

# Introduction to JavaScript

(JSC101 version 2.2.1)

## Copyright Information

© Copyright 2017 Webucator. All rights reserved.

## The Authors

### **Nat Dunn**

Nat Dunn founded Webucator in 2003 to combine his passion for web development with his business expertise and to help companies benefit from both.

### **Brian Hoke**

Brian Hoke is Principal of Bentley Hoke, a web consultancy in Syracuse, New York. The firm serves the professional services, education, government, nonprofit, and retail sectors with a variety of development, design, and marketing services. Core technologies for the firm include PHP and Wordpress, JavaScript and jQuery, Ruby on Rails, and HTML5/CSS3. Previously, Brian served as Director of Technology, Chair of the Computer and Information Science Department, and Dean of Students at Manlius Pebble Hill School, an independent day school in DeWitt, NY. Before that, Brian taught at Insitut auf dem Rosenberg, an international boarding school in St. Gallen, Switzerland. Brian holds degrees from Hamilton and Dartmouth colleges.

## Accompanying Class Files

This manual comes with accompanying class files, which your instructor or sales representative will point out to you. Most code samples and exercise and solution files found in the manual can also be found in the class files at the locations indicated at the top of the code listings.

Due to space limitations, the code listings sometimes have line wrapping, where no line wrapping occurs in the actual code sample. This is indicated in the manual using three greater than signs: `>>>` at the beginning of each wrapped line.

In other cases, the space limitations are such that we have inserted a forced line break in the middle of a word. When this occurs, we append the following symbol at the end of the line before the actual break: `»»`

# Table of Contents

<b>1. JavaScript Basics.....</b>	<b>1</b>
The Name "JavaScript".....	1
What is ECMAScript?.....	1
The HTML DOM.....	2
JavaScript Syntax.....	2
Basic Rules.....	2
Accessing Elements.....	3
Dot Notation.....	3
Square Bracket Notation.....	3
Where Is JavaScript Code Written?.....	4
The "javascript" Pseudo-Protocol.....	6
JavaScript Objects, Methods and Properties.....	8
Methods.....	9
Properties.....	10
The Implicit window Object.....	10
<i>Exercise 1: Alerts, Writing, &amp; Changing Background Color.....</i>	<i>12</i>
<b>2. Variables, Arrays and Operators.....</b>	<b>17</b>
JavaScript Variables.....	17
A Loosely-typed Language.....	17
Variable Naming.....	18
Storing User-Entered Data.....	18
<i>Exercise 2: Using Variables.....</i>	<i>21</i>
Constants.....	23
Arrays.....	24
<i>Exercise 3: Working with Arrays.....</i>	<i>28</i>
Associative Arrays.....	31
Array Properties and Methods.....	31
JavaScript Operators.....	33
<i>Exercise 4: Working with Operators.....</i>	<i>38</i>
<b>3. JavaScript Functions.....</b>	<b>43</b>
Global Functions.....	43
Number(object) .....	43
String(object) .....	44
isNaN(object).....	46
parseFloat() and parseInt().....	47
<i>Exercise 5: Working with Global Functions.....</i>	<i>49</i>
User-defined Functions.....	54
Function Syntax.....	54
Passing Values to Functions.....	55
A Note on Variable Scope.....	57
<i>Exercise 6: Writing a JavaScript Function.....</i>	<i>58</i>
Returning Values from Functions.....	64

---

## Table of Contents

<b>4. Event Handlers.....</b>	<b>65</b>
Event Handlers.....	65
The getElementById() Method.....	67
<i>Exercise 7: Using Event Handlers.....</i>	<i>69</i>
Dot Notation and Square Bracket Notation.....	74
CSS Selectors.....	74
Type Selectors.....	75
Descendant Selectors.....	75
Child Selectors.....	75
Class Selectors.....	76
ID Selectors.....	77
Attribute Selectors.....	77
The Universal Selector.....	78
Grouping.....	78
querySelector().....	78
<i>Exercise 8: Working with querySelector().....</i>	<i>81</i>
Accessing Element and Text Nodes Hierarchically.....	84
<i>Exercise 9: Working with Hierarchical Node Properties.....</i>	<i>87</i>
<b>5. Built-In JavaScript Objects.....</b>	<b>91</b>
String.....	91
Math.....	93
Date.....	95
The typeof Operator.....	99
Helper Functions.....	99
<i>Exercise 10: Returning the Day of the Week as a String.....</i>	<i>101</i>
<b>6. Conditionals and Loops.....</b>	<b>105</b>
Conditionals.....	105
if - else if - else Conditions.....	105
Switch / Case.....	112
<i>Exercise 11: Conditional Processing.....</i>	<i>116</i>
Loops.....	120
while Loop Syntax.....	120
do...while Loop Syntax.....	120
for Loop Syntax.....	120
for...in Loop Syntax.....	121
for...of Loop Syntax.....	121
<i>Exercise 12: Working with Loops.....</i>	<i>125</i>

<b>7. JavaScript Form Validation.....</b>	<b>131</b>
Accessing Form Data.....	131
<i>Exercise 13: Textfield to Textfield.....</i>	<i>134</i>
Basics of Form Validation.....	138
Cleaner Validation.....	140
<i>Exercise 14: Validating a Registration Form.....</i>	<i>144</i>
Validating Radio Buttons.....	150
Validating Check Boxes.....	153
Validating Select Menus.....	155
Focus, Blur, and Change Events.....	157
Focus and Blur.....	157
Change.....	159
Validating Textareas.....	161
<i>Exercise 15: Improving the Registration Form.....</i>	<i>163</i>
<b>8. The HTML Document Object Model.....</b>	<b>171</b>
The innerHTML Property.....	172
Accessing Element Nodes.....	172
getElementById().....	172
getElementsByName().....	174
getElementsByClassName().....	176
querySelectorAll().....	176
querySelector().....	176
<i>Exercise 16: Accessing Elements.....</i>	<i>178</i>
Accessing Attribute Nodes.....	183
getAttribute().....	183
attributes[].....	183
hasAttribute().....	183
setAttribute().....	183
removeAttribute().....	184
Removing Nodes from the DOM.....	184
Creating New Nodes.....	184
Identifying the Target of an Event.....	189
<i>Exercise 17: Manipulating the DOM.....</i>	<i>191</i>
<b>9. CSS Object Model.....</b>	<b>197</b>
Changing CSS with JavaScript.....	197
Hiding and Showing Elements.....	200
<i>Exercise 18: Showing and Hiding Elements.....</i>	<i>204</i>
Manipulating Tables.....	208
<i>Exercise 19: Tracking Results in the Math Quiz.....</i>	<i>212</i>
Dynamically Changing Dimensions.....	217
Creating a Timed Slider.....	219
Positioning Elements Dynamically.....	221
Creating a Different Timed Slider.....	223
<i>Exercise 20: Changing the Math Quiz Timer to a Slider.....</i>	<i>226</i>
Changing the Z-Index.....	229
The CSS Object Model.....	231

---

## Table of Contents

<b>10. Images, Windows and Timers.....</b>	<b>235</b>
Preloading Images.....	235
Windows.....	237
Timers.....	240
<i>Exercise 21: Popup Timed Slide Show.....</i>	<i>243</i>
<b>11. Debugging and Testing with Chrome.....</b>	<b>247</b>
Chrome DevTools.....	247
The Panels.....	247
The Elements Panel.....	249
The Console Panel.....	250
<i>Exercise 22: Using the Chrome DevTools "Elements" and "Console" Panels.....</i>	<i>252</i>
The Sources Panel.....	257
<i>Exercise 23: Using the Chrome DevTools "Sources" Panel.....</i>	<i>260</i>
Other DevTools Panels.....	264
Chrome DevTools API and Extensions.....	264
<i>Exercise 24: Accessibility Developer Tools Extension.....</i>	<i>265</i>
<b>Appendix 1. Navigator, History, and Location Objects.....</b>	<b>267</b>
The navigator Object.....	267
Checking for Disabled Features.....	271
Feature Detection.....	272
Modernizr.....	273
How Modernizr Works.....	274
The history Object.....	275
The location Object.....	278
<i>Exercise 25: Creating a Simple Quiz.....</i>	<i>281</i>

# 1. JavaScript Basics

## In this lesson, you will learn...

1. To work with the HTML DOM.
2. To follow JavaScript syntax rules.
3. To write JavaScript inline.
4. To write JavaScript in script blocks.
5. To create and link to external JavaScript files.
6. To work with JavaScript objects, methods, and properties.
7. To reference HTML elements with JavaScript.

## 1.1 The Name "JavaScript"

In this manual, we refer to the language we are learning as *JavaScript*, which is what it is usually called. However, *JavaScript* was invented by Netscape Communications and is now [owned by Oracle Corporation](#)<sup>1</sup>. Microsoft calls its version of the language *JScript*. JavaScript and JScript are both implementations of *EcmaScript*, but everyone still refers to the language as JavaScript.

## 1.2 What is ECMAScript?

ECMAScript, sometimes abbreviated as "ES", is a scripting language specification maintained and trademarked by [Ecma International \(http://www.ecma-international.org/memento/index.html\)](http://www.ecma-international.org/memento/index.html), a Europe-based industry association dedicated to technology and communications standards. The specification for the most-recent standard version of ECMAScript - "ES6" or, as it is officially known, "ECMA-262 6th Edition, The ECMAScript 2015 Language Specification," can be found at:

<http://www.ecma-international.org/ecma-262/6.0/>

As we mentioned above, JavaScript - the scripting language you are learning here and whose code is run by the browsers you (or others) use to visit the pages you build - is an implementation of ECMAScript.

Keep in mind that ECMAScript evolves over time: new features are added, new syntax is adopted, etc. Like CSS, HTML, and other client-side technologies, JavaScript is an implementation of a standard (ECMAScript) by browsers - please be aware that all browsers won't implement (or implement in the same manner) all

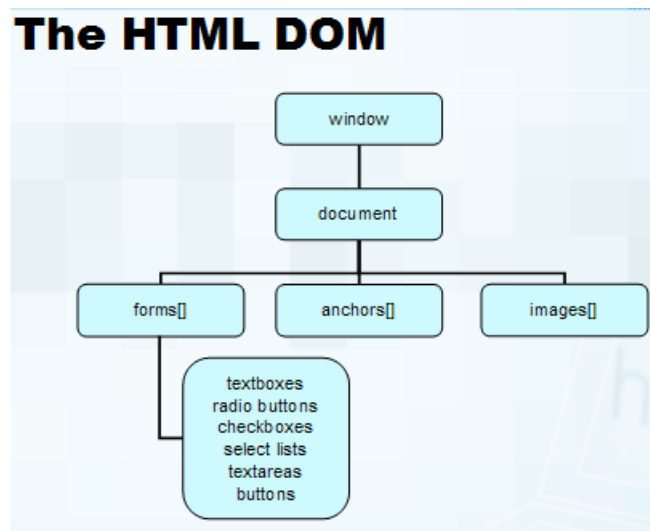
---

1. See <http://en.wikipedia.org/wiki/JavaScript#Trademark>.

newer features of ECMAScript, and that later versions of browsers will implement newer features over time.

## 1.3 The HTML DOM

The HTML Document Object Model (DOM) is the browser's view of an HTML page as an object hierarchy, starting with the browser window itself and moving deeper into the page, including all of the elements on the page and their attributes. Below is a simplified version of the HTML DOM.



As shown, the top-level object is window. The document object is a child of window and all the objects (i.e., elements) that appear on the page (e.g., forms, links, images, tables, etc.) are descendants of the document object. These objects can have children of their own. For example, form objects generally have several child objects, including text boxes, radio buttons, and select menus.

## 1.4 JavaScript Syntax

### Basic Rules

1. JavaScript statements end with semi-colons.
2. JavaScript is case sensitive.
3. JavaScript has two forms of comments:
  - Single-line comments begin with a double slash (/).
  - Multi-line comments begin with "/\*" and end with "\*/".



**Syntax**

```
// This is a single-line comment.  
  
/*  
  This is  
  a multi-line  
  comment.  
*/
```

## 1.5 Accessing Elements

### Dot Notation

In JavaScript, elements (and other objects) can be referenced using dot notation, starting with the highest-level object (i.e., window). Objects can be referred to by name or id or by their position on the page. For example, if there is a form on the page named "loginform", using dot notation you could refer to the form as follows:

**Syntax**

```
window.document.loginform
```

Assuming that loginform is the first form on the page, you could also refer to it in this way:

**Syntax**

```
window.document.forms[0]
```

A document can have multiple form elements as children. The number in the square brackets ( [ ] ) indicates the specific form in question. In programming speak, every document object contains a *collection* of forms. The length of the collection could be zero (meaning there are no forms on the page) or greater. In JavaScript, collections (and arrays) are zero-based, meaning that the first form on the page is referenced with the number zero (0) as shown in the syntax example above.

### Square Bracket Notation

Objects can also be referenced using square bracket notation as shown below:

**Syntax**

```
window['document']['loginform']  
  
// and  
  
window['document']['forms'][0]
```

Dot notation and square bracket notation are completely interchangeable. Dot notation is much more common; however, as we will see later in the course, there are times when it is more convenient to use square bracket notation.

## 1.6 Where Is JavaScript Code Written?

JavaScript code can be written inline (e.g., within HTML tags called event handlers), in `script` blocks, and in external JavaScript files. The page below shows examples of all three.

**Code Sample****JavaScriptBasics/Demos/javascript-1.html**

```
1.  <!DOCTYPE HTML>  
2.  <html>  
3.  <head>  
4.  <meta charset="UTF-8">  
5.  <title>JavaScript Page</title>  
6.  <link href="style.css" rel="stylesheet">  
7.  <script>  
8.    window.alert("The page is loading");  
9.  </script>  
10. </head>  
11. <body>  
12. <p>  
13.   <span onclick="document.body.style.backgroundColor = 'red';">Red</span>  
    >>> |  
14.   <span onclick="document.body.style.backgroundColor =  
    >>> 'white';">White</span>  
15. </p>  
16. <script src="script-1.js"></script>  
17. </body>  
18. </html>
```

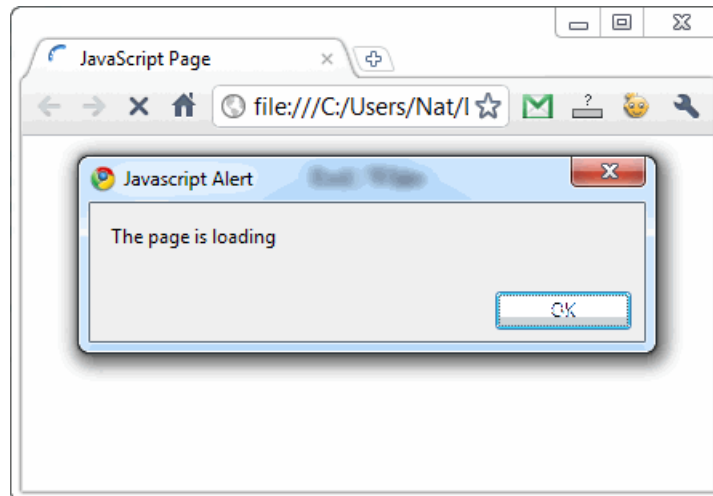
## Code Sample

### JavaScriptBasics/Demos/script-1.js

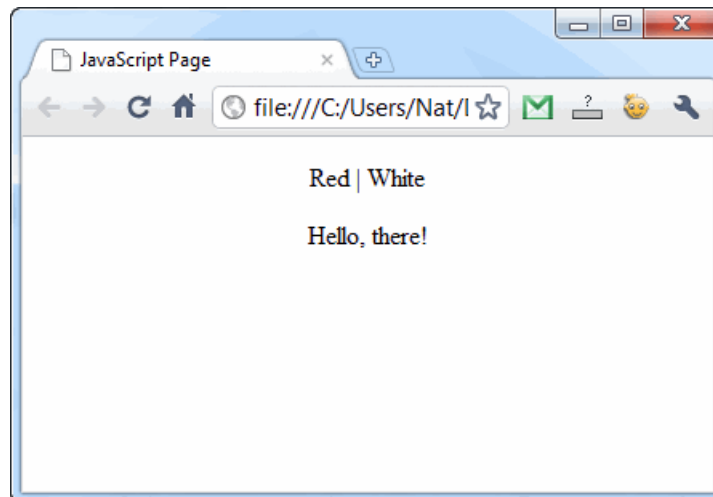
```
1. document.write("<p>Hello, there!</p>");
```

## Code Explanation

As this page loads, an alert will pop up that says "The page is loading" as shown below:



After the user clicks the OK button, the page will finish loading and will appear as follows:



The text "Hello, there!" is written dynamically by the code in [JavaScriptBasics/Demos/script-1.js](#). We will look at the code in this file and in [JavaScriptBasics/Demos/javascript-1.html](#) again shortly.

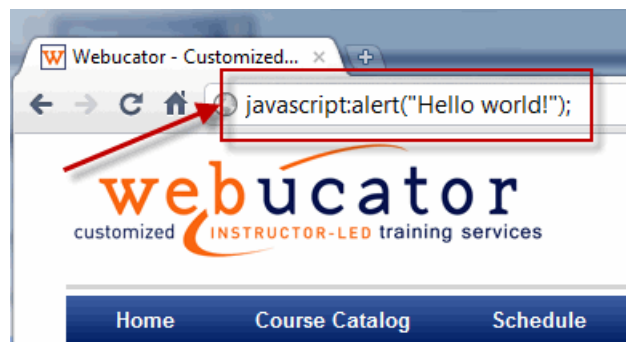
In HTML 4, the `<script>` tag must contain a type attribute set to `text/javascript` like this: `<script type="text/javascript">`.

In HTML5, the assumed (default) value for type is `text/javascript` so it's not necessary to include the attribute, but it doesn't hurt.

## 1.7 The "javascript" Pseudo-Protocol

Try this:

1. Open your browser.
2. In the location bar, type in `javascript:alert("Hello world!");` like so:



3. Press **Enter**.

You should get an alert reading "Hello world!". The `javascript:` prefix is called a pseudo-protocol, because it mimics the protocol syntax (e.g., like `http:`, `0:`, and `mailto:`). It provides an easy way to test JavaScript on a page. It can also be used in links as in the demo below:

### Code Sample

#### JavaScriptBasics/Demos/psuedo-protocol.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Pseudo-Protocol</title>
6.    </head>
7.    <body>
8.    <p><a href="javascript:alert('Hello world!');">Say hello</a></p>
9.    </body>
10.   </html>
```

### **Code Explanation**

When the user clicks the link, the JavaScript alert will execute.

This is generally considered a bad practice. The preferred way to handle this is to use the href attribute to provide an alternative page in case JavaScript is not enabled. You could provide a generic page with a "JavaScript is required" message or, even better, you could provide a page that accomplishes the same task as the alert as in the following the demo:

### Code Sample

#### JavaScriptBasics/Demos/better-than-psuedo-protocol.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Better than Pseudo-Protocol</title>
6.    </head>
7.    <body>
8.    <p><a href="hello-world.html" onclick="alert('Hello world!'); return
      >>> false;">Say hello</a></p>
9.    </body>
10.   </html>
```

### **Code Explanation**

When the user clicks the link...

- If JavaScript is enabled, the alert pops up and the `return false;` statement explicitly cancels the link's default behavior.
- If JavaScript is NOT enabled, the link takes the user to [hello-world.html](#).

## 1.8 JavaScript Objects, Methods and Properties

JavaScript is used to manipulate or get information about objects in the HTML DOM. Objects in an HTML page have methods (actions, such as opening a new window or submitting a form) and properties (attributes or qualities, such as color and size).

To illustrate objects, methods and properties, we will look at the code in [JavaScriptBasics/Demos/javascript-2.html](#), a slightly modified version of [JavaScriptBasics/Demos/javascript-1.html](#), which we looked at earlier, and at the code in [JavaScriptBasics/Demos/script-2.js](#).

**Code Sample****JavaScriptBasics/Demos/javascript-2.html**

```

1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>JavaScript Page</title>
6.    <link href="style.css" rel="stylesheet">
7.    <script>
8.        //Pop up an alert
9.        window.alert("The page is loading");
10.   </script>
11. </head>
12. <body>
13. <p>
14.   <span onclick="document.body.style.backgroundColor = 'red';">Red</span>
15.   >>> |
16.   <span onclick="document.body.style.backgroundColor =
17.   >>> 'white';">White</span> |
18.   <span onclick="document.body.style.backgroundColor =
19.   >>> 'green';">Green</span> |
20.   <span onclick="document.body.style.backgroundColor =
21.   >>> 'blue';">Blue</span>
22. </p>
23. <script src="script-2.js"></script>
24. </body>
25. </html>

```

**Code Sample****JavaScriptBasics/Demos/script-2.js**

```

1.    /*
2.    This script simply outputs
3.    "Hello, there!"
4.    to the browser.
5.    */
6.    document.write("<p>Hello, there!</p>");

```

**Methods**

Methods are the verbs of JavaScript. They cause things to happen.

## **window.alert()**

HTML pages are read and processed from top to bottom. The JavaScript code in the initial `script` block at the top of [JavaScriptBasics/Demos/javascript-2.html](#) calls the `alert()` method of the `window` object. When the browser reads that line of code, it will pop up an alert box and will not continue processing the page until the user presses the OK button. Once the user presses the button, the alert box disappears and the rest of the page loads.

## **document.write()**

The `write()` method of the `document` object is used to write out code to the page as it loads. In [JavaScriptBasics/Demos/script-2.js](#), it simply writes out "Hello, there!"; however, it is more often used to write out dynamic data, such as the date and time on the user's machine.

## **Arguments**

Methods can take zero or more arguments separated by commas.

### **Syntax**

```
object.method(argument1, argument2);
```

The `alert()` and `write()` methods shown in the example above each take only one argument: the message to show or the HTML to write out to the browser.

## **Properties**

Properties are the adjectives of JavaScript. They describe qualities of objects and, in some cases are writable (can be changed dynamically).

## **document.body.style.backgroundColor**

The `style` property of the `document` object is read-write. Looking back at [JavaScriptBasics/Demos/javascript-2.html](#), the four `span` elements use the `onclick` event handler to catch click events. When the user clicks on a `span`, JavaScript is used to set the style of the body to a new color, in the same way that we might use CSS to style the page with `background-color:red` or `background-color:white`.



## **The Implicit window Object**

The window object is always the implicit top-level object and therefore does not have to be included in references to objects. For example, `window.document.write()` can be shortened to `document.write()`. Likewise, `window.alert()` can be shortened to just `alert()`.

## **Exercise 1   Alerts, Writing, & Changing Background Color**

*5 to 15 minutes*

In this exercise, you will practice using JavaScript to popup an alert, write text to the screen, and set the background color of the page.

1. Open [JavaScriptBasics/Exercises/alert-write-bgcolor.html](#) for editing.
2. In the `head` of the file, add a JavaScript alert which pops up the message "Welcome to my page!" when the page loads.
3. Add click handlers to the two buttons to allow the user to change the background color of the page to red or to blue.
4. In the `script` at the bottom of the page, use JavaScript to write the text "This text was generated by JavaScript" to the page.
5. Test your solution in a browser.



### Exercise Solution

#### JavaScriptBasics/Solutions/alert-write-bgcolor.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Alert, Write, BGColor</title>
6.    <script>
7.        window.alert("Welcome to my page!");
8.    </script>
9.    </head>
10.   <body>
11.   <form>
12.       <p>Click the button to turn the page:
13.       <button onclick="document.body.style.backgroundColor = 'red'; return
           >>> false">Red</button>
14.       <p>Click the button to turn the page:
15.       <button onclick="document.body.style.backgroundColor='blue'; return
           >>> false">Blue</button>
16.   </form>
17.   <hr>
18.   <script>
19.       document.write('This text was generated by JavaScript');
20.   </script>
21. </body>
22. </html>
```

### Code Explanation

We use `alert()` to generate the popup the head and `document.write()` to write to the screen at the bottom of the page. We use `onclick="document.body.style.backgroundColor='red'; return >>> false"` to add a click handler to the button; `return false` ensures that the page does not reload when the button is clicked.

## **1.9 Conclusion**

In this lesson, you have learned the basics of JavaScript. Now you're ready for more.



## 2. Variables, Arrays and Operators

**In this lesson, you will learn...**

1. To create, read and modify JavaScript variables.
2. To work with JavaScript arrays.
3. To work with JavaScript operators.

### 2.1 JavaScript Variables

Variables are used to hold data in memory. JavaScript variables are declared with the `var` keyword.

```
var age;
```

Multiple variables can be declared in a single step.

```
var age, height, weight, gender;
```

After a variable is declared, it can be assigned a value.

```
age = 34;
```

Variable declaration and assignment can be done in a single step.

```
var age = 34;
```

### A Loosely-typed Language

JavaScript is a loosely-typed language. This means that you do not specify the data type of a variable when declaring it. It also means that a single variable can hold different data types at different times and that JavaScript can change the variable type on the fly. (This "casual" approach to typing can lead to difficulties in writing and maintaining large applications - tools like TypeScript offer a solution to this challenge.)<sup>2</sup>

For example, the `age` variable above is an integer. However, the variable `strAge` below would be a string (text) because of the quotes.

---

2. [TypeScript \(http://www.typescriptlang.org/\)](http://www.typescriptlang.org/) is an open-source programming language developed by Microsoft. Developers writing in TypeScript compile their code to valid JavaScript, which they can use anywhere one might use JavaScript. A useful feature of TypeScript is **static typing**, meaning that a developer might specify the type of a given variable - to be a string, say, or a Boolean true/false variable - when declaring the variable. Violations of this static typing - trying to work with a

```
var strAge = "34";
```

If you were to try to do a math function on `strAge` (e.g., multiply it by 4), JavaScript would dynamically change it to an integer. Although this is very convenient, it can also cause unexpected results, so be careful.

## Variable Naming

1. Variable names must begin with a letter, underscore (`_`), or dollar sign (`$`).
2. Variable names cannot contain spaces or special characters (other than the underscore and dollar sign).
3. Variable names can contain numbers (but not as the first character).
4. Variable names are case sensitive.
5. You cannot use keywords (e.g., `window` or `function`) as a variable name.

## Storing User-Entered Data

The following example uses the `prompt()` method of the `window` object to collect user input. The value entered by the user is then assigned to a variable, which is accessed when the user clicks on one of the `span` elements.

---

number value as if it were a string value, for example - produces an error when compiling the TypeScript code into JavaScript, and thus adds a check against a dangerous bug creeping into the code.

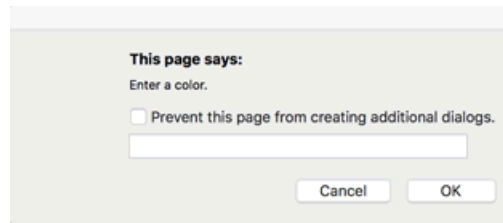


**Code Sample****VariablesArraysOperators/Demos/Variables.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>JavaScript Variables</title>
6.    <link href="style.css" rel="stylesheet">
7.    <script>
8.        //Pop up a prompt
9.        var userColor = window.prompt("Enter a color.", "");
10.    </script>
11. </head>
12. <body>
13. <p>
14.     <span onclick="document.body.style.backgroundColor = 'red';">Red</span>
15.         >>> |
16.     <span onclick="document.body.style.backgroundColor =
17.         >>> 'white';">White</span> |
18.     <span onclick="document.body.style.backgroundColor =
19.         >>> 'green';">Green</span> |
20.     <span onclick="document.body.style.backgroundColor =
21.         >>> 'blue';">Blue</span> |
22.     <span onclick="document.body.style.backgroundColor = userColor;">
23.     <script>
24.         document.write(userColor);
25.     </script>
26. </span>
27. </p>
28. </body>
29. </html>
```

### Code Explanation

As the page loads, a prompt pops up asking the user to enter a color.



This is done with the `prompt()` method of the `window` object. The `prompt()` method is used to get input from the user. It takes two arguments:

1. The message in the dialog box (e.g., "Enter a color.").
2. The default value that appears in the text box. In the example above this is an empty string (e.g., "").

If the OK button is pressed, the prompt returns the value entered in the text box. If the Cancel button or the close button (the red X) is pressed, the prompt returns `null`.<sup>3</sup> The line below assigns whatever is returned to the variable `userColor`.

```
var userColor = window.prompt("Enter a color.", "");
```

A `script` block with a call to `document.write()` is then used to output the color entered by the user. This output is contained within a `span` element, which has an `onclick` event handler that will be used to turn the background color of the page to the user-entered color.

```
<span onclick="document.body.style.backgroundColor = userColor;">  
  <script type="text/javascript">  
    document.write(userColor);  
  </script>  
</span>
```

---

3. In JavaScript, `null` is a datatype with only one value: `null`. It represents a value that we don't know or that is missing

## Exercise 2 Using Variables

*5 to 15 minutes*

In this exercise, you will practice using variables.

1. Open [VariablesArraysOperators/Exercises/Variables.html](#) for editing.
2. Below the `ADD PROMPT HERE` comment, write code that will prompt the user for her first name and assign the result to a variable.
3. Add a button below the Ringo button that reads "Your Name". Add functionality so that when this button is pressed an alert pops up showing the user's first name.
4. Test your solution in a browser.

### Exercise Code

#### **VariablesArraysOperators/Exercises/Variables.html**

```
1. <!DOCTYPE HTML>
2. <html>
3. <head>
4. <meta charset="UTF-8">
5. <title>JavaScript Variables</title>
6. <link href="style.css" rel="stylesheet">
7. <script>
8.   //ADD PROMPT HERE
9. </script>
10. </head>
11. <body>
12. <form>
13.   <input type="button" value="Paul" onclick="alert('Paul');">
14.   <input type="button" value="John" onclick="alert('John');">
15.   <input type="button" value="George" onclick="alert('George');">
16.   <input type="button" value="Ringo" onclick="alert('Ringo');">
17.   <!--ADD BUTTON HERE-->
18. </form>
19. </body>
20. </html>
```

### Exercise Solution

#### VariablesArraysOperators/Solutions/Variables.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>JavaScript Variables</title>
6.    <link href="style.css" rel="stylesheet">
7.    <script>
8.        var firstName = window.prompt("What's your name?", "");
9.    </script>
10.   </head>
11.   <body>
12.   <form>
13.       <input type="button" value="Paul" onclick="alert('Paul');">
14.       <input type="button" value="John" onclick="alert('John');">
15.       <input type="button" value="George" onclick="alert('George');">
16.       <input type="button" value="Ringo" onclick="alert('Ringo');">
17.       <input type="button" value="Your Name" onclick="alert(firstName);">
18.   </form>
19.   </body>
20.   </html>
```

## 2.2 Constants

The most-recent ("ES6") ECMAScript standard introduced the `const` keyword into JavaScript. Variables declared as `const`:

```
const NUM = 1;
```

cannot be reassigned; that is, a later statement like `NUM = 2;` would fail, meaning that the value of variable `NUM` would remain 1; depending on how the browser you are using handles `const`, the later statement may either cause the code to fail or simply not assign the new value to `NUM`.

Please note that, as this is a relatively-recent JavaScript feature, not all browsers may yet implement it, or may implement it differently. (The useful website [caniuse.com](http://caniuse.com) lists current browser support for this - and many other - JavaScript features; see <http://caniuse.com/#feat=const> for a list of how browsers handle `const`.)

While constants can be declared with uppercase or lowercase names, the convention is to use all-uppercase names for constants.

Let's look at a simple example:

**Code Sample****VariablesArraysOperators/Demos/Const.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>JavaScript const</title>
6.    <link href="style.css" rel="stylesheet">
7.    </head>
8.    <body>
9.    <script>
10.    const MY_VAR = 1;
11.    MY_VAR = 2;
12.    document.write(MY_VAR);
13.    /*
14.    value "1" gets written to the screen or
15.    code fails at line 11, depending on browser
16.    */
17.    </script>
18.    </body>
19.    </html>
```

**Code Explanation**

We assign a value 1 to `const` variable `MY_VAR`. On the next line, we try to assign value 2 to `MY_VAR`. The result of the code differs by browser: for Firefox, the code fails (with console error message "invalid assignment to const MY\_VAR") and no value is written to the screen; on Chrome, the second assignment statement fails silently and the value 1 is written to the screen.

## 2.3 Arrays

An array is a grouping of objects that can be accessed through subscripts. At its simplest, an array can be thought of as a list. In JavaScript, the first element of an array is considered to be at position zero (0), the second element at position one (1), and so on. Arrays are useful for storing related sets of data.

Arrays are declared using the `new` keyword.

```
var myarray = new Array();
```

It is also possible and very common to use the `[ ]` literal to declare a new Array object.

```
var myarray = [];
```

Values are assigned to arrays as follows:

```
myarray[0] = value1;  
myarray[1] = value2;  
myarray[2] = value3;
```

Arrays can be declared with initial values.

```
var myarray = new Array(value1, value2, value3);  
//or, using the [] notation:  
var myarray = [value1, value2, value3];
```

The following example is similar to the previous one, except that it prompts the user for four different colors and places each into the `userColors` array. It then displays the values in the `userColors` array in the spans and assigns them to `document.body.style.backgroundColor` when the user clicks on the spans.

Unlike in some languages, values in JavaScript arrays do not all have to be of the same data type.

**Code Sample****VariablesArraysOperators/Demos/Arrays.html**

```
-----Lines 1 through 6 Omitted-----
7.  <script>
8.    //Pop up four prompts and create an array
9.    var userColors = new Array();
10.   userColors[0] = window.prompt("Choose a color.", "");
11.   userColors[1] = window.prompt("Choose a color.", "");
12.   userColors[2] = window.prompt("Choose a color.", "");
13.   userColors[3] = window.prompt("Choose a color.", "");
14. </script>
15. </head>
16. <body>
17. <p>
18.   <span onclick="document.body.style.backgroundColor = userColors[0];">
19.     >>>
20.     <script>
21.       document.write(userColors[0]);
22.     </script>
23.   </span> |
24.   <span onclick="document.body.style.backgroundColor = userColors[1];">
25.     >>>
26.     <script>
27.       document.write(userColors[1]);
28.     </script>
29.   </span> |
30.   <span onclick="document.body.style.backgroundColor = userColors[2];">
31.     >>>
32.     <script>
33.       document.write(userColors[2]);
34.     </script>
35.   </span> |
36.   <span onclick="document.body.style.backgroundColor = userColors[3];">
37.     >>>
38.     <script>
39.       document.write(userColors[3]);
40.     </script>
41.   </span>
42. </p>
43. </body>
44. </html>
```



## Code Explanation

As the page loads, an array called `userColors` is declared.

```
userColors = new Array();
```

The next four lines populate the array with user-entered values.

```
userColors[0] = window.prompt("Choose a color.", "");  
userColors[1] = window.prompt("Choose a color.", "");  
userColors[2] = window.prompt("Choose a color.", "");  
userColors[3] = window.prompt("Choose a color.", "");
```

The body of the page contains a paragraph with four `span` tags, the text of which is dynamically created with values from the `userColors` array.

## Exercise 3 Working with Arrays

*15 to 25 minutes*

In this exercise, you will practice working with arrays.

1. Open [VariablesArraysOperators/Exercises/Arrays.html](#) for editing.
2. Below the comment, declare a `rockStars` array and populate it with four values entered by the user.
3. Add functionality to the buttons, so that alerts pop up with values from the array when the buttons are clicked.
4. Test your solution in a browser.

### Exercise Code

#### **VariablesArraysOperators/Exercises/Arrays.html**

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>JavaScript Arrays</title>
6.  <link href="style.css" rel="stylesheet">
7.  <script>
8.    /*
9.     Declare a rockStars array and populate it with
10.    four values entered by the user.
11.    */
12. </script>
13. </head>
14. <body>
15. <form>
16.   <input type="button" value="Favorite">
17.   <input type="button" value="Next Favorite">
18.   <input type="button" value="Next Favorite">
19.   <input type="button" value="Next Favorite">
20. </form>
21. </body>
22. </html>
```



### Exercise Solution

#### VariablesArraysOperators/Solutions/Arrays.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>JavaScript Arrays</title>
6.    <link href="style.css" rel="stylesheet">
7.    <script>
8.        var rockStars = new Array();
9.        rockStars[0] = window.prompt("Who is your favorite rock star?", "");
10.       rockStars[1] = window.prompt("And your next favorite rock star?", "");
11.           >>>
12.       rockStars[2] = window.prompt("And your next favorite rock star?", "");
13.           >>>
14.       rockStars[3] = window.prompt("And your next favorite rock star?", "");
15.           >>>
16.    </script>
17.    </head>
18.    <body>
19.    <form>
20.        <input type="button" value="Favorite" onclick="alert(rockStars[0]);">
21.           >>>
22.        <input type="button" value="Next Favorite" onclick="alert(rock »»
23.           >>> Stars[1]);">
24.        <input type="button" value="Next Favorite" onclick="alert(rock »»
25.           >>> Stars[2]);">
26.        <input type="button" value="Next Favorite" onclick="alert(rock »»
27.           >>> Stars[3]);">
28.    </form>
29.    </body>
30.    </html>
```

## Associative Arrays

Whereas regular (or enumerated) arrays are indexed numerically, associative arrays are indexed using names as keys. The advantage of this is that the keys can be meaningful, which can make it easier to reference an element in an array. The example below illustrates how an associative array is used.

