

● How the Internet Works

- Client - Server
 - Client makes a **request** to go to a webpage "www.google.com"
 - **request** is sent to the Internet Service Provider (ISP)
 - Then the ISP will relay the message to the DNS Server
 - DNS Server
 - Input: web url
 - Output: IP Address of website
 - IP Address is obtained
 - Browser sends message to the server based on the IP address
 - Server sends back the data

● How websites work

- HTML
 - Structure
- CSS
 - Styling
- JS
 - Actions

● Introduction to HTML

- HyperText Markup Language
- File Structure
 - <!DOCTYPE html>
 - Symbolizes that the file is HTML5
 - <html>
 - Everything contained in the tags, will be html code
 - <head>
 - Holds information about the webpage
 - <title>
 - <meta>
 - Tells the browser how to display the html web page
 - charset=
 - utf-8
 - name="description"
 - Brief description of the web page, that tells search engines how to display the page hyperlink
 - <body>
- Tags
 - Header
 - <h#>
 - h1,h2,h3,h4,h5,h6
 - Line Break

-

 - White space
- Horizontal Rule
 - <hr>
 - Horizontal Line
- Paragraph
 - <p>
 - Goes into a new line
- Italicize
 - vs <i>
 - Use em over i, as i is just for style, but em conveys more information to the user as well as to the browser
 - Emphasis
 -
 - Puts an emphasis on the tag
 - Italicize
 - <i>
 - Styles the text
- Bold
 - vs
 - Use strong over b, as it highlights an added importance
- Lists
 - Ordered List
 - Numbers
 -
 - Unordered List
 - Bullet points
 -
- Image
 -
 - src=
 - Can be a URL to another source where image is on the internet
 - Can be a local file in the path of the page directory
 - alt=
 - Alternative, if src fails to obtain an image to render to the page
- HyperLink
 - Anchor
 - <a>
 - href=
 - URL to link to
 - download

- Downloads the contents of the URL
- Table
 - <table>
 - <thead>
 - Table Head
 - <tbody>
 - Table Body
 - <tfoot>
 - Table Footer
 - <tr>
 - Table Row
 - <td>
 - Table Data
 - <th>
 - Table header cell
- Forms
 - <form>
 - Labels
 - <label>
 - Inputs
 - <input>
 - email
 - color
 - password
 - checkbox
 - date
 - range
 - radio
 - ...

● Introduction to CSS

- Every elements in a web page is essentially all composed of a series of boxes
- Inline CSS
 - CSS attributes that can be placed in the html file to style the page in the form of being in the individual html tag themselves
- Internal CSS
 - Use of the <style> tag inside the <head>
- External CSS
 - Link to the .css file with <link rel="stylesheet" href="css_file"> in the <head> of html file
- Anatomy of CSS Syntax
 - selector { property : value; }
- CSS Selectors

- Tag Selector
 - Ex. h1, img, body
- Class Selector
 - Inside HTML, of the html tag include class=" "
 - Then in the CSS, select the class by including a . in the front
 - .class_identifier { property : value; }
- ID Selector
 - Inside HTML, of the html tag, include id=" "
 - Then in the CSS, select the id by including a # in the front
 - #id_identifier { property : value; }
- ID vs Class Selectors
 - ID - use only for a single html tag where you want to modify its style
 - There can only be one id_identifier for a given html file
 - Class - use for a group of html tags where you want to have a consistent style for the group
 - A html tag can have multiple classes

● Intermediate CSS

- <div>
 - Combines a group of html elements into a single box
 - Useful to structure and divide up web content
-
 - An inline element where you can use to pick a sub-section of a html element, which can then be used to style
- The Box Model
 - Width
 - Height
 - Padding
 - Border
 - Border Size
 - Solid, Dashed, ...
 - Margin
- Display Property
 - We can also modify the display property for any html element
 - Ex. display : inline;
 - Block
 - Elements that will take up it's own line, where the width takes up the whole screen
 - Inline
 - Inline-Block
 - Allows other elements to sit to their left or right side
 - None

- Html element is hidden as if it never existed
 - Alternative, visibility : hidden
 - The element can not be seen, but it fills the original size position
- CSS Static and Relative Positioning
 - 1. Content dictates everything
 - 2. Order of content comes from the code
 - 3. Parent - Children relationship
 - Children elements are layered on top of the parent elements
 - Position
 - 4 Methods to change the layout positioning of elements
 - Static
 - All html elements are static by default
 - Relative
 - The adjustment position is applied in **relative to the static** positioning
 - position : static;
 - Then we can only see changes once we change the coordinate values.
 - top, bottom, left, right
 - Relative positioning acts independently on the html element
 - Absolute
 - Moves element **relative to its parent** element
 - Adds margins relative to its parent element
 - Absolute positioning acts dependently on the html element
 - Fixed
 - The element will be fixed in place, even when the user scrolls around the web page
 - Useful for navigation bars
- Centering Elements with CSS
 - text-align : center
 - Centers all elements, where it does not have a width set
 - margin : top right bottom left
 - auto - will be used to center the element
- Font Styling
 - font-family
 - serif
 - sans-serif
 - Note that the availability of fonts varies based on the browser and operating system. So if you choose a font for your web site, be mindful of its availability to your end users.

- em
 - Example.
 - font-size: 2rem;
 - 1 em = 16px = 100%
 - Dynamic sizing of text, in proportion to a standard, being 16px = 1em
 - Dynamic sizing of text, relative to the html elements parent
- rem
 - Dynamic sizing of text, relative to the root
 - Preferable to use for texts
- CSS Float and Clear
 - float
 - The html element “floats”, so that other elements can wrap around the element
 - clear
 - Opposite of float, the html element does not allow wrapping around of other elements

● Bootstrap 4

- Front-end library/framework
- Benefits
 - Adaptive UI display for various kinds of screens
 - Catalog of pre-styled elements
- Installation
 - 1. Download the Bootstrap files to local and link to the local in HTML <head>
 - Though this can cause low performance, as the browser will not recognize the file, and will download it.
 - Method is not recommended
 - 2. Include the individual Bootstrap reference link
 - 3. Include the Start Template from Bootstrap, which also includes Popper.js and JQuery
- Design
 - Wireframe
 - Low fidelity black and white sketch, often done using pencil, to highlight the main aspects of a web page
 - Mockup
 - High fidelity UI draft of what the final web page can look like.
 - Prototype
 - An interactive UI
- Navigation Bar
 - <nav>
 - Similar to a div, but more clear for readability

- Navbar
- Nav-brand
- Margin
 - ml-auto
 - As much margin is used to push the item all the way to the right
- Toggler - Dropdown Menu
- Colors
- Grid Layout System
 - Consists of 12 columns
 - Can further specify by indicating col-#
 - Further specify by screen size
 - Large - lg
 - Desktops/Laptops
 - Medium - md
 - Small desktops/laptops
 - Ex.
 - col-md-6
 - Take up 6 units for medium sizes and upward
 - Small - sm
 - Tablets
 - Extra small - xs
 - Phones
- Containers
 - Basic building block of the bootstrap grid system
 - All grid elements are containing inside of a container element
 - class="container"
 - Responsive to screen size
 - By default, contains margins
 - class="container-fluid"
 - Adaptive to the width of the screen.
 - Takes up 100% width of the screen
- Buttons
 - Check Bootstrap documentation
- Carousel
 - Can adjust time
 - Can put transition styles
- Cards
 - Container, that usually consists of a image and text enclosed in a box
- Z-Index & Stacking Order
 - Natural stacking order
 - The first html elements will be placed towards the back.

