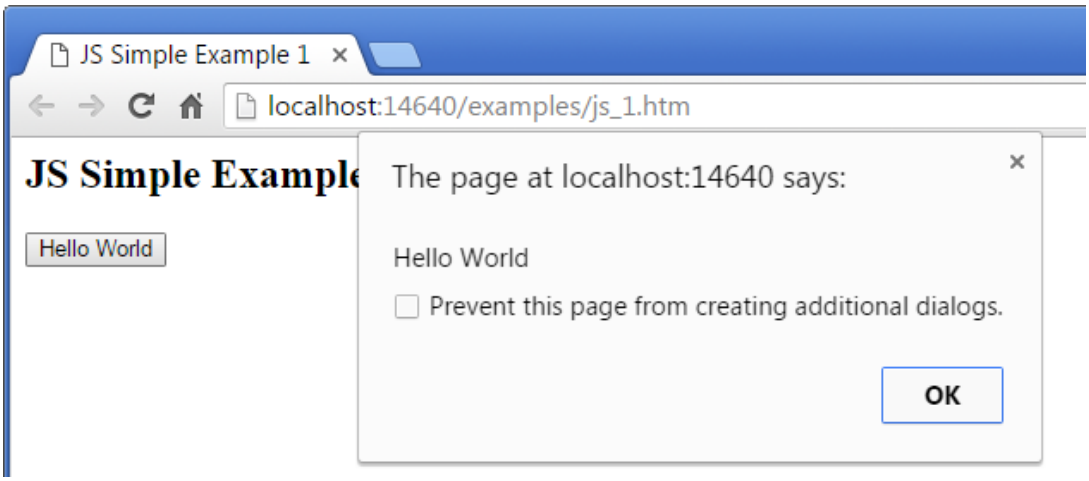
# JavaScript Notes

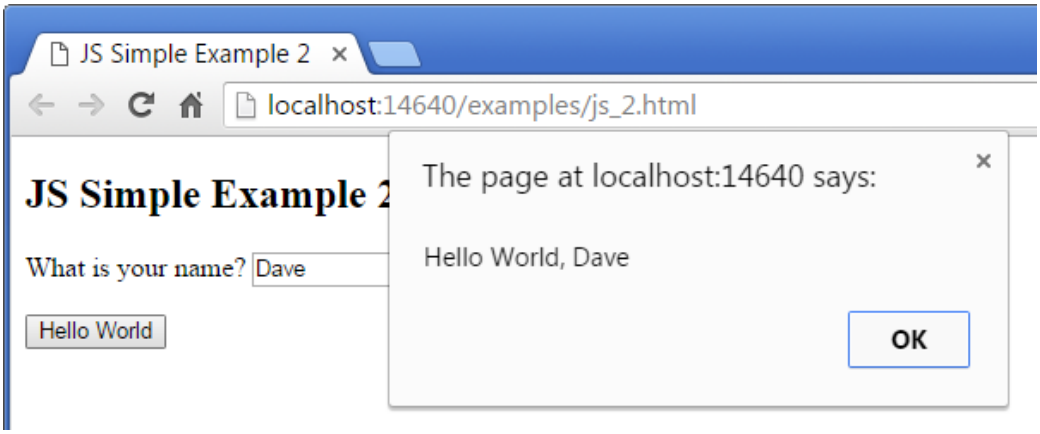
## Event Driven Programming

1. Example 1 – Illustrates the event-driven paradigm: press the button and display a message in a popup.




```
<html>
<head>
    <title>JS Simple Example 1</title>

    <script type="text/javascript">
        function helloWorld() {
            alert("Hello World");
        }
    </script>

</head>

<body>
    <h2>JS Simple Example 1</h2>
    <form id="hello">
        <button type="button" onclick="helloWorld()">Hello World</button>
    </form>

</body>
</html>
```