### Code Sample

#### VariablesArraysOperators/Demos/AssociativeArrays.html

```

1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>Associative Arrays</title>
6.  <link href="style.css" rel="stylesheet">
7.  <script>
8.      var beatles = [];
9.      beatles["singer1"] = "Paul";
10.     beatles["singer2"] = "John";
11.     beatles["guitarist"] = "George";
12.     beatles["drummer"] = "Ringo";
13. </script>
14. </head>
15. <body>
16. <p>
17.     <script>
18.         document.write(beatles["singer1"]);
19.         document.write(beatles["singer2"]);
20.         document.write(beatles["guitarist"]);
21.         document.write(beatles["drummer"]);
22.     </script>
23. </p>
24. </body>
25. </html>

```

## Array Properties and Methods

The tables below show some of the most useful array properties and methods. All of the examples assume an array called `beatles` that holds "Paul", "John", "George", and "Ringo".

```
var beatles = ["Paul", "John", "George", "Ringo"];
```

**Array Properties**

Property	Description
length	Holds the number of elements in an array.
	<code>beatles.length // 4</code>

**Array Methods**

Property	Description
<code>join(delimiter)</code>	Returns a delimited list of the items indexed with integers in the array. The default delimiter is a comma.
	<code>beatles.join(":") // Paul:John:George:Ringo</code>
<code>push()</code>	Appends an element to an array.
	<code>beatles.push("Steve")</code>
<code>pop()</code>	Removes the last item in an array and returns its value.
	<code>beatles.pop() // Returns Ringo</code>
<code>shift()</code>	Removes the first item in an array and returns its value.
	<code>beatles.shift() // Returns Paul</code>
<code>slice(start, end)</code>	Returns a subarray from start to end. If end is left out, it includes the remainder of the array.
	<code>beatles.slice(1, 2) //Returns [John, George]</code>
<code>splice(start, count)</code>	Removes count items from start in the array and returns the resulting array.
	<code>beatles.splice(1, 2) //Returns [Paul, Ringo]</code>
<code>sort()</code>	Sorts an array alphabetically.
	<code>beatles.sort() //Returns [George, John, Paul, Ringo]</code>

## 2.4 JavaScript Operators

### Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (remainder)
++	Increment by one
--	Decrement by one

### Assignment Operators

Operator	Description
=	Assignment
+=	One step addition and assignment (a+=3 is the same as a=a+3)
-=	One step subtraction and assignment (a-=3 is the same as a=a-3)
*=	One step multiplication and assignment (a*=3 is the same as a=a*3)
/=	One step division and assignment (a/=3 is the same as a=a/3)
%=	One step modulus and assignment (a%=3 is the same as a=a%3)

### String Operators

Operator	Description
+	Concatenation
	<code>var greeting = "Hello " + firstname;</code>
+=	One step concatenation and assignment
	<code>var greeting = "Hello "; greeting += firstname;</code>

The following code sample shows these operators in use:

### Code Sample

#### VariablesArraysOperators/Demos/Operators.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>JavaScript Operators</title>
6.    <link href="style.css" rel="stylesheet">
7.    <script>
8.        var userNum1 = window.prompt("Choose a number.", "");
9.        alert("You chose " + userNum1);
10.       var userNum2 = window.prompt("Choose another number.", "");
11.       alert("You chose " + userNum2);
12.       var numsAdded = userNum1 + userNum2;
13.       var numsSubtracted = userNum1 - userNum2;
14.       var numsMultiplied = userNum1 * userNum2;
15.       var numsDivided = userNum1 / userNum2;
16.       var numsModulused = userNum1 % userNum2;
17.    </script>
18.    </head>
19.    <body>
20.    <p>
21.        <script>
22.            document.write(userNum1 + " + " + userNum2 + " = ");
23.            document.write(numsAdded + "<br>");
24.            document.write(userNum1 + " - " + userNum2 + " = ");
25.            document.write(numsSubtracted + "<br>");
26.            document.write(userNum1 + " * " + userNum2 + " = ");
27.            document.write(numsMultiplied + "<br>");
28.            document.write(userNum1 + " / " + userNum2 + " = ");
29.            document.write(numsDivided + "<br>");
30.            document.write(userNum1 + " % " + userNum2 + " = ");
31.            document.write(numsModulused + "<br>");
32.        </script>
33.    </p>
34.    </body>
35.    </html>
```



## Code Explanation

The file above illustrates the use of the concatenation operator and several math operators. It also illustrates a potential problem with loosely-typed languages. This page is processed as follows:

1. The user is prompted for a number and the result is assigned to `userNum1`.
2. An alert pops up telling the user what number she entered. The concatenation operator (+) is used to combine two strings: "You chose " and the number entered by the user. Note that all user-entered data is always treated as a string of text, even if the text consists of only digits.
3. The user is prompted for another number and the result is assigned to `userNum2`.
4. Another alert pops up telling the user what number she entered.
5. Five variables are declared and assigned values.

```
var numsAdded = userNum1 + userNum2;
var numsSubtracted = userNum1 - userNum2;
var numsMultiplied = userNum1 * userNum2;
var numsDivided = userNum1 / userNum2;
var numsModulus = userNum1 % userNum2;
```

6. The values the variables contain are output to the browser.

$5 + 4 = 54$

$5 - 4 = 1$

$5 * 4 = 20$

$5 / 4 = 1.25$

$5 \% 4 = 1$

So,  $5 + 4$  is 54! It is when 5 and 4 are strings, and, as stated earlier, all user-entered data is treated as a string. In the lesson on JavaScript Functions (see page 43), you will learn how to convert a string to a number.

## Ternary Operator

Operator	Description
?:	Conditional evaluation.
	<code>var evenOrOdd = (number % 2 == 0) ? "even" : "odd";</code>

The code sample below shows how the ternary operator works:

**Code Sample****VariablesArraysOperators/Demos/Ternary.html**

```
-----Lines 1 through 5 Omitted-----
6.  <script>
7.    var num = prompt("Enter a number.", "");
8.
9.    //without ternary
10.   if (num % 2 == 0) {
11.     alert(num + " is even.");
12.   } else {
13.     alert(num + " is odd.");
14.   }
15.
16.   //with ternary
17.   var term = num % 2 == 0 ? "even" : "odd";
18.   alert(num + " is " + term);
19. </script>
-----Lines 20 through 24 Omitted-----
```

**Code Explanation**

Lines 10-14 show a regular if-else statement, which we will cover in detail in Conditionals and Loops (see page 105).

Lines 17-18 shows how to accomplish the same thing in a couple of lines of code with the ternary operator.

**Default Operator**

Operator	Description
	Used to assign a default value. (This is explained in greater detail later in the course.)
	<code>var yourName = prompt("Your Name?", "")    "Stranger";</code>

The code sample below shows how the default operator works:

### Code Sample

#### VariablesArraysOperators/Demos/Default.html

```
-----Lines 1 through 5 Omitted-----  
6.  <script>  
7.    var yourName = prompt("Your Name?","") || "Stranger";  
8.  
9.    alert("Hi " + yourName + "!");  
10. </script>  
-----Lines 11 through 15 Omitted-----
```

### Code Explanation

If the user presses **OK** without filling out the prompt or presses **Cancel**, the default value "Stranger" is assigned to yourName.

## Exercise 4 Working with Operators

*15 to 25 minutes*

In this exercise, you will practice working with JavaScript operators.

1. Open [VariablesArraysOperators/Exercises/Operators.html](#) for editing.
2. Add code to prompt the user for the number of cds she owns of her favorite and second favorite rockstars'.
3. In the body, let the user know how many more of her favorite rockstar's cds she has than of her second favorite rockstar's albums.
4. Test your solution in a browser.

### Exercise Code

#### **VariablesArraysOperators/Exercises/Operators.html**

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>JavaScript Operators</title>
6.  <link href="style.css" rel="stylesheet">
7.  <script>
8.    var rockStars = [];
9.    rockStars[0] = prompt("Who is your favorite rock star?", "");
10.   /*
11.    Ask the user how many of this rockstar's cds she owns and store
12.    the result in a variable.
13.   */
14.    rockStars[1] = prompt("And your next favorite rock star?", "");
15.   /*
16.    Ask the user how many of this rockstar's cds she owns and store
17.    the result in a variable.
18.   */
19.  </script>
20. </head>
21. <body>
22. <!--
23.   Let the user know how many more of her favorite rockstar's albums
24.   she has than of her second favorite rockstar's cds.
25. -->
26. </body>
27. </html>
```

**\*Challenge**

1. Open [VariablesArraysOperators/Exercises/Operators-challenge.html](#) for editing.
2. Modify it so that it outputs an unordered list as shown below:

- $5 + 4 = 54$
- $5 - 4 = 1$
- $5 * 4 = 20$
- $5 / 4 = 1.25$
- $5 \% 4 = 1$

### Exercise Solution

#### VariablesArraysOperators/Solutions/Operators.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>JavaScript Operators</title>
6.    <link href="style.css" rel="stylesheet">
7.    <script>
8.        var rockStars = [];
9.        var cdTotals = [];
10.    rockStars[0] = prompt("Who is your favorite rock star?", "");
11.    cdTotals[0] = prompt("How many " + rockStars[0] + " cds do you own?",
    >>>    "");
12.    rockStars[1] = prompt("And your next favorite rock star?", "");
13.    cdTotals[1] = prompt("How many " + rockStars[1] + " cds do you own?",
    >>>    "");
14. </script>
15. </head>
16. <body>
17.     <script>
18.         var diff = cdTotals[0] - cdTotals[1];
19.         document.write("You have " + diff + " more cds of " + rockStars[0]);
20.         document.write(" than you have of " + rockStars[1] + ".");
21.     </script>
22. </body>
23. </html>
```

## Challenge Solution

### VariablesArraysOperators/Solutions/Operators-challenge.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>JavaScript Operators</title>
6.    <link href="style.css" rel="stylesheet">
7.    <script>
8.        var userNum1 = window.prompt("Choose a number.", "");
9.        alert("You chose " + userNum1);
10.       var userNum2 = window.prompt("Choose another number.", "");
11.       alert("You chose " + userNum2);
12.       var numsAdded = userNum1 + userNum2;
13.       var numsSubtracted = userNum1 - userNum2;
14.       var numsMultiplied = userNum1 * userNum2;
15.       var numsDivided = userNum1 / userNum2;
16.       var numsModulused = userNum1 % userNum2;
17.    </script>
18.    </head>
19.    <body>
20.    <ul>
21.        <script>
22.            document.write("<li>" + userNum1 + " + " + userNum2 + " = ");
23.            document.write(numsAdded + "</li>");
24.            document.write("<li>" + userNum1 + " - " + userNum2 + " = ");
25.            document.write(numsSubtracted + "</li>");
26.            document.write("<li>" + userNum1 + " * " + userNum2 + " = ");
27.            document.write(numsMultiplied + "</li>");
28.            document.write("<li>" + userNum1 + " / " + userNum2 + " = ");
29.            document.write(numsDivided + "</li>");
30.            document.write("<li>" + userNum1 + " % " + userNum2 + " = ");
31.            document.write(numsModulused + "</li>");
32.        </script>
33.    </ul>
34.    </body>
35.    </html>
```

## 2.5 Conclusion

In this lesson, you have learned to work with JavaScript variables, arrays and operators.



## 3. JavaScript Functions

**In this lesson, you will learn...**

1. To work with some of JavaScript's global functions.
2. To create your own functions.
3. To return values from functions.

### 3.1 Global Functions

JavaScript has a number of global functions. We will examine some of them in this section.

#### **Number(object)** <sup>4</sup>

The `Number ( )` function takes one argument: an object, which it attempts to convert to a number. If it cannot, it returns NaN, for "Not a Number."

---

4. `Number ( )` is not really a function, but rather a constructor for creating a Number object.

### Code Sample

#### JavaScriptFunctions/Demos/Number.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Number() Function</title>
6.    <link href="style.css" rel="stylesheet">
7.    <script>
8.        var strNum1 = "1";
9.        var strNum2 = "2";
10.       var strSum = strNum1 + strNum2; //returns 12
11.       alert(strSum);
12.
13.       var intNum1 = Number(strNum1);
14.       var intNum2 = Number(strNum2);
15.       var intSum = intNum1 + intNum2; //returns 3
16.       alert(intSum);
17.    </script>
18.    </head>
19.    <body>
20.        <p>Nothing to show here.</p>
21.    </body>
22.    </html>
```

### Code Explanation

Because `strNum1` and `strNum2` are both strings, the `+` operator concatenates them, resulting in "12".

```
var strNum1 = "1";
var strNum2 = "2";
var strSum = strNum1 + strNum2; //returns 12
alert(strSum);
```

After the `Number ( )` function has been used to convert the strings to numbers, the `+` operator performs addition, resulting in 3.

```
var intNum1 = Number(strNum1);
var intNum2 = Number(strNum2);
var intSum = intNum1 + intNum2; //returns 3
alert(intSum);
```

## String(object)<sup>5</sup>

The `String()` function takes one argument: an object, which it converts to a string.

### Code Sample

#### JavaScriptFunctions/Demos/String.html

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>String() Function</title>
6.  <link href="style.css" rel="stylesheet">
7.  <script>
8.      var intNum1 = 1;
9.      var intNum2 = 2;
10.     var intSum = intNum1 + intNum2; //returns 3
11.     alert(intSum);
12.
13.     var strNum1 = String(intNum1);
14.     var strNum2 = String(intNum2);
15.     var strSum = strNum1 + strNum2; //returns 12
16.     alert(strSum);
17. </script>
18. </head>
19. <body>
20.     <p>Nothing to show here.</p>
21. </body>
22. </html>
```

### Code Explanation

Because `intNum1` and `intNum2` are both numbers, the `+` operator performs addition, resulting in 3.

```
var intNum1 = 1;
var intNum2 = 2;
var intSum = intNum1 + intNum2; //returns 3
alert(intSum);
```

---

5. `String()` is not really a function, but rather a constructor for creating a `String` object.

After the `String()` function has been used to convert the numbers to strings, the `+` operator performs concatenation, resulting in "12".

```
var strNum1 = String(intNum1);
var strNum2 = String(intNum2);
var strSum = strNum1 + strNum2; //returns 12
alert(strSum);
```

### isNaN(object)

The `isNaN()` function takes one argument: an object. The function checks if the object is *not* a number (or cannot be converted to a number). It returns `true` if the object is not a number and `false` if it is a number.

#### Code Sample

##### JavaScriptFunctions/Demos/isNaN.html

```
-----Lines 1 through 8 Omitted-----
9.   <table>
10.  <tr>
11.    <th>Function</th><th>Result</th>
12.  </tr>
13.  <script>
14.    document.write("<tr><td>isNaN(4)</td>");
15.    document.write("<td>" + isNaN(4) + "</td></tr>");
16.    document.write("<tr><td>isNaN(\"4\")</td>");
17.    document.write("<td>" + isNaN("4") + "</td></tr>");
18.    document.write("<tr><td>isNaN(0/0)</td>");
19.    document.write("<td>" + isNaN(0/0) + "</td></tr>");
20.    document.write("<tr><td>isNaN(\"hello\")</td>");
21.    document.write("<td>" + isNaN("hello") + "</td></tr>");
22.    var strAge = "twelve";
23.    document.write("<tr><td>isNaN(strAge)</td>");
24.    document.write("<td>" + isNaN(strAge) + "</td></tr>");
25.    var intAge = 12;
26.    document.write("<tr><td>isNaN(intAge)</td>");
27.    document.write("<td>" + isNaN(intAge) + "</td></tr>");
28.  </script>
29.  </table>
-----Lines 30 through 31 Omitted-----
32.
```

## Code Explanation

The output will look like this:

Function	Result
isNaN(4)	false
isNaN("4")	false
isNaN(0/0)	true
isNaN("hello")	true
isNaN(strAge)	true
isNaN(intAge)	false

## parseFloat() and parseInt()

The `parseFloat()` function takes one argument: a string. If the string begins with a number, the function reads through the string until it finds the end of the number, hacks off the remainder of the string, and returns the result. If the string does not begin with a number, the function returns NaN.

The `parseInt()` function also takes one argument: a string. If the string begins with an integer, the function reads through the string until it finds the end of the integer, hacks off the remainder of the string, and returns the result. If the string does not begin with an integer, the function returns NaN.

### Code Sample

#### JavaScriptFunctions/Demos/ParsingNumbers.html

```
-----Lines 1 through 8 Omitted-----
9.   <table>
10.  <tr>
11.    <th>Function</th><th>Result</th>
12.  </tr>
13.  <script>
14.    var race = "26.2 miles";
15.    document.write("<tr><td>parseFloat(race)</td>");
16.    document.write("<td>" + parseFloat(race) + "</td></tr>");
17.    document.write("<tr><td>parseInt(race)</td>");
18.    document.write("<td>" + parseInt(race) + "</td></tr>");
19.    race = "Marathon";
20.    document.write("<tr><td>parseFloat(race)</td>");
21.    document.write("<td>" + parseFloat(race) + "</td></tr>");
22.    document.write("<tr><td>parseInt(race)</td>");
23.    document.write("<td>" + parseInt(race) + "</td></tr>");
24.  </script>
25. </table>
-----Lines 26 through 27 Omitted-----
```

### Code Explanation

The output will look like this:

Function	Result
parseFloat(race)	26.2
parseInt(race)	26
parseFloat(race)	NaN
parseInt(race)	NaN

These "global" functions we have discussed above are actually methods of the window object, but as window is assumed if no object is referenced, we don't need to explicitly write `window.parseFloat()` or `window.parseInt()`.

## Exercise 5 Working with Global Functions

*10 to 15 minutes*

In this exercise, you will practice working with JavaScript's global functions.

1. Open [JavaScriptFunctions/Exercises/BuiltinFunctions.html](#) for editing.
2. Modify the file so that it outputs the sum of the two numbers entered by the user.

### Exercise Code

#### JavaScriptFunctions/Exercises/BuiltinFunctions.html

```

1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>JavaScript Built-in Functions</title>
6.  <link href="style.css" rel="stylesheet">
7.  <script>
8.    var userNum1, userNum2, numsAdded;
9.    userNum1 = window.prompt("Choose a number.", "");
10.   alert("You chose " + userNum1);
11.   userNum2 = window.prompt("Choose another number.", "");
12.   alert("You chose " + userNum2);
13.   numsAdded = userNum1 + userNum2;
14. </script>
15. </head>
16. <body>
17. <p>
18.   <script>
19.     document.write(userNum1 + " + " + userNum2 + " = ");
20.     document.write(numsAdded + "<br/>");
21.   </script>
22. </p>
23. </body>
24. </html>

```

### **\*Challenge**

Create a new HTML file that prompts the user for his name, the age at which he first worked on a computer, and his current age. After gathering this information,

---

## JavaScript Functions

pop up an alert that tells the user how many years he's been working on a computer. The images below show the steps:

The image displays four sequential screenshots of JavaScript alert dialog boxes, illustrating a form process:

- First dialog:** Titled "This page says:", it asks "What's your name?". The input field contains "Nat". There are "Cancel" and "OK" buttons.
- Second dialog:** Titled "This page says:", it asks "How old were you when you first used a computer?". It includes a checkbox "Prevent this page from creating additional dialogs." and an input field containing "12 years old". There are "Cancel" and "OK" buttons.
- Third dialog:** Titled "This page says:", it asks "How old are you now?". It includes a checkbox "Prevent this page from creating additional dialogs." and an input field containing "35 years old". There are "Cancel" and "OK" buttons.
- Fourth dialog:** Titled "This page says:", it displays the message "Nat, you have used computers for 23 years." and the checkbox "Prevent this page from creating additional dialogs.". There is an "OK" button.

Notice that the program is able to deal with numbers followed by strings (e.g., "12 years old").





### Exercise Solution

#### JavaScriptFunctions/Solutions/BuiltinFunctions.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>JavaScript Built-in Functions</title>
6.    <link href="style.css" rel="stylesheet">
7.    <script>
8.        var userNum1, userNum2, numsAdded;
9.        userNum1 = window.prompt("Choose a number.", "");
10.       alert("You chose " + userNum1);
11.       userNum2 = window.prompt("Choose another number.", "");
12.       alert("You chose " + userNum2);
13.       numsAdded = Number(userNum1) + Number(userNum2);
14.    </script>
15.    </head>
16.    <body>
17.    <p>
18.        <script>
19.            document.write(userNum1 + " + " + userNum2 + " = ");
20.            document.write(numsAdded + "<br/>");
21.        </script>
22.    </p>
23.    </body>
24.    </html>
```

## Challenge Solution

### JavaScriptFunctions/Solutions/BuiltInFunctions-challenge.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>JavaScript Built-in Functions</title>
6.    <link href="style.css" rel="stylesheet">
7.    <script>
8.        var userName = prompt("What's your name?", "");
9.        var age1 = prompt("How old were you when you first used a computer?",
    >>>    "");
10.    var age2 = prompt("How old are you now?", "");
11.    var diff = parseFloat(age2) - parseFloat(age1);
12.    alert(userName + ", you have used computers for " + diff + " years.");
    >>>
13.    </script>
14.    </head>
15.    <body>
16.        <p>Nothing to show here.</p>
17.    </body>
18.    </html>
```

## 3.2 User-defined Functions

Writing functions makes it possible to reuse code for common tasks. Functions can also be used to hide complex code. For example, an experienced developer can write a function for performing a complicated task. Other developers do not need to know how that function works; they only need to know how to call it.

### Function Syntax

JavaScript functions generally appear in the head of the page or in external JavaScript files. A function is written using the `function` keyword followed by the name of the function.

**Syntax**

```
function doSomething(){  
    //function statements go here  
}
```

As you can see, the body of the function is contained within curly brackets (`{ }`). The following example demonstrates the use of simple functions:

## Code Sample

### JavaScriptFunctions/Demos/SimpleFunctions.html

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>JavaScript Simple Functions</title>
6.  <link href="style.css" rel="stylesheet">
7.  <script>
8.      function changeBgRed(){
9.          document.body.style.backgroundColor = "red";
10.     }
11.
12.     function changeBgWhite(){
13.         document.body.style.backgroundColor = "white";
14.     }
15. </script>
16. </head>
17. <body>
18. <p>
19.     <span onclick="changeBgRed();">Red</span> |
20.     <span onclick="changeBgWhite();">White</span>
21. </p>
22. </body>
23. </html>
```

## Passing Values to Functions

The functions above aren't very useful because they always do the same thing. Every time we wanted to add another color, we would have to write another function. Also, if we want to modify the behavior, we will have to do it in each function. The example below shows how to create a single function to handle changing the background color.

### Code Sample

#### JavaScriptFunctions/Demos/PassingValues.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Passing Values</title>
6.    <link href="style.css" rel="stylesheet">
7.    <script>
8.        function changeBg(color){
9.            document.body.style.backgroundColor = color;
10.        }
11.    </script>
12.    </head>
13.    <body>
14.    <p>
15.        <span onclick="changeBg('red');">Red</span> |
16.        <span onclick="changeBg('white');">White</span>
17.    </p>
18.    </body>
19.    </html>
```

### Code Explanation

As you can see, when calling the `changeBg( )` function, we pass a value (e.g., 'red'), which is assigned to the `color` variable. We can then refer to the `color` variable throughout the function. Variables created in this way are called function arguments or parameters. A function can have any number of arguments, separated by commas.

Adding parameters to functions makes them reusable and, thus, more useful; as you saw above, we can call the `changeBg( )` function many times, passing to it a different color as needed. We can make our functions even more useful by providing default values for parameters so that, if the function is called without a parameter, we assign some default to the parameter. (Note that, since the ability to add a default value is recent addition to JavaScript, this won't work in all browsers.) Here's how we might modify our earlier example:

## Code Sample

### JavaScriptFunctions/Demos/PassingValuesDefaultParam.html

```

1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Passing Values - Default Param</title>
6.    <link href="style.css" rel="stylesheet">
7.    <script>
8.        function changeBg(color='blue'){
9.            document.body.style.backgroundColor = color;
10.        }
11.    </script>
12.    </head>
13.    <body>
14.    <p>
15.        <span onclick="changeBg('red');">Red</span> |
16.        <span onclick="changeBg('white');">White</span> |
17.        <span onclick="changeBg();">Blue (no param)</span>
18.    </p>
19.    </body>
20.    </html>

```

## Code Explanation

We've added a default value for function `changeBg`'s `color` parameter, giving it the value `blue` if no value is supplied when the function is called. We've also added a third `span` on which the user can click; here we call `changeBg ( )` (without a parameter for `color`) and thus get the default color "blue".

## A Note on Variable Scope

Variables created through function arguments or declared within a function with `var` are local to the function, meaning that they cannot be accessed outside of the function.

Variables declared with `var` outside of a function and variables that are used without being declared are global, meaning that they can be used anywhere on the page.

## **Exercise 6   Writing a JavaScript Function**

*15 to 25 minutes*

In this exercise, you will modify a page called [ColorMania.html](#), which will contain a form <sup>6</sup> with four buttons. Each button will show the name of a color (e.g., red) and, when clicked, call a function that changes the background color. The buttons you will create will be of type `button`. For example,

```
<input type="button" value="red"
onclick="functionCall();" >
```

1. Open [JavaScriptFunctions/Exercises/ColorMania.html](#) for editing.
2. Write code to prompt the user for her name.

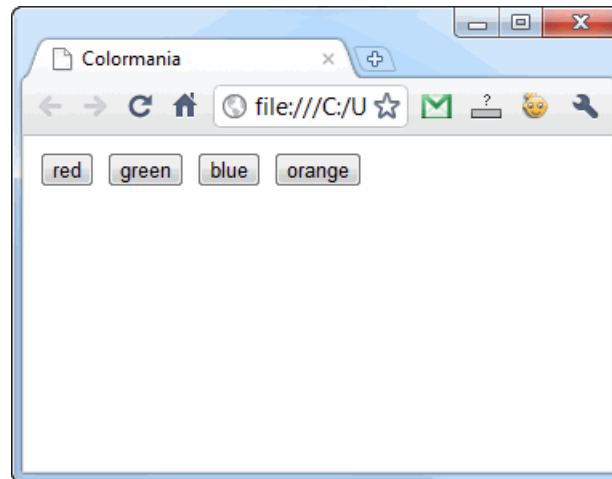
---

6. The method and action attributes can be omitted as this form is not ever submitted.



3. Write a function called `changeBg ( )` that changes the background color and then pops up an alert telling the user, by name, what the new background color is.
4. In the form, add four buttons that, when clicked, call the `changeBg ( )` function and pass it a color value.

The resulting page should look like this:



### Exercise Code

#### JavaScriptFunctions/Exercises/ColorMania.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Colormania</title>
6.    <script>
7.        //PROMPT USER FOR NAME
8.
9.        /*
10.     Write a function called changeBg() that changes the background
11.     color and then pops up an alert telling the user, by name, what
12.     the new background color is.
13.     */
14.    </script>
15.    </head>
16.    <body>
17.    <form>
18.        <!--ADD BUTTONS HERE-->
19.    </form>
20.    </body>
21.    </html>
```

#### **\*Challenge**

Add another button called "custom" that, when clicked, prompts the user for a color, then changes the background color to the user-entered color and alerts the user to the change.



### Exercise Solution

#### JavaScriptFunctions/Solutions/ColorMania.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Colormaniaman</title>
6.    <script>
7.        var userName = prompt("Your name?", "");
8.
9.        function changeBg(color){
10.            document.body.style.backgroundColor = color;
11.            alert(userName + ", the background color is " + color + ".");
12.        }
13.    </script>
14.    </head>
15.    <body>
16.    <form>
17.        <input type="button" value="red" onclick="changeBg('red');">
18.        <input type="button" value="green" onclick="changeBg('green');">
19.        <input type="button" value="blue" onclick="changeBg('blue');">
20.        <input type="button" value="orange" onclick="changeBg('orange');">
21.    </form>
22.    </body>
23.    </html>
```

## Challenge Solution

### JavaScriptFunctions/Solutions/ColorMania-challenge.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Colormania</title>
6.    <script>
7.        var userName = prompt("Your name?", "");
8.
9.        function changeBg(color){
10.            document.body.style.backgroundColor = color;
11.            alert(userName + ", the background color is " + color + ".");
12.        }
13.
14.        function customBg(){
15.            var userColor = prompt("Choose a color:", "");
16.            changeBg(userColor);
17.        }
18.    </script>
19. </head>
20. <body>
21. <form>
22.     <input type="button" value="red" onclick="changeBg('red');">
23.     <input type="button" value="green" onclick="changeBg('green');">
24.     <input type="button" value="blue" onclick="changeBg('blue');">
25.     <input type="button" value="orange" onclick="changeBg('orange');">
26.     <input type="button" value="custom" onclick="customBg();">
27. </form>
28. </body>
29. </html>
```

## Returning Values from Functions

The `return` keyword is used to return values from functions as the following example illustrates:

### Code Sample

#### JavaScriptFunctions/Demos/ReturnValue.html

```
-----Lines 1 through 6 Omitted-----
7.  <script>
8.  function setBgColor(){
9.    document.body.style.backgroundColor = prompt("Set Background Color:",
    >>>  "");
10. }
11.
12. function getBgColor(){
13.   return document.body.style.backgroundColor;
14. }
15. </script>
16. </head>
17. <body>
18. <form>
19.   <input type="button" value="Set Background Color"
20.     onclick="setBgColor();">
21.   <input type="button" value="Get Background Color"
22.     onclick="alert(getBgColor());">
23. </form>
24. </body>
25. </html>
```

### Code Explanation

When the user clicks on the "Get Background Color" button, an alert pops up with a value returned from the `getBgColor()` function. This is a very simple example. Generally, functions that return values are a bit more involved. We'll see many more functions that return values throughout the course.

## 3.3 Conclusion

In this lesson, you have learned to work with JavaScript's global functions and to create functions of your own.

## 4. Event Handlers

### In this lesson, you will learn...

1. To understand what JavaScript event handlers are.
2. To recognize commonly-used event handlers.
3. To use dot notation and square-bracket notation.
4. To get DOM elements with `getElementById()`.
5. To get DOM elements with `querySelector()`.
6. To access element and text nodes hierarchically.

JavaScript event handlers allow us to listen for user actions and to respond to those events with custom code.

### 4.1 Event Handlers

Event handlers are attributes that force an element to "listen" for a specific event to occur. Event handlers all begin with the letters "on".

We might, for instance, listen for a user to click on a specific `div` element, listen for a form submission, or listen for the user to pass his or her mouse over any `input` element of a given class.

The table below lists commonly-used HTML event handlers with descriptions.

---

## Event Handlers

### HTML Event Handlers

Event Handler	Elements Supported	Description
onblur	a, area, button, input, label, select, textarea	the element lost the focus
onchange	input, select, textarea	the element value was changed
onclick	All elements except applet, base, basefont, bdo, br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a pointer button was clicked
ondblclick	All elements except applet, base, basefont, bdo, br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a pointer button was double clicked
onfocus	a, area, button, input, label, select, textarea	the element received the focus
onkeydown	All elements except applet, base, basefont, bdo, br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a key was pressed down
onkeypress	All elements except applet, base, basefont, bdo, br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a key was pressed and released
onkeyup	All elements except applet, base, basefont, bdo, br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a key was released
onload	frameset	all the frames have been loaded
onload	body	the document has been loaded



Event Handler	Elements Supported	Description
onmousedown	All elements except applet, base, basefont, bdo, br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a pointer button was pressed down
onmousemove	All elements except applet, base, basefont, bdo, br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a pointer was moved within
onmouseout	All elements except applet, base, basefont, bdo, br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a pointer was moved away
onmouseover	All elements except applet, base, basefont, bdo, br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a pointer was moved onto
onmouseup	All elements except applet, base, basefont, bdo, br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a pointer button was released
onreset	form	the form was reset
onselect	input, textarea	some text was selected
onsubmit	form	the form was submitted
onunload	frameset	all the frames have been removed
onunload	body	the document has been removed

## The getElementById() Method

A very common way to reference HTML elements is by their ID using the `getElementById()` method of the document object as shown in the example below. Once we have the element - that is, once we get a given `div`, `p`, `input` or

other DOM element via the `getElementById()` method - we can then listen for events on that element. Let's look at an example:

### Code Sample

#### **EventHandlers/Demos/getElementById.html**

```
-----Lines 1 through 8 Omitted-----
9.  <p>
10.  <span onclick="document.getElementById('divRed').style.backgroundColor
    >>> = 'red';">
11.    Red</span> |
12.  <span onclick="document.getElementById('divOrange').style.background >>>
    >>> Color = 'orange';">
13.    Orange</span> |
14.  <span onclick="document.getElementById('divGreen').style.backgroundColor
    >>> = 'green';">
15.    Green</span> |
16.  <span onclick="document.getElementById('divBlue').style.backgroundColor
    >>> = 'blue';">
17.    Blue</span>
18. </p>
19. <div id="divRed">Red</div>
20. <div id="divOrange">Orange</div>
21. <div id="divGreen">Green</div>
22. <div id="divBlue">Blue</div>
23. </body>
24. </html>
```

Clicking the color names listed horizontally across the top of the page sets the style of the corresponding `div` element, whose `id` is gotten via a call to `getElementById()`.

## Exercise 7 Using Event Handlers

*15 to 25 minutes*

In this exercise, you will use some of the event handlers from the table above to allow the user to change the background color of the page.

1. Open [EventHandlers/Exercises/color-changer.html](#) for editing.
2. Modify the page so that...
  - as the page loads, pop up an alert reading "The page is loading!" You can make use of [hello.js](#).
  - when the "Red" button is *clicked*, the background color turns red and an alert pops up reading "The background color will be Red."
  - when the "Green" button is *double-clicked*, the background color turns green and an alert pops up reading "The background color will be Green."
  - when the "Orange" button is *clicked down*, the background color turns orange and an alert pops up reading "The background color will be Orange."
  - when the mouse button is *released* over the "Blue" button, the background color turns blue and an alert pops up reading "The background color will be Blue."

### Exercise Code

#### EventHandlers/Exercises/color-changer.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Color Changer</title>
6.    <link href="style.css" rel="stylesheet">
7.    <script>
8.        window.alert("The page is loading.");
9.    </script>
10.   </head>
11.   <body>
12.   <form>
13.       <p>Click the button to turn the page:
14.       <input type="button" value="Red"></p>
15.       <p>Double click the button to turn the page:
16.       <input type="button" value="Green"></p>
17.       <p>Click down on the button to turn the page:
18.       <input type="button" value="Orange"></p>
19.       <p>Release the mouse while on the button to turn the page:
20.       <input type="button" value="Blue"></p>
21.   </form>
22.   <hr>
23. </body>
24. </html>
```

#### **\*Challenge**

1. Add functionality so that when the user presses any key, the background color turns white.
2. Add a "Black" button. When the user hovers over this button and presses the mouse button down, the background color should turn black. When the user releases the mouse button, the background color should turn white.



**Exercise Solution****EventHandlers/Solutions/color-changer.html**

```
-----Lines 1 through 6 Omitted-----
7.  <script>
8.    window.alert("The page is loading.");
9.  </script>
10. </head>
11. <body>
12. <form>
13.   <p>Click the button to turn the page:
14.   <input type="button" value="Red"
15.     onclick="document.body.style.backgroundColor = 'red';
16.     alert('The background color is now Red.');"></p>
17.   <p>Double click the button to turn the page
18.   <input type="button" value="Green"
19.     ondblclick="document.body.style.backgroundColor = 'green';
20.     alert('The background color is now Green.');"></p>
21.   <p>Click down on the button to turn the page
22.   <input type="button" value="Orange"
23.     onmousedown="document.body.style.backgroundColor = 'orange';
24.     alert('The background color is now Orange.');"></p>
25.   <p>Release the mouse while on the button to turn the page
26.   <input type="button" value="Blue"
27.     onmouseup="document.body.style.backgroundColor = 'blue';
28.     alert('The background color is now Blue.');"></p>
29. </form>
30. <hr>
31. <script src="hello.js"></script>
32. </body>
33. </html>
```

## Challenge Solution

### EventHandlers/Solutions/color-changer-challenge.html

```
-----Lines 1 through 10 Omitted-----
11. <body onkeypress="document.body.style.backgroundColor = 'white';
12.     alert('The background color is now White.');">
13. <form>
-----Lines 14 through 29 Omitted-----
30. <p>Press any key to turn the page back to white.</p>
31. <p>Click the button to turn the page
32. <input type="button" value="Black"
33.     onmousedown="document.body.style.backgroundColor = 'black';"
34.     onmouseup="document.body.style.backgroundColor = 'white';"></p>
35. </form>
36. <hr>
37. <script src="hello.js"></script>
38. </body>
39. </html>
```

## 4.2 Dot Notation and Square Bracket Notation

In the first lesson of this course we took a look at two ways in which we can access elements in JavaScript: dot notation and square bracket notation. Let's review these concepts again, since we will use both tactics repeatedly throughout this course.

Dot notation lets us refer to hierarchical DOM elements starting with the top-most element (window, say) then a set of dot-separated names, referencing elements by their name or id. For instance, to get an input element with id `fname` inside a form with id `form1`, we might use the following:

```
window.document.form1.fname
```

If the form were the first form on the page, we could also refer to it this way:

```
window.document.forms[0].fname
```

Since a document can have multiple form elements as children, we can reference the specific form by its order on the page, as an element in the collection of forms. Arrays and collections in JavaScript start with index 0, so the first form on the page would be `forms[0]`.