- Following html elements will be stacked on top of the previous
- Z-Index
 - Default z-index of html elements is 0
 - z-index of greater than 0 will be placed on top
 - z-index of less than 0 will be placed towards the back
 - Doesn't work for position:static
- Media Query Breakpoints
 - Mobile-First
 - More people go on websites on their phones than on their desktops
 - Google ranks searches based on mobile-friendly-ness
 - @media <type> <feature> { ... }
 - Works like a True/False statement. If true, run the code
 - <type>
 - print
 - screen
 - speech
 - <feature>
 - Example:
 - @media screen (min-width: 900px) { ... }
 - Code should only be activated if the media is a screen with a min-width of 900px
 - Can also combine multiple features
 - @media screen (min-width: 900px) and (max-width: 1500px) { ... }
 - Different types of views
 - Device
 - Size of the device, fixed
 - Browser Size
 - Viewport
 - Size of the screen that your website is being displayed on
- Code Refactoring
 - 1. Readability
 - 2. Modularity
 - 3. Efficiency
 - 4. Length
- Combining Selectors
 - Multiple Selectors
 - selector1, selector2, selector# { ... }
 - Styling is applied to all selectors in the list seperated by commas
 - Hierarchical Selectors
 - selector1 selector2 { ... }

- selector1 is the parent, selector2 is the child
 - Styling is performed on the child element
 - Note: This is not really used for id selectors, as that is redundant
- Combined Selectors (Non-hierarchical)
 - selector1.selector2{ ... }
 - OR
 - selector#selector2{ ... }
- Selector Priority
 - Selector which is more specific gets higher priority
 - Element Selector
 - Class Selector
 - Try to only have one class for a html element, even though Bootstrap doesn't follow this
 - ID Selector
 - Use ID sparingly
 - ID has the highest selector priority
 - Avoid inline styling

● Introduction to Javascript

- Javascript runs on the browser rather than the server. Whereas before in order to perform action, data and operations were being sent to the server.
- Javascript is an interpreted programming language
- Javascript Alerts
 - alert("Text to Display Alert");
- JS Data Types
 - Different Types
 - String
 - Boolean
 - Number
 - typeof(<argument to check type>)
 - Output of function
 - "number"
 - "boolean"
 - "string"
- JS Variables
 - prompt("Dialogue to display for prompt");
 - To declare a variable use, var.
 - var, variables can later be changed to a different value
 - Example:
 - var myName = "Anand";
- Naming and Naming Conventions for JS Variables
 - Variable names cannot start with a number

- Variable can only contain numbers, letters, \$, and/or _
- Variables follow a camel-case format.
 - First word contains lowercase, then subsequent words contain capital letters for their names
- String Concatenation
 - Can combine strings using +
 - Example:
 - "Hello" + "World" -> "HelloWorld"
- String Lengths and Retrieving the Number of Characters
 - To check the number of characters in a string
 - `stringVariable.length`
- Slicing and Extracting Parts of a String
 - Slice Function
 - `stringslice(x, y);`
 - Slices the string from x to y. Including x, up to but not including y
 - Example:
 - `"Anand".slice(0,3);`
 - "Ana"
- Changing Casing in Text
 - `toUpperCase()`
 - Example:
 - `"word".toUpperCase()`
 - "WORD"
 - `toLowerCase()`
- Basic Arithmetic and the Modulo Operator in Javascript
 - Modulo
 - %
 - Gives the remainder of a division
 - Example:
 - `9 % 6 = 3`
- Increment and Decrement Expressions
 - `x++`
 - `x = x + 1;`
 - `x--`
 - `x = x - 1;`
- Creating and Calling Functions
 - Declaring functions:
 - `function myFunction() { <Code to be executed> }`
 - Calling a function
 - `myFunction();`
- Function Parameters and Arguments
 - Example:

- `function getMilk(money) { ... }`
- **Function Outputs and Return Values**
 - Include a return statement in the function to make it a function that has an output
- **Control Statements: If-Else Conditionals and Logic**
 - `if (condition) { < code to run, if condition is true > }`
 - `else { < code to run, if condition is false > }`
- **Comparators and Equality**
 - `===`
 - Is equal to
 - Checks both for the value and the data types
 - `!==`
 - Is not equal to
 - `>, <, <=, >=`
 - `==`
 - Only checks if the values equate to each other.
 - Doesn't take into account the data types
 - So, `1 == "1"`, will be equal to true
- **Combining Comparators**
 - `&&`
 - AND
 - `||`
 - OR
 - `!`
 - NOT
- **Javascript Arrays**
 - Creating an array
 - Example:
 - Empty Array
 - `var names = [];`
 - Prefilling an array with elements
 - `var names = ["Bob", "John", "Jennifer"];`
 - Retrieving an element from an array
 - Example:
 - `names[1]; // "John"`
 - Array elements always start at 0
 - Retrieving the number of elements in an array
 - `Array.length` // Return the number of elements
 - Checking if an array contains a specific element
 - `Array.includes(<element to check if it exists in the array>);`
 - Returns a boolean
- **Adding Elements and Intermediate Array Techniques**
 - Adding an element at the end of an array

- `Array.push(<Element to push into the array>);`
- Remove and retrieve the last element of an array
 - `Array.pop();` // Returns the last element of the array
- While Loops
 - `while (<some condition is true>) { // Code to run }`
- For Loops
 - `for(startingCondition ; conditionToRunLoopIfTrue; conditionModifier) { // Code to run if condition is true }`
 - Example:
 - `for(i = 0; i < 100; i++) { // Run code }`

● Document Object Model (DOM)

- Adding Javascript to Websites
 - Adding Inline Javascript (NOT GOOD PRACTICE. DON'T USE INLINE JS)
 - Inside the `<body>`, add an `onload=" <Javascript to execute when the page loads> "`
 - Example
 - `<body onload="alert('Hello!');">`

- Internal Javascript
 - Include a `<script>` inside of the html

```

8   <body>
9     <h1>Hello</h1>
10
11   <script type="text/javascript">
12     ?
13     alert("Hello");
14
15   </script>
16
17   </body>
18 </html>

```

- External Javascript
 - Include a `<script>` with a `src=" <Path to .js file> "`

```

<body>
  <h1>Hello</h1>

  <script src="index.js" charset="utf-8"></script>

</body>
</html>

```

- The placement of the `<script>` with the `.js` link matters!
 - Include CSS in the beginning, inside the `<head>`
 - Include JavaScript at the bottom, just before the closing `</body>` tag
- Intro to the Document Object Model
 - Catalogs a webpage into individual models/objects where we can alter/manipulate its behavior and style

