About Software Training for Students

Software Training for Students is an organization on campus that provides free software training to all students and faculty. Our services include custom workshops, open-enrollment classes, one-on-one project help, and access to Lynda.com. For more information on the Software Training for Students (STS) program, visit our website at wisc.edu/sts.

STS is part of the Division of Information Technology (DoIT) - Academic Technology at UW-Madison. For more information regarding DoIT Academic Technology, visit at.doit.wisc.edu.

© University of Wisconsin Board of Regents.

This manual and any accompanying files were developed for use by current students at the University of Wisconsin-Madison. The names of software products referred to in these materials are claimed as trademarks of their respective companies or trademark holder.

If you are not a current member of the UW-Madison community and would like to use STS materials for self-study or to teach others, please contact sts@doit.wisc.edu. Thank you.
<table>
<thead>
<tr>
<th></th>
<th>Topics Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Introduction to JavaScript</td>
</tr>
<tr>
<td>3</td>
<td>JavaScript Basics</td>
</tr>
<tr>
<td>4</td>
<td>ToDo List Web Application</td>
</tr>
<tr>
<td>5</td>
<td>Variables, Functions, Objects, and the DOM</td>
</tr>
<tr>
<td>6</td>
<td>ToDo List Web App Continued</td>
</tr>
<tr>
<td>7</td>
<td>Documentation</td>
</tr>
<tr>
<td>8</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>
Introduction

About this Class

JavaScript is used as a client-side scripting language on many of today’s most popular web sites. It may also be used as a server side programming language using a framework like Node.js. JavaScript extends the basic functionality of HTML and CSS to provide interactive features on a web page. In this class, you will learn some basic JavaScript commands and build a to-do list web application using JavaScript to dynamically alter the appearance of the web page.

Topics

This class will cover the following concepts:

1. Introduction to JavaScript
2. JavaScript Basics
3. Using JavaScript Objects
4. Using the Document Object Model (DOM)
5. Responding to JavaScript Events

Required Skills

Since JavaScript is used in conjunction with HTML and CSS, this class requires moderate experience with HTML and basic experience with CSS. If you have taken the HTML 1 and CSS 1 classes offered by Software Training for Students, you should meet these requirements.

Also, familiarity with a programming language or scripting in general is helpful, but not required for this course. JavaScript uses many of the same control structures as other programming languages (PHP, Java, C, C++, etc.) and scripting languages (BASH, etc.), but this will not be a focus of this class.

Other Requirements

We recommend that you use Google Chrome as a web browser, because it has a proper JavaScript console that we can run JavaScript statements in. Other browsers have similar developer tools, but for this class we will use Chrome.

You will also need a text editor to create JavaScript files, and view the HTML and CSS behind our web site. Example text editors for Mac are Sublime Text 2, TextWrangler, or the default text editor, TextEdit. Example text editors for Windows are SciTE, Notepad++, or the default text editor, Notepad. If you can, use a text editor that has syntax highlighting. It will highlight your code for you, and can make fixing errors a lot easier.

Pre-made files will be given to you, but if you are working on your own, feel free to navigate to the STS website to download the most recent class files.

Accessibility
When designing web pages, it is crucial to make pages accessible; that is, viewable by the broadest audience possible. The University of Wisconsin has enacted a web accessibility policy that provides guidelines on how web pages should be written so all users can access them.

The standard being used is the US Federal Government Standard Section 508. This standard provides a set of rules that, when followed, allow all users to access a web page. The rules are not difficult or intrusive, and obeying them from the start is the easiest way to make accessible pages.

This policy applies to all University programs and web pages posted on University servers. If you are writing pages for any University site, they must meet this standard. Even if you aren't required to meet this standard, you might want to consider it practice and can help to broaden your audience.

Introduction to JavaScript

Today, we will be using JavaScript as a client-side scripting language to provide more interactive features on a web page. Client-side means that the JavaScript commands are run by the web browser on the user's (client's) computer. This differs from server-side scripting languages (such as PHP), in which the commands are executed by the web server before the web pages are transferred to the user. Although JavaScript can be used as a server-side programming language, we will only be using it as a client-side scripting language in this class.

JavaScript History

What we know of today as JavaScript was originally written for the Netscape Navigator web browser in the early 1990's under the name LiveScript. It was rebranded in 1995 with the present name after a deal with Sun Microsystems to also include the Java runtime environment with the Netscape Navigator v2.0 browser.

Although JavaScript sounds similar to Java, the two are unrelated, except for some common syntax inherited from the C programming languages. Java is a full-fledged object-oriented programming (OOP) language, which could take an entire semester class to learn (See Computer Science 302). JavaScript is very simplified for ease of use, meaning you should be able to become proficient in basic statements after this two-hour class.

Since JavaScript is licensed to the Netscape family of browsers (including Mozilla Firefox among others), Microsoft actually uses a different language, called JScript to implement JavaScript features. There are a few differences between JavaScript and JScript, but in this class, we will consider both as the same.

JavaScript is weakly-typed, meaning that declaration of specific variable types is not necessary, but rather it is "figured out" by the context of a variable assignment. JavaScript is also a prototype-based language, which means that instead of pure classes (as in object-oriented programming languages) there are structures called prototypes that allow for customized objects. We will cover more on these concepts later in the class.

Client-Side Scripting

Because JavaScript is a client-side scripting language, the code to be processed is run on the user's (client's) computer. This means that it is impossible to "hide" or protect JavaScript code, as it must be readable by the user's computer to run (just as it is impossible to hide HTML or CSS). By running code locally, it can be much faster to respond to user input, for example in form validation, mouse movement, or graphical effects.

