
JavaScript for the AlmostNovice Programmer

*© 2012 Sam Scott, Sheridan Institute of
Technology and Advanced Learning.*

June 2012 draft

Java Connection

AlmostNovice. If you have just completed an introductory course in Java, you're a member of the class of AlmostNovice programmers. (In JavaScript we would say you're an extension of the AlmostNovice prototype.) Either way, this text was written just for you. 😊

Table of Contents

1. Getting Started.....	1
1.1 Who is This Text For?	1
1.2 What is JavaScript?	2
1.3 Where Is JavaScript Going?.....	2
1.4 Is JavaScript Like Java?	3
2. What You Already Know	4
3. Two Things You Didn't Already Know	5
3.1 Imperative Programming... ..	5
3.1.1 ... in a Browser Console	5
3.1.2 ... In an HTML <script> Element	7
3.1.3 A Note on IDE's	7
3.2 Weakly Typed Variables.....	8
3.2.1 Weak Typing and Boolean Expressions.....	9
3.2.2 Explicit Casting	10
4. The document Object and the DOM	11
4.1 The Structure of the DOM	11
4.2 What is an Object?.....	12
4.3 The document.getElementById Function	13
4.3.1 Changing an Element's innerHTML	13
4.3.2 Changing an Element's style.....	14
4.3.3 Changing an Element's class.....	15
5. Functions and Events	16
5.1 Defining a Function	16
5.1.1 Responding to an Event	16
5.1.2 Clicking on Buttons	18
5.2 Parameters, Global Variables, and Return Values	18
5.3 Application: Drop-Down Menus	20
6. Arrays in JavaScript	22
6.1 Consequences of Weak Typing	22

6.2 Arrays and the DOM	23
7. Form Input	24
7.1 Form Basics	24
7.1.1 Possible <code><input></code> Types.....	25
7.2 Forms with JavaScript	27
7.2.1 Responding to Events.....	27
7.2.2 Accessing User Input	28
7.2.3 Building Client-Side Web Applications.....	30
7.3 Advanced Form Features	30
8. Autonomous Client-Side Web Apps.....	33
8.1 Web Storage.....	33
8.1.1 Initializing Web Storage	34
8.1.2 Advice on Sharing Web Storage.....	35
8.2 The App Cache	36
8.2.1 The <code>manifest</code> Attribute.....	36
8.2.2 The Manifest File.....	37
8.2.3 Updating the Manifest	37
9. The jQuery Library.....	38
9.1 Selectors and Effects	38
9.1.1 The jQuery Sandbox.....	38
9.1.1 The <code>\$</code> Function	39
9.1.2 Inner HTML, Attributes, and Styles.....	40
9.1.3 Effects and Animations	42
9.2 Events in jQuery and the Truth About JavaScript Functions.....	43
9.2.1 The Truth About Functions	43
9.2.2 Adding Event Handlers.....	45
9.2.3 Event Handlers in jQuery	45
9.2.4 jQuery Callback functions	46
11. Using AJAX with jQuery.....	47
12. Gaming on the HTML5 Canvas.....	47
13. Object-Oriented JavaScript	47

1. Getting Started

This text is intended to be a complete course in JavaScript programming for web application design. You will learn the basics of the JavaScript language and how to embed JavaScript programs into a web page. You will learn how to create client-side web apps. You will learn how to effectively use the extensions provided in the widely-used jQuery package, including its built-in AJAX capabilities. You will learn how to use the new HTML5 canvas to create dynamic, graphical web content. You will learn how to access the new local and session storage capabilities of HTML5. And you will learn how to implement object-oriented software design in JavaScript.

1.1 Who is This Text For?

This text is aimed at the “AlmostNovice” programmer, though it could also be used effectively by experienced programmers who are new to JavaScript.

AlmostNovice programmers have completed a first programming course (preferably in C, C++, or Java) and a first course in web design (HTML and CSS, preferably including some of the new HTML5 and CSS3 capabilities). AlmostNovice programmers already have a good understanding of variables, data types, if statements, loops, methods (or functions), HTML elements (tags and attributes), and CSS rules (properties and values). They have designed simple web pages and they have written computer programs that are split up into multiple methods (possibly also multiple classes) to process user input and produce new output.