2. Example 2 – Illustrates taking input from a textbox and displaying a message in a popup.



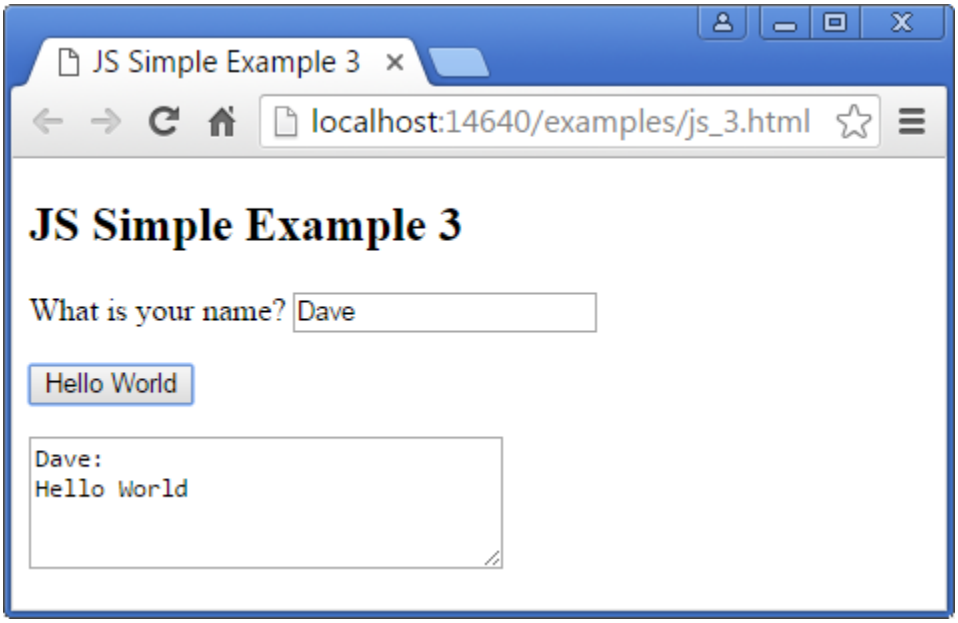

JavaScript

```
function helloWorld() {
    var txtName = document.getElementById("name");
    var name = txtName.value;
    alert("Hello World, " + name);
}
```

HTML

```
<form id="hello">
    <p>What is your name?
        <input id="name" type="text" />
    </p>
    <button type="button" onclick="helloWorld()">Hello World</button>
</form>
```

3. Example 3 – Illustrates displaying results in a TextArea.




JavaScript

```
function helloWorld() {
    var name = document.getElementById("name").value;
    var message = name + ":" + "\n" + "Hello World"
    var txaResult = document.getElementById("message");
    txaResult.value = message;
}
```

HTML

```
<form id="hello">
    <p>What is your name? <input id="name" type="text" /></p>

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

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

</form>
```

4. Example 4 – Illustrates displaying result in a Paragraph, Div, and Table

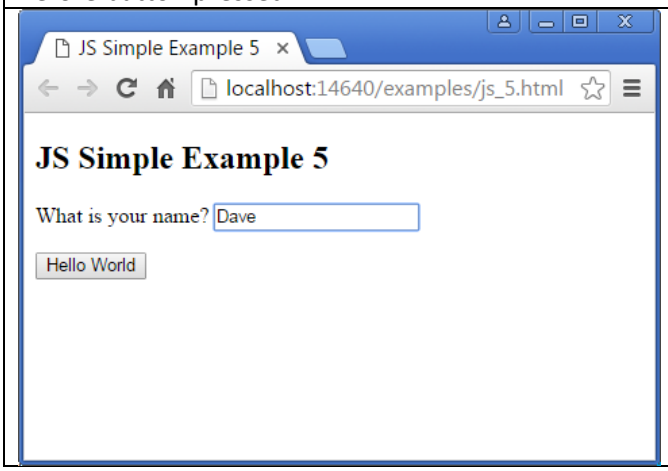
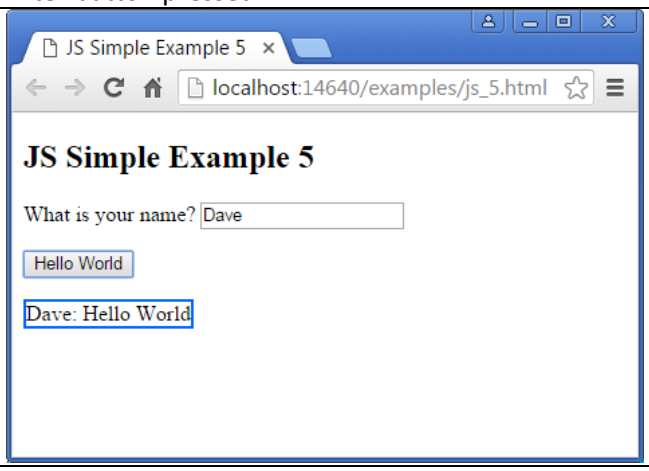