JavaScript also has limited rights, including restricting the modification of local files. In practical JavaScript use, you will be limited to modifying the appearance of the current web page only. Any more complex interactions with a computer file system will likely require server-side programming languages, such as PHP.
Even with its limited rights, there are still several security risks involving JavaScript, most of which related to out-of-date web browsers. To completely protect a computer from malicious attacks, it is often recommended to disable JavaScript. This adds a new level of complexity for web design - web pages should only JavaScript only for extra features. For this reason, we must obey the following guideline:

**Your web page should be fully usable even if JavaScript is disabled to be completely accessible.**

For this reason, it is often imperative to use JavaScript in conjunction with a server-side programming languages for form validation and good web authoring practices in all other cases.

**Uses of JavaScript**

JavaScript is a prevalent part of current web design practices. Many features of web pages that change based on user input are the result of JavaScript code. Here are a few examples:

**Lightbox**

The original lightbox script ([http://lokeshdhakar.com/projects/lightbox2/](http://lokeshdhakar.com/projects/lightbox2/)) displays images in an interactive way, giving the effect of the image floating above the page. Try navigating to the link above, and scrolling down to the “Examples” section. Click on an image in this section to view it.

![Lightbox Example](image.jpg)

**Parallax**

You can use JavaScript to make scrolling through a web page a unique experience. Parallax scrolling is the idea of having the content of your web page scroll faster, or slower, than other elements on your page. Dave Gamache has a nice web page that uses parallax ([http://davegamache.com/parallax/](http://davegamache.com/parallax/)).
Parallax, and many of these other examples can be easily used in a webpage using a JavaScript library called jQuery. You will learn more about jQuery in our JavaScript 2 class.

Full Screen Forms

JavaScript can be used to make filling out forms a simple experience on your web page, by using full screen HTML forms. Full screen forms are used in a demo shown below. ([https://tympanus.net/codrops/2014/07/30/fullscreen-form-interface/](https://tympanus.net/codrops/2014/07/30/fullscreen-form-interface/)).

Although we will only reach the very basics of these effects in this class, check out the JavaScript 2 class for more practical examples of these JavaScript effects and more.

**JavaScript Basics**

**In-Line Scripting**

The simplest way to use JavaScript on a web page is very similar to internal style sheets using CSS. Instead of using the `<style>` tag as in CSS, in-line JavaScript uses the `<script>` tag to specify JavaScript code. JavaScript commands typically use (but are not required to have) a semicolon after each line.

**Hello World Script**
1 Open the class file folder named "Basics" inside of the "JS1_ClassFiles" folder.

2 Right Click the file "index.html" and open with a text editor of your choice.

3 Between the script tags, Write the following.

```html
<script type="text/javascript">
    console.log("hello world!");
</script>
```

The console.log() method takes a string-type or number-type argument (between the parentheses) and writes it to the console. You can either use single quotation marks (') or double quotation marks (") to specify a string of text. Whatever you use, just be sure to stay consistent. If you are writing a number to the console, do not use quotation marks.

4 Save the file "index.html"

5 Open index.html in Google Chrome

6 If you are on a Mac, Use the shortcut: command-option-J. If you are on Windows, Use the shortcut: control-shift-J. Alternatively, you can Navigate to View > Developer > JavaScript Console.

You should see hello world printed to the console. Next, we will use the console to run some simple JavaScript commands.

### Writing JavaScript to the Console

We can use the JavaScript console in Google Chrome to write and execute JavaScript statements "on the fly". Next, we will learn about some simple statements we can execute in JavaScript. Using the console to practice these statements is recommended, because it eliminates the issues you might run into saving and refreshing an internal script in your HTML code.

### Writing Strings to the Web Page

1 If the JavaScript console is not open, Open it now. Remember, Use the keyboard shortcut: command-option-J if you are on a Mac, or control-shift-J if you are on Windows.

2 Type the following into your JavaScript console.
Clearing the Console

If you ever want to clear the JavaScript console, just type the command: `clear();` and hit enter.

Return Values in the Console

After you execute a statement in the console, you may see 'undefined' written below your statement. This is the return value, and we will learn more about these later.

Writing HTML Code to the Web Page

1. Insert your cursor into the console, if it is not already there.
2. Press the Up Arrow key to access previously typed statements.
   
   The Down Arrow can be used also, in order to navigate through the console history.
3. Using the arrow keys, Find the statement we typed in the previous exercise. **Do Not** press enter yet.

   ```javascript
   document.write("I just wrote to the web page!");
   ```

4. Insert HTML h2 tags into the statement.

   ```javascript
   document.write("<h2>I just wrote to the web page!</h2>");
   ```

5. Press Enter to execute the statement.

   Writing HTML code to the page is a more practical example of how you might write formatted text to the HTML document.

Displaying a Pop-Up Alert

1. Open the JavaScript console, if it is not already open.
2. Type the following into your console.
Complex Math

For more complex mathematical operations, such as rounding, you should use the Math object that we will talk about later.

Modal Windows

Alert dialog boxes prevent the user from accessing the rest of the web page until the alert is closed. They can be a bit intrusive, so be careful no to overuse them.

Basic Math

1. Open the JavaScript console, if it is not already open.

   We can do some basic mathematical operations in the JavaScript console using some standard mathematical operators shown below. If you have had experience with other programming languages, these operators might be familiar.

   - Addition: +
   - Subtraction: -
   - Multiplication: *
   - Division: /
   - Modulus (Remainder): %
   - Grouping Parentheses: ( )

2. Type the following into your console.

   ```javascript
   (48%13)/2
   ```

3. Hit Enter.

   You should get a result of 4.5 because 48%13 = 9 and 9/2 = 4.5

ToDo List Web Application

For the rest of this class, we will build a ToDo list 'web application'. This web application will not have any back-end, or server side functionality. All this means is that if we refresh our page, all of our changes to the ToDo list will be lost. All of the functionality will be implemented using JavaScript. We will not use any other libraries or languages. A ToDo list web application, like this one, could be made quicker with a JavaScript library like jQuery, but that will not be
covered in this course. If you would like to learn more about jQuery, we recommend that you take the JavaScript 2 course offered by STS.

The Finished Product

A finished version of the ToDo list web application is given to you in the class files.