If you are an AlmostNovice programmer, or an experienced programmer who has never used JavaScript, this text will leverage your prior knowledge of programming to skim over stuff you already know and focus on the differences and unique capabilities of JavaScript, jQuery, and HTML5. It will also leverage your existing web design skills to get you creating client-side web applications in JavaScript more quickly.

Java Connection

What is this? Look for these boxes to get a quick summary of the similarities and differences between Java and JavaScript.

Try it Yourself

If you have never seen <http://w3schools.com> before, take a moment to visit it now. I will often refer you to this site for more detailed information, and you are encouraged to move back and forth between this text and the w3schools site to help your understanding.

W3Schools is a great quick reference if you need a refresher on HTML, HTML5, CSS, CSS3, and the Java-like portion of JavaScript (loops, if statements, basic syntax, etc.). It can also be read as a JavaScript textbook with tutorials and hands-on exercises, or as a complete JavaScript reference for more experienced programmers.

(You will see “Try it Yourself” sections like this throughout the text. They are here to give you something to do, to keep you engaged and thinking, and to break up the monotony of reading. Think of these sections as an opportunity to experiment and test out what you have learned.)

1.2 What is JavaScript?

JavaScript was first shipped with the now-defunct NetScape browser in 1995. It is a fully-functional programming language that was designed to be embedded in an HTML page, creating **Dynamic HTML (DHTML)**. Dynamic HTML pages make use of JavaScript statements and **functions** (i.e. methods) to change their appearance and contents both as they are loading and after they are loaded. Static HTML pages, on the other hand, always look the same and do not change once they are loaded.¹

JavaScript was standardized as **ECMAScript** a couple of years after its initial release. The name “JavaScript” is a trademark of the Oracle corporation, but other implementations of the ECMAScript also exist (notably Microsoft’s JScript used in Internet Explorer and Adobe’s ActionScript used in Flash).

Java Connection

Methods: JavaScript functions are very similar to Java methods, although as we shall see, there are also some very important differences.

Almost all web applications make use of JavaScript for drop down menus, pop-up windows, form validation, and other dynamic elements that can be implemented on the client side of the client/server architecture. Many web sites also make use of JavaScript with AJAX (Asynchronous JavaScript and XML) to load new content from the server and refresh parts of the page after load time. This is how Google makes suggestions as you type into the search bar, and how Facebook loads new content when you get to the bottom of your activity feed.

1.3 Where Is JavaScript Going?

JavaScript is not just for web pages any more. JavaScript interpreters are also embedded into the Microsoft Windows Desktop, Adobe Acrobat, Open Office, and a number of other environments to allow programmers to add apps and other dynamic functionality to these products.

With the growth of HTML5, JavaScript is poised to become the language of choice for web-based, hybrid, and standalone mobile apps, and will soon replace Adobe’s Flash as the primary way to produce graphics-intensive web applications like games, interactive diagrams, and fancy menu systems. It may not be long before a solid working knowledge of JavaScript is a requirement for many programming jobs.

¹ CSS (especially CSS3) also makes HTML more dynamic (with hover styles, media queries, animated transitions, etc.) but JavaScript adds almost unlimited potential to modify styles, structure, and content on the fly.

1.4 Is JavaScript Like Java?

There are two widely-held misconceptions about JavaScript. The first is that Java and JavaScript are the same language, or two versions of the same language. In fact, JavaScript is a very different programming language from Java. The similarity in names is the result of a marketing decision made by NetScape in the 1990's. It's true that the two languages can look similar at first glance. Like Java, JavaScript derives much of its basic syntax from the C programming language. But don't be fooled by surface appearances. JavaScript's variables, arrays, objects, and methods are implemented very differently from their counterparts in C, C++ and Java.

The second misconception is that JavaScript is somehow a less powerful, simpler, or more "lightweight" language than Java. In this case, there is a kernel of truth. It is true that JavaScript is designed to run in a restricted environment (e.g. a web page, a PDF document, the Windows desktop, etc.) and that historically, it has often been used for very simple tasks (e.g. image rollovers and drop-down menus). It is also true that it runs more slowly (it's interpreted rather than compiled), making it less suited to high performance applications. And it is also true that JavaScript does not have easy access to the same kind of standard plugin libraries that Java, C, and C++ all enjoy.

