Typical Application Tasks

Module 2: Creating and Styling HTML Pages	Re-factor to HTML5 DOM: Document Object Model from previous versions
~~Module 3: Introduction to JavaScript~~	
Module 4: ~~Creating~~ Forms ~~to~~ Collect ~~and~~ Validate User Input	Capture Forms - Form Validation code - JavaScript Client Side - C# Server Side
Module 5: Communicating with a Remote Server	Fetch API vs. XmlHttpRequest
Module 6: Styling HTML5 ~~by Using CSS3~~	CSS3: Add Style, Position, Layout models
Module 7: Creating Objects and Methods by Using JavaScript	OOP: Object Oriented Programming - Logic - Proprietary Code
Module 8: ~~Creating~~ Interactive Pages ~~by Using~~ HTML5 APIs	Audio, Video, File uploads, Geolocation
Module 9: ~~Adding~~ Offline Support ~~to Web Applications~~	Client side storage APIs - When it goes offline - What happens?
Module 10: ~~Implementing an~~ Adaptive User Interface	Device, Screen sizing, orientation, Printing
Module 11: ~~Creating~~ Advanced Graphics	SVG: Scalable Vector Graphics - Canvas API
Module 12: Animating ~~the~~ User Interface	Animating UI: (UX: User Experience?)
Module 13: ~~Implementing~~ Real-time Communication ~~by Using~~ WebSockets	WebSocket – Real Time Communication
Module 14: ~~Performing~~ Background Processing ~~by~~ Using Web Workers	Web Workers: Background JavaScript processes
Module 15: Packaging ~~JavaScript for~~ Production Deployment	Webpack, Babel CLI Transpiler

Time per itemized task requires: Each task Itemized, Create New vs. Update Existing. Plug in vs. originate solution code. Lines that must be designed, manually typed vs. Find/Replace. Number of lines in a re-factor. Testing.

The Contoso Conference web application contains the following pages:

- Home page: index.htm – Mod 08
  - Provides a brief overview of the conference, the speakers, and the sponsors.
  - The Home page also includes a **video** from the previous conference.
  - **video.js**. contains the code that students will write to play and pause the video and display the elapsed time while the video runs.
  - _namespace.js.
  - datetime.js.
  - video.js.
- Offline support for Home, About, Schedule, and Location pages – Mod 09
  - **offline.js**.
    - Methods that hide or show links depending on whether a page is currently running in the online or offline mode.
    - The pages that support the offline mode reference this script.
      - hideLinksThatRequireOnline() function
  - **appcache.manifest**
    - Lists pages that will operate offline by using the **application cache**.
  - Update the JavaScript code for the Schedule page to record an attendee's selected sessions
  - This information should be saved locally on an attendee's computer, so that it is available offline.
  - **LocalStarStorage.js**. Local Storage API

        - The Schedule page will use the addStar, removeStar, isStarred, and initialize functions defined in this file. These functions call the save and load methods that you will implement in this task.
        - LocalStarStorage save method
        - LocalStarStorage load method
    - **ScheduleItem.js**.
        - Use the local storage wrapper to save and load data in the Schedule page
- About page
    - Provides more detail about the conference and the technologies that it covers.
- Schedule.htm page:
    - Lists the conference sessions.
    - The conference has two concurrent tracks
    - Sessions are organized by track.
    - Some sessions are common to both tracks.
    - This page uses JavaScript code to connect to the web service, retrieve the list of sessions, and dynamically populate the body of this page with the session information.
    - Select the session Moving the Web forward with HTML5 and click the star icon.
    - When this happens, notice that the star changes color and that number next to the star increases.
        - This number indicates how many attendees have expressed an interest in this session; to get a good seat, the user may need to arrive early for popular sessions.
    - Click the star again to deselect the session. The number of interested attendees drops by one.
    - The functionality is implemented by a combination of CSS and JavaScript code that sends information to another web service about the sessions that a user selects.
    - In this page, notice that the list of sessions in the **<section class="page-section schedule">** element is empty;
    - it is populated when the page is displayed by using the JavaScript code in the **schedule.bundel.js** script referenced near the end of the file.
- Register page:
    - Enables the user to provide their details and register for the conference.
    - contains an HTML form in the **<section class="page-section register">** element. This form validates the data that an attendee enters. When the user **submits the form**, their details are posted to the registration service at the **URL registration/new**
- Location page: location.htm – Mod 08
    - This file contains an HTML page that
        - Displays the distance of the user from the conference site,
        - With a venue map.
    - The distance to the conference site is calculated by using JavaScript code that calls the Geolocation API, in the script **location.js**.
    - The script displays the distance in the <h2> element with the id of distance in the **<section class="travel">** element.
    - The venue map is drawn by using Scalable Vector Graphics in the **<section class="venue">** element.
    - **location-venue.js**. Mod 10
    - Complete the partially completed SVG markup of the venue map.
    - A user should be able to view more information about a room by clicking it in the map.
- Live page