Viewing in the Browser

1. Close the previous window containing "index.html", if you have not done so already.
2. Open the "JS1_ClassFiles" folder.
3. Open the "Finished_ToDo" folder.
   - Observe how the finished web application is laid out, with an "index.html" page in the root folder, along with folders containing any JavaScript files and CSS files named "js" and "css", respectively.
4. Right-Click "index.html" and open with Google Chrome.
5. Click on the different buttons and input elements on the web page, and experiment to learn how it works.
6. Add a few items to your ToDo list by using the "ADD ITEM" text field.
7. Refresh the browser window, and view the changes.

You should lose all changes you just made. Once again, this is because our 'web application' has no back-end or server side code. If you would like to learn more about server-side programming, feel free to take the PHP and MySQL classes offered by STS.

Viewing the JavaScript Code

1. Open the "JS1_ClassFiles" folder.
2. Open the "Finished_ToDo" folder.
3. Open the "js" folder containing the JavaScript file named "app.js".
4. Right-Click "app.js" and Open With a text editor of your choice.

The JavaScript file is broken up into three sections. There are global variables, functions, and the point where the JavaScript will start executing. When the browser reads our JavaScript file, it will start actually running our code at the "Execution Start" section. The global variables, and functions are just there for code to be referenced as the JavaScript runs. Just realize that the browser will create the global variables, and the functions, but will not do anything if we do not use these variables and functions. The idea of having global variables, and functions at the top of our JavaScript file is common, and thought of as a best-practice technique. If you are confused by any of this, do not worry, we will go over most of this as we work through the course.

Intro to Building the Application

Next, we will work on the incomplete web application. The end goal of this class is to have a working ToDo list web application. There are a few steps that we will have to go through, but we can start out by linking our JavaScript file
with our HTML page.

External Scripting and Linking

1. Close "app.js", if you still have it open.
2. Open the "JSI_ClassFiles" folder.
3. Open the "ToDo" folder.

For the remainder of the class, we will assume that you are working within this folder; the "ToDo" folder.

4. Open "index.html" in Google Chrome
5. Try Clicking on the buttons on your web page.

None of the buttons are working! This is because we did not link our "index.html" file with our "app.js" file. Let's do that now.

6. Navigate back to your file window.
7. Open "index.html" in a text editor of your choice.
8. Scroll Down towards the bottom of the HTML document. There should be an area that says:

   <!-- external JavaScript link goes here -->

This is the syntax for an HTML comment. HTML comments are useful to mark important areas in your HTML document, and are not executed by the browser.

9. Type the following directly below the comment.

   <script type="text/javascript" src="js/app.js"></script>

This is another way of linking to a JavaScript file, called external linking. We can write JavaScript code in a separate file (saved with a .js extension), and then we can execute JavaScript anywhere in our HTML file by linking to the external JavaScript file. This is usually more readable than internal scripting that we saw earlier in the course.

10. Save the "index.html" file, and Open it in Google Chrome.

You might notice that nothing really happened. There are some errors in our JavaScript file! Let's check out the JavaScript console for more information.

Debugging

1. Open the JavaScript console in Google Chrome for the "index.html" ToDo list web application page.
Declaring Variables

Always put the "var" keyword in your variable declarations. This will reduce errors in your code.

Remember, on a Mac, use **command-option-J** to open the JavaScript console. On Windows, use **control-shift-J**. You should see some red statements with errors.

2 In Google Chrome’s JavaScript console, Click on the gray, drop down arrow to the left of the error statement to see more information.

It should say (anonymous function). We will learn more about functions later, so do not worry about it now. Just remember that the drop down arrow can give you more information that can be useful in finding what exactly went wrong in your code. On the right side of the error, Chrome lists the line that the error took place at.

3 Click the link that displays the line number. This link is on the right side of the error.

The JavaScript console will take you to the JavaScript code that had thrown an error. Now that we've found the error, we need to fix it. This process of finding errors and fixing them is common in most programming languages, and is called "debugging". Keep in mind that in JavaScript, you usually need to open the JavaScript console in order to find errors in your code. If there is an error in your JavaScript code, the rest of the JavaScript code (from the error to the end of the script) will not run. Before we fix these errors, we need a little more background about JavaScript.

Variables, Functions, Objects, and the DOM

For the remainder of the class, we will be creating and storing objects into variables so users of our web application can dynamically interact with their ToDo lists. We access objects on our HTML page through JavaScript by using the Document Object Model (DOM).

Variables

A variable can be thought of as a box that contains some sort of information. For example, you could look inside the box and see what's inside, or you could change what is stored in the box. In JavaScript, you should think of a variable as a box that holds a value. If you decide to store something into a variable, this process is called "assigning" a variable. It would be similar to storing something into an empty box. If you have the name of a variable, you can easily read the value stored in that variable by simply using the name wherever you need the value.

In JavaScript we can declare variables using the following syntax,

```
var variableName;
```

where "var" is a special keyword that denotes a variable, and "variableName" is whatever you choose to name the variable. There are some restrictions as to what you can name your variables. Good reference for this information and reference for JavaScript in general can be found on the Mozilla Developer Network, or MDN ([https://developer.mozilla.org](https://developer.mozilla.org)).
If you would like to store a value into the variable, you can do so on the same line by using the following syntax,

```javascript
var variableName = some value here;
```

where "some value here" could be a string, number, or object.

You can declare and assign variables on separate lines as well. Use the following syntax.

```javascript
var variableName;
variableName = some value here;
```

This syntax works in the same way as the single line declaration and assignment seen above.

Keep in mind that in JavaScript, variables are case-sensitive. This means variables named "message", "Message", and "MESSAGE" could all represent different things. Also, since JavaScript is a loosely-typed language, variables can be assigned any type of data with one general statement.

Functions

Functions in JavaScript are used to reduce redundancy in your code. It is a good idea to follow the DRY (don't repeat yourself) programming principle when writing JavaScript. This means that any duplicated JavaScript code should probably be put into a function. Wherever you use that code, you may call that function.