Similarly, we can reference objects with square bracket notation, where the argument of the brackets is the `id` of the element or the number-index of the element in a collection:

```
window['document']['form1']['fname']  
  
// or  
  
window['document']['forms'][0]['fname']
```

Both of the references above are equivalent to the dot-notation references we showed earlier and can be used interchangeably. With either dot notation or square bracket notation, keep in mind that the reference moves from general on the left to specific on the right. In our example, the window contains the document, which contains a form with id `form1`, which in turn contains an input element with id `fname`.

## 4.3 CSS Selectors

Here we present an introduction/review of CSS selectors. There are several different types of selectors:



## Types of Selectors

- Type
- Descendant
- Child
- Class
- ID
- Attribute
- Universal

Selectors identify the element(s) affected by the CSS rule.

## Type Selectors

Type selectors specify elements by tag name and affect every instance of that element type. Looking again at a previous example:

```
p {  
  color: darkgreen;  
  font-family: Verdana;  
  font-size: 10pt;  
}
```

This rule specifies that the text of every `<p>` element should be darkgreen and use a 10-point Verdana font.

## Descendant Selectors

Descendant selectors specify elements by ancestry. Each "generation" is separated by a space. For example, the following rule states that `<strong>` tags within `<p>` tags should have red text:

```
p strong {  
  color: red;  
}
```

With descendant selectors generations can be skipped. In other words, the code above does not require that the `<strong>` tag is a direct child of the `<p>` tag.

## Child Selectors

Child selectors specify a direct parent-child relationship and are indicated by placing a `>` sign between the two tag names:

```
p > strong {  
  color: red;  
}
```

In this case only `<strong>` tags that are direct children of `<p>` tags are affected.

## Class Selectors

In HTML, almost all elements can take the `class` attribute, which assigns a class name to an element. The names given to classes are arbitrary, but should be descriptive of the purpose of the class. In CSS, class selectors begin with a dot. For example, the following rule creates a class called "warning," which makes the text of all elements of that class bold and red:

```
.warning {  
  font-weight: bold;  
  color: #ff0000;  
}
```

Following are a couple of examples of elements of the warning class:

```
<h1 class="warning">WARNING</h1>  
<p class="warning">Don't go there!</p>
```

If the class selector is preceded by an element name, then that selector only applies to the specified type of element. To illustrate, the following two rules indicate that `h1` elements of the class "warning" will be underlined, while `p` elements of the class "warning" should not be:

```
h1.warning {  
  color: #ff0000;  
  text-decoration: underline;  
}  
p.warning {  
  color: #ff0000;  
  font-weight: bold;  
}
```

Because both rules indicate that the color should be red (`#ff0000`), this could be rewritten as follows:

```
.warning {
  color: #ff0000;
}
h1.warning {
  text-decoration: underline;
}
p.warning {
  font-weight: bold;
}
```

Note that you can assign an element any number of classes simply by separating the class names with spaces like this:

#### Syntax

##### Assigning Multiple Classes to an Element

```
<div class="class1 class2 class3">...
```

## ID Selectors

As with the `class` attribute, in HTML, almost all elements can take the `id` attribute, which is used to uniquely identify an element on the page. In CSS, ID selectors begin with a pound sign (`#`) and have arbitrary names. The following rule will indent the element with the `"maintext"` id 20 pixels from the left and right:

```
#maintext {
  margin-left: 20px;
  margin-right: 20px;
}

<div id="maintext">
  This is the main text of the page...
</div>
```

## Attribute Selectors

Attribute selectors specify elements that contain a specific attribute. They can also specify the value of that attribute.

The following selector affects all links with a `target` attribute:

```
a[target] {
  color: red;
}
```

The following selector would only affect links whose `target` attribute is `"_blank"`.

```
a[target="_blank"] {  
  color: red;  
}
```

## The Universal Selector

The universal selector is an asterisk (\*). It matches every element.

```
* {  
  color: red;  
}
```

## Grouping

Selectors can share the same declarations by separating them with commas. The following rule will underline all `i` elements, all elements of the class "warning" and the element with the id of "important."

```
i, .warning, #important {  
  text-decoration: underline;  
}
```

## 4.4 `querySelector()`

We can exploit the various CSS selectors, as described above, by using `querySelector()`. The `document.querySelector()` method returns the first element found that matches its argument. Unlike the `getElementById()` method, which finds an element by its id, `document.querySelector()` offers us a way to find an element by tag name, class name, or id. The following line of code would return the first list item found in an `ol` tag:

```
var firstOrderedListItem = document.querySelector("ol>li");
```

Let's look at an example:

## Code Sample

### EventHandlers/Demos/query-selector.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>querySelector()</title>
6.    <link href="style.css" rel="stylesheet">
7.    <script>
8.        function setColors() {
9.            var firstParagraph = document.querySelector("p");
10.           firstParagraph.style.backgroundColor='red';
11.            var spanText = document.querySelector("span");
12.            spanText.style.backgroundColor='blue';
13.            var firstListItem = document.querySelector("li");
14.            firstListItem.style.backgroundColor='green';
15.            var listItemClassSpecial = document.querySelector("li.special");
16.            listItemClassSpecial.style.fontStyle='italic';
17.        }
18.    </script>
19.    </head>
20.    <body onload="setColors();">
21.        <p>This is the first paragraph. This is the first paragraph. This is
            >>> the first paragraph. This is the first paragraph. This is the
            >>> first paragraph. This is the first paragraph.</p>
22.        <p>This is the second paragraph. This is the second paragraph. This
            >>> is the second paragraph. This is the second paragraph.
            >>> <span>This text is wrapped in a span tag.</span> This is the
            >>> second paragraph. This is the second paragraph. This is the
            >>> second paragraph.</p>
23.        <ul>
24.            <li>List item 1</li>
25.            <li>List item 2</li>
26.            <li class="special">List item 3</li>
27.            <li>List item 4</li>
28.        </ul>
29.    </body>
30.    </html>
```

### Code Explanation

The markup for our page has two paragraphs, the second of which includes some text wrapped in a `span` tag, followed by an unordered list with four items; the third list item has class `special`.

We use the event handler `onload()` to run our custom function `setColors()`; the `onload` event handler fires only after the DOM has been loaded for the page, ensuring that all tags, ids, and classes have been loaded and are accessible to our JavaScript code.

We use `document.querySelector()` to:

- Find the first paragraph, and color it red.
- Find the first `span`, and color it blue.
- Find the first list item, and color it green.
- Find the first list item with class `special`, and style it with italic text.

Let's have you try out using `querySelector()` with an exercise:

## Exercise 8 Working with `querySelector()`

*10 to 15 minutes*

In this exercise, you will practice working with JavaScript's `querySelector()` function.

1. Open [EventHandlers/Exercises/query-selector.html](#) for editing.
2. Add a click handler to the button so that the function `setColor()` is called when the user clicks the button.
3. Write the contents of function `setColor()` to do the following:
  - Use `prompt()` to get from the user the element he/she wishes to color - expected input would be a tag (like `p`), a class (like `.a`), or an id (like `#x`).
  - Use `prompt()` to get from the user their desired color (like `blue` or `red`).
  - Use `querySelector()` to get the user's desired element, then set the color of that element.

**Exercise Solution****EventHandlers/Solutions/query-selector.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>querySelector()</title>
6.    <link href="style.css" rel="stylesheet">
7.    <script>
8.        function setColor() {
9.            var userElement = prompt("Element to color?", "");
10.           var userColor = prompt("What color?", "");
11.           document.querySelector(userElement).style.backgroundColor = userColor;
12.               >>>
13.        }
14.    </script>
15.    </head>
16.    <body>
17.        <button onclick="setColor();" type="button">Go</button>
18.        <ul>
19.            <li>Item 1</li>
20.            <li class="a">Item 2 | class "a"</li>
21.            <li>Item 3</li>
22.            <li class="b">Item 4 | class "b"</li>
23.            <li>Item 5</li>
24.            <li class="c">Item 6 | class "c"</li>
25.            <li>Item 7</li>
26.            <li id="x">Item 8 | id "x"</li>
27.            <li class="a">Item 9 | class "a"</li>
28.            <li>Item 10</li>
29.        </ul>
30.    </body>
31. </html>
```

**Code Explanation**

We add `onclick="setColor();"`  as an attribute to the button at the top of the page so that function `setColor()` is called when the user clicks the button.

We use `prompt()` to get from the user the desired element (storing it in variable `userElement`) and the desired color (storing it in `userColor`).

Lastly, we use the code



```
document.querySelector(userElement).style.backgroundColor = userCol »»  
or;
```

to set the color of the first matched element supplied by the user:

- If the user entered "ul" and "green", the entire unordered list would be colored green.
- If the user entered "li" and "red", the first bullet would be colored red.
- If the user entered ".b" and "blue", the fourth bullet would be colored blue.
- If the user entered "#x" and "orange", the eighth bullet would be colored orange.
- If the user entered ".w" and "red", nothing would happen, as there is no element with class "w". (Actually an error would occur, as the returned value of `document.querySelector( )` would be null.)

## 4.5 Accessing Element and Text Nodes Hierarchically

JavaScript provides a variety of methods and properties that provide access to nodes - elements on the page - based on their hierarchical relationship. The most common and best supported of these are shown in the table below:

### Properties for Accessing Element Nodes

Property	Description
<code>childNodes[ ]</code>	A <code>nodeList</code> containing all of a node's child nodes.
<code>firstChild</code>	A reference to a node's first child node.
<code>lastChild</code>	A reference to a node's last child node.
<code>nextSibling</code>	A reference to the next node at the same level in the document tree.
<code>parentNode</code>	A reference to a node's parent node.
<code>previousSibling</code>	A reference to the previous node at the same level in the document tree.

As suggested by the square brackets, `childNodes[ ]` returns a collection of elements. The other properties return a single node.

These properties offer us a more flexible way to get elements on the page, relative to their parents, siblings, or children. We can do anything with the returned elements that we did previously with `getElementById( )` or `querySelector( )` - set the background color, change the font style, etc.

In the example below, we'll use the `innerHTML` property to get the HTML text contained inside the nodes we get. Let's take a look at how we might use these properties:

## Code Sample

### EventHandlers/Demos/elementNodes.html

```

1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Accessing Element Nodes</title>
6.    <link href="style.css" rel="stylesheet">
7.    <script>
8.        function modify() {
9.            var childFirst = document.getElementById("list1").firstChild;
10.           childFirst.style.backgroundColor = "red";
11.           var childLast = document.getElementById("list1").lastChild;
12.           childLast.style.backgroundColor = "blue";
13.           var siblingPrev = childLast.previousSibling;
14.           siblingPrev.style.backgroundColor = "green";
15.           document.getElementById("message").innerHTML = 'First child text is
                >>> <strong>'+childFirst.innerHTML+'</strong><br><br>Last child
                >>> text is <strong>'+childLast.innerHTML+'</strong><br><br>Next
                >>> to last (previous sibling) text is <strong>'+siblingPrev.inner >>>
                >>> HTML+'</strong>';
16.        }
17.    </script>
18.    </head>
19.    <body>
20.        <button type="button" onclick="modify()">Go</button>
21.        <ul id="list1"><li>Item 1</li><li>Item 2</li><li>Item 3</li><li>Item
                >>> 4</li><li>Item 5</li></ul>
22.        <p id="message"></p>
23.    </body>
24.    </html>

```

## Code Explanation

Our simple page displays a button and five unordered list items, with text "Item 1", "Item 2", etc.

When the user clicks the button, the `onclick` event handler calls function `Modify()`, which does the following:

- Gets the first child of the list using `firstChild`, and sets its background to red.
- Gets the last child of the list using `lastChild`, and sets its background to blue.

- Gets the next-to-last child of the list using `previousSibling` (relative to the already-gotten `childLast`), and sets its background to green.
- Sets the `innerHTML` of the paragraph (at the bottom of the page) to display the text from the first, last, and next-to-last list items we got earlier. Note that we use the JavaScript string concatenation operator, `+`, to build a string from our variables.

Note the special way in which we created the unordered list, with no spaces between any of the tags - that is, with all of the `ul` and `li` opening and closing tags on the same line. We did this because, if there were spaces between the tags, the spaces would have counted as "children" (as text nodes) and calls to properties like `firstChild` would have returned the text node (the space) rather than the first list item as we wished.

We'll ask you to try out these properties in the next exercise.

## Exercise 9 Working with Hierarchical Node Properties

*10 to 15 minutes*

In this exercise, you will practice working with JavaScript's hierarchical node properties.

1. Open [EventHandlers/Exercises/elementNodes.html](#) for editing.
2. Note that a click handler has been added to the button so that the function `setFontSize()` is called when the user clicks the button.
3. Write the body of function `setFontSize()` to set the size of the font for each list item to be successively greater, with the first list item having font size 10px, the second 12px, etc.

**Exercise Solution****EventHandlers/Solutions/elementNodes.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Element Nodes</title>
6.    <link href="style.css" rel="stylesheet">
7.    <script>
8.        function setFontSize() {
9.            var list = document.querySelector("ul");
10.           var childFirst = list.firstChild;
11.           childFirst.style.fontSize = "10px";
12.           var nextSib = childFirst.nextSibling;
13.           nextSib.style.fontSize = "12px";
14.           nextSib = nextSib.nextSibling;
15.           nextSib.style.fontSize = "14px";
16.           nextSib = nextSib.nextSibling;
17.           nextSib.style.fontSize = "16px";
18.           nextSib = nextSib.nextSibling;
19.           nextSib.style.fontSize = "18px";
20.           nextSib = nextSib.nextSibling;
21.           nextSib.style.fontSize = "20px";
22.           nextSib = nextSib.nextSibling;
23.           nextSib.style.fontSize = "22px";
24.           nextSib = nextSib.nextSibling;
25.           nextSib.style.fontSize = "24px";
26.           nextSib = nextSib.nextSibling;
27.           nextSib.style.fontSize = "26px";
28.           nextSib = nextSib.nextSibling;
29.           nextSib.style.fontSize = "28px";
30.       }
31.    </script>
32.    </head>
33.    <body>
34.        <button onclick="setFontSize()" type="button">Go</button>
35.        <ul><li>Item 1</li><li>Item 2</li><li>Item 3</li><li>Item 4</li><li>Item
            >>> 5</li><li>Item 6</li><li>Item 7</li><li>Item 8</li><li>Item
            >>> 9</li><li>Item 10</li></ul>
36.    </body>
37.    </html>
```

**Code Explanation**

In function `setFontSize()` we use `document.querySelector("ul")` to get the unordered list, `firstChild` to get the first list item, and `nextSibling` to get each next item. Of course, if we were to use a loop (at which we'll look later in the course) we could have written much more economical code.

## 4.6 Conclusion

In this lesson, you have learned

- How to use event handlers to respond to user events.
- How to access elements with `getElementById()`, `querySelector()`, and the hierarchical node properties like `firstChild()`.



## 5. Built-In JavaScript Objects

### In this lesson, you will learn...

1. To work with the built-in String object.
2. To work with the built-in Math object.
3. To work with the built-in Date object.

JavaScript has some predefined, built-in objects that enable you to work with Strings and Dates, and perform mathematical operations.

### 5.1 String

In JavaScript, there are two types of string data types: primitive strings and *String* objects. String objects have many methods for manipulating and parsing strings of text. Because these methods are available to primitive strings as well, in practice, there is no need to differentiate between the two types of strings.<sup>7</sup>

Some common string properties and methods are shown below. In all the examples, the variable `webucator` contains "Webucator".

#### Common String Properties

Property	Description
length	Read-only value containing the number of characters in the string.
	<code>webucator.length; //Returns 9</code>

7. If you're feeling geeky, you can take a look at [BuiltInObjects/Demos/stringsAndStrings.html](#), which compares primitive type strings to String objects.

**Common String Methods**

Method	Description
<code>charAt(position)</code>	Returns the character at the specified position.
	<code>webucator.charAt(4); //Returns c</code>
<code>charCodeAt(position)</code>	Returns the Unicode character code of the character at the specified position.
	<code>webucator.charCodeAt(4); //Returns 99</code>
<code>fromCharCode(charCodes)</code>	Returns the text representation of the specified comma-delimited character code. Used with String rather than a specific String object.
	<code>String.fromCharCode(169); //Returns</code>
<code>indexOf(substr, startPos)</code>	Searches from <code>startPos</code> (or the beginning of the string, if <code>startPos</code> is not supplied) for <code>substr</code> . Returns the first position at which <code>substr</code> is found or -1 if <code>substr</code> is not found.
	<code>webucator.indexOf("cat"); //Returns 4</code>
	<code>webucator.indexOf("cat", 5); //Returns -1</code>
<code>lastIndexOf(substr, endPos)</code>	Searches from <code>endPos</code> (or the end of the string, if <code>endPos</code> is not supplied) for <code>substr</code> . Returns the last position at which <code>substr</code> is found or -1 if <code>substr</code> is not found.
	<code>webucator.lastIndexOf("cat"); //Returns 4</code>
	<code>webucator.lastIndexOf("cat", 5); //Returns 4</code>
<code>substring(startPos, endPos)</code>	Returns the substring beginning at <code>startPos</code> and ending with the character before <code>endPos</code> . <code>endPos</code> is optional. If it is excluded, the substring continues to the end of the string.
	<code>webucator.substring(4, 7); //Returns cat</code>
	<code>webucator.substring(4); //Returns cator</code>

Method	Description
<code>substr(startPos, len)</code>	Returns the substring of Length characters beginning at startPos. len is optional. If it is excluded, the substring continues to the end of the string.
	<pre>webucator.substr(4, 3); //Returns cat webucator.substr(4); //Returns cator</pre>
<code>slice(startPos, endPos)</code>	Same as <code>substring(startPos, endPos)</code> .
	<pre>webucator.slice(4, 7); //Returns cat</pre>
<code>slice(startPos, posFromEnd)</code>	posFromEnd is a negative integer. Returns the substring beginning at startPos and ending posFromEnd characters from the end of the string.
	<pre>webucator.slice(4, -2); //Returns cat</pre>
<code>split(delimiter)</code>	Returns an array by splitting a string on the specified delimiter.
	<pre>var s = "A,B,C,D"; var a = s.split(","); document.write(a[2]); //Returns C</pre>
<code>toLowerCase()</code>	Returns the string in all lowercase letters.
	<pre>webucator.toLowerCase() //Returns webucator</pre>
<code>toUpperCase()</code>	Returns the string in all uppercase letters.
	<pre>webucator.toUpperCase(); //Returns WEBUCATOR</pre>

You can see these examples in a browser by opening [BuiltInObjects/Demos/String PropertiesAndMethods.html](#).

## 5.2 Math

The `Math` object is a built-in static object. The `Math` object's properties and methods are accessed directly (e.g., `Math.PI`) and are used for performing complex math operations. Some common math properties and methods are shown below:

**Common Math Properties**

Property	Description
Math.PI	The value of Pi ( $\pi$ )
	Math.PI; //3.141592653589793
Math.SQRT2	Square root of 2.
	Math.SQRT2; //1.4142135623730951

**Common Math Methods**

Method	Description
Math.abs(number)	Absolute value of number.
	Math.abs(-12); //Returns 12
Math.ceil(number)	number rounded up.
	Math.ceil(5.4); //Returns 6
Math.floor(number)	number rounded down.
	Math.floor(5.6); //Returns 5
Math.max(numbers)	Highest Number in numbers.
	Math.max(2, 5, 9, 3); //Returns 9
Math.min(numbers)	Lowest Number in numbers.
	Math.min(2, 5, 9, 3); //Returns 2
Math.pow(number, power)	number to the power of power.
	Math.pow(2, 5); //Returns 32
Math.round(number)	Rounded number.
	Math.round(2.5); //Returns 3
Math.random()	Random number between 0 and 1.
	Math.random(); //Returns random //number from 0 to 1

You can see these examples in a browser by opening [BuiltInObjects/Demos/Math PropertiesAndMethods.html](#).

## Method for Generating Random Integers

Because `Math.random()` returns a decimal value greater than or equal to 0 and less than 1, we can use the following code to return a random integer between `low` and `high`, inclusively:

```
var low = 1;
var high = 10;
var rndDec = Math.random();
var rndInt = Math.floor(rndDec * (high - low + 1) + low);
```

## 5.3 Date

The *Date* object has methods for manipulating dates and times. JavaScript stores dates as the number of milliseconds since January 1, 1970. The sample below shows the different methods of creating date objects, all of which involve passing arguments to the `Date()` constructor.

### Code Sample

#### BuiltInObjects/Demos/DateObject.html

```
-----Lines 1 through 6 Omitted-----
7.  <body>
8.  <h1>Date Object</h1>
9.  <h2>New Date object with current date and time</h2>
10. <pre>
11.  //Syntax: new Date();
12.  var now = new Date();
13. </pre>
14. <strong>Result:</strong>
15. <script>
16.   var now = new Date();
17.   document.write(now);
18. </script>
19.
20. <h2>New Date object with specific date and time</h2>
21. <pre>
22.  //Syntax: new Date("month dd, yyyy hh:mm:ss");
23.  var redSoxWin = new Date("October 21, 2004 12:01:00");
24. </pre>
25. <strong>Result:</strong>
26. <script>
27.   var redSoxWin = new Date("October 21, 2004 12:01:00");
28.   document.write(redSoxWin);
29. </script>
30.
31. <pre>
32.  //Syntax: new Date(year, month, day, hours, min, sec, millisec);
33.  redSoxWin = new Date(2004, 9, 21, 12, 01, 00, 00);
34. </pre>
35. <strong>Result:</strong>
36. <script>
37.   redSoxWin = new Date(2004, 9, 21, 12, 01, 00, 00);
38.   document.write(redSoxWin);
39. </script>
40. </body>
41. </html>
```

## Code Explanation

This page is shown in a browser below.

# Date Object

## New Date object with current date and time

```
//Syntax: new Date();  
var now = new Date();
```

**Result:** Thu Mar 02 2017 17:29:18 GMT-0500 (EST)

## New Date object with specific date and time

```
//Syntax: new Date("month dd, yyyy hh:mm:ss");  
var redSoxWin = new Date("October 21, 2004 12:01:00");
```

**Result:** Thu Oct 21 2004 12:01:00 GMT-0400 (EDT)

```
//Syntax: new Date(year, month, day, hours, min, sec, millisec);  
redSoxWin = new Date(2004, 9, 21, 12, 01, 00, 00);
```

**Result:** Thu Oct 21 2004 12:01:00 GMT-0400 (EDT)

A few things to note:

1. To create a `Date` object containing the current date and time, the `Date ( )` constructor takes no arguments.
2. When passing the date as a string to the `Date ( )` constructor, the time portion is optional. If it is not included, it defaults to 00:00:00. Also, other date formats are acceptable (e.g., "3/2/2017" and "03-02-2017").
3. When passing date parts to the `Date ( )` constructor, `dd`, `hh`, `mm`, `ss`, and `ms` are all optional. The default for `dd` is 1; the other parameters default to 0.
4. Months are numbered from 0 (January) to 11 (December). In the example above, 9 represents October.

Some common date methods are shown below. In all the examples, the variable `rightNow` contains "Thu Mar 02 00:23:54:650 EDT 2017".

**Common Date Methods**

Method	Description
getDate( )	Returns the day of the month (1-31).
	<code>rightNow.getDate();</code> //Returns 2
getDay( )	Returns the day of the week as a number (0-6, 0=Sunday, 6=Saturday).
	<code>rightNow.getDay();</code> //Returns 4
getMonth( )	Returns the month as a number (0-11, 0=January, 11=December).
	<code>rightNow.getMonth();</code> //Returns 2
getFullYear( )	Returns the four-digit year.
	<code>rightNow.getFullYear();</code> //Returns 2017
getHours( )	Returns the hour (0-23).
	<code>rightNow.getHours();</code> //Returns 0
getMinutes( )	Returns the minute (0-59).
	<code>rightNow.getMinutes();</code> //Returns 23
getSeconds( )	Returns the second (0-59).
	<code>rightNow.getSeconds();</code> //Returns 54
getMilliseconds( )	Returns the millisecond (0-999).
	<code>rightNow.getMilliseconds();</code> //Returns 650
getTime( )	Returns the number of milliseconds since midnight January 1, 1970.
	<code>rightNow.getTime();</code> //Returns 1488432234650
getTimezoneOffset( )	Returns the time difference in minutes between the user's computer and GMT.
	<code>rightNow.getTimezoneOffset();</code> //Returns 300



Method	Description
<code>toLocaleString()</code>	Returns the Date object as a string.  <code>rightNow.toLocaleString();</code> //Returns 3/2/2017, 12:23:54 AM
<code>toGMTString()</code>	Returns the Date object as a string in GMT timezone.  <code>rightNow.toGMTString();</code> //Returns Thu, 02 Mar 2017 05:44:30 GMT

You can see these examples in a browser by opening [BuiltInObjects/Demos/DateMethods.html](#).

## 5.4 The typeof Operator

The `typeof` operator is used to find out the type of a piece of data. The screenshot below (of [BuiltInObjects/Demos/typeof.html](#)) shows what the `typeof` operator returns for different data types:

Type	Example	Returns
Boolean	<code>typeof false</code>	boolean
Number	<code>typeof 5</code>	number
String	<code>typeof "hello"</code>	string
Object	<code>typeof []</code>	object
Function	<code>typeof function() {}</code>	function
Function	<code>typeof window.alert</code>	function
Object	<code>typeof window</code>	object
Null	<code>typeof null</code>	object
undefined	<code>typeof undefined</code>	undefined
foo	<code>typeof foo</code>	undefined

## 5.5 Helper Functions

Some languages have functions that return the month as a string. JavaScript doesn't have such a built-in function. The sample below shows a user-defined "helper"

function that handles this and how the `getMonth()` method of a `Date` object can be used to get the month.

### Code Sample

#### BuiltInObjects/Demos/MonthAsString.html

```
-----Lines 1 through 5 Omitted-----
6.  <script>
7.    function monthAsString(num){
8.      var months = ["January", "February", "March", "April", "May", "June",
          >>> "July", "August", "September", "October", "November", "Decem >>>
          >>> ber"];
9.      return months[num-1];
10.   }
11.
12.  function enterMonth(){
13.    var userMonth = prompt("What month were you born?", "");
14.    alert("You were born in " + monthAsString(userMonth) + ".");
15.  }
16.
17.  function getCurrentMonth(){
18.    var today = new Date();
19.    alert(monthAsString(today.getMonth()+1));
20.  }
21. </script>
-----Lines 22 through 29 Omitted-----
```

## Exercise 10 Returning the Day of the Week as a String

*15 to 25 minutes*

In this exercise, you will create a function that returns the day of the week as a string.

1. Open [BuiltInObjects/Exercises/DateUDFs.html](#) for editing.
2. Write a `dayAsString()` function that returns the day of the week as a string, with "0" returning "Sunday", "1" returning "Monday", etc.
3. Write an `enterDay()` function that prompts the user for the day of the week (as a number) and then alerts the string value of that day by calling the `dayAsString()` function.
4. Write a `getCurrentDay()` function that alerts today's actual day of the week according to the user's machine.
5. Add a "CHOOSE DAY" button that calls the `enterDay()` function.
6. Add a "GET CURRENT DAY" button that calls the `getCurrentDay()` function.
7. Test your solution in a browser.

**Exercise Solution****BuiltInObjects/Solutions/DateUDFs.html**

```
-----Lines 1 through 5 Omitted-----
6.  <script>
-----Lines 7 through 24 Omitted-----
25.  function dayAsString(num){
26.      var weekDays = [];
27.      weekDays[0] = "Sunday";
28.      weekDays[1] = "Monday";
29.      weekDays[2] = "Tuesday";
30.      weekDays[3] = "Wednesday";
31.      weekDays[4] = "Thursday";
32.      weekDays[5] = "Friday";
33.      weekDays[6] = "Saturday";
34.
35.      return weekDays[num-1];
36.  }
-----Lines 37 through 41 Omitted-----
42.
43.  function getCurrentMonth(){
44.      var today = new Date();
45.      alert(monthAsString(today.getMonth()+1));
46.  }
47.
48.  function enterDay(){
49.      var userDay = prompt("What day of the week is it?", "");
50.      alert("Today is " + dayAsString(userDay) + ".");
51.  }
52.
53.  function getCurrentDay(){
54.      var today = new Date();
55.      alert(dayAsString(today.getDay()+1));
56.  }
57.  </script>
-----Lines 58 through 60 Omitted-----
61.  <input type="button" value="CHOOSE MONTH" onclick="enterMonth();">
62.  <input type="button" value="GET CURRENT MONTH" onclick="getCurrent »»
    >>> Month();"><br>
63.  <input type="button" value="CHOOSE DAY" onclick="enterDay();">
64.  <input type="button" value="GET CURRENT DAY" onclick="getCurrentDay();">
    >>>
-----Lines 65 through 67 Omitted-----
```

## **5.6 Conclusion**

In this lesson, you have learned to work with some of JavaScript's most useful built-in objects.



## 6. Conditionals and Loops

**In this lesson, you will learn...**

1. To write `if - else if - else` blocks.
2. To write `switch / case` blocks.
3. To work with loops in JavaScript.

### 6.1 Conditionals

There are two types of conditionals in JavaScript.

1. `if - else if - else`
2. `switch / case`

#### **if - else if - else Conditions**

##### **Syntax**

```
if (conditions) {  
  statements;  
} else if (conditions) {  
  statements;  
} else {  
  statements;  
}
```

Like with functions, each part of the `if - else if - else` block is contained within curly brackets (`{ }`). There can be zero or more `else if` blocks. The `else` block is optional.

**Comparison Operators**

Operator	Description
==	Equals
!=	Doesn't equal
===	Strictly equals
!==	Doesn't strictly equal
>	Is greater than
<	Is less than
>=	Is greater than or equal to
<=	Is less than or equal to

Note the difference between == (equals) and === (strictly equals). For two objects to be **strictly equal** they must be of the same value **and** the same type, whereas to be **equal** they must only have the same value. See the code sample below:



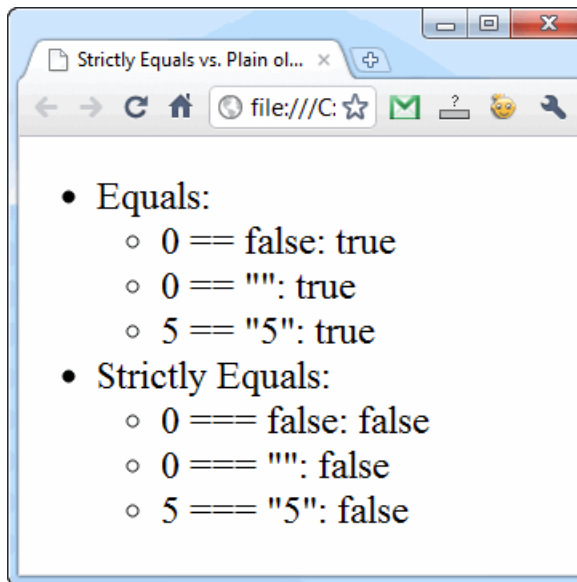
**Code Sample****ConditionalsAndLoops/Demos/StrictlyEquals.html**

```
-----Lines 1 through 8 Omitted-----
9.     <li>Equals:
10.     <ul>
11.         <li>0 == false:
12.             <script>
13.                 document.write(0 == false);
14.             </script>
15.         </li>
16.         <li>0 == "":
17.             <script>
18.                 document.write(0 == "");
19.             </script>
20.         </li>
21.         <li>5 == "5":
22.             <script>
23.                 document.write(5 == "5");
24.             </script>
25.         </li>
26.     </ul>
27. </li>
28. <li>Strictly Equals:
29.     <ul>
30.         <li>0 === false:
31.             <script>
32.                 document.write(0 === false);
33.             </script>
34.         </li>
35.         <li>0 === "":
36.             <script>
37.                 document.write(0 === "");
38.             </script>
39.         </li>
40.         <li>5 === "5":
41.             <script>
42.                 document.write(5 === "5");
43.             </script>
44.         </li>
45.     </ul>
46. </li>
```

-----Lines 47 through 49 Omitted-----

### Code Explanation

Open this file in your browser to see that all of the value pairs are equal, but none is strictly equal:



### Logical Operators

Operator	Description
&&	and (a == b && c != d)
	or (a == b    c != d)
!	not !(a == b    c != d)

The example below shows a function using an if - else if - else condition.

**Code Sample****ConditionalsAndLoops/Demos/IfElseifElse.html**

```
-----Lines 1 through 6 Omitted-----
7.  <script>
8.    function checkAge(){
9.      var age = prompt("Your age?", "") || "";
10.
11.      if (age >= 21) {
12.        alert("You can vote and drink!");
13.      } else if (age >= 18) {
14.        alert("You can vote, but can't drink.");
15.      } else {
16.        alert("You cannot vote or drink.");
17.      }
18.    }
19.  </script>
-----Lines 20 through 23 Omitted-----
24.  <form>
25.    <input type="button" value="Age Check" onclick="checkAge();">
26.  </form>
-----Lines 27 through 28 Omitted-----
```

### Code Explanation

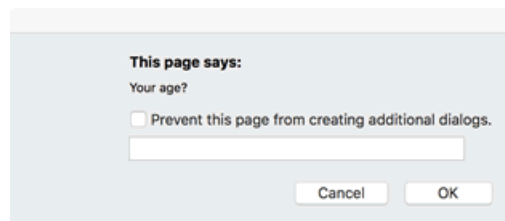
The display of the page is shown below:

## JavaScript if - else if - else Demo

### Age Check

Age Check

When the user clicks on the [Age Check](#) button, the following prompt pops up:



After the user enters his age, an alert pops up. The text of the alert depends on the user's age. The three possibilities are shown below:



### Compound Conditions

Compound conditions are conditions that check for multiple things. See the sample below.

```
if (age > 18 && isCitizen) {
    alert("You can vote!");
}

if (age >= 16 && (isCitizen || hasGreenCard)) {
    alert("You can work in the United States");
}
```

## Short-circuiting

JavaScript is lazy (or efficient) about processing compound conditions. As soon as it can determine the overall result of the compound condition, it stops looking at the remaining parts of the condition. This is useful for checking that a variable is of the right data type before you try to manipulate it.

To illustrate, take a look at the following sample:

### Code Sample

#### ConditionalsAndLoops/Demos/PasswordCheckBroken.html

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>Password Check</title>
6.  <script>
7.      var userPass = prompt("Password:", ""); //ESC here causes problems
8.      var password = "xyz";
9.  </script>
10. </head>
11. <body>
12. <script>
13.     if (userPass.toLowerCase() == password) {
14.         document.write("<h1>Welcome!</h1>");
15.     } else {
16.         document.write("<h1>Bad Password!</h1>");
17.     }
18. </script>
19. </body>
20. </html>
```

### Code Explanation

Everything works fine as long as the user does what you expect. However, if the user clicks on the Cancel button when prompted for a password, the value `null` will be assigned to `userPass`. Because `null` is not a string, it does not have the `toLowerCase()` method. So the following line will result in a JavaScript error:

```
if (userPass.toLowerCase() == password)
```

This can be fixed by using the `typeof()` function to first check if `userPass` is a string as shown in the sample below.