- Enables an attendee to submit technical questions to the speakers running the conference sessions.
- Displays the answer from the speaker with questions (with answers) posted by other conference attendees.
- Questions and reporting are managed by using a **web socket** server.
  - The application connects to this server by opening a client connection and sending requests asynchronously.
  - The JavaScript code that implements the web socket code is located in the **live.bundel.js** file
- As other attendees post questions, **the JavaScript code behind this page** automatically updates the list of questions that is displayed.
- This file contains a form in the **<section class="page-section Live">** element that enables a user to submit questions.
- Questions are posted to a server listening on a web socket. Questions posted by other users are received by using a web socket, and then added to the list on the page.
- **LivePage.js**. Mod 13 Websocket
  - initializeSocket. This method arranges to handle the onmessage event of the socket and call the handleSocketMessage() method whenever a message arrives.
  - askQuestion. This method runs when the user has typed a question and clicked Ask. The text is formatted as a message and serialized as JSON before being sent to the web server for processing.
  - handleSocketMessage. This method runs when the web server sends a message back to the browser. The method examines the incoming message, and if it contains questions it calls the handleQuestionsMessage() method to display the questions on the web page. If the message contains a remove request, the method calls the handleRemoveMessage() method to remove the question from the web page.
  - handleQuestionsMessage. This method parses a message and displays the questions that it contains on the web page.
  - reportQuestion. This method runs when the user clicks the Report link; it sends a report message to the web server that contains the id of the question to be removed.
- space

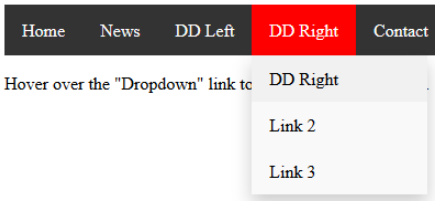
- Feedback page
  - Enables the user to rate conference sessions and speakers.
  - This page contains the feedback form in the **<section class="page-section feedback">** element, enabling attendees to provide their feedback on the conference.
  - The input fields for the first four questions are rendered as stars by using the JavaScript code in the **StartRatingView.js** file and the styles in the **feedback.css** style sheet.
  - Properties of the input fields define the maximum and minimum ratings, and each rating is displayed as a single yellow star.
  - The input field for the comments feedback is a **<textarea>** element.
    - Mod 12 - When the user submits the feedback, JavaScript code in the **feedback.js** file and styles in the **feedback.css** style sheet animates the form to make it fly off the screen.
      - Keyframe animation by using CSS
    - Mod 12 - Animate the star icons on the Feedback page so they react when moving the cursor over them
    - Mod 12 - Animate the Register link on the Home page
  - space

- speaker-badge.htm – Mod 08
    - The Speaker Badge page is not accessible from the menu bar in the application because this feature is only intended for use by the speakers. To view this page, you must browse directly to the speaker-badge.htm page on the website.
    - **speaker-badge.js**. complete the code for the SpeakerBadgePage object in this script to handle dragover and dragdrop events.
    - When the dragdrop event occurs, the event handler will read the file dropped on the image element and display the contents on the web page.
    - The displayImage function is another utility function that displays an image in the control referenced by the imageElement variable, which is created by the **initialize method**, and refers to the <img> element in speaker-badge.htm.
    - **SpeakerBadgePage.js**.
        - Read the image by using a FileReader
    - Mod 10 - Previously, you used an <img> element to drag an image of the speaker onto this page. This <img> element has been removed because you are going to modify the page to use a canvas as target for the drop.
- space

any other pages or re fits?