A common way to create a function in JavaScript can be done using the following syntax.

```javascript
var myFunctionName = function() {
    // repeatable code goes here!
}
```

This way of creating a function and storing it into a variable is common in JavaScript. If you want to call the function we declared above, use the following syntax.

```javascript
myFunctionName();
```

Call the function wherever you need to repeat the code that is inside of the function declaration.

There is another way to declare functions in JavaScript using the following syntax. However, in the exercises in this class, we will only use the declaration shown before for our functions.

```javascript
function myFunctionName() {
    // repeatable code goes here!
}
```
Calling a function declared like the function above, is the same as before.

"function" is a keyword in JavaScript and always has to be used when declaring a function. You may add arguments to the function declaration between the parenthesis, and you may also return values from the function. Below is an example function declaration and function call that takes two numbers, adds them together, and stores them in a variable named "result".

```javascript
var addNumbers = function(num1, num2) {
  return num1+num2;
}
var result = addNumbers(4,5);
```

In this example, "result" should hold the integer 9.

**Variable Scope**

Variables in JavaScript have functional scoping. Scoping is a concept that describes where declared variables can be "seen". If you declare a variable at the top of a script file, it can be "seen" and used throughout the whole script. If you declare a variable within a function, it can only be seen and used within that function. This can be problematic sometimes because it is possible to accidentally write new values to existing variables.

Just be sure that whenever you declare a variable, you always use the "var" keyword. This will reduce errors in your code. Also, a good rule of thumb is to declare any variables that you will use throughout the whole script at the top of your JavaScript file, and declare any variables that you will use within certain functions at the top of the function declaration.

**Objects**

Objects in JavaScript represent something that has properties and methods. You can think of an object like a car. Properties of a car would be make, model, year, and more. Some methods that the car would have could be drive, play radio, lock doors, and more. We use objects in JavaScript and other languages. Some common objects in JavaScript are the Math object, Date object, and most importantly, the Window object. Documentation on these objects can be found on the Mozilla Developer Network, or MDN (https://developer.mozilla.org).

**Document Object Model (DOM)**

The DOM is a web Application Programming Interface (API) that allows access to and modification of the current HTML document being displayed. You can think of the DOM as a virtual representation of the entire web page, including the browser that the HTML is being viewed in.
The DOM contains the following objects.

- **Document**: A representation of the HTML source that allows JavaScript to dynamically edit any aspect of a web page. The `write()` function that we used previously is a part of the document object (does `document.write` make sense now?).
- **Window**: A representation of the instance of the browser window. It can be used to move, resize, or otherwise affect the entire browser window. The `alter()` function is a part of the window object. The Document object is a property of the Window object and can be accessed by typing `window.document`. It is usually shortened to just `document`, however.
- **Navigator**: A representation of the specific browser program used to view the web page. It contains information like the browser's name, version, language, and other properties.
- **Screen**: A representation of the computer's monitor. It contains information like the screen resolution and number of colors.
- **History**: A representation of the browser's history. It can be used to command a browser to go forward or back through the list of visited web pages.
- **Location**: A representation of the current web site URL or address. It can be used to redirect the browser to a new web site.

### ToDo List Web App Continued

For the remainder of the class, we will work towards finishing the ToDo list web application. Remember, the Mozilla Developer Network, or MDN ([https://developer.mozilla.org](https://developer.mozilla.org)) is a good reference for the many things you can do with the DOM, and with JavaScript in general.

### Global Variables

To fix the error that was displayed in the section before, we need to assign some global variables in our JavaScript code. Global variables are simply variables that can be seen by your entire program, or script. They can be used on any line of your script and changed on any line of your script. In the next exercise, you will assign global variables that represent parts of your HTML page. By assigning these variables, we are able to connect objects in our HTML page to our JavaScript code.

### Incomplete Tasks Holder Variable
The previous "Uncaught ReferenceError" you saw in the console can be fixed by a simple assignment of a global variable.

1. Open the "js" folder.
2. Open "app.js" in the text editor of your choice.
3. In the Global Variables section (between the rows of comment slashes), towards the top of the file, Type the following.

```
var incompleteTasksHolder = document.getElementById("incomplete-tasks");
```

4. Save the file "app.js".
5. Right-click "index.html" and Open With Google Chrome to view the changes.
6. Click the Edit or Save buttons for the items in the TODO section of the web application, and view their functionality. Try changing the name of one of the items.
7. Click the Delete button for the items in the TODO section of the web application, and view the functionality.

In the code you wrote above, we created a variable named <incompleteTasksHolder> and stored the TODO section unordered list, <ul>, element object into the variable. Every item under the TODO section of the web application is a list item. We used the DOM to access this HTML element, by getting the element from the document with the id of "incomplete-tasks". Check out the HTML code for the web application, and you will see that the <ul> element under the TODO section has an id="incomplete-tasks". The getElementById method is a method of the document object. We call this method using "dot notation", and pass it an argument of the id of the element we want.

**Completed Tasks Variable Holder**

We will need a variable that stores the unordered list under the COMPLETED section of the web application. List items in this unordered list are displayed with a line through them using CSS. All we need to worry about is creating the variable.

1. In the Global Variables section, towards the top of the file, Type the following.

```
var completedTasksHolder = document.getElementById("completed-tasks");
```

2. Save the file "app.js".

**New Task Input Text Field Variable**

Next, we will create a variable that represents the text box under the ADD ITEM section of the web application.

1. In the Global Variables section, towards the top of the file, Type the following.
Add Button Variable

We need to store the "Add" button under the ADD ITEM section of the web application into a variable. The following steps outline how to do so.

1. In the Global Variables section, towards the top of the file, Type the following.

```javascript
var addButton = document.getElementsByTagName("button")[0];
```

2. Save the file "app.js".

You may notice that the assignment of this variable is done a bit differently. We used the DOM to get a "collection" of elements. Because the method call returns a collection, we can get the first element in the collection by adding [0] to the end of the method call. Looking at the whole method, we get all button tags on our document, and then assign the very first button found to the variable named "addButton".