JavaScript

```
function helloWorld() {
    var name = document.getElementById("name").value;
    var message = name + ", " + "Hello World"
    document.getElementById("pMessage").innerHTML = message;
    document.getElementById("divMessage").innerHTML = message;
    document.getElementById("tdMessage").innerHTML = message;
}
```

HTML

```
<h2>JS Simple Example 4</h2>
<form id="hello">

    <p>What is your name?
        <input id="name" type="text" />
    </p>

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

    <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>
</form>
```

5. Example 5 – Illustrates displaying result in a Div which is initially hidden

| Before button pressed | After button pressed |
|---|---|
|  |  |



JavaScript

```
function helloWorld() {
    var name = document.getElementById("name").value;
    var message = name + ":" + "\n" + "Hello World"
    var div = document.getElementById("divMessage");
    div.innerHTML = message;
    div.style.display = "inline-block";
}
```

HTML

```
<h2>JS Simple Example 5</h2>
<form id="hello">

    <p>What is your name?
        <input id="name" type="text" />
    </p>

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

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

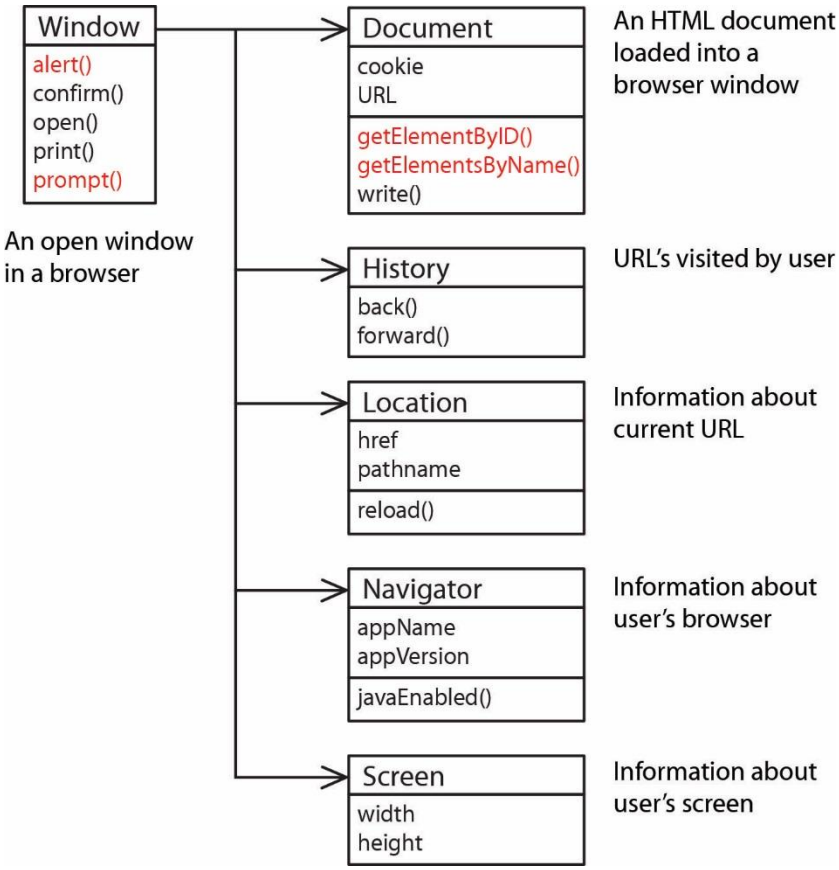
6. Now that we have seen some basic examples we will go into more depth.

## 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 Objects

1. 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.




2. In the examples above we used the alert() method, e.g.