- Document is the root of the webpage
 - document.firstChild;
 - HTML Root of web page
 - document.firstChild.firstChild;
 - <head>
 - document.firstChild.lastElementChild;
 - <body>
- Selecting HTML Elements with JavaScript
 - GetElementBy (Broad selector)
 - Get elements by tag name (li, ul, h1,)
 - document.getElementsByTagName(< tag >);
 - Get elements by class name
 - document.getElementsByClassName(< className >);
 - Get element by id
 - document.getElementById(< id >);
 - Query Selector (More complex and precise selector)
 - document.querySelector(< CSS Selector >);
 - Returns a singular html element
 - If the query selector obtains multiple elements, the querySelector only returns the first element
 - document.querySelectorAll(< CSS Selector >);
 - Returns a list of all the html elements that match the CSS Selector
- Manipulating and Changing Styles of HTML Elements with JavaScript
 - Properties may, and often are, different than the css property.
 - To set the value, have to include the property in “ “
 - Changing style
 - <selected>.style.<property> = “<new value>”;
 - Changing color
 - <selected>.style.color = “< color >”;
- Separation of Concerns: Structure vs Style vs Behavior
 - CSS files should only be in charge of styling a webpage.
 - So we shouldn't use DOM - Javascript to modify the style of a web page.
 - What we can do is, include a style modifier in the CSS file, and using DOM we can add a new class property for the .classList to add/modify styling of a html element.
 - .classList
 - Returns a list of the classes that are attached to the element
 - We can then add properties to the list, to further add new properties for a html element
 - .classList.add();

- Add a new CSS style property
 - .classList.remove();
 - Remove a CSS style property
 - .classList.toggle();
 - Toggle's a CSS style property on/off.
- Text Manipulation and the Text Content Property
 - <Selected>.textContent
 - Returns the actual text string value
 - <Selected>.innerHTML
 - Returns the html code
 - Can also modify the html
- Manipulating HTML Element Attributes
 - Attributes consist of things like, class=, href=, src=, ...
 - <Selected>.attributes
 - Returns the mapping of all the attributes
 - <Selected>.getAttribute(" < Attribute to get > ");
 - <Selected>.setAttribute(" < Which attribute to change> ", " < Value to change the attribute to > ");

● Advanced Javascript and DOM Manipulation

- Adding Event Listeners to a Button
 - <EventTarget>.addEventListener(<type>, <listener>);
 - <type> consists of a variety of different types to cause to execute the event listener
 - e.g. "click"
 - <listener> usually a javascript function
 - Can also be an anonymous function
- Higher Order Functions and Passing Functions as Arguments
- How to play sounds on a website
 - Create a new Audio(< File Path >);
 - Call the .play(); of the audio object
- Javascript Objects
 - Javascript objects have one or more fields : data
 - Example:
 - var anand = { name : "Anand", dob : "12/20/1993"}
 - Constructor
 - function <Name of Class>(< Arguments >) { <this.field = argument;>}
 - Use the 'new' keyword to create an object
- Switch Statements
 - switch (expression)
 - case < >:
 - Run the code for the case

- break
 - default:
 - "Else statement"
- Keyboard Event Listener
 - document.addEventListener('keypress', handler)
- Understanding Callbacks and How to Respond to Event Listener
 - Callback function
 - Function that gets passed in as an input

● jQuery

- What is jQuery?
 - The most popular JS library
 - Takes the traditional DOM manipulation methods, and makes it easier and efficient in less lines of code
 - Example:
 - Traditional
 - document.querySelector("h1")
 - JQuery
 - \$("h1")
- How to incorporate jQuery into Websites
 - Easiest way is to use a CDN method
 - CDN is basically a network of connected servers. CDN then finds the easiest and fastest method to retrieve files.
 - Place the jQuery CDN script above the javascript at directly above the ending body tag
 - `<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>`
- Selecting Elements with jQuery
 - jQuery(" < CSS Selector > ")
 - OR
 - \$(" < CSS Selector > ")
 - \$() == jQuery()
 - There is no difference between selecting all elements vs one element
- Manipulating Styles with jQuery
 - To set the style
 - <jQuery Selected>.css(< Attribute to change>, <What to change it to>)
 - Example
 - \$("h1").css("color", "red");
 - To get the style
 - <jQuery Selected>.css(< Attribute to get >);
 - Separate concerns

- jQuery is implemented in the .js file, and we can also use it to style the page
- But styling the page in a .js file is bad practice
- So the correct thing to do is create a css class inside the styles.css, and use jQuery to set the style to that class in the styles.css via .addClass()
-
- Add/Remove class
 - <jQuery Selected>.addClass(< CSS class to change style to, can be multiple classes which are separated by a single space>);
 - <jQuery Selected>.removeClass(< CSS class to change style to, can be multiple classes which are separated by a single space >);
- Check if an element has a class
 - <jQuery Selected>.hasClass(< class to check if exists >)
- Manipulating Text with jQuery
 - 2 Methods
 - 1. <jQuery Selected>.text(< New Text>);
 - 2. <jQuery Selected>.html(< New HTML >);
- Manipulating Attributes with jQuery
 - Set Attribute
 - <jQuery Selected>.attr(< Attribute to change>, < New Attribute Value>)
 - Get Attribute
 - <jQuery Selected>.attr(< Attribute to check>)
- Adding Event Listeners with jQuery
 - Click Listener
 - <jQuery Selected>.click(< Callback function>);
 - Key Press Listener
 - <jQuery Selected>.keypress(< Callback function>);
 - General/Flexible Method
 - <jQuery Selected>.on(< Event > ,< Callback function>);
- Adding and Removing Elements with jQuery
 - Adding an Element
 - Adding a new element before a selected element
 - <jQuery Selected>.before(< HTML of New Element >);
 - Adding a new element after a selected element
 - <jQuery Selected>.after(< HTML of New Element >);
 - Prepend a new element inside of a selected element

- <jQuery Selected>.prepend(< HTML of New Element >);
 - Append a new element inside of a selected element
 - <jQuery Selected>.append(< HTML of New Element >);
- Web Animations with jQuery
 - Hiding an element
 - <jQuery Selected>.hide();
 - Element gets removed from the flow of the HTML structure of the web page
 - Show an element
 - <jQuery Selected>.show();
 - Toggle hide and show
 - <jQuery Selected>.toggle();
 - FadeOut
 - Hides the element but in a less sudden way
 - <jQuery Selected>.fadeOut();
 - FadeIn
 - Shows the element but in a less sudden way
 - <jQuery Selected>.fadeIn();
 - Toggle fade in and fade out
 - <jQuery Selected>.fadeToggle();
 - Slide Up
 - Collapses an element
 - <jQuery Selected>.slideUp();
 - Slide Down
 - Uncollapse an element
 - <jQuery Selected>.slideDown();
 - Toggle Slide In and Slide Out
 - <jQuery Selected>.slideToggle();
 - Method for more Control Over Animations
 - <jQuery Selected>.animate({ < New CSS Rule to Animate>});
 - Can only add CSS rules that have a numeric value
 - Can also chain animation methods back to back
 - Example:
 - \$("h1").slideUp().slideDown().animate({opacity: 0.5});

● Node.js

- What is Node.js?
 - Allows the creation of a backend using Javascript
 - Fast execution on the backend

- Allows Javascript to be run outside of the browser, independently, and to allow Javascript code to interact directly with the computer, that will allow it to access files, listen to network requests, ...
 - Allows the creation of full desktop applications using Javascript
 - Allows the execution of Javascript programs on the server
 - All of the heavy calculations and executions happen on the server, rather than the client's browser
- The Node REPL (Read Evaluation Print Loops)
- How to Use the Native Node Modules
 - Check out the different types of API's on nodejs.org/API/
 - Accessing File Systems
 - Create a file system object
 - Example:
 - `const fs = require("fs");`
 - `require(<">);`
 - Not part of standard Javascript
 - In NodeJs, a built in function to load modules into the javascript code
- The NPM Package Manager and Installing External Node Modules
 - NPM (Node Package Manager)
 - Package managers for external modules
 - Gets bundled with Node
 - Creating your own Package
 - run `node init` and follow the steps
 - It will create a `package.json` file
 - Downloading an external node module
 - Find the module you want from searching from npmjs.com
 - Check the usage, usually it is in the form of `npm install <keyword>`

