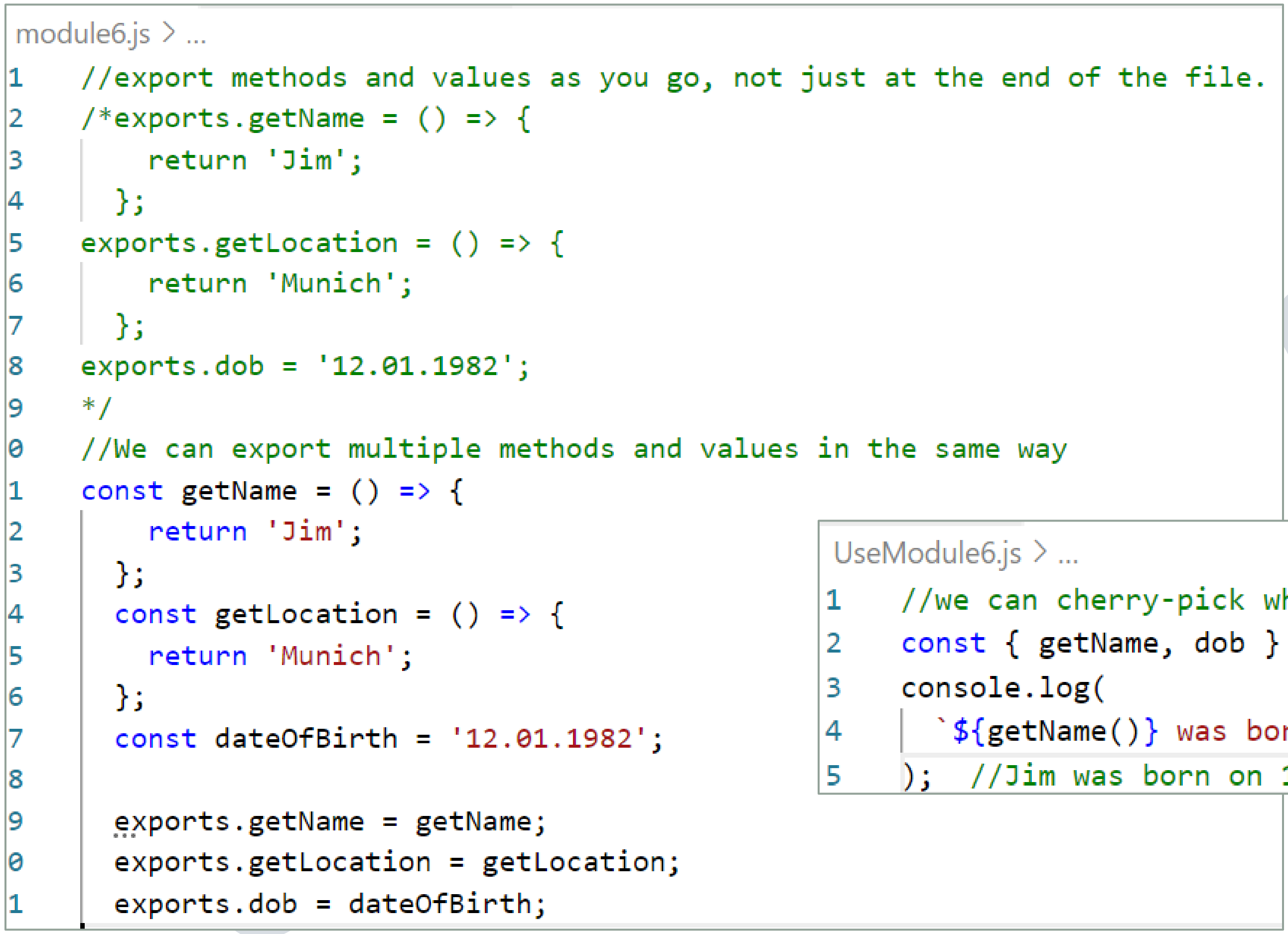
```javascript
function printA() {
console.log('A');
}
function printB() {
console.log('B');
}
function printC() {
console.log('C');
}

module.exports.pi = Math.PI;
module.exports.printA = printA;
module.exports.printB = printB;
```



UseModule5.js > ...

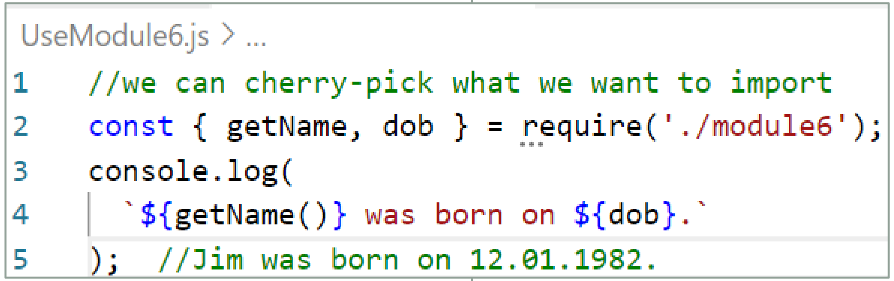
```javascript
var module5 = require('./module5');
module5.printA(); // -> A
module5.printB(); // -> B
console.log(module5.pi); // -> 3.141592653589793
```

Example



module6.js > ...

```
//export methods and values as you go, not just at the end of the file.
/*exports.getName = () => {
    return 'Jim';
  };
exports.getLocation = () => {
    return 'Munich';
  };
exports.dob = '12.01.1982';
*/
//We can export multiple methods and values in the same way
const getName = () => {
    return 'Jim';
  };
  const getLocation = () => {
    return 'Munich';
  };
  const dateOfBirth = '12.01.1982';

  exports.getName = getName;
  exports.getLocation = getLocation;
  exports.dob = dateOfBirth;
```

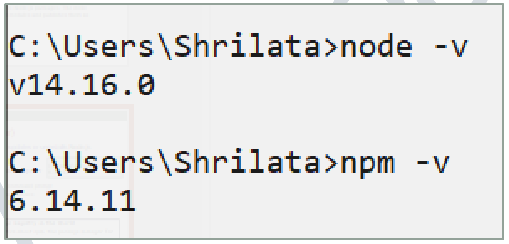


UseModule6.js > ...

```
//we can cherry-pick what we want to import
const { getName, dob } = require('./module6');
console.log(
  `${getName()} was born on ${dob}.`
);  //Jim was born on 12.01.1982.
```

NPM (Node Package Manager)

- Loading a module(Third party) installed via NPM
  - To use the modules written by other people in the Node community and published on the Internet (**npmjs.com**).
  - We can install those third party modules using the **Node Package Manager** which is installed by default with the node installation.
  - Node Package Manager (NPM) is a command line tool that installs, updates or uninstalls Node.js packages in your application.
  - It is also an online repository for open-source Node.js packages. The node community around the world creates useful modules and publishes them as packages in this repository.
  - NPM is a command line tool that installs, updates or uninstalls Node.js packages in your application.
  - After you install Node.js, verify NPM installation : **npm -v**



```
C:\Users\Shrilata>node -v
v14.16.0

C:\Users\Shrilata>npm -v
6.14.11
```

NPM (Node Package Manager)

- Installing Packages
  - In order to use a module, you must install it on your machine.
  - To install a package, type npm install, followed by the package name

- There are two ways to install a package using npm: globally and locally.
- **Globally** − This method is generally used to install development tools and CLI based packages. To install a package globally, use the following code.
  - **npm install -g <package-name>**
  - Eg to install expressJS : npm install -g express
  - Eg to install Typescript : npm install –g typescript
  - Eg to install Angular : npm install -g @angular/cli

- **Locally** − This method is generally used to install frameworks and libraries. A locally installed package can be used only within the directory it is installed.
  - To install a package locally, use the same command as above without the -g flag.
  - **npm install <package-name>**
  - Eg : To install cookie parser in Express : npm install --save cookie-parser
  - Eg: to install bootstrap : npm install bootstrap@4.1.1

NPM (Node Package Manager)

- When packages are installed, they are saved on local machine
- npm installs module packages to the node_modules folder.

- **Installing a package using NPM :** *$ npm install [g] <Package Unique Name>*

- **To remove an installed package :** *npm uninstall [g] < Package Unique Name>*

- **To update a package to its latest version :** *npm update [g] < Package Unique Name>*

Loading a third party module : package.json

- The package.json file in Node.js is the heart of the entire application.
  - It is basically the manifest file that contains the metadata of the project.
  - package.json is a configuration file from where the npm can recognize dependencies between packages and installs modules accordingly.
  - It must be located in project's root directory.
  - It contains human-readable metadata about the project (like the project name and description) as well as functional metadata like the package version number and a list of dependencies required by the application.
  - Your project also must include a package.json before any packages can be installed from NPM.
  - Eg : a minimal package.json:

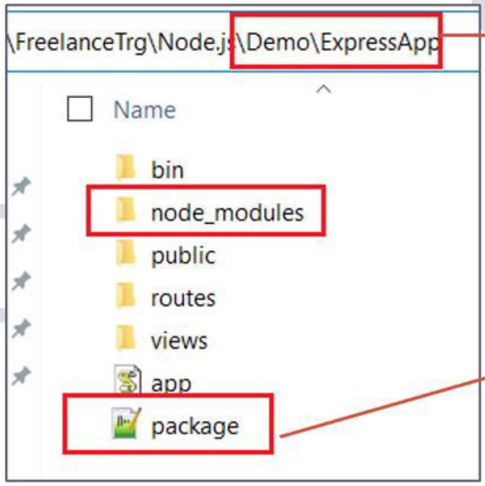
```
{
  "name" : "barebones",
  "version" : "0.0.0",
}
```

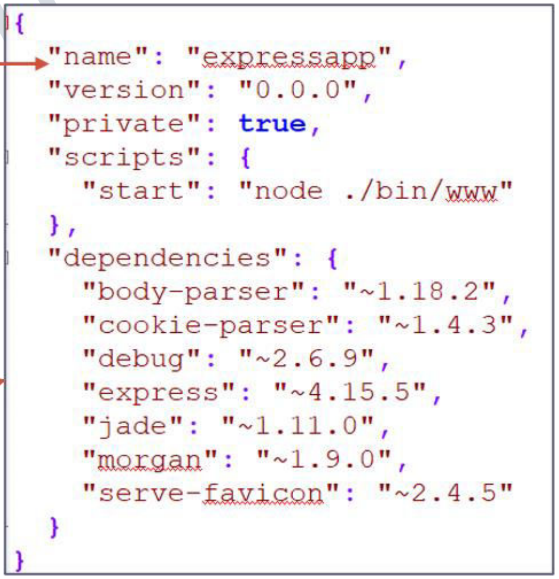
  - The name field should explain itself: this is the name of your project. The version field is used by npm to make sure the right version of the package is being installed.

Loading a third party module

- Lets say I want to create a ExpressJS application. I will install ExpressJs locally.
  - Step-1) choose a empty folder
  - Step-2) run npm init to create a package.json file
  - Step-3) install express : npm install express –save (to update package.json)
  - Step-4) check the updated json file to see new dependencies

  - See how package.json is installed in root folder along with node_modules folder
  - My Express app is dependent on a number of other modules
  - All these dependencies will have an entry in package.json





```
{
  "name": "expressapp",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "body-parser": "~1.18.2",
    "cookie-parser": "~1.4.3",
    "debug": "~2.6.9",
    "express": "~4.15.5",
    "jade": "~1.11.0",
    "morgan": "~1.9.0",
    "serve-favicon": "~2.4.5"
  }
}
```

Loading a file module

- Loading a file module (User defined module)
  - We load non-core modules by providing the absolute path / relative path.
  - Node will automatically add the .js extension to the module referred.
  - var myModule = require(‘d:/shrilata/nodejs/module'); // Absolute path for module.js
  - var myModule = require('**../**module'); // Relative path for module.js (one folder up level)
  - var myModule = require('./module'); // Relative path for module.js (Exists in current directory)

If the given path does not exist, require() will throw an Error with its code property set to 'MODULE_NOT_FOUND'.

package.json

- The package.json file in Node.js is the heart of the entire application.
- It is basically the manifest file that contains the metadata of the project.
- package.json is a configuration file from where the npm can recognize dependencies between packages and installs modules accordingly.
  - It must be located in project’s root directory.

- package-lock.json file
  - Introduced in version 5; keeps track of the exact version of every package that is installed so that a product is 100% reproducible in the same way even if packages are updated by their maintainers.
  - The package-lock.json sets your currently installed version of each package in stone, and npm will use those exact versions when running npm install.

Buffers

- A buffer is an area of memory; It represents a fixed-size chunk of memory (can't be resized) allocated outside of the V8 JavaScript engine.
  - You can think of a buffer like an array of integers, which each represent a byte of data.
  - It is implemented by the Node.js Buffer class.

- **Creating Buffer**
  - It is possible to create your own buffer! Aside from the one Node.js will automatically create during a stream, it is possible to create and manipulate your own buffer
  - A buffer is created using the : Buffer.alloc(), Buffer.allocUnsafe() , Buffer.from()

  - Buffer.alloc(size, fill, encoding);
    - Size: Desired length of new Buffer. It accepts integer type of data.
    - Fill: The value to prefill the buffer. The default value is 0.It accepts any of the following: integer, string, buffer type of data.
    - Encoding: It is Optional. If buffer values are string , default encoding type is utf8. Supported values are: ("ascii","utf8","utf16le","ucs2","base64","latin1", "binary", "hex")

  - Eg : Create a buffer of length 20, with initializing all the value to fill as 0 in hexadecimal format

```
var zerobuf = Buffer.alloc(20);
console.log(zerobuf);
//<Buffer 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00>
```

Node.js fs (File System) Module

- The fs module provides a lot of very useful functionality to access and interact with the file system.
  - There is no need to install it. Being part of the Node.js core, it can be used by simply requiring it:
  - const fs = require('fs')
- This module provides a wrapper for the standard file I/O operations.
- All the methods in this module has asynchronous and synchronous forms.
  - synchronous methods in this module ends with 'Sync'. For instance renameSync() is the synchronous method for rename() synchronous method.
  - The asynchronous form always take a completion callback as its last argument.
  - The arguments passed to the completion callback depend on the method, but the first argument is always reserved for an exception. If the operation was completed successfully, then the first argument will be null or undefined.
  - When using the synchronous form any exceptions are immediately thrown. You can use try/catch to handle exceptions or allow them to bubble up.

Asynchronous method is preferred over synchronous method because it never blocks the program execution where as the synchronous method blocks.

Node.js File System

- Node fs module provides an API to interact with FileSystem and to perform some IO operations like create a file, read a File, delete a File etc..
  - fs module is responsible for all the async or synchronous file I/O operations.

```
var fs = require('fs');
// write
fs.writeFileSync('test.txt', 'Hello fs!');
// read
console.log(fs.readFileSync('test.txt')); //<Buffer 48 65 6c 6c 6f 20 66 73 21>
console.log(fs.readFileSync('test.txt').toString());   //Hello fs!

console.log(fs.readFileSync('test.txt','utf8'));  //Hello fs!
```

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

// Asynchronous read
fs.readFile('test.txt', function (err, data) {
   if (err) {
      return console.error(err);
   }
   console.log("Asynchronous read: " + data.toString());
});

// Synchronous read
var data = fs.readFileSync('test.txt');
console.log("Synchronous read: " + data.toString());

console.log("Program Ended");
```

```
Synchronous read: Hello fs!
Program Ended
Asynchronous read: Hello fs!
```

```
var fs = require('fs');
fs.writeFile('test.txt', 'Hello World!', function (err) {
        if (err)
            console.log(err);
        else
            console.log('Write operation complete.');
});
```

Node.js File System

- fs.readFile(fileName [,options], callback) : read the physical file asynchronously.
- fs.writeFile(filename, data [, options], callback) : writes data to a file
- fs.appendFile(): appends the content to an existing file
- fs.unlink(path, callback); delete an existing file

```
var fs = require("fs")
fs.unlink("test1.txt", function(err){
    if(err) console.log("Err : " , err);
    console.log("delete successful")
 })
```

- fs.exists(path, callback) : determines if specified file exists or not
- fs.close(fd, callback(err));
- fs.rename(oldPath, newPath, callback): rename a file or folder

```
fs.rename("src.json", "tgt.json", err => {
    if (err) {
      return console.error(err)
    }
    console.log('Rename operation complete.');
  });
```

Node.js File System

- The exists() and existsSync() methods are used to determine if a given path exists.
- Both methods take a path string as an argument.
  - If existsSync() is used, a Boolean value representing path's existence is returned.
  - If exists() is used, the same Boolean value is passed as an argument to the callback function.

```
var fs = require("fs");
var path = "/";

fs.exists(path, function(exists) {
    if (exists)
        console.log(path + " exists: " + exists);
    else
        console.error("Something is wrong!");
});
```

- Reading Directories
  - We can use the fs.readdir() method to list all the files and directories within a specified path:

```
const fs = require('fs')
fs.readdir('./', (err, files) => {
    if (err) {
        console.error(err)
        return
    }
    console.log('files: ', files)
})
```

Node.js File System

- fs.open(path, flags[, mode], callback) : opens a file for reading or writing in async
  - path - string having file name including path.
  - flags - tells the behavior of the file to be opened
  - mode - sets the file mode; defaults to 0666, readable and writeable.
  - callback - function which gets two arguments **(err, fd)**.

```
// Asynchronous - Opening File
console.log("Going to open file!");
fs.open(test.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
   console.log("File opened successfully!");
});
```

Flag	Description
r	Open file for reading. An exception occurs if the file does not exist.
r+	Open file for reading and writing. An exception occurs if the file does not exist.
rs	Open file for reading in synchronous mode.
w	Open file for writing. The file is created (if it does not exist) or overwritten (if it exists).
wx	Like 'w' but fails if path exists.
w+	Open file for read+write. The file is created (if it doesn't exist) or overwritten (if it exists).
wx+	Like 'w+' but fails if path exists.
a	Open file for appending. The file is created if it does not exist.
ax	Like 'a' but fails if path exists.
a+	Open file for reading and appending. The file is created if it does not exist.

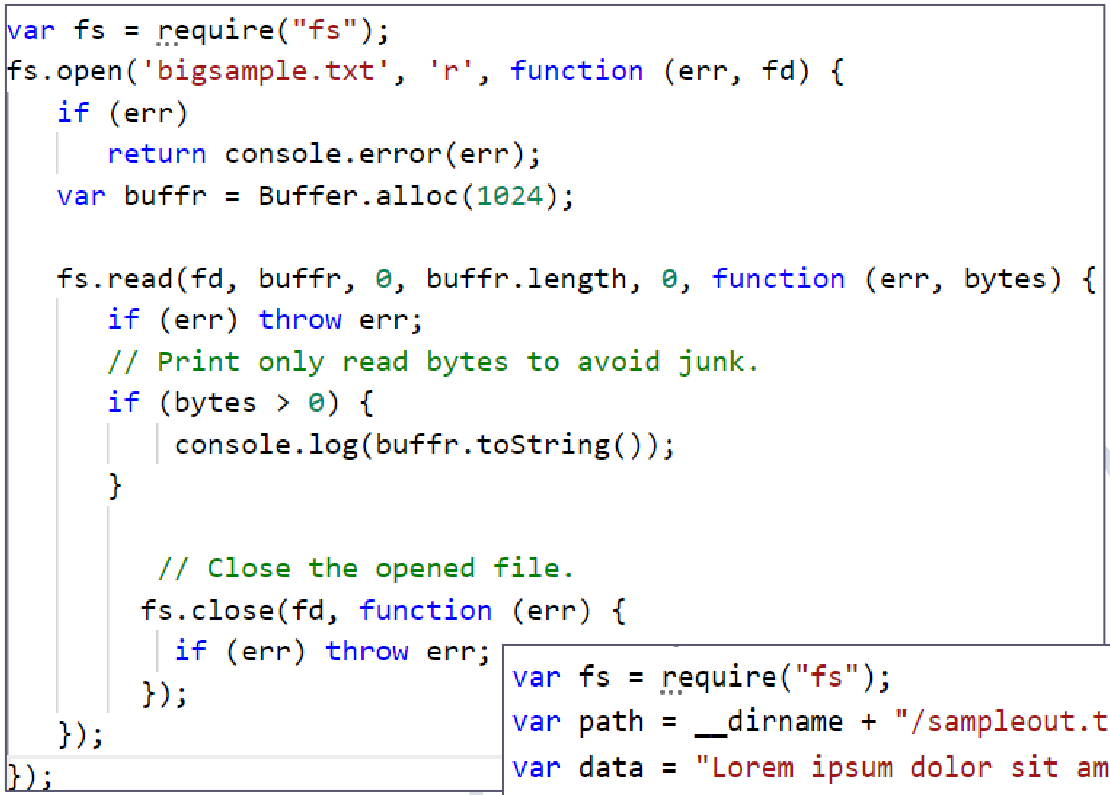
Node.js File System

- fs.read(fd, buffer, offset, length, position, callback) : reads an opened file; from the file specified by fd.
  - fd <Integer> - is a file descriptor, a handle, to the file
  - buffer <String> | <Buffer> : buffer into which the data will be read
  - offset <Integer> : offset in the buffer to start reading at.
  - length <Integer> : specifies the number of bytes to read
  - position <Integer> : specifies where to begin reading from in the file
  - callback (err, bytesRead)

- fs.write(fd, string[, position[, encoding]], callback): write into an opened file; specified by fd
- Alternatively :fs.write(fd, buffer[, offset[, length[, position]]], callback)

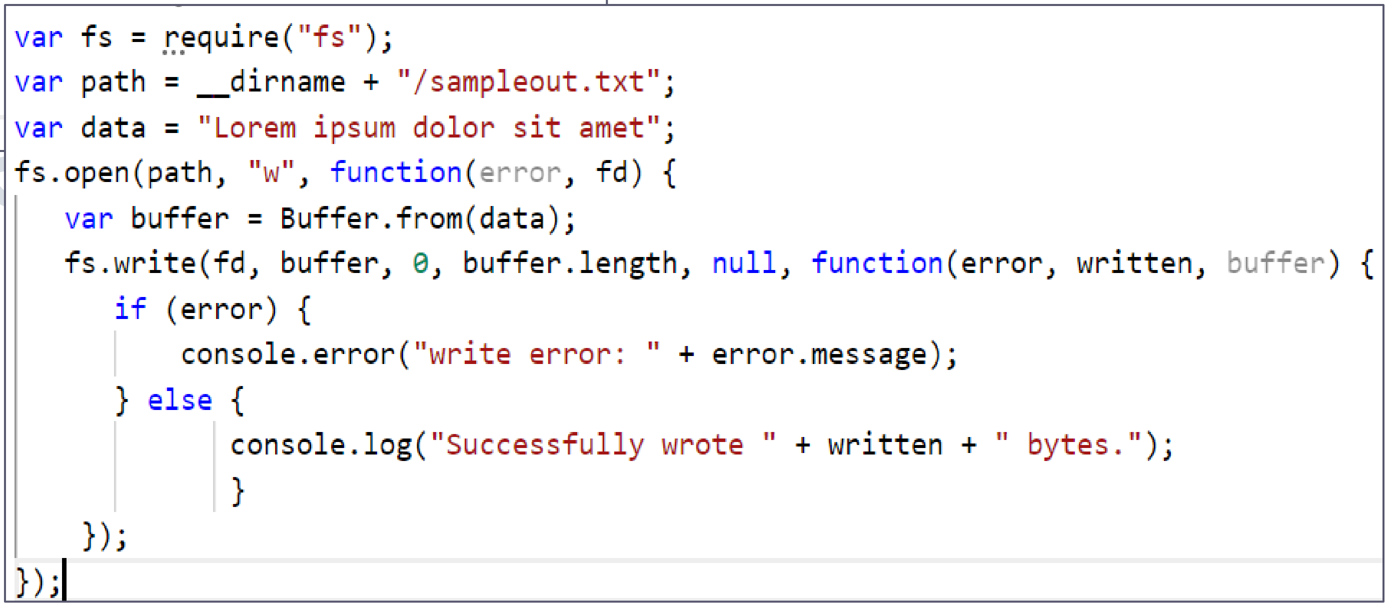
  - offset : offset in the buffer to start writing at
  - length : specifies the number of bytes to write.
  - position : specifies where to begin writing
  - The callback will be given three arguments (err, bytesWritten, buffer) where bytesWritten specifies how many bytes were written from buffer.

Example : read file and write to file