But JavaScript is not "lightweight" or "simpler" in the sense of being easier to learn. Writing effective code in JavaScript can be just as difficult as any other language, and a deep understanding of how the language works is as important to being an effective JavaScript programmer as it is for a programmer in any other language.

And JavaScript is also not "less powerful" in the theoretical sense than any other language. JavaScript is a fully developed, **Turing complete**, programming language. This means that there is no computational problem that can be solved in Java that could not also be solved in JavaScript. This is partly why the future of application development is looking like it will involve more and more JavaScript.

2. What You Already Know

Because JavaScript and Java are both based on C syntax, and because you are an AlmostNovice (or better) with experience in C, C++, or Java, you should have no problem with many of the nuts and bolts of the language. Statements, comments, operators, comparisons, selection statements (`if`, `switch`), repetition statements (`for`, `while`, etc.) and even catching and throwing exceptions all work pretty much exactly how you would expect them to. The `String` and `Math` classes also contain most of the same functions as their Java counterparts (e.g. `String.charAt`, `Math.random`, etc.)

Java Connection

Syntax. JavaScript's core syntax is very similar to Java. But don't be fooled by surface appearances. Some deep differences lurk beneath.

Try it Yourself

If you are an AlmostNovice (or better) but you have no experience with a C-like language, or if you need to brush up, w3schools.com is a great place to go for a refresher. Just click on the JavaScript link and then choose the topics from the list on the left. Even if you're a Java programmer, you should probably take a quick look at the sections on **JS String** and **JS Math** because there are some slight differences from the Java `String` and `Math` classes.

But wait! Don't actually try to program anything yet. There are two crucial things you need to know before you can leverage your existing knowledge.

3. Two Things You Didn't Already Know

So now you know what you already know. But before you can start writing JavaScript programs, you need to understand JavaScript's support for **imperative programming** and **weakly-typed variables**.

3.1 Imperative Programming...

Imperative programs consist of a list of commands (statements) in the order they are to be executed. These statements do not need to be part of a class, object, method, function or any other structure.

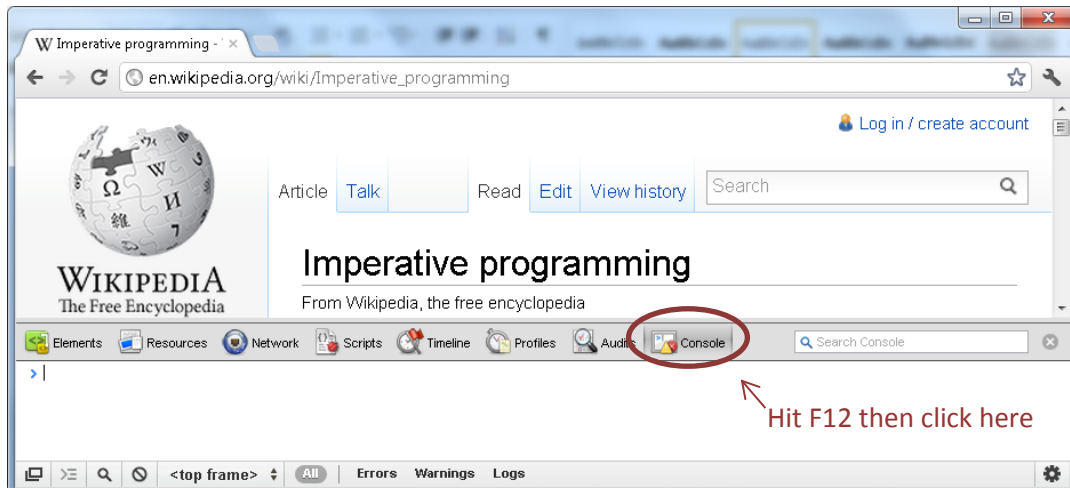
Java Connection

Statements. In Java, every statement must be part of a class or method. This is not true in JavaScript.

