Java Script

Contents

[1 Introduction 1](#_Toc156381526)

[2 Java Script DOM Tutorial 2](#_Toc156381527)

[3 Simple Java Script Example 4](#_Toc156381528)

[4 Programming the HTML DOM 6](#_Toc156381529)

[4.1 Browser Object Model 6](#_Toc156381530)

[4.2 HTML DOM Objects 7](#_Toc156381531)

[5 Accessing Elements via the DOM 9](#_Toc156381532)

[5.1 TextBox 9](#_Toc156381533)

[5.2 Check Boxes 9](#_Toc156381534)

[5.3 Radio Buttons 10](#_Toc156381535)

[5.4 Drop-Down (Single Selection) 10](#_Toc156381536)

[5.5 Drop-Down (Multiple Selection) 11](#_Toc156381537)

[6 Events 12](#_Toc156381538)

[7 Expectations 12](#_Toc156381539)

[Appendix 1 N/A 13](#_Toc156381540)

# Introduction

Java Script (JS):

* Is an object-oriented; however, many programmers don’t use OO very much. With JS, you frequently write event-handlers that might call helper functions.
* It is a way to make a page interactive: update content dynamically, data validation, control multimedia, pop-up menus, *etc.*
* It is used on both the client-side and the server-side. We will only use it on the client-side. On the client-side, it runs in the browser.
* Can manipulate HTML elements, *e.g.* hide or show a paragraph of text, or build a table dynamically, *etc.* This is possible because of the HTML DOM (Document Object Model) which we consider a later [Section](#_Programming_the_HTML).

For this course, we will create Form based apps that work only in the browser. This approach is occasionally useful in the real world. However, it is a great approach to learn JS and how to manipulate the DOM.

that runs in the browser. Among other ways, JS can be used in these 3 ways:

1. As a stand-alone feature in a web page to provide some functionality, *e.g.* a monthly payment calculator.
2. Data validation, *e.g.* determining if a number a user entered is an integer between 1 and 10..
3. Manipulate HTML elements, *e.g.* hide or show a paragraph of text, or build a table dynamically. This would usually be done in the context of item 1 or 2 above.

We may cover the highlighted sections of the W3School’s [Java Script Tutorial](https://www.w3schools.com/js/default.asp) in class, or I may have you cover it outside of class.

|  |  |
| --- | --- |
| 1. JS HOME – Read carefully 2. JS Introduction – Read carefully 3. JS Where To – Read carefully 4. JS Output – Read carefully 5. JS Statements – Scan 6. JS Syntax – Read carefully 7. JS Comments – Scan 8. JS Variables – Scan 9. JS Operators – Scan 10. JS Arithmetic – Scan 11. JS Assignment - Scan 12. JS Data Types – Scan 13. JS Functions – Read carefully 14. JS Objects – Omit 15. JS Events – Read carefully 16. JS Strings – Scan 17. JS String Methods – Scan 18. JS Numbers – Scan 19. JS Number Methods – Scan 20. JS Arrays – Read carefully. Omit: “Adding Array Elements”, “Associative Arrays?” and everything below “The Difference Between Arrays and Objects” 21. JS Array Methods – Scan 22. JS Array Sort – Scan 23. Js Array Iteration – Scan, this shows newer ways, we will use syntax similar to java 24. JS Dates – Omit 25. JS Date Formats - Omit 26. JS Date Get Methods – Omit 27. JS Date Set Methods – Omit | 1. JS Array Methods – Omit 2. JS Math – Scan 3. JS Random - Omit 4. JS Booleans – Scan 5. JS Comparisons – Scan 6. JS Conditions – Scan 7. JS Switch – Scan 8. JS Loop For – Scan 9. JS Loop While – Scan 10. JS Break – Scan 11. JS Type Conversion – Omit 12. JS Bitwise – Omit 13. JS RegExp – Omit 14. JS Errors – Omit 15. JS Scope – Read carefully 16. JS Hoisting – Omit 17. JS Strict Mode – Omit 18. JS this Keyword – OmitJS Let – Omit 19. JS Const - Omit 20. JS Debugging – Omit 21. JS Style Guide – Scan 22. JS Best Practices – Omit 23. JS Mistakes – Omit 24. JS Performance – Omit 25. JS Reserved Words – Omit 26. JS Versions – Omit 27. JS Versions ES5 – Omit 28. JS Versions ES6 - Omit 29. JS JSON – Omit. |