```
var fs = require("fs");
fs.open('bigsample.txt', 'r', function (err, fd) {
   if (err)
      return console.error(err);
   var buffr = Buffer.alloc(1024);

   fs.read(fd, buffr, 0, buffr.length, 0, function (err, bytes) {
      if (err) throw err;
      // Print only read bytes to avoid junk.
      if (bytes > 0) {
         console.log(buffr.toString());
      }

         // Close the opened file.
        fs.close(fd, function (err) {
         if (err) throw err;
        });
   });
});
```

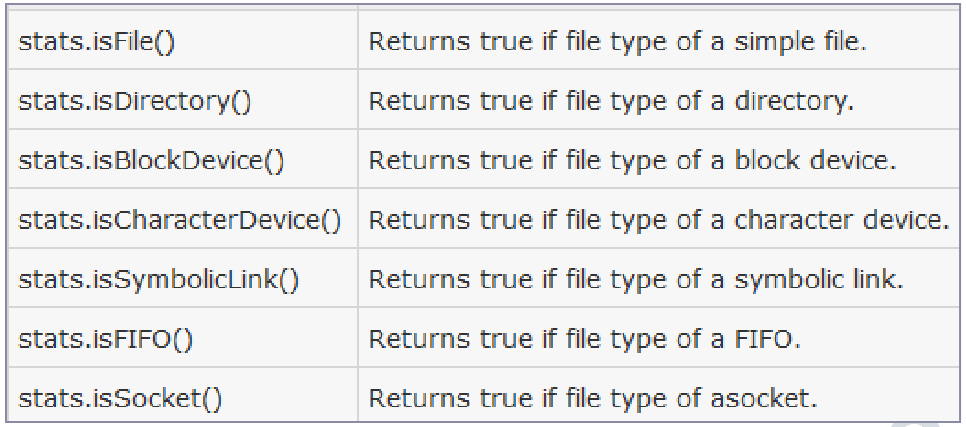
- __filename, is the absolute path of the currently executing file.
- __dirname is the absolute path to the directory containing the currently executing file



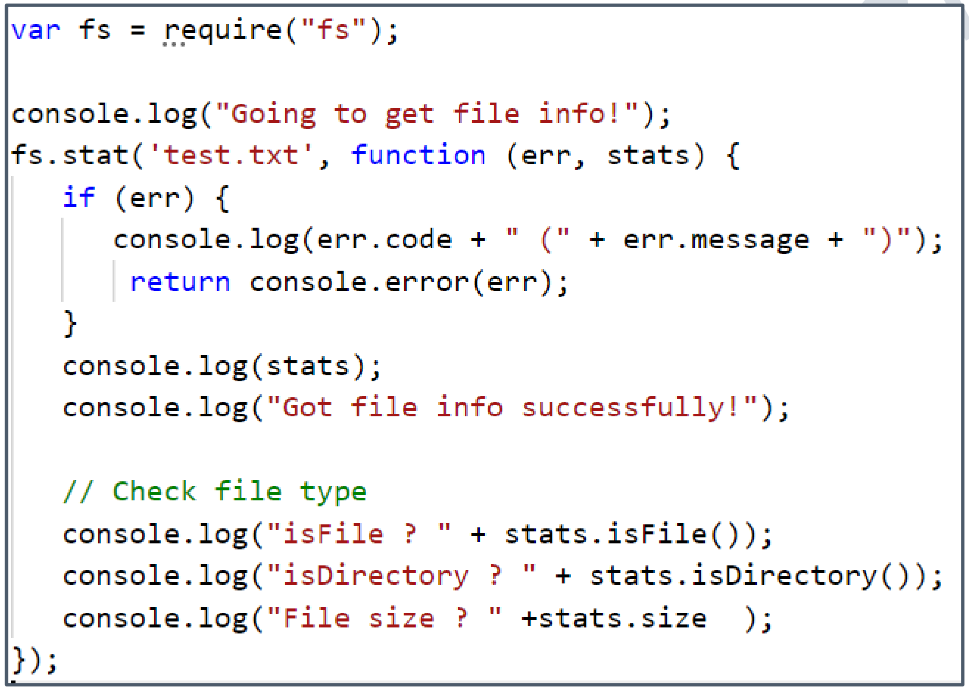
```
var fs = require("fs");
var path = __dirname + "/sampleout.txt";
var data = "Lorem ipsum dolor sit amet";
fs.open(path, "w", function(error, fd) {
   var buffer = Buffer.from(data);
   fs.write(fd, buffer, 0, buffer.length, null, function(error, written, buffer) {
      if (error) {
          console.error("write error:  " + error.message);
      } else {
             console.log("Successfully wrote " + written + " bytes.");
             }
    });
});
```

File System

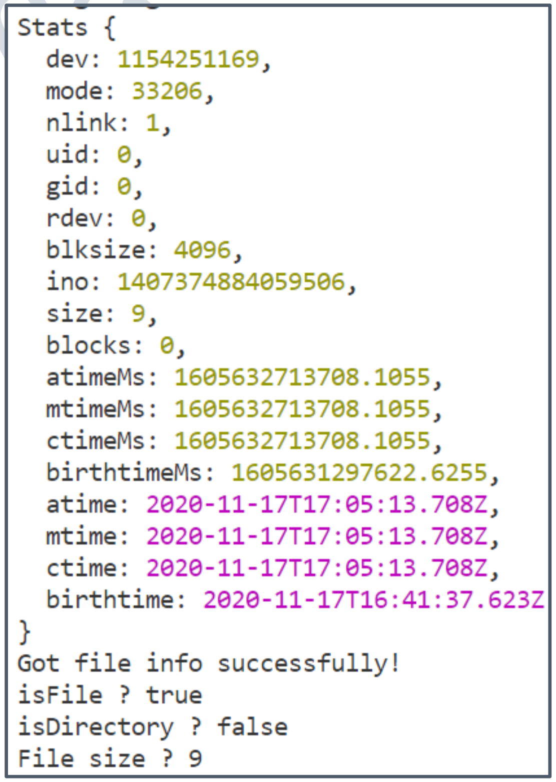
- fs.stat(path, callback) : gets the information about file on path
    - callback function gets two arguments (err, stats) where stats is an object of fs.Stats type



stats.isFile()	Returns true if file type of a simple file.
stats.isDirectory()	Returns true if file type of a directory.
stats.isBlockDevice()	Returns true if file type of a block device.
stats.isCharacterDevice()	Returns true if file type of a character device.
stats.isSymbolicLink()	Returns true if file type of a symbolic link.
stats.isFIFO()	Returns true if file type of a FIFO.
stats.isSocket()	Returns true if file type of asocket.

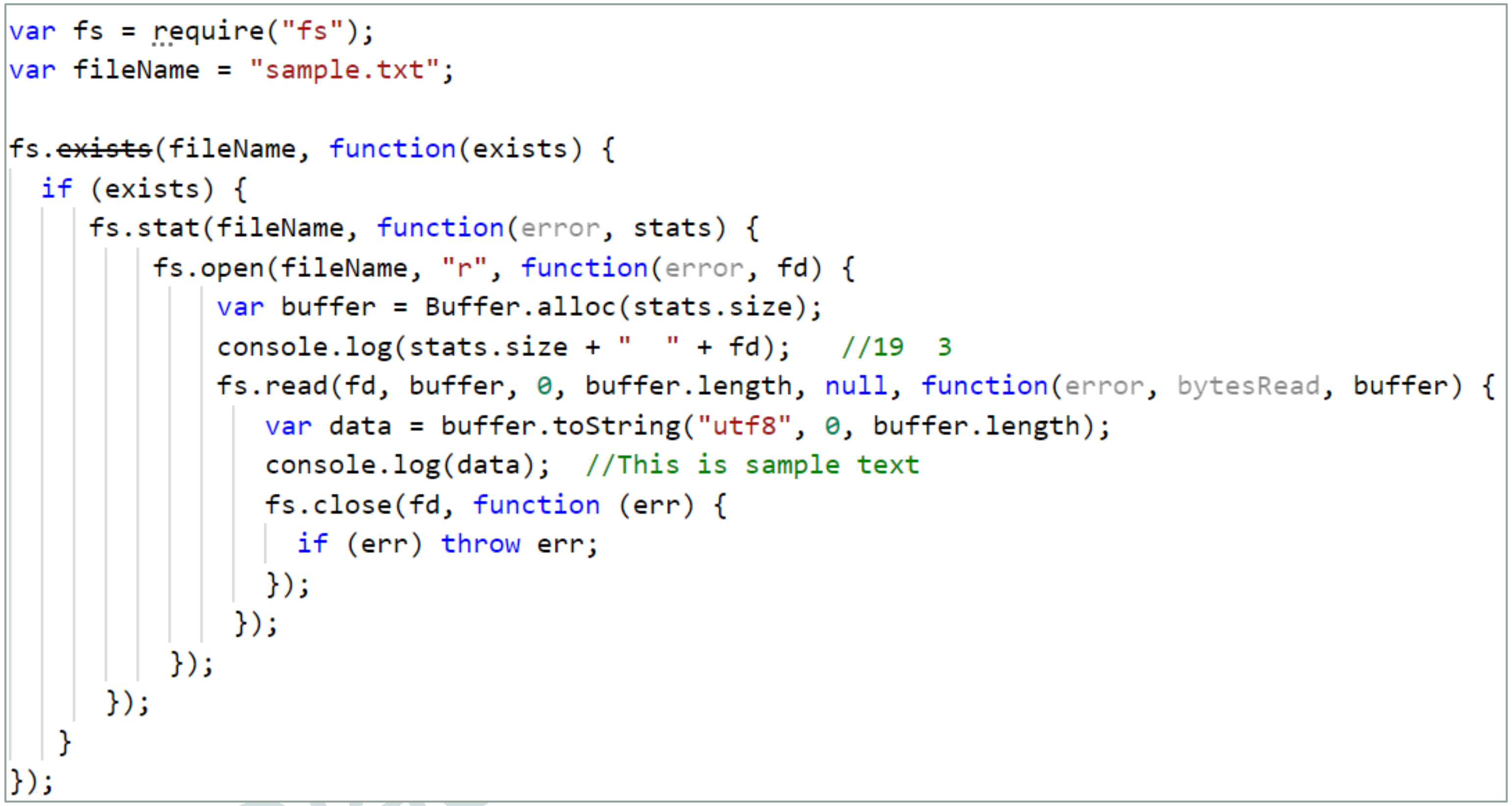


```
var fs = require("fs");

console.log("Going to get file info!");
fs.stat('test.txt', function (err, stats) {
   if (err) {
      console.log(err.code + " (" + err.message + ")");
       return console.error(err);
   }
   console.log(stats);
   console.log("Got file info successfully!");

   // Check file type
   console.log("isFile ? " + stats.isFile());
   console.log("isDirectory ? " + stats.isDirectory());
   console.log("File size ? " +stats.size  );
});
```



```
Stats {
  dev: 1154251169,
  mode: 33206,
  nlink: 1,
  uid: 0,
  gid: 0,
  rdev: 0,
  blksize: 4096,
  ino: 1407374884059506,
  size: 9,
  blocks: 0,
  atimeMs: 1605632713708.1055,
  mtimeMs: 1605632713708.1055,
  ctimeMs: 1605632713708.1055,
  birthtimeMs: 1605631297622.6255,
  atime: 2020-11-17T17:05:13.708Z,
  mtime: 2020-11-17T17:05:13.708Z,
  ctime: 2020-11-17T17:05:13.708Z,
  birthtime: 2020-11-17T16:41:37.623Z
}
Got file info successfully!
isFile ? true
isDirectory ? false
File size ? 9
```

File System : example

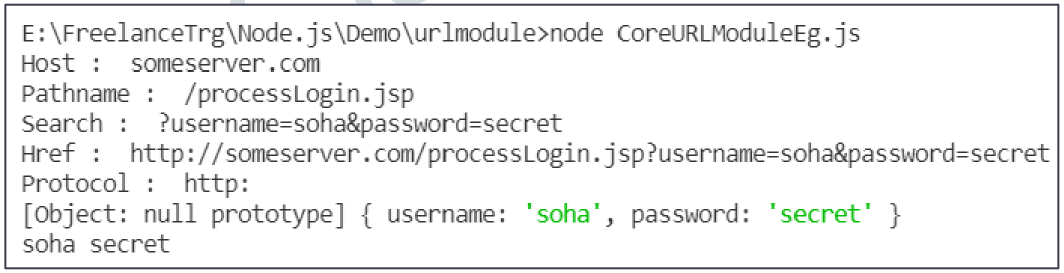


```
var fs = require("fs");
var fileName = "sample.txt";

fs.~~exists~~(fileName, function(exists) {
  if (exists) {
    fs.stat(fileName, function(error, stats) {
      fs.open(fileName, "r", function(error, fd) {
        var buffer = Buffer.alloc(stats.size);
        console.log(stats.size + "   " + fd);   //19  3
        fs.read(fd, buffer, 0, buffer.length, null, function(error, bytesRead, buffer) {
          var data = buffer.toString("utf8", 0, buffer.length);
          console.log(data);  //This is sample text
          fs.close(fd, function (err) {
            if (err) throw err;
          });
        });
      });
    });
  }
});
```

URL Core module

- The url module provides utilities for URL resolution and parsing.
  - It splits up a web address into readable parts.

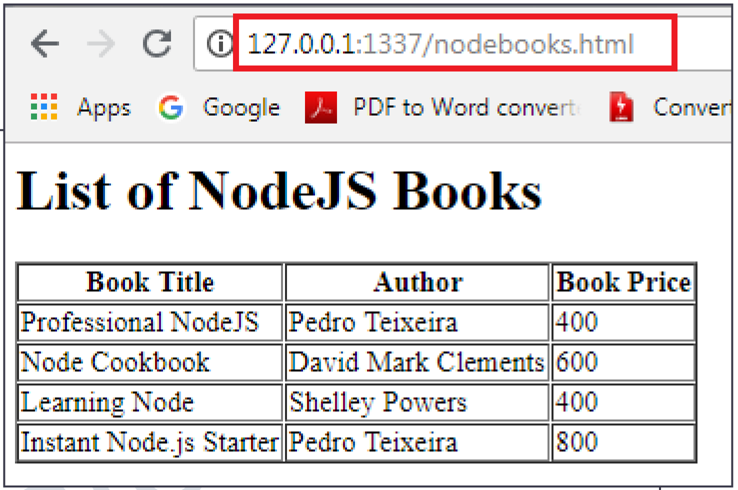
```
var url = require('url');
//var adr = 'http://localhost:8080/MyApp/welcome.html?year=2017&month=february';
var adr = 'http://someserver.com/processLogin.jsp?username=soha&password=secret';

var q = url.parse(adr, true);    //Parse an address

console.log("Host : " , q.host); //returns 'someserver.com'
console.log("Pathname : " , q.pathname); //returns '/processLogin.jsp'
console.log("Search : " , q.search); //returns '?username=soha&password=secret'
console.log("Href : " , q.href);  //returns 'http://someserver.com/processLogin.jsp?username=soha
&password=secret'
console.log("Protocol : " , q.protocol);  //returns 'http:'

var qdata = q.query; //returns an object: { year: 2017, month: 'february' }
console.log(qdata);   //returns '{ username: 'soha', password: 'secret' }'
console.log(qdata.username, qdata.password); //returns 'soha secret'
```



```
E:\FreelanceTrg\Node.js\Demo\urlmodule>node CoreURLModuleEg.js
Host :  someserver.com
Pathname :  /processLogin.jsp
Search :  ?username=soha&password=secret
Href :  http://someserver.com/processLogin.jsp?username=soha&password=secret
Protocol :  http:
[Object: null prototype] { username: 'soha', password: 'secret' }
soha secret
```

URL Core module

```
var http = require('http');
var url = require('url');
var fs = require('fs');
function process_request(req, res) {
  var q = url.parse(req.url, true);
  var filename = "." + q.pathname;
  fs.readFile(filename, function(err, data) {
  if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      res.end("404 Not Found");
  }
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(data);
  res.end();
 });
}

var s = http.createServer(process_request);
s.listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```



List of NodeJS Books

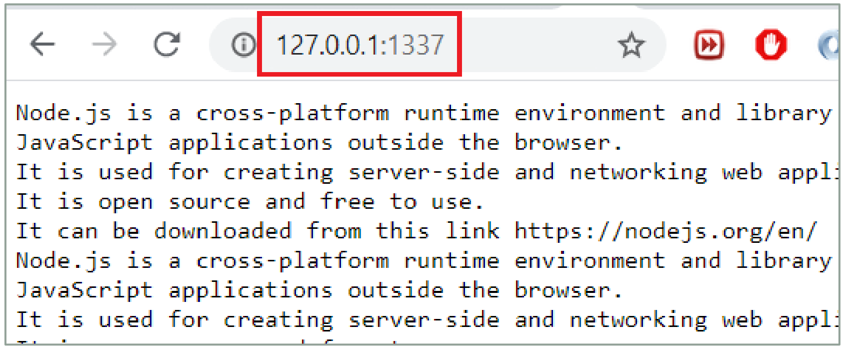
Book Title	Author	Book Price
Professional NodeJS	Pedro Teixeira	400
Node Cookbook	David Mark Clements	600
Learning Node	Shelley Powers	400
Instant Node.js Starter	Pedro Teixeira	800

Web development with Node : http.ServerRequest

- When listening for request events, the callback gets an http.ServerRequest object as the first argument (function(req,res))
- This object contains some properties:
  - req.url: This property contains the requested URL as a string
    - It does not contain the schema, hostname, or port, but it contains everything after that.
    - Eg : if URL is :http://localhost:3000/about?a=20 then req.url will return /about?a=20
  - req.method: This contains the HTTP method used on the request. It can be, for example, GET, POST, DELETE, or HEAD.
  - req.headers: This contains an object with a property for every HTTP header on the request.

  - Eg : Serving file on request : Reading a file at server end and serving to browser!

```
var http = require('http');
var fs = require('fs');
function process_request(req, res) {
   fs.readFile('bigsample.txt', function(err, data) {
      res.writeHead(200, {'Content-Type': 'text/plain'});
      res.write(data);
      res.end();
   });
}
var s = http.createServer(process_request);
s.listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```



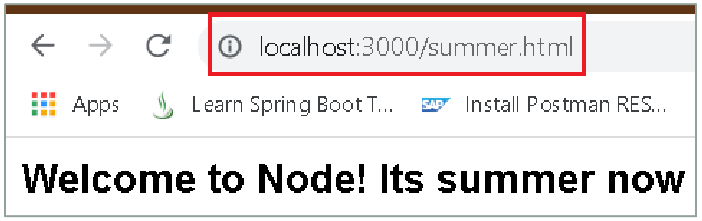
```
Node.js is a cross-platform runtime environment and library
JavaScript applications outside the browser.
It is used for creating server-side and networking web appl
It is open source and free to use.
It can be downloaded from this link https://nodejs.org/en/
Node.js is a cross-platform runtime environment and library
JavaScript applications outside the browser.
It is used for creating server-side and networking web appl
```

Serving file on request

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

 http.createServer(function (req, res) {
   var q = url.parse(req.url, true);
   var filename = "." + q.pathname;

  fs.readFile(filename, function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Not Found");
    }
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(3000);     //invoke as http://localhost:3000/summer.html+
```

./summer.html

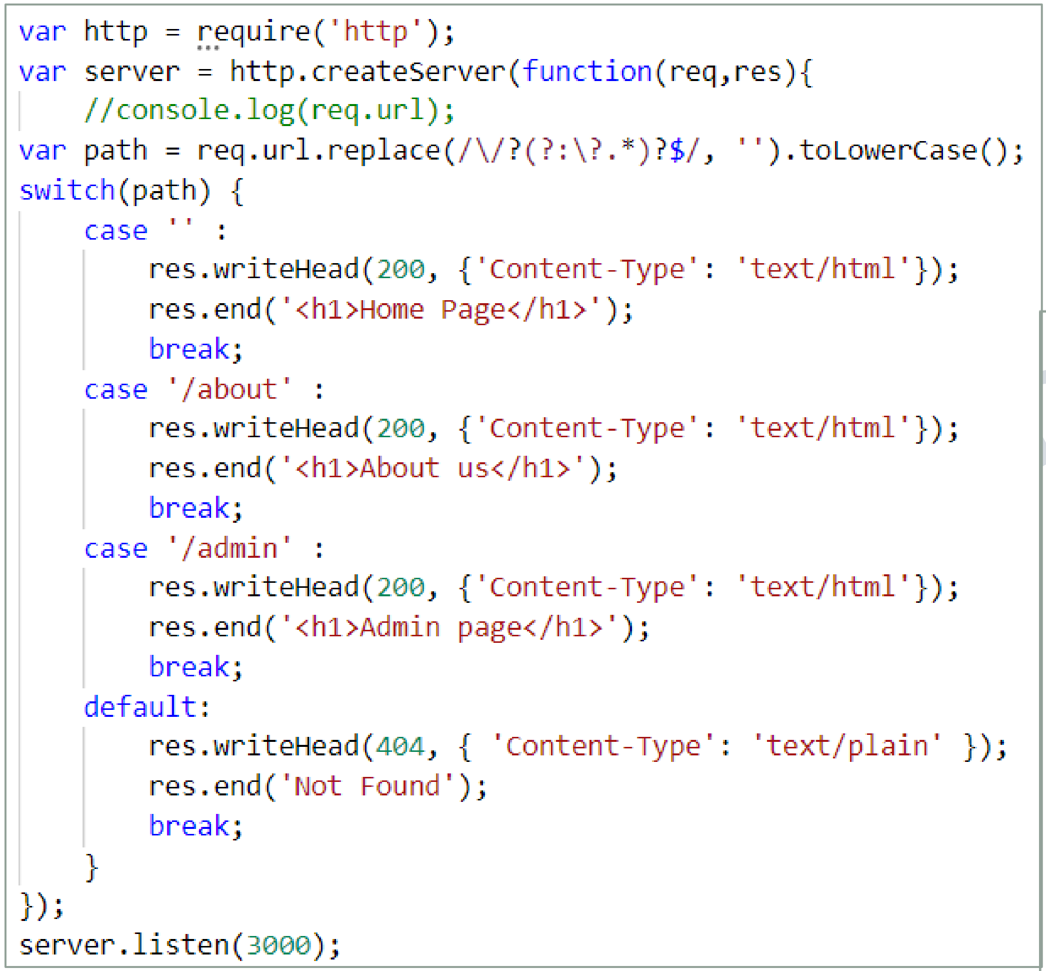
```
<!-- summer.html -->
<html>
<head>
<title>Hello there</title>
</head>
<body>
<h3>Welcome to Node!
Its summer now</h3>
</body>
</html>
```