New Task Function

We need to write the code for what happens when we click the "Add" button under the ADD ITEM section. We will store this function into a variable so we can use it where we need to.

Creating the Function

1. Scroll Down to the "Functions" section of the JavaScript Code.
2. Find the comment that says "add a new task".
3. Below the comment, Type the following.

```javascript
var addTask = function() {
    console.log("add task...");
}
```

In the code above, we create a function that takes no parameters, and logs "add task..." to the console. By writing what's going on in the console, it will help us later if we run into any errors, or "bugs". We will add more to this function next.

4. Save the file "app.js".
Conditional Statements

We would like to only add ToDo items to our list if there is something typed into the text input field under the ADD ITEM section. If there is nothing typed in the text field, and the user clicks the add button, we would like to alert the user and tell them to type something into the text input. This issue can be solved with conditional statements.

1. Inside of the addTask function, Add an if-else statement. Make your code look similar to the code below.

```javascript
var addTask = function () {
    console.log("add task...");
    if (taskInput.value !== "") {
    
    } else {
        alert("please enter a ToDo item");
    }
}
```

In the code above, dot notation is used to get the value of what is typed into the text input under the ADD ITEM section. If there is something typed into the field, the code will enter the if statement. If there is nothing typed into the field, it will alert the user. This is called control flow in programming. Documentation on control flow can be found on the Mozilla Developer Network, or MDN (https://developer.mozilla.org).

2. Save the file "app.js".

3. Open "index.html" in Google Chrome, and see if the alert works.

You may notice that if we press the "Add" button, it does not do anything. In order to call this function when the "Add" button is pressed, we need to use events in JavaScript. We will link the function to the button in the following exercise.

Adding an Event Listener

Events in JavaScript refer to when users click, scroll, mouse over, or interact in other ways with certain elements on the page. Right now, nothing happens when we press the "Add" button, because we have are not currently "listening" for any events. We would like to respond to a click event on the "Add" button. Follow the steps below to do so.

1. Scroll Down to the section that has the "Execution Start" comment.

2. Find the comment that says "set the click handler to the addTask function".

3. Below the comment, Type the following.

```javascript
addButton.addEventListener("click", addTask);
```
This method takes in two parameters. The first parameter is the event that you would like to listen for, stated as a string. In this case, we want to listen for anytime a "click" occurs on the "Add" button. The next parameter passed into the method call is the function that we want to run if this event actually happens. By writing this code, it acts as a link between the "addTask" function and the "Add" button.

4. Save the file "app.js".

5. Open "index.html" in Google Chrome, and view the changes.

If the "Add" button is pressed without anything entered into the text field next to it, then our web application should display an alert. If there is something entered into the text field, and the "Add" button is pressed, it should not do anything. Next, we will write the code to handle what happens when a ToDo item is actually added.

Creating the New List Item

When the user clicks the "Add" button, a new list item containing a checkbox, buttons and more needs to be created. A function that creates this list item is given to you, but we need to call it in our function to create the new ToDo item.

1. Inside of the addTask function, Declare a new variable that holds the new list item. Make your code look similar to the code below.

```javascript
var addTask = function() {
    console.log("add task..."y);
    if (taskInput.value !== "") {
        var listItem = createNewTaskElement(taskInput.value);
    } else {
        alert("please enter a ToDo item");
    }
}
```

In the code above, the function named "createNewTaskElement" is called with a parameter passed into it containing the input fields value. Check out the "createNewTaskElement" function to further understand how everything is created.

2. Save the file "app.js".

Appending the New List Item

Now that the new list item is created, it is important that the list item is added to the incomplete-tasks list. We will do so in this exercise.

1. Inside of the addTask function, Call a Method that will append the list item. Make your code look similar to the code below.
In the code above, "incompleteTasksHolder" is a variable that holds the <ul> element under the TODO section of the web page. We call the method "appendChild" using dot notation, and pass it a parameter of the new list item. The "appendChild" method will append a list item to an unordered list.

Save the file "app.js".

Open "index.html" in Google Chrome, and view what happens when you add a new ToDo item.

You may notice that the "Edit" and "Delete" buttons, and the checkboxes for our new events do not work. We will fix this soon. We will also write the code that will clear out the text input under the ADD ITEM section when the "Add" button is pressed. The latter is not necessary for our web application to work properly, but it is a good idea to add this functionality.

### Binding More Events

We need to bind event listeners to the buttons and checkbox on the new list items that we add. There is a function already written that does this for you. A simple function call will solve the issue of the buttons and checkboxes not working on new events.

1. Inside of the addTask function, Call a Function that binds the necessary events. Make your code look similar to the code below.

```javascript
var addTask = function() {
  console.log("add task...");
  if (taskInput.value !== "") {
    var listItem = createNewTaskElement(taskInput.value);
    incompleteTasksHolder.appendChild(listItem);
  } else {
    alert("please enter a ToDo item");
  }
}
```

The "bindTaskEvents" function takes two parameters. The first parameter, "listItem", is the list item that we want to bind events to. For the purposes of this code, we want it to be the new list item we are adding. The second parameter, "taskCompleted" is a function that marks tasks as complete. We pass it into the function
call to create the code that determines what the checkbox will do if checked. If you want to look into what
code is written in the "bindTaskEvents" function, feel free to do so.

2 Save the file "app.js".

3 Open "index.html" in Google Chrome, and view the changes.

The buttons and checkboxes for any new ToDo item should now work properly. In the next exercise, we will write the
code that will clear out the text input under the ADD ITEM section when a new ToDo item is created.

Clearing the ADD ITEM Text Input Field

All that needs to be added to the "addTask" function is the code that will clear the text input field under the ADD
ITEM section.