### Code Sample

#### ConditionalsAndLoops/Demos/PasswordCheck.html

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>Password Check</title>
6.  <script>
7.    var userPass = prompt("Password:", "") || "";
8.    var password = "xyz";
9.  </script>
10. </head>
11. <body>
12. <script>
13.   if (typeof userPass == "string" && userPass.toLowerCase() == password)
14.     >>> {
15.       document.write("<h1>Welcome!</h1>");
16.     } else {
17.       document.write("<h1>Bad Password!</h1>");
18.     }
19. </script>
20. </body>
    </html>
```

## Switch / Case

### Syntax

```
switch (expression) {  
  case value :  
    statements;  
  case value :  
    statements;  
  default :  
    statements;  
}
```

Like if - else if - else statements, switch/case statements are used to run different code at different times. Generally, switch/case statements run faster than if - else if - else statements, but they are limited to checking for equality. Each case is checked to see if the *expression* matches the *value*.

Take a look at the following example:

### Code Sample

#### ConditionalsAndLoops/Demos/SwitchWithoutBreak.html

```
1.  <!DOCTYPE HTML>  
2.  <html>  
3.  <head>  
4.  <meta charset="UTF-8">  
5.  <title>Switch</title>  
6.  <script>  
7.    var quantity = 1;  
8.    switch (quantity) {  
9.      case 1 :  
10.       alert("quantity is 1");  
11.      case 2 :  
12.       alert("quantity is 2");  
13.      default :  
14.       alert("quantity is not 1 or 2");  
15.    }  
16.  </script>  
17. </head>  
18. <body>  
19.   <p>Nothing to show here.</p>  
20. </body>  
21. </html>
```

### Code Explanation

When you run this page in a browser, you'll see that all three alerts pop up, even though only the first case is a match. That's because if a match is found, none of the remaining cases is checked and all the remaining statements in the `switch` block are executed. To stop this process, you can insert a `break` statement, which will end the processing of the `switch` statement.

The corrected code is shown in the example below.

### Code Sample

#### ConditionalsAndLoops/Demos/SwitchWithBreak.html

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>Switch</title>
6.  <script>
7.    var quantity = 1;
8.    switch (quantity) {
9.      case 1 :
10.        alert("quantity is 1");
11.        break;
12.      case 2 :
13.        alert("quantity is 2");
14.        break;
15.      default :
16.        alert("quantity is not 1 or 2");
17.    }
18.  </script>
19.  </head>
20.  <body>
21.    <p>Nothing to show here.</p>
22.  </body>
23.  </html>
```

The following example shows how a `switch/case` statement can be used to record the user's browser type:



## Code Sample

### ConditionalsAndLoops/Demos/BrowserSniffer.html

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>Simple Browser Sniffer</title>
6.  <script>
7.      switch (navigator.appName) {
8.          case "Microsoft Internet Explorer" :
9.              alert("This is IE!");
10.             break;
11.          case "Opera" :
12.              alert("This is Opera!");
13.              break;
14.          default :
15.              alert("This is something other than IE or Opera!");
16.      }
17.  </script>
18.  </head>
19.  <body>
20.      <p>Your browser's <code>navigator.appName</code> is:
21.      <script>
22.          document.write(navigator.appName);
23.      </script>.
24.      </p>
25.  </body>
26.  </html>
```

## Code Explanation

The `navigator` object, which is a child of `window`, holds information about the user's browser. In this case we are looking at the `appName` property, which has a value of "Microsoft Internet Explorer" for Internet Explorer and "Netscape" for Mozilla-based browsers (e.g., Firefox, Chrome).

In conditional statements it's generally a good practice to test for the most likely cases/matches first so the browser can find the correct code to execute more quickly.

## Exercise 11 Conditional Processing

*20 to 30 minutes*

In this exercise, you will practice using conditional processing.

1. Open [ConditionalsAndLoops/Exercises/Conditionals.html](#) for editing.
2. Notice that there is an `onload` event handler that calls the `greetUser()` function. Create this function in the `script` block.
3. The function should do the following:
  - A. Prompt the user for his/her gender and last name and store the results in variables.
  - B. If the user enters a gender other than "Male" or "Female", prompt him/her to try again.
  - C. If the user leaves the last name blank, prompt him/her to try again.
  - D. If the user enters a number for the last name, tell him/her that a last name can't be a number and prompt him/her to try again.
  - E. After collecting the gender and last name:
    - If the gender is valid, pop up an alert that greets the user appropriately (e.g., "Hello Ms. Smith!")
    - If the gender is not valid, pop up an alert that reads something like "XYZ is not a gender!"
4. Test your solution in a browser.

### **\*Challenge**

1. Allow the user to enter his/her gender in any case.
2. If the user enters a last name that does not start with a capital letter, prompt him/her to try again.



### Exercise Solution

#### ConditionalsAndLoops/Solutions/Conditionals.html

```
-----Lines 1 through 5 Omitted-----
6.  <script>
7.  function greetUser(){
8.      var gender, lastName;
9.
10.     gender = prompt("Are you Male or Female?", "") || "";
11.     if (gender != "Male" && gender != "Female") {
12.         gender = prompt("Try again: Male or Female?", "") || "";
13.     }
14.
15.     lastName = prompt("What's your last name?", "") || "";
16.     if (lastName.length === 0) {
17.         lastName = prompt("You must have a last name. Please re-enter:", "")
18.             >>> || "";
19.     } else if (!isNaN(lastName)) {
20.         lastName = prompt("A last name can't be a number. Please re-enter:",
21.             >>> "") || "";
22.     }
23.
24.     switch (gender) {
25.     case "Male" :
26.         alert("Hello Mr. " + lastName + "!");
27.         break;
28.     case "Female" :
29.         alert("Hello Ms. " + lastName + "!");
30.         break;
31.     default :
32.         alert(gender + " is not a gender!");
33.     }
34. }
35. </script>
-----Lines 34 through 38 Omitted-----
```

**Challenge Solution****ConditionalsAndLoops/Solutions/Conditionals-challenge.html**

```

-----Lines 1 through 5 Omitted-----
6.  <script>
7.  function greetUser(){
8.      var gender, lastName;
9.
10.     gender = prompt("Are you Male or Female?", "") || "";
11.     if (gender.toLowerCase() != "male" && gender.toLowerCase() != "female")
        >>> {
12.         gender = prompt("Try again: Male or Female?", "") || "";
13.     }
14.
15.     lastName = prompt("What's your last name?", "");
16.     if (lastName.length === 0) {
17.         lastName = prompt("You must have a last name. Please re-enter:",
            >>> "");
18.     } else if (!isNaN(lastName)) {
19.         lastName = prompt("A last name can't be a number. Please re-enter:",
            >>> "");
20.     } else if (lastName.substr(0, 1) == lastName.substr(0, 1).toLowerCase())
        >>> {
21.         lastName = prompt("The first letter must be capitalized. Re-enter:",
            >>> "") || "";
22.     }
23.
24.     switch (gender.toLowerCase()) {
25.     case "male" :
26.         alert("Hello Mr. " + lastName + "!");
27.         break;
28.     case "female" :
29.         alert("Hello Ms. " + lastName + "!");
30.         break;
31.     default :
32.         alert(gender + " is not a gender!");
33.     }
34. }
35. </script>
-----Lines 36 through 40 Omitted-----

```

## 6.2 Loops

There are several types of loops in JavaScript.

- `while`
- `do...while`
- `for`
- `for...in`
- `for...of`

### **while Loop Syntax**

**Syntax**

```
while (conditions) {  
    statements;  
}
```

Something, usually a statement within the `while` block, must cause the condition to change so that it eventually becomes false and causes the loop to end. Otherwise, you get stuck in an infinite loop, which can bring down the browser.

### **do...while Loop Syntax**

**Syntax**

```
do {  
    statements;  
} while (conditions);
```

Again something, usually a statement within the `do` block, must cause the condition to change so that it eventually becomes false and causes the loop to end.

Unlike with `while` loops, the statements in `do...while` loops will always execute at least one time because the conditions are not checked until the end of each iteration.

## for Loop Syntax

**Syntax**

```
for (initialization; conditions; change) {  
    statements;  
}
```

In for loops, the initialization, conditions, and change are all placed up front and separated by semi-colons. This makes it easy to remember to include a change statement that will eventually cause the loop to end.

for loops are often used to iterate through arrays. The `length` property of an array can be used to check how many elements the array contains.

## for...in Loop Syntax

**Syntax**

```
for (var item in array) {  
    statements;  
}
```

`for...in` loops are specifically designed for looping through arrays. For reasons that will be better understood when we look at object augmentation, the above syntax has a slight flaw. If the `Array` class is changed, it is possible that the `for...in` loop includes more items than what you anticipated.

To be on the safe side, we suggest that you use a more verbose syntax as seen below.

**Syntax**

```
for (var item in array) if (array.hasOwnProperty(item)) {  
    statements;  
}
```

The `hasOwnProperty()` call will ensure that the item is indeed an element that you added to the array, not something that was inherited because of object augmentation.

## for...of Loop Syntax

### Syntax

```
for (variable of iterable) {  
  statement;  
}
```

`for...of` loops are a recent addition to JavaScript, allowing us to loop through any *iterable object* - objects of classes like `Array`, `Map`, `Set`, and other similar classes - rather than just arrays. (An iterable object is an object of a class that is enumerable, where the elements are in a definite order.) As `for...of` loops aren't supported by all browsers, you might avoid this loop type for now - but a good construct for you to recognize should you see it in the code.



**Code Sample****ConditionalsAndLoops/Demos/Loops.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>JavaScript Loops</title>
6.    <script>
7.        var index;
8.        var beatles = [];
9.        beatles["Guitar1"] = "John";
10.       beatles["Bass"] = "Paul";
11.       beatles["Guitar2"] = "George";
12.       beatles["Drums"] = "Ringo";
13.       var ramones = ["Joey", "Johnny", "Dee Dee", "Mark"];
14.    </script>
15.    </head>
16.    <body>
17.    <h1>JavaScript Loops</h1>
18.    <h2>while Loop</h2>
19.    <script>
20.        index = 0;
21.        while (index < 5) {
22.            document.write(index);
23.            index++;
24.        }
25.    </script>
26.
27.    <h2>do...while Loop</h2>
28.    <script>
29.        index = 1;
30.        do {
31.            document.write(index);
32.            index++;
33.        } while (index < 5);
34.    </script>
35.
36.    <h2>for Loop</h2>
37.    <script>
38.        for (index=0; index<=5; index++) {
39.            document.write(index);
```

---

## Conditionals and Loops

```
40.  }
41.  </script>
42.
43.  <h2>for Loop with Arrays</h2>
44.  <ol>
45.    <script>
46.      var len = ramones.length;
47.      for (var index=0; index<len; index++) {
48.        document.write("<li>" + ramones[index] + "</li>");
49.      }
50.    </script>
51.  </ol>
52.
53.  <h2>for...in Loop</h2>
54.  <ol>
55.    <script>
56.      for (var prop in beatles) {
57.        document.write("<li>" + prop + " : " + beatles[prop] + "</li>");
58.      }
59.    </script>
60.  </ol>
61.
62.  <h2>for...of Loop</h2>
63.  <ol>
64.    <script>
65.      for (var bandMember of ramones) {
66.        document.write("<li>" + bandMember + "</li>");
67.      }
68.    </script>
69.  </ol>
70.  </body>
71.  </html>
```

### Code Explanation

The sample above shows demos of all the aforementioned loops.

## Exercise 12 Working with Loops

20 to 30 minutes

In this exercise, you will practice working with loops.

1. Open [ConditionalsAndLoops/Exercises/Loops.html](#) for editing. You will see that this file is similar to the solution from the last exercise.
2. Declare an additional variable called `greeting`.
3. Create an array called `presidents` that contains the last names of four or more past presidents.
4. Currently, the user only gets two tries to enter a valid `gender` and `lastName`. Modify the code so that, in both cases, the user continues to get prompted until he enters valid data.
5. Change the `switch` block so that it assigns an appropriate value (e.g., "Hello Ms. Smith") to the `greeting` variable rather than popping up an alert.
6. After the `switch` block, write code that alerts the user by name if he has the same last name as a president. There is no need to alert those people who have non-presidential names.

### \*Challenge

1. Modify the code so that the first prompt for gender reads "What gender are you: Male or Female?", but all subsequent prompts for gender read "You must enter 'Male' or 'Female'. Try again:".
2. Modify the code so that the first prompt for last name reads "Enter your last name:", but all subsequent prompts for last name read "Please enter a valid last name:".
3. If the user presses the *Cancel* button on a prompt dialog, it returns `null`. Test this. It very likely results in a JavaScript error. If so, fix the code so that no JavaScript error occurs.
4. For those people who do not have presidential names, pop up an alert that tells them their names are not presidential.

### Exercise Solution

#### ConditionalsAndLoops/Solutions/Loops.html

```
-----Lines 1 through 5 Omitted-----
6.  <script>
7.  function greetUser(){
8.    var gender, lastName, greeting;
9.    var presidents = ["Washington", "Jefferson", "Lincoln", "Kennedy"];
10.
11.    do {
12.      gender = prompt("What gender are you: Male or Female?", "") || "";
13.    } while (gender.toLowerCase() != "male" && gender.toLowerCase() !=
        >>> "female");
14.
15.    do {
16.      lastName = prompt("Enter your last name:", "") || "";
17.    } while (lastName.length == 0 || ! isNaN(lastName));
18.
19.    switch (gender.toLowerCase()) {
20.    case "male" :
21.      greeting = "Hello Mr. " + lastName + "!";
22.      break;
23.    case "female" :
24.      greeting = "Hello Ms. " + lastName + "!";
25.      break;
26.    default :
27.      greeting = "Hello Stranger!";
28.    }
29.
30.    for (var pres in presidents) {
31.      if (lastName.toLowerCase() == presidents[pres].toLowerCase()) {
32.        alert(greeting + " You have the same last name as a president!");
33.      }
34.    }
35.  }
36.  </script>
-----Lines 37 through 41 Omitted-----
```

**Challenge Solution****ConditionalsAndLoops/Solutions/Loops-challenge.html**

```

-----Lines 1 through 5 Omitted-----
6.  <script>
7.  function greetUser(){
8.      var gender, lastName, greeting;
9.      var presidents = ["Washington", "Jefferson", "Lincoln", "Kennedy"];
10.
11.      var repeat = false;
12.
13.      do {
14.          if (!repeat) {
15.              gender = prompt("What gender are you: Male or Female?", "") || "";
16.              repeat = true;
17.          } else {
18.              gender = prompt("You must enter 'Male' or 'Female'. Try again:",
19.                  >>> "") || "";
20.          }
21.      } while (typeof gender != "string"
22.          || (gender.toLowerCase() != "male"
23.          && gender.toLowerCase() != "female"));
24.      repeat = false;
25.
26.      do {
27.          if (!repeat) {
28.              lastName = prompt("Enter your last name:", "") || "";
29.              repeat=true;
30.          } else {
31.              lastName = prompt("Please enter a valid last name:", "") || "";
32.          }
33.      } while (typeof lastName != "string"
34.          || (lastName.length === 0
35.          || ! isNaN(lastName)));
36.
37.      switch (gender.toLowerCase()) {
38.          case "male" :
39.              greeting = "Hello Mr. " + lastName + "!";
40.              break;
41.          case "female" :
42.              greeting = "Hello Ms. " + lastName + "!";

```

---

## Conditionals and Loops

```
43.     break;
44.     default :
45.         greeting = "Hello Stranger!";
46.     }
47.
48.     var match = false;
49.
50.     for (var pres in presidents) if (presidents.hasOwnProperty(pres)) {
51.         if (lastName.toLowerCase() == presidents[pres].toLowerCase()) {
52.             alert(greeting + " You have the same last name as a president!");
53.             match = true;
54.         }
55.     }
56.
57.     if (!match) {
58.         alert(greeting + " Your last name is not presidential.");
59.     }
60. }
61. </script>
-----Lines 62 through 66 Omitted-----
```

## **6.3 Conclusion**

In this lesson, you learned to work with JavaScript `if-else` `if-else` and `switch/case` conditionals and several types of loops.





## 7. JavaScript Form Validation

### In this lesson, you will learn...

1. To access data entered by users in forms.
2. To validate text fields and passwords.
3. To validate radio buttons.
4. To validate check boxes.
5. To validate select menus.
6. To validate textareas.
7. To write clean, reusable validation functions.
8. To catch focus, blur, and change events.

### 7.1 Accessing Form Data

All forms on a web page are stored in the `document.forms[ ]` array. The first form on a page is `document.forms[0]`, the second form is `document.forms[1]`, and so on. However, it is usually easier to reference forms via their `id` attribute and refer to them that way - especially as we may add, remove, or move around form elements. For example, a form with `id LoginForm` can be referenced as `document.getElementById( 'LoginForm' )`. The major advantage of referencing forms by `id` is that the forms can be repositioned on the page without affecting the JavaScript.

Elements within a form are properties of that form and can be referenced as follows:

#### Syntax

```
document.formName.elementName
```

But we can access form elements by their `ids`, too, just as we can forms - it's concise, simple, and clear.

Text fields and passwords have a `value` property that holds the text value of the field. The following example shows how JavaScript can access user-entered text:

**Code Sample****FormValidation/Demos/FormFields.html**

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>Form Fields</title>
6.  <script>
7.  function changeBg(){
8.      var userName = document.getElementById("userName").value;
9.      var backgroundClr = document.getElementById("color").value;
10.
11.      document.body.style.backgroundColor = backgroundClr;
12.      alert(userName + ", the background color is " + backgroundClr + ".");
13.      >>>
14.  }
15.
16.  window.onload = function() {
17.      document.getElementById("changebutton").addEventListener("click",
18.          >>> function(){
19.              changeBg();
20.          });
21.  }
22. </script>
23. </head>
24. <body>
25. <h1>Change Background Color</h1>
26. <form name="colorForm">
27.     Your Name: <input type="text" name="userName" id="userName"><br>
28.     Background Color: <input type="text" name="color" id="color"><br>
29.     <input type="button" value="Change Background" id="changebutton">
30. </form>
31. </body>
32. </html>
```

**Code Explanation**

Some things to notice:

1. When the user clicks on the "Change Background" button, the changeBg ( ) function is called. Note how we added a click handler via the addEventListener ( ) method: whereas earlier in the course we used the

`onclick` attribute on the button itself to call a JavaScript function, we now use `addEventListener()` to call the function. Separating the event handler from the element makes for cleaner, more-easily-maintainable code. Similarly, we call the `onload` method using `window.onload()`, rather than (as we had done previously) adding an attribute to the `body` tag.

2. The values entered into the `userName` and `color` fields are stored in variables (`userName` and `bgColor`).
3. We reference the two fields directly via their `ids`.

## Exercise 13 Textfield to Textfield

*15 to 25 minutes*

In this exercise, you will write a function that bases the value of one text field on the value of another.

1. Open [FormValidation/Exercises/TextfieldToTextField.html](#) for editing.
2. Write a function called `getMonth( )` that passes the month number entered by the user to the `monthAsString( )` function in [DateUDFs.js](#) and writes the result in the `monthName` field.

### Exercise Code

#### FormValidation/Exercises/TextfieldToTextField.html

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>Textfield to Textfield</title>
6.  <script src="DateUDFs.js"></script>
7.  <script>
8.  /*
9.      Write a function called getMonth() that passes the
10.     month number entered by the user to the monthAsString()
11.     function in DateUDFs.js and writes the result in
12.     the MonthName field.
13.  */
14. </script>
15. </head>
16. <body>
17. <h1>Month Check</h1>
18. <form name="dateForm">
19.     Month Number: <input type="text" name="MonthNumber" id="MonthNumber"
20.         >>> size="2">
21.     <input type="button" value="Get Month" id="getmonthbutton"><br>
22.     Month Name: <input type="text" name="MonthName" id="MonthName"
23.         >>> size="10">
24. </form>
25. </body>
26. </html>
```

**\*Challenge**

1. If the user enters a number less than 1 or greater than 12 or a non-number, have the function write "Bad Number" in the `MonthName` field.
2. If the user enters a decimal between 1 and 12 (inclusive), strip the decimal portion of the number.

**Exercise Solution****FormValidation/Solutions/TextfieldToTextField.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Textfield to Textfield</title>
6.    <script src="DateUDFs.js"></script>
7.    <script>
8.    function getMonth(){
9.        var elemMonthNumber = document.getElementById("monthNumber");
10.       var elemMonthName = document.getElementById("monthName");
11.       var month = monthAsString(elemMonthNumber.value);
12.
13.       elemMonthName.value = month;
14.   }
15.
16.   window.onload = function() {
17.       document.getElementById("getMonthButton").addEventListener("click",
18.           >>> function(){
19.               getMonth();
20.           });
21.   }
22. </script>
23. </head>
24. <body>
25. <h1>Month Check</h1>
26. <form name="dateForm">
27.     Month Number: <input type="text" name="monthNumber" id="monthNumber"
28.         >>> size="2">
29.     <input type="button" value="Get Month" id="getMonthButton"><br>
30.     Month Name: <input type="text" name="monthName" id="monthName"
31.         >>> size="10">
32. </form>
33. </body>
34. </html>
```

**Challenge Solution****FormValidation/Solutions/TextfieldToTextField-challenge.html**

```

1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Textfield to Textfield</title>
6.    <script src="DateUDFs.js"></script>
7.    <script>
8.        function getMonth(){
9.            var elemMonthNumber = document.getElementById("monthNumber");
10.           var elemMonthName = document.getElementById("monthName");
11.           var monthNum = parseInt(elemMonthNumber.value);
12.
13.           var month = (monthNum >= 1 && monthNum <= 12) ? monthAsString(monthNum)
14.               >>> : "Bad Number";
15.           elemMonthName.value = month;
16.       }
17.
18.       window.onload = function() {
19.           document.getElementById("getMonthButton").addEventListener("click",
20.               >>> function(){
21.                   getMonth();
22.               });
23.       }
24.   </script>
25.   </head>
26.   <body>
27.   <h1>Month Check</h1>
28.   <form name="dateForm">
29.       Month Number: <input type="text" name="monthNumber" id="monthNumber"
30.           >>> size="2">
31.       <input type="button" value="Get Month" id="getMonthButton"><br>
32.       Month Name: <input type="text" name="monthName" id="monthName"
33.           >>> size="10">
34.   </form>
35.   </body>
36.   </html>

```

**Code Explanation**

Notice the use of the ternary operator.

## 7.2 Basics of Form Validation

When the user clicks on a *submit* button or presses the return/enter key while in a form field, an event occurs that can be caught with the form's `onsubmit` event handler. Unless JavaScript is used to explicitly cancel the submit event, the form will be submitted. The `return false;` statement explicitly cancels the submit event. For example, the following form will never be submitted:

```
<form action="Process.html" onsubmit="return false;">
  <!--Code for form fields would go here-->
  <input type="submit" value="Submit Form">
</form>
```

Note that, for brevity, we here use the `onsubmit` attribute to listen for the submit event; we'll generally listen for the submit event (and other events) with `addEventListener()` to keep our JavaScript separate from our HTML markup.

Of course, when validating a form, we only want the form *not* to submit if something is wrong. We can use the `preventDefault()` method to stop the form from submitting if the data entered by the user doesn't match what we expect. Let's look at an example

---

8. HTML is not a programming language and cannot normally be used to process a form. We use it here only as a placeholder page.



**Code Sample****FormValidation/Demos/Login.html**

```

1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Login</title>
6.    <script>
7.    function validate(e){
8.        var userName = document.getElementById("Username").value;
9.        var password = document.getElementById("Password").value;
10.
11.        if (userName.length === 0) {
12.            alert("You must enter a username.");
13.            e.preventDefault();
14.            return;
15.        }
16.
17.        if (password.length === 0) {
18.            alert("You must enter a password.");
19.            e.preventDefault();
20.            return;
21.        }
22.    }
23.
24.    window.onload = function() {
25.        document.getElementById("logInForm").addEventListener("submit", func >>>
26.            >>> tion(e){
27.                validate(e);
28.            });
29.    }
30.    </script>
31.    </head>
32.    <body>
33.    <h1>Login Form</h1>
34.    <form method="post" action="Process.html" id="logInForm">
35.        Username: <input type="text" name="Username" id="Username"
36.            >>> size="10"><br>
37.        Password: <input type="password" name="password" id="Password"
38.            >>> size="10"><br>

```

```
38.     <input type="submit" value="Submit">
39.     <input type="reset" value="Reset Form">
40.     </form>
41.     </body>
42.     </html>
```

### Code Explanation

1. When the user submits the form, the `addEventListener` event handler captures the `submit` event and calls the `validate()` function, passing in the event object (`"e"`).
2. The values entered into the Username and Password fields are stored in variables (`userName` and `password`).
3. An `if` condition is used to check if `userName` is an empty string. If it is, an alert pops up explaining the problem and we use `preventDefault()` to prevent the form from submitting. We use `return` to halt the function - the function does not check for any additional validation errors.
4. An `if` condition is used to check that `password` is an empty string. If it is, an alert pops up explaining the problem and we use `preventDefault()` to prevent the form from submitting. We use `return` to halt the function - the function does not check for any additional validation errors.
5. If neither `if` condition catches a problem, then there is no call to `preventDefault()` and the form submits.

### Cleaner Validation

There are a few improvements we can make on the last example.

One problem is that the `validation()` function only checks for one problem at a time. That is, if it finds an error, it reports it immediately and does not check for additional errors. Why not just tell the user all the mistakes that need to be corrected, so (s)he doesn't have to keep submitting the form to find each subsequent error?

Another problem is that the code is not written in a way that makes it easily reusable. For example, checking for the length of user-entered values is a common thing to do, so it would be nice to have a ready-made function to handle this.

These improvements are made in the example below.

## Code Sample

### FormValidation/Demos/Login2.html

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>Login</title>
6.  <script>
7.  function validate(e){
8.      var userName = document.getElementById("Username").value;
9.      var password = document.getElementById("Password").value;
10.     var errors = [];
11.
12.     if (!checkLength(userName,1,100)) {
13.         errors[errors.length] = "You must enter a username.";
14.     }
15.
16.     if (!checkLength(password,1,100)) {
17.         errors[errors.length] = "You must enter a password.";
18.     }
19.
20.     if (errors.length > 0) {
21.         reportErrors(errors);
22.         e.preventDefault();
23.     }
24. }
25.
26. function checkLength(text, min, max){
27.
28.     if (text.length < min || text.length > max) {
29.         return false;
30.     }
31.     return true;
32. }
33.
34. function reportErrors(errors){
35.     var msg = "There were some problems...\n";
36.     var numError;
37.     for (var i=0; i<errors.length; i++) {
38.         numError = i + 1;
39.         msg += "\n" + numError + ". " + errors[i];
```

```
40.     }
41.     alert(msg);
42. }
43.
44. window.onload = function() {
45.     document.getElementById("logInForm").addEventListener("submit", func >>
        >>> tion(e){
46.         validate(e);
47.     });
48. }
49. </script>
50. </head>
51. <body>
52. <h1>Login Form</h1>
53. <form method="post" action="Process.html" id="logInForm">
54.
55.     Username: <input type="text" name="Username" id="Username"
        >>> size="10"><br>
56.     Password: <input type="password" name="Password" id="Password"
        >>> size="10"><br>
57.
58.     <input type="submit" value="Submit">
59.     <input type="reset" value="Reset Form">
60. </form>
61. </body>
62. </html>
```

### Code Explanation

Some things to notice:

1. Two additional functions are created: `checkLength()` and `reportErrors()`.
  - The `checkLength()` function takes three arguments, the text to examine, the required minimum length, and the required maximum length.
  - The `reportErrors()` function takes one argument, an array holding the errors. It loops through the errors array creating an error message and then it pops up an alert with a message. The `\n` is an escape character for a newline.
2. In the main `validate()` function, a new array, `errors`, is created to hold any errors that are found.

3. `userName` and `password` are passed to `checkLength()` for validation.
  - If errors are found, they are appended to `errors`.
4. If there are any errors in `errors` (i.e., if its `length` is greater than zero), then `errors` is passed to `reportErrors()`, which pops up an alert letting the user know where the errors are. The `validate()` function then uses `e.preventDefault()` to prevent the form from submitting.

By modularizing the code in this way, it makes it easy to add new validation functions. In the next examples we will move the reusable validation functions into a separate JavaScript file called `FormValidation.js`.

## Exercise 14 Validating a Registration Form

*25 to 40 minutes*

In this exercise, you will write code to validate a registration form. You may find it useful to use the Common String Methods table (see page 92) as a reference.

1. Open [FormValidation/Exercises/FormValidation.js](#) for editing.
  - Create a function called `compareValues()` that takes two arguments: `val1` and `val2`. The function should return:
    - 0 if the two values are equal
    - -1 if `val1` is greater than `val2`
    - 1 if `val2` is greater than `val1`
  - Create a function called `checkEmail()` that takes one argument: `email`. The function should return:
    - `false` if `email` has fewer than 6 characters
    - `false` if `email` does not contain an @ symbol
    - `false` if `email` does not contain a period (.)
    - `true` otherwise
2. Open [FormValidation/Exercises/Register.html](#) for editing.
  - Add code so that the functions in [FormValidation.js](#) are accessible from this page.
  - Create a `validate()` function that does the following:
    - Checks that the `FirstName`, `LastName`, `City`, `Country`, `UserName`, and `Password1` fields are filled out.
    - Checks that the middle initial is a single character.
    - Checks that the state is exactly two characters.
    - Checks that the email is a valid email address.
    - Checks that the values entered into `Password1` and `Password2` are the same.
    - If there are errors, passes `reportErrors()` the `errors` array and use `e.preventDefault()` to prevent the form from submitting.
3. Test your solution in a browser.

In [FormValidation/Exercises/FormValidation.js](#), modify the `checkEmail()` function so that it also checks to see that the final period (.) is after the final @ symbol. The solution is included in [FormValidation/Solutions/FormValidation.js](#).



**Exercise Solution****FormValidation/Solutions/FormValidation.js**

```
1.  /*
2.   Function Name: checkLength
3.   Arguments: text,min?,max?
4.   Returns:
5.     false if text has fewer than min characters
6.     false if text has more than max characters
7.     true otherwise
8.  */
9.  function checkLength(text, min, max){
10.
11.    if (text.length < min || text.length > max) {
12.      return false;
13.    }
14.    return true;
15.  }
16.  /*
17.   Function Name: compareValues
18.   Arguments: val1, val2
19.   Returns:
20.     0 if two values are equal
21.     -1 if val1 is greater than val2
22.     1 if val2 is greater than val1
23.  */
24.  function compareValues(val1, val2){
25.    if (val1 > val2) {
26.      return -1;
27.    } else if(val2 > val1) {
28.      return 1;
29.    } else {
30.      return 0;
31.    }
32.  }
33.
34.  /*
35.   Function Name: checkEmail
36.   Arguments: email
37.   Returns:
38.     false if email has fewer than 6 characters
39.     false if email does not contain @ symbol
```



```
40.     false if email does not contain a period (.)
41.     true otherwise
42. */
43. function checkEmail(email){
44.     if (!checkLength(email, 6)) {
45.         return false;
46.     } else if (email.indexOf("@") == -1) {
47.         return false;
48.     } else if (email.indexOf(".") == -1) {
49.         return false;
50.     }
51.     /* THIS LAST ELSE IF FROM CHALLENGE */
52.     else if (email.lastIndexOf(".") < email.lastIndexOf("@")) {
53.         return false;
54.     }
55.     return true;}
```

**Exercise Solution****FormValidation/Solutions/Register.html**

```
-----Lines 1 through 5 Omitted-----
6.  <script src="FormValidation.js"></script>
7.  <script>
8.  function validate(e){
9.      var firstName = document.getElementById("FirstName").value;
10.     var midInit = document.getElementById("MidInit").value;
11.     var lastName = document.getElementById("LastName").value;
12.     var city = document.getElementById("City").value;
13.     var state = document.getElementById("State").value;
14.     var country = document.getElementById("Country").value;
15.     var zipCode = document.getElementById("Zip").value;
16.     var email = document.getElementById("Email").value;
17.     var userName = document.getElementById("Username").value;
18.     var password1 = document.getElementById("Password1").value;
19.     var password2 = document.getElementById("Password2").value;
20.     var errors = [];
21.
22.     if (!checkLength(firstName,1,100)) {
23.         errors[errors.length] = "You must enter a first name.";
24.     }
25.
26.     if (!checkLength(midInit, 1, 1)) {
27.         errors[errors.length] = "You must enter a one-letter middle initial.";
28.         >>>
29.     }
30.     if (!checkLength(lastName,1,100)) {
31.         errors[errors.length] = "You must enter a last name.";
32.     }
33.
34.     if (!checkLength(city,1,100)) {
35.         errors[errors.length] = "You must enter a city.";
36.     }
37.
38.     if (!checkLength(state, 2, 2)) {
39.         errors[errors.length] = "You must enter a state abbreviation.";
40.     }
41.
42.     if (!checkLength(country,1,100)) {
```

```

43.     errors[errors.length] = "You must enter a country.";
44. }
45.
46. if (!checkLength(zipCode, 5, 10)) {
47.     errors[errors.length] = "You must enter a valid zip code.";
48. }
49. if (!checkEmail(email)) {
50.     errors[errors.length] = "You must enter a valid email address.";
51. }
52.
53. if (!checkLength(userName,1,100)) {
54.     errors[errors.length] = "You must enter a username.";
55. }
56.
57. if (!checkLength(password1,1,100)) {
58.     errors[errors.length] = "You must enter a password.";
59. } else if (compareValues(password1, password2) !== 0) {
60.     errors[errors.length] = "Passwords don't match.";
61. }
62.
63. if (errors.length > 0) {
64.     reportErrors(errors);
65.     e.preventDefault();
66. }
67. }function reportErrors(errors){
68.     var msg = "There were some problems...\n";
69.     var numError;
70.     for (var i=0; i<errors.length; i++) {
71.         numError = i + 1;
72.         msg += "\n" + numError + ". " + errors[i];
73.     }
74.     alert(msg);
75. }
76.
77. window.onload = function() {
78.     document.getElementById("registrationform").addEventListener("submit",
79.         >>> function(e){
80.             validate(e);
81.         });
82. }
</script></head>
-----Lines 83 through 140 Omitted-----

```

## **7.3 Validating Radio Buttons**

Radio buttons that have the same name are grouped as arrays. Generally, the goal in validating a radio button array is to make sure that the user has checked one of the options. Individual radio buttons have the `checked` property, which is `true` if the button is checked and `false` if it is not. The example below shows a simple function for checking radio button arrays.

## Code Sample

### FormValidation/Demos/RadioArrays.html

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>Radio Arrays</title>
6.  <script>
7.  function validate(e){
8.      var errors = [];
9.      var f = e.currentTarget;
10.
11.     if (!checkRadioArray(f.container) ) {
12.         errors[errors.length] = "You must choose a cup or cone.";
13.     }
14.
15.     if (errors.length > 0) {
16.         reportErrors(errors);
17.         e.preventDefault();
18.     }
19. }
20.
21. function checkRadioArray(radioButton){
22.     for (var i=0; i < radioButton.length; i++) {
23.         if (radioButton[i].checked) {
24.             return true;
25.         }
26.     }
27.     return false;
28. }
29.
30. function reportErrors(errors){
31.     var msg = "There were some problems...\n";
32.     var numError;
33.     for (var i=0; i<errors.length; i++) {
34.         numError = i + 1;
35.         msg += "\n" + numError + ". " + errors[i];
36.     }
37.     alert(msg);
38. }
39.
```

---

## JavaScript Form Validation

```
40. window.onload = function() {
41.     document.getElementById("IceCreamForm").addEventListener("submit",
42.         >>> function(e){
43.             validate(e);
44.         });
45.     }
46. </script>
47. </head>
48. <body>
49. <h1>Ice Cream Form</h1>
50. <form method="post" action="Process.html" id="IceCreamForm">
51.     <strong>Cup or Cone?</strong>
52.     <input type="radio" name="container" id="Container_Cup" value="cup">
53.         >>> Cup
54.     <input type="radio" name="container" id="Container_Plaincone" value="plainCone">
55.         >>> Plain cone
56.     <input type="radio" name="container" id="Container_Sugarcone" value="sugarCone">
57.         >>> Sugar cone
58.     <input type="radio" name="container" id="Container_Wafflecone" value="waffleCone">
59.         >>> Waffle cone
60.     <br><br>
61.     <input type="submit" value="Place Order">
62. </form>
63. </body>
64. </html>
```

### Code Explanation

The `checkRadioArray()` function takes a radio button array as an argument, loops through each radio button in the array, and returns `true` as soon as it finds one that is checked. Since it is only possible for one option to be checked, there is no reason to continue looking once a checked button has been found. If none of the buttons are checked, the function returns `false`.

Note that we get the form from the event object `e` with `e.currentTarget`, store the form as JavaScript variable `f`, and then get the radio button set by name (`f.container`). While not needed in this example, we adopt the common practice of giving radio buttons unique ids, where each id is the shared name, underscore, then unique value, like `container_cup`. This is useful if we need to refer to a specific radio button.

## 7.4 Validating Check Boxes

Like radio buttons, check boxes have the `checked` property, which is `true` if the button is checked and `false` if it is not. However, unlike radio buttons, check boxes are not stored as arrays. The example below shows a simple function for checking to make sure a check box is checked.

**Code Sample****FormValidation/Demos/CheckBoxes.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Checkboxes</title>
6.    <script>
7.    function validate(e){
8.        var errors = [];
9.
10.     if ( !checkCheckBox(document.getElementById("terms")) ) {
11.         errors[errors.length] = "You must agree to the terms.";
12.     }
13.
14.     if (errors.length > 0) {
15.         reportErrors(errors);
16.         e.preventDefault();
17.     }
18.
19. }function checkCheckBox(cb){
20.     return cb.checked;
21. }
22.
23. function reportErrors(errors){
24.     var msg = "There were some problems...\n";
25.     var numError;
26.     for (var i = 0; i<errors.length; i++) {
27.         numError = i + 1;
28.         msg += "\n" + numError + ". " + errors[i];
29.     }
30.     alert(msg);
31. }
32.
33. window.onload = function() {
34.     document.getElementById("noicecreamform").addEventListener("submit",
35.         >>> function(e){
36.             validate(e);
37.         });
38. }
39. </script>
```



```
39. </head>
40. <body>
41. <h1>Ice Cream Form</h1>
42. <form method="post" action="Process.html" id="noicecreamform">
43.   <input type="checkbox" name="terms" id="terms">
44.   I understand that I'm really not going to get any ice cream.
45.   <br><br>
46.   <input type="submit" value="Place Order">
47. </form>
48.
49. </body>
50. </html>
```

## 7.5 Validating Select Menus

Select menus contain an array of options. The `selectedIndex` property of a select menu contains the index of the option that is selected. Often the first option of a select menu is something meaningless like "Please choose an option..." The `checkSelect()` function in the example below makes sure that the first option is not selected.