Welcome to Node! Its summer now

Routing

- Routing refers to the mechanism for serving the client the content it has asked



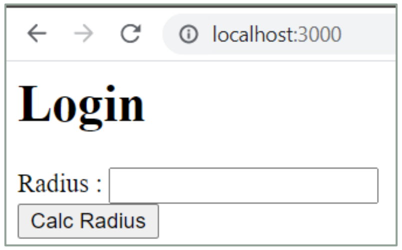
```
var http = require('http');
var server = http.createServer(function(req,res){
    //console.log(req.url);
var path = req.url.replace(/\/?(?:\?.*)?$/, '').toLowerCase();
switch(path) {
    case '' :
        res.writeHead(200, {'Content-Type': 'text/html'});
        res.end('<h1>Home Page</h1>');
        break;
    case '/about' :
        res.writeHead(200, {'Content-Type': 'text/html'});
        res.end('<h1>About us</h1>');
        break;
    case '/admin' :
        res.writeHead(200, {'Content-Type': 'text/html'});
        res.end('<h1>Admin page</h1>');
        break;
    default:
        res.writeHead(404, { 'Content-Type': 'text/plain' });
        res.end('Not Found');
        break;
    }
});
server.listen(3000);
```

```
var http = require("http");
http.createServer(function(request, response) {
  if (request.url === "/" && request.method === "GET") {
       response.writeHead(200, { "Content-Type": "text/html“  });
       response.end("Hello <strong>home page</strong>");
  }
   else if (request.url === "/foo" && request.method === "GET") {
       response.writeHead(200, { "Content-Type": "text/html“ });
       response.end("Hello <strong>foo</strong>");
  }
  else if (request.url === "/bar" && request.method === "GET") {
        response.writeHead(200, { "Content-Type": "text/html“ });
        response.end("Hello <strong>bar</strong>");
  }
  else {
       response.writeHead(404, { "Content-Type": "text/html“ });
       response.end("404 Not Found");
}
}).listen(8000);
```

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

function process_req(req, res) {
  if (req.method == 'GET' && req.url == '/') {
    fs.readFile('radius.html', function(err, data) {
      res.writeHead(200, {'Content-Type': 'text/html'});
      res.write(data);
      res.end();
   });
  }
  else if(req.method == 'GET' && req.url.substring(0,8) == '/process'){
     var q = url.parse(req.url, true);
     var qdata = q.query;
     var r = qdata.radius;

     var rad = Math.PI + r * r;
     res.writeHead(200, {'Content-Type': 'text/html'});
     res.write("The area is : " + rad);
     res.end();
   }
   else
     res.end("not found");
}
var server = http.createServer(process_req)
server.listen(3000);
console.log('server listening on localhost:3000');
```

{ radius: '100' }

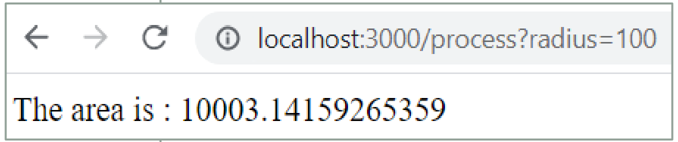
```
<html>
<body>
  <h1>Login</h1>
   <form action="process">
    Radius : <input name="radius">
   <input type="submit"
          value="Calc Radius">
   </form>
</body></html>
```



Login

Radius :

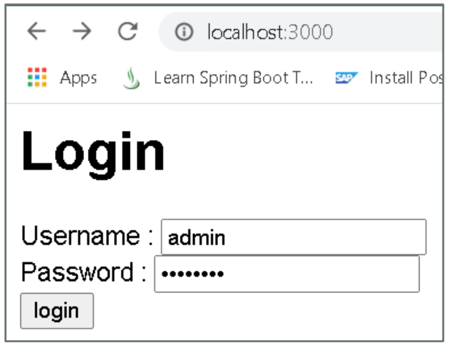
Calc Radius



The area is : 10003.14159265359

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

function process_req(req, res) {
  if (req.method == 'GET' && req.url == '/') {
     fs.readFile('loginPost.html', function(err, data) {
      res.writeHead(200, {'Content-Type': 'text/html'});
      res.write(data);
      res.end();
    });
  }
  else if(req.method == 'POST'){
     var body = "";
     req.on("data",function(data){
        body += data;
        res.writeHead(200, {'Content-Type': 'text/html'});
        var arr = body.split("&");
        res.write("Welcome " + arr);
        res.end();
      })
  }
}
var server = http.createServer(process_req)
server.listen(3000);
console.log('server listening on localhost:3000');
```

```
<body>   //loginpost.html
  <h1>Login</h1>
   <form action="processlogin"
         method="post">
    Username :
      <input name="uname"><br>
    Password :
      <input type="password"
             name="passwd"><br>
      <input type="submit"
             value="login">
   </form>
</body>
```

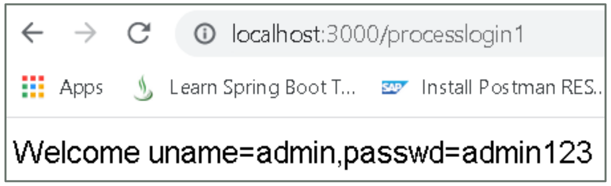


Login

Username : admin

Password : ••••••••

login



Welcome uname=admin,passwd=admin123

Promise

- Promises are a new feature of ES6.
  - It's a method to write asynchronous code; it represents the completion of an asynchronous function.
  - It is a Javascript object that might return a value in the future.
  - It accomplishes the same basic goal as a callback function, but with many additional features and a more readable syntax.
- Creating a Promise
  - Promises are created by using a constructor called Promise and passing it a function that receives two parameters, resolve and reject, which allow us to indicate to it that it was resolved or rejected.

```
let promise = new Promise(function(resolve, reject) {
  // executor code - things to do to accomplish your promise
});
```

```
let promise = new Promise(function(resolve, reject) {
  // things to do to accomplish your promise

  if(/* everything turned out fine */) {
      resolve('Stuff worked')
  } else { // for some reason the promise doesn't fulfilled
      reject(new Error('it broke'))
  }
})
```

Consuming a Promise

- The promise we created earlier has fulfilled with a value, now we want to be able to access the value.
- Promises have a method called then() that will run after a promise reaches resolve in the code.
- The then() method returns a Promise. It takes up to two arguments: callback functions for the success and failure cases of the Promise.
- Syntax : p.then(onFulfilled[, onRejected]);
  - onFulfilled function called if the Promise is fulfilled. This function has one argument, the fulfillment value.
  - Example

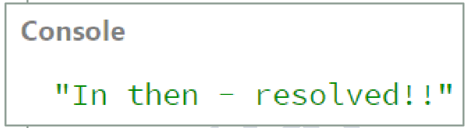
```
promise.then(
  function(result) { /* handle a successful result */ },
  function(error) { /* handle an error */ }
);
```

```
promise.then(function(result) {
  console.log("Promise worked");
}, function(err) {
  console.log("Something broke");
});
```

Consuming a Promise

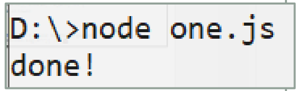
- Full example

```
const promise = new Promise((resolve, reject) => {
  if(true)
     resolve("resolved!!")
});

promise.then(msg => console.log("In then - "+ msg))
```



```
Console

  "In then - resolved!!"
```

```
let promise = new Promise(function(resolve, reject) {
    setTimeout(() => resolve("done!"), 3000);
  });

  // resolve runs the first function in .then
  promise.then(
    result =>console.log(result), //shows "done!" after 3 seconds
    error =>console.log(error) // doesn't run
  );
```



```
D:\>node one.js
done!
```

```
let promise = new Promise(function(resolve, reject) {
  setTimeout(() => reject(new Error("Whoops!")), 1000);
});

// reject runs the second function in .then
promise.then(
  result => alert(result), // doesn't run
  error => alert(error) // shows "Error: Whoops!" after 1 second
);
```

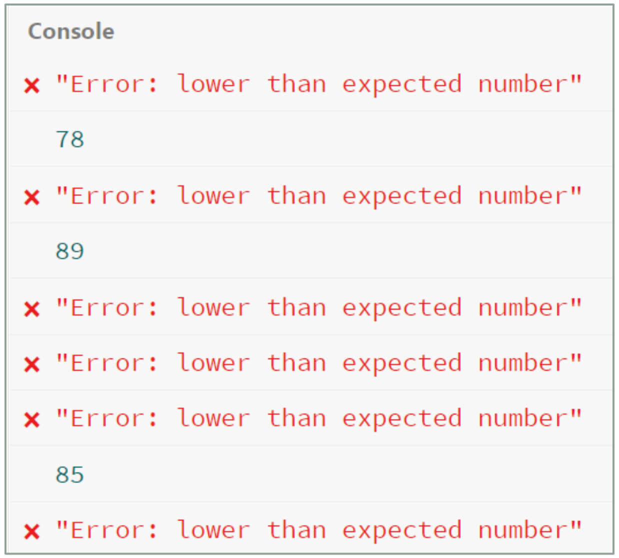
Consuming a Promise

- Alternatively, instead of using this syntax of then(fulfilled, rejected), you can also use catch()
- The **then** and **catch** handlers are asynchronous.
  - Basically, then and catch will be executed once Javascript finished reading the code

```
promise.then(function(result) {
  console.log(result)
}).catch(function(err) {
  console.log(err)
})

console.log('Hello world')
```

```
const promise = new Promise((resolve, reject) => {
  // Note: only 1 param allowed
  return reject('Hi')
})

// Parameter passed into reject would be the arguments passed into
 catch.
promise.catch(err => console.log(err)) // Hi
```

```
function randomDelayed(max = 10, expected = 5, delay =  1000) {
  return new Promise((resolve, reject) => {
    const number = Math.floor(Math.random() * max)

    setTimeout(
      () => number > expected
        ? resolve(number)
        : reject(new Error('lower than expected number')), 1000
    );
  });
}
randomDelayed(100, 75, 2500)
    .then(number => console.log(number))
    .catch(error => console.error(error.toString()));
```



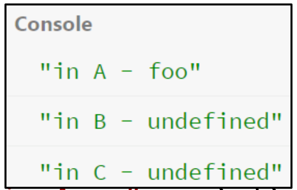
```
Console
✕ "Error: lower than expected number"
  78
✕ "Error: lower than expected number"
  89
✕ "Error: lower than expected number"
✕ "Error: lower than expected number"
✕ "Error: lower than expected number"
  85
✕ "Error: lower than expected number"
```

Chained Promises

- The methods promise.then(), promise.catch(), and promise.finally() can be used to associate further action with a promise that becomes settled.
  - Each .then() returns a newly generated promise object, which can optionally be used for chaining

```
const myPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('foo');
  }, 300);
});

myPromise
  .then(handleResolvedA, handleRejectedA)
  .then(handleResolvedB, handleRejectedB)
  .then(handleResolvedC, handleRejectedC);
```

```
const myPromise = new Promise((resolve) => {
  setTimeout(() => {
    resolve('foo');
  }, 300);
});

myPromise
  .then((x)=>{console.log("in A - " + x);})
  .then((x)=>{console.log("in B - " + x);})
  .then((x)=>{console.log("in C - " + x);});
```



```
"in A – foo"
"in B – undefined"
"in C – undefined"
```

```
import fs from 'fs';

function readAFile(path) {
  return new Promise((resolve, reject) => {
    fs.readFile(path, 'utf8', (error, data) => {
      if (error) return reject(error);
      return resolve(data);
    });
  });
}

readAFile('./file.txt')
  .then(data => console.log(data))
  .catch(error => console.error(error));
```

```
function myAsyncFunction(url) {
  return new Promise((resolve, reject) => {
    const xhr = new XMLHttpRequest()
    xhr.open("GET", url)
    xhr.onload = () => resolve(xhr.responseText)
    xhr.onerror = () => reject(xhr.statusText)
    xhr.send()
  });
}
```

Ajax example with promise

```
<!DOCTYPE html>
<html>
<body>
  <div id="msg"></div>
  <button id="btnGet">Get Message</button>
</body>
</html>
```

- You'll see Promises used a lot when fetching data from an API
- Promises can be confusing, both for new developers and experienced programmers that have never worked in an asynchronous environment before.
- However, it is much more common to consume promises than create them. Usually, a browser's Web API or third party library will be providing the promise, and you only need to consume it

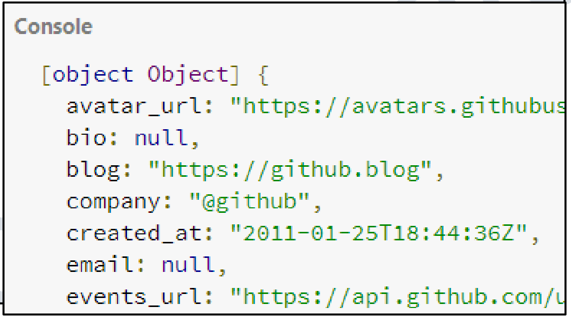
```
function load(url) {
   return new Promise(function (resolve, reject) {
     const request = new XMLHttpRequest();
     request.onreadystatechange = function (e) {
        if (this.readyState === 4) {
           if (this.status == 200) {
              resolve(this.response);
            } else {
              reject(this.status);
            }
        }
      }
      request.open('GET', url, true);
      request.send();
    });
  }
```

```
btn.onclick = function () {
  load('data.json')
    .then(
      response => {
        const result = JSON.parse(response);
         $(“#msg”).text(result.message);
      },
      error =>  $(“#msg”).text(`Error getting message,
 HTTP status: ${error}`
  );
}
```

Using Fetch

- One of the most useful and frequently used Web APIs that returns a promise is the Fetch API.
  - It allows you to make an asynchronous resource request over a network.
  - fetch() is a two-part process, and therefore requires chaining then()

```
// Fetch a user from the GitHub API
fetch('https://api.github.com/users/octocat')
  .then((response) => {
    return response.json()
  })
  .then((data) => {
    console.log(data)
  })
  .catch((error) => {
    console.error(error)
  })
```



Console

```
[object Object] {
  avatar_url: "https://avatars.githubus
  bio: null,
  blog: "https://github.blog",
  company: "@github",
  created_at: "2011-01-25T18:44:36Z",
  email: null,
  events_url: "https://api.github.com/u
```

async/await

- async/await added in ECMAScript 2017 (ES8)
- async/await is built on promises
- **async** keyword put in front of a function declaration turns it into an async function
- An async function is a function that knows how to expect the possibility of the **await** keyword being used to invoke asynchronous code.
  - We use the async keyword with a function to represent that the function is an asynchronous function.
  - The async function returns a promise.

```
//simple synchromous JS function and invocation
function hello() { return "Hello" };
hello();
```

```
//converting above JS function into async function
async function hello() { return "Hello" };
hello();
```

```
// You can also create an async function expression like this:
let hello = async function() { return "Hello" };
hello();
```

```
//you can also use arrow functions:
let hello = async () => { return "Hello" };
```

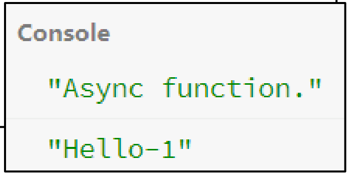
async/await

- To consume the value returned when the promise fulfills (since it is returning a promise) use a .then() block

```
hello().then((value) => console.log(value))
//or even just shorthand such as
hello().then(console.log)
```

```
//complete example
async function hello(){
  return "hello-1"
}
hello().then((x)=>console.log(x))
// returns"hello-1"
```

```
let f = async () => {
  console.log('Async function.');
  return Promise.resolve("Hello-1");  //function returns a promise
}

f().then(function(result) {
  console.log(result)
});
```

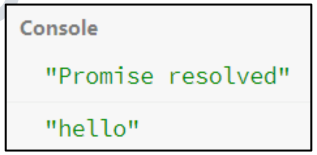


```
Console
  "Async function."
  "Hello-1"
```

The await keyword

- The **await** keyword is used inside the async function to wait for the asynchronous operation
  - await can be put in front of any async promise-based function to pause your code on that line until the promise fulfills, then return the resulting value.
  - Syntax: `let result = await promise;`
  - You can use await when calling any function that returns a Promise, including web API functions

```
// a promise
let promise = new Promise(function (resolve, reject) {
  setTimeout(function () {
     resolve('Promise resolved')}, 4000);
});


async function asyncFunc() {// async function

  // wait until the promise resolves
  let result = await promise;

  console.log(result);
  console.log('hello');
}

asyncFunc(); // calling the async function
```



```
Console
  "Promise resolved"
  "hello"
```

```
function logFetch(url) {
  return fetch(url)
    .then(response => response.text())
    .then(text => {
      console.log(text);
    }).catch(err => {
      console.error('fetch failed', err);
    });
}
```

```
async function logFetch(url) {
  try {
    const response = await fetch(url);
    console.log(await response.text());
  }
  catch (err) {
    console.log('fetch failed', err);
  }
}
```

Next gen Javascript

- **const** : from JS 1.5 onwards.- to define constants
  - Eg :

```
const myBirthday = '18.04.1982';
myBirthday = '01.01.2001';          // error, can't reassign the constant!
```

```
const LANGUAGES = ['Js', 'Ruby', 'Python', 'Go'];
LANGUAGES = "Javascript"; // shows error.
LANGUAGES.push('Java'); // Works fine.
console.log(LANGUAGES); // ['Js', 'Ruby', 'Python', 'Go', 'Java']
```

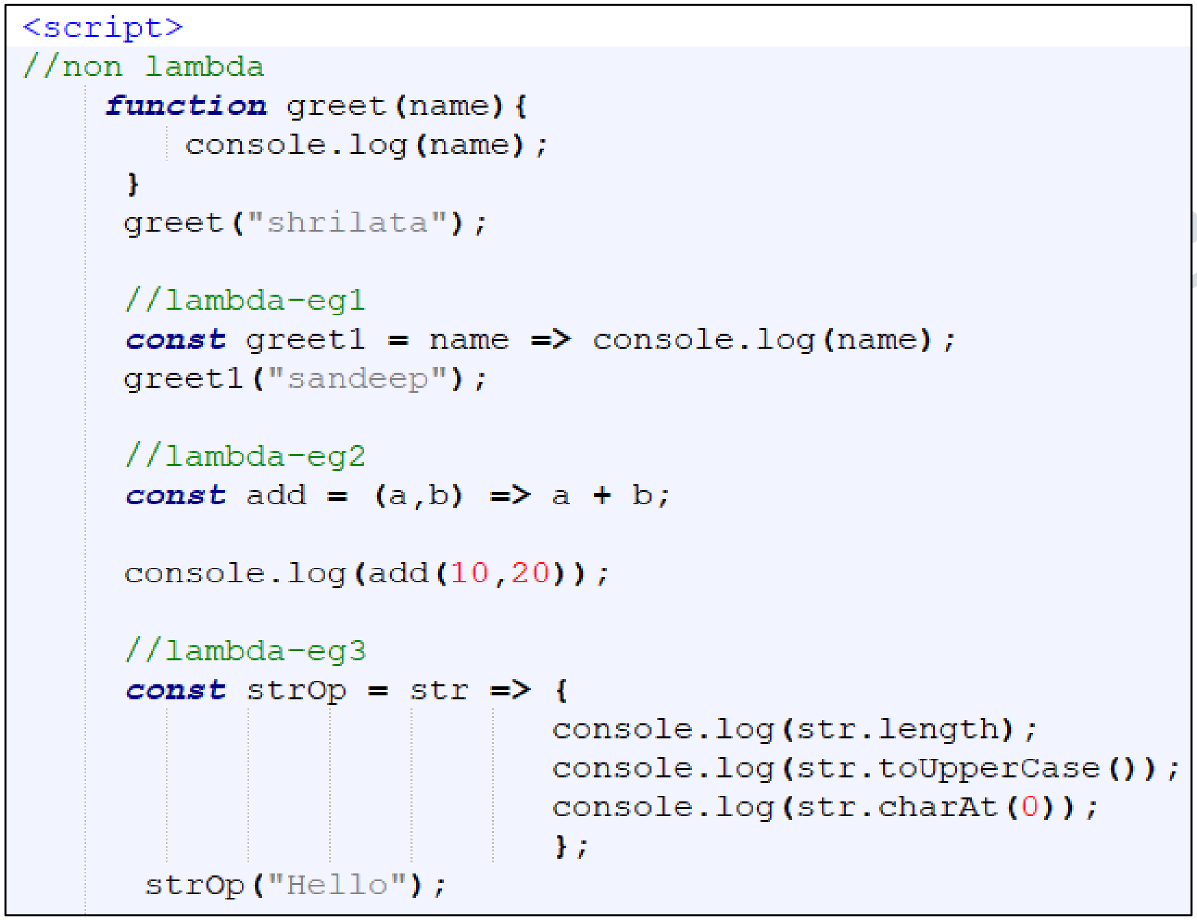
- **let** : to define block-scoped variables; can be used in four ways:
  - as a variable declaration like var; in a for or for/in loop, as a substitute for var;
  - as a block statement, to define new variables and explicitly delimit their scope
  - to define variables that are scoped to a single expression.
  - Eg : let message = 'Hello!';
  - Eg : let user = 'John', age = 25, message = 'Hello';

```
if (true) {
 let a = 40;
 console.log(a); //40
}
console.log(a); // undefined
```

```
let a = 50;  let b = 100;
if (true) {
   let a = 60;
   var c = 10;
   console.log(a/c); // 6
   console.log(b/c); // 10
}
console.log(c); // 10
console.log(a); // 50
```