1 Inside of the addTask function, Type the line of code that clears the text input. Make your code look similar to
the code below.

```
var addTask = function() {
    console.log("add task...");
    if (taskInput.value !== "") {
        var listItem = createNewTaskElement(taskInput.value);
        incompleteTasksHolder.appendChild(listItem);
        bindTaskEvents(listItem, taskCompleted);
        taskInput.value = "";
    } else {
        alert("please enter a ToDo item");
    }
}
```

By setting taskInput's value to the empty string character (""), or two blank quotes, it clears out anything typed
into the text field.

2 Save the file "app.js".

3 Open "index.html" in Google Chrome, and view the changes.

There are still a couple of bugs in our web application! You may have noticed that the checkbox and buttons for any
pre-existing COMPLETED ToDo items do not work. We will write the code to fix these errors in the following
exercises.

Binding Events for Completed Items

When our web app loads, there are some pre-existing ToDo items in the TODO and COMPLETED sections. All of the
TODO items work properly, but the buttons and checkbox on the single, pre-existing COMPLETED item does not
work properly. If our web application was able to save our progress as we closed our web browser, it would be
important to fix this issue, because we could have multiple items in the COMPLETED section. Currently, the web
application does not save any progress you make, and always reverts to the default ToDo list that is first displayed
upon browser refresh.
Cycling Over COMPLETED Pre-Existing Items

To cycle over the items in the COMPLETED section, we will use a loop. Loops in JavaScript simply run the same code for a specified amount of time or runs. Some loops can be based on conditions of what is currently going on. For the purposes of cycling over the items in the COMPLETED section, we will use something called a "For-Loop" which is common in many programming languages, including JavaScript.

1. Scroll Down to the section that has the "Execution Start" comment.
2. Find the comment that says "cycle over the completeTasksHolder ul list items".
3. Below the comment, Type the following.

   ```javascript
   for (var i=0; i<completeTasksHolder.children.length; i++) {
   ```

4. Find the comment that says "bind events to list item's children (taskIncomplete)".
5. Below the comment, Type the following closing curly brace to close the "For-Loop".

   ```javascript
   }
   ```

The "For-Loop" declaration above will run any code between it’s curly braces the same number of times as the number of list items under the COMPLETED section’s unordered list. This will give the effect of cycling through the items under the COMPLETED section. Every time the "For-Loop" runs, the variable "i" is incremented by one and the code inside runs until it is no longer less than the number of children that the unordered list under the COMPLETED section holds.

6. Save the file "app.js".

In the following exercise, we will complete the inside of the "For-Loop" to bind events to the pre-existing list items.

Getting the Current List Item

As the "For-Loop" loops through the children (list items) of the unordered list, we can use the counter variable (i) to get the current list item, and bind events to it. To get the current list item, follow these steps.

1. Find the comment that says "bind events to list item's children (taskIncomplete)".
2. Below this comment, but Above the closing "For-Loop" curly brace, type the following.

   ```javascript
   var listItem = completedTasksHolder.children[i];
   ```

Your code should look like the following, below.
By typing completedTasksHolder.children, you get all the children (list items) of the unordered list under the COMPLETED section. It is possible to get the i'th element of the collection, by using square brackets. This code will grab the current list item for the given iteration of the "For-Loop", and store it into a variable named "listItem".

Save the file "app.js".

Now that you have the current list item stored into a variable, you can write the code to bind events to it.

### Binding Events to the Current List Item

1. Inside of the "For-Loop", Type the code to bind events to the list item. Make your code look similar to the code below.

```javascript
// cycle over the completeTasksHolder ul list items
for (var i=0; i<completedTasksHolder.children.length; i++) {
  // bind events to list item's children (taskIncomplete)
  var listItem = completedTasksHolder.children[i];
  bindTaskEvents(listItem, taskIncomplete);
}
```

The "bindTaskEvents" function takes two parameters. The first parameter, "listItem", is the list item that we want to bind events to. For the purposes of this code, "listItem" is the current list item that the "For-Loop" is on at that time. The second parameter, "taskIncomplete" is a function that marks tasks as incomplete. We pass it into the function call to create the code that determines what the checkbox will do if unchecked. If you want to look into what code is written in the "bindTaskEvents" function, feel free to do so.

2. Save the file "app.js".

3. Open "index.html" in Google Chrome, and view the changes.

The checkbox and buttons on the pre-existing COMPLETED item that is displayed upon page refresh should work properly. There is one more line of code to be written, that will change the functionality of the list item if the pre-existing item under the COMPLETED section is in edit mode or not. Right now, it will always not be in "edit mode" upon page refresh, but it is a good idea to write the code in case this web application had saving functionality, and you accidently left the page and closed out of the browser while you were editing a COMPLETED ToDo item.

### Checking For "Edit Mode" on Pre-Existing List Items

1. Inside of the "For-Loop", Type the code to check if the list item is in "edit mode" or not. Make your code look similar to the code below.
// cycle over the completeTasksHolder ul list items
for (var i=0; i<completedTasksHolder.children.length; i++) {
  // bind events to list item's children (taskIncomplete)
  var listItem = completedTasksHolder.children[i];
  bindTaskEvents(listItem, taskIncomplete);
  checkForEditMode(listItem);
}

The function, "checkForEditMode" takes one parameter. The single parameter is the list item to check if in "edit mode" or not. Feel free to view the code that is run within the "checkForEditMode" function.

2 Save the file "app.js".

3 Open "index.html" in Google Chrome, and view the changes.

This completes our ToDo List web application. Hopefully, this real life example gave you a better idea of what JavaScript can do to a web page. It adds interactivity, and really enhances the user experience. If you are still feeling confused about anything relating to JavaScript, feel free to check out the documentation attached to the end of this manual. Once again, a good online reference is the Mozilla Developer Network, or MDN (https://developer.mozilla.org). It is very uncommon for a JavaScript developer to write an entire script without having to look something up or use documentation or some sort of reference. Do not be ashamed of simply doing a web search if you are stuck on a certain part while writing your code, also.