**Code Sample****FormValidation/Demos/SelectMenus.html**

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>Select Menus</title>
6.  <script>
7.  function validate(e){
8.      var errors = [];
9.
10.     if (!checkSelect(document.getElementById("flavor"))) {
11.         errors[errors.length] = "You must choose a flavor.";
12.     }
13.
14.     if (errors.length > 0) {
15.         reportErrors(errors);
16.         e.preventDefault();
17.     }
18.
19. }
20.
21. function checkSelect(select){
22.     return (select.selectedIndex > 0);
23. }
24. function reportErrors(errors){
25.     var msg = "There were some problems...\n";
26.     var numError;
27.     for (var i=0; i<errors.length; i++) {
28.         numError = i + 1;
29.         msg += "\n" + numError + ". " + errors[i];
30.     }
31.     alert(msg);
32. }
33.
34. window.onload = function() {
35.     document.getElementById("flavorform").addEventListener("submit",
36.         >>> function(e){
37.             validate(e);
38.         });
39. }
```

```
39. </script>
40. </head>
41. <body>
42. <h1>Ice Cream Form</h1>
43. <form method="post" action="Process.html" id="flavorform">
44.   <strong>Flavor:</strong>
45.   <select name="flavor" id="flavor">
46.     <option value="0" selected></option>
47.     <option value="choc">Chocolate</option>
48.     <option value="straw">Strawberry</option>
49.     <option value="van">Vanilla</option>
50.   </select>
51.   <br><br>
52.   <input type="submit" value="Place Order">
53. </form>
54. </body>
55. </html>
```

## 7.6 Focus, Blur, and Change Events

*Focus*, *blur* and *change* events can be used to improve the user experience.

### Focus and Blur

Focus and blur events are caught with the `onfocus` and `onblur` event handlers. These events have corresponding `focus()` and `blur()` methods. The example below shows

1. how to set focus on a field.
2. how to capture when a user leaves a field.
3. how to prevent focus on a field.

**Code Sample****FormValidation/Demos/FocusAndBlur.html**

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>Focus and Blur</title>
6.  <script src="DateUDFs.js"></script>
7.  <script>
8.  function getMonth(){
9.      var elemMonthNumber = document.DateForm.MonthNumber;
10.     var monthNumber = elemMonthNumber.value;
11.
12.     var elemMonthName = document.DateForm.MonthName;
13.     var month = monthAsString(elemMonthNumber.value);
14.
15.     elemMonthName.value = (monthNumber > 0 && monthNumber <=12) ? month :
        >>> "Bad Number";
16. }
17.
18. window.onload = function() {
19.     document.getElementById("MonthNumber").focus();
20.
21.     document.getElementById("MonthNumber").addEventListener("blur", func >>>
        >>> tion(e){
22.         getMonth();
23.     });
24.
25.     document.getElementById("MonthName").addEventListener("focus", func >>>
        >>> tion(e){
26.         this.blur();
27.     });
28. }
29. </script>
30. </head>
31. <body>
32. <h1>Month Check</h1>
33. <form name="DateForm" id="DateForm">
34.     Month Number:
35.     <input type="text" name="MonthNumber" id="MonthNumber" size="2">
36.     Month Name:
37.     <input type="text" name="MonthName" id="MonthName" size="10">
```

```
38. </form>
39. </body>
40. </html>
```

## Code Explanation

Things to notice:

1. When the document is loaded, we call `focus()` on the `MonthNumber` element, making that field active for the user to start typing.
2. When focus leaves the `MonthNumber` field, we capture the `blur` event and call the `getMonth()` function.
3. The `onfocus` event handler of the `MonthName` element triggers a call to the `blur()` method of `this` (the `MonthName` element itself) to prevent the user from focusing on the `MonthName` element.<sup>9</sup>

## Change

Change events are caught when the value of a text element changes or when the selected index of a select element changes. The example below shows how to capture a change event.

---

9. In Internet Explorer, the cursor may appear in the element, but the user is not able to make any changes to the element's value.

**Code Sample****FormValidation/Demos/Change.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Change</title>
6.    <script src="DateUDFs.js"></script>
7.    <script>
8.    function getMonth(){
9.        var elemMonthNumber = document.DateForm.MonthNumber;
10.       var i = elemMonthNumber.selectedIndex;
11.       var monthNumber = elemMonthNumber[i].value;
12.
13.       var elemMonthName = document.DateForm.MonthName;
14.       var month = monthAsString(monthNumber);
15.
16.       elemMonthName.value = (i === 0) ? "" : month;
17.   }
18.
19.   window.onload = function() {
20.       document.getElementById("MonthNumber").focus();
21.
22.       document.getElementById("MonthNumber").addEventListener("change",
23.           >>> function(e){
24.               getMonth();
25.           });
26.       document.getElementById("MonthName").addEventListener("focus", func >>>
27.           >>> tion(e){
28.               this.blur();
29.           });
30.   }
31. </script>
32. </head>
33. <body>
34. <h1>Month Check</h1>
35. <form name="DateForm">
36.     Month Number:
37.     <select name="MonthNumber" id="MonthNumber">
38.         <option>--Choose--</option>
```

```
38.     <option value="1">1</option>
39.     <option value="2">2</option>
40.     <option value="3">3</option>
41.     <option value="4">4</option>
42.     <option value="5">5</option>
43.     <option value="6">6</option>
44.     <option value="7">7</option>
45.     <option value="8">8</option>
46.     <option value="9">9</option>
47.     <option value="10">10</option>
48.     <option value="11">11</option>
49.     <option value="12">12</option>
50.     </select><br>
51.     Month Name: <input type="text" name="MonthName" id="MonthName"
    >>> size="10">
52. </form>
53. </body>
54. </html>
```

### Code Explanation

This is similar to the last example. The only major difference is that `MonthNumber` is a select menu instead of a text field and that the `getMonth( )` function is called when a different option is selected - that is, when the select menu changes.

## 7.7 Validating Textareas

Textareas can be validated the same way that text fields are by using the `checkLength( )` function shown earlier. However, because textareas generally allow for many more characters, it's often difficult for the user to know if he's exceeded the limit. It could be helpful to let the user know if there's a problem as soon as focus leaves the textarea. The example below, which contains a more complete form for ordering ice cream, includes a function that alerts the user if there are too many characters in a textarea.

**Code Sample****FormValidation/Demos/IceCreamForm.html**

```
-----Lines 1 through 44 Omitted-----
45.  function checkLength(text, min=1, max=10000){
46.    if (text.length < min || text.length > max) {
47.      return false;
48.    }
49.    return true;
50.  }
51.  function checkTextArea(textArea, max){
52.    var numChars, chopped, message;
53.    if (!checkLength(textArea.value, 0, max)) {
54.      numChars = textArea.value.length;
55.      chopped = textArea.value.substr(0, max);
56.      message = 'You typed ' + numChars + ' characters.\n';
57.      message += 'The limit is ' + max + '.';
58.      message += 'Your entry will be shortened to:\n\n' + chopped;
59.      alert(message);
60.      textArea.value = chopped;
61.    }
62.  }
63.
-----Lines 64 through 102 Omitted-----
103.  <p>
104.    <strong>Special Requests:</strong><br>
105.    <textarea name="requests" id="requests" cols="40" rows="6" wrap="vir »»
      >>> tual"></textarea>
106.  </p>
-----Lines 107 through 115 Omitted-----
```



## Exercise 15 Improving the Registration Form

*15 to 25 minutes*

In this exercise, you will make some improvements to the registration form from the last exercise.

1. Open [FormValidation/Exercises/FormValidation2.js](#) in your editor. You will see that the functions discussed above have been added:  
`checkRadioArray()`, `checkCheckBox()`, `checkSelect()`, and `checkTextArea()`.
2. Open [FormValidation/Exercises/Register2.html](#) for editing.
  - Notice that the following changes have been made to the form:
    - `State` has been changed from a textfield to a select menu. The first option is meaningless. The next 51 options are U.S. states. The rest of the options are Canadian provinces.
    - `Country` has been changed to a radio array.
    - A `Comments` field has been added.
    - A `Terms` check box has been added.
  - Write code that:
    - Checks that a country is selected.
    - Checks that the country and state selection are in sync.
    - Checks that the terms have been accepted.
  - Add an event handler for the `Comments` textarea (as you saw in our earlier demo) that alerts the user if the comment is too long.
3. Test your solution in a browser.

**Exercise Solution****FormValidation/Solutions/Register2.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Registration Form</title>
6.    <script src="FormValidation2.js"></script>
7.    <script>
8.        function validate(e){
9.            var f = e.currentTarget;
10.           var firstName = document.getElementById("FirstName").value;
11.           var midInit = document.getElementById("MidInit").value;
12.           var lastName = document.getElementById("LastName").value;
13.           var city = document.getElementById("City").value;
14.           var state = document.getElementById("State").value;
15.           var zipCode = document.getElementById("Zip").value;
16.           var email = document.getElementById("Email").value;
17.           var userName = document.getElementById("Username").value;
18.           var password1 = document.getElementById("Password1").value;
19.           var password2 = document.getElementById("Password2").value;
20.           var errors = [];
21.
22.           if (!checkLength(firstName,1,100)) {
23.               errors[errors.length] = "You must enter a first name.";
24.           }
25.
26.           if (!checkLength(midInit, 1, 1)) {
27.               errors[errors.length] = "You must enter a one-letter middle initial.";
28.               >>>
29.           }
30.           if (!checkLength(lastName,1,100)) {
31.               errors[errors.length] = "You must enter a last name.";
32.           }
33.
34.           if (!checkLength(city,1,100)) {
35.               errors[errors.length] = "You must enter a city.";
36.           }
37.
38.           if (!checkLength(zipCode, 5, 10)) {
```

```

39.     errors[errors.length] = "You must enter a valid zip code.";
40. }
41. if (!checkEmail(email)) {
42.     errors[errors.length] = "You must enter a valid email address.";
43. }
44.
45. if (!checkLength(userName,1,100)) {
46.     errors[errors.length] = "You must enter a username.";
47. }
48.
49. if (!checkLength(password1,1,100)) {
50.     errors[errors.length] = "You must enter a password.";
51. } else if (compareValues(password1, password2) !== 0) {
52.     errors[errors.length] = "Passwords don't match.";
53. }
54.
55. if (!checkRadioArray(f.Country)) {
56.     errors[errors.length] = "You must select a country.";
57. } else if ((!document.getElementById("Country_Other").checked &&
    >>> f.State.selectedIndex === 0)
58. || (document.getElementById("Country_USA").checked && f.State.selecte >>
    >>> dIndex > 51)
59. || (document.getElementById("Country_CA").checked && f.State.selecte >>
    >>> dIndex <= 51)
60. || (document.getElementById("Country_Other").checked && f.State.se >>
    >>> lectedIndex > 0)) {
61.
62.     errors[errors.length] = "Country and State don't match.";
63. }
64.
65. if (!checkCheckBox(document.getElementById("Terms"))) ) {
66.     errors[errors.length] = "You must agree to the terms.";
67. }
68.
69. if (errors.length > 0) {
70.     reportErrors(errors);
71.     e.preventDefault();
72. }
73.
74. }
75.
76. function reportErrors(errors){
77.     var msg = "There were some problems...\n";

```

---

## JavaScript Form Validation

```
78.     var numError;
79.     for (var i=0; i<errors.length; i++) {
80.         numError = i + 1;
81.         msg += "\n" + numError + ". " + errors[i];
82.     }
83.     alert(msg);
84. }
85.
86. window.onload = function() {
87.     document.getElementById("registrationform").addEventListener("submit", function(e){
88.         validate(e);
89.     });
90.
91.     document.getElementById("Comments").addEventListener("blur", function(){
92.         checkTextArea(this, 100);
93.     });
94. }
95. </script>
96. </head>
97. <body>
98. <h1>Registration Form</h1>
99. <form method="post" id="registrationform" action="Process.html">
100. <table border="0" cellspacing="2" cellpadding="2">
101. <tr>
102. <td>First Name:</td>
103. <td><input type="text" name="FirstName" id="FirstName" size="20"></td>
104. </tr>
105. <tr>
106. <td>Middle Initial:</td>
107. <td><input type="text" name="MidInit" id="MidInit" size="20"></td>
108. </tr>
109. <tr>
110. <td>Last Name:</td>
111. <td><input type="text" name="LastName" id="LastName" size="20"></td>
112. </tr>
113. <tr>
114. <td>City:</td>
115. <td><input type="text" name="City" id="City" size="20"></td>
116. </tr>
117. <tr>
```

```
118. <td>State/Province: </td>
119. <td>
120.   <select name="State" id="State">
121.     <option value="0">Please Choose...</option>
122.     <option value="AL">Alabama</option>
123.     <option value="AK">Alaska</option>
124.     <option value="AZ">Arizona</option>
125.     <option value="AR">Arkansas</option>
126.     <option value="CA">California</option>
127.     <option value="CO">Colorado</option>
128.     <option value="CT">Connecticut</option>
129.     <option value="DE">Delaware</option>
130.     <option value="DC">District of Columbia</option>
131.     <option value="FL">Florida</option>
132.     <option value="GA">Georgia</option>
133.     <option value="HI">Hawaii</option>
134.     <option value="ID">Idaho</option>
135.     <option value="IL">Illinois</option>
136.     <option value="IN">Indiana</option>
137.     <option value="IA">Iowa</option>
138.     <option value="KS">Kansas</option>
139.     <option value="KY">Kentucky</option>
140.     <option value="LA">Louisiana</option>
141.     <option value="ME">Maine</option>
142.     <option value="MD">Maryland</option>
143.     <option value="MA">Massachusetts</option>
144.     <option value="MI">Michigan</option>
145.     <option value="MN">Minnesota</option>
146.     <option value="MS">Mississippi</option>
147.     <option value="MO">Missouri</option>
148.     <option value="MT">Montana</option>
149.     <option value="NE">Nebraska</option>
150.     <option value="NV">Nevada</option>
151.     <option value="NH">New Hampshire</option>
152.     <option value="NJ">New Jersey</option>
153.     <option value="NM">New Mexico</option>
154.     <option value="NY">New York</option>
155.     <option value="NC">North Carolina</option>
156.     <option value="ND">North Dakota</option>
157.     <option value="OH">Ohio</option>
158.     <option value="OK">Oklahoma</option>
159.     <option value="OR">Oregon</option>
```

```
160.     <option value="PA">Pennsylvania</option>
161.     <option value="RI">Rhode Island</option>
162.     <option value="SC">South Carolina</option>
163.     <option value="SD">South Dakota</option>
164.     <option value="TN">Tennessee</option>
165.     <option value="TX">Texas</option>
166.     <option value="UT">Utah</option>
167.     <option value="VT">Vermont</option>
168.     <option value="VA">Virginia</option>
169.     <option value="WA">Washington</option>
170.     <option value="WV">West Virginia</option>
171.     <option value="WI">Wisconsin</option>
172.     <option value="WY">Wyoming</option>
173.     <option value="AB">Alberta</option>
174.     <option value="BC">British Columbia</option>
175.     <option value="MB">Manitoba</option>
176.     <option value="NB">New Brunswick</option>
177.     <option value="NF">Newfoundland</option>
178.     <option value="NS">Nova Scotia</option>
179.     <option value="ON">Ontario</option>
180.     <option value="PE">Prince Edward Island</option>
181.     <option value="QC">Quebec</option>
182.     <option value="SK">Saskatchewan</option>
183. </select>
184. </td>
185. </tr>
186. <tr valign="top">
187.   <td>Country:</td>
188.   <td>
189.     <input type="radio" name="Country" id="Country_USA" value="USA">
190.       >>> United States<br>
191.     <input type="radio" name="Country" id="Country_CA" value="CA"> Cana >>
192.       >>> da<br>
193.     <input type="radio" name="Country" id="Country_Other" value="Other">
194.       >>> Other
195.   </td>
196. </tr>
197. <tr>
198.   <td>Zip: </td>
199.   <td><input type="text" name="Zip" id="Zip" size="10"></td>
200. </tr>
201. <tr>
202.   <td>Email: </td>
```

```
200. <td><input type="text" name="Email" id="Email" size="30"></td>
201. </tr>
202. <tr>
203. <td>Username:</td>
204. <td><input type="text" name="Username" id="Username" size="10"></td>
205. </tr>
206. <tr>
207. <td>Password:</td>
208. <td><input type="password" name="Password1" id="Password1"
    >>> size="10"></td>
209. </tr>
210. <tr>
211. <td>Repeat Password: </td>
212. <td><input type="password" name="Password2" id="Password2"
    >>> size="10"></td>
213. </tr>
214. <tr valign="top">
215. <td>Comments: </td>
216. <td>
217. <textarea name="Comments" id="Comments" cols="30" rows="3"
    >>> wrap="virtual"></textarea>
218. </td>
219. </tr>
220. <tr>
221. <td colspan="2">
222. <input type="checkbox" name="Terms" id="Terms" value="checkbox">
223. I agree to the site terms.
224. </td>
225. </tr>
226. <tr>
227. <td colspan="2" align="center">
228. <input name="submit" type="submit" value="Submit">
229. <input name="reset" type="reset" value="Reset Form">
230. </td>
231. </tr>
232. </table>
233. </form>
234. </body>
235. </html>
```

## **7.8 Conclusion**

In this lesson, you have learned to capture form events, to reference form fields, and to write clean, reusable form validation scripts.



## 8. The HTML Document Object Model

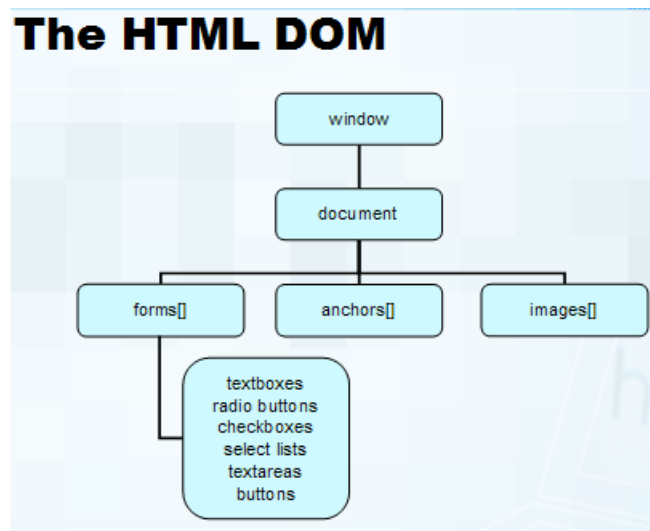
**In this lesson, you will learn...**

1. To work with the HTML DOM.
2. To access specific nodes.
3. To access nodes by class name.
4. To remove a node.
5. To create a new node.
6. To dynamically create an HTML page.
7. To modify node styles.
8. To modify node classes.
9. To add events to nodes.

The HTML Document Object Model (DOM) is a W3C standard that defines a set of HTML objects and their methods and properties. JavaScript can be used to create and destroy these objects, to invoke their methods, and to manipulate their properties.

In HTML5, the DOM has been expanded and some already widely supported previously unofficial features have been included as part of the specification.

If you have done any JavaScript programming on the web then you have almost definitely worked with the HTML DOM. A subset of the object hierarchy is shown below.



This course assumes that you have some experience working with the DOM. This lesson is concerned with the different ways of identifying and manipulating document nodes. While we have looked at some of these features in previous lessons, we present them here together for completeness.

## 8.1 The innerHTML Property

Most HTML elements have an `innerHTML` property, which can be used to access and modify the HTML within that element. The `innerHTML` property wasn't included in any specification until HTML5. However, as it's extremely useful and well supported, we will use it widely throughout this course.

### innerHTML Illustration

Given the code:

```
<p>I <strong>love</strong> Webucator.</p>
```

the `innerHTML` property of the `<p>` tag would be: "I `<strong>love</strong>` Webucator."

Tip: you can use the `innerHTML` property to either **get** the element's `innerHTML` value (as shown above) or you can use it to **set** the element's `innerHTML` value. More on this later in the lesson.

## 8.2 Accessing Element Nodes

JavaScript provides several different ways to access elements on the page. Let's have a look.

### `getElementById()`

Chances are you have already seen the `document.getElementById(id)` method, which returns the first element with the given `id` (there shouldn't be more than one on the page!) or `null` if none is found. The following example illustrates how `getElementById()` works:

**Code Sample****HTMLDOM/Demos/getElementById.html**

```
-----Lines 1 through 5 Omitted-----
6.  function show() {
7.    var elem = document.getElementById("beatleList");
8.    alert(elem.innerHTML);
9.  }
-----Lines 10 through 12 Omitted-----
13. <body onload="show();">
14. <h1>Rockbands</h1>
15. <h2>Beatles</h2>
16. <ol id="beatleList">
17.   <li>Paul</li>
18.   <li>John</li>
19.   <li>George</li>
20.   <li>Ringo</li>
21. </ol>
22. <h2>Rolling Stones</h2>
23. <ol id="stonesList">
24.   <li>Mick</li>
25.   <li>Keith</li>
26.   <li>Charlie</li>
27.   <li>Bill</li>
28. </ol>
29. </body>
30. </html>
```

### Code Explanation

When this page loads, the following alert box will pop up:



### **getElementsByTagName ( )**

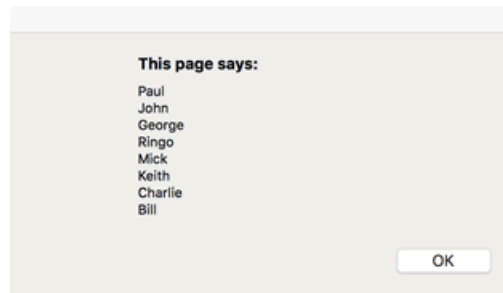
The `getElementsByTagName ( )` method of an element node retrieves all descendant (children, grandchildren, etc.) elements of the specified tag name and stores them in a `NodeList`, which is similar, but not exactly the same as an array of nodes. The following example illustrates how `getElementsByTagName ( )` works:

**Code Sample****HTMLDOM/Demos/getElementsByTagName.html**

```
-----Lines 1 through 5 Omitted-----
6.  function getElements()
7.  {
8.    var elems = document.getElementsByTagName("li");
9.    var msg = "";
10.   for (var i=0; i < elems.length; i++)
11.   {
12.     msg += elems[i].innerHTML + "\n";
13.   }
14.   alert(msg);
15. }
16. </script>
17. <title>getElementsByTagName()</title>
18. </head>
19.
20. <body onload="getElements();">
21. <h1>Rockbands</h1>
22. <h2>Beatles</h2>
23. <ol>
24.   <li>Paul</li>
25.   <li>John</li>
26.   <li>George</li>
27.   <li>Ringo</li>
28. </ol>
29. <h2>Rolling Stones</h2>
30. <ol>
31.   <li>Mick</li>
32.   <li>Keith</li>
33.   <li>Charlie</li>
34.   <li>Bill</li>
35. </ol>
36. </body>
37. </html>
```

### Code Explanation

When this page loads, the following alert box will pop up:



### `getElementsByClassName()`

The `getElementsByClassName()` method is well supported by browsers and is officially part of HTML5. Applicable to all elements that can have descendant elements, `getElementsByClassName()` is used to retrieve all the descendant (children, grandchildren, etc.) elements of a specific class. For example, the following code would return a node list containing all elements of the "warning" class:

```
var warnings = document.getElementsByClassName("warning");
```

### `querySelectorAll()`

The `document.querySelectorAll()` method gives you access to elements using the CSS selector syntax. For example, the following code would return a node list containing all list items that are direct children of `ol` tags:

```
var orderedListItems = document.querySelectorAll("ol>li");
```

### `querySelector()`

The `document.querySelector()` method is the same as `document.querySelectorAll()` but rather than returning a node list, it returns only the first element found. The following two lines of code would both return the first list item found in an `ol` tag:

```
var firstOrderedListItem = document.querySelectorAll("ol>li")[0];  
var firstOrderedListItem = document.querySelector("ol>li");
```

## The `this` Object

The `this` keyword provides access to the current object. It references the object in which it appears.

The following example shows a page ([HTMLDOM/Demos/Hierarchy.html](http://HTMLDOM/Demos/Hierarchy.html)) on which a mouse click on any element will display a message giving information about that element's position in the object hierarchy.

### Heading 1

Click on any element to get information about its position in the DOM.

### Heading 2

- List Item 1
- List Item 2
- List Item 3



Table Heading Table Heading Table Heading

[Link](#) Table Data Table Data  
Table Data Table Data Table Data

 Option 1 

When you click on the middle list item, for example, you get the following message:

**You clicked on a LI element.**

1. parentNode: UL
2. firstChild: #text
3. lastChild: #text
4. nextSibling: #text
5. previousSibling: #text
6. childNodes.length: 1

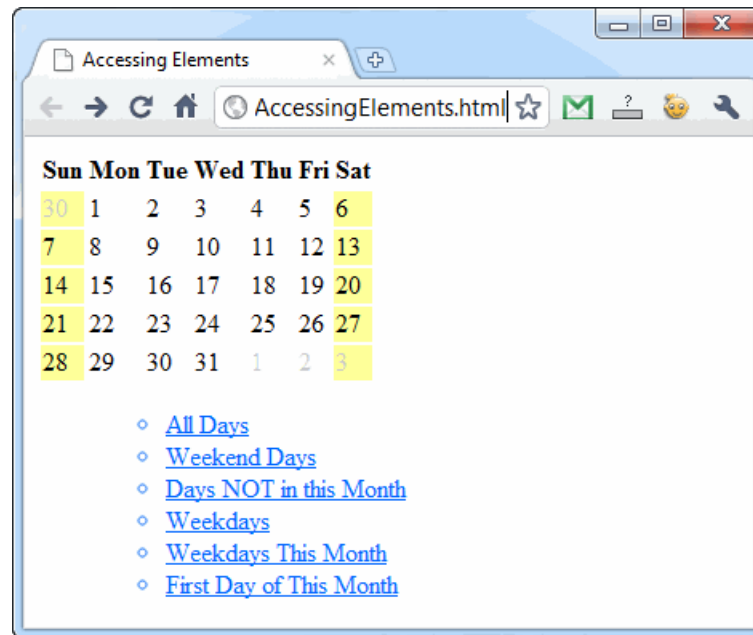
This shows that the list item's parent is an unordered list (UL), its first child is a text node ("List Item 2"), its last child is the same as it only has one child, its next and previous siblings are both text nodes, and it has only one child node (the text node). Again, this message was generated in Firefox, but would be the same in other modern browsers.

## Exercise 16 Accessing Elements

*25 to 40 minutes*

In this exercise, you will practice using the methods for accessing elements.

1. Open [HTMLDOM/Solutions/AccessingElements.html](#) in your **browser**. It will look like this:



2. Click on the links and notice how the calendar changes. For example, click on **Weekdays This Month** and the calendar will appear as follows:

Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3

3. Now open [HTMLDOM/Exercises/AccessingElements.html](#) in your **editor**.



4. Notice that `ClassFiles/lib.js` is included and that there are four functions already written:
  - A. `highlight()` - adds the "highlight" class to the passed-in node or nodes.
  - B. `unhighlight()` - removes the "highlight" class from the passed-in node or nodes.
  - C. `unhighlightDays()` - helper function removing the "highlight" class from all the `td` elements to reset the calendar.
  - D. `weekEndDays()` - gets all the weekend days using the `getElementsByClassName()` helper function from the `ClassFiles/lib.js` library and passes it to the `highlight()` function after calling `unhighlightDays()` to reset the calendar.
5. Complete the `offDays()` function to highlight the days that are not in the current month (that have the "off" class).
6. Complete the `allDays()` function to highlight all the days of the month.
7. Complete the `firstDayOfMonth()` function to highlight the first day of the current month (the one that contains a value of "1"). Note that every `table` element contains a `rows` array, so you can access the first row in a table with `table.rows[0]`.
8. Complete the `weekDays()` function to highlight all the week days. Use `querySelectorAll` if the browser supports it.
9. **Challenge:** Complete the `weekDaysThisMonth()` function to highlight all the weekdays of the month. **Hint:** You can make use of the `weekDays()` function.

### Exercise Solution

#### HTMLDOM/Solutions/AccessingElements.html

```
-----Lines 1 through 74 Omitted-----
75.  function offDays() {
76.    //first get the calendar by its ID
77.    var calendar = document.getElementById("calendar");
78.    //from the calendar, get all the weekend days (the td tags with a css
    >>> class of "off")
79.    var offDays = getElementsByClassName(calendar,"off");
80.    //first "unhighlight" all the days
81.    unhighlightDays();
82.    //now, highlight the "off" days
83.    highlight(offDays);
84.    return offDays;
85.  }
86.
87.  function allDays() {
88.
89.    //first get the calendar by its ID
90.    var calendar = document.getElementById("calendar");
91.    //from the calendar, get all the days (the td tags)
92.    var days = calendar.getElementsByTagName("td");
93.    //now, highlight all the "days"
94.    highlight(days);
95.    return days;
96.  }
97.
98.  function firstDayOfMonth() {
99.
100.   //first get the calendar by its ID
101.   var calendar = document.getElementById("calendar");
102.   //the "calendar" is a table, so it has rows. The first row in the
    >>> calendar array is at index position 0 (the header)
103.   var headerRow = calendar.rows[0];
104.   //the 2nd row in the calendar array is at index position 1 (the first
    >>> data row)
105.   var firstRow = calendar.rows[1];
106.   //get all the td tags
107.   var tds = firstRow.getElementsByTagName("td");
108.   //unhighlight all the days
109.   unhighlightDays();
110.   //loop all the td tags in the calendar
```

```
111.     for (var i=0; i<tds.length; i++) {
112.         //as you loop the td tags, if the html in "this" td tag is 1 (the
113.         >>> 1st day of the month), then highlight this td tag
114.         if ( tds[i].innerHTML==1 ) {
115.             highlight(tds[i]);
116.             return tds[i];
117.         }
118.     }
119.
120.     function weekDays() {
121.         var weekDays;
122.         if ( document.querySelectorAll ) {
123.
124.             //get all the days that are not of class "weekend" - in other words,
125.             >>> get the weekdays
126.             weekDays = document.querySelectorAll("td:not(.weekend)");
127.         } else {
128.             var calendar = document.getElementById("calendar");
129.             var days = calendar.getElementsByTagName("td");
130.             var weekDays = [];
131.             for (var i=0; i<days.length; i++) {
132.                 if ( !hasClass(days[i], "weekend") ) {
133.                     weekDays.push(days[i]);
134.                 }
135.             }
136.             //unhighlight all days first
137.             unhighlightDays();
138.             //now just highlight all the "weekDays"
139.             highlight(weekDays);
140.             return weekDays;
141.         }
142.
143.         function weekDaysThisMonth() {
144.
145.             //create empty array to receive days as we find them
146.             var weekDaysThisMonth = [];
147.
148.             // start by getting all weekdays, regardless of month
149.             var days=weekDays();
150.             var i;
```

---

## The HTML Document Object Model

```
151.    //loop through all the week days, if it does NOT have a css class of
      >>> "off", then add the day to the weekDaysThisMonth array.
152.    //Note that the push() method simply adds an element to an array
153.    for (i=0; i<days.length; ++i) {
154.        if ( !hasClassName(days[i],"off") ) {
155.            weekDaysThisMonth.push(days[i]);
156.        }
157.    }
158.    //now unhighlight all the days
159.    unhighlightDays();
160.    //and finally, highlight all the "weekDaysThisMonth"
161.    highlight(weekDaysThisMonth);
162.    return weekDaysThisMonth;
163. }
```

-----Lines 164 through 239 Omitted-----

## 8.3 Accessing Attribute Nodes

### **getAttribute()**

The `getAttribute()` method of an element node returns the attribute value as a string or null if the attribute doesn't exist. The syntax is shown below:

```
myNode.getAttribute("AttName");
```

Most HTML attributes are also directly available as properties of the element node; for example, for an `img` tag element, the `width`, `height`, and `source` are available as the `width`, `height`, and `src` properties.

### **attributes[]**

The `attributes[]` property references the collection of a node's attributes; in practice, though, it's better to access attributes with the `getAttribute()` method.

### **hasAttribute()**

The `hasAttribute(attName)` method returns a Boolean (true/false) value indicating whether or not the element to which the method is applied includes the given attribute. For example, the code

```
var newsletterlink = document.getElementById("newsletterlink");
if (newsletterlink.hasAttribute("target")) {
    alert("newsletter link opens in new window");
}
```

would pop up a JavaScript alert if the link with `id newsletterlink` had a `target` attribute, such as:

```
<a href="nl.pdf" id="newsletterlink" target="_blank">View Newslet »»
ter</a>
```

Note that method `hasAttribute()` does not check the value of the attribute, only if the attribute exists for the given element.

### **setAttribute()**

Method `setAttribute(attName, attValue)` allows us to add the given attribute to an element (if the attribute does not exist) or to change it (if the attribute does exist), and assign the given value. For instance, if we had this HTML:

```
<a href="nl.pdf" id="newsletterlink" target="_blank">View Newslet »»
ter</a>
```

then the following code:

```
var newsletterlink = document.getElementById("newsletterlink");
newsletterlink.setAttribute("target", "_self");
```

would change the value of the `target` attribute to `_self`.

## **removeAttribute()**

Method `removeAttribute(attName)` removes the attribute from an element; attempting to remove an attribute that does not exist does not produce an error. The following code

```
var newsletterlink = document.getElementById("newsletterlink");
newsletterlink.removeAttribute("target");
```

would remove the `target` attribute from the link, if the link had that attribute.

## **8.4 Removing Nodes from the DOM**

Element nodes have a `removeChild()` method, which takes a single parameter: the child node to be removed. There is no W3C method for a node to remove itself, but the following code will do the trick:

```
elem.parentNode.removeChild(elem);
elem = null;
```

`removeChild(elem)` only removes the element from its parent. Setting `elem` to `null` destroys the element.

Sometimes it's useful to remove all of a node's children in one fell swoop. The code below will handle this:

```
while (parent.hasChildNodes()) {
    removeElement(parent.childNodes[0]);
}
```

## **8.5 Creating New Nodes**

The document node has separate methods for creating element nodes and creating text nodes: `createElement()` and `createTextNode()`. These methods each

create a node in memory that then has to be placed somewhere in the object hierarchy. A new node can be inserted as a child to an existing node with that node's `appendChild()` and `insertBefore()` methods.

You can also use the `appendChild()` and `insertBefore()` methods to move an existing node - the node will be removed from its current location and placed at the new location (since the same node cannot exist twice in the same document).

These methods and some others are described in the table below.

#### Methods for Inserting Nodes

Method	Description
<code>appendChild()</code>	Takes a single parameter: the node to insert, and inserts that node after the last child node.
<code>insertBefore()</code>	Takes two parameters: the node to insert and the child node that it should precede. The new child node is inserted before the referenced child node.
<code>replaceChild()</code>	Takes two parameters: the new node and the node to be replaced. It replaces the old node with the new node and returns the old node.
<code>setAttribute()</code>	Takes two parameters: the attribute name and value. If the attribute already exists, it replaces it with the new value. If it doesn't exist, it creates it.

The sample below illustrates how these methods work.

### Code Sample

#### HTMLDOM/Demos/InsertingNodes.html

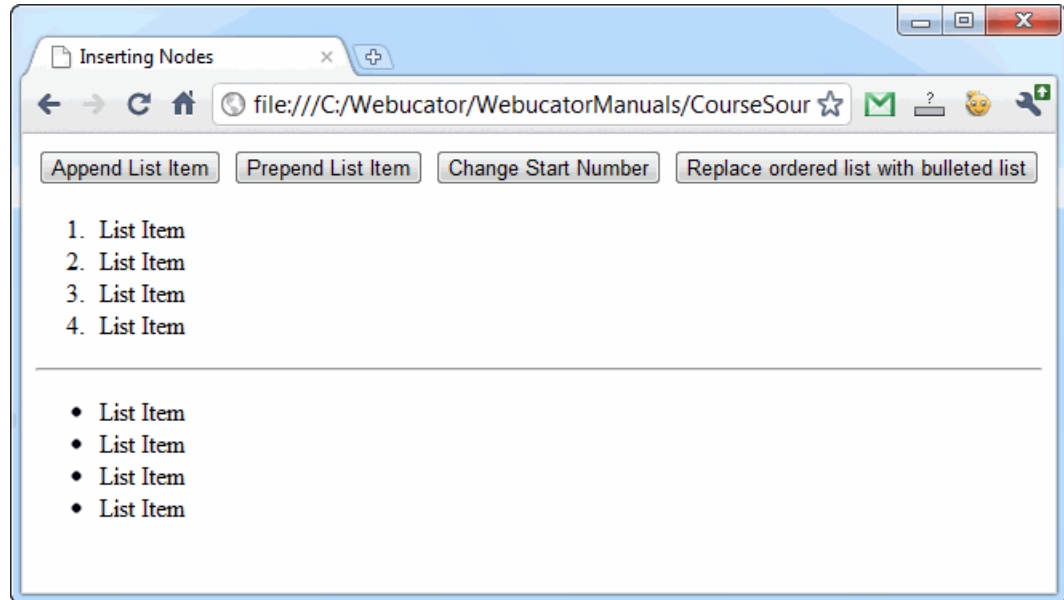
```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <script>
6.  window.onload = function() {
7.    document.getElementById('append').addEventListener('click', append »»
      >>> NewListItem);
8.    document.getElementById('prepend').addEventListener('click', prepend »»
      >>> NewListItem);
9.    document.getElementById('change-num').addEventListener('click',
      >>> changeStartNum);
10.   document.getElementById('replace').addEventListener('click', replace »»
      >>> OlWithUl);
11.  };
12.
13.  function appendNewListItem() {
14.    var li = document.createElement("li");
15.    var liText = document.createTextNode("New List Item");
16.    li.appendChild(liText);
17.    document.getElementById("first-list").appendChild(li);
18.  }
19.
20.  function prependNewListItem() {
21.    var li = document.createElement("li");
22.    var liText = document.createTextNode("New List Item");
23.    var firstList = document.getElementById("first-list");
24.    li.appendChild(liText);
25.    firstList.insertBefore(li,firstList.firstChild);
26.  }
27.
28.  function changeStartNum() {
29.    var firstList = document.getElementById("first-list");
30.    var start = firstList.getAttribute("start");
31.    if (start) {
32.      start++;
33.    } else {
34.      start=2;
35.    }
36.    firstList.setAttribute("start",start);
```



```
37.  }
38.
39.  function replaceOlWithUl() {
40.    var list = document.getElementById("first-list");
41.    var list2 = document.getElementById("second-list");
42.    list.parentNode.replaceChild(list2,list);
43.  }
44.  </script>
45.  <title>Inserting Nodes</title>
46.  </head>
47.
48.  <body>
49.  <form id="menu">
50.    <button id="append" type="button">Append List Item</button>
51.    <button id="prepend" type="button">Prepend List Item</button>
52.    <button id="change-num" type="button">Change Start Number</button>
53.    <button id="replace" type="button">Replace ordered list with bulleted
    >>> list</button>
54.  </form>
55.  <ol id="first-list">
56.    <li>List Item</li>
57.    <li>List Item</li>
58.    <li>List Item</li>
59.    <li>List Item</li>
60.  </ol>
61.  <hr>
62.  <ul id="second-list">
63.    <li>List Item</li>
64.    <li>List Item</li>
65.    <li>List Item</li>
66.    <li>List Item</li>
67.  </ul>
68.  </body>
69.  </html>
```

### Code Explanation

The page is shown below in a browser. Click on any of the menu items to see the methods in action. The unordered list will move when it replaces the ordered list, so it will then appear above the horizontal rule line.