● Express.js

- What is Express?
 - A node framework, that was built to make programmers write less repetitive code
 - Built specifically for web developers
- Creating Our First Server with Express
 - `npm init`
 - `npm install express`
 - In the `server.js` File
 - `//jshint esversion:6`
 - Place this comment above to remove warnings
 - `const express = require('express');`
 - Function that represents the express module, and gets binded to the variable `app`

- `const app = express();`
 - Listen to a specific port
 - `app.listen(3000);`
- Handling Requests and Responses: the GET Request
 - `localhost:3000`
 - Equivalent to the root of a home page
 - Root Page
 - Browser sends a GET request to the server
 - `app.get(<"Location">, <Call back function, tells the server what to do when that request happens>)`
 - Example
 - `app.get("/", function(req, res) {});`
- Understanding and Working with Routes
 - `app.get(<Route>, <Responding Callback>)`
 - Specifies the route that we are going to respond to
 - When a route is encountered, the `get()` is executed
 - <Route>
 - Path
 - Example:
 - Root Path
 - `"/"`
 - <Responding Callback>
 - Responds for the specifies Route
 - Example:
 - `function(req, res) {`
 - `res.send();`
 - Nodemon.io
 - Will monitor for changes in your js code and will automatically change your code
 - Activate nodemon
 - `nodemon <.js file>`
 - Responding to Requests with HTML Files
 - `res.sendFile()`
 - `__dirname`
 - Gives the current file path
 - Usage:
 - `__dirname + "<name of html file>"`
 - Processing POST Requests with Body Parser
 - <form> has an action and a method
 - method
 - POST, so we are sending data somewhere
 - action

- Path/location to interact with (Like sending data via a POST)
- app.post(<Route>, <Callback Method>)
 - Specifies that when there is a POST request for the route, the callback method will be executed
- body-parser
 - Used to parse incoming POST data
 - To install, call the command "npm install body-parser"
 - In the JS file, include
 - const bodyParser = require('body-parser');
 - app.use(bodyParser);
 - body-parser Modes
 - app.use(bodyParser.<Mode>)
 - bodyParser.text()
 - Parse all the requests into text/string
 - bodyParser.json()
 - bodyParser.urlencoded()
 - Used when parsing data that is send via a HTML form
 - bodyParser.urlencoded({extended: true})
 - Allows us to POST nested objects
- Javascript Starter Template

```

1  //jshint esversion: 6
2
3  const express = require("express");
4  const bodyParser = require("body-parser");
5
6  const app = express();
7
8  app.use(bodyParser.urlencoded({extended: true}));

```

• Application Programming Interface

- API
 - Interface layer to interact with an external system.
 - The interactions are via the form of GET and POST requests
- Using the Request Module to Get Data from an API
 - request - NPM module
 - Setup
 - const request = require("request");
 - Usage

- request(<URL to Request To>, <Callback>);
 - <Callback>
 - function(error, response, body);
- Understanding the JSON format and Working with JSON
 - JSON
 - JavaScript Object Notation
 - Used to communicate data back and forth in the web
 - XML
 - eXtensive Markup Language
 - Alternative to JSON, came before JSON.
 - Less frequently used now
 - JavaScript object -> string
 - JSON.stringify(<JavaScript Object>);
 - string -> JavaScript Object
 - JSON.parse(<JSON String Equivelant>);
 - Sending more than one thing
 - res.write(<HTML to Respond with>);
 - res.send();

```
res.write("The current data is " + currentDate);
res.write("<h1>The price of " + req.body.cryptoSelection + " is " + price + " " + req.body.currencySelection + "</h1>");
res.send();
```

- API Calls with Parameters

```
var options = {
  url: "https://api2.bitcoinaverage.com/convert/global",
  method: "GET",
  qs: {
    from: crypto,
    to: fiat,
    amount: amount
  }
};

request(options, function(error, response, body){

  var data = JSON.parse(body);
  var price = data.price;

  console.log(price);

  var currentDate = data.time;

  res.write("<p>The current date is " + currentDate + "</p>");

  res.write("<h1>" + amount + crypto + " is currently | " + price + fiat + "</h1>");

  res.send();

});
```

- Newsletter Walkthrough
 - Incorporated and modified an existing Bootstrap Sign In Example
 - In the HTML, some files are directly obtained from the internet via the use of the web urls
 - However, some are static files such as css and images files.
 - In order to use them, we must notify Express that they are static
 - To do this, we can create a folder where all the static files will be contained,
 - And then call the `express.static()`
 - ```
app.use(express.static("public"));
```
  - In the form, be sure to specify action and method
    - ```
<body class="text-center">  
  <form class="form-signin" method="POST" action="/">
```
 - So the picture above, the form will POST to the location in action
- Posting Data to Mailchimp Servers
 - Set up MailChimp Account & Obtain API Key
 - Set up the request url and headers
 - Authentication
 - Look through the API documentation on authentication

```

app.post("/", function(req, res) {
  var firstName = req.body.fName;
  var lastName = req.body.lName;
  var email = req.body.email;

  var data = {
    'members': [
      {
        email_address: email,
        status: 'subscribed',
        merge_fields: {
          FNAME: firstName,
          LNAME: lastName
        }
      }
    ],
  }

  var jsonData = JSON.stringify(data)

  console.log(firstName, lastName, email);

  var options = {
    url: 'https://us3.api.mailchimp.com/3.0/lists/c88fb2ef24',
    method: 'POST',
    headers: {
      'Authorization': "anand1 8a8caf31357e4496e62d3e3690b8797b-us3"
    },
    body: jsonData
  }

  request(options, function(error, response, body) {
    if (error) {
      console.log(error);
    } else {
      console.log(response.statusCode);
    }
  });
});

```

-
- Adding Success/Failure Pages to the Website
 - Instead of sending text back upon success or failure. We now send back an individual html file
 - Redirecting back to a specific page
 - Method: Create a button, when pressed it will create a post request.
 - In the server, handle the post request to the location, by redirecting to the intended page
- Deploy the project to Heroku and Make it Live
 - What is Heroku?

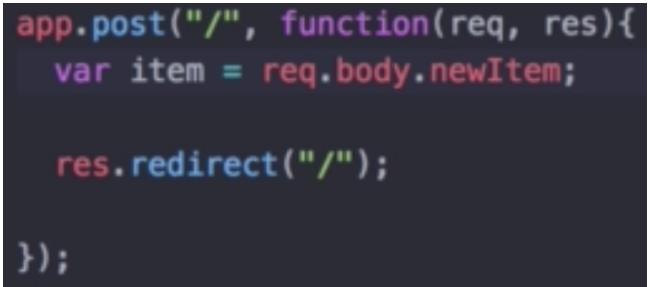
- Heroku allows users to deploy their websites on Heroku's servers, so that anyone can access the website.
 - To use Heroku:
 - 1. Modify the prior `app.listen(3000, <callback>);` to `app.listen(process.env.PORT || 3000, <callback>);`