3.1.1 ... in a Browser Console

The easiest way to execute JavaScript statements is using the JavaScript console of your favorite browser. In Google Chrome, hit F12 to open the developer tools, then choose "Console" from the tabs that appear at the bottom of the page. (If Chrome is not your favorite browser, you can either make Chrome your new favorite, or you can search through the menus on any another browser for "Developer Tools", "JavaScript Console" or something similar.)

On Chrome, the console will appear as a new panel beneath the web page, like this.



Now you're ready to execute your first JavaScript statement. At the flashing cursor beside the '>', type:

```
window.alert("Hello, World!");
```

...and hit enter. See the popup message?

What you have just done is execute a JavaScript **function** named **alert** that is part of the built-in **window** object. Functions are a lot like Java methods or C functions – you **call** them by typing their name

followed by a list of arguments inside round brackets. If a function belongs to an object, you type the name of the object first followed by the `.` operator.

No matter what browser you are using, you will always have access to a `window` object containing variables and functions pertaining to the currently active panel of the browser.² And because you are always “inside” the `window` object when executing JavaScript, you don’t actually need to type the “`window.`” part. So this will work, too:

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

Here are a couple more `window` functions to try:

```
prompt('What is your name? ');
confirm("You're programming in
JavaScript!");
```

Java Connection

Semicolons. In JavaScript, you often don’t need a semicolon at the end of a statement, but sometimes you do. Instead of trying to remember which case is which, it’s better to always include one.

Note the **return values** of each function that appear in the console window. The `prompt`, `alert` and `confirm` functions are pretty straightforward and are discussed under the heading “JS Popup Boxes” on w3Schools.com.

Just like Java and other languages, you can use the return value of one function as an argument to another. Try this...

```
alert(prompt('What do you want to say? '));
```

or this...

```
if (confirm('OK?')) alert("You're OK"); else alert("You're not OK?");
```

Java Connection

Quoted Strings. In Java, double quotes are for Strings, single quotes are for chars. JavaScript doesn’t have a char type, so you’re free to use either single or double quotes for a String, as shown in the examples above. This is handy to avoid escape sequences (instead of `"\"Murder,\" , she wrote"`, you can use single quotes around the whole string: `'\"Murder,\" , she wrote'`).

You can also use the console to evaluate JavaScript expressions for you. Try the following:

```
5 + 3 * 2
5 <= 3
Math.round(Math.PI*2)
Math.round(Math.PI*2) < Math.PI*Math.PI
```

² In this case, the active panel is the console window itself. If the commands were part of a web page, the active panel would be the one that displays the web page content.

Try it Yourself

The console is a great tool for testing out JavaScript statements and expressions to get the syntax right before incorporating them into a web page. The Chrome console can help you remember method names too (e.g. try typing "Math.r" and look at the suggestions that pop up), and if you ever want to repeat a command, you can always hit the up arrow on your keyboard to cycle through recent commands, tweak them, and try again.

3.1.2 ... In an HTML `<script>` Element

Of course the real point of JavaScript is to embed a program into a web page. You can put JavaScript statements anywhere you want in an HTML document, as long as they are inside a `<script>` element with a `type` attribute set to "text/javascript", as in the example below.

```
<script type="text/javascript">
    alert("Hello, world!");
</script>
```

You can put any number of `<script>` elements anywhere inside the `<body>` or `<head>` elements of an HTML document. The browser will read the page from top to bottom as it loads it. When it gets to a `<script>` element, it will execute the commands inside before moving on. So if the `<script>` element appears inside the `<head>`, the page will be blank until the script has finished executing. If it appears in the middle of the `<body>`, the first part of the page will be displayed, then the `<script>` element will run, then the rest of the page will be displayed.

3.1.3 A Note on IDE's

You can write HTML, CSS, and JavaScript in a text editor and test it in a browser, but there are much more programmer-friendly environments out there for code creation. I recommend that you at least use a programming editor like **NotePad++**. This program will use color to highlight keywords, literals, and other elements, and will also do some simple bracket matching for you.