Next gen Javascript

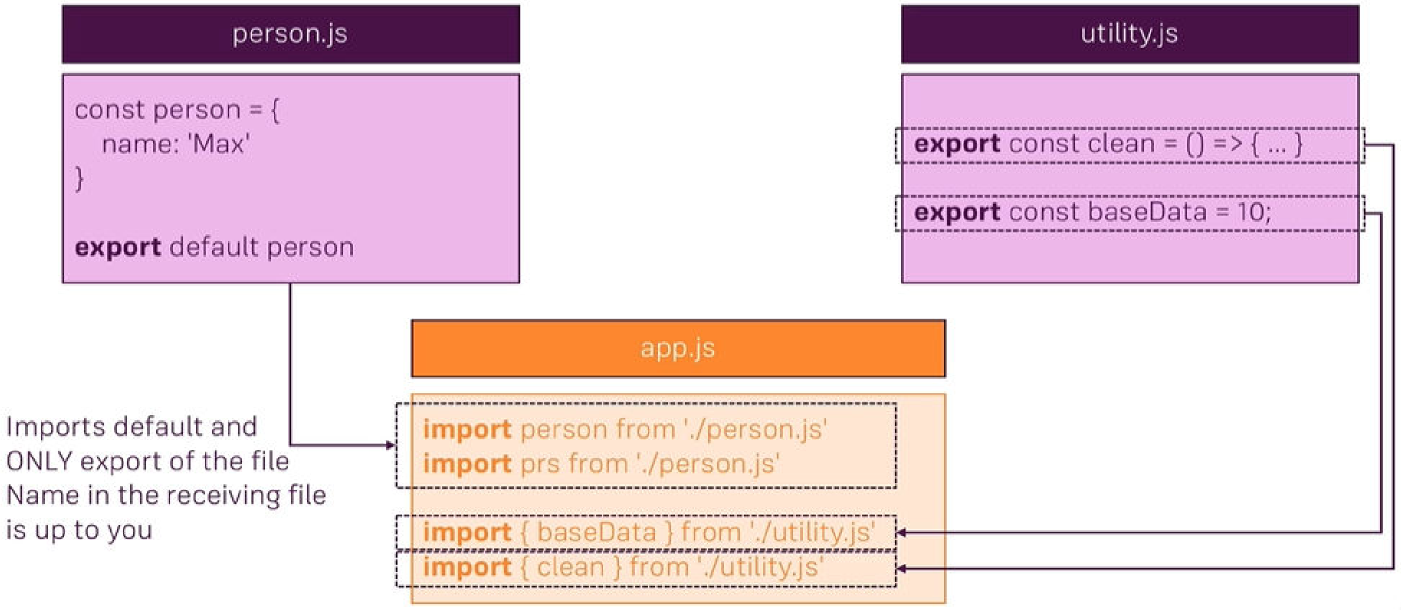
- Arrow functions : same as lambda in TS



```
<script>
//non lambda
    function greet(name){
        console.log(name);
     }
     greet("shrilata");

     //lambda-eg1
     const greet1 = name => console.log(name);
     greet1("sandeep");

     //lambda-eg2
     const add = (a,b) => a + b;

     console.log(add(10,20));

     //lambda-eg3
     const strOp = str => {
                     console.log(str.length);
                     console.log(str.toUpperCase());
                     console.log(str.charAt(0));
                     };
      strOp("Hello");
```

Next gen Javascript

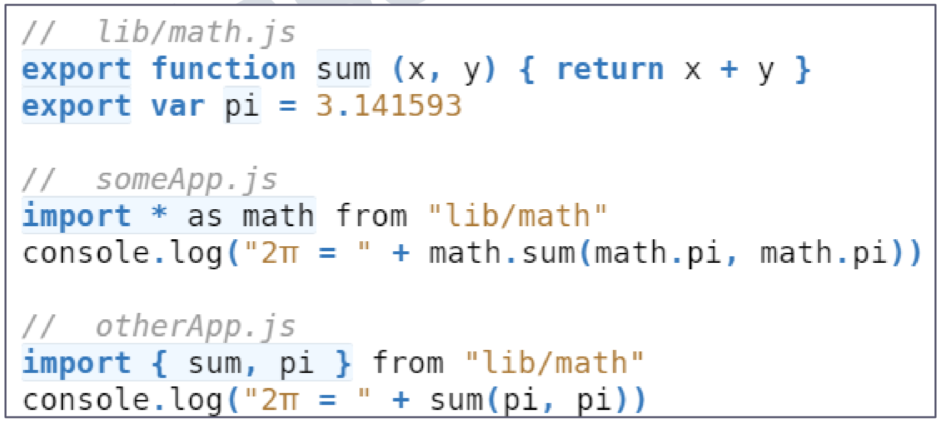
- Exports and imports


person.js
utility.js
app.js
Imports default and ONLY export of the file
Name in the receiving file is up to you

```
// person.js
const person = {
   name: 'Max'
}

export default person
```

```
// utility.js
export const clean = () => { … }

export const baseData = 10;
```

```
// app.js
import person from './person.js'
import prs from './person.js'

import { baseData } from './utility.js'
import { clean } from './utility.js'
```



```
//  lib/math.js
export function sum (x, y) { return x + y }
export var pi = 3.141593

//  someApp.js
import * as math from "lib/math"
console.log("2π = " + math.sum(math.pi, math.pi))

//  otherApp.js
import { sum, pi } from "lib/math"
console.log("2π = " + sum(pi, pi))
```

Next gen Javascript : Classes, properties and methods

```
class Person{
        fname = "Anil";
        lname = "Patil";
        getFullName = () => this.fname + " " + this.lname;
        }

        const p1 = new Person();
        console.log(p1.getFullName()); //Anil Patil


class Stud{
    constructor(name){
        this.sname=name;
    }
    getName(){
        console.log(this.sname); //sunita
     }
}
const s1 = new Stud("sunita");
s1.getName();
```

constructor method is always defined with the name "constructor"

Classes can have methods, which defined as functions, albeit without needing to use the function keyword.

```
class GradStud extends Stud{
    constructor(){
        super("Kavita");
        this.gpa = 4.9;
    }
    getDetails(){
        super.getName();    //Kavita
        console.log("GPA : " + this.gpa);
    }
}

const gs = new GradStud();
gs.getDetails();
```

Next gen Javascript : Classes, properties and methods

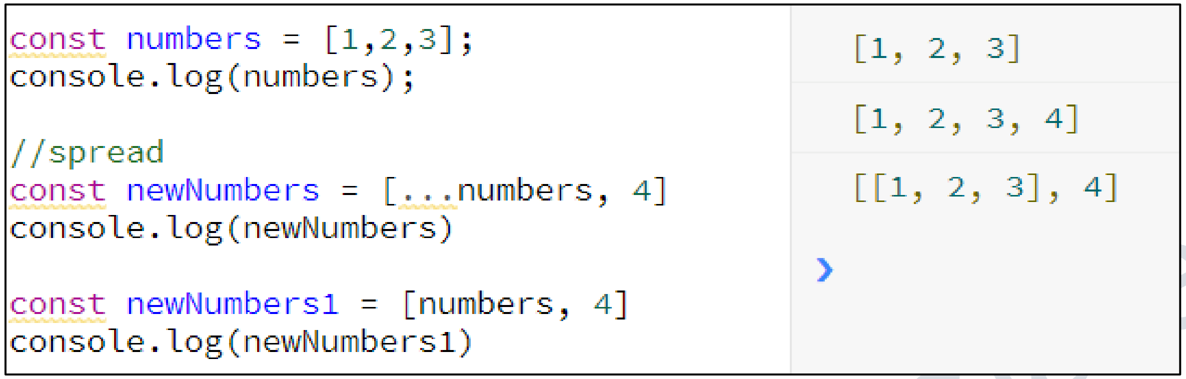
- To declare a class, you use the class keyword with the name of the class
- There can only be one "constructor" in a class; SyntaxError will be thrown if the class contains more than one occurrence of a constructor method.

```
class Rectangle {
    constructor(height, width) {
      this.height = height;
      this.width = width;
    }
  }
```

```
class Rectangle {
    constructor(height, width) {
      this.height = height;
      this.width = width;
    }

    // Method
    calcArea() {
      return this.height * this.width;
    }
  }

  const square = new Rectangle(10, 10);

  console.log(square.calcArea()); // 100
```

Next gen Javascript : Spread and rest operators

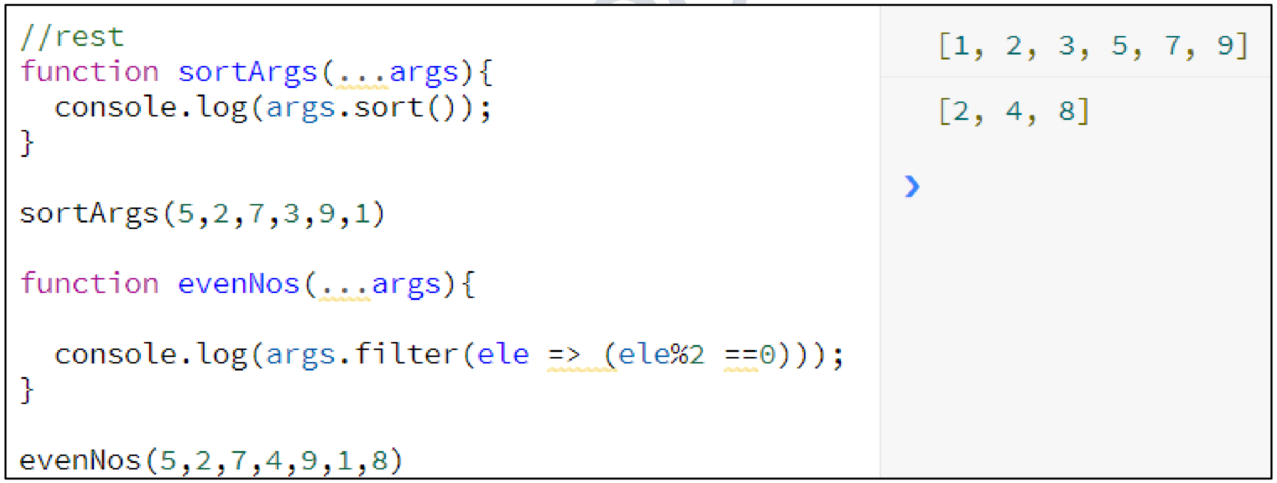
- spread:



```
const numbers = [1,2,3];
console.log(numbers);

//spread
const newNumbers = [...numbers, 4]
console.log(newNumbers)

const newNumbers1 = [numbers, 4]
console.log(newNumbers1)
```

```
[1, 2, 3]
[1, 2, 3, 4]
[[1, 2, 3], 4]
>
```

- rest operator : used to merge a list of function arguments into an array



```
//rest
function sortArgs(...args){
  console.log(args.sort());
}

sortArgs(5,2,7,3,9,1)

function evenNos(...args){

  console.log(args.filter(ele => (ele%2 ==0)));
}

evenNos(5,2,7,4,9,1,8)
```

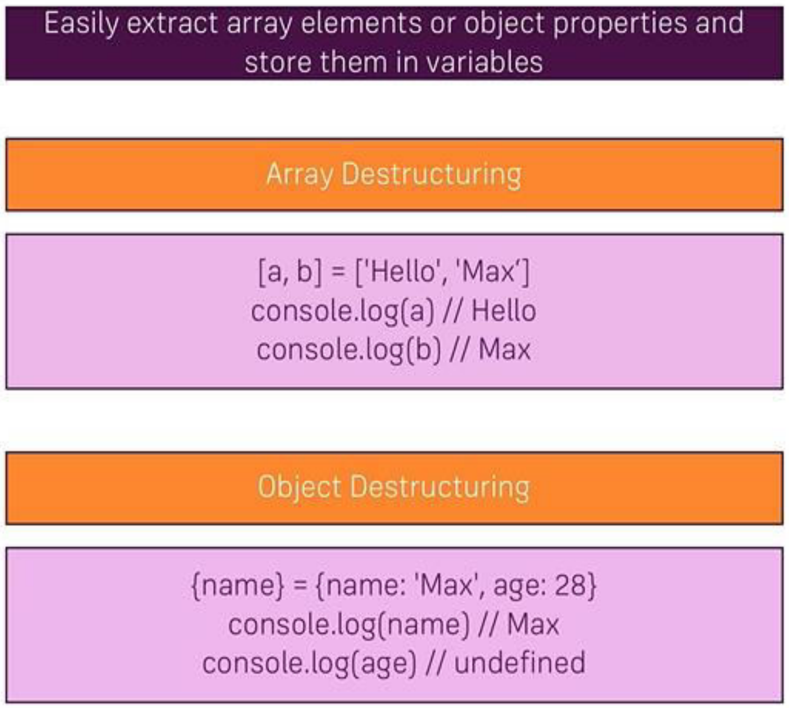
```
[1, 2, 3, 5, 7, 9]
[2, 4, 8]
>
```

Next gen Javascript

- Destructuring : allows to easily extract array elements or object properties and store them in variables
  - Destructuring is useful because it allows you to do in a single line, what would otherwise require multiple lines

```
var rect = { x: 0, y: 10, width: 15, height: 20 };

// Destructuring assignment
var {x, y, width, height} = rect;
console.log(x, y, width, height); // 0,10,15,20

rect.x = 10;

// assign to existing variables using outer parentheses
({x, y, width, height} = rect);
console.log(x, y, width, height); // 10,10,15,20
```

```
const arr=[1,2,3,4,5]
var [a,b] = arr
console.log(a,b)  //1,2
```



Easily extract array elements or object properties and store them in variables

Array Destructuring

```
[a, b] = ['Hello', 'Max']
console.log(a) // Hello
console.log(b) // Max
```

Object Destructuring

```
{name} = {name: 'Max', age: 28}
console.log(name) // Max
console.log(age) // undefined
```

Array functions

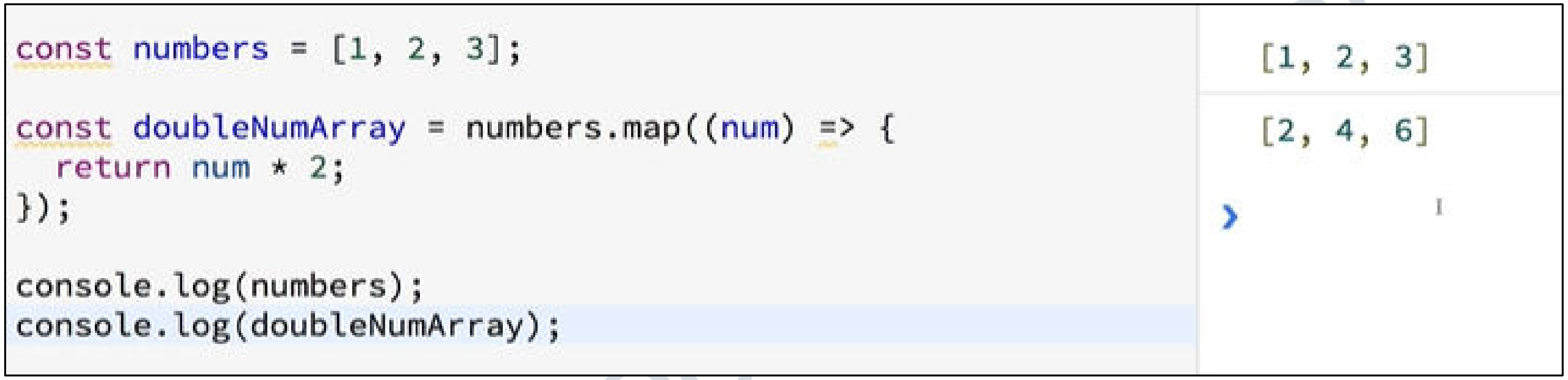
- Array.filter
  - You can filter arrays by using the .filter(callback) method.
  - The result will be another array that contains 0 or more elements based on the condition (or the "check") that you have in the callback.

```
var nums = [1, 2, 3, 21, 22, 30];
var evens = nums.filter(i => i % 2 == 0);
```

```
const grades = [10, 2, 21, 35, 50, -10, 0, 1];

//get all grades > 20
const result = grades.filter(grade => grade > 20); // [21, 35, 50];

// get all grades > 30
grades.filter(grade => grade > 30); // ([35, 50])
```

Array functions

- Array.map() : returns a new array containing the result of invoking the callback function for every item in the array.



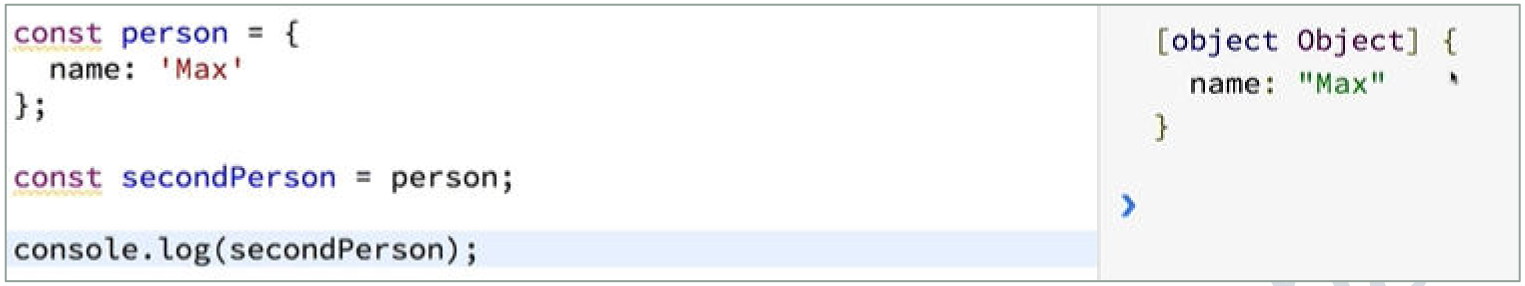
```
const numbers = [1, 2, 3];

const doubleNumArray = numbers.map((num) => {
  return num * 2;
});

console.log(numbers);
console.log(doubleNumArray);
```

```
[1, 2, 3]
[2, 4, 6]
```

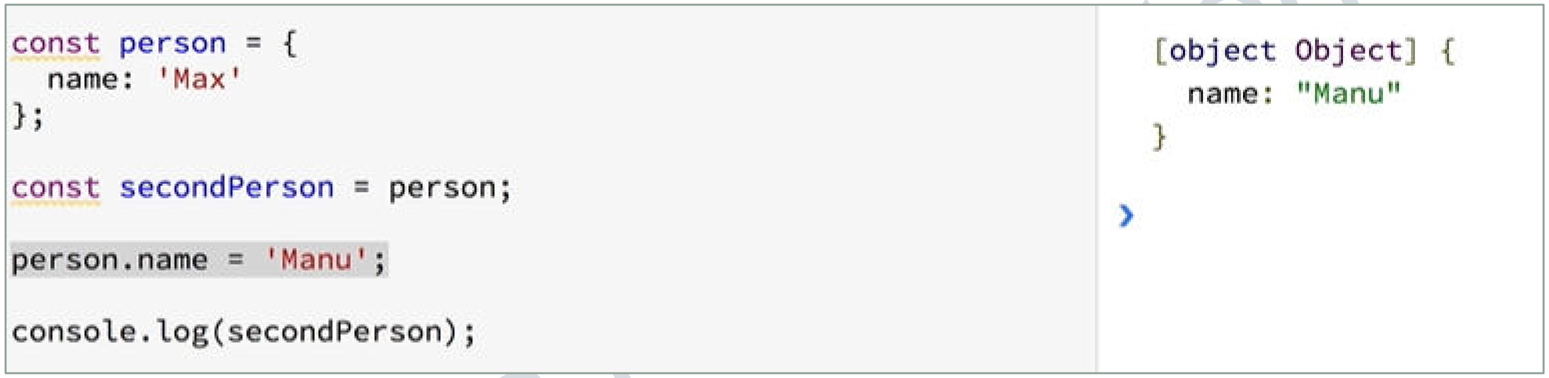
```
var numbers = [1, 4, 9];
var roots = numbers.map(Math.sqrt);
console.log("roots is : " + roots );  //1,2,3
```

```
var numbers = [1, 2, 3, 4];
var doubled = numbers.map(i => i * 2);
var doubled = [for (i of numbers) i * 2];  //same as above
console.log(doubled); // logs 2,4,6,8
```

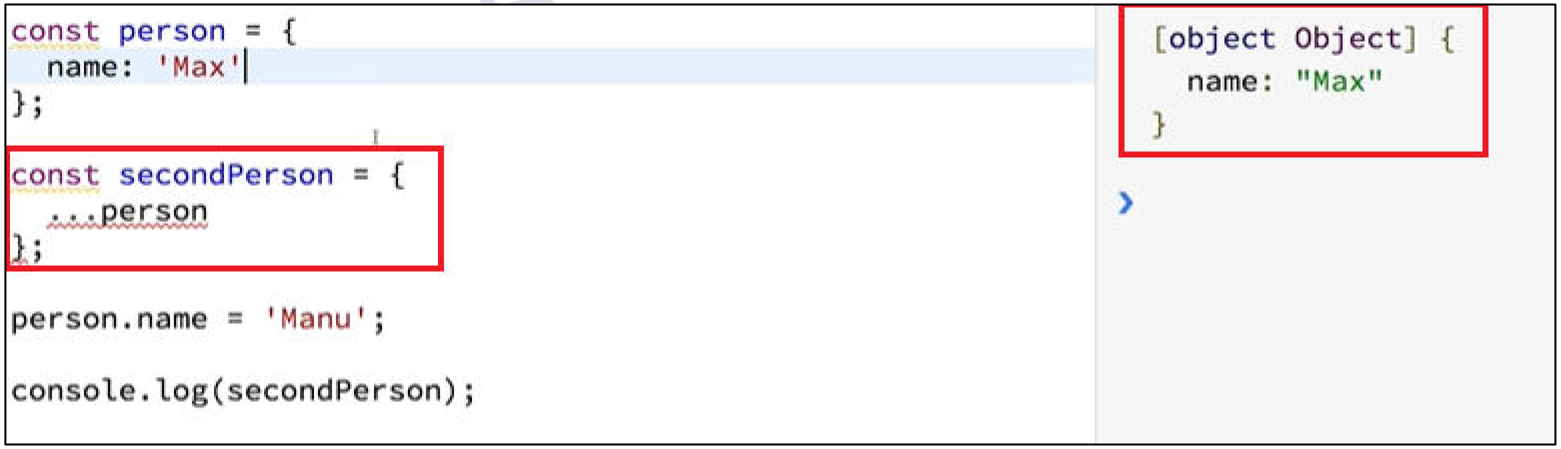
Reference and primitive types


```
const person = {
  name: 'Max'
};

const secondPerson = person;

console.log(secondPerson);
```

```
[object Object] {
  name: "Max"
}
```


```
const person = {
  name: 'Max'
};

const secondPerson = person;

person.name = 'Manu';

console.log(secondPerson);
```

```
[object Object] {
  name: "Manu"
}
```


```
const person = {
  name: 'Max'
};

const secondPerson = {
  ...person
};

person.name = 'Manu';

console.log(secondPerson);
```

```
[object Object] {
  name: "Max"
}
```

Next gen Javascript

- Template strings provide us with an alternative to string concatenation. They also allow us to insert variables into a string.
  - Traditional string concatenation uses plus signs or commas to compose a string using variable values and strings.
  - console.log(lastName + ", " + firstName + “ “ + middleName)

- With a template, we can create one string and insert the variable values by surrounding them with ${variable}.
  - console.log(`${lastName}, ${firstName} ${middleName}`)
  - Any JavaScript that returns a value can be added to a template string between the ${ } in a template string.

```
//Javascript : generating an html string
var msg1 = 'Have a great day';
var html = '<div>' + msg1 + '</div>';
document.write(html)

//Using template strings
var msg2 = 'Never give up';
var html1 = `<div>${msg2}</div>`;
document.write(html1)
```

BOOTSTRAP 4

Pre-reqs:

- HTML
- CSS
- JavaScript

Introduction

- Bootstrap is an opensource frontend framework developed by Twitter.
  - It is the most popular ***HTML, CSS, and JavaScript*** framework for developing responsive, mobile first web sites.
  - Bootstrap is a free and open source collection of tools for creating websites and web applications.
  - Bootstrap contains a set of CSS- and HTML-based templates for styling forms, elements, buttons, navigation, typography, and a range of other UI components.
  - It also comes with optional JavaScript plugins to add interactivity to components.