```
app.listen(process.env.PORT || 3000, function() {
```

 - `process.env.PORT` - Heroku will handle the setting of the port
 - `|| 3000` - This will be used in the case that Heroku is not being used - 2. Define a Procfile
 - Create a file named Procfile that is in the same folder as the main root of the website
 - Inside, write the same command as you would write in the terminal to execute your website
 - `web: node app.js`
 - 3. Initialize and Commit the web project using git
 - In the web project root directory, execute `git init`
 - Add the files, `git add .`
 - Commit the files, `git commit -m "Commit message"`
 - 4. Create a Heroku Reference to your Web Project, Git
 - Execute, **heroku create**
 - By default, heroku creates a random name to your project
 - But you can pass a parameter to set the name
 - Execute, **git push heroku master**
 - Web project is now deployed on Heroku
 - 5. Running the web site on Heroku
 - 1. Can run **heroku open** on the root directory of the web project
 - 2. Can run the heroku url given after heroku create
- Checking Heroku logs
 - Run **heroku logs**

• EJS

- What is EJS?
 - Templating language that lets you generate HTML markup with plain JavaScript
 - Useful when you want to use a single HTML page, but with different contents/data based on conditions
- Creating your first EJS Template
 - `npm install ejs`

- In .js,
 - `app.set('view engine', 'ejs');`
- Create a new folder called 'views'
 - Add a file with .ejs
 - .ejs file follows HTML format, but has ejs variable placeholders
- Placeholder marker
 - `<%= VARIABLE NAME %>`
- To display a ejs webpage,
 - `res.render(<EJS file without the .ejs, <JSON object, where the key is the ejs variable and the value is the thing to display>>);`
- Running Code Inside the EJS Template
 - We can add JavaScript code in the EJS file.
 - Each line of the JavaScript needs to have an EJS scriptlet of `<% JAVASCRIPT CODE %>`
 - Use scriptlet only for control-flow, but only in select situations.
 - Most logic needs to reside in the server code
- Passing Data from Your Webpage to Your Server
 - 1. This will be done via a POST request from the web page.
 - 2. Server, let's say app.js, will then handle the POST request.
 - 3. It can use body-parser, to then obtain the data being send, and work with the data
 - `res.redirect(<EJS file to redirect to, app.get() will be invoked>)`
- 

```

app.post("/", function(req, res){
  var item = req.body.newItem;

  res.redirect("/");
});

```
- The Concept of Scope in the Context of Javascript
 - Variables inside of a function, have local scope to that function
 - Variables outside of a function, have global scope
 - If/Else or For/While
 - Variable created will have global scope
 - let will have local scope
 - const will have local scope
 - **Use let or const whenever you can, over var**
- Adding Pre-Made CSS Stylesheets to your Website
 - All files that will be used must be served up in express app.js, these are static files
 - Static files are files that clients download as they are from the server.

```

4  const app = express();
5
6  var listItems = [];
7
8  app.set('view engine', 'ejs');
9
10 app.use(bodyParser.urlencoded({extended: true}));
11 app.use(express.static("public"));

```

- Understanding Templating vs Layouts

- Templating

- Used mainly when you want to reuse the structure and format of a web page, but changing the data within them

- Layouts

- When you want to reuse certain parts of a webpage, like the headers, footers, and styling, in many different webpages
 - But overall, the body structure differs
 - Use `<%- include(<EJS name without .ejs>) -%>` for reusing parts of the webpage

- Reused component pages are called **partials**

```

1
2 <%- include("header") -%>
3
4 <body>
5
6   <div class="box" id="heading">
7     <h1>
8       <%= listTitle %>
9     </h1>
10   </div>
11
12   <div class="box">
13     <% for(var i = 0; i < listItems.length; i++){ %>
14     <div class="item">
15       <input type="checkbox">
16       <p>
17         <%= listItems[i] %>
18       </p>
19     </div>
20     <% } %>
21
22     <form class="item" action="/" method="post">
23       <input type="text" name="newTodo" placeholder="New Item" autocomplete="off">
24       <button type="submit" name="listSubmit" value=<%= listTitle %>></button>
25     </form>
26   </div>
27
28 </body>
29
30 <%- include("footer") -%>
31

```

- Understanding Node Module Exports: How to Pass Functions and Data Between Files

- Using functions between files:

- Node.js contains an object called **module**
 - If you want to create functions in a class that you want it to be reusable in other files:

- You can set **module.exports**.<Function Reference Name> = <name of the function that you want to reuse>
 - **module.exports** can also be shortened down, and **exports** can be used
- To use the function:
 - Create a reference variable, and link it to the file where the reusable functions reside
 - Be sure to use `__dirname` + if it's custom, because we didn't install using npm
 - ```
const date = require(__dirname + "/date.js");
```
  - Then invoke the function using <reference variable>.<name of reusable function>
    - ```
var day = date.getDate();
```
- Blog Website Challenge
 - Usually, the npm packages that are being used are gitignored, and when a new person works with the project they would then initially go ahead and run npm install
 - Express Parameter Routing
 - Lodash

● Databases

- SQL vs NoSQL
 - SQL
 - Structured Query Language
 - Most Popular
 - MySQL
 - PostgreSQL
 - NoSQL
 - Not Only Structured Query Language
 - Most Popular
 - MongoDB
 - Redis
 - Differences
 - 1. Structure
 - SQL
 - Table Format, kind of like a Excel Sheet
 - Missing cells, will contain Null
 - Consistent Structure
 - NoSQL
 - JSON Object structure
 - Flexible Structure

- 2. Relationships
 - SQL
 - Relational
 - Good for many-to-many relationships
 - Can have multiple tables, and we can link a relationship between them
 - Will lead to less repetitive data
 - NoSQL
 - Non-Relational
 - Can lead to repetitive data
 - Good for 1-to-many type relationships
- 3. Scalability
 - SQL
 - Becomes slower with more rows of data
 - NoSQL
 - Better for scalability

• SQL

- CREATE Table and INSERT Data
 - Create Table
 - CREATE TABLE <table name> (<column name> <data type>, ... PRIMARY KEY (<column name to be the primary key>));
 - Data Types
 - INT, String, Money, char, ...
 - Insert Data
 - INSERT INTO <table name> VALUES (value1, value2, value3, ...);
- READ, SELECT, and WHERE
 - Selecting specific columns
 - SELECT <column(s)> FROM <Table>;
 - Selecting specific rows
 - SELECT <column(s)> FROM <Table> WHERE <condition>;
- Updating Single Values and Adding Columns in SQL
 - UPDATE <table> SET <column> = <New Cell Value> WHERE <column> = <condition>;
 - Adding a new column
 - ALTER TABLE (Add, Delete, or Modify Columns in a Table)
 - ALTER TABLE <table name> ADD <new column name> <data type>;
- DELETE
 - DELETE FROM <table> WHERE <condition>;
- Relationships, Foreign Keys, and Inner Joins
 - Foreign Key
 - Links between table

- Inner Join

- MongoDB

- Install MongoDB on Mac
 - Follow directions on Mongoddb
- MongoDB CRUD Operations
 - Create
 - Create a database
 - terminal: use <name of database>
 - Create a collection / Insert new items into a collection
 - db.<name of new collection>.insertOne()
 - db.<name of new collection>.insertMany()
 - If the collection doesn't exist, then a new one will be created
 - Read / Query
 - terminal: show collections
 - Shows all the collections
 - db.<name of collection>.find()
 - Displays all the data in the collection
 - db.<name of collection>.find(<Query>, <Projection>)
 - Displays all the data in the collection that meets the query criteria
 - Example of Query:
 - {name : "Pencil"}
 - {price: {\$gt: 1}}
 - Example of Projection:
 - {name : 1, address: 1}
 - 1 means true
 - 0 means false
 - We can dictate which fields we want back from the query
 - So in this case, where the data has a name and an address
 - Update
 - db.<collection>.updateOne(<Query>, <Update To>)
 - <Query>
 - The data to update
 - <Update To>
 - {\$set: {<field>: <new value>}}
 - Delete
 - db.<collection>.deleteOne(<Query Criteria>)
 - db.<collection>.deleteMany()
- Relationships in MongoDB