As a coding practice, it is better to create a structure before inserting it into the page than to put the parent element in the page and then append to it. This saves the browser from unnecessary redrawing of the page as each child is added.

### A Note on the `setAttribute()` Method

You can use the `setAttribute()` method to change the value of all attributes by name as you would expect with one exception. In Internet Explorer 8 and earlier, the `class` attribute must be referred to as `"className"` in the `setAttribute()` method. This means that you shouldn't use `setAttribute()` for setting the class. Instead, change the `className` property.

#### Syntax

```
node.className = "new-class-name";
```

Or you can use the `addClass()` function in our [ClassFiles/lib.js](#) file.

## 8.6 Identifying the Target of an Event

Often you will want to take action on the target of an event. For example, in a drag-and-drop application, you click down on an element and drag it around the screen. To write that code, you need to be able to identify which element receives the click.

In most modern browsers, this is done with the `target` property of the event itself. The following demo illustrates:

### Code Sample

#### HTMLDOM/Demos/target.html

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <link rel="stylesheet" href="../Styles/propagation.css">
6.  <script>
7.      window.onload = function() {
8.          document.getElementById('outer').addEventListener('click', function(e)
9.              >>> {
10.             findTarget(e);
11.         });
12.         document.getElementById('inner').addEventListener('click', function(e)
13.             >>> {
14.             findTarget(e);
15.         });
16.         document.getElementById('heading').addEventListener('click', func
17.             >>> tion(e) {
18.             findTarget(e);
19.         });
20.     }
21.
22.     function findTarget(e) {
23.         e.target.innerHTML += " | click | ";
24.         e.stopPropagation();
25.     }
26.
27. </script>
28. <title>Target</title>
29. </head>
30. <body>
31.     <h1 id="heading">Target</h1>
32.     <div id="outer">outer
33.         <div id="inner">inner</div>
34.     </div>
35. </body>
36. </html>
```

### Code Explanation

In your browser, click on any of the elements: **outer**, **inner**, or the h1 element and the word "click" will get added to the element.

## Exercise 17 Manipulating the DOM

*25 to 35 minutes*

In this exercise, you will add JavaScript code to a page that displays information about several Beatles records - users will be able to toggle the display of more details about each record and add records to a "favorites" list.

1. Open [HTMLDOM/Exercises/ManipulatingDOM.html](http://HTMLDOM/Exercises/ManipulatingDOM.html) in a code editor and in a browser.
2. Note that the CSS for the page is done for you: there are style rules to display the records (floated into two-across blocks) and other layout details for the page. In particular, note the rules for `span.more` and `span.more`; these elements are set to display (or not) depending on whether their parent element (the `div`s of class `record`) have a class of `active`.
3. Write the body of function `addToFavorites` which adds the title of the record to the "favorites" element when its "Add To Favorites" link is clicked.
4. Add code to toggle the details content for each record: toggle the class `active` on the clicked `.record` element, which will (through the already-written CSS rules) change the display of the "More..." link and the detail content.
5. Test your work.

### Exercise Solution

#### HTMLDOM/Solutions/ManipulatingDOM.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <style>
6.        section#records {
7.            float:left;
8.            width:70%;
9.        }
10.    aside#favorites {
11.        float:right;
12.        width:25%;
13.    }
14.    .record {
15.        width:40%;
16.        float:left;
17.        padding:2%;
18.        background:#efefef;
19.        margin:0 2% 20px 0;
20.    }
21.    .record:nth-child(2n){
22.        clear:left
23.    }
24.    .record span.more {
25.        font-style: italic;
26.    }
27.    .record span.details {
28.        display: none;
29.    }
30.    .record.active span.details {
31.        display: inline;
32.    }
33.    .record.active span.more {
34.        display: none;
35.    }
36.    .footer {
37.        clear:both;
38.        font-style: italic;
39.        padding-top: 30px;
```

```

40.     }
41. </style>
42. <script>
43.     window.onload = function() {
44.
45.         function addToFavorites(node) {
46.             var recordListItem = document.createElement('p');
47.             var recordListItemText = document.createTextNode(document.querySelector('#' + node.id + ' h3 span.title').innerHTML);
48.             recordListItem.appendChild(recordListItemText);
49.             document.getElementById('favorites').appendChild(recordListItem);
50.         }
51.
52.         var detailElements = document.getElementsByClassName('record');
53.         for (var i=0; i < detailElements.length; i++) {
54.             detailElements[i].addEventListener('click', function(e) {
55.                 if (e.target.className == 'addtofavorites') {
56.                     addToFavorites(e.target.parentNode.parentNode);
57.                     e.target.parentNode.removeChild(e.target);
58.                 } else {
59.                     if (e.currentTarget.className == 'record') {
60.                         e.currentTarget.className = 'record active';
61.                     } else {
62.                         e.currentTarget.className = 'record';
63.                     }
64.                     var myToggle = document.querySelector('#' + e.currentTarget.id +
65.                         ' h3 span.toggle');
66.                     var myToggleContents = myToggle.innerHTML;
67.                     if (myToggleContents == '+') {
68.                         myToggle.innerHTML = '-';
69.                     } else {
70.                         myToggle.innerHTML = '+';
71.                     }
72.                 });
73.             }
74.
75.         };
76. </script>
77. <title>Manipulating the DOM</title>
78. </head>
79.
80. <body>

```

```
81. <section id="records">
82.   <h1>Selected Beatles Records</h1>
83.   <div class="record" id="harddaysnight">
84.     <h3><span class="title">A Hard Day's Night</span> <span class="tog »»
      >>> gle">+</span></h3>
85.     <p>A Hard Day's Night is the third studio album by the English rock
      >>> band the Beatles, released on 10 July 1964, with side one con »»
      >>> taining songs from the soundtrack to their film A Hard Day's
      >>> Night.<span class="more"> More...</span><span class="details">
      >>> The American version of the album was released two weeks ear »»
      >>> lier, on 26 June 1964 by United Artists Records, with a differ »»
      >>> ent track listing. In contrast to their first two albums, all
      >>> 13 tracks on A Hard Day's Night were written by John Lennon
      >>> and Paul McCartney showcasing the development of their songwrit »»
      >>> ing talents. The album includes the title track, with its dis »»
      >>> tinct opening chord,[4] and the previously released "Can't Buy
      >>> Me Love", both transatlantic number-one singles for the
      >>> band.</span></p>
86.     <p><a href="#" class="addtofavorites">Add To Favorites</a></p>
87.   </div>
88.   <div class="record" id="rubbersoul">
89.     <h3><span class="title">Rubber Soul</span> <span class="tog »»
      >>> gle">+</span></h3>
90.     <p>Rubber Soul is an album by the Beatles, their sixth UK album, and
      >>> the tenth released in America.<span class="more">
      >>> More...</span><span class="details"> Released on 3 December
      >>> 1965, it met with a highly favourable critical response and
      >>> topped record charts in the United Kingdom for several weeks,
      >>> as well as in the United States, where it was issued with a
      >>> different selection of tracks.</span></p>
91.     <p><a href="#" class="addtofavorites">Add To Favorites</a></p>
92.   </div>
93.   <div class="record" id="abbeyroad">
94.     <h3><span class="title">Abbey Road</span> <span class="tog »»
      >>> gle">+</span></h3>
95.     <p>Abbey Road is the eleventh studio album by English rock band the
      >>> Beatles, released on 26 September 1969 by Apple Records.<span
      >>> class="more"> More...</span><span class="details"> The
      >>> recording sessions for the album were the last in which all
      >>> four Beatles participated. Although Let It Be was the final
      >>> album that the Beatles completed before the band's dissolution
      >>> in April 1970, most of the album had been recorded before the
      >>> Abbey Road sessions began.[1] A double A-side single from the
      >>> album, "Something"/"Come Together", released in October, topped
      >>> the Billboard chart in the US.</span></p>
96.     <p><a href="#" class="addtofavorites">Add To Favorites</a></p>
97.   </div>
98.   <div class="record" id="letitbe">
```



```

99.     <h3><span class="title">Let It Be</span> <span class="tog »
      >>> gle">+</span></h3>
100.    <p>Let It Be is the twelfth and final studio album by the English
      >>> rock band the Beatles.<span class="more"> More...</span><span
      >>> class="details"> It was released on 8 May 1970, almost a month
      >>> after the group's break-up. Like most of the band's previous
      >>> releases, it was a number one album in many countries, includ »
      >>> ing both the US and the UK, and was released in tandem with
      >>> the motion picture of the same name.</span></p>
101.    <p><a href="#" class="addtofavorites">Add To Favorites</a></p>
102.  </div>
103. </section>
104. <aside id="favorites">
105.   <h2>My Favorites</h2>
106. </aside>
107. <p class="footer">Content taken from <a
      >>> href="https://en.wikipedia.org/wiki/A_Hard_Day%27s_Night_(al »
      >>> bum)">Wikipedia</a></p>
108. </body>
109. </html>

```

### Code Explanation

When the "Add To Favorites" link is clicked for any record, we call the `addToFavorites` function, passing to it the `.record` node element from which the click came, and remove the "Add To Favorites" link (so that the user can't add the record twice). Function `addToFavorites` creates a new paragraph node, adds text (the title of the record) to it, and adds the node to the sidebar "Favorites" area.

When the user clicks any record, we toggle the active class (using `e.currentTarget.className`) and toggle between the + and - characters (using `myToggle.innerHTML`.)

## 8.7 Conclusion

In this lesson, you have learned to work with the HTML DOM to create and modify HTML page elements dynamically with JavaScript.

## 9. CSS Object Model

### In this lesson, you will learn...

1. To change the values of CSS properties dynamically.
2. To hide and show elements.
3. To dynamically modify the content of elements.
4. To manipulate tables dynamically.
5. To position elements dynamically.
6. To change the z-index order of elements.

We can use JavaScript to both retrieve information about element's CSS styles and to set those styles programmatically.

### 9.1 Changing CSS with JavaScript

Throughout this course we've changed the style of a DOM element with JavaScript - setting the background color of the page, say, or altering the text color of an element. Typically, we use the following syntax:

```
element.style.cssproperty = 'cssvalue';
```

where `element` might be a DOM element gotten from its `id` (via `getElementById()`), `cssproperty` something like `color` (to set the text color), and `cssvalue` something like `red`. We can both get and set any styles for most any element.

Each CSS property has a corresponding property of the JavaScript `style` object:

- If the CSS property is a simple word (e.g., `color`) then the JavaScript property is the same (e.g., `style.color`).
- If the CSS property has a dash in it (e.g., `background-color`) then the JavaScript property uses lower camel case (e.g., `style.backgroundColor`).

The `style` object is a collection of an element's styles that are either defined within that HTML element's `style` attribute or directly in JavaScript. Styles defined in the `<style>` tag or in an external style sheet are not part of the `style` object.

The W3C specifies a method for getting at the current (or actual) style of an object: the window object's `getComputedStyle()` method.

```
window.getComputedStyle(Element)
```

Note that the reference to window can be excluded as window is the implicit object. For example:

```
var div = document.getElementById("divTitle");  
var computedStyle = getComputedStyle(div);  
alert(computedStyle.fontWeight);  
var curStyle = getComputedStyle(div);  
alert(curStyle.fontWeight);
```

Using this method - with `getComputedStyle()`, as opposed to `element.style` - we can get for any element the styles set with inline CSS, from CSS in the head of the page, or in an external stylesheet. Furthermore, as the name of the method `getComputedStyle()` suggests, these are computed (calculated) styles: whereas `element.style` just gives us style info as set in CSS, `getComputedStyle()` gives us the real-time calculated CSS.

Let's take a look at a simple example to make this more clear.

**Code Sample****CSSObjectModel/Demos/styles.html**

```

1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Styles</title>
6.    <link href="customstyles.css" rel="stylesheet">
7.    <script>
8.        window.onload = function() {
9.            document.getElementById("getstyles").addEventListener("click", func >>
                >>> tion() {
10.                var pgraph = document.getElementById("pgraph");
11.                var styleString = "Styles\n";
12.                styleString += 'color: ' + pgraph.style.color + "\n";
13.                styleString += 'margin: ' + pgraph.style.margin + "\n";
14.                styleString += 'padding: ' + pgraph.style.padding + "\n";
15.                styleString += 'border: ' + pgraph.style.borderBottom + "\n";
16.                styleString += 'width: ' + pgraph.style.width + "\n";
17.                alert(styleString);
18.
19.                var computedStyle = getComputedStyle(pgraph);
20.                var computedStyleString = "Computed Styles\n";
21.                computedStyleString += 'color: ' + computedStyle.color + "\n";
22.                computedStyleString += 'margin: ' + computedStyle.margin + "\n";
23.                computedStyleString += 'padding: ' + computedStyle.padding + "\n";
24.                computedStyleString += 'border: ' + computedStyle.borderBottom +
                >>> "\n";
25.                computedStyleString += 'width: ' + computedStyle.width + "\n";
26.                alert(computedStyleString);
27.                return false;
28.            });
29.            document.getElementById("setstyles").addEventListener("click", func >>
                >>> tion() {
30.                var pgraph = document.getElementById("pgraph");
31.                pgraph.style.borderBottom = 'none';
32.                pgraph.style.fontStyle = 'normal';
33.                pgraph.style.width = 'auto';
34.                pgraph.style.padding = 'auto';
35.                pgraph.style.margin = 'auto';
36.                return false;
37.            });

```

```
38.     }
39.   </script>
40.   <style>
41.     #pgraph {
42.       border-bottom:2px solid blue;
43.       width:80%;
44.       padding:10px;
45.     }
46.   </style>
47. </head>
48. <body>
49. <h1>Styles</h1>
50. <p id="pgraph" style="margin:10px auto 30px auto">This is a paragraph</p>
    >>>
51. <button id="getstyles">Get Styles</button>
52. <button id="setstyles">Set Styles</button>
53. </body>
54. </html>
```

### Code Explanation

The paragraph, with id `pgraph`, has CSS styles set in three different ways: inline (margin), in the head of the page (border-bottom, width, and padding), and from an external stylesheet `customstyles.css` (font-size, color, and font-style).

Clicking the "Get Styles" button invokes a handler which pops up two JavaScript alerts. The first lists the styles retrieved `pgraph.style`; note that color, padding, border, and width are all blank. The second alert, however, displays style information retrieved from `getComputedStyle(pgraph)`; note that all of the style information - regardless of whether it was added inline, in the head of the page, or from the external stylesheet, is displayed.

The "Set Styles" button, when clicked, changes the styles of the paragraph. Check out how the two methods of getting the styles differ when you click "Get Styles" after setting the styles.

## 9.2 Hiding and Showing Elements

Elements can be hidden and shown by changing their `visibility` or `display` values. The `visibility` style can be set to `visible` or `hidden` and the `display` property can be set to `block`, `table-row`, `none`, and several other values. The two work slightly differently as the following example illustrates:

**Code Sample****CSSObjectModel/Demos/Visibility.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Showing and Hiding Elements with JavaScript</title>
6.    <link href="visibility.css" rel="stylesheet">
7.    <script>
8.    function changeVisibility(id){
9.        var elem = document.getElementById(id);
10.       var visibility;
11.       if (elem.style.visibility == "hidden") {
12.           elem.style.visibility = "visible";
13.       } else {
14.           elem.style.visibility = "hidden";
15.       }
16.       visibility = getCurrentStyle(elem,"visibility");
17.       msg("The <em>visibility</em> of row <em>" + id + "</em> is <em>" +
18.           >>> visibility + "</em>.");
19.
20.   function changeDisplay(id){
21.       var elem = document.getElementById(id);
22.       var display;
23.       if (elem.style.display == "none") {
24.           elem.style.display = "";
25.       } else {
26.           elem.style.display = "none";
27.       }
28.       display = getCurrentStyle(elem,"display");
29.       msg("The <em>display</em> of row <em>" + id + "</em> is <em>" + display
30.           >>> + "</em>.");
31.
32.   function msg(text) {
33.       document.getElementById("msg").innerHTML = text;
34.   }
35. </script>
36. </head>
37. <body>
```

```
38. <h1>Hiding and Showing Elements</h1>
39. <table>
40.   <tr id="tr1"><td>Row 1</td></tr>
41.   <tr id="tr2"><td>Row 2</td></tr>
42.   <tr id="tr3"><td>Row 3</td></tr>
43.   <tr id="tr4"><td>Row 4</td></tr>
44. </table>
45. <div id="msg">Style messages</div>
46. <h2>visibility</h2>
47. <button onclick="changeVisibility('tr1')">Row 1</button>
48. <button onclick="changeVisibility('tr2')">Row 2</button>
49. <button onclick="changeVisibility('tr3')">Row 3</button>
50. <button onclick="changeVisibility('tr4')">Row 4</button>
51.
52. <h2>display</h2>
53. <button onclick="changeDisplay('tr1')">Row 1</button>
54. <button onclick="changeDisplay('tr2')">Row 2</button>
55. <button onclick="changeDisplay('tr3')">Row 3</button>
56. <button onclick="changeDisplay('tr4')">Row 4</button>
57. </body>
58. </html>
```

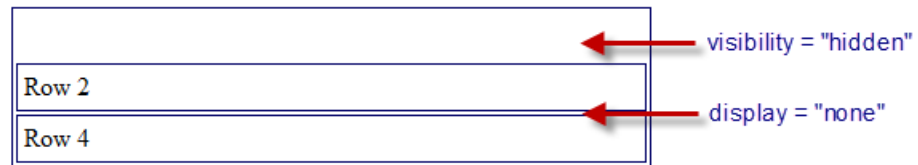
### Code Explanation

This page has two functions: `changeVisibility()` and `changeDisplay()`. The `changeVisibility()` function checks the value of the `visibility` style of the passed element and changes it to its opposite. The `changeDisplay()` function does the same with the `display` style. The functions are called with buttons on the page and are passed in ids of table rows from the table on the page.

The main difference between setting `visibility` to `hidden` and setting `display` to `none` is that setting `visibility` to `hidden` does not modify the layout of the page; it simply hides the element. Setting `display` to `none`, on the other hand, collapses the element, so that the surrounding relatively positioned elements re-position themselves.



Check out the screenshot below:



*Row 1* has `visibility` set to `hidden`, so you can see the space for the row, but you cannot see the row itself. *Row 3* has `display` set to `none` so it's as if the row was not even there.

## Exercise 18 Showing and Hiding Elements

*20 to 30 minutes*

In this exercise, you will modify a Math Quiz to only show the countdown timer when it is running.

1. Open [CSSObjectModel/Exercises/MathQuiz.html](#) for editing.
2. Modify the code so that the table row with the timer in it (the last row) only shows up when the timer is running.

Category:	Addition
Question:	--Please Choose--
Answer:	<input type="text"/> Check Answer
Good luck!	

Category:	Addition
Question:	5 + 4
Answer:	<input type="text"/> Check Answer
Timer:	2 seconds left
Good luck!	

*Hint:* You will make your changes in `resetTimer()` and `questionChanged()`.

3. Test your solution in a browser.

### \*Challenge

Only show the Answer row when a question is selected.



**Exercise Solution****CSSObjectModel/Solutions/MathQuiz.html**

```
-----Lines 1 through 45 Omitted-----
46.
47.     function resetTimer(seconds) {
48.         var timerRow = document.getElementById("timerRow");
49.         timerRow.style.display = "none";
50.         document.getElementById("timeLeft").value = seconds;
51.         clearInterval(timer);
52.     }
-----Lines 53 through 96 Omitted-----
97.
98.     function questionChanged() {
99.         document.Quiz.answer.value="";
100.        var timerRow;
101.        if (document.Quiz.question.selectedIndex === 0) {
102.            document.Quiz.btnCheck.disabled = true;
103.            document.Quiz.answer.disabled = true;
104.            resetTimer(timePerQuestion);
105.        } else {
106.            document.Quiz.btnCheck.disabled = false;
107.            document.Quiz.answer.disabled = false;
108.            document.Quiz.answer.focus();
109.            timer = setInterval(decrementTimer,1000);
110.            timerRow = document.getElementById("timerRow");
111.            timerRow.style.display = "";
112.        }
113.    }
-----Lines 114 through 161 Omitted-----
162. <tr id="timerRow">
163.     <td>Timer:</td>
164.     <td><input type="text" name="timeLeft" id="timeLeft" size="2"> seconds
        >>> left</td>
165. </tr>
166. <tr>
167.     <td id="msg" colspan="2">Good luck!</td></tr>
168. </tr>
169. </table>
170. </form>
171. </body>
172. </html>
```

**Challenge Solution****CSSObjectModel/Solutions/MathQuiz-challenge.html**

```

-----Lines 1 through 45 Omitted-----
46.
47.     function resetTimer(seconds) {
48.         var timerRow = document.getElementById("timerRow");
49.         var answerRow = document.getElementById("answerRow");
50.         timerRow.style.display = "none";
51.         answerRow.style.display = "none";
52.         document.getElementById("timeLeft").value = seconds;
53.         clearInterval(timer);
54.     }
-----Lines 55 through 98 Omitted-----
99.
100.    function questionChanged() {
101.        document.Quiz.answer.value="";
102.        var timerRow,answerRow;
103.        if (document.Quiz.question.selectedIndex === 0) {
104.            document.Quiz.btnCheck.disabled = true;
105.            document.Quiz.answer.disabled = true;
106.            resetTimer(timePerQuestion);
107.        } else {
108.            document.Quiz.btnCheck.disabled = false;
109.            document.Quiz.answer.disabled = false;
110.            timer = setInterval(decrementTimer,1000);
111.            timerRow = document.getElementById("timerRow");
112.            timerRow.style.display = "";
113.            answerRow = document.getElementById("answerRow");
114.            answerRow.style.display = "";
115.            document.Quiz.answer.focus();
116.        }
117.    }
-----Lines 118 through 158 Omitted-----
159.    <tr id="answerRow">
-----Lines 160 through 178 Omitted-----

```

## **9.3 Manipulating Tables**

HTML tables can be created and manipulated dynamically with JavaScript. Each `table` element contains a `rows` array and methods for inserting and deleting rows: `insertRow()` and `deleteRow()`. Each `tr` element contains a `cells` array and methods for inserting and deleting cells: `insertCell()` and `deleteCell()`. The following example shows how these objects can be used to dynamically create HTML tables:

**Code Sample****CSSObjectModel/Demos/Table.html**

```

1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Manipulating Tables</title>
6.    <link href="table.css" rel="stylesheet">
7.    <script>
8.    function addRow(tableId, cells){
9.        var tableElem = document.getElementById(tableId);
10.       var newRow = tableElem.insertRow(tableElem.rows.length);
11.       var newCell;
12.       for (var i=0; i < cells.length; i++) {
13.           newCell = newRow.insertCell(newRow.cells.length);
14.           newCell.innerHTML = cells[i];
15.       }
16.       return newRow;
17.   }
18.
19.   function deleteRow(tableId, rowNumber){
20.       var tableElem = document.getElementById(tableId);
21.       if (rowNumber > 0 && rowNumber < tableElem.rows.length) {
22.           tableElem.deleteRow(rowNumber);
23.       } else {
24.           alert("Failed");
25.       }
26.   }
27.
28.   window.onload = function() {
29.       var btnAdd = document.getElementById("btnAdd");
30.       var btnDelete = document.getElementById("btnDelete");
31.       btnAdd.addEventListener('click', function() {
32.           var cells = [btnAdd.form.FirstName.value, btnAdd.form.LastName.value];
33.           >>>
34.           addRow('tblPeople', cells);
35.       });
36.       btnDelete.addEventListener('click', function() {
37.           deleteRow('tblPeople', btnDelete.form.RowNum.value)
38.       });
39.   };

```

## CSS Object Model

```
39. </script>
40. </head>
41. <body>
42. <table id="tblPeople">
43. <tr>
44. <th>First Name</th>
45. <th>Last Name</th>
46. </tr>
47. </table>
48. <hr>
49. <form name="formName">
50. <label for="FirstName">First Name:</label>
51. <input type="text" name="FirstName"><br>
52. <label for="LastName">Last Name:</label>
53. <input type="text" name="LastName"><br>
54. <input type="button" id="btnAdd" value="Add Name">
55. <hr>
56. <label for="RowNum">Remove Row:</label>
57. <input type="text" size="1" name="RowNum">
58. <input type="button" id="btnDelete" value="Delete Row">
59. </form>
60. </body>
61. </html>
```

### Code Explanation

The body of the page contains a table with an id of `tblPeople`. The table contains a single row of headers.

```
<table id="tblPeople">
<tr>
<th>First Name</th>
<th>Last Name</th>
</tr>
</table>
```

Below the table is a form that allows the user to enter a first and last name. When the "Add Name" button is clicked, the `addRow( )` function is called and passed in the id of the table (`tblPeople`) and a new array containing the user-entered values:

```
var cells = [btnAdd.form.FirstName.value, btnAdd.form.LastName.val »»
ue];
addRow('tblPeople', cells);
```



The `addRow()` function uses the `insertRow()` method of the `table` to add a new row at the end of the table and then loops through the passed-in array, creating and populating one cell for each item. The function also returns the new row. Although the returned value isn't used in this example, it can be useful if you then want to manipulate the new row further.

```
function addRow(tableId, cells){
var tableElem = document.getElementById(tableId);
var newRow = tableElem.insertRow(tableElem.rows.length);
var newCell;
for (var i=0; i < cells.length; i++) {
    newCell = newRow.insertCell(newRow.cells.length);
    newCell.innerHTML = cells[i];
}
return newRow;
}
```

The form also contains a "Delete Row" button that, when clicked, passes the `id` of the table (`tblPeople`) and the number entered by the user in the `RowNum` text field.

```
deleteRow('tblPeople', btnDelete.form.RowNum.value)
```

The `deleteRow()` function checks to see if the row specified exists and is not the first row (row 0, which is the header row). If both conditions are true, it deletes the row. Otherwise, it alerts "Failed".

```
function deleteRow(tableId, rowNumber){
var tableElem = document.getElementById(tableId);
if (rowNumber > 0 && rowNumber < tableElem.rows.length) {
    tableElem.deleteRow(rowNumber);
} else {
    alert("Failed");
}
}
```

## Exercise 19 Tracking Results in the Math Quiz

15 to 25 minutes

In this exercise, you will dynamically create a table that shows the user how she is doing on the Math Quiz. The screenshot below shows how the result will look:

Category:	Subtraction ▼
Question:	--Please Choose-- ▼
Sorry. The correct answer is 8.	

Question	Your answer	Correct answer
5 + 4	9	9
9 + 3	12	12
7 + 12	19	19
13 + 24	36	37
5 - 1	4	4
12 - 4	7	8

1. Open [CSSObjectModel/Exercises/MathQuizTable.html](#) for editing.
2. Change the `msg()` function so that it takes a second argument: `color`, and uses it to change the text of the "msg" div to the passed-in color.
3. Change all calls to `msg()` to pass in a `color` as well as the text: green if the answer is correct and red if it is not.
4. Option objects have a `text` property that holds the displayed text of the option. Add a `getQuestion()` function that returns the text of the selected question.
5. Notice that there is a table at the bottom of the body of the page with an id of `tblResults`.
6. Modify the `checkAnswer()` function so that it adds a new row to the results table showing the question, user's answer, and correct answer.
7. Test your solution in a browser.

### \*Challenge

Modify the code so that the new row's background color is #00ff00 if the answer is correct and #ff9999 if it is not. *Hint:* the `addRow()` function returns the newly added row.



**Exercise Solution****CSSObjectModel/Solutions/MathQuizTable.html**

```
-----Lines 1 through 64 Omitted-----
65.
66.
67.     function addRow(tableId, cells) {
68.         var tableElem = document.getElementById(tableId);
69.         var newRow = tableElem.insertRow(tableElem.rows.length);
70.         var newCell;
71.         for (var i = 0; i < cells.length; i++) {
72.             newCell = newRow.insertCell(newRow.cells.length);
73.             newCell.innerHTML = cells[i];
74.         }
75.         return newRow;
76.     }
77.
78.     function checkAnswer() {
79.         var userAnswer = document.getElementById("answer").value;
80.         var correctAnswer = getAnswer();
81.         var question = getQuestion();
82.         var arrCells = [question, userAnswer, correctAnswer];
83.         addRow("tblResults", arrCells);
84.
85.         if (userAnswer === correctAnswer) {
86.             msg("Right! The answer is " + correctAnswer + ".","green");
87.         } else {
88.             msg("Sorry. The correct answer is " + correctAnswer + ".","red");
89.         }
90.         removeOption();
91.         questionChanged();
92.     }
93.
94.     function getAnswer() {
95.         var i = document.Quiz.question.selectedIndex;
96.         var answer = document.Quiz.question[i].value;
97.         return answer;
98.     }
99.
100.    function getQuestion(){
101.        var i = document.Quiz.question.selectedIndex;
102.        var question = document.Quiz.question[i].text;
```

```
103.     return question;
104. }
-----Lines 105 through 151 Omitted-----
152.
153. function msg(text, color) {
154.     document.getElementById("msg").innerHTML = text;
155.     document.getElementById("msg").style.color = color;
156. }
157. </script>
158. </head>
159. <body>
-----Lines 160 through 199 Omitted-----
200. <table id="tblResults">
201. <tr>
202.   <th>Question</th>
203.   <th>Your answer</th>
204.   <th>Correct answer</th>
205. </tr>
206. </table>
207. </body>
208. </html>
```

**Challenge Solution****CSSObjectModel/Solutions/MathQuizTable-challenge.html**

```
-----Lines 1 through 76 Omitted-----
77.  function checkAnswer() {
78.      var userAnswer = document.getElementById("answer").value;
79.      var correctAnswer = getAnswer();
80.      var question = getQuestion();
81.      var arrCells = [question, userAnswer, correctAnswer];
82.      var row = addRow("tblResults", arrCells);
83.
84.      if (userAnswer === correctAnswer) {
85.          msg("Right! The answer is " + correctAnswer + ".", "green");
86.          row.style.backgroundColor="#00ff00";
87.      } else {
88.          msg("Sorry. The correct answer is " + correctAnswer + ".", "red");
89.          row.style.backgroundColor="#ff9999";
90.      }
91.      removeOption();
92.      questionChanged();
93.  }
-----Lines 94 through 209 Omitted-----
```

## 9.4 Dynamically Changing Dimensions

The dimensions of an object can be changed by modifying the `width` and `height` properties of the element's `style` property. The following example demonstrates this:

**Code Sample****CSSObjectModel/Demos/Dimensions.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Dimensions</title>
6.    <link href="dimensions.css" rel="stylesheet">
7.    <script>
8.        function grow(elem){
9.            var curWidth = parseInt(getComputedStyle(elem).getPropertyVal >>
                >>> ue("width"));
10.           var curHeight = parseInt(getComputedStyle(elem).getPropertyVal >>
                >>> ue("height"));
11.           elem.style.width = (curWidth * 1.5) + 'px';
12.           elem.style.height = (curHeight * 1.5) + 'px';
13.           showDimensions(elem);
14.       }
15.
16.       function shrink(elem){
17.           var curWidth = parseInt(getComputedStyle(elem).getPropertyVal >>
                >>> ue("width"));
18.           var curHeight = parseInt(getComputedStyle(elem).getPropertyVal >>
                >>> ue("height"));
19.           elem.style.width = (curWidth / 1.5) + 'px';
20.           elem.style.height = (curHeight / 1.5) + 'px';
21.           showDimensions(elem);
22.       }
23.
24.       function showDimensions(elem) {
25.           elem.innerHTML = "w: " + getComputedStyle(elem).getPropertyVal >>
                >>> ue("width") + "<br>h: " + getComputedStyle(elem).getProperty >>
                >>> Value("height");
26.       }
27.
28.       window.onload = function() {
29.           var block = document.getElementById("divBlock");
30.           block.addEventListener('mouseover', function() {
31.               grow(block);
32.           })
33.           block.addEventListener('mouseout', function() {
34.               shrink(block);
```



```
35.     })
36.     showDimensions(block);
37.   };
38. </script>
39. </head>
40. <body>
41.   <div id="divBlock"></div>
42. </body>
43. </html>
```

### Code Explanation

When the page loads, we begin observing mouseover and mouseout events on the block div. We call `grow()` on mouse overs and `shrink()` on mouse outs.

The `grow()` function uses `parseInt()` to cut off the units (e.g., px) from the value of the width and height of the div and assigns the resulting integers to variables: `curWidth` and `curHeight`. It then modifies the width and height properties of the element by multiplying the current values by 1.5.

The `shrink()` function does the same thing, but it divides by 1.5 instead of multiplying.

## Creating a Timed Slider

The example below shows how a timed slider can be created by dynamically changing an element's dimensions:

**Code Sample****CSSObjectModel/Demos/Slider.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Slider</title>
6.    <link href="slider.css" rel="stylesheet">
7.    <script>
8.        var timer, timesUp;
9.        function resetTimer() {
10.            var slider = document.getElementById("divSlider");
11.            timesUp = true;
12.            slider.style.width = "0px";
13.            clearInterval(timer);
14.        }
15.
16.        function decrementTimer() {
17.            var slider = document.getElementById("divSlider");
18.            var curWidth = parseInt(getComputedStyle(slider).getPropertyVal
    >>> ue("width"));
19.            timesUp = false;
20.            if (curWidth < 200) {
21.                slider.style.width = curWidth + 2 + "px";
22.            } else {
23.                alert("Time's up!");
24.                resetTimer();
25.            }
26.        }
27.
28.        window.onload = function() {
29.            var btnStart = document.getElementById("btnStart");
30.            resetTimer();
31.            btnStart.addEventListener('click', function() {
32.                timer = setInterval(decrementTimer, 100);
33.            });
34.        };
35.    </script>
36. </head>
37. <body>
38.    <div id="divSliderBG">
```

```
39.   <div id="divSlider"></div>
40.   </div>
41.   <button id="btnStart">Start Timer</button>
42.   </body>
43.   </html>
```

## 9.5 Positioning Elements Dynamically

The position of an object can be changed by modifying the `left` and `top` properties of the element's `style` property. The following example demonstrates this:

**Code Sample****CSSObjectModel/Demos/Position.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Position</title>
6.    <link href="position.css" rel="stylesheet">
7.    <script>
8.        function moveH(elem, distance){
9.            var curLeft = parseInt(getComputedStyle(elem).getPropertyVal >>
                >>> ue("left"));
10.           elem.style.left = (curLeft + distance) + "px";
11.        }
12.
13.        function moveV(elem, distance){
14.            var curTop = parseInt(getComputedStyle(elem).getPropertyValue("top"));
                >>>
15.            elem.style.top = (curTop + distance) + "px";
16.        }
17.
18.        window.onload = function() {
19.            var btnLeft = document.getElementById("btnLeft");
20.            var btnRight = document.getElementById("btnRight");
21.            var btnUp = document.getElementById("btnUp");
22.            var btnDown = document.getElementById("btnDown");
23.            var block = document.getElementById("divBlock");
24.
25.            btnLeft.addEventListener('click', function() {
26.                moveH(block,-10);
27.            });
28.            btnRight.addEventListener('click', function() {
29.                moveH(block,10);
30.            });
31.            btnUp.addEventListener('click', function() {
32.                moveV(block,-10);
33.            });
34.            btnDown.addEventListener('click', function() {
35.                moveV(block,10);
36.            });
37.        };
```

```
38. </script>
39. </head>
40. <body>
41. <div id="field">
42.   <button id="btnLeft">Left</button>
43.   <button id="btnRight">Right</button>
44.   <button id="btnUp">Up</button>
45.   <button id="btnDown">Down</button>
46.   <div id="divBlock"></div>
47. </div>
48. </body>
49. </html>
```

### Code Explanation

When the page loads, we begin observing `click` events on the buttons to call `moveH()` and `moveV()`.