Better still is to use a fully-developed **Integrated Development Environment** (IDE) like **NetBeans** or **Eclipse**. For web programming, I like NetBeans the best.³ Just download the latest copy, start a Java Project, right click and create an HTML document (you may have to go **New... Other...** to find HTML the first time). You'll get a starting template, errors and warnings, as well as shortcuts and suggestions. You can also create Cascading Style Sheet files and JavaScript files this way as well.

Try it Yourself

Create a new project in your favorite IDE, and inside that project create a new HTML file (you may have to choose the "other" file type at first). Make sure you can view the HTML page, then include the `<script>` element shown above somewhere in the page and view the page again.

³ You might also like the Aptana version of Eclipse (www.aptana.com), which is specifically configured for web programming.

Try moving the `<script>` element to different positions in the file. What do you notice about the behavior of the page when you reload it with the script element in different positions? Is there anywhere you can't put it?

3.2 Weakly Typed Variables

The second important thing you don't already know is that variables in JavaScript are **weakly typed**. JavaScript supports a number of **data types** including **Number**, **String**, **Boolean**, and **Object**, but you do not have to declare a variable's type in order to use it and you can change the type of a variable as the program runs. Variable declaration is done with the generic keyword `var`.

Try the statements below in the JavaScript console of your browser.

```
var x = 100;
alert(x);
```

```
x = "I'm a string now";
alert(x);
```

```
x = 45.3;
alert(x);
```

```
x = "$"+x;
alert(x);
```

Tip: To reset the console and make it forget all the variables you have defined in this session, just hit the refresh button on your browser.

Notice that not only can `x` change its type from a number to a string and back again, the `alert` function can also take an argument of any type.

Other than the fact that variables are weakly typed, they work in pretty much the way you are used to. Here's a little loop for counting to 10. You can try it out by creating a simple HTML file with a `<script>` element. (You could also try it in your JavaScript console using **SHIFT-ENTER** to separate the lines, then **ENTER** at the end to execute.)

```
for(var i = 1; i <= 10; i++)
    alert(i);
```

Here's another one to try.

```
var n = prompt("Enter a number");
if (n < 1000)
    alert("that was a small
number");
else
    alert("that was a big number");
```

Java Connection

Data Types. Java is strongly typed, but JavaScript is weakly typed. This seemingly small difference has ripple effects throughout the entire language.

You can often use variables without declaring them at all, but this is considered very bad programming practice, for reasons we will discuss later. **Always declare your variables!**

Try it Yourself

Write a web page that presents the user with an arithmetic problem (e.g. “What is $23 + -4$?”) and asks for the answer using a `prompt`. If they get it right, the word “Correct” should appear in green on the web page. If they get it wrong, the word “Wrong” should appear in red.

Extend the web page so that it generates the integers for the arithmetic problem randomly. Keep presenting the question until they get it right. Then show a success message on the page.

Java Connection

Random Numbers. `Math.random()` works the same way in both Java and JavaScript. But you can’t cast to an integer with `(int)` in JavaScript (why not?) so you will have to use `Math.round()`. Look up the JavaScript `Math` object on w3schools for more info.

3.2.1 Weak Typing and Boolean Expressions

In **Boolean expressions** (i.e. the expressions that evaluate to `true` or `false` to control `if` statements and loops) aggressive implicit type casting is used to compare anything to anything else. **Type casting** is the act of converting one data type to another. **Implicit type casting** is when the compiler or interpreter does the conversion automatically for you.

Try it Yourself

To see JavaScript’s aggressive implicit type casting in action, try the following in the console:

```
var x = "hi";
x == 45.3      // legal, but false
x >= 45.3      // legal, but false
x == "hi"      // true
```

Java Connection

Implicit Type Casting. “`double x=45`” is an example of implicit type casting in Java (Why?) JavaScript does this a *lot* more than Java.

If a `String` contains a representation of a number, it can be compared to integers and floats...

```
var y = "50";
x = 50;
y == x          // true, even though y is a string and x is a number
```

Because of this “type-blindness”, there is a special Boolean operator “`===`” that returns true if two values are equal and also of the same type. Similarly, “`!==`” is a “not equals” operator that respects the types of the operands.

```
y === x          // false because y is a String and x is a number
y === "50"       // true
y !== x          // true
y !== "50"       // false
```

Java Connection

String Comparison. In Java you write `x.equals("hi")`. In JavaScript you just use `==`.