- One to Many Relationships are great for NoSQL - MongoDB
 - Example:
 - Product
 - Review 1
 - Review 2
 - Review 3 ...
 - Mongo shell
 - terminal: mongod
 - Starts up the mongo server
 - When you see “waiting for connections on port 27017” then the server has started
 - terminal: mongo
 - Starts up the mongo shell
 - Troubleshoot:
 - sudo killall -15 mongod
 - If unable to start up the server using mongod
 - MongoDB with Node.JS
 - 2 ways to incorporate MongoDB app with Node.js
 - 1. Native MongoDB Driver
 - Install a MongoDB driver from the documentation
 - One that corresponds to the programming language
 - In this case, Node.js MongoDB driver
 - npm install mongodb
 - 2. mongoose
 - ODM
 - Object Data Modeling library
 - Most popular package to work with MongoDB and Node.JS
- mongoose
 - Terminology
 - Collection
 - Grouping of MongoDB documents
 - Equivalent to a Table in SQL
 - Document
 - A single { <key : value>, ... }
 - Which can consist of 1 or many key : value pairs
 - Intro to Mongoose
 - Allow node.js app to talk with MongoDB
 - Allows the user to define the schema for the documents in a particular collection
 - npm install mongoose
 - Create reference to mongoose

- `const mongoose = require('mongoose');`
- Create new database, or connect to an existing database

```
const mongoose = require('mongoose');  
  
// URL: <server>/<Database, if it doesn't exist a new one will be created>  
mongoose.connect("mongodb://localhost:27017/fruitsDB", { useNewUrlParser: true});
```

-
- Closing connection
 - `mongoose.connection.close();`
 - Create a mongoose scheme

```
const fruitSchema = new mongoose.Schema ({  
  name: String,  
  rating: Number,  
  review: String  
});
```

-
- Use the scheme to create a Mongoose model

```
// First Parm: String, singular, name of collection,  
// in MongoDB it will be saved all lower case  
// Second Parm: Scheme  
const Fruit = mongoose.model("Fruit", fruitSchema);
```

-
- Adding a new item to the collection
 - Create a new object based on the model
 - Then call `.save()` on that new object

```
const fruit = new Fruit({  
  name: "Apple",  
  rating: 7,  
  review: "Pretty solid as a fruit."  
});  
  
fruit.save();
```

- Saving in bulk
 - `<Model>.insertMany(<Array of objects>, <Callback>)`

```

const banana = new Fruit({
  name: "Banana",
  rating: 6,
  review: "Weird texture."
});

const kiwi = new Fruit({
  name: "Kiwi",
  rating: 4,
  review: "Too sweet for me."
});

const orange = new Fruit({
  name: "Orange",
  rating: 8,
  review: "Tastes great."
});

Fruit.insertMany([banana, kiwi, orange], function(err){
  if(err){
    console.log(err);
  }else{
    console.log("Successfully saved all the fruits");
  }
});

```

- Reading from your database with Mongoose
 - <Model>.find(<Callback(err, objects if successful)>);

```

Fruit.find(function(err, fruits){
  if(err){
    console.log(err);
  }else{
    fruits.forEach(function(fruit){
      console.log(fruit.name);
    });
  }
});

```

- Data Validation with Mongoose
 - Add validation in the scheme object
 - Check for the different types of validations in the mongoose documentations


```
const fruitSchema = new mongoose.Schema ({
  name: String,
  rating: {
    type: Number,
    min: 1,
    max: 10
  },
  review: String
});
```

-
- Updating and Deleting Data with Mongoose
 - Update
 - <Model>.updateOne(<Query, item to update>, <What fields to update?>, Callback);
 - Delete
 - <Model>.deleteOne(<Query, item to delete>, Callback);
- Establishing Relationships and Embedding Documents using Mongoose
 - MongoDB - NoSQL is great for 1-to-Many relationships
 - So for establishing relationships, we modify the schema:
 - In the “1” schema, place in the key:value pair, the “many schema”

```
const personSchema = new mongoose.Schema({
  name: String,
  age: Number,
  favoriteFruit: fruitSchema
});
```

-
- Creating Custom Lists using Express Route Parameters
 - Creating and responding to a dynamic route
 - Express Route Parameters

Express Route Parameters

```
app.get("/category/:<paramName>", function(req, res){
  //Access req.params.<paramName>
});
```

Mongoose findOneAndUpdate()

```
<modelName>.findOneAndUpdate(  
  {conditions},  
  {$pull: {field: {query}}},  
  function(err, results){}  
);
```

-
- \$pull
 - Removes from an existing array all instances of a value or values that matches a specific condition

● Deploying your Web Application

- How to Deploy Apps with a Database
 - We can't simply place the app w/database on Heroku, as the app and database will be separated.
 - We need to place both the app and the database on servers
 - **MongoDB Atlas**
 - Server that hosts the database
- How to setup MongoDB Atlas
 - 1. Setup an account with Atlas
 - 2. Create a cluster
 - 3. Setup security access
 - 4. Cluster > Connect > Connect to an application > Use the url obtained to connect with Node.js application
- Deploying app with a database to Heroku

● Build Your Own RESTful API From Scratch

- What is REST?
 - REpresentational State Transfer
 - Client - Server Architecture
 - Client (analogy of a customer) makes a request to a Server
 - Server (analogy of a waiter) responds to the request
 - Sent through (analogy of a language):
 - HTTP Request
 - FTP Request
 - HTTPS Request
 - Secured
 - API (analogy of a menu)

- Set of services that the client can interact with the server
- REST is a architectural style of designing API's
- Rules to be RESTful
 - HTTP Request Verbs
 - GET
 - Read
 - PUT
 - Full Update
 - PATCH
 - Partial Update
 - POST
 - Create
 - DELETE
 - Delete
 - Using specific pattern of routes/endpoint URLs
 - `/ {routes}`
- Creating a Database with Robo 3T
 - Robo 3T
 - GUI for MongoDB
 - By default Robo 3T will connect to localhost:27017
 - First, we must startup the mongod server, by running "mongod" in the terminal
- Set up Server Challenge
- GET all articles
- POST an article
 - Install Postman
 - Postman allows us to test out our APIs without creating the additional components needed
- Chained Route Handlers Using Express
 - `app.route()`
 - `get`
 - `post`
 - `delete`
 - `...`
 - Using Express Route can reduce redundancy
- GET a specific article
- PUT a specific article

```

<modelName>.update(
    {conditions},
    {updates},
    {overwrite: true}
    function(err, results){}
);

```

- PATCH a specific article
- DELETE a specific article

● Authentication & Security

- Level 1 - Register Users with Username and Password
 - Setup mongoose and mongoDB
 - npm i mongoose
 - const mongoose = require("mongoose");
 - mongoose.connect("mongodb://localhost:27017/credentials");
 - Run mongod
 - terminal: mongod
 - By default it listens to port 27017
 - 1. Create schema
 - const userSchema = { ... };
 - 2. Create model based on the schema
 - const User = new mongoose.model("User (singular)", userSchema);
 - Set up post to /register
- Level 2 - Database Encryption
 - Encryption
 - The process of encoding and decoding messages
 - Cipher
 - An algorithm to performing encryptions and decryptions
 - Install and use mongoose-encryption