The `moveH()` function uses `parseInt()` to cut off the units (e.g., `px`) from the value of the `left` property and assigns the resulting integers to the `curLeft` variable. It then adds the passed-in `distance` to the `left` property.

The `moveV()` function does the same thing, but it modifies the `top` property rather than the `left` property.

### Creating a Different Timed Slider

The example below shows how a different type of timed slider can be created by dynamically changing an element's position:

**Code Sample****CSSObjectModel/Demos/Slider2.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Slider</title>
6.    <link href="slider2.css" rel="stylesheet">
7.    <script>
8.        var timer, timesUp;
9.        function resetTimer(){
10.            var slider = document.getElementById("divSlider");
11.            timesUp = true;
12.            slider.style.left = "1px";
13.            clearInterval(timer);
14.        }
15.
16.        function decrementTimer(){
17.            var slider = document.getElementById("divSlider");
18.            var curLeft = parseInt(getComputedStyle(slider).getPropertyVal >>
                >>> ue("left"));
19.            var curWidth = parseInt(getComputedStyle(slider).getPropertyVal >>
                >>> ue("width"));
20.            timesUp = false;
21.            if (curLeft < 198 - curWidth) {
22.                slider.style.left = curLeft + 2 + "px";
23.            } else {
24.                alert("Time's up!");
25.                resetTimer();
26.            }
27.        }
28.
29.        window.onload = function() {
30.            var btnStart = document.getElementById("btnStart");
31.            resetTimer();
32.            btnStart.addEventListener('click', function() {
33.                timer = setInterval(decrementTimer, 100);
34.            });
35.        };
36.    </script>
37.    </head>
```

```
38. <body>
39. <div id="divSliderBG">
40.   <div id="divSlider"></div>
41. </div>
42. <button id="btnStart">Start Timer</button>
43. </body>
44. </html>
```

## Exercise 20 Changing the Math Quiz Timer to a Slider

*15 to 25 minutes*

In this exercise, you will modify the Math Quiz so that the timer is a slider rather than a count down. The result will look like this:

Category:	Addition ▾	
Question:	5 + 4 ▾	
Answer:	<input type="text"/>	<input type="button" value="Check Answer"/>
Timer:	<div style="background-color: blue; width: 100%; height: 20px; position: relative;"><div style="background-color: red; width: 50%; height: 100%; position: absolute; left: 0;"></div></div>	
<b>Good luck!</b>		

---

Question	Your answer	Correct answer
----------	-------------	----------------

1. Open [CSSObjectModel/Exercises/MathQuizSlider.html](#) for editing.
2. Notice that the timer on the page has been changed from an `input` element to two `div`s.

```
<tr id="timerRow">
<td>Timer:</td>
<td>
  <div id="divSliderBG">
    <div id="divSlider"></div>
  </div>
</td>
</tr>
```

3. Also, all references to the `timeLeft` input have been removed and the code within the `decrementTimer()` function has been commented out.
4. Rewrite the `decrementTimer()` function to use a slider like the one shown in the screenshot above.
5. Test your solution in a browser.



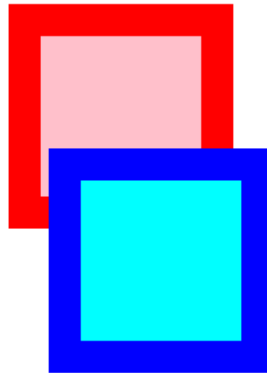


**Exercise Solution****CSSObjectModel/Solutions/MathQuizSlider.html**

```
-----Lines 1 through 51 Omitted-----
52.
53.  function decrementTimer() {
54.      var slider = document.getElementById("divSlider");
55.      var curLeft = parseInt(getComputedStyle(slider).getPropertyVal >>
        >>> ue("left"));
56.      var curWidth = parseInt(getComputedStyle(slider).getPropertyVal >>
        >>> ue("width"));
57.      if (curLeft < 198 - curWidth) {
58.          slider.style.left = curLeft + 2 + "px";
59.      } else {
60.          resetTimer(timePerQuestion);
61.          msg("Time's up!  The answer is " + getAnswer() + ".", "red");
62.          removeOption();
63.      }
64.  }
-----Lines 65 through 213 Omitted-----
```

## 9.6 Changing the Z-Index

The `z-index` value of an element indicates its relative position in the "stack" of elements on the page. Elements with higher `z-index` values sit on top of elements with lower values. You can think of a stack of papers thrown on a table. The ones at the top have a higher `z-index` than the ones on the bottom. In the screenshot below, the blue box has a higher `z-index` than the red box:



The `z-index` of an object can be changed by modifying the `zIndex` property of the element's `style` property. The following example demonstrates this:

### Code Sample

#### CSSObjectModel/Demos/Zindex.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>zIndex</title>
6.    <link href="zindex.css" rel="stylesheet">
7.    <script>
8.        var z = 0;
9.        function changeZ(elem){
10.            z += 10;
11.            elem.style.zIndex = z;
12.            elem.innerHTML = "z: " + z;
13.        }
14.
15.    window.onload = function() {
16.        var divRed = document.getElementById("divRed");
17.        var divBlue = document.getElementById("divBlue");
18.        divRed.addEventListener('click', function() {
19.            changeZ(divRed);
20.        });
21.        divBlue.addEventListener('click', function() {
22.            changeZ(divBlue);
23.        });
24.    };
25.    </script>
26.    </head>
27.    <body>
28.        <div id="divRed"></div>
29.        <div id="divBlue"></div>
30.    </body>
31.    </html>
```

### Code Explanation

The variable `z` always holds the highest z-index. The function `changeZ ( )` simply adds 10 to `z` and assigns the resulting value to the `zIndex` property of the passed in object.

```
function changeZ(elem){  
  z += 10;  
  elem.style.zIndex = z;  
  elem.innerHTML = "z: " + z;  
}
```

## 9.7 The CSS Object Model

Behind all of our work manipulating and reading CSS styles with JavaScript lies the CSS Object Model API, a complete (and ever changing) specification from the W3C that defines how to control CSS with JavaScript. The complete specification is at <https://www.w3.org/TR/cssom/>; you may also find useful the Mozilla Developer Networ's CSS [Object Model documentation \(https://developer.mozilla.org/en-US/docs/Web/API/CSS\\_Object\\_Model\)](https://developer.mozilla.org/en-US/docs/Web/API/CSS_Object_Model).

While delving too deeply into the API itself is beyond the scope of this course, we will take a quick look at a way in which we can check to see if a user's browser offers support for a given CSS property and value. Consider the following example:

**Code Sample****CSSObjectModel/Demos/supports.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Supports</title>
6.    <script>
7.        window.onload = function() {
8.            document.getElementById("checkit").addEventListener("click", function()
9.                >>> {
10.                var p = document.getElementById("p").value;
11.                var v = document.getElementById("v").value;
12.                if (CSS.supports(p, v)) {
13.                    alert('Your browser DOES support property ' + p + ' with value ' +
14.                        >>> v);
15.                } else {
16.                    alert('Your browser does NOT support property ' + p + ' with value
17.                        >>> ' + v);
18.                }
19.            return false;
20.        });
21.    }
22.    </script>
23.    </head>
24.    <body>
25.    <h1>Supports</h1>
26.    <form action="#">
27.        <input type="text" id="p" placeholder="Property"> <input type="text"
28.            >>> id="v" placeholder="Value">
29.        <button id="checkit">Check</button>
30.    </form>
31.    </body>
32.    </html>
```

**Code Explanation**

The page presents two textfields in which the user can enter a CSS property and a value for that property. Clicking the "Check" button fires an event handler which includes the code `CSS.supports(p, v)`; an alert indicates whether the current browser does or does not support the given property/value.

As of this writing, you can try property `hyphens` and value `manual`: the current version of Chrome does not support that property but the current version of Firefox does support it.

## **9.8 Conclusion**

In this lesson, you have learned

- how to dynamically modify the content of an HTML page and to dynamically modify CSS styles of HTML elements.





## 10. Images, Windows and Timers

**In this lesson, you will learn...**

1. To preload images.
2. To create a slide show.
3. To open and control a new window.
4. To create timers.

### 10.1 Preloading Images

When using JavaScript to dynamically load an image, perhaps in response to some user interaction, preloading the image (the image not initially showing on the screen) can eliminate a delay in the image appearing for the user. Images can be preloaded by creating an `Image` object with JavaScript and assigning a value to the `src` of that `Image`. A sample is shown below:

### Code Sample

#### ImagesWindowsTimers/Demos/PreloadingImages.html

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>Preloading Images</title>
6.  <link href="style.css" rel="stylesheet">
7.  <script>
8.
9.      var imagePaths = [];
10.     imagePaths[0] = "Images/Hulk.jpg";
11.     imagePaths[1] = "Images/Batman.jpg";
12.
13.     var imageCache = [];
14.
15.     for (var i=0; i<imagePaths.length; i++) {
16.         imageCache[i] = new Image();
17.         imageCache[i].src = imagePaths[i];
18.     }
19.
20.     function imageRollover(img, imgSrc) {
21.         img.src = imgSrc;
22.     }
23. </script>
24. </head>
25. <body>
26. <div style="text-align:center;">
27. <h1>Simple Image Rollover Function</h1>
28. 
31. 
34. <p>Who are you calling simple?</p>
35. </div>
36. </body>
37. </html>
```

## Code Explanation

Notice that the code is not in a function. It starts working immediately as follows:

1. An array called `imagePaths` is created to hold the paths to the images that need to be preloaded.

```
var imagePaths = [];
```

2. An array element is added for each image to be preloaded.

```
imagePaths[0] = "Images/Hulk.jpg";  
imagePaths[1] = "Images/Batman.jpg";
```

3. An array called `imageCache` is created to hold the `Image` objects that will hold the preloaded images.

```
var imageCache = [];
```

4. A `for` loop is used to create an `Image` object and load in an image for each image path in `imagePaths`.

```
for (var i=0; i<imagePaths.length; i++) {  
    imageCache[i] = new Image();  
    imageCache[i].src = imagePaths[i];  
}
```

## 10.2 Windows

These days, popup windows are generally frowned upon; however, they can be useful in some cases. We'll look at a couple of examples, but first, let's see how to open a new window.

### Syntax

```
var newWin = window.open(URL,name,features,replace);
```

All four parameters are options:

1. `URL` - the URL of the page to load. If it is left blank, a blank window is open and can be written to with `newWin.document.write()`.
2. `name` - the `target` attribute of the window. This can be used to reuse an existing window if it is open.

3. `features` - a comma-delimited list of window features. Some of the most common are:
  - A. `height` - the height of the window
  - B. `width` - the width of the window
  - C. `left` - the left position of the window
  - D. `top` - the top position of the window
  - E. `location` - whether or not to include the location bar
  - F. `menubar` - whether or not to include the menubar
  - G. `resizable` - whether or not the window should be resizable
  - H. `scrollbars` - whether or not to include scrollbars
  - I. `status` - whether or not to include the status bar
  - J. `toolbar` - whether or not to include the toolbar
4. `replace` - true or false. If set to true, the new page replaces the current page (if there is one) in the browser window's history.

The `height`, `width`, `left`, and `top` features should be set in pixels.

The `location`, `menubar`, `resizable`, `scrollbars`, `status`, and `toolbar` features are boolean values: "true" or "false", "yes" or "no", or "1" or "0" will work.

The [HTML5 specification](#)<sup>10</sup> advises browsers to ignore the `features` arguments completely, and some modern browsers do choose to ignore all but the size and positioning features.

The example below shows how to open a new window:

---

10. See <http://www.w3.org/TR/html5/browsers.html#apis-for-creating-and-navigating-browsing-contexts-by-name>.

**Code Sample****ImagesWindowsTimers/Demos/window.html**

```

1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>New Window</title>
6.    <script>
7.    var eula;
8.
9.    function openWin() {
10.    eula = window.open("eula.html","eu »»
    >>> la","height=200,width=300,left=100,top=100");
11.    eula.focus();
12.    }
13.
14.    function eulaChecked() {
15.    if (document.getElementById("confirmEula").checked && typeof eula ==
    >>> "undefined") {
16.    alert("Come on now. Don't you think you should read the EULA first?");
    >>>
17.    document.getElementById("confirmEula").checked = false;
18.    openWin();
19.    } else if (document.getElementById("confirmEula").checked && !eu »»
    >>> la.closed) {
20.    eula.close();
21.    } else if (!document.getElementById("confirmEula").checked) {
22.    openWin();
23.    }
24.    }
25.    </script>
26.    </head>
27.    <body>
28.    <form action="process.xyz">
29.    <input type="checkbox" id="confirmEula" name="confirmEula"
    >>> onclick="eulaChecked()"> Check to confirm that you agree with
    >>> our <a href="eula.html" onclick="openWin(); return
    >>> false;">user agreement</a>.
30.    </form>
31.    </body>
32.    </html>

```

## Code Explanation

Things to notice:

1. We make `eula` a global variable so that we can access the window object from within multiple functions.
2. In the `openWin()` function, we call `eula.focus()`; after opening the window. That's to bring the window to the foreground if it's already been opened.
3. In the `eulaChecked()` function:
  - A. We first check to see if the `confirmEula` check box has been checked before the EULA window has been opened (i.e., before a window object has been assigned to the `eula` variable). In this case, we scold the user and pop up the EULA window.
  - B. We then check to see if the `confirmEula` check box has been checked and the EULA window is left open, in which case we close the EULA window via the window object's `close()` method.
  - C. Finally, if the `confirmEula` check box has been unchecked, we pop the EULA window back open.

## 10.3 Timers

Timers are started and stopped with the following four methods of the windows object:

1. `setTimeout(code_to_execute,wait_time_in_milliseconds)`
2. `clearTimeout(timer)`
3. `setInterval(code_to_execute,interval_in_milliseconds)`
4. `clearInterval(interval)`

Let's take a look at how `setTimeout()` and `clearTimeout()` work first:

## Code Sample

### ImagesWindowsTimers/Demos/Timer.html

```

1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>Timer</title>
6.  <script>
7.      var timer;
8.      function changeBg(color) {
9.          timer = setTimeout(function() { document.body.style.backgroundCol >>
              >>> or=color; }, 1000);
10.     }
11.
12.     function stopTimer() {
13.         clearTimeout(timer);
14.     }
15. </script>
16. </head>
17. <body>
18. <button onclick="changeBg('red')">Change Background to Red</button>
19. <button onclick="changeBg('white')">Change Background to White</button>
    >>>
20. <button onclick="stopTimer()">Wait! Don't do it!</button>
21. </body>
22. </html>

```

## Code Explanation

Things to notice:

1. We make `timer` a global variable so that we can access the timer object from within multiple functions.
2. In the `changeBg( )` function, we create the timer. Note that we need to place `document.body.style.backgroundColor=color` in a function; otherwise, the code will execute immediately. In this case, we use an anonymous function, which works just like a regular function. Note that because the function

is nested within the `changeBg ( )` function, it has access to the local variable `color`.

3. The `stopTimer ( )` function simply clears the `timer` using `clearTimeout ( )`.

The `setInterval ( )` and `clearInterval ( )` methods work the same way. The only difference is that the code gets executed repeatedly until the interval is cleared.



## Exercise 21 Popup Timed Slide Show

*15 to 25 minutes*

In this exercise, you will create a popup slideshow that runs by itself.

1. Open [ImagesWindowsTimers/Exercises/Timed-SlideShow.html](#) in your editor.
2. Add code to the `startShow()` function to pop up [popup-show.html](#) in a new 200px by 300px window. Make sure it comes to the foreground.
3. Open [ImagesWindowsTimers/Exercises/popup-show.html](#) in your editor.
4. Notice the `play()` function is called when the window loads. Add code to the `play()` function so that it changes the slide every second (1000 milliseconds).
5. Add code to the `stop()` function to stop the slideshow.

### Exercise Solution

#### ImagesWindowsTimers/Solutions/Timed-SlideShow.html

```
-----Lines 1 through 5 Omitted-----
6.  <script>
7.  var showWin;
8.
9.  function startShow() {
10.   showWin = window.open("popup-
    >>> show.html","showWin","height=300,width=200,left=300,top=100");
    >>>
11.   showWin.focus();
12.  }
13. </script>
    -----Lines 14 through 18 Omitted-----
```

### Exercise Solution

#### ImagesWindowsTimers/Solutions/popup-show.html

```
-----Lines 1 through 32 Omitted-----
33. var showInterval;
34. function play() {
35.   showInterval = setInterval(function() { changeSlide(1); },1000);
36. }
37.
38. function stop() {
39.   clearInterval(showInterval);
40. }
41. </script>
    -----Lines 42 through 53 Omitted-----
```

## 10.4 Conclusion

In this lesson, you have learned how JavaScript can be used to manipulate HTML images to create slide shows. You also learned how to open and control new windows and how to start and stop timers and intervals with JavaScript.



## 11. Debugging and Testing with Chrome

**In this lesson, you will learn...**

1. How to use Google's DevTools to debug JavaScript

Chrome's Developer Tools are a valuable resource for writing and debugging JavaScript code.

### 11.1 Chrome DevTools

Google's Chrome browser offers a wide range of tools useful to us as front-end JavaScript developers. By right-clicking and choosing "Inspect" from the context menu (or by choosing "Developer Tools" from the "Tools" option under the Chrome menu at the upper right of the browser), we open a set of panels - positioned by default at the bottom of the browser - by which we can inspect markup code, access the JavaScript console, view network activity, set breakpoints, and find other useful resources as we create and debug our JavaScript code.

In addition to the existing "out of the box" tools, Google Chrome offers an API by which you can write your own tools to meet a specific need, or use an extension built by someone else.

### 11.2 The Panels

Before we take a look at the most useful of the DevTools panels, let's look at a simple page with some JavaScript we can use those tools to review and debug:

### Code Sample

#### DebuggingtestingChrome/Demos/debugging.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>JavaScript Loops</title>
6.    <script>
7.        var index;
8.        var beatles = [];
9.        beatles["Guitar1"] = "John";
10.       beatles["Bass"] = "Paul";
11.       beatles["Guitar2"] = "George";
12.       beatles["Drums"] = "Ringo";
13.
14.
15.
16.    window.onload = function() {
17.        for (var prop in beatles) {
18.            var beatle = document.createElement("li");
19.            var textnode = document.createTextNode(prop + " : " + beatles[prop]);
20.                >>>
21.            beatle.appendChild(textnode);
22.            document.getElementById("beatles").appendChild(beatle);
23.        }
24.        document.getElementById("redtext").addEventListener("click", function()
25.            >>> {
26.            document.getElementById("beatles").style.color = 'red';
27.            return false;
28.        });
29.        document.getElementById("blacktext").addEventListener("click", func »»
30.            >>> tion() {
31.            document.getElementById("beatles").style.color = 'black';
32.            return false;
33.        });
34.        document.getElementById("italictext").addEventListener("click",
35.            >>> function() {
36.            document.getElementById("beatles").style.fontStyle = 'italic';
37.            return false;
38.        });
39.    }
```

```

36.     document.getElementById("normaltext").addEventListener("click",
    >>> function() {
37.         document.getElementById("beatles").style.fontStyle = 'normal';
38.         return false;
39.     });
40. }
41. </script>
42. </head>
43. <body>
44. <h1>JavaScript Debugging</h1>
45. <ol id="beatles">
46. </ol>
47. <div class="buttons">
48.     <ul>
49.         <li><a href="#" id="redtext">Set Text Red</a></li>
50.         <li><a href="#" id="blacktext">Set Text Black</a></li>
51.         <li><a href="#" id="italictext">Set Text Italic</a></li>
52.         <li><a href="#" id="normaltext">Set Text Plain</a></li>
53.     </ul>
54. </div>
55. </body>
56. </html>

```

## Code Explanation

The HTML markup for our page has an empty ordered list with id `beatles` and a set of four links ("Set Text Red", "Set Text Black", "Set Text Italic", and "Set Text Plain"). In the head of the document we include JavaScript that:

- Creates an array of members of the Beatles, where the index is the instrument and the value the name of each member;
- After the DOM loads, we use a `for...in` loop to append each Beatle array-element to the unordered list;
- Also after the DOM loads, we create event listeners for each of the four links, changing the text color or style when the user clicks each link.

Let's use this relatively-simple code to see how the various resources Google DevTools gives us might be of use.

## The Elements Panel

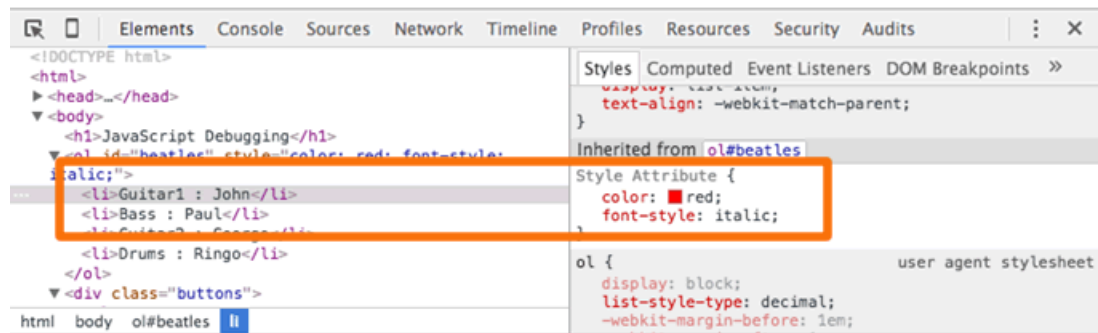
The "Elements" panel is the default panel which comes up when one right-clicks on a DOM element on the page. While not strictly JavaScript, this panel is great for finding the id of an element and looking at its associated CSS styling. Information

about HTML elements and their CSS styling are dynamic in this panel: changes we make via JavaScript - changing the text color, say, as we do in our sample code, and adding the Beatles order-list elements - are reflected in real time in the "Elements" panel, just as if the original HTML of the page had included what we changed via JavaScript.

For our example, we might use the "Elements" panel for a quick check of the `id` of the "Set Text Red" link ("what did I set that `id` to?") or to verify that the CSS style of the Beatles members list is set, as we intend, with `color:red`:

## JavaScript Debugging

1. *Guitar1 : John*
  2. *Bass : Paul*
  3. *Guitar2 : George*
  4. *Drums : Ringo*
- [Set Text Red](#)
  - [Set Text Black](#)
  - [Set Text Italic](#)
  - [Set Text Plain](#)



## The Console Panel

The "Console" panel gives us valuable JavaScript error information - a message from the browser that we have, sadly, written some buggy code. We might, for instance, have misspelled a JavaScript keyword, tried to access the property of a null element (in trying to get an element with an incorrect `id`, say), or attempted to access an element of an array that doesn't exist.

Perhaps even more usefully, we can write to this console as a simple debugging tactic. The following code would write the value of an array to the console:



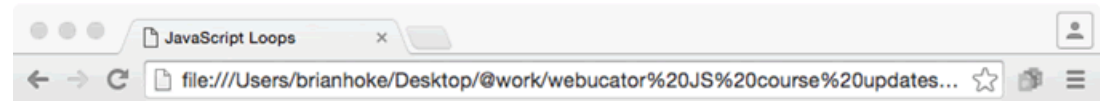
```
var seconditem = shoppingcart[2];  
console.log(seconditem);
```

If `shoppingcart` were an array - the items a user might have added to their shopping cart, for instance - then `seconditem` would represent the third item in the cart (the third item in the array, since JavaScript indexes arrays starting at 0). The code `console.log(seconditem)` writes not to the screen, but rather to the console, which we can view with the "Console" panel. A great feature here is that we can view structured data - arrays and collections - in a useful manner, along with simple data like string and arrays.

For our example, if we were to add the following code after we create and populate the Beatles array:

```
console.log(beatles);
```

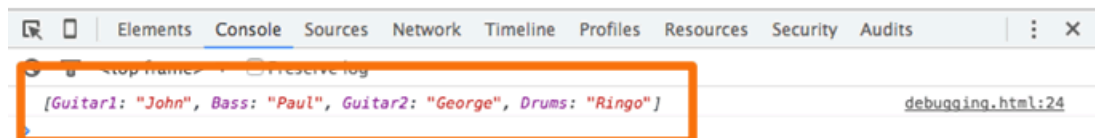
Then we would be able to inspect the array:



## JavaScript Debugging

1. Guitar1 : John
2. Bass : Paul
3. Guitar2 : George
4. Drums : Ringo

- [Set Text Red](#)
- [Set Text Black](#)
- [Set Text Italic](#)
- [Set Text Plain](#)



Before reviewing more DevTools panels, let's have you try out the tools we've reviewed so far with an exercise:

## Exercise 22 Using the Chrome DevTools "Elements" and "Console" Panels

*5 to 15 minutes*

1. Open [DebuggingTestingChrome/Exercises/elements-console.html](#) in a browser and in a code editor to review the code.
2. This page - a modified version of the example code we looked at earlier - is designed to set the text of the "George" list item to red when the user clicks the "Set George Red" link; however the code does not work as intended.
3. Note that we use `beattle.setAttribute("id", prop);` to set an `id` for each list item as we iterate over the array to build our ordered list.
4. Use the Chrome DevTools "Elements" and "Console" panels to help you diagnose and fix the error.



### Exercise Solution

#### DebuggingtestingChrome/Solutions/elements-console.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>JavaScript Loops</title>
6.    <script>
7.        var index;
8.        var beatles = [];
9.        beatles["Guitar1"] = "John";
10.       beatles["Bass"] = "Paul";
11.       beatles["Guitar2"] = "George";
12.       beatles["Drums"] = "Ringo";
13.
14.       window.onload = function() {
15.           for (var prop in beatles) {
16.               var beatle = document.createElement("li");
17.               beatle.setAttribute("id", prop);
18.               var textnode = document.createTextNode(prop + " : " + beatles[prop]);
19.               beatle.appendChild(textnode);
20.               document.getElementById("beatles").appendChild(beatle);
21.           }
22.
23.           console.log(beatles);
24.
25.           document.getElementById("redgeorge").addEventListener("click", function() {
26.               document.getElementById("Guitar2").style.color='red';
27.               return false;
28.           });
29.       }
30.    </script>
31.    </head>
32.    <body>
33.    <h1>JavaScript Debugging</h1>
34.    <ol id="beatles">
35.    </ol>
36.    <div class="buttons">
37.    <ul>
```

```

38.     <li><a href="#" id="redgeorge">Set George Red</a></li>
39.   </ul>
40. </div>
41. </body>
42. </html>

```

### Code Explanation

The issue here is that the event handler for a user click on the "Set George Red" link references the wrong id: instead of setting the element with id `Guitar2` to be red, we (wrongly) set the element with id `Guitar1` to be red.

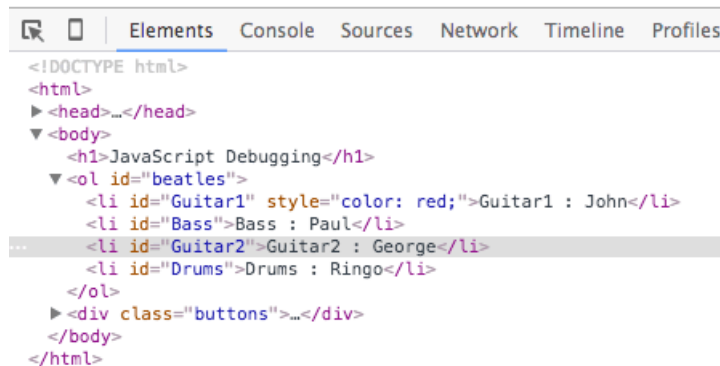
We can use the Chrome DevTools in a couple of ways to help us in our task of finding the bug in the code. First, we might have used the "Elements" console to quickly scan the ids of the list items:

```

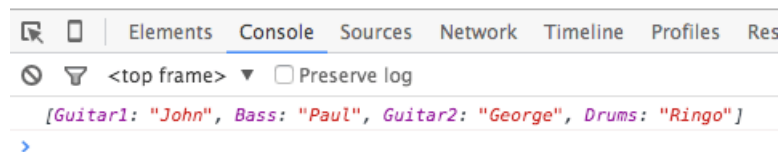
1. Guitar1 : John
2. Bass : Paul
3. Guitar2 : George
4. Drums : Ringo

```

- [Set George Red](#)



and seeing that the "George" list item has id `Guitar2`. We might also have used `console.log(beatles)` to list the elements of our array, to compare the indices of our array against the ids we see in the "Elements" panel:



Of course, we don't *really* need the DevTools here - we could have gone through the source of our code, rather than reviewing it in the browser and inspecting via

DevTools - to find and fix the bug. But because DevTools makes it so easy to inspect DOM elements and view collections like arrays in the console, DevTools is often the preferred method of debugging, especially as your JavaScript code gets more complicated.

## 11.3 The Sources Panel

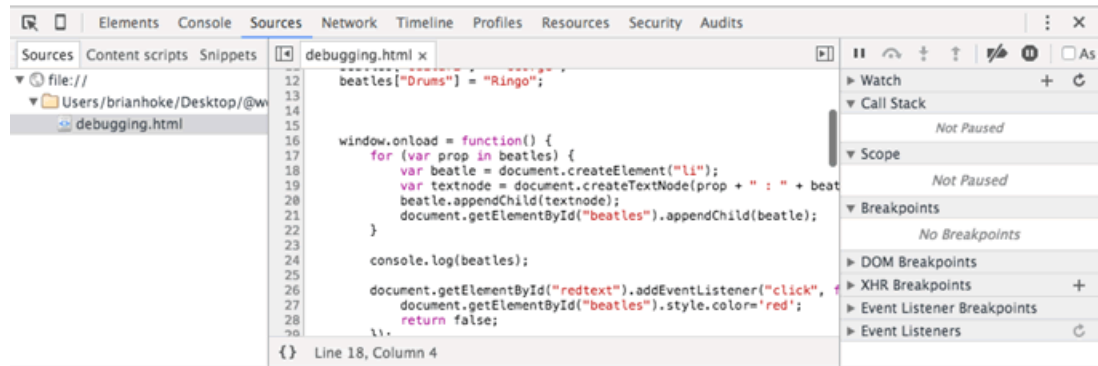
The "Sources" panel of Chrome's DevTools gives us the opportunity to add breakpoints ("pauses" in our code, where the browser will let us stop to review how things are going), to step through the execution of our code, and to watch the values of any given variable or expression. Tracking down that bug (hopefully just one!) in your thousands of lines of JavaScript code gets much easier if you can have the browser let you check the values of some variables each time you run through a loop and check out the current state of an array.

Let's look again at how we might use the tools from the "Sources" panel on our example page [DebuggingTestingChrome/Demos/debugging.html](http://DebuggingTestingChrome/Demos/debugging.html). Right clicking on any part of the screen, choosing the "Inspect" from the popup menu, selecting

sources, clicking on "debugging.html" from the left part of the panel gives us the following:

### JavaScript Debugging

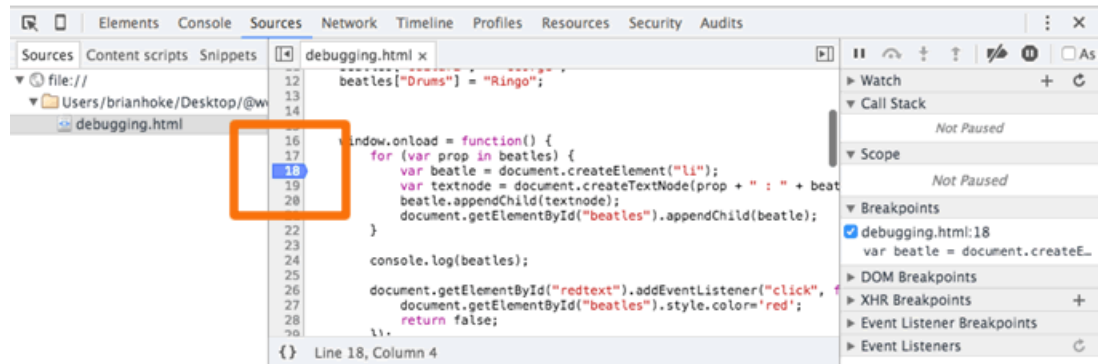
1. Guitar1 : John
2. Bass : Paul
3. Guitar2 : George
4. Drums : Ringo



We can click on the line number of any line of JavaScript code; in the example below we've clicked on line 18, in which we create a list-item node for a different Beatle on each iteration of our `for` loop:

### JavaScript Debugging

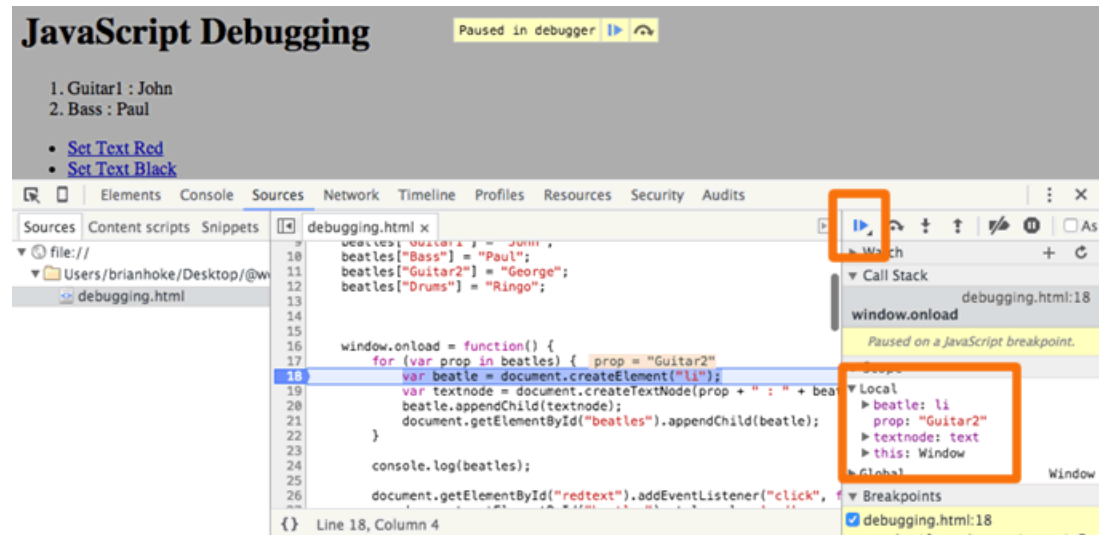
1. Guitar1 : John
2. Bass : Paul
3. Guitar2 : George
4. Drums : Ringo



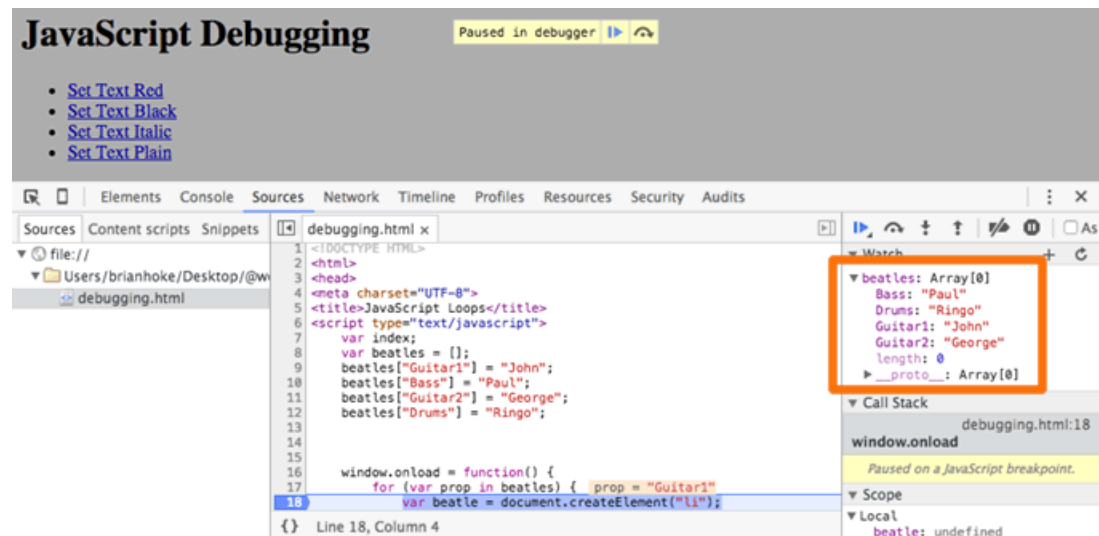
If we refresh the page, we see that Chrome pauses the execution of our JavaScript code each time we reach line 18 - pausing, in this case, in the middle of each loop. We can iterate through the loop using the "play" icon (highlighted in orange in the



screenshot below) and we can see values for the various variables on the right side of the panel:



We can use the "Watch" (in the upper right) to view the value of any variable or expression; in the example below, we clicked the "+" sign and added a watch for `beatles`, which lets us view the value of the array:



You can try out the Chrome DevTools "Sources" panel in the next exercise.