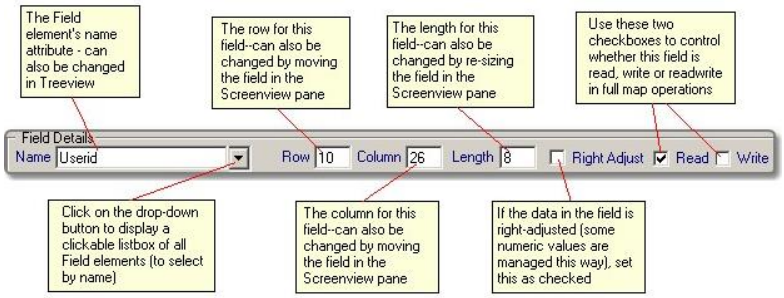
List dependencies

1. ContosoConf discussion: This app uses MVC to generate data that is stored in local storage and API CRUD URLs mapped to controllers.
2. HTML Web Pages that receive content
    a. Re-factor to HTML5 DOM: Document Object Model from previous versions.
    b. Navigation Dynamic Controls
        i. Getting values from a database or other data source for controls.
        ii.



3. JavaScript/C# - generally speaking
    a. OOP: Object Oriented Programming - Logic - Proprietary Code
    b. With a good design and solid direction: This always depends on the project whiteboard, number of developers, security considerations, business governance rules, availability of needed technology, time to production and funding.
4. Form Creation - Validating User Input (Create new records, UPDATE existing records)
    a. HTML5 Input Types and Attributes = Controls on a page or form.
    b. Validating User JavaScript
    c. C.R.U.D Operations/ORM: Object Relational Mapping
        i. Map form field to a column.
        ii. Use a form field to change current values.

iii.



5. Communicate with Remote Servers – Data Sources
    a. Fetch API vs. XmlHttpRequest to GET/POST or otherwise open a remote file to move data and handle the data with a file handler.
    b. File handler maps client side to actions on the server such as a SELECT statement.
6. CSS3: Add Style, Position, Layout models, Transforms
    a. CSS and Bootstrap Themes all work from mapping HTML elements to STYLE with selectors.
7. **Module** 7: Creating Objects and Methods by Using JavaScript(6)(2015) - https://www.w3schools.com/js/js_es6.asp
    a. Refactor the existing code for better organizational practices. The resulting code will be more maintainable and provide a good pattern for implementing features in the future.
    b. Create a new class ScheduleList
        i. Move the existing functions and variables relating to the schedule list into this new class.
        ii. Replace the code which creates LocalStarStorage and invokes startDownlaod method with creation of the ScheduleList class and invocation of the refactored method.
    c. Scoping rules for local variables, and describe how hoisting works.
        i. Var
            1. A variable has global scope if it defined outside of a function.
            2. A variable has function scope if it is defined inside a function.
        ii. ECMA-262 6th edition, also known as ECMAScript 2015, introduced block-scoped variables.
        iii. Block-scoped variables are scoped to the block statement they were defined in as well as in any contained sub-blocks.
        iv. Block-scoped variables are defined with the let or const keywords.
    d. Use immediately invoked functions, strict mode, and namespaces to minimize global name clashes in a web application.
    e. Describe ES2015 Modules
    f. Use common global objects and functions
8. Creating Interactive Pages by Using HTML5 APIs
    a. Dragging and Dropping
    b. Video/Audio
    c. File Upload
        i. Progress bar examples:
        ii. https://codepen.io/PerfectIsShit/pen/zogMXP
        iii. http://www.developphp.com/video/JavaScript/File-Upload-Progress-Bar-Meter-Tutorial-Ajax-PHP
        iv. https://www.w3.org/TR/FileAPI/
        v. https://www.sitepoint.com/html5-javascript-file-upload-progress-bar/
        vi. https://www.sitepoint.com/html5-ajax-file-upload/