# Java Script DOM Tutorial

We may cover the highlighted sections of the W3School’s DOM and BOM tutorials in class, or I may have you cover it outside of class.

|  |  |
| --- | --- |
| [JavaScript HTML DOM Tutorial](http://www.w3schools.com/js/js_htmldom.asp)   1. DOM Intro – Read carefully 2. DOM Methods – Read carefully 3. DOM Document – Read carefully. Omit “Adding and Deleting Elements” and “Finding HTML Objects”. 4. DOM Elements – Read carefully. Omit last two sections starting with, “Finding HTML Elements by CSS Selectors .” 5. DOM HTML – Read carefully. 6. DOM CSS – Read carefully only the second section, “Using Events” 7. DOM Animations – Omit 8. DOM Events – Read carefully 9. DOM EventListener – Useful, interesting, and powerful, but you can omit. 10. DOM Navigation – Omit 11. DOM Nodes – Omit 12. DOM Collections - Omit 13. DOM Node Lists – Omit | [JavaScript Browser BOM Tutorial](http://www.w3schools.com/js/js_window.asp)   1. JS Window – Read carefully two sections only. Scan from, “Window Size” and below 2. JS Screen – Omit 3. JS Location – Omit 4. JS History – Omit 5. JS Navigator – Omit 6. JS Popup Alert – Read carefully 7. JS Timing – Omit 8. JS Cookies – Omit |

# Simple Java Script Example

Consider this example where a user types in a name, and then a “hello world” message is displayed in different ways: text area, paragraph, div, table.

|  |  |
| --- | --- |
|  | The Java Script is:  function helloWorld() {  var txbName = document.getElementById("name");  var message = txbName.value + ", " + "Hello World"  alert(message);  var txaResult = document.getElementById("txaMessage");  txaResult.value = message;  document.getElementById("pMessage").innerHTML = message;  document.getElementById("divMessage").innerHTML = message;  document.getElementById("tdMessage").innerHTML = message;  txbName.value = null;  txbName.focus();  } |

The HTML is:

<form id="hello">

<p>

What is your name?

<input id="name" type="text" />

</p>

<p><button type="button" onclick="helloWorld()">Hello World</button></p>

<p><textarea rows="4" id="txaMessage" cols="30"></textarea></p>

<p id="pMessage"></p>

<div id="divMessage" style="border:2px solid #0066FF; display:inline-block;

margin-bottom:10px"></div>

<table border="1">

<tr>

<td>Message</td>

<td id="tdMessage"></td>

</tr>

</table>

<div id="divMessage" style="border:2px solid #0066FF; display:none; margin-bottom:10px"></div>

</form>

Takeaways:

|  |  |
| --- | --- |
| **Statement** | **Description** |
| document.getElementById(id) | Returns the HTML element with *id*. |
| .value | Used to get or set the value in a form control: input(type=”text”), textarea, select, & others. |
| .innerHTML | Get or set the HTML content of tags that are not form controls. For example: p, div, td. In the example above, we have this line to put the message in the paragraph:  document.getElementById("pMessage").innerHTML = message;  We could have supplied some HTML as well as the message:  document.getElementById("pMessage").innerHTML = "<b>"+message+"</b>"; |
| .innerText | Get or set the text content of tags that are not form controls. For example: p, div, td. In the example above, we could have used: *innerText* instead of *innerHTML*:  document.getElementById("pMessage").innerText = message; |