- Bootstrap is promoted as being **One framework, every device**.
  - This is because websites built with Bootstrap will automatically scale between devices — whether the device is a mobile phone, tablet, laptop, desktop computer, screen reader, etc.

- Responsive web design is about creating web sites which automatically adjust themselves to look good on all devices, from small phones to large desktops.
  - Developers can then create a single design that works on any kind of device: mobiles, tablets, smart TVs, and PCs

Where to Get Bootstrap 4?

- There are two ways to start using Bootstrap 4 on your own web site.
  - Download Bootstrap 4 from getbootstrap.com : https://getbootstrap.com/docs/4.5/getting-started/download/
  - If you don't want to download and host Bootstrap 4 yourself, you can include it from a CDN (Content Delivery Network).
  - MaxCDN provides CDN support for Bootstrap's CSS and JavaScript. You must also include jQuery
  - The https://getbootstrap.com/docs/4.5/getting-started/introduction/ page gives CDN links for CSS and js files

```html
<!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

<!-- jQuery library -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

<!-- Popper JS -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>

<!-- Latest compiled JavaScript -->
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
```

Create First Web Page With Bootstrap 4

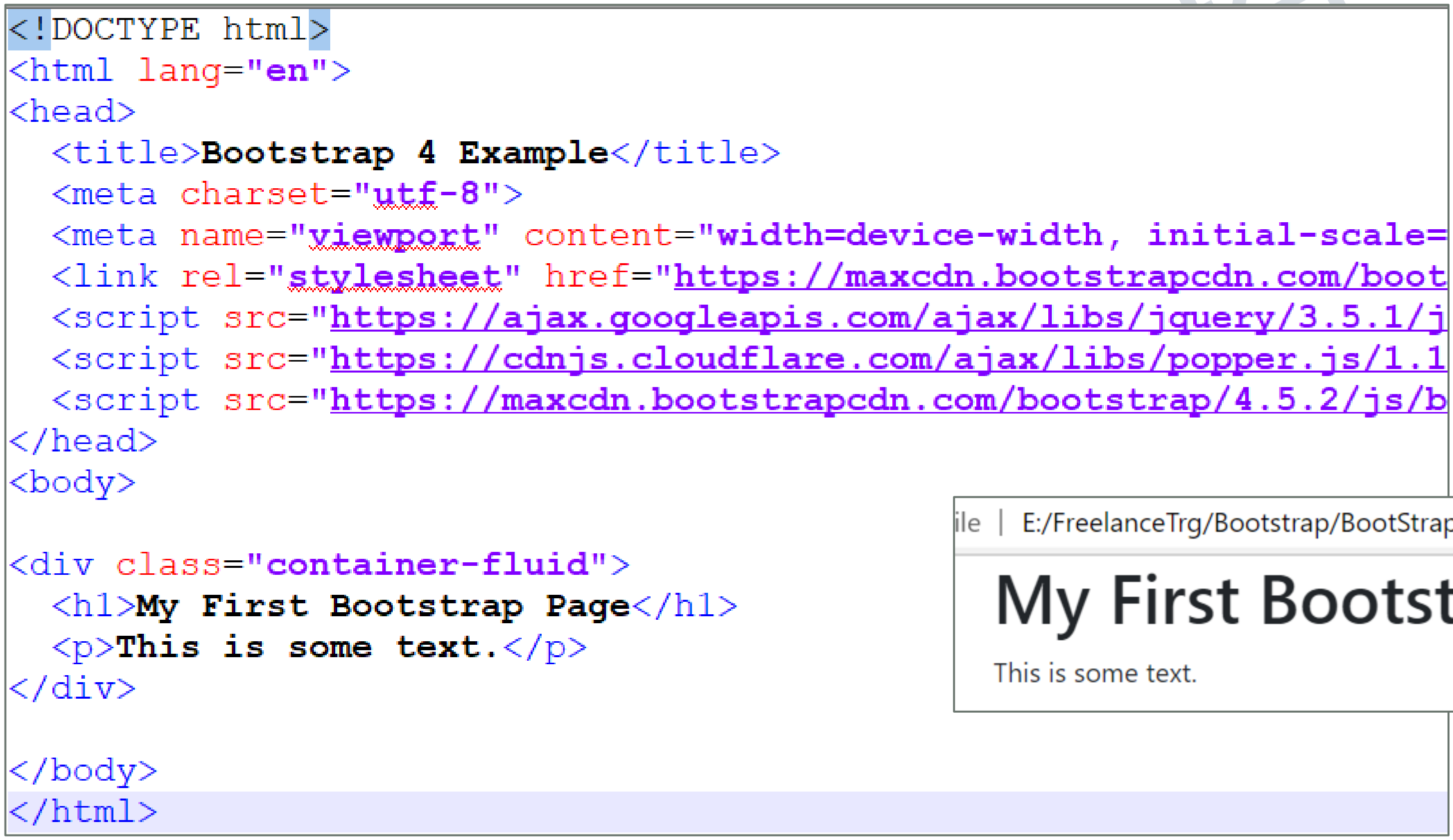
```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap 4 Example</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
        href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js">
  </script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js">
  </script>
</head>
<body>
   //body comes here
</body>
</html>
```

To ensure proper rendering and touch zooming, add this <meta> tag

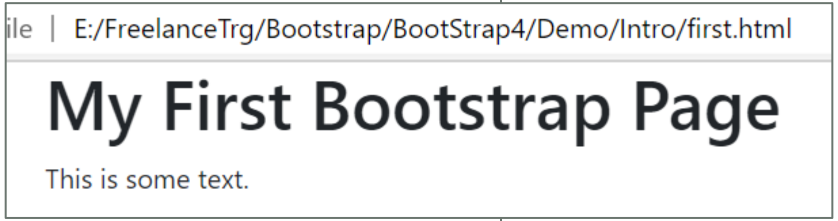
- The width=device-width part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).
- The initial-scale=1 part sets the initial zoom level when the page is first loaded by the browser.

Create First Web Page With Bootstrap 4

- Bootstrap 4 also requires a containing element to wrap site contents.
- There are two container classes to choose from:
  - The .container class provides a responsive fixed width container
  - The .container-fluid class provides a full width container, spanning the entire width of the viewport



```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap 4 Example</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/boot
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/j
  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.1
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/b
</head>
<body>

<div class="container-fluid">
  <h1>My First Bootstrap Page</h1>
  <p>This is some text.</p>
</div>

</body>
</html>
```



ile | E:/FreelanceTrg/Bootstrap/BootStrap4/Demo/Intro/first.html

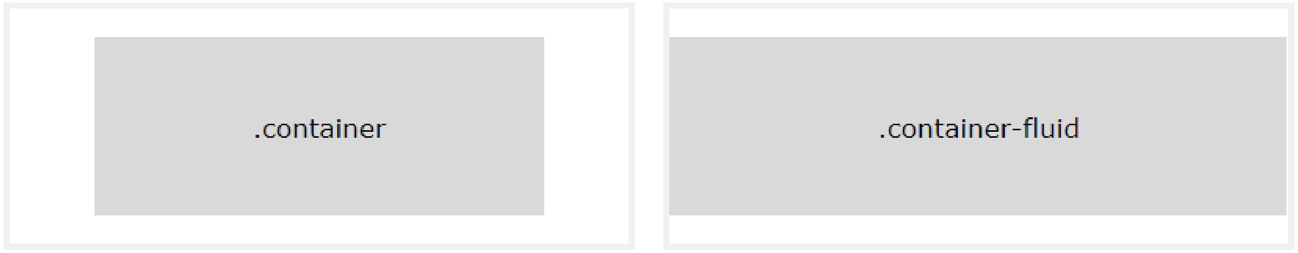
My First Bootstrap Page

This is some text.

Bootstrap Container

- Bootstrap container is basically used in order to create a centered area that lies within the page and generally deals with the margin of the content and the behaviors that are responsible for the layout.
  - It contains the grid system (row elements, which in turn are the container of columns).
- There are two container classes in Bootstrap:
  - .container: provides a fixed width container with responsiveness. It will not take the complete width of its viewport.
  - .container-fluid: provides a full width container of the viewport and its width will change (expand or shrink) on different screen sizes.

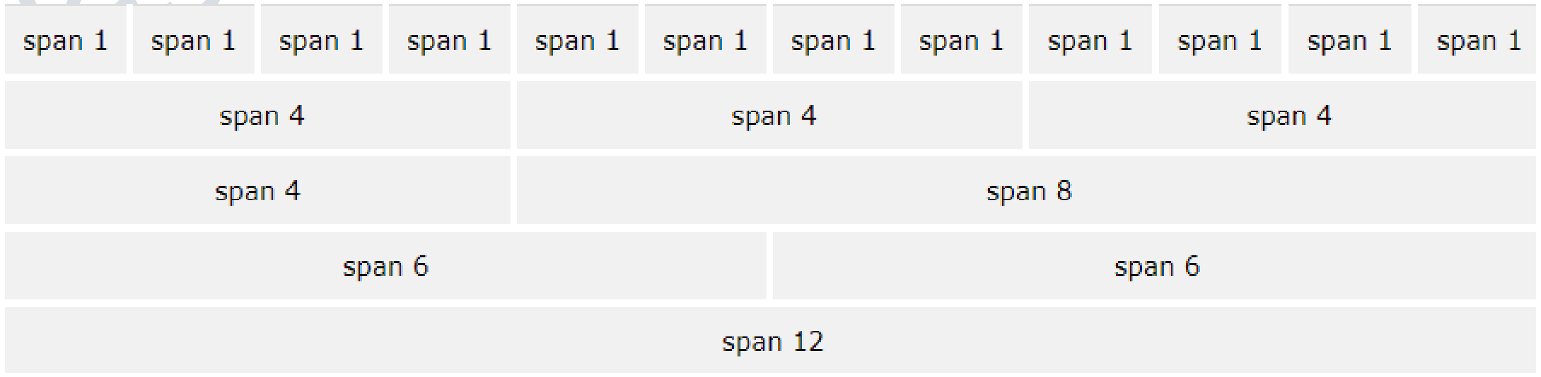
```
<body>
    <div class="container">
        <h1>Container</h1>
    </div>
</body>
```



Bootstrap Grid System

- Bootstrap grid system divides the screen into columns—up to 12 in each row. (rows are infinite)
  - The column widths vary according to the size of screen they're displayed in.
  - Bootstrap's grid system is responsive, as the columns resize themselves dynamically when the size of browser window changes.
  - If you do not want to use all 12 columns individually, you can group the columns together to create wider columns
  - it is a good practice to wrap all the contents within a container; create a row (with class row) inside a container, then start creating the columns.

```
<div class="container">
    <div class="row">
      //add desired number of cols here
    </div>
</div>
```



span 1 | span 1 | span 1 | span 1 | span 1 | span 1 | span 1 | span 1 | span 1 | span 1 | span 1 | span 1

span 4 | span 4 | span 4

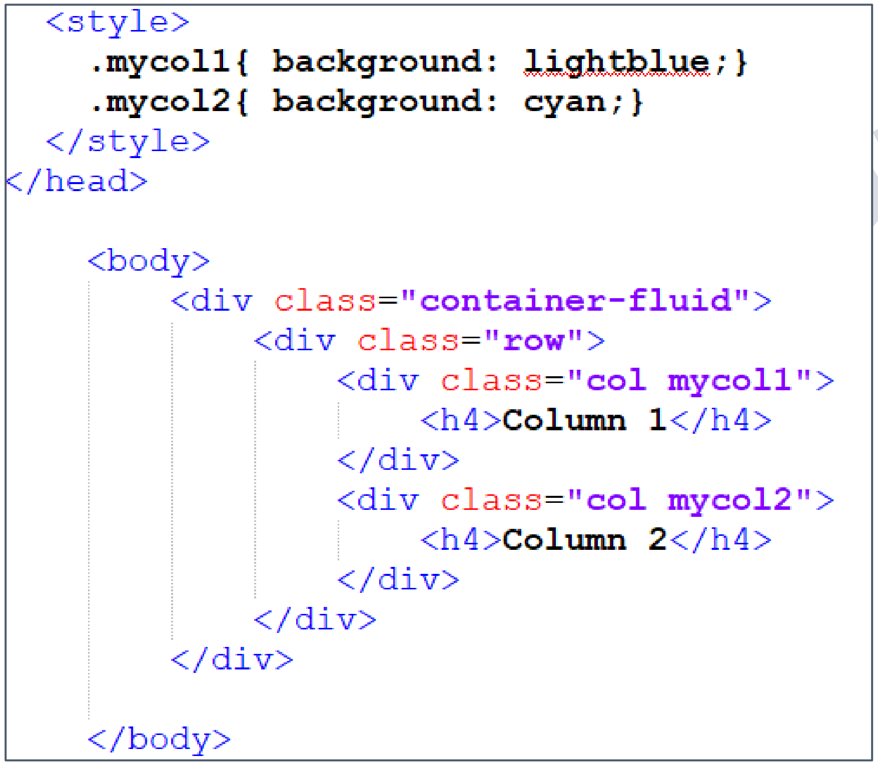
span 4 | span 8

span 6 | span 6

span 12

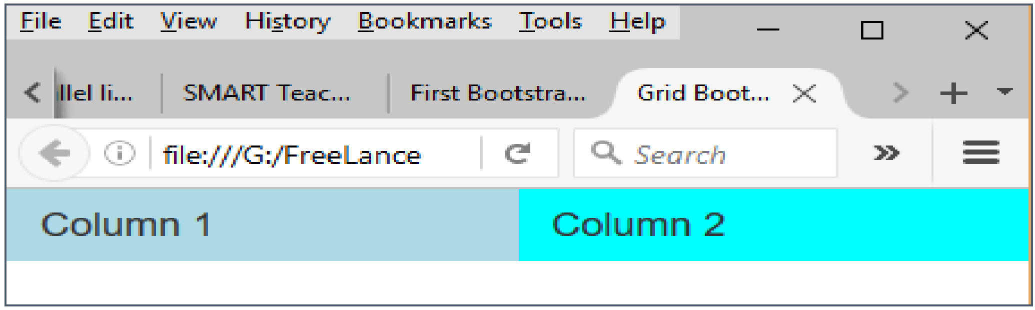
Grid Classes

- The Bootstrap 4 grid system has five classes:
  - .col- (extra small devices - screen width less than 576px)
  - .col-sm- (small devices - screen width equal to or greater than 576px)
  - .col-md- (medium devices - screen width equal to or greater than 768px)
  - .col-lg- (large devices - screen width equal to or greater than 992px)
  - .col-xl- (xlarge devices - screen width equal to or greater than 1200px)



```
  <style>
    .mycol1{ background: lightblue;}
    .mycol2{ background: cyan;}
  </style>
</head>

    <body>
        <div class="container-fluid">
            <div class="row">
                <div class="col mycol1">
                    <h4>Column 1</h4>
                </div>
                <div class="col mycol2">
                    <h4>Column 2</h4>
                </div>
            </div>
        </div>

    </body>
```

Example

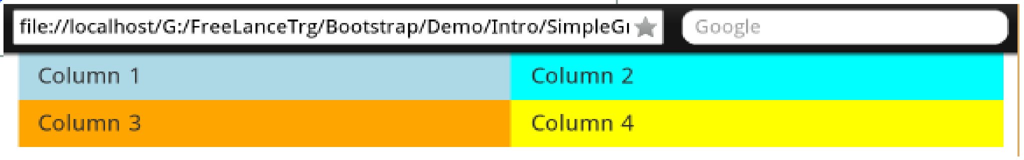

File Edit View History Bookmarks Tools Help
llel li... SMART Teac... First Bootstra... Grid Boot...
file:///G:/FreeLance
Search

Column 1 Column 2

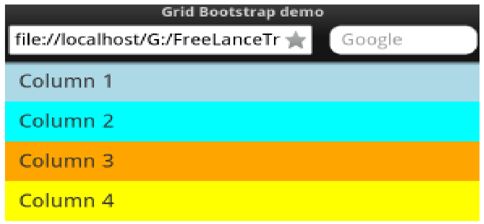
Building a Basic Grid

```
  <style>
        .col1{ background: lightblue;}
        .col2{ background: cyan;}
        .col3{ background: orange;}
        .col4{ background: yellow;}
        </style>
</head>
<body>
  <div class="container">
    <div class="row">
      <div class="col-12 col-sm-6 col1">
        <h4>Column 1</h4>
      </div>
      <div class="col-12 col-sm-6 col2">
        <h4>Column 2</h4>
      </div>
      <div class="col-12 col-sm-6 col3">
        <h4>Column 3</h4>
      </div>
      <div class="col-12 col-sm-6 col4">
        <h4>Column 4</h4>
      </div>
    </div>
  </div>
```

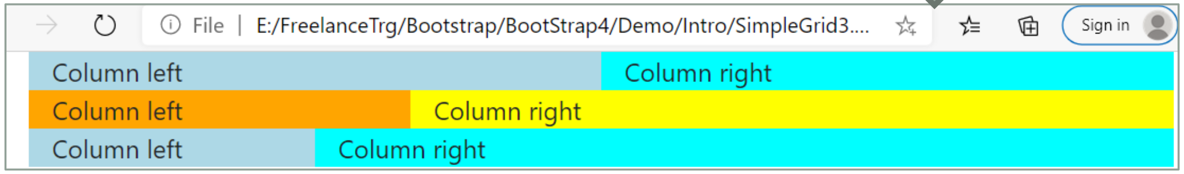
```
  <style>
        .col1{ background: lightblue;}
        .col2{ background: cyan;}
        .col3{ background: orange;}
        .col4{ background: yellow;}
        </style>
    </head>
<body>
<div class="container">
    <!--Row with two equal columns-->
    <div class="row">
        <div class="col-md-6 col1">Column left</div>
        <div class="col-md-6 col2">Column right</div>
    </div>

    <!--Row with two columns divided in 1:2 ratio-->
    <div class="row">
        <div class="col-md-4 col3">Column left</div>
        <div class="col-md-8 col4">Column right</div>
    </div>

    <!--Row with two columns divided in 1:3 ratio-->
    <div class="row">
        <div class="col-md-3 col1">Column left</div>
        <div class="col-md-9 col2">Column right</div>
    </div>
</div>
```


file://localhost/G:/FreeLanceTrg/Bootstrap/Demo/Intro/SimpleGı
Google

Column 1 | Column 2
Column 3 | Column 4


Grid Bootstrap demo
file://localhost/G:/FreeLanceTr
Google

Column 1
Column 2
Column 3
Column 4

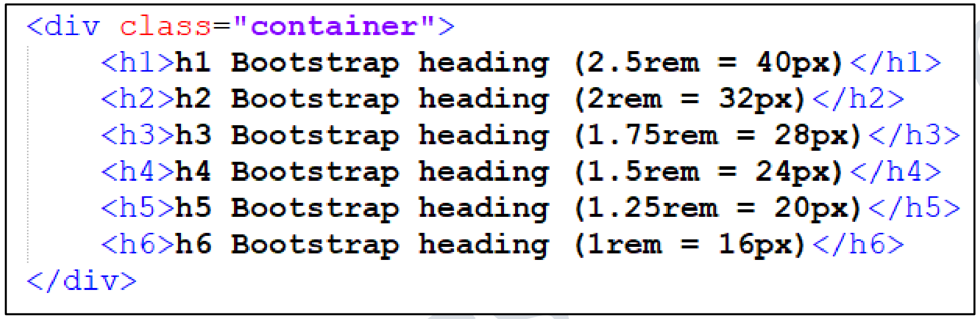

File | E:/FreelanceTrg/Bootstrap/BootStrap4/Demo/Intro/SimpleGrid3....
Sign in

Column left | Column right
Column left | Column right
Column left | Column right

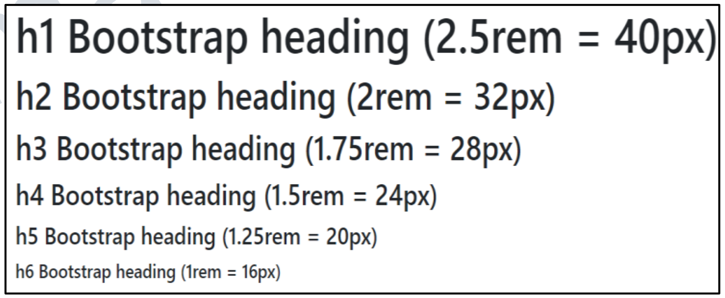
MISC COMPONENTS

<h1> - <h6>

- Typography refers to the various styles present in Bootstrap style sheets which define how various text elements will appear on the web page.
  - HTML uses default font and style to create headings, paragraphs, lists and other inline elements.
  - Bootstrap overrides default and provides consistent styling across browsers for common typographic elements.
- Bootstrap 4 styles HTML headings (<h1> to <h6>) with a bolder font-weight and an increased font-size

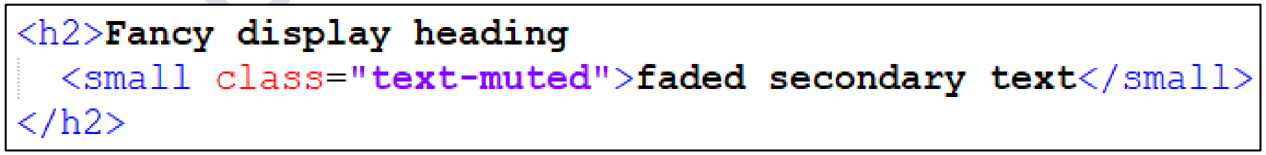


```
<div class="container">
    <h1>h1 Bootstrap heading (2.5rem = 40px)</h1>
    <h2>h2 Bootstrap heading (2rem = 32px)</h2>
    <h3>h3 Bootstrap heading (1.75rem = 28px)</h3>
    <h4>h4 Bootstrap heading (1.5rem = 24px)</h4>
    <h5>h5 Bootstrap heading (1.25rem = 20px)</h5>
    <h6>h6 Bootstrap heading (1rem = 16px)</h6>
</div>
```

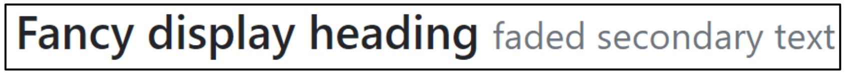


h1 Bootstrap heading (2.5rem = 40px)
h2 Bootstrap heading (2rem = 32px)
h3 Bootstrap heading (1.75rem = 28px)
h4 Bootstrap heading (1.5rem = 24px)
h5 Bootstrap heading (1.25rem = 20px)
h6 Bootstrap heading (1rem = 16px)

- Additionally, you can use the <small> tag with .text-muted class to display the secondary text of any heading in a smaller and lighter variation.