   d. Geolocation
9. Client-side storage APIs - When it goes offline - What happens?
   a. After an attendee has visited the online website once, their browser will have downloaded and cached the important pages.
   b. If a Wi-Fi connection is unavailable, the attendee will still be able to view the website by using the cached information.
      i. Cookies
      ii. Session Storage API
      iii. Local Storage API
      iv. IndexDB API
      v. Application Cache API
10. Adaptive UI: Device, Screen sizing, orientation, Printing
    a. Screens/Devices
    b. Page orientation: Landscape vs. Portrait
    c. Printing
11. SVG: Scalable Vector Graphics - Canvas API
12. Animating UI: (UX: User Experience?)
    a. CSS Transitions/Transforms
    b. Keyframes
    c. (jQuery)
13. WebSocket – Real Time Communication: Chat
    a. (SignalR)
14. Web Worker: Background tasks
15. Packaging
    a. Webpack
    b. Babel CLI: Transpilers, or source-to-source compilers, are tools that read source code written in one programming language, and produce the equivalent code in another language. Languages you write that transpile to JavaScript are often called compile-to-JS languages, and are said to target JavaScript.
16. Space

Mod 07 Expanded T Notes

- The concept is part SOC(Separation of Concerns – put code into files by language or functionality)
- Modularizing JavaScript functionality by namespace, class, prototype, encapsulation or file structure.

Earlier versions of JavaScript emulated modules by using immediately invoked functions and namespaces. ECMAScript-262 6th edition (ES2015) has introduced a standard format for modules in JavaScript. ES2015 modules are written in files. One module per file and one file per module.

https://www.w3schools.com/js/js_es6.asp

https://www.w3schools.com/js/tryit.asp?filename=tryjs_const_array

Triple-Slash Directives: https://www.typescriptlang.org/docs/handbook/triple-slash-directives.html

To load a JavaScript file as a module, we need to add a script tag with a type attribute whose value is module.

<script src="mymodule.js" type="module" ></script>

The /// <reference path="..." /> serves as a declaration of dependency between files and instruct the compiler to include additional files.

There are two types of exports:

- **Named exports**

A **module** can export multiple functionalities by prefixing them with the **export** declaration.

```
//------ calc.js ------

export function sum(x,y) {

   return x + y;

}

export function multiply(x, y) {

   return x * y;

}

//------ main.js ------

import { sum, multiply } from 'calc.js';

console.log(sum(4,4)); // 8

console.log(multiply(5,2)); // 10
```

The **calc** module is exporting the sum and multiply functions and they are imported by using their names inside t he **main.js** file.

A module can also be imported as a whole and its named exports will be referred by using the **property** notation.

```
//------ main.js ------

import * as calc from 'calc.js';

console.log(calc.sum(4,4)); // 8

console.log(calc.multiply(5,2)); // 10
```

A module can define a single default export, the main exported value.

```
//------ sum.js ------

export default function sum(x,y) {

   return x + y;

}

//------ main.js ------

import sumFunc from 'sum.js';

console.log(sumFunc(4,4)); // 8
```

Coding patterns: Using Prototypes, Create, Inheritance: When doe properties isolate, methods share when re using an object as a prototype.

Useful to separate the prototype that defines the functionality of an object from the constructor that specifies the properties that an object contains. This approach is a f undamental part of the technique that many JavaScript developers use to implement inheritance

JavaScript supports the singleton pattern, and there are several global singleton objects that come as part of the st andard JavaScript library, such as **Math** and **JSON**.