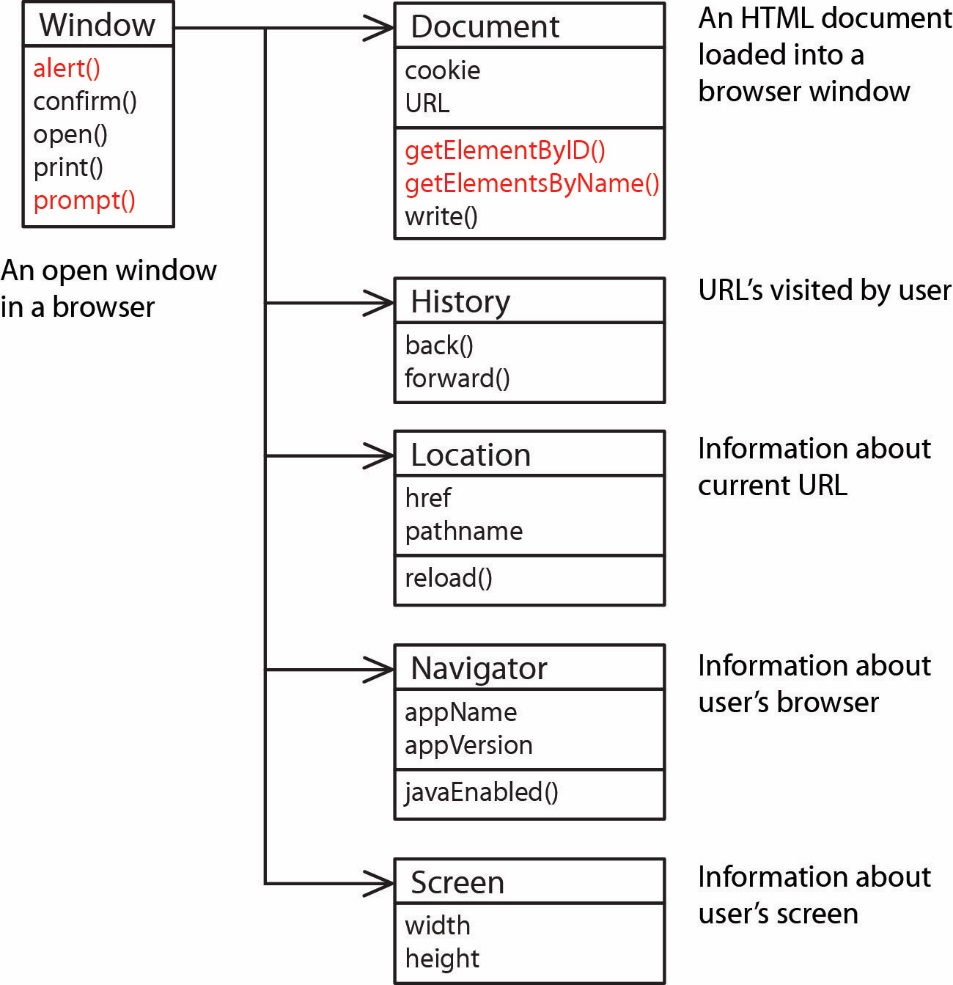
# Programming the HTML DOM

This is an introduction to HTML DOM (document object model) programming and briefly the HTML BOM (browser object model). The DOM (and BOM) are API’s that provide JavaScript (JS) code access to any HTML elements on a webpage. For example, we can access the values a user types in (or selects), process them, and then display output.

## Browser Object Model

Practically anything you could want to know about a web page is available in HTML BOM (Browser Object Model). An instance of each of these classes is available in any webpage. We will use the methods shown in red. The class diagram is not complete. There are many more properties and methods for each class. Reference:

<https://www.w3schools.com/js/js_window.asp>



In the example in [Section3](#_Simple_Java_Script), we used the alert() method, e.g.

alert("Hello World");

However, we could have written this:

window.alert("Hello World");

Thus, we see that JS is pretty loose about some things: e.g. the *window* object is available in any page and since there is only one *alert* method, JS knows what to do. Similarly, we used this to obtain a reference to the textbox with the id “name”:

var txbName = document.getElementById("name");