```
alert("Hello World");
```

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 txtName = document.getElementById("name");
```

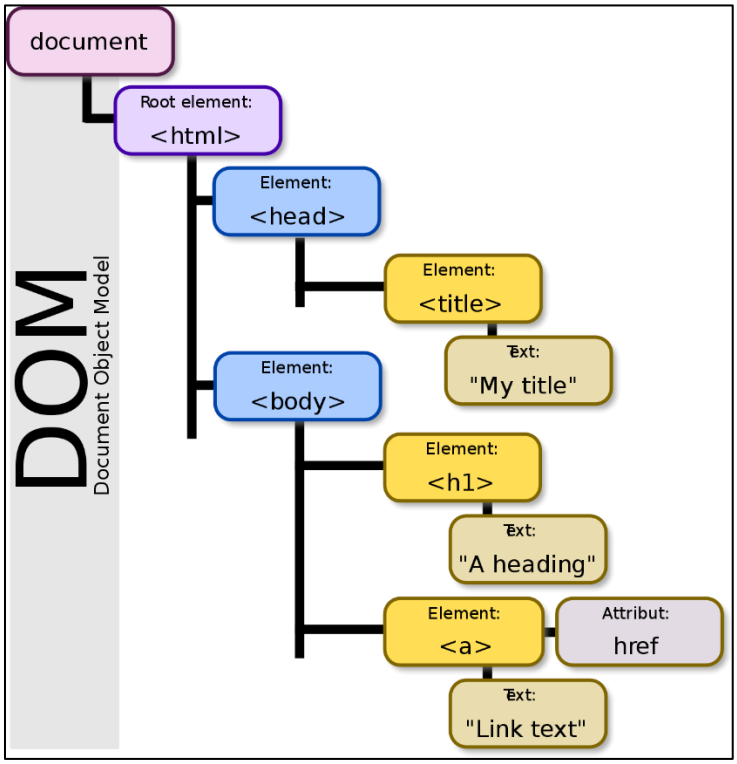
Note that the line above gave a reference to the textbox itself. The line below shows that we use the textbox’s *value* property to obtain the value that was typed in. This will be explained a bit more later.

```
var name = txtName.value;
```

And, as with *window.alert(…)* above, we could have preceded *document.getElementById(“name”)* with *window,* but usually don’t.

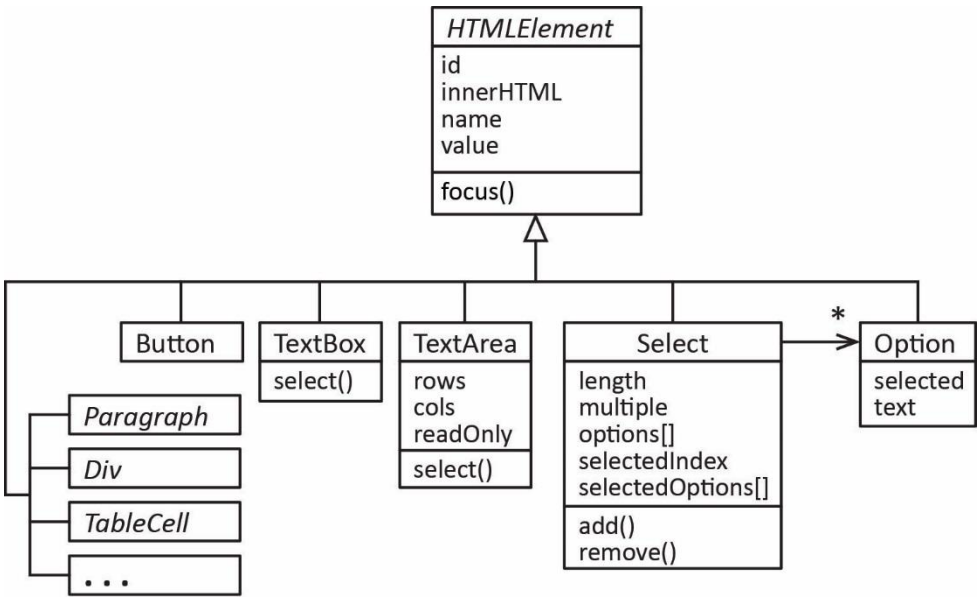
## HTML DOM Objects

3. 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 a node, and each node contains objects.