•The **Math** object provides mathematical functions and constants. You access these functions and constant values directly through the **Math** object; you do not create a **Math** object first. The following example shows how to acc ess some of the members of the **Math** object:

```
let radius = 100 * Math.random();

let circumference = 2 * Math.PI * radius;
```

```
let area = Math.PI * Math.pow(radius, 2);
```

The **JSON** object provides methods for converting values to JavaScript Object Notation (JSON) strings, and for converting JSON strings back to values. The following example shows how to use the **JSON** object:

```
let anObject;

...

let anObjectAsJsonString = JSON.stringify(anObject);

let anObjectAgain = JSON.parse(anObjectAsJsonString);
```

JavaScript also provides a set of global common functions and properties that can be used with all the built-in JavaScript objects, such as **parseInt()**, **parseFloat()**, and **isNan()**. The following example shows how to use these global functions:

```
let ageEnteredByUser;

let heightEnteredByUser;

…

let age = parseInt(ageEnteredByUser);

let height = parseFloat(heightEnteredByUser);

if (isNan(age) || isNan(height))

   alert("Invalid input");
```

```
 singleton as an IIFE
var User;
(function() {
  var instance;User = function User() {
    if (instance) {
      return instance;
    }instance = this;// all the functionality
```

```
  this.firstName = 'John';
  this.lastName = 'Doe';return instance;
 };
}());
```

https://codeburst.io/javascript-global-variables-vs-singletons-d825fcab75f9

Create Objects 2 ways with "let"

```
let employee1 = new Object();
```

```
let employee2 = {};
```

Add some data properties and a method to an **employee** object:

```
let employee1 = {};

employee1.name = "John Smith";

employee1.age = 21;

employee1.salary = 10000;

employee1.payRise = function(amount) {

  // Inside a method, "this" means the current object.

  this.salary += amount;

  return this.salary;

}
```

how to access data properties and invoke methods on an **employee** object:

```
let newSalary = employee1.payRise(1000);
```

```
document.write("New salary for employee1 is " + newSalary);
```

Create Objects, Properties & Methods using "const" keyword

he following example shows how to create an object that contains properties and methods. The methods referenc e properties that are part of the same object:

```
const employee2 = {
   name: "Mary Jones",
   age: 42,
   salary: 20000,
   payRise: function(amount) {
      this.salary += amount;
      return this.salary;
   },
   displayDetails: function() {
      alert(this.name + " is " + this.age + " and earns " + this.salary);
   }
};
```

The following example defines a constructor function named **Account**, which assigns properties that represent a b ank account:

```
const Account = function (id, name) {
```

```
  this.id = id;

  this.name = name;

  this.balance = 0;

  this.numTransactions = 0;

};
```

The following example creates two **Account** objects and sets their properties. After these statements, **acc1.constructor** and **acc2.constructor** both reference the **Account** constructor function.

```
let acc1 = new Account(1, "John");

let acc2 = new Account(2, "Mary");
```

Prototypes give you a way to share functions between objects created by using the same constructor.

Prototyping & Chaining

Currently each object gets its own copy of the methods defined by the constructor, despite the fact that they are logically all the same piece of code, and each copy of the method occupies its own space in memory and has a corresponding management overhead.

- Prototypes give you a way to share functions between objects created by using the same constructor. All JavaScript objects, including constructor functions, have a special property named **prototype**.
- The prototype is really just another object to which you can assign new properties and methods; you use it as a blueprint for creating new objects.
- It automatically provides a set of functions and other properties that are inherited from the prototype of the Object type by using a mechanism known as prototype chaining.
- The data properties defined in the Account constructor (id, name, balance, and numtransactions) are specific to each object (acc1 and acc2), whereas the methods defined by the prototype (deposit, display, and withdraw) are shared by all instances.
- The "this" object used by these functions references the appropriate instance:

ES6 introduced classes

A class is a type of function, but instead of using the keyword function to initiate it, we use the keyword class, and the properties is assigned inside a constructor() method.

Use the keyword class to create a class, and always add a constructor method.

The constructor method is called each time the class object is initialized.

Example

A simple class definition for a class named "Car":

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
}
```

Now you can create objects using the Car class:

Example

Create an object called "mycar" based on the Car class:

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
}
mycar = new Car("Ford");
```