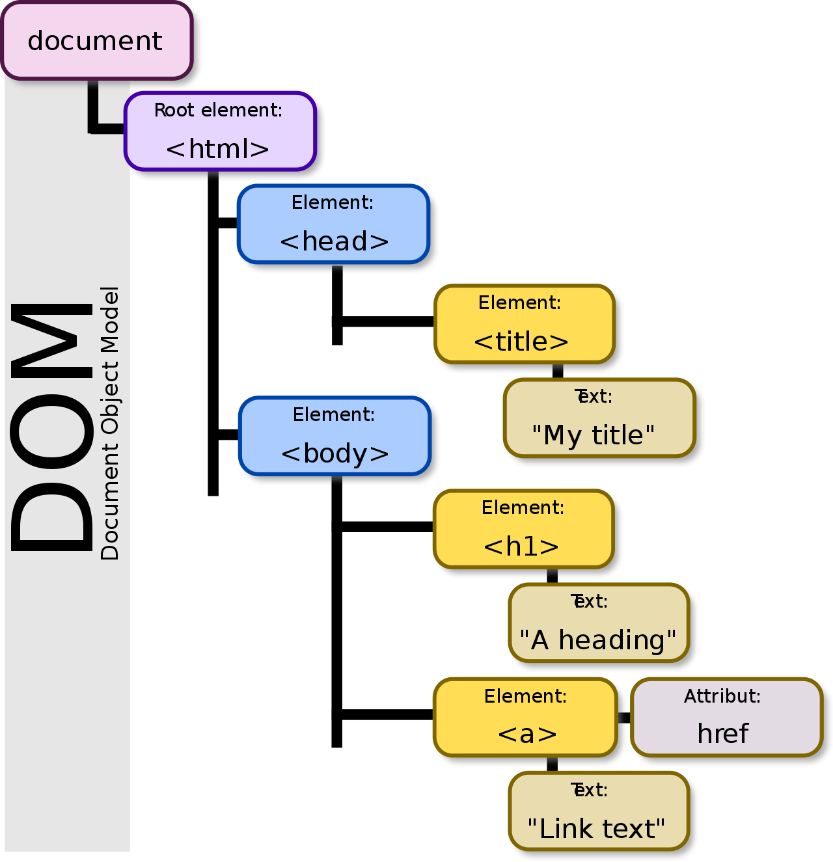
Note that the line above gave a reference to the textbox itself. The line below shows that we use the textbox’s *value* property to obtain the value that was typed in. This will be explained a bit more later.

var name = txtName.value;

And, we could have used *window* to precede *document,* but usually don’t.

var txbName = window.document.getElementById("name");

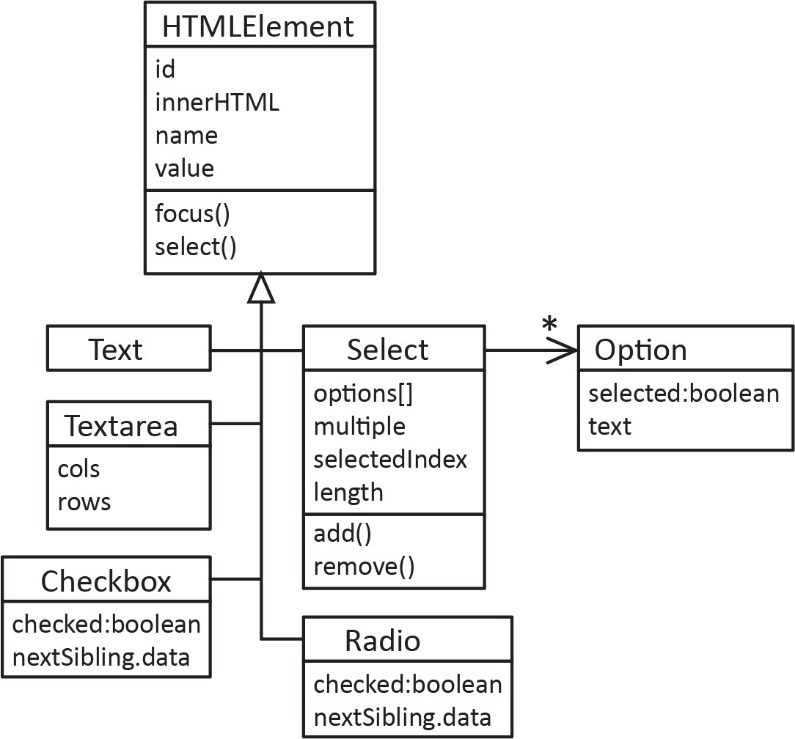
## HTML DOM Objects

The *Document Object Model* (DOM) is an API for accessing, changing, and manipulating all aspects of an HTML (or XML) document. The DOM represents an HTML page as a tree. Each branch of the tree ends in an HTML element, elements can contain elements, and elements can contain attributes and text.