## Exercise 23 Using the Chrome DevTools "Sources" Panel

*10 to 20 minutes*

1. Open [DebuggingTestingChrome/Exercises/watch.html](http://DebuggingTestingChrome/Exercises/watch.html) in a browser and in a code editor to review the code.
2. This page - a modified version of the example code we looked at earlier - is designed to add the text " - Fave!" to the "George" list element; however the code does not work as intended.
3. Note that we use the trinary operator in our loop, appending (or trying to append) the text " - Fave!" when we reach the list element corresponding to the "George" the array element.
4. Use the Chrome DevTools "Source" panels to help you diagnose and fix the error.



### Exercise Solution

#### DebuggingtestingChrome/Solutions/watch.html

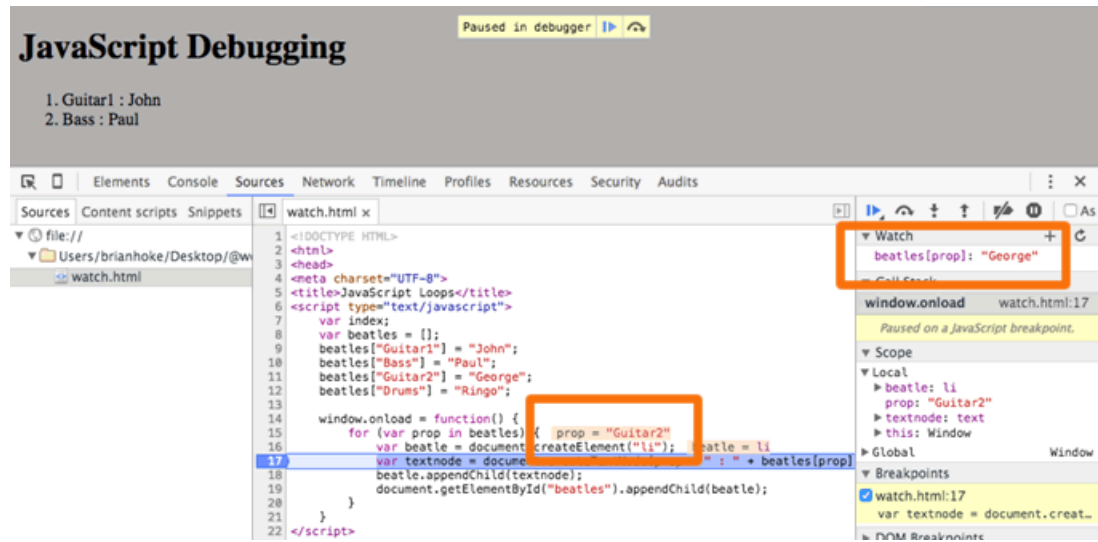
```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>JavaScript Loops</title>
6.    <script>
7.        var index;
8.        var beatles = [];
9.        beatles["Guitar1"] = "John";
10.       beatles["Bass"] = "Paul";
11.       beatles["Guitar2"] = "George";
12.       beatles["Drums"] = "Ringo";
13.
14.       window.onload = function() {
15.           for (var prop in beatles) {
16.               var beatle = document.createElement("li");
17.               var textnode = document.createTextNode(prop + " : " + beatles[prop]
18.               >>> + (prop == 'Guitar2' ? ' Fave!' : ''));
19.               beatle.appendChild(textnode);
20.               document.getElementById("beatles").appendChild(beatle);
21.           }
22.       }
23.    </script>
24.    </head>
25.    <body>
26.    <h1>JavaScript Debugging</h1>
27.    <ol id="beatles">
28.    </ol>
29.    </body>
30.    </html>
```

### Code Explanation

The issue here is that the test for whether this is "George" array element is checking `beatles[prop]` to match the value `Guitar2`; however it is the index (`prop`) that we should be checking, not the value of the array.

We can set a breakpoint in the middle of our loop and also set a watch (on `beatles[prop]`) to check why our test for the "George" element of the array

never returns true. Chrome's DevTools make it easy to see that we mistakenly evaluated the index of the array element rather than it's value.



## 11.4 Other DevTools Panels

There are other resources - other panels - available from DevTools; while beyond the scope of this course, we encourage you to check them out. These include:

- The "Network" panel, which is especially useful when debugging JavaScript Ajax functionality;
- The "Timeline" panel, which can be used to diagnose the time required for various sections of a page to load or complete;
- The "Security" panel, which is useful for checking on SSL security on a page.

For more information, visit the [Google Developers DevTools documentation \(https://developers.google.com/web/tools/chrome-devtools/\)](https://developers.google.com/web/tools/chrome-devtools/).

## 11.5 Chrome DevTools API and Extensions

Google offers an API for developers to write their own DevTools panels, to add whatever functionality might be needed for a particular developer, team, or application.

Even better than writing extensions ourselves is to make use of the work of others. Chrome lists some [featured DevTools extensions \(https://developer.chrome.com/devtools/docs/extensions-gallery\)](https://developer.chrome.com/devtools/docs/extensions-gallery) on their website; a quick Google search will turn up others. You can find extensions to make easier the process of debugging and evaluating platforms and libraries like Ruby on Rails, Angular, and Ember; tools to better diagnose page speed; and tools to extract code snippets for sharing with others.

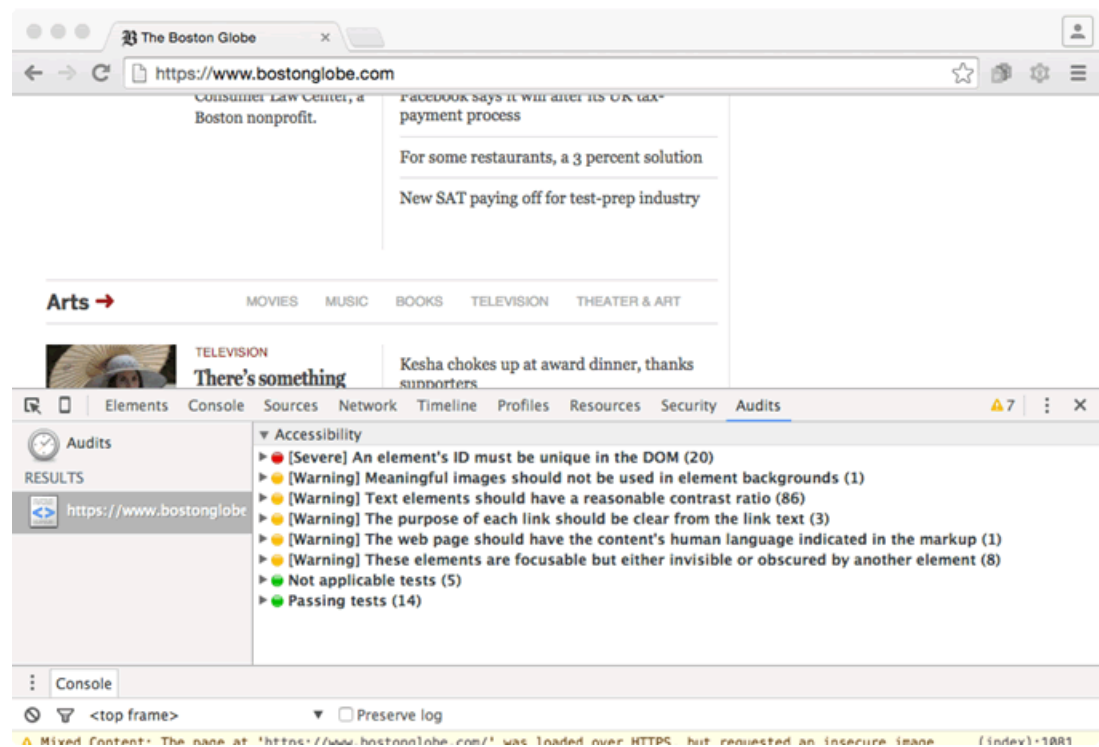
In the next exercise, we'll ask you to install and try out one of these extensions.

## Exercise 24 Accessibility Developer Tools Extension

5 to 15 minutes

1. Visit the Chrome Webstore page for the [Accessibility Developer Tools](https://chrome.google.com/webstore/detail/accessibility-developer-t/fpkknkljclfencbdbgkenhalefipecmb) (<https://chrome.google.com/webstore/detail/accessibility-developer-t/fpkknkljclfencbdbgkenhalefipecmb>) extension from a Chrome browser.
2. Install the extension and restart Chrome.
3. Visit a website of your choice and use the tools to evaluate how well the page presents code in a manner friendly for users who employ assistive technology like screen readers:
  - Right-click on any part of the page and select "Inspect".
  - Select the "Audit" panel.
  - Check the "Accessibility" option.
  - Run the audit.

The results shown in the screenshot below give a sense of how well the page uses meaningful alt tags, marks up elements, codes forms, and other aspects of how friendly the code is for users who employ assistive technology.



## 11.6 Conclusion

In this lesson, you have learned

- How to use Google's DevTools to debug JavaScript



# A1. Navigator, History, and Location Objects

**In this lesson, you will learn...**

1. To get information about the user's environment.
2. To use the `history` object to create back and forward buttons and to prevent the user from going back to a previous page.
3. To work with the `location` object.

## A1.1 The navigator Object

In the days of Netscape 4, it was common practice to branch code based on your user's browser. This is almost always a bad idea. It's better practice to check for and respond to feature support than to base your code on the type or version of the user's browser. That said, in some cases, it can be useful to be able to know what browser type and version your visitor is using. Consider, for example, a support form. Many support forms ask users what browser and operating system they're using as this can be useful information for debugging the problem. But users sometimes do not provide accurate information. And sometimes the support tech needs additional information like whether cookies are enabled. The `window.navigator` object to the rescue!

The following demo shows some of the `navigator` properties and their values. Open it in your browser to see what it reports.

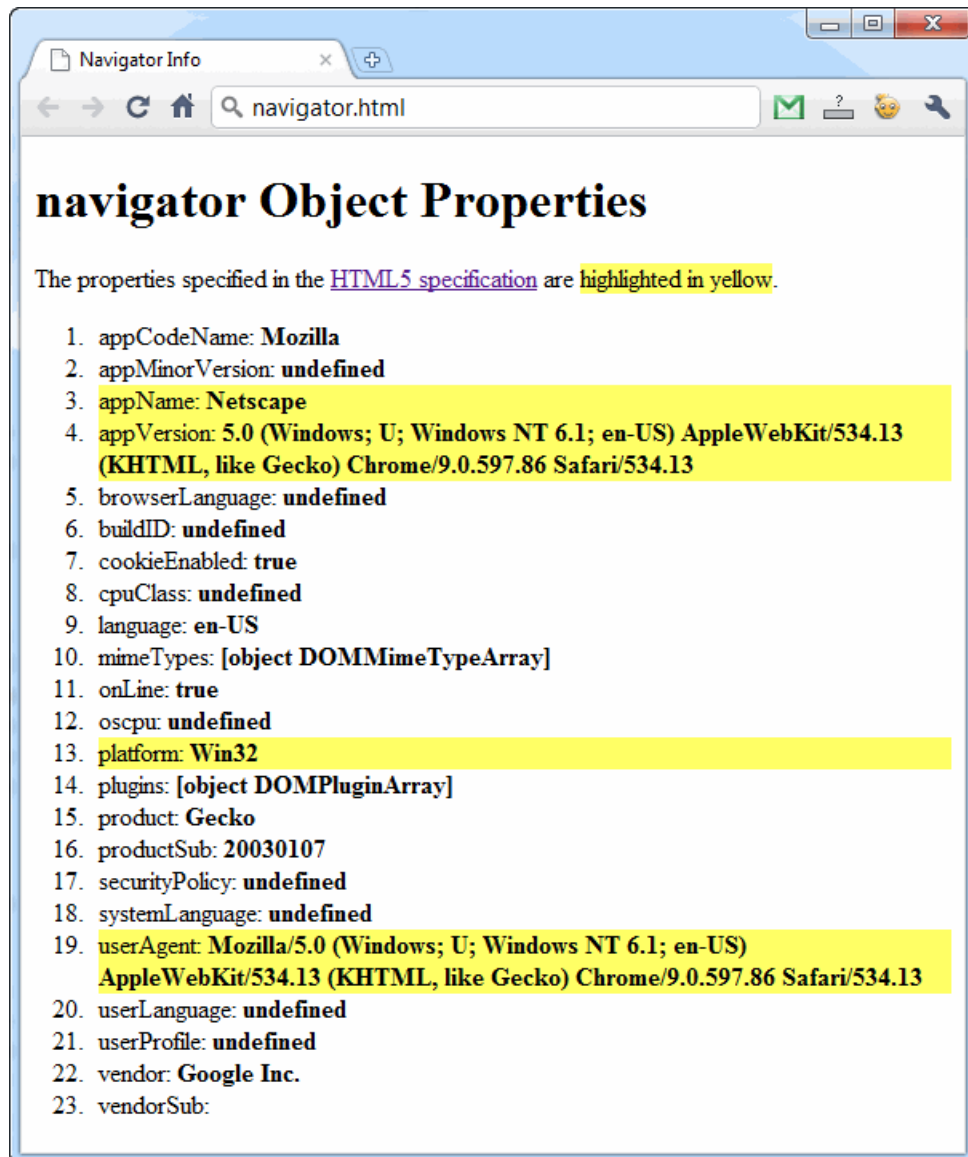
**Code Sample****NavigatorHistoryLocation/Demos/navigator.html**

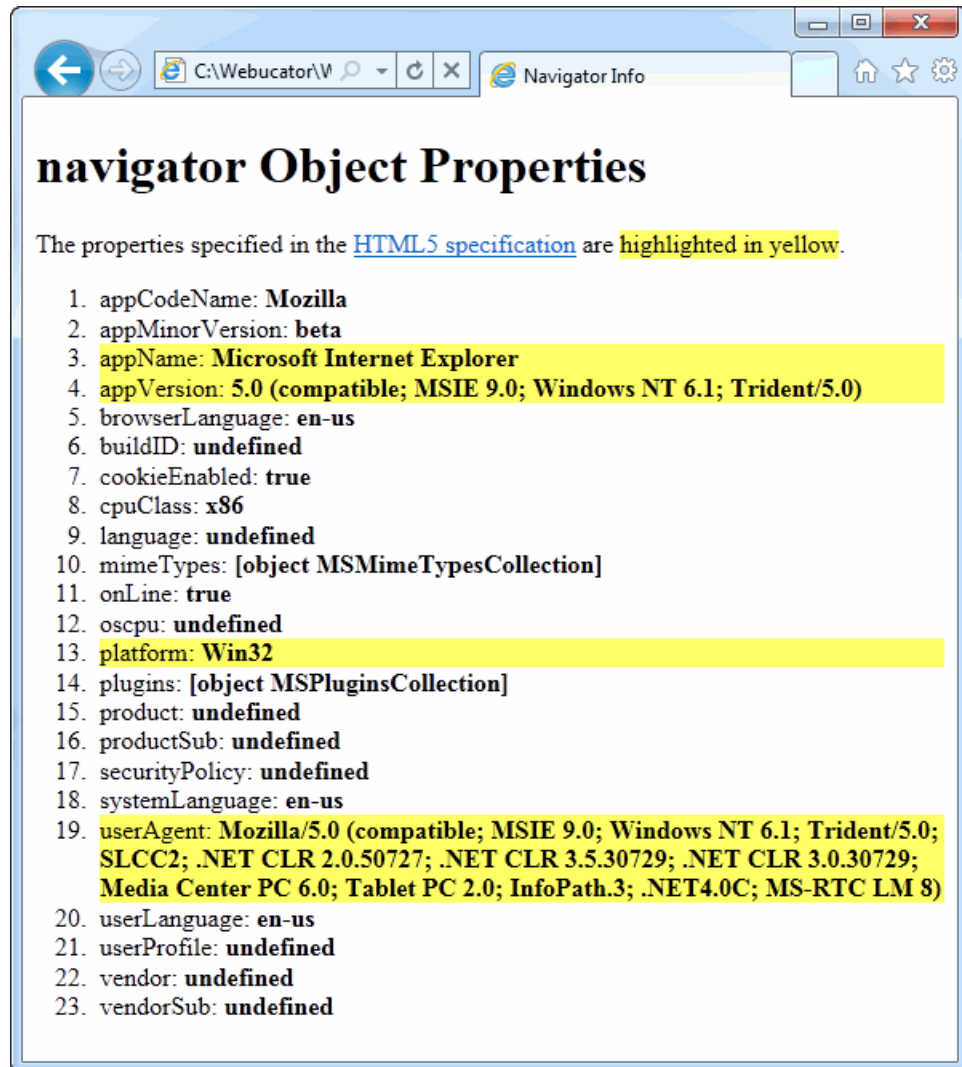
```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Navigator Info</title>
6.    <style>
7.    .html5 {
8.        background-color:#ff6;
9.    }
10.   </style>
11.   </head>
12.   <body>
13.   <h1>navigator Object Properties</h1>
14.   <p>The properties specified in the <a
      >>> href="http://www.w3.org/TR/html5/timers.html#system-state-and-
      >>> capabilities:-the-navigator-object">HTML5 specification</a>
      >>> are <span class="html5">highlighted in yellow</span>.</p>
15.   <ol>
16.   <li>appName: <strong><script>document.write(navigator.appCode >>
      >>> Name);</script></strong></li>
17.   <li>appMinorVersion: <strong><script>document.write(navigator.appMi >>
      >>> norVersion);</script></strong></li>
18.   <li class="html5">appName: <strong><script>document.write(navigator.app >>
      >>> Name);</script></strong></li>
19.   <li class="html5">appVersion: <strong><script>document.write(naviga >>
      >>> tor.appVersion);</script></strong></li>
20.   <li>browserLanguage: <strong><script>document.write(navigator.browser >>
      >>> Language);</script></strong></li>
21.   <li>buildID: <strong><script>document.write(navigator.buil >>
      >>> dID);</script></strong></li>
22.   <li>cookieEnabled: <strong><script>document.write(navigator.cookieEn >>
      >>> abled);</script></strong></li>
23.   <li>cpuClass: <strong><script>document.write(navigator.cpu >>
      >>> Class);</script></strong></li>
24.   <li>language: <strong><script>document.write(navigator.lan >>
      >>> guage);</script></strong></li>
25.   <li>mimeTypes: <strong><script>document.write(navigator.mime >>
      >>> Types);</script></strong></li>
26.   <li>onLine: <strong><script>document.write(navigator.on >>
      >>> Line);</script></strong></li>
27.   <li>oscpu: <strong><script>document.write(navigator.os >>
      >>> cpu);</script></strong></li>
28.   <li class="html5">platform: <strong><script>document.write(naviga >>
      >>> tor.platform);</script></strong></li>
```

```
29. <li>plugins: <strong><script>document.write(navigator.plug >>
    >>> ins);</script></strong></li>
30. <li>product: <strong><script>document.write(navigator.prod >>
    >>> uct);</script></strong></li>
31. <li>productSub: <strong><script>document.write(navigator.product >>
    >>> Sub);</script></strong></li>
32. <li>securityPolicy: <strong><script>document.write(navigator.security >>
    >>> Policy);</script></strong></li>
33. <li>systemLanguage: <strong><script>document.write(navigator.systemLan >>
    >>> guage);</script></strong></li>
34. <li class="html5">userAgent: <strong><script>document.write(naviga >>
    >>> tor.userAgent);</script></strong></li>
35. <li>userLanguage: <strong><script>document.write(navigator.userLan >>
    >>> guage);</script></strong></li>
36. <li>userProfile: <strong><script>document.write(navigator.userPro >>
    >>> file);</script></strong></li>
37. <li>vendor: <strong><script>document.write(navigator.ven >>
    >>> dor);</script></strong></li>
38. <li>vendorSub: <strong><script>document.write(navigator.vendor >>
    >>> Sub);</script></strong></li>
39. </ol>
40. </body>
41. </html>
```

### Code Explanation

Below we show the results in older browsers - Google Chrome 9 and Internet Explorer 9:





## Checking for Disabled Features

You may have noticed the `cookieEnabled` property in the screenshots above. In some cases, a browser may support a feature, but the user might have disabled it. This is most common with cookies, which users sometimes disable for (perceived) security reasons. The following demo shows how to check whether cookies are disabled:

**Code Sample****NavigatorHistoryLocation/Demos/cookie-check.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Cookie Check</title>
6.    </head>
7.    <body>
8.    <script>
9.    if (!navigator.cookieEnabled) {
10.    document.write("<h1>Warning: Cookies Required</h1>");
11.    document.write("<p>Please turn your cookies on and refresh the
    >>> page.</p>");
12.    document.body.style.backgroundColor = "red";
13.    }
14.    </script>
15.    <p>Rest of page goes here...</p>
16.    </body>
17.    </html>
```

## **A1.2 Feature Detection**

Although browsers have come a long way in the past several years, they unfortunately do not all support the W3C specifications to the same degree. This is particularly true with the introduction of HTML5. It is often necessary to branch the code based on the browser's support of a feature. A case in point is the HTML5 canvas element, which is used to create drawings natively in the browser. Take a look at the following code:

**Code Sample****NavigatorHistoryLocation/Demos/canvas.html**

```

1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Canvas</title>
6.    <script>
7.    function drawPath() {
8.        var canvas=document.getElementById("my-canvas");
9.        if (canvas.getContext) {
10.            alert("Canvas Supported.");
11.            //create your awesome drawing here
12.        } else {
13.            alert("Canvas Not Supported.");
14.        }
15.    }
16.    </script>
17.    </head>
18.    <body>
19.    <canvas id="my-canvas" height="200" width="200">Your browser doesn't
        >>> support canvas.</canvas>
20.    <button onclick="drawPath();">Draw</button>
21.    </body>
22.    </html>

```

**Code Explanation**

If the browser properly supports canvas, it will recognize the `getContext` method of the canvas object and execute the code in the `if` condition.

The major advantage of checking for the feature rather than checking for the type or version of the browser is that you don't have to keep updating your code to account for changes in browser versions and support.

**A1.3 Modernizr**

The example above, in which we tested whether the user visiting our [canvas.html](#) page was using a browser that supported canvas drawing; we did this through some custom JavaScript code:

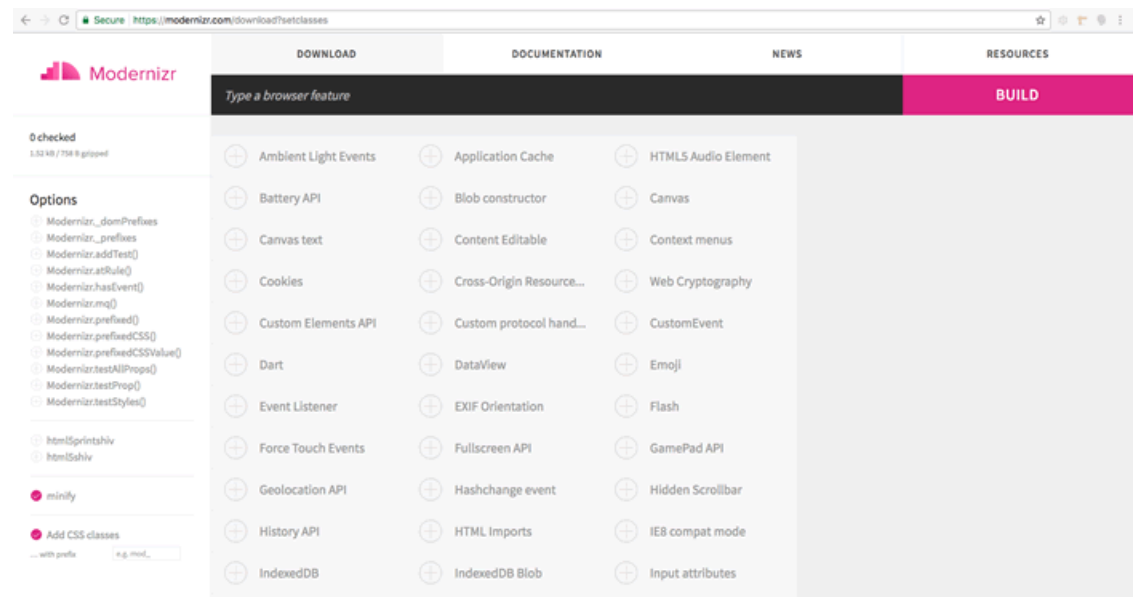
```
var canvas=document.getElementById( "my-canvas" );
if ( canvas.getContext() ) {
    //...
}
```

It worked fine for testing the current user's browser for the canvas feature. But what if there were some other features we wanted to detect? And what of the issue of the code needed to detect those features changing over time? Luckily, there's a library for that: Modernizr.

[Modernizr \(https://modernizr.com/\)](https://modernizr.com/), as described on their site, is "a collection of superfast tests - or 'detects as we like to call them - which run as your web page loads, then you can use the results to tailor the experience to the user."

## How Modernizr Works

Visiting [Modernizr's download page \(https://modernizr.com/download?setclasses\)](https://modernizr.com/download?setclasses) offers us the option of selecting which features we wish to detect:



We select the feature(s) we wish to detect, then download the custom-built JavaScript file. Once including this JavaScript file in our page, we can then reference the Modernizr object to test support for a given feature:

```
if (Modernizr.canvas) {
    //draw on canvas
}
```

The next demo gives an example of the use of Modernizr:



**Code Sample****NavigatorHistoryLocation/Demos/canvas-modernizr.html**

```

1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Canvas</title>
6.    <script src="modernizr-custom.js"></script>
7.    <script>
8.
9.    function drawPath() {
10.     if (Modernizr.canvas) {
11.       alert("Canvas Supported.");
12.       //create your awesome drawing here
13.     } else {
14.       alert("Canvas Not Supported.");
15.     }
16.   }
17. </script>
18. </head>
19. <body>
20. <canvas id="my-canvas" height="200" width="200">Your browser doesn't
    >>> support canvas.</canvas>
21. <button onclick="drawPath();">Draw</button>
22. </body>
23. </html>

```

**Code Explanation**

We downloaded Modernizr - a custom build, in this case, in which we ask to detect only the canvas feature - and included the downloaded JavaScript file in our page. We can then use Modernizr to test for canvas support: `Modernizr.canvas`.

## A1.4 The history Object

Like navigator, the history object is a property of the window object. For security and privacy reasons, the information you can get about the user's session history is limited to the number of entries, which you get by reading the `history.length` property. That's generally not so useful. However, there are a few methods of the history object, which you may indeed find useful:

**history Methods**

Method	Description
<code>go(int)</code>	Advanced or moves back (negative int) through the history.
<code>back()</code>	The same as <code>go(-1)</code>
<code>forward()</code>	The same as <code>go(1)</code>

If any of the above methods fail, no error is reported. In other words, if `history.back()` is called on a page that has no history, nothing at all happens.

**Code Sample****NavigatorHistoryLocation/Demos/history-1.html**

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8">
5.  <title>Page 1</title>
6.  <style>
7.    .there {
8.      color:blue;
9.      font-weight:bold;
10.   }
11. </style>
12. </head>
13. <body>
14. <h1>Page 1</h1>
15. <h2>Table of Contents</h2>
16. <ol>
17.   <li class="there">Page 1</li>
18.   <li><a href="history-2.html">Page 2</a></li>
19.   <li><a href="history-3.html">Page 3</a></li>
20. </ol>
21. <button onclick="history.back();">Back</button>
22. <button onclick="history.forward();">Forward</button>
23. </body>
24. </html>
```

**Code Explanation**

Try this:

1. Open [NavigatorHistoryLocation/Demos/history-1.html](#) in your browser.
2. Click the **Page 2** link.

3. Click the **Page 3** link.
4. Now click the **Back** and **Forward** buttons several times to navigate back and forth through the pages.

If you took the above steps exactly, it should have worked as expected, but now try this:

1. Open [NavigatorHistoryLocation/Demos/history-1.html](#) in your browser.
2. Click the **Page 3** link.
3. Now click the **Back** button.

Notice it goes back from **Page 3** to **Page 1**, which doesn't seem intuitive from the user's point of view. The problem is that the JavaScript isn't aware of the "appropriate" order of the pages as laid out in the **Table of Contents**. The JavaScript is just using the `history` object to simulate the browser's **Back** and **Forward** buttons.

As such, this makes using the `history` object in this way relatively useless and not recommended. In the exercise later in this lesson, we'll look at a more useful way of using the `history` object.

## A1.5 The location Object

### location Properties

Property	Description
href	Returns the full location of the page. The value can be set to change the page.
protocol	Returns the protocol used to deliver the page (e.g., "http").
host	Returns the host of the page (e.g., "www.webucator.com"). It will include the port if included.
hostname	Returns the host of the page (e.g., "www.webucator.com"), but not the port.
port	Returns the port of the host if included (e.g., "80").
pathname	Returns the path after the domain name (e.g., "/webdesign/javascript.cfm" for <a href="http://www.webucator.com/webdesign/javascript.cfm">http://www.webucator.com/webdesign/javascript.cfm</a> ).
search	Returns the querystring, including the question mark (e.g., "?q=javascript+training" for <a href="http://www.google.com/search?q=javascript+training">http://www.google.com/search?q=javascript+training</a> ).
hash	Returns the hash, including the hash mark (e.g., "#dom-location-hash" for <a href="http://www.w3.org/TR/html5/history.html#dom-location-hash">http://www.w3.org/TR/html5/history.html#dom-location-hash</a> ).

### location Methods

Method	Description
assign(url)	Navigates to the url argument. Same as <code>location.href=url;</code>
replace(url)	Replaces the current page in the history stack with the url argument.
reload()	Reloads the page.

The properties and methods above are demonstrated in the following code sample:

**Code Sample****NavigatorHistoryLocation/Demos/location.html**

```

1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>Location Info</title>
6.    </head>
7.    <body>
8.    <h1>location Object Properties</h1>
9.    <ol>
10.   <li><a href="?firstName=Nat">Add Search</a></li>
11.   <li><a href="#props">Add Hash</a></li>
12.   <li><a href="javascript:location.replace('#props')">Add Hash with
    >>> <code>replace()</code></a></li>
13.   <li><a href="javascript:location.assign('#props')">Add Hash with
    >>> <code>assign()</code></a></li>
14.   <li><a href="javascript:location.reload()>Reload</a></li>
15.   <li><a href="location.html">Start Over</a></li>
16. </ol>
17.
18. <ol id="props">
19.   <li>href: <strong><script>document.write(loca »»
    >>> tion.href);</script></strong></li>
20.   <li>protocol: <strong><script>document.write(location.proto »»
    >>> col);</script></strong></li>
21.   <li>host: <strong><script>document.write(loca »»
    >>> tion.host);</script></strong></li>
22.   <li>hostname: <strong><script>document.write(location.host »»
    >>> name);</script></strong></li>
23.   <li>port: <strong><script>document.write(loca »»
    >>> tion.port);</script></strong></li>
24.   <li>pathname: <strong><script>document.write(location.path »»
    >>> name);</script></strong></li>
25.   <li>search: <strong><script>document.write(loca »»
    >>> tion.search);</script></strong></li>
26.   <li>hash: <strong><script>document.write(loca »»
    >>> tion.hash);</script></strong></li>
27.   <li>history.length: <strong><script>document.write(histo »»
    >>> ry.length);</script></strong></li>
28. </ol>
29. </body>
30. </html>

```

### Code Explanation

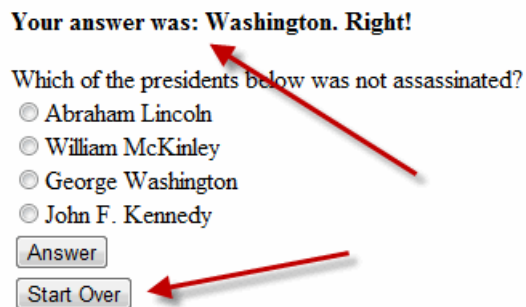
Most of the `location` properties and methods above are only used in advanced applications that require customizing how the site/application responds to the user pressing the **Back** and **Forward** buttons. The exception is the `href` property, which is commonly used to create "button links" as you'll see in the exercise that follows:

## Exercise 25 Creating a Simple Quiz

15 to 25 minutes

In this exercise, you will create a simple quiz that asks one question per page and does not allow the user to navigate back to past questions.

1. Open [NavigatorHistoryLocation/Exercises/question1.html](#) for editing.
  - A. Add code so that the page tries to navigate forward.
2. Open [NavigatorHistoryLocation/Exercises/question2.html](#) for editing.
  - A. Again, add code so that the page tries to navigate forward.
  - B. Add code to determine if the user chose the correct answer ("Washington") and report the answer.
  - C. Add a button to the end of the form that, when clicked, takes the user back to [start-quiz.html](#). If the user answers correctly, the page should look like this:



3. Make the same modifications to [question3.html](#) and [end-quiz.html](#).
4. Test your solution in a browser by opening [NavigatorHistoryLocation/Exercises/start-quiz.html](#) and working through the quiz. You should not be able to go back to a previous question without starting the whole quiz over.

### Exercise Solution

#### NavigatorHistoryLocation/Solutions/question1.html

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>History Test: Question 1</title>
6.    <script>
7.        history.go(1);
8.    </script>
9.    </head>
10.   <body>
11.   <form action="question2.html">
12.       Who was the first president of the United States?<br>
13.       <input type="radio" name="answer" value="Lincoln">Abraham Lincoln<br>
14.           >>>
15.       <input type="radio" name="answer" value="McKinley">William McKinley<br>
16.           >>>
17.       <input type="radio" name="answer" value="Washington">George Washing »»
18.           >>> ton<br>
19.       <input type="radio" name="answer" value="Kennedy">John F. Kennedy<br>
20.           >>>
21.       <input type="submit" value="Answer">
22.   </form>
23. </body>
24. </html>
```



**Exercise Solution****NavigatorHistoryLocation/Solutions/question2.html**

```

1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>History Test: Question 2</title>
6.    <script>
7.        history.go(1);
8.        var s = location.search;
9.        var start = s.indexOf("=")+1;
10.       var answer = s.substring(start);
11.       var response = (answer == "Washington") ? "Right!" : "Wrong!";
12.    </script>
13.    </head>
14.    <body>
15.    <p style="font-weight:bold;">
16.        <script>
17.            document.write("Your answer was: " + answer + ". " + response);
18.        </script>
19.    </p>
20.    <form action="question3.html">
21.        Which of the presidents below was not assassinated?<br>
22.        <input type="radio" name="answer" value="Lincoln">Abraham Lincoln<br>
23.            >>>
24.        <input type="radio" name="answer" value="McKinley">William McKinley<br>
25.            >>>
26.        <input type="radio" name="answer" value="Washington">George Washing >>
27.            >>> ton<br>
28.        <input type="radio" name="answer" value="Kennedy">John F. Kennedy<br>
29.            >>>
30.        <input type="submit" value="Answer"><br>
31.        <input type="button" onclick="location.href='start-quiz.html';" val >>
32.            >>> ue="Start Over">
33.    </form>
34.    </body>
35.    </html>

```

**Exercise Solution****NavigatorHistoryLocation/Solutions/question3.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>History Test: Question 3</title>
6.    <script>
7.        history.go(1);
8.        var s = location.search;
9.        var start = s.indexOf("=")+1;
10.       var answer = s.substring(start);
11.       var response = (answer == "Washington") ? "Right!" : "Wrong!";
12.    </script>
13.    </head>
14.    <body>
15.    <p style="font-weight:bold;">
16.        <script>
17.            document.write("Your answer was: " + answer + ". " + response);
18.        </script>
19.    </p>
20.    <form action="end-quiz.html">
21.        Which president was known as "His Excellency?"<br>
22.        <input type="radio" name="answer" value="Lincoln">Abraham Lincoln<br>
23.            >>>
24.        <input type="radio" name="answer" value="McKinley">William McKinley<br>
25.            >>>
26.        <input type="radio" name="answer" value="Washington">George Washing »»
27.            >>> ton<br>
28.        <input type="radio" name="answer" value="Kennedy">John F. Kennedy<br>
29.            >>>
30.        <input type="submit" value="Answer"><br>
31.        <input type="button" onclick="location.href='start-quiz.html';" val »»
32.            >>> ue="Start Over">
33.    </form>
34.    </body>
35.    </html>
```

**Exercise Solution****NavigatorHistoryLocation/Solutions/end-quiz.html**

```
1.    <!DOCTYPE HTML>
2.    <html>
3.    <head>
4.    <meta charset="UTF-8">
5.    <title>History Test: End</title>
6.    <script>
7.        var s = location.search;
8.        var start = s.indexOf("=")+1;
9.        var answer = s.substring(start);
10.    var response = (answer == "Washington") ? "Right!" : "Wrong!";
11.    history.go(1);
12. </script>
13. </head>
14. <body>
15. <p style="font-weight:bold;">
16.     <script>
17.         document.write("Your answer was: " + answer + ". " + response);
18.     </script>
19. </p>
20. <p>You're done.</p>
21. <input type="button" onclick="location.href='start-quiz.html';" val »»
    >>> ue="Start Over">
22. </body>
23. </html>
```

## A1.6 Conclusion

In this lesson, you have learned to work with the navigator, history, and location objects.