Because of the confusion that implicit type casting can cause, it is often considered best practice to only use `===` and `!==` and never use `==` and `!=`.

Java Connection

Boolean Operators. JavaScript adds two new Boolean operators, `===` and `!==`.

3.2.2 Explicit Casting

Sometimes you really want to treat a String as a number. For example, the `prompt` function always returns a string, which works for most things, but suppose we are asking the user to enter a number and then we want to add 1 to it. The code below won't do it.

```
var y=prompt("Enter a number");    // suppose the user enters "50".
y = y + 1;                        // now y is the string "501".
```

Java Connection

Explicit Type Casting. In Java this can often be done with a casting operator. For example `(int)23.5` converts the double 23.5 to the int 23. There is no such mechanism in JavaScript. Instead you use functions like `parseInt`.

The above doesn't do what we want because `y` is a string, so it treats `+` as concatenation (much like Java). In this case, we need to use **explicit type casting** to force the type we want. JavaScript has built in `parseInt` and `parseFloat` functions for this very purpose. Unlike `alert`, `prompt`, and `confirm` which belong to the window object, `parseInt` and `parseFloat` are truly global. They do not belong to any object in the system and can be used anywhere.

```
y = parseInt(y)+1;                // now y is the number 502
```

But what if `parseInt` or `parseFloat` fail (i.e. the user typed something that can't be interpreted as a number)? In that case, they return the special value `NaN` (Not a Number) which can be detected with the Boolean function `isNaN`. So to be fully robust, you could do the following:

```
var n;
do {
    n = prompt("Enter a number");
} while (isNaN(n));
n = parseFloat(n) + 1;
alert(n);
```

JavaScript always tries to do the best it can to avoid crashing. So `parseInt("50.5")` will work and return 50. So will `parseInt("50x6")`. This can be both a blessing or a curse depending on what the programmer needs.

Try It Yourself

1. Get the `whatswrong.html` file from the example pack, read the instructions in the comment header of the file, and run it to see what goes wrong. Explain exactly where the error happens, and fix it.
2. Further fix the code from question 1 so that it recovers gracefully when the user types a bad number.

4. The document Object and the DOM

Now you are up and running, writing JavaScript code. But up to this point, your interactions with the user have entirely made use of pop-ups. While this method of communication is sometimes used on web pages, most input to a web app consists of mouse or keyboard events and “output” comes in the form of modifications to the document (e.g. changing the color of a box, popping up a menu, changing the text of an element, dynamically loading and displaying new content, etc.). This chapter will get you started down the road to more realistic user interaction.

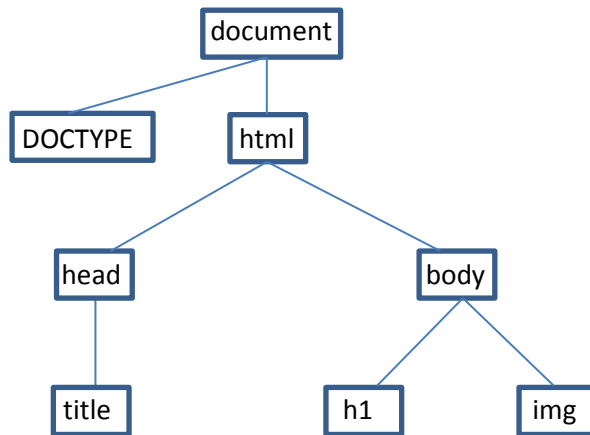
Every JavaScript program running in a web browser has access to a `document` object. This object contains the browser’s internal representation of the **Document Object Model (DOM)**, and contains all the information the browser gets from the HTML tags and attributes, CSS style rules, images, and other components that make up the page.