- 1. npm i mongoose-encryption
- 2. const encrypt = require('mongoose-encryption');
- 3. Schema should be created used new mongoose.Schema
- 4. Two methods of generating keys
 - Method 1
- 5. Modifying to **only encrypt certain fields**, in this case only passwords


```
userSchema.plugin(encrypt, {secret : secret, encryptedFields: ['password']});
```
- Encryption occurs when 'save'
- Decryption occurs when 'find'
- Using Environment Variables to Keep Secrets Safe
 - What are environment variables and how does it keep secrets safe?
 - Basically a file where secret keys are contained in
 - **dotenv**
 - Very popular npm package that is used for working with environment variables
 - Installation & Usage
 - 1. npm i dotenv
 - 2. In .js file:
 - Place require('dotenv').config() at the very top of the js file
 - 3. Create a ".env" file in the root of the directory of project
 - 4. Add environment variables to the .env file
 - Format:
 - NAME=VALUE
 - 5. Accessing environment value
 - process.env.<key name>
 - 6. Place the .env file into a .gitignore
- Level 3 - Hashing Passwords
 - We won't need an encryption key for hashing
 - How it works
 - Password -> <Hash Function> -> Hash
 - Store the Hash in the database
 - This makes it so that we do not have to store the user's password
 - Hash Functions are 1-way function
 - Easy to "encrypt" a password, but almost mathematically impossible to go back and be able to find the password
 - md5
 - npm package for Hashing

- Hacking 101
 - 1. Hash of the password is obtained
 - 2. Reverse hash table (pre-build hashtables) can be generated and a powerful computer can be used to hack the password
 - 3. Dictionary Attack
- Level 4 - Salting and Hashing Passwords with bcrypt
 - Prevents dictionary attacks or hash table attacks
 - Salting
 - Password + Salt (Random set of characters) are given to the Hash Function to generate the Hash
 - Salt value is stored in the database, along with the hash
 - bcrypt
 - Industry standard hashing algorithms that developer's use
 - Makes it much slower to generate the hashes
 - Only works for specific and stable versions of node.js
 - Also uses salt rounds
 - Salt Rounds
 - Number of rounds/iterations to salt the password
 - More rounds -> More secure it is
 - nvm
 - Node version manager
 - Allows using switching between multiple versions of node
- What are cookies and sessions?
 - Cookies and Sessions store information from browser interaction
 - Cookies
 - Stored on the browser
 - Session
 - Stored on the browser, as well as the server
 - Cookies are created in response to a POST request
 - Cookies are saved, and can be retrieved in the GET request
 - Session
 - Period of time that a browser interacts with a server
 - Cookies are used to maintain a session
 - Maintains authentication until logging out, which is when the cookie and thus session ends
- Using passport.js to Add Cookies and Sessions
 - npm install:
 - passport
 - passport-local
 - passport-local-mongoose
 - express-session
- Level 6 - OAuth 2.0
 - Open standard for token based authorization

- Delegate security to larger companies (Facebook, LinkedIn)
- Why Oath?
 - 1. Grant granular level of access
 - Can request specific data accordingly
 - 2. Read Only or Read + Write Access
 - 3. Revoke Access
- How to setup Oath?
 - 1. Setup App with 3rd party (Facebook, Google, LinkedIn)
 - Returns an appId or clientId
 - 2. Redirect to Authenticate
 - 3. User logs in
 - 4. User grants permissions
 - 5. Receive authorization code from (Facebook)
 - 6. Exchange authentication code for access token
 - Receive access token and save into our database
- Auth code vs Access Token
 - Auth code - works one time
 - Access Token - works multiple times

• React.js

- What is React?
 - Front-end javascript library that makes it easier to make user interfaces
 - Very popular in 2019
 - Benefits
 - Re-usability
 - Organization
 - Concise code
 - Scales well
 - Components
 - Contains its own html, css, js
 - Can update itself without affecting other components
- Intro to Code Sandbox and Structure to the Module
 - Code Sandbox is a nice to use IDE on the browser that contains all the setup needed for coding
- Intro to JSX and Babel
 - Every react html has a <div id="root">

```

1  var React = require("react");
2  var ReactDOM = require("react-dom");
3
4  // .render(<>):
5  // 1. What to Show
6  // 2. Where to Show inside of
7  // 3. Callback, when the render function has been completed
8  ReactDOM.render(<h1>Hello World</h1>, document.getElementById("root"))
  
```

https://h7y...
Hello World

- <ReactDOM>.render()

- Inserts HTML into the website, through the javascript file/code
- Arguments
 - 1. What to show
 - Note: only takes 1 element
 - 2. Where to show
 - In this case, document.getElementById('root')
 - 3. Callback
- React creates JSX files
 - HTML code inside of a Javascript file
- Babel
 - Javascript compiler inside of React module
 - Takes any Javascript code and converts it into an equivalent Javascript code that can be read by any browser
- Javascript Expressions in JSX & ES6 Template Literals
 - We can insert Javascript inside an HTML inside of a javascript
 - {<Javascript code>}
 - Only values or expressions can be placed, not statements
- JSX Attributes & Styling React Elements
 - In the JSX file, instead of using "class=", we would have to use "className="
 - HTML attributes for JSX must be camelCase
- Inline Styling for React Elements
 - Even though it looks like HTML, it is not, it's JSX
 - For inline styling, we need to pass in a javascript object
- React Components
 - Components can help us break down a large codebase into smaller ones, plus the benefit of reuseability
 - To create a React Component, create a function that has the naming convention in Pascal form
 - Create a new file with .jsx extension

```

1  import React from "react";
2
3  function Heading() {
4    return <h1>My Favourite Foods</h1>;
5  }
6
7  export default Heading;

```

- In the index.js file, import the .jsx file and use the component like how you would use an HTML element


```

3   import Heading from "../Heading.jsx";
4
5   ReactDOM.render(
6     <div>
7       <Heading />

```

- Normally, React app's in their index.js have an App component, which comprises of all of the UI
- All the components will have their own .jsx file
- And all the components will be organized into sub-directories
- Javascript ES6 - Import, Export and Modules
 - export default <reference>
 - There can only be 1 default export for a .jsx file
 - Export multiple functions, values, objects,
 - export {<ref 1, ref 2, ...>}
 - import * as <ref> from "<.jsx file>"
 - wildcard import is discouraged, as it reduces readability
- Starting up a React app
- React Props
 - Is a method to pass in parameters from JSX html to the React functions
 - We can also pass in functions as well

```

function Card(props) {
  return <div>
    <h2>{props.name}</h2>
    <img
      src={props.imageUrl}
      alt="avatar_img"
    />
    <p>{props.tel}</p>
    <p>{props.email}</p>
  </div>;
}

ReactDOM.render(
  <div>
    <h1>My Contacts</h1>
    <Card name="Beyonce" imageUrl="https://blackhistorywall.files.wordpress.com/2010/02/picture-device-independent-b
    <Card name="Jack Bauer" imageUrl="https://pbs.twimg.com/profile_images/625247595825246208/X3XLea04_400x400.jpg"