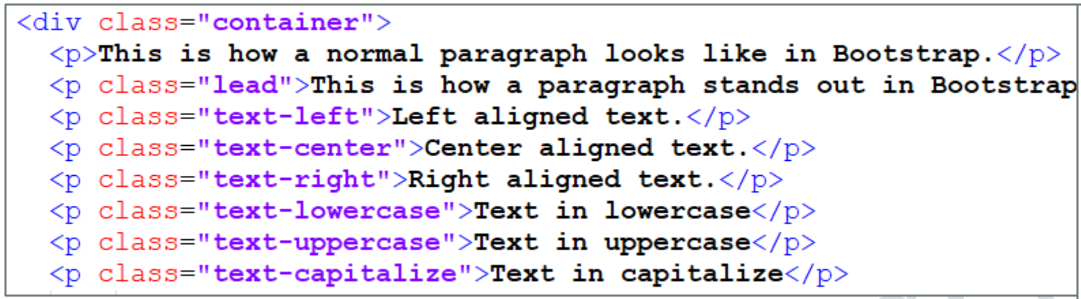
```
<h2>Fancy display heading
  <small class="text-muted">faded secondary text</small>
</h2>
```



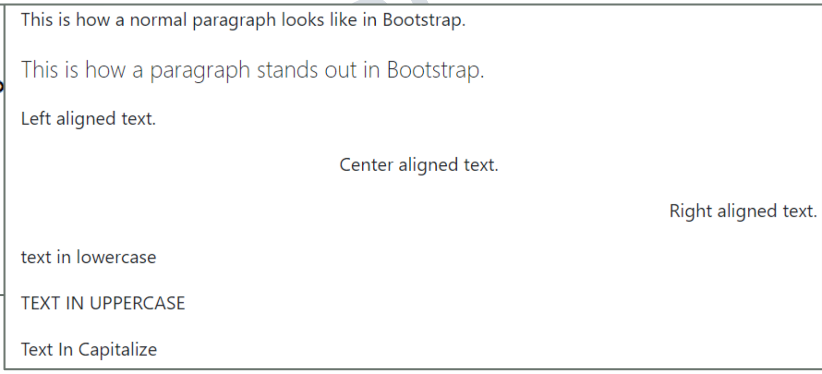
Fancy display heading faded secondary text

Working with Paragraphs

- Bootstrap's global default font-size is 1rem (typically 16px), with a line-height of 1.5. This is applied to the <body> and all paragraphs
  - You can also make a paragraph stand out by adding the class .lead on it.
  - You can also transform the text to lowercase, uppercase or make them capitalize.

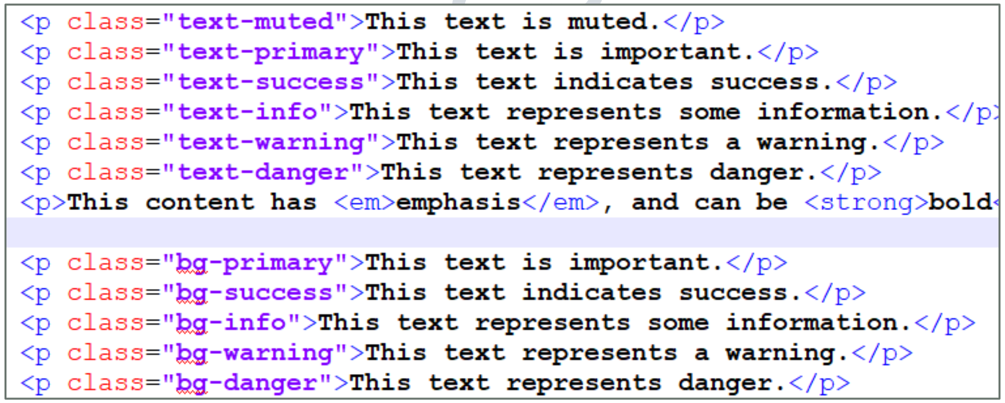


```
<div class="container">
  <p>This is how a normal paragraph looks like in Bootstrap.</p>
  <p class="lead">This is how a paragraph stands out in Bootstrap
  <p class="text-left">Left aligned text.</p>
  <p class="text-center">Center aligned text.</p>
  <p class="text-right">Right aligned text.</p>
  <p class="text-lowercase">Text in lowercase</p>
  <p class="text-uppercase">Text in uppercase</p>
  <p class="text-capitalize">Text in capitalize</p>
```

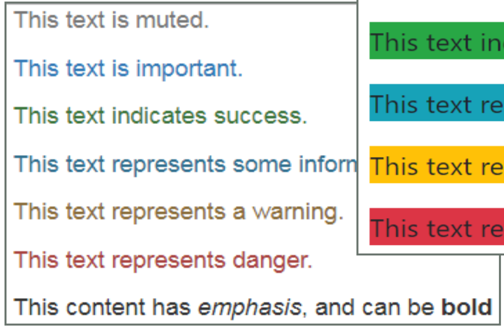


```
This is how a normal paragraph looks like in Bootstrap.

This is how a paragraph stands out in Bootstrap.

Left aligned text.

                         Center aligned text.

                                                   Right aligned text.

text in lowercase

TEXT IN UPPERCASE

Text In Capitalize
```

- Text Coloring
  - Colors are the powerful method of conveying important information in website design.



```
<p class="text-muted">This text is muted.</p>
<p class="text-primary">This text is important.</p>
<p class="text-success">This text indicates success.</p>
<p class="text-info">This text represents some information.</p
<p class="text-warning">This text represents a warning.</p>
<p class="text-danger">This text represents danger.</p>
<p>This content has <em>emphasis</em>, and can be <strong>bold

<p class="bg-primary">This text is important.</p>
<p class="bg-success">This text indicates success.</p>
<p class="bg-info">This text represents some information.</p>
<p class="bg-warning">This text represents a warning.</p>
<p class="bg-danger">This text represents danger.</p>
```



This text is muted.

This text is important.

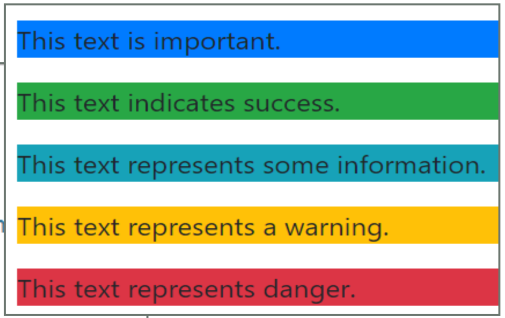
This text indicates success.

This text represents some inforn

This text represents a warning.

This text represents danger.

This content has *emphasis*, and can be **bold**



This text is important.

This text indicates success.

This text represents some information.

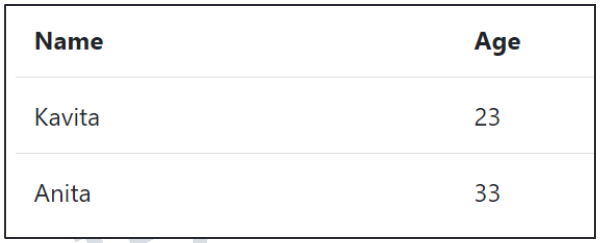
This text represents a warning.

This text represents danger.

Tables

- Bootstrap provides an efficient layout to build elegant tables
  - You can create tables with basic styling that has horizontal dividers and small cell padding, by just adding the Bootstrap's class .table to the <table> element.

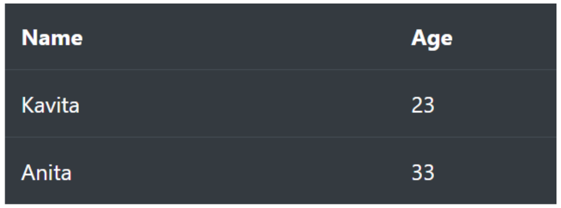
```
<table class="table">
    <tr><th>Name</th><th>Age</th></tr>
    <tr><td>Kavita</td><td>23</td></tr>
    <tr><td>Anita</td><td>33</td></tr>
</table>
```



Name	Age
Kavita	23
Anita	33

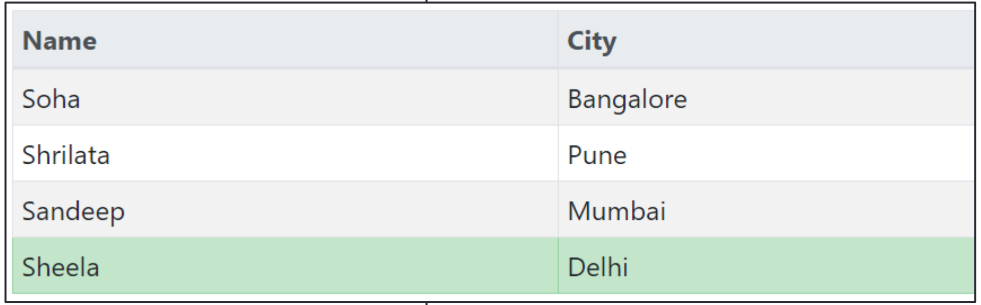
  - The .table-striped class adds zebra-stripes to a table
  - The .table-bordered class adds borders on all sides of the table and cells
  - The .table-condensed class makes a table more compact by cutting cell padding in half
  - The .table-dark class create inverted version of this table, i.e. table with light text on dark backgrounds

<table class="table table-dark">



Name	Age
Kavita	23
Anita	33

```
<table class = "table table-striped table-bordered table-condensed table-sm">
    <caption>Basic Table Layout</caption>
    <thead class="thead-light">
        <tr><th>Name</th><th>City</th></tr>
    </thead>
    <tbody>
        <tr><td>Soha</td><td>Bangalore</td></tr>
        <tr><td>Shrilata</td><td>Pune</td></tr>
        <tr><td>Sandeep</td><td>Mumbai</td></tr>
        <tr class="table-success">
            <td>Sheela</td>
            <td>Delhi</td>
        </tr>
    </tbody>
</table>
```

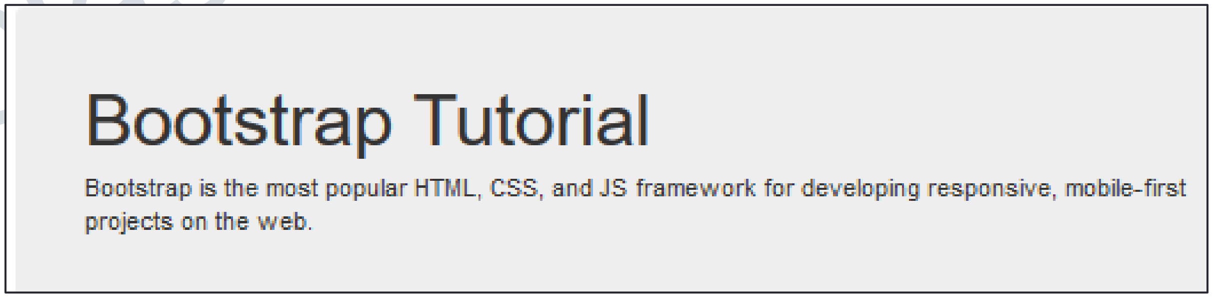


Name	City
Soha	Bangalore
Shrilata	Pune
Sandeep	Mumbai
Sheela	Delhi

Jumbotron

- A jumbotron indicates a big box for calling extra attention to some special content or information.
  - A jumbotron is displayed as a grey box with rounded corners. It also enlarges the font sizes of the text inside it.
  - Just wrap your featured content like heading, descriptions etc. in a <div> element and apply the class .jumbotron on it.
  - Tip: Inside a jumbotron you can put nearly any valid HTML, including other Bootstrap elements/classes.

```
<div class="jumbotron">
   <h1>Bootstrap Tutorial</h1>
   <p>Bootstrap is the most popular HTML, CSS, and JS framework for
        developing responsive, mobile-first projects on the web.</p>
 </div>
```



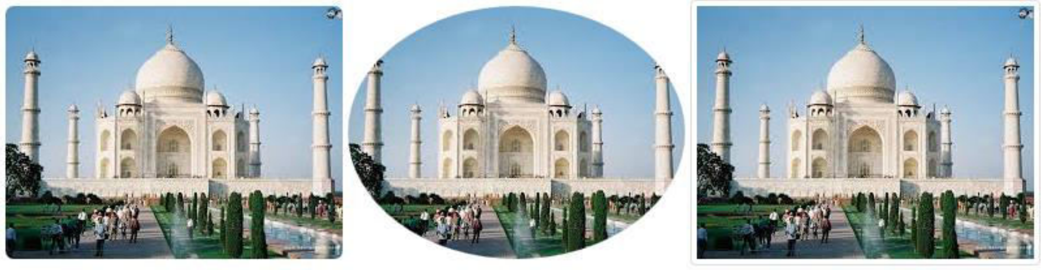
Bootstrap Tutorial

Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile-first projects on the web.

images

- To add images on the webpage use element <img> , it has **three** classes to apply simple styles to images.
  - .img-rounded : To add rounded corners around the edges of the image, radius of the border is **6px**.
  - .img-circle : To create a circle of radius is **500px**
  - .img-thumbnail : To add some padding with grey border , making the image look like a polaroid photo.

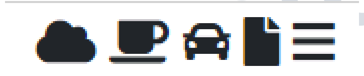
```
<img src="taj.jpg" class="img-rounded"> <!-- rounded  edges-->
<img src="taj.jpg." class="img-circle"> <!-- circle -->
<img src="taj.jpg" class="img-thumbnail"> <!-- thumbnail -->
```

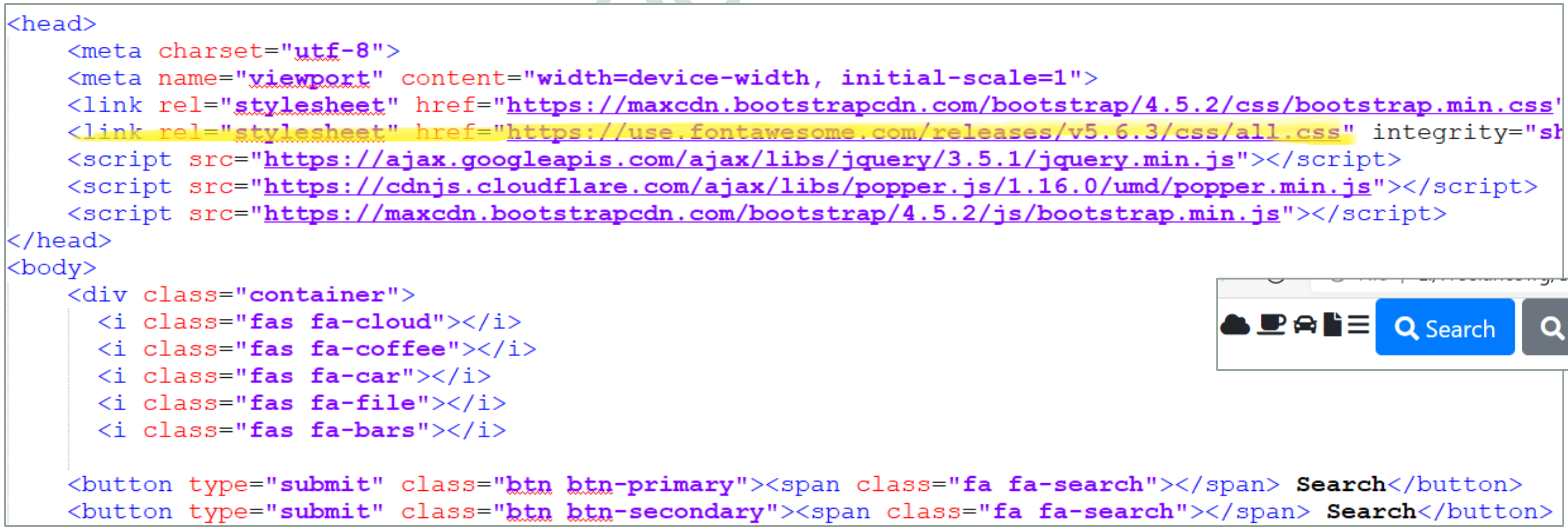


Bootstrap 4 Icons

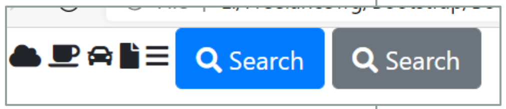
- Bootstrap 4 does not have its own icon library; but there are many free icon libraries to choose from, such as Font Awesome and Google Material Design Icons
  - To use Font Awesome icons, add the following CDN link to your HTML page
  - <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.6.3/css/all.css" integrity="sha384-UHRtZLI+pbxtHCWp1t77Bi1L4ZtiqrqD80Kn4Z8NTSRyMA2Fd33n5dQ8lWUE00s/" crossorigin="anonymous">

```
<i class="fas fa-cloud"></i>
<i class="fas fa-coffee"></i>
<i class="fas fa-car"></i>
<i class="fas fa-file"></i>
<i class="fas fa-bars"></i>
```





```
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css'
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.6.3/css/all.css" integrity="sh
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</head>
<body>
    <div class="container">
      <i class="fas fa-cloud"></i>
      <i class="fas fa-coffee"></i>
      <i class="fas fa-car"></i>
      <i class="fas fa-file"></i>
      <i class="fas fa-bars"></i>

    <button type="submit" class="btn btn-primary"><span class="fa fa-search"></span> Search</button>
    <button type="submit" class="btn btn-secondary"><span class="fa fa-search"></span> Search</button>
```



Alerts

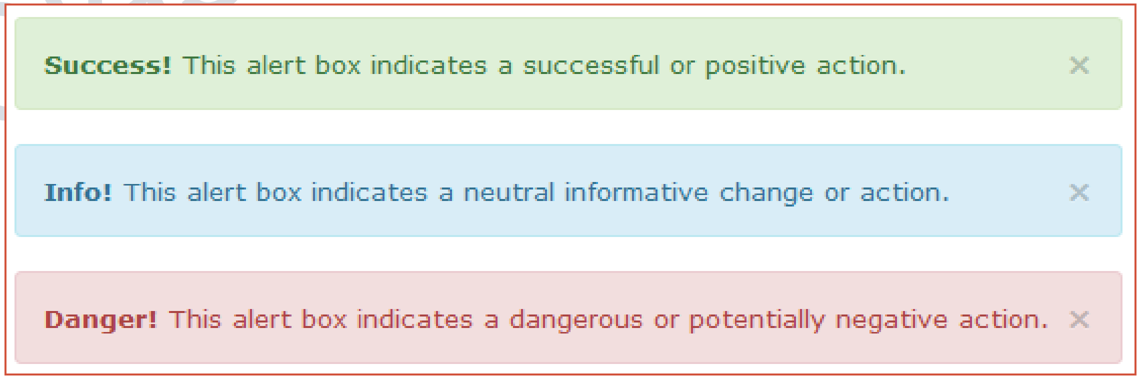
- Bootstrap comes with a very useful component for displaying alert messages in various sections of our website
  - You can use them for displaying a success message, a warning message, a failure message, or an information message.
  - These messages can be annoying to visitors, hence they should have dismiss functionality added to give visitors the ability to hide them.

```
<div class="alert alert-success">
    Amount has been transferred successfully.
</div>
```

contextual classes for alert messages:
alert-success, alert-info, alert-danger, alert-warning

```
<div class="alert alert-success alert-dismissable">
    <button type="button" class="close" data-dismiss="alert">&times;</button>
            Amount has been transferred successfully.
</div>
```

dismissible alert box



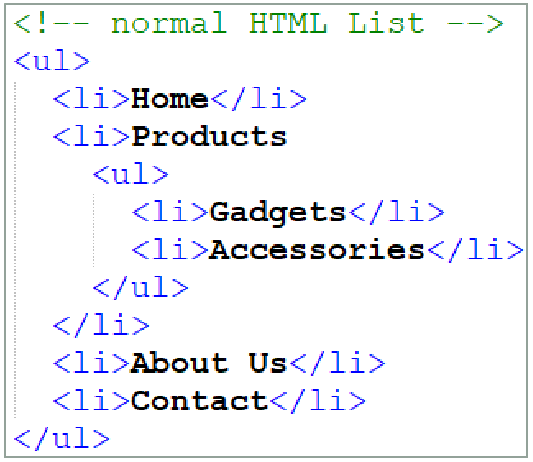
Success! This alert box indicates a successful or positive action. ×

Info! This alert box indicates a neutral informative change or action. ×

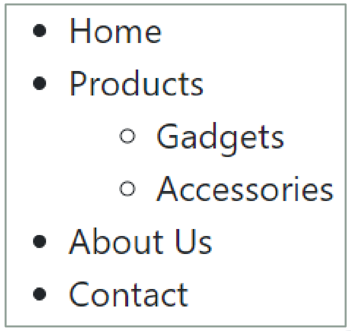
Danger! This alert box indicates a dangerous or potentially negative action. ×

Bootstrap Lists

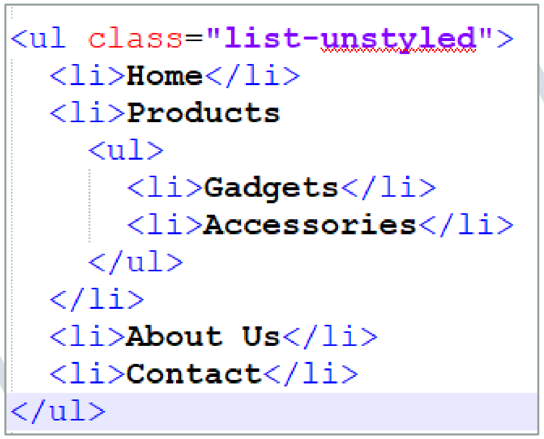
- Unstyled Ordered and Unordered Lists
  - Sometimes you might need to remove the default styling form the list items. You can do this by simply applying the class .list-unstyled to the respective <ul> or <ol> elements



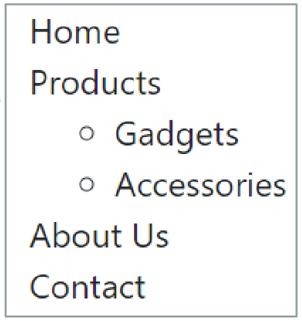
```
<!-- normal HTML List -->
<ul>
  <li>Home</li>
  <li>Products
    <ul>
      <li>Gadgets</li>
      <li>Accessories</li>
    </ul>
  </li>
  <li>About Us</li>
  <li>Contact</li>
</ul>
```



- Home
- Products
  - Gadgets
  - Accessories
- About Us
- Contact

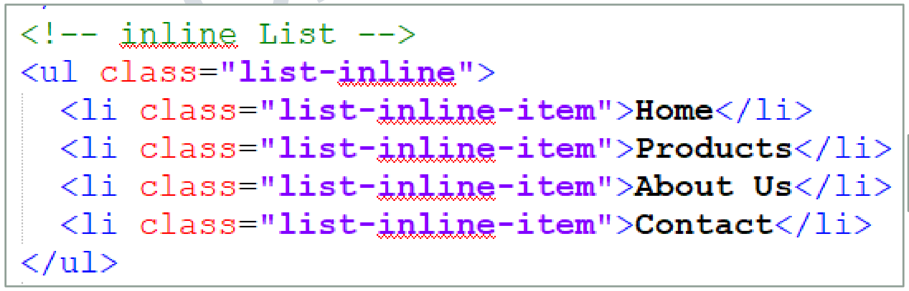


```
<ul class="list-unstyled">
  <li>Home</li>
  <li>Products
    <ul>
      <li>Gadgets</li>
      <li>Accessories</li>
    </ul>
  </li>
  <li>About Us</li>
  <li>Contact</li>
</ul>
```

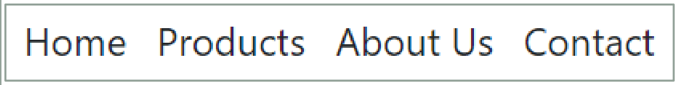


Home
Products
  ○ Gadgets
  ○ Accessories
About Us
Contact

- If you want to create a horizontal menu using ordered or unordered list you need to place all list items in a single line i.e. side by side.
  - You can do this by simply applying the class .list-inline to the respective <ul> or <ol>, and the class .list-inline-item to the <li> elements.