By Birger Eriksson - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=18034500

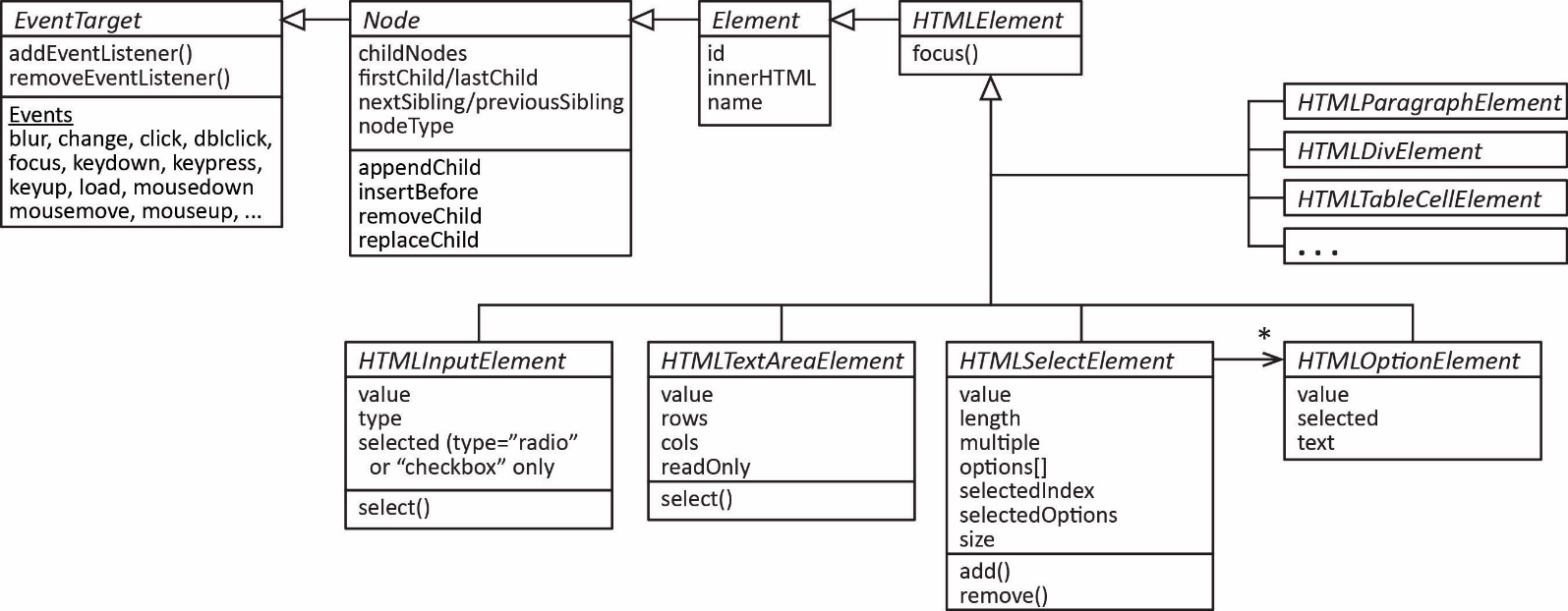
You can see a visualization of the DOM for a page on this site: <https://bioub.github.io/dom-visualizer/>. You can also turn on the Developer Tools (select the 3 vertical dots in upper-right of Chrome, choose: More Tools, Developer Tools and select the Elements tab.

We can use JavaScript to access the DOM methods to change the page. When an HTML page is rendered, the browser also creates a DOM object which is an object-oriented representation of the page. Each element on the page has properties, methods, and events. Here, we are interested mostly in the form controls (button, text box, text area, check box, radio button, drop down); however, we will also access paragraphs, divs, and table cells. An abbreviated (somewhat generalized) class diagram is shown below:



Note:

* We see that all the classes share a number of common properties and methods (every object has an *id*, *name*, and *value*, *innerHTML, etc.*).
* Each of these classes corresponds to an HTML tag. For instance, *input type=text* is represented by the *TextBox* class; however, I have used simplified names, not the actual class names.
* A fuller class diagram, with the actual interface names is shown below.