```

- - name, imageUrl, tel, email are like parameters that get passed to Card, with all the parameters inside of props
- Props
 - Creates a new javascript object with parameters
- React Dev Tools
 - Displays React DOM Tree
 - Install extension: React Developer Tools
- Mapping Data to Components
 - .map()
 - "key" - unique key value for props to distinguish array values
 - Only used by the backend in React

- Javascript ES6 Map/Filter/Reduce

- Map

- `<Array>.map(<Function to perform on individual items in the array>)`
- Creates a new array by performing function on items in the original array

```
//Map - Create a new array by doing something with each item in an array.  
const numMap = numbers.map(function(number) {  
  return number * 2;  
});
```

- Filter

- `<Array>.filter(<Function conditional that takes in item, and evaluates true or false>)`
- Creates a new array by keeping the items that evaluate to true

```
//Filter - Create a new array by keeping the items that return true.  
const numFilter = numbers.filter(function(num) {  
  if (num % 2 === 0) {  
    return true;  
  } else {  
    return false;  
  }  
});
```

- Reduce

- `<Array>.reduce(<Function(accumulator, currentValue) to perform on each item in the array>)`
- Creates an accumulated value, based on operation to each item in the original array

```
//Reduce - Accumulate a value by doing something to each item in an array.  
const numReduce = numbers.reduce(function(accumulator, currentNumber) {  
  return accumulator + currentNumber;  
});
```

- Find

- `<Array>.find(<Function>)`
- Finds and returns the first item that matches the condition

```
//Find - find the first item that matches from an array.  
const numFind = numbers.find(function(num) {  
  return num > 10;  
});
```

- FindIndex

- `<Array>.findIndex(<Function>)`
- Finds and returns the first item index that matches the condition

```
//FindIndex - find the index of the first item that matches.
const numFindIndex = numbers.findIndex(function(num) {
  return num > 10;
});
```

■ Javascript ES6 Arrow Functions (Fat Arrow)

- Shorter way of writing a javascript function
 - Can omit the keyword function
- Used for anonymous functions
- 1 parameter (no need for parentheses)
 - <parm> => {function body}
- 2 or more parameters (need parantheses)
 - (<parm1, parm2> => {function body})
- If function body is only 1 line, then we can omit the {} and "return"

```
var numbers = [3, 56, 2, 48, 5];
```

```
const newNumbers = numbers.map( x => x * x);
```

■ React Conditional Rendering with Ternary Operations & AND

- Ternary Operator

```
CONDITION ? DO IF TRUE : DO IF FALSE
```

- Ternary Operator is treated as an expression, so it can be placed inside the JSX file inside with HTML-looking code
- In Javascript, we can do a similar method to ternary operator by using && because if the first condition in && is true, the second condition will be executed

```
currentTime > 12 && <h1>Why are you still working?</h1>
```

- If currentTime is greater than true, the <h1> will be called

■ State in React - Declarative vs Imperative Programming

- Declarative
 - React uses
 - UI is dependent on a conditions of a state variable
 - Given <statement>, the <view> should be ...
 - Benefits
 - Easier to reason about
 - Fewer things can go wrong
 - Cons
 - Needs to re-do a lot of work all the time

- Expensive
 - Can create brand new ui elements and rerendering
 - Imperative
 - Getting hold of an item and setting it's attribute to a new value
- React Hooks - useState
 - Adds dynamic interactivity
 - Must use a hook inside a component
 - Each state is local to a component
 - useState(<parm>)
 - parm - Starting state value
 - Returns an array with 2 items
 - 1. State value
 - 2.
 - Benefits
 - Will only re-render the ui element that is needed
- Javascript Object & Array Destructuring
 - Data

```
const cars = [
  {
    model: "Honda Civic",
    //The top colour refers to the first item in the array below:
    //I.e. hondaTopColour = "black"
    coloursByPopularity: ["black", "silver"],
    speedStats: {
      topSpeed: 140,
      zeroToSixty: 8.5
    }
  },
  {
    model: "Tesla Model 3",
    coloursByPopularity: ["red", "white"],
    speedStats: {
      topSpeed: 150,
```

- Destructuring complex data

```

6  const [honda, tesla] = cars;
7  const {
8    speedStats: { topSpeed: teslaTopSpeed }
9  } = tesla;
10 const {
11   speedStats: { topSpeed: hondaTopSpeed }
12 } = honda;
13 const {
14   coloursByPopularity: [teslaTopColour]
15 } = tesla;
16 const {
17   coloursByPopularity: [hondaTopColour]
18 } = honda;
19

```

Browser Tests

< > ↻ https://mu634.csb.app/

Brand	Top Speed	
Tesla Model 3	150	red
Honda Civic	140	black

■ Event Handling in React

● HTML Event Attributes

- https://www.w3schools.com/html/html_attributes.asp
- Event Attributes can add more event handling types, such as onmouseout, onclick, ...

■ React Forms

- <input onChange=>
- Controlled Components
 - <https://reactjs.org/docs/forms.html#controlled-components>
- <form> when submitted, will refresh to make a POST or a GET request
 - Will call method called onSubmit
- event.preventDefault()
 - Prevents the default next step of the event
 - 1. Override <form onSubmit=>

```

<form onSubmit={handleClick}>
  <input
    onChange={handleChange}
    type="text"
    placeholder="What's your name?"
    value={name}
  />
  <button type="submit">Submit</button>
</form>

```

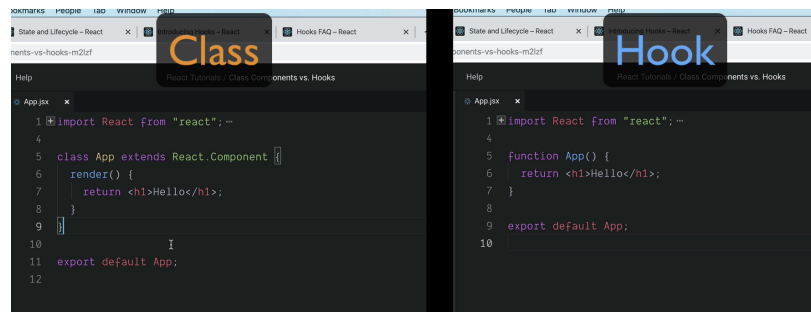
-
- 2. Add preventDefault

```

function handleClick(event) {
  setHeading(name);
  event.preventDefault();
}

```

- Class Components vs Functional Components
 - Hooks vs Classes



-
- Class
 - class App extends React.Component { ... render() { ... } }
- Hook
 - function App() { ... }
 - Can only use for Components
 - Makes it easier to maintain state

- Changing Complex State
 - We can have useState store an object to initialize
 - In the update function of useState()

- We can pass in an anonymous function, which has the previous value as the parameter
- JavaScript ES6 Spread Operator
 - ...
 - Can work with arrays and object inserts
 - Gives easy functionality to insert arrays and objects into other arrays and objects
- Managing a Component Tree
 - Can pass in function pointers in props from parent components to child components

● Design School 101

- Understanding Color Palette
 - Colors Invoke Emotion
 - Red
 - Exciting, Intensity, Love, Fast
 - Yellow
 - Attention grabbing, Joy
 - Green
 - Freshness, Safety, Growth
 - Blue
 - Stability, Trust, Serenity
 - Purple
 - Wealth. femininity

● Angela AMA

- Always be learning
 - New technologies will constantly come up, you should be constantly learning so you don't stagnate
 - Spend 1 hour everyday to learn
 - You learn the most from being out of your comfort zone
- Be able to figure out and research things on your own