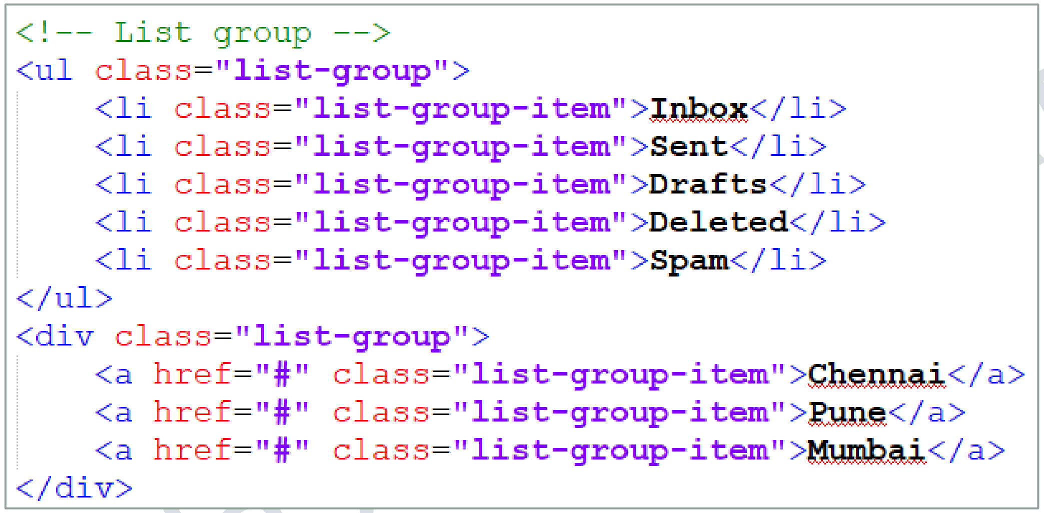
```
<!-- inline List -->
<ul class="list-inline">
  <li class="list-inline-item">Home</li>
  <li class="list-inline-item">Products</li>
  <li class="list-inline-item">About Us</li>
  <li class="list-inline-item">Contact</li>
</ul>
```



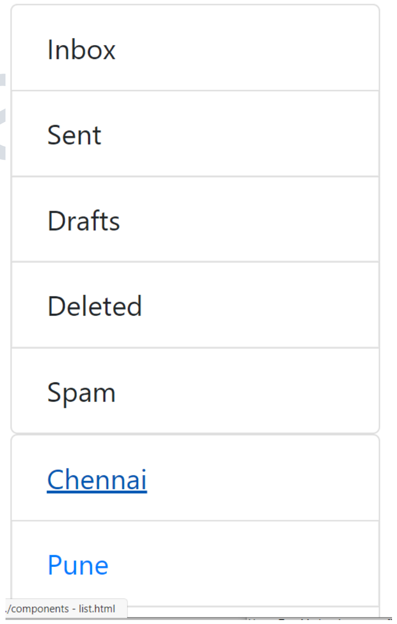
Home   Products   About Us   Contact

Page Components : List Group

- List group is used for creating lists; eg a list of useful resources or a list of recent activities
  - Add class list-group to a <ul> or <div> element to make its children appear as a list.
  - The children can be li or a element, depending on your parent element choice.
  - The child should always have the class list-group-item.



```html
<!-- List group -->
<ul class="list-group">
    <li class="list-group-item">Inbox</li>
    <li class="list-group-item">Sent</li>
    <li class="list-group-item">Drafts</li>
    <li class="list-group-item">Deleted</li>
    <li class="list-group-item">Spam</li>
</ul>
<div class="list-group">
    <a href="#" class="list-group-item">Chennai</a>
    <a href="#" class="list-group-item">Pune</a>
    <a href="#" class="list-group-item">Mumbai</a>
</div>
```

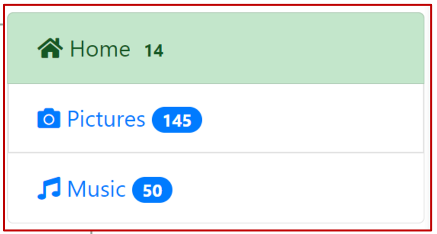


Inbox
Sent
Drafts
Deleted
Spam
Chennai
Pune

Page Components : List Group

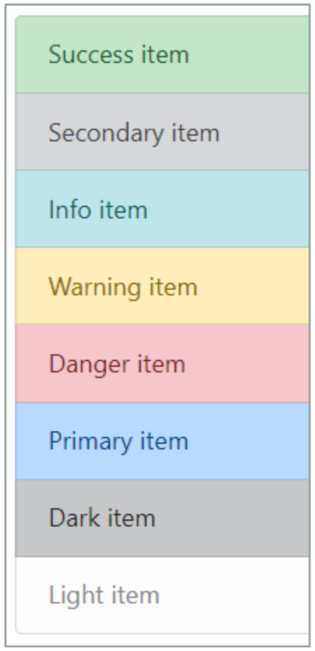
- We can display a number beside each list item using the badge component.
  - Add this inside each “list-group-item” to display badge; badges align to the right of each list item

```
<div class="list-group">
    <a href="#" class="list-group-item list-group-item-success">
        <i class="fa fa-home"></i> Home
        <span class="badge">14</span> </a>
    <a href="#" class="list-group-item">
        <i class="fa fa-camera"></i> Pictures
        <span class="badge badge-pill badge-primary pull-right">145</span></a>
    <a href="#" class="list-group-item">
        <i class="fa fa-music"></i> Music
        <span class="badge badge-pill badge-primary pull-right">50</span></a>
```



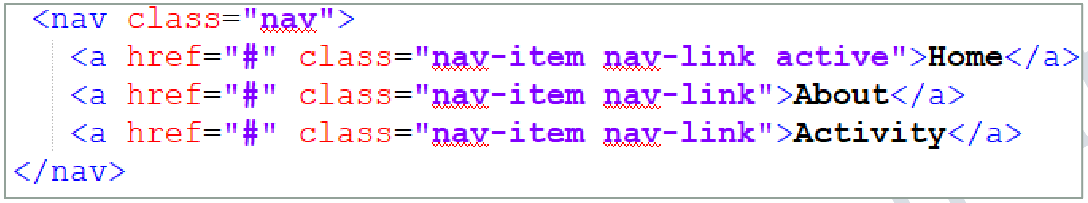
- We can also apply various colors to each list item by adding list-group-item-* classes along with list-group-item.

```
<ul class="list-group">
    <li class="list-group-item list-group-item-success">Success item</li>
    <li class="list-group-item list-group-item-secondary">Secondary item</li>
    <li class="list-group-item list-group-item-info">Info item</li>
    <li class="list-group-item list-group-item-warning">Warning item</li>
    <li class="list-group-item list-group-item-danger">Danger item</li>
    <li class="list-group-item list-group-item-primary">Primary item</li>
    <li class="list-group-item list-group-item-dark">Dark item</li>
    <li class="list-group-item list-group-item-light">Light item</li>
</ul>
```

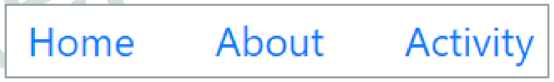


Bootstrap 4 Navs

- Navs : a group of links placed inline with each other to be used for navigation.
  - There are options to make this group of links appear either as tabs or small buttons, the latter known as pills in Bootstrap.
  - If you want to create a simple horizontal menu, add the .nav class to a <ul> element, followed by .nav-item for each <li> and add the .nav-link class to their links:

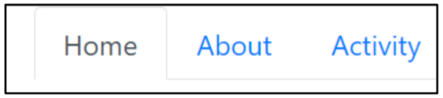


```
 <nav class="nav">
   <a href="#" class="nav-item nav-link active">Home</a>
   <a href="#" class="nav-item nav-link">About</a>
   <a href="#" class="nav-item nav-link">Activity</a>
</nav>
```

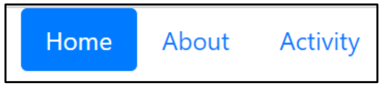


  - Add the class .nav-tabs to the basic nav to generate a tabbed navigation.
  - Similarly, you can create pill based navigation by adding the class .nav-pills on the basic nav instead of class .nav-tabs
  - Vertically stack these pills by attaching an additional class `flex-column`

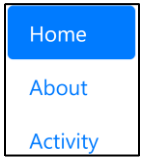
```
<nav class="nav nav-tabs">
```



```
<nav class="nav nav-pills">
```

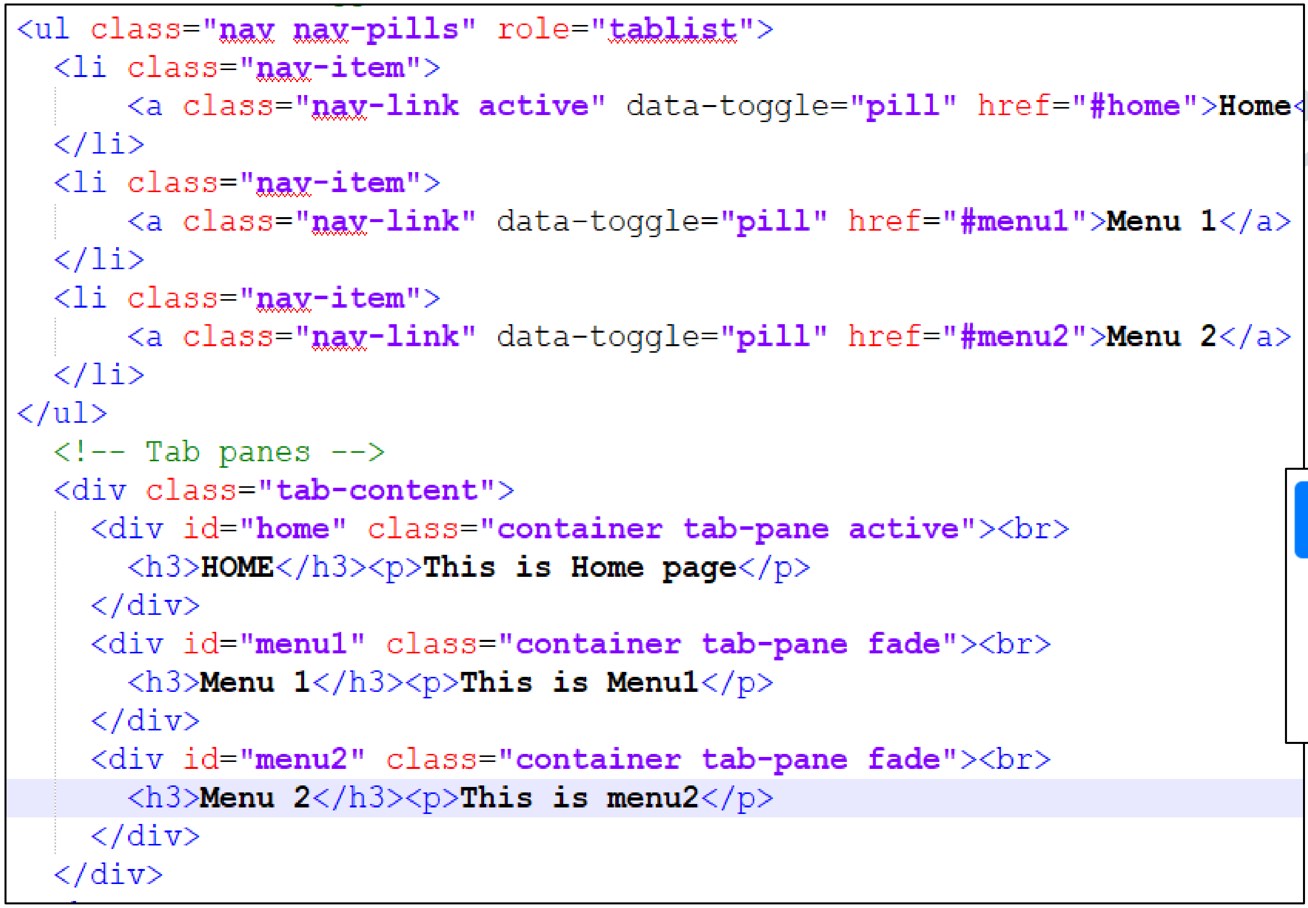


```
<nav class="nav nav-pills flex-column">
```



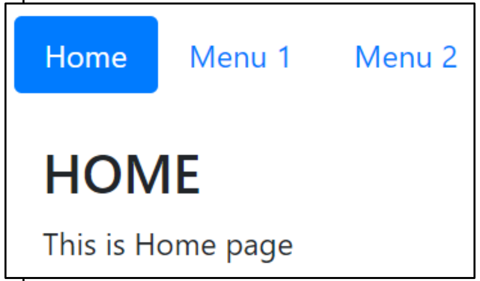
Toggleable / Dynamic Pills

- To make the tabs toggleable, add the data-toggle="tab" attribute to each link.
  - Then add a .tab-pane class with a unique ID for every tab and wrap them inside a <div> element with class .tab-content.



```html
<ul class="nav nav-pills" role="tablist">
  <li class="nav-item">
      <a class="nav-link active" data-toggle="pill" href="#home">Home<
  </li>
  <li class="nav-item">
      <a class="nav-link" data-toggle="pill" href="#menu1">Menu 1</a>
  </li>
  <li class="nav-item">
      <a class="nav-link" data-toggle="pill" href="#menu2">Menu 2</a>
  </li>
</ul>
  <!-- Tab panes -->
  <div class="tab-content">
    <div id="home" class="container tab-pane active"><br>
      <h3>HOME</h3><p>This is Home page</p>
    </div>
    <div id="menu1" class="container tab-pane fade"><br>
      <h3>Menu 1</h3><p>This is Menu1</p>
    </div>
    <div id="menu2" class="container tab-pane fade"><br>
      <h3>Menu 2</h3><p>This is menu2</p>
    </div>
  </div>
```

- To fade the tabs in and out when clicking on them, add the .fade class to .tab-pane



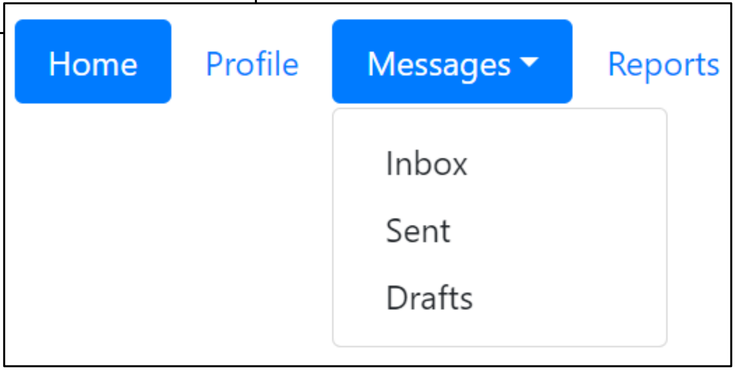
HOME

This is Home page

Navs with dropdown

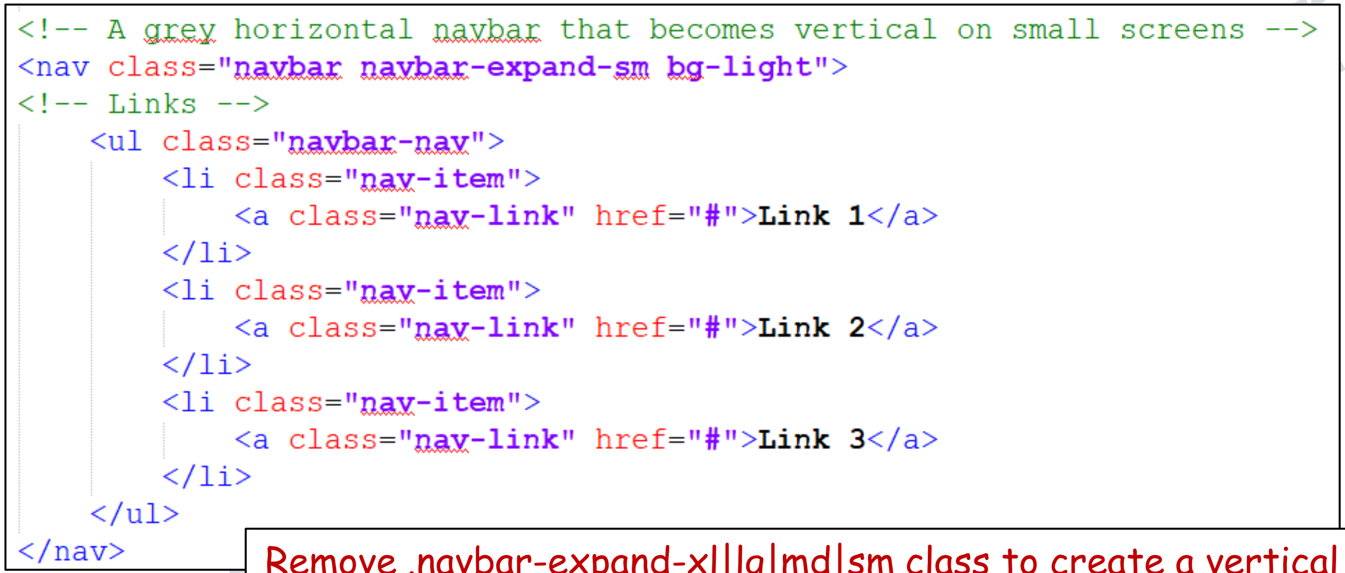
- You can add dropdown menus to a link inside tabs and pills nav with a little extra markup.
  - The four CSS classes .dropdown, .dropdown-toggle, .dropdown-menu and .dropdown-item are required

```
<nav class="nav nav-pills">
  <a href="#" class="nav-item nav-link active">Home</a>
  <a href="#" class="nav-item nav-link">Profile</a>
  <div class="nav-item dropdown">
      <a href="#" class="nav-link dropdown-toggle"
              data-toggle="dropdown">Messages</a>
      <div class="dropdown-menu">
          <a href="#" class="dropdown-item">Inbox</a>
          <a href="#" class="dropdown-item">Sent</a>
          <a href="#" class="dropdown-item">Drafts</a>
      </div>
  </div>
  <a href="#" class="nav-item nav-link">Reports</a>
</nav>
```

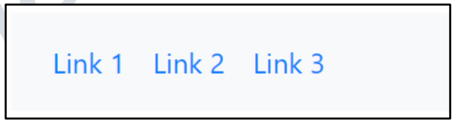


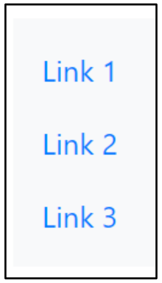
Navbar

- A navbar is a navigation header that is placed at the top of the page
  - A standard navigation bar is created with the .navbar class, followed by a responsive collapsing class: .navbar-expand-xl|lg|md|sm
  - To add links inside the navbar, use a <ul> element with class="navbar-nav".
  - Then add <li> elements with a .nav-item class followed by an <a> element with a .nav-link class



```
<!-- A grey horizontal navbar that becomes vertical on small screens -->
<nav class="navbar navbar-expand-sm bg-light">
<!-- Links -->
    <ul class="navbar-nav">
        <li class="nav-item">
            <a class="nav-link" href="#">Link 1</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Link 2</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Link 3</a>
        </li>
    </ul>
</nav>
```



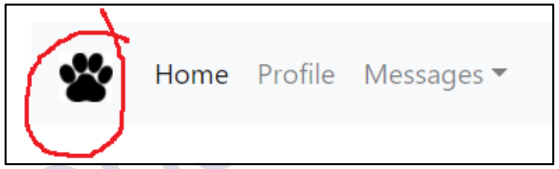


Remove .navbar-expand-xl|lg|md|sm class to create a vertical nav bar

Navbar

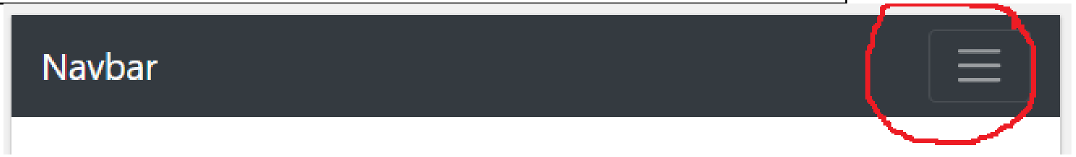
- Brand / Logo
  - The .navbar-brand class is used to highlight the brand/logo/project name of your page

```
<a href="#" class="navbar-brand">
   <img src="paws.png" height="28" alt="CoolBrand">
</a>
```

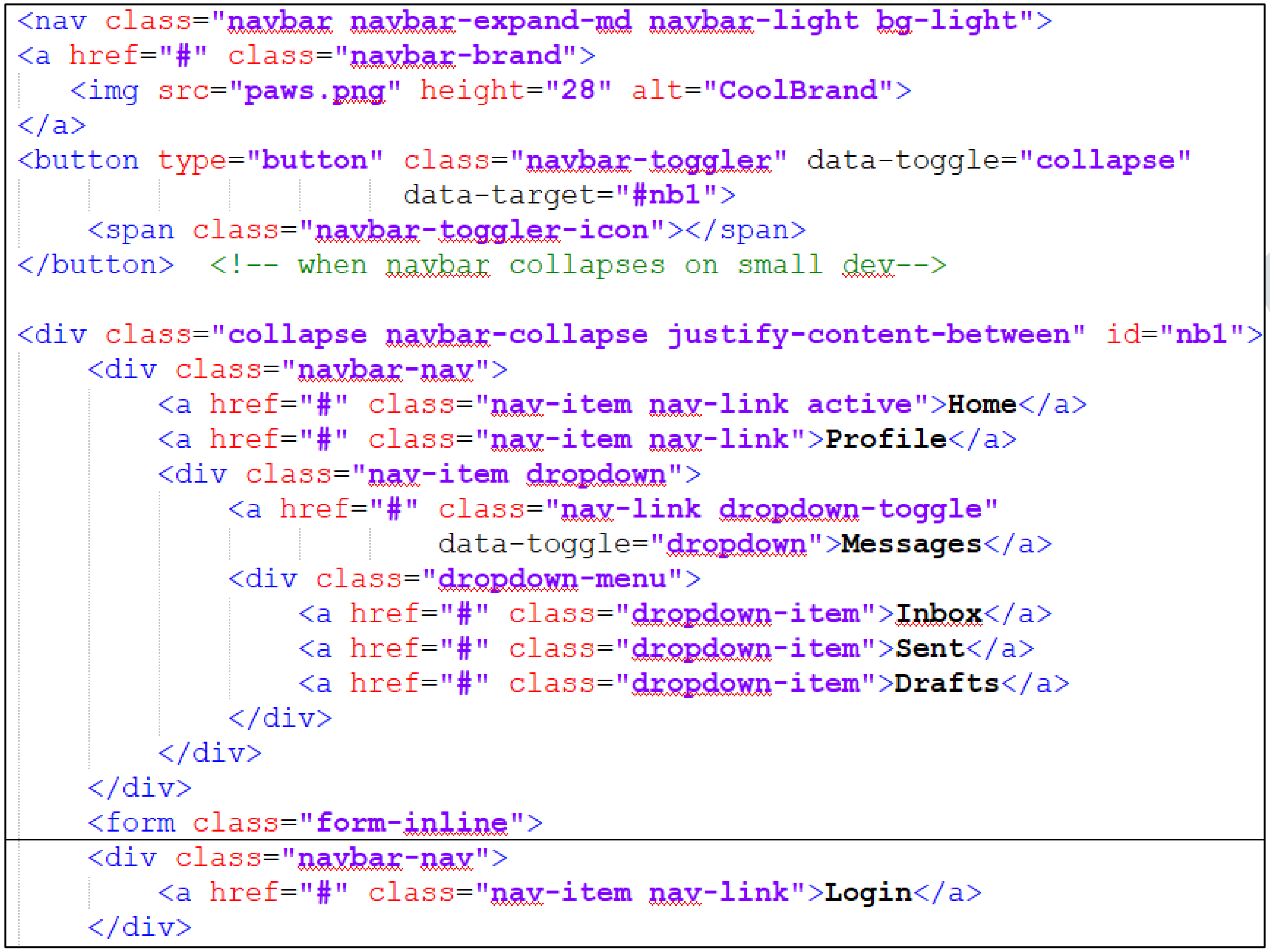


- Collapsing The Navigation Bar
  - Very often, especially on small screens, you want to hide the navigation links and replace them with a button that should reveal them when clicked on.
  - To create a collapsible navigation bar, use a button with class="navbar-toggler", data-toggle="collapse" and data-target="#thetarget".
  - Then wrap the navbar content (links, etc) inside a div element with class="collapse navbar-collapse", followed by an id that matches the data-target of the button: "thetarget".