****

An excellent reference for the DOM:

<https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model>

An excellent high-level discussion of the DOM:

<https://web.stanford.edu/class/cs98si/slides/the-document-object-model.html>

# Accessing Elements via the DOM

## TextBox

The first question to address is where do we get the references for the objects in the DOM? The most common way is to use the *getElementById* method in the Document class. Every element should have a unique Id.

Example: An HTML text box is defined:

**<input type="text" id="txtName">**

Then, in JS, we could obtain a reference to the textbox:

**var tbox = document.getElementById("txtName");**

Then, we could access the value typed in:

**var name = tbox.value;**

Or, we could set text into the textbox:

**tbox.value = "what is your name?";**

Or we could set the focus and select the text in a text box:

**tbox.focus();**

**tbox.select();**

Note: your source code does not change, just the DOM. In other words, after you execute the JS, and view the page source, it does not show an element where you have set the innerHTML. Can right-click element, and choose: Inspect to see.

## Check Boxes

Often, it is more convenient to process check boxes as a group. For instance, we may have check boxes where a user can select any number of deserts they want:

**<input type="checkbox" name="chkDesert" value="1.25"/>Ice Cream ($1.25)<br>**

**<input type="checkbox" name="chkDesert" value="2.50"/>Apple Pie ($2.50)<br>**

**<input type="checkbox" name="chkDesert" value="0.75"/>Cookies ($0.75)<br>**

Notice that the check boxes all have the same *name* attribute. This is convenient because we can now access them as an array of check boxes using the *getElementsByName(name)* method.

**var chkDes = document.getElementsByName("chkDesert");**

And, a general-purpose loop to process all the check boxes:

**for (i=0; i<chkDes.length; i++) {**

**if ( chkDes[i].checked ) {**

**// Do something useful with chkDes[i].value**

**}**

**}**

## Radio Buttons

Radio buttons are similar:

**<input type="radio" value="1.00" name="optDrink" id="d1"/>Soda ($1.00)<br>**

**<input type="radio" value="2.00" name="optDrink" id="d2"/>Beer ($2.00)<br>**

**<input type="radio" value="0.75" name="optDrink" id="d3"/>Coffee ($0.75)**

Notice again that the radio buttons all have the same *name* attribute. Now, we access them as an array of radio buttons in JS:

**var optD = document.getElementsByName("optDrink");**

And, a general-purpose loop to process all the radio buttons:

**for (i=0; i<optD.length; i++) {**

**if ( optD[i].checked ) {**

**// Do something useful with optD[i].value**

**break;**

**}**

**}**

## Drop-Down (Single Selection)

A drop-down (single selection) for a user to select an interest rate might be defined like this:

**<select size="1" id="ddIntRate">**

**<option value="0.040">4.00%</option>**

**<option value="0.045">4.50%</option>**

**<option value="0.050">5.00%</option>**

**</select>**

To access the drop down:

**var dd = document.getElementById("ddIntRate");**

We can access the *value* of the selected item directly by using the *value* property:

**alert(dd.value);**

We can also obtain the *index* of the item that is selected by using the *selectedIndex* property (zero-based):

**alert(dd.selectedIndex);**

We remember that the Select class also has an *options* collection. Thus, an alternate way to obtain the value of the selected item is:

**alert(dd.options[dd.selectedIndex].value);**

Similarly, we can drill down into the *options* collection to see the individual items:

alert(dd.options[dd.selectedIndex].selected);

alert(dd.options[dd.selectedIndex].text);

alert(dd.options[dd.selectedIndex].value);

Due to a feature in JS, we can write the code above, shorter, as shown below. The two sets are equivalent. Essentially, treating the object reference as an array itself automatically drills down into the *options* collection. In .NET languages this is called an *indexer*. I’m not sure what it is called in JS.