### Documentation

#### Operators

#### Arithmetic Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>x + y</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>x - y</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>x * y</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>x / y</td>
</tr>
<tr>
<td>%</td>
<td>Modulus (Remainder)</td>
<td>x % y</td>
</tr>
<tr>
<td>++</td>
<td>Increment</td>
<td>x ++ y</td>
</tr>
<tr>
<td>--</td>
<td>Decrement</td>
<td>x -- y</td>
</tr>
</tbody>
</table>

#### Assignment Operators
### Operator Table

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Same As</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>x = y</td>
<td>x = y</td>
</tr>
<tr>
<td>+=</td>
<td>x += y</td>
<td>x = x + y</td>
</tr>
<tr>
<td>-=</td>
<td>x -= y</td>
<td>x = x - y</td>
</tr>
<tr>
<td>*=</td>
<td>x *= y</td>
<td>x = x * y</td>
</tr>
<tr>
<td>/=</td>
<td>x /= y</td>
<td>x = x / y</td>
</tr>
<tr>
<td>%=</td>
<td>x %= y</td>
<td>x = x % y</td>
</tr>
</tbody>
</table>

### Comparison Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal To</td>
<td>x == y</td>
</tr>
<tr>
<td>===</td>
<td>Equal To and Same Type</td>
<td>x === y</td>
</tr>
<tr>
<td>!=</td>
<td>Not Equal To</td>
<td>x != y</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater Than</td>
<td>x &gt; y</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less Than</td>
<td>x &lt; y</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater Than or Equal To</td>
<td>x &gt;= y</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less Than or Equal To</td>
<td>x &lt;= y</td>
</tr>
</tbody>
</table>

### Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>And</td>
<td>(x &lt; y) &amp;&amp; (x &lt; z)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Not</td>
<td>!(x &lt; y)</td>
</tr>
</tbody>
</table>

### Conditional Statements

**If / Else If / Else Statement**
Switch Statement

```java
if(condition1) {
  // code to execute if condition1 is true
} else if (condition2) {
  // code to execute if condition1 is false
  // and condition2 is true
} else {
  // code to execute if condition1 and
  // condition2 are both false
}
```

Switch Statement

```java
switch(x) {
  case 1:
    // code to execute if x==1
    break;
  case 2:
    // code to execute if x==2
    break;
  default:
    // code to execute if x is neither 1 nor 2
}
```

Loops

For Loop

```java
for(x=start; x<=end; x+=increment) {
  // code to execute for each loop
}
```

While Loop

```java
while(condition) {
  // code to execute for each loop
}
```
For ... In Loop

The For ... In loop works much like an iterator in Java or a foreach loop in many other languages.

```javascript
for(variable in object) {
    // code to execute for each variable in object
}
```

Date Object

The date object provides a structure for working with both dates and times, as well as getting information about the current date and time. Again, since JavaScript is a client-side scripting language, it is able to be "fooled" if you change your computer time so its use is somewhat limited but it can provide some useful effects.

Here are the most common date object functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date()</td>
<td>Gets the current system date and time</td>
</tr>
<tr>
<td>getDate()</td>
<td>Extracts the date (1-31) from a Date Object</td>
</tr>
<tr>
<td>getDay()</td>
<td>Extracts the day of the week (0-6) from a Date Object</td>
</tr>
<tr>
<td>getMonth()</td>
<td>Extracts the month (0-11) from a Date Object</td>
</tr>
<tr>
<td>getFullYear()</td>
<td>Extracts the year (4-digit) from a Date Object</td>
</tr>
<tr>
<td>getHours()</td>
<td>Extracts the hour from a Date Object</td>
</tr>
<tr>
<td>getMinutes()</td>
<td>Extracts the minutes from a Date Object</td>
</tr>
<tr>
<td>getSeconds()</td>
<td>Extracts the seconds from a Date Object</td>
</tr>
</tbody>
</table>

String Object

The string object provides built-in functions for dealing with text. We will find it to be useful when we get to form validation and when we want to check the capitalization or length of words typed into text boxes.

Here are the most common string object functions.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>charAt(i)</td>
<td>Extracts the character at position i</td>
</tr>
<tr>
<td>concat(s1, s2)</td>
<td>Combines (concatenates) two or more strings</td>
</tr>
<tr>
<td>indexOf(s)</td>
<td>Gets the index of a given substring</td>
</tr>
<tr>
<td>replace(s1, s2)</td>
<td>Replaces some characters with different characters</td>
</tr>
<tr>
<td>search(s)</td>
<td>Searches through the string for a specific substring</td>
</tr>
<tr>
<td>substr(i, l)</td>
<td>Extracts a substring of length l at a starting position i</td>
</tr>
<tr>
<td>toLowerCase()</td>
<td>Converts a string to all lower-case characters</td>
</tr>
<tr>
<td>toUpperCase()</td>
<td>Converts a string to all upper-case characters</td>
</tr>
<tr>
<td>length</td>
<td>Gets the number of characters in a string</td>
</tr>
</tbody>
</table>

**Math Object**

The math object provides access to commonly-used mathematical functions.

Here are the most common math object functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(x)</td>
<td>Absolute value of x</td>
</tr>
<tr>
<td>ceil(x)</td>
<td>Ceiling of x (round up to nearest integer)</td>
</tr>
<tr>
<td>floor(x)</td>
<td>Floor of x (round down to nearest integer)</td>
</tr>
<tr>
<td>cos(x)</td>
<td>Cosine of x</td>
</tr>
<tr>
<td>exp(x)</td>
<td>Exponential of x</td>
</tr>
<tr>
<td>log(x)</td>
<td>Logarithm (base e) of x</td>
</tr>
<tr>
<td>min(x,y)</td>
<td>Minimum of x and y</td>
</tr>
<tr>
<td>max(x,y)</td>
<td>Maximum of x and y</td>
</tr>
<tr>
<td>pow(x,y)</td>
<td>x raised to the y power</td>
</tr>
<tr>
<td>random()</td>
<td>Random number between 0 and 1</td>
</tr>
<tr>
<td>round(x)</td>
<td>Rounds to the nearest integer</td>
</tr>
<tr>
<td>sin(x)</td>
<td>Sine of x</td>
</tr>
<tr>
<td>sqrt(x)</td>
<td>Square root of x</td>
</tr>
<tr>
<td>tan(x)</td>
<td>Tangent of x</td>
</tr>
</tbody>
</table>
Navigator Object