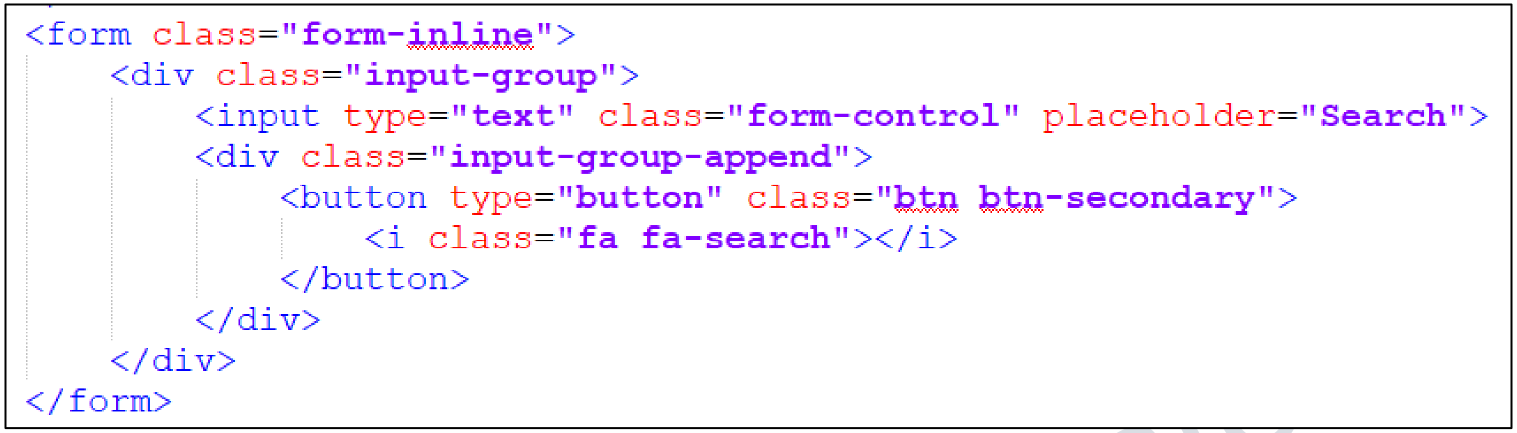
```
<!-- Toggler/collapsibe Button -->
 <button class="navbar-toggler" type="button" data-toggle="collapse"
             data-target="#collapsibleNavbar">
   <span class="navbar-toggler-icon"></span>
 </button>
 <!-- Navbar links -->
 <div class="collapse navbar-collapse" id="collapsibleNavbar">
```



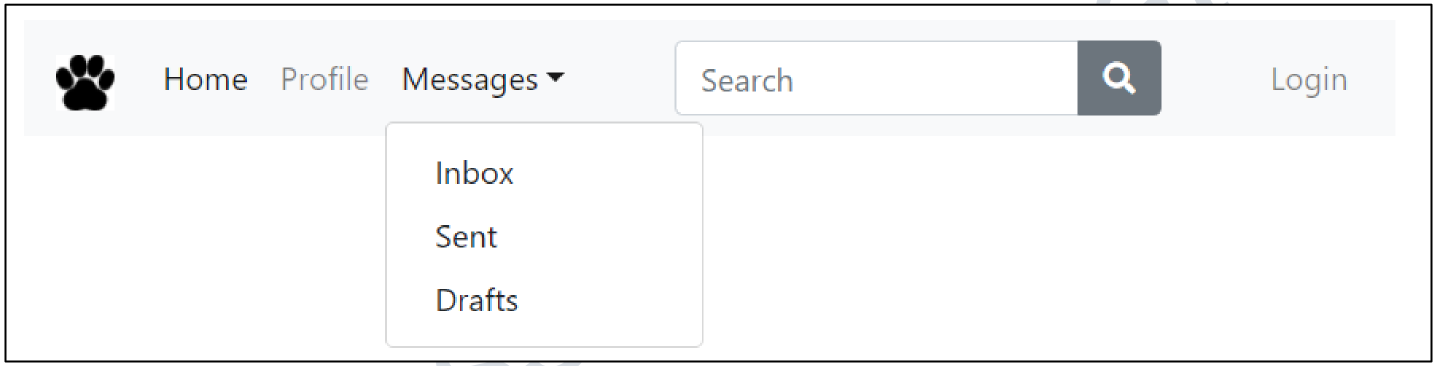
Navbar

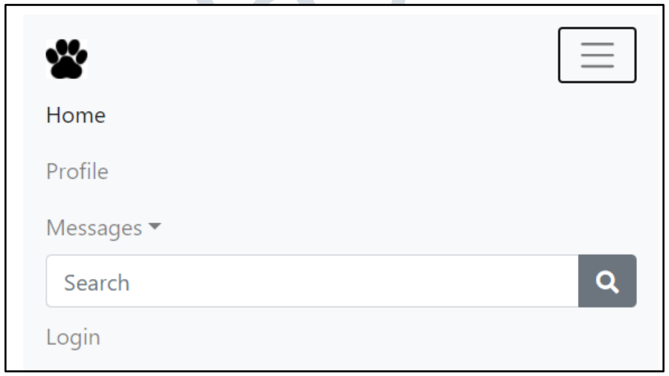


```html
<nav class="navbar navbar-expand-md navbar-light bg-light">
<a href="#" class="navbar-brand">
   <img src="paws.png" height="28" alt="CoolBrand">
</a>
<button type="button" class="navbar-toggler" data-toggle="collapse"
                      data-target="#nb1">
    <span class="navbar-toggler-icon"></span>
</button>  <!-- when navbar collapses on small dev-->

<div class="collapse navbar-collapse justify-content-between" id="nb1">
    <div class="navbar-nav">
        <a href="#" class="nav-item nav-link active">Home</a>
        <a href="#" class="nav-item nav-link">Profile</a>
        <div class="nav-item dropdown">
            <a href="#" class="nav-link dropdown-toggle"
                        data-toggle="dropdown">Messages</a>
            <div class="dropdown-menu">
                <a href="#" class="dropdown-item">Inbox</a>
                <a href="#" class="dropdown-item">Sent</a>
                <a href="#" class="dropdown-item">Drafts</a>
            </div>
        </div>
    </div>
    <form class="form-inline">
    <div class="navbar-nav">
        <a href="#" class="nav-item nav-link">Login</a>
    </div>
```



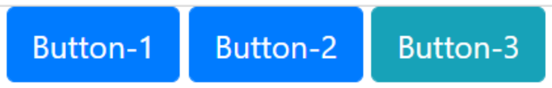
```
<form class="form-inline">
    <div class="input-group">
        <input type="text" class="form-control" placeholder="Search">
        <div class="input-group-append">
            <button type="button" class="btn btn-secondary">
                <i class="fa fa-search"></i>
            </button>
        </div>
    </div>
</form>
```

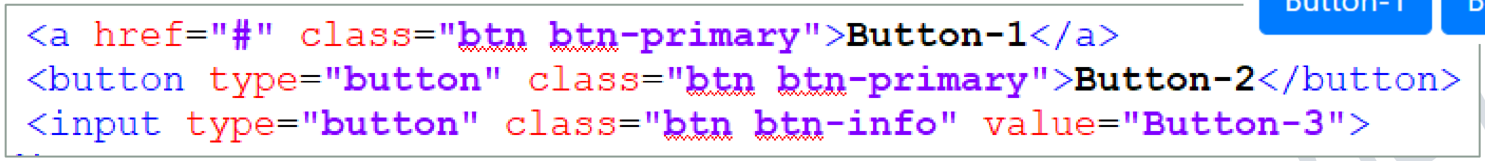

Home
Profile
Messages
Search
Login
Inbox
Sent
Drafts


Home
Profile
Messages
Search
Login

Standing Out : Buttons

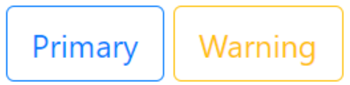
- Its easy to convert an a, button, or input element into a fancy bold button in Bootstrap; just have to add the btn class

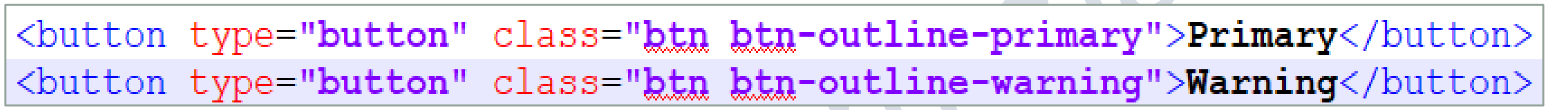




```html
<a href="#" class="btn btn-primary">Button-1</a>
<button type="button" class="btn btn-primary">Button-2</button>
<input type="button" class="btn btn-info" value="Button-3">
```

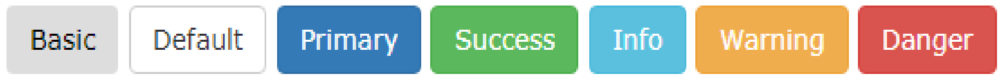
  - You can also create outline buttons by replacing the button modifier classes

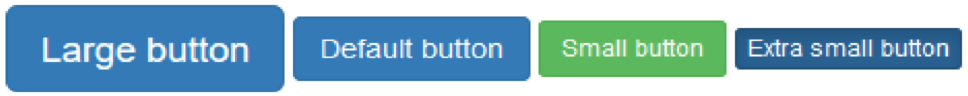




```html
<button type="button" class="btn btn-outline-primary">Primary</button>
<button type="button" class="btn btn-outline-warning">Warning</button>
```

  - Buttons come in various color options:
    - btn-default for white
    - btn-primary for dark blue
    - btn-success for green
    - btn-info for light blue
    - btn-warning for orange
    - btn-danger for red
  - And in various sizes:
    - btn-lg for large buttons
    - btn-sm for small buttons
    - btn-xs for extra small button



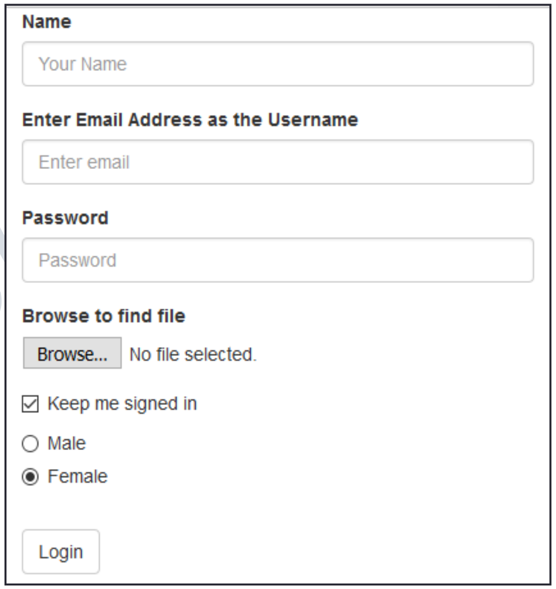


```html
<button type="button" class="btn btn-primary btn-lg">Large button</button>
<button type="button" class="btn btn-primary">Default button</button>
<button type="button" class="btn btn-primary btn-sm btn-success">Small button</button>
```

Creating Forms

- Bootstrap provides three different types of form layouts:
  - Vertical Form (default form layout)
  - Horizontal Form
  - Inline Form

Eg ->


Name

Your Name

Enter Email Address as the Username

Enter email

Password

Password

Browse to find file

Browse... No file selected.

☑ Keep me signed in

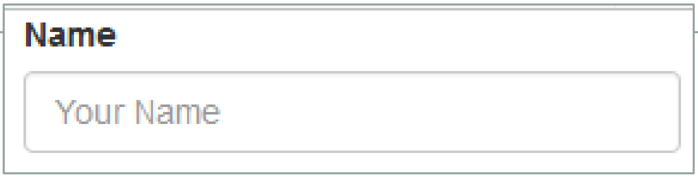
○ Male

● Female

Login

- Standard rules for all three form layouts:
  - Wrap labels & form controls in <div class="form-group"> (for optimum spacing)
  - Add class .form-control to all textual <input>, <textarea>, and <select> elements

```
<form class="form">
    <div class="form-group">
        <label for="n1">Name</label>
        <input type="text" class="form-control" id="n1"  placeholder="Your Name" />
    </div>
</form>
```


Name

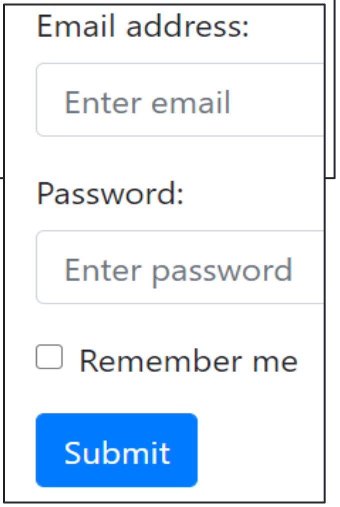
Your Name

class form-control in an input element will make it a full-width element

Creating Forms : Vertical Form Layout

- This is the default Bootstrap form layout in which styles are applied to form controls without adding any base class to the <form> element or any large changes in the markup.
- The form controls in this layout are stacked with left-aligned labels on the top.

```
<form action="#">
   <div class="form-group">
       <label for="email">Email address:</label>
       <input type="email" class="form-control" placeholder="Enter email" id="email">
   </div>
   <div class="form-group">
       <label for="pwd">Password:</label>
       <input type="password" class="form-control" placeholder="Enter password" id="pwd">
   </div>
   <div class="form-group form-check">
       <label class="form-check-label">
           <input class="form-check-input" type="checkbox"> Remember me
       </label>
   </div>
   <button type="submit" class="btn btn-primary">Submit</button>
</form>
```


Email address:

Enter email

Password:

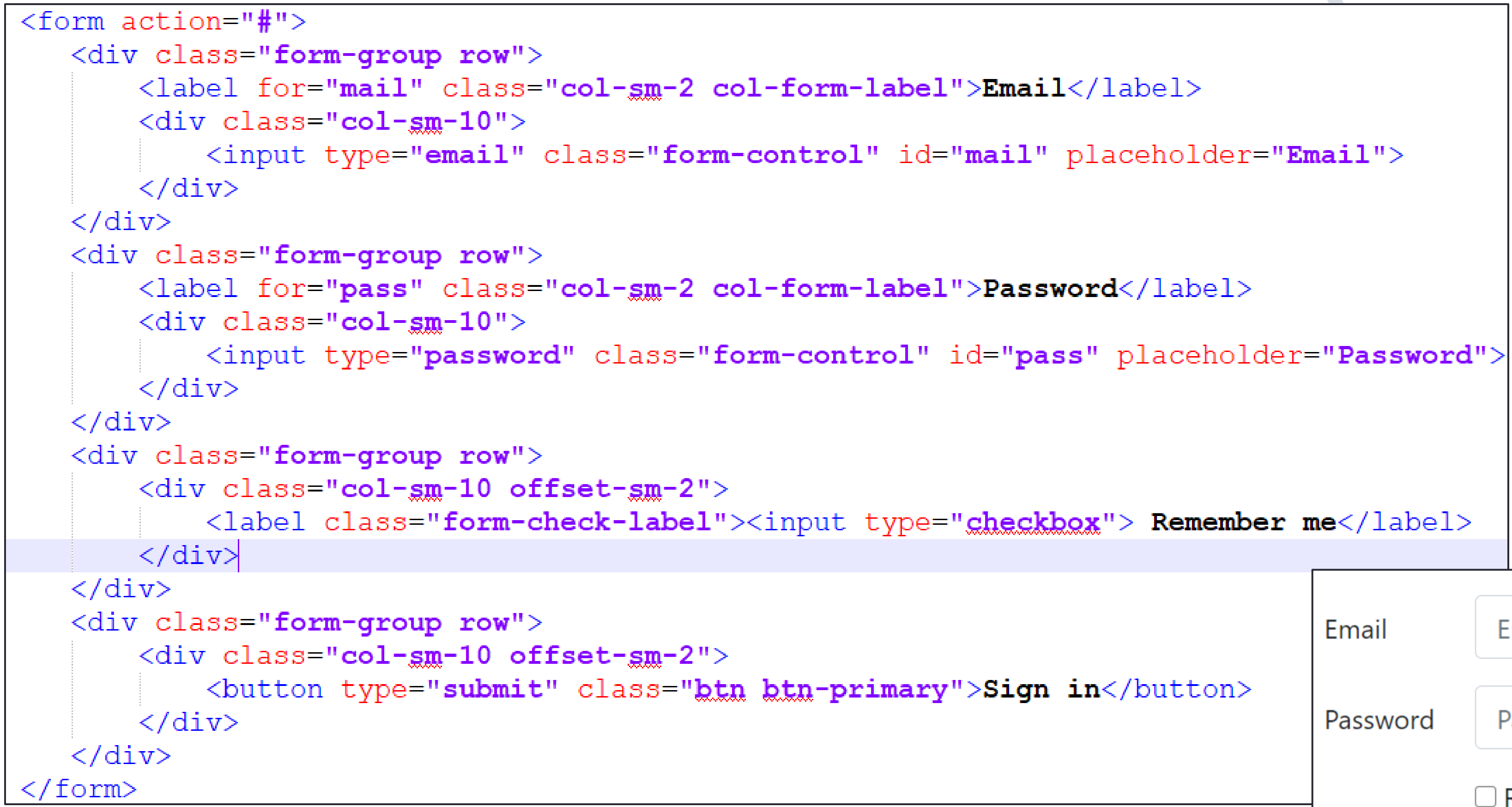
Enter password

Remember me

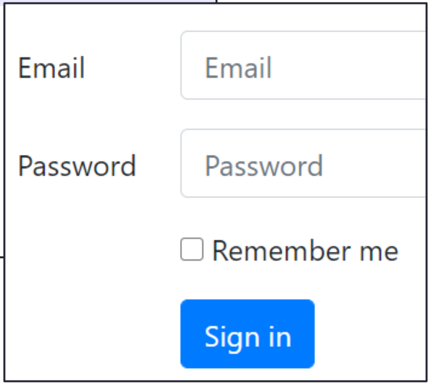
Submit

Creating Forms : Horizontal Form Layout

- Labels and form controls are aligned side-by-side using the Bootstrap grid classes.
- To create this layout add the class .row on form groups and use the .col-*-* grid classes to specify the width of your labels and controls.



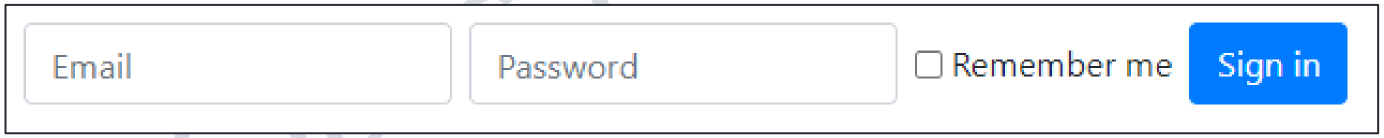
```
<form action="#">
   <div class="form-group row">
       <label for="mail" class="col-sm-2 col-form-label">Email</label>
       <div class="col-sm-10">
           <input type="email" class="form-control" id="mail" placeholder="Email">
       </div>
   </div>
   <div class="form-group row">
       <label for="pass" class="col-sm-2 col-form-label">Password</label>
       <div class="col-sm-10">
           <input type="password" class="form-control" id="pass" placeholder="Password">
       </div>
   </div>
   <div class="form-group row">
       <div class="col-sm-10 offset-sm-2">
           <label class="form-check-label"><input type="checkbox"> Remember me</label>
       </div>
   </div>
   <div class="form-group row">
       <div class="col-sm-10 offset-sm-2">
           <button type="submit" class="btn btn-primary">Sign in</button>
       </div>
   </div>
</form>
```



Creating Forms : Inline Form Layout

- Additional rule for an inline form: Add class .form-inline to the <form> element

```
<form class="form-inline">
   <div class="form-group mr-2">
       <label class="sr-only" for="inputEmail">Email</label>
       <input type="email" class="form-control"
              id="inputEmail" placeholder="Email">
   </div>
   <div class="form-group mr-2">
       <label class="sr-only" for="inputPassword">Password</label>
       <input type="password" class="form-control"
              id="inputPassword" placeholder="Password">
   </div>
   <div class="form-group mr-2">
       <label><input type="checkbox" class="mr-1"> Remember me</label>
   </div>
   <button type="submit" class="btn btn-primary">Sign in</button>
</form>
```


Email
Password
Remember me
Sign in