alert(dd[dd.selectedIndex].selected);

alert(dd[dd.selectedIndex].text);

alert(dd[dd.selectedIndex].value);

## Drop-Down (Multiple Selection)

A drop-down (multiple selection) for a user to select multiple food items might be defined like this:

<select size="4" id="ddFood" multiple>

<option value="3.25">Hamburger ($3.25)</option>

<option value="2.75">Tofo Burger ($2.75)</option>

<option value="1.00">Soda ($1.00)</option>

<option value="2.00">Beer ($2.00)</option>

<option value="4.00">Wings ($4.00)</option>

<option value="0.50">Cookie ($0.50)</option>

</select>

And, to access the drop down:

var dd = document.getElementById("ddFood");

And, a general-purpose loop to process all the selections:

for( i=0; i<dd.length; i++ ) {

if ( dd.options[i].selected ) {

// Do something useful with dd.options[i].value

}

}

There is a *selectedoptions* property that contains just the selected options. Thus,

for(i=0; i<dd.selectedoptions.length; i++ ) {

selOption = dd.selectedoptions[i]

// Do something useful with selOption.value, etc

}

# Events

Form elements can fire events. For instance, when a user clicks on a button, an event is fired. An event is handled by an *event handler*, which is usually a call to a JS function.

Example, a button might be defined like this:

**<input type="button" value="Enter" id="btnCalc" onclick="helloWorld()">**

The *onClick* attributes specifies the event handler. In this case, it is a call to the *helloWorld* JS function which resides in the *head* section of the HTML:

**function helloWorld() {**

**window.alert("Hello World");**

**}**

The available events are shown below (copied from W3Schools). The most common ones have been highlighted.

|  |  |
| --- | --- |
| **Attribute** | **The event occurs when...** |
| [onblur](http://www.w3schools.com/jsref/event_onblur.asp) | An element loses focus |
| [onchange](http://www.w3schools.com/jsref/event_onchange.asp) | The content of a field changes |
| [onclick](http://www.w3schools.com/jsref/event_onclick.asp) | Mouse clicks an object |
| [ondblclick](http://www.w3schools.com/jsref/event_ondblclick.asp) | Mouse double-clicks an object |
| [onerror](http://www.w3schools.com/jsref/event_onerror.asp) | An error occurs when loading a document or an image |
| [onfocus](http://www.w3schools.com/jsref/event_onfocus.asp) | An element gets focus |
| [onkeydown](http://www.w3schools.com/jsref/event_onkeydown.asp) | A keyboard key is pressed |
| [onkeypress](http://www.w3schools.com/jsref/event_onkeypress.asp) | A keyboard key is pressed or held down |
| [onkeyup](http://www.w3schools.com/jsref/event_onkeyup.asp) | A keyboard key is released |
| [onload](http://www.w3schools.com/jsref/event_onload.asp) | A page or image is finished loading |
| [onmousedown](http://www.w3schools.com/jsref/event_onmousedown.asp) | A mouse button is pressed |
| [onmousemove](http://www.w3schools.com/jsref/event_onmousemove.asp) | The mouse is moved |
| [onmouseout](http://www.w3schools.com/jsref/event_onmouseout.asp) | The mouse is moved off an element |
| [onmouseover](http://www.w3schools.com/jsref/event_onmouseover.asp) | The mouse is moved over an element |
| [onmouseup](http://www.w3schools.com/jsref/event_onmouseup.asp) | A mouse button is released |
| [onresize](http://www.w3schools.com/jsref/event_onresize.asp) | A window or frame is resized |
| [onselect](http://www.w3schools.com/jsref/event_onselect.asp) | Text is selected |
| [onunload](http://www.w3schools.com/jsref/event_onunload.asp) | The user exits the page |

# Expectations

These are expectations I have of you for testing:

1. Given a description of client-side dynamic behavior write the HTML and JavaScript to implement it. An abbreviated version of the HTML DOM is provided.
2. Apply the onclick event and write a corresponding event handler.
3. Write code to obtain input from form elements (text box, text area, drop-down, check boxes, radio buttons)
4. Display output to a form element, div, paragraph, text area *etc*.

Appendix

1. N/A

.