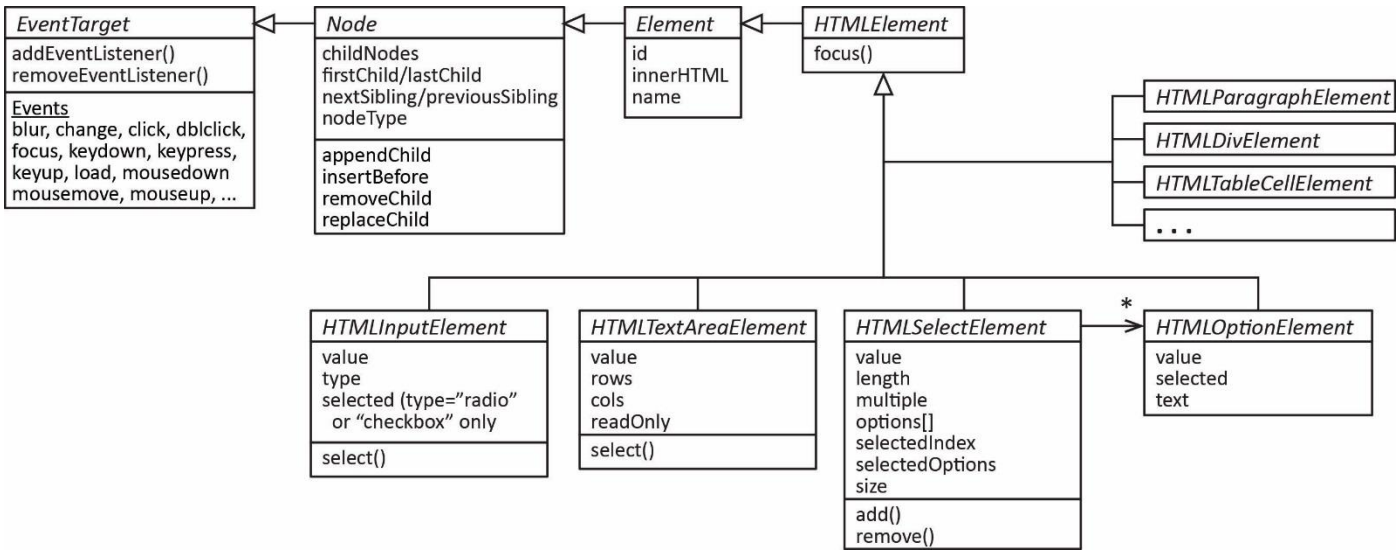

By Birger Eriksson - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=18034500

4. We can use JavaScript to access the DOM methods to change the page. When an HTML page is rendered, the browser also creates DOM object which is an object-oriented representation of the 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*, *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.




5. An excellent reference for the DOM is:

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

## Getting Elements

6. 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.

7. Example: An HTML text box is defined:

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

Then, in JS, we could place a value in a textbox with code like this:

```
var tbox = document.getElementById("txtName");
tbox.value = "what is your name?";
```

Or access the value in a text box:

```
var tbox = document.getElementById("txtName");
var name = tbox.value;
```

Or to set the focus and select the text in a text box:

```
tbox.focus();
tbox.select();
```

## Check Boxes

8. 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 in JS:

```
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

9. 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)

10. 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:

```
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)

11. 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
       }
}
```

12. 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

13. 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.

14. 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");
}
```

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

| Attribute | The event occurs when... |
|---|---|
| onblur | An element loses focus |
| onchange | The content of a field changes |
| onclick | Mouse clicks an object |
| ondblclick | Mouse double-clicks an object |
| onerror | An error occurs when loading a document or an image |
| onfocus | An element gets focus |
| onkeydown | A keyboard key is pressed |
| onkeypress | A keyboard key is pressed or held down |
| onkeyup | A keyboard key is released |
| onload | A page or image is finished loading |
| onmousedown | A mouse button is pressed |
| onmousemove | The mouse is moved |
| onmouseout | The mouse is moved off an element |
| onmouseover | The mouse is moved over an element |
| onmouseup | A mouse button is released |
| onresize | A window or frame is resized |
| onselect | Text is selected |
| onunload | The user exits the page |

## Examples

See the examples that are provided on Client-Side Web Programming page (*back* in browser)

## Expectations

You have completed the required reading on JavaScript and these are the expectations for this topic:

| Expectations |
| --- |
| 1. Given a description of client-side dynamic behavior write the HTML and JavaScript to implement which uses the abbreviated version of the HTML DOM that has been provided.<br>2. Apply the onclick event and write a corresponding event handler.<br>3. Write code to obtain input from form elements<br>4. Display output to a form element, div, paragraph, *etc*. |