The navigator object is very useful to find out what type of browser the user is using (this is called browser detection). It is a fact of life that different browsers will handle web pages differently (especially older versions of browsers), so being able to detect an incompatible browser may be beneficial. In order to find out what browser the user is currently viewing your website on, get the userAgent object, and check to see if it has a specific browser name key as a property.

```javascript
var usrAg = navigator.userAgent;
var browserName;
if (usrAg.indexOf("Chrome") > -1) {
    browserName = "Google Chrome";
}
else if (usrAg.indexOf("Safari") > -1) {
    browserName = "Safari";
}
else if (usrAg.indexOf("Opera") > -1) {
    browserName = "Opera";
}
else if (usrAg.indexOf("Firefox") > -1) {
    browserName = "Mozilla Firefox";
}
else if (usrAg.indexOf("MSIE") > -1) {
    browserName = "Microsoft Internet Explorer";
}
alert("You are using: " + browserName);
```

The code above finds out what browser you are using and displays it in an alert. You should type " > -1" in the conditional, because the "indexOf" method returns -1 if an index for the given String argument could not be found.

Document Object

In addition to having a set of functions associated with it, the document object also is a way to "view" the HTML of a page in a tree structure.

In this tree, the important relationship to keep in mind is the parent-child link (think family tree). A child tag (or node as it is sometimes referred to) is any tag directly inside of its parent tag. In the above example, `<head>` and `<title>`
are children of `<html>` and `<html>` is the parent of `<head>` and `<title>`.

Here are the most common document object functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getElementById()</code></td>
<td>Gets the HTML tag with a given id attribute</td>
</tr>
<tr>
<td><code>getElementByName()</code></td>
<td>Gets a set of HTML tags with a given name attribute</td>
</tr>
<tr>
<td><code>getElementsByTagName()</code></td>
<td>Gets a set of HTML tags of a certain tag name</td>
</tr>
<tr>
<td><code>parentNode</code></td>
<td>Gets the parent HTML tag</td>
</tr>
<tr>
<td><code>firstChild</code></td>
<td>Gets the first child HTML tag</td>
</tr>
<tr>
<td><code>childNodes[i]</code></td>
<td>Gets the &quot;i&quot;th child node</td>
</tr>
<tr>
<td><code>lastChild</code></td>
<td>Gets the last child HTML tag</td>
</tr>
</tbody>
</table>

Once you start editing an individual tag, you can use several other functions to do things like modify the text and CSS styling. Some of the most useful functions that can be used on individual tags include:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>innerHTML</code></td>
<td>Sets the text of textual tags</td>
</tr>
<tr>
<td><code>src</code></td>
<td>Sets the source URL for image tags</td>
</tr>
<tr>
<td><code>href</code></td>
<td>Sets the hyper reference for links</td>
</tr>
<tr>
<td><code>className</code></td>
<td>Sets the CSS class of the tag</td>
</tr>
<tr>
<td><code>style.color</code></td>
<td>Sets the CSS foreground color of the tag</td>
</tr>
<tr>
<td><code>style.backgroundColor</code></td>
<td>Sets the CSS background color of the tag</td>
</tr>
</tbody>
</table>

Most of the other CSS properties follow the same pattern as `color` and `backgroundColor`, namely that the properties are "camel-case" meaning that the first word is all lower-case and each successive word starts with an upper case letter. Also, more specific tags (like lists, tables and other higher-order formattings) include specialized function to easily add list items or table rows.

**Dynamic Effects**

The most common JavaScript events help trigger segments of JavaScript code on specified user actions. Here are a few of the most common events:
### Event Descriptions

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onload, onunload</td>
<td>Triggers when the user loads (enters) or unloads (leaves) the page</td>
</tr>
<tr>
<td>onsubmit</td>
<td>Triggers when a form is submitted via a &quot;submit&quot; button</td>
</tr>
<tr>
<td>onfocus, onblur</td>
<td>Triggers when a user tabs onto (or leaves) a link or form object</td>
</tr>
<tr>
<td>onchange</td>
<td>Triggers when a user changes the value of a form field</td>
</tr>
<tr>
<td>onmouseover, onmouseout</td>
<td>Triggers when the mouse cursor is positioned over (or leaves) an object</td>
</tr>
<tr>
<td>onclick, ondblclick</td>
<td>Triggers when the mouse is clicked (or doubleclicked) on an object</td>
</tr>
</tbody>
</table>

### Conclusion

From the exercises and material covered in this class, you should have a pretty good idea of the capabilities and limitations associated with JavaScript. Keep in mind a couple of key points:

- JavaScript is most useful to create dynamic effects on a website
- JavaScript can be turned off, so don’t require its use for any feature on your website (especially form validation). Sometimes this is referred to as progressive enhancement, if you are looking for further information about it on the web.
- Great references for JavaScript can be found online, specifically the Mozilla Developer Network, or MDN (https://developer.mozilla.org)

### Next Steps

Now that you have some experience with JavaScript, there are many other STS classes to look into:

#### JavaScript 2: jQuery and AJAX

This class expands on the topics covered in JavaScript 1 to introduce some pre-built JavaScript packages that make the creation of professional-looking effects much easier. In this class you will work to create effects like expanding/collapsing menus, clicking-and-dragging effects, and immersive user interactions called AJAX (asynchronous JavaScript and XML).

#### PHP/SQL in a Day

This class covers everything about PHP and SQL used for server-side programming in web pages. It is a good idea to get a general knowledge of both client-side, and server-side languages.