4.1 The Structure of the DOM

Like any bracketed structure, an HTML page can be viewed as a hierarchical “tree” of elements containing other elements.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <h1 id="message">Hello, World!</h1>
    
  </body>
</html>
```

In the example above, the document consists of a `<!DOCTYPE>` element and an `<html>` element. The `<html>` element contains a `<head>` and a `<body>`. The `<head>` contains a `<title>` element and the `<body>` contains an `<h1>` element and an `` element. This set of relationships can be diagrammed as a kind of “family tree” of elements, as shown below.



Each node is a JavaScript object representing a single element. Each element may contain attributes, content (which might include other elements), and CSS style information.

In a tree diagram like this, each element is referred to as a **node**. Each node can have one **parent** (above it), any number of **children** directly below it, and **siblings** (nodes with the same parent). The **root** node is at the top and the leaf nodes are the ones with no children. Each node in the tree is represented by a JavaScript **object**. When you are using JavaScript in a web page, `document` is the name for a special object variable representing the root node.

Try it Yourself

Open any web page in Chrome. Hit F12 and instead of the Console tab, go to the Elements tab. What you are looking at here is the browser's DOM. You can click to expand an element.

Now go back to the JavaScript Console, type in `document`, and hit return. Click the response to open the DOM and explore it. This is how you access the browser's DOM from within a JavaScript program.

4.2 What is an Object?

An object is just a package of variables and functions stored together. If you have experience with Java or C++ you have probably used objects before, and you may even have created your own objects.

For example, in JavaScript every String object has a variable associated with it named `length` and a function named `charAt`. The code below creates a string and displays its length and first character. Note the use of the variable name (`s`) and, like in Java and C++, the `.` operator to access the variables and functions that belong to it.

```
var s = "JavaScript is Cool. ";
alert( s.length );
alert( s.charAt(0) );
```


One thing that is a little different about JavaScript is that you can also use square bracket notation to access an object's variables and functions. For example, instead of `s.length` and `s.charAt`, you can write `s["length"]` and `s["charAt"]`, as shown below.⁴

```
var s = "JavaScript is Cool. ";
alert( s["length"] );
alert( s["charAt"](0) );
```

The usefulness of this will become clear soon.

Java Connection

Objects. There are two ways of accessing an object's fields in JavaScript. Like Java, you can write `objectName.fieldName`. Unlike Java, you can also write `objectName["fieldname"]`.

4.3 The `document.getElementById` Function

If an element has an `id` attribute, you can use the `getElementById` function (note the lower-case `d`) of the `document` object to reach into the DOM and grab the `Node` object corresponding to that element. Once you have it, you can access and/or change the content of the element, its style information, and any of its attributes.

Any element you retrieve using `document.getElementById` will be an object of type `Node`. Every `Node` object contains the following fields

<code>innerHTML</code> :	a String representing the contents of the node
<code>style</code> :	the CSS style information associated with the node
<code>className</code> :	the HTML <code>class</code> attribute
<i>plus...</i>	there will be one field for every attribute specified in the HTML for that node (e.g. <code>id</code> , <code>src</code> , <code>href</code> , <code>value</code> , <code>type</code> , etc.)

4.3.1 Changing an Element's `innerHTML`

The `innerHTML` property returns a string representation of the contents of the node. The contents is everything that appears between the opening and closing tags that define that node.

```
<h1 id="heading">A Simple Example</h1>
<script type="text/javascript">
    alert("Press OK to see me change my own heading.");
    document.getElementById("heading").innerHTML = "Done";
</script>
```

⁴ This square bracket object notation looks and behaves exactly like a data structure that is often referred to as an “associative array”, a “map”, or a “dictionary”. Indeed, there is little difference in many cases between an object, an associative array, a map, or a dictionary in JavaScript, so both styles of referencing are included for the convenience of the programmer.

In the example above, after the user presses OK, they will see the heading change from “A Simple Example” to “Done”.

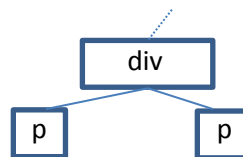
You can also write HTML into the contents of an element, and the DOM will be automatically updated with the new elements you created, as in the example below.

```
<div id="target">A Simple Example</div>
<script type="text/javascript">
  alert("Press OK to see me rewrite the DOM.");
  document.getElementById("target").innerHTML =
    "<p>First paragraph</p><p>Second paragraph</p>";
</script>
```

Before Pressing OK



After Pressing OK



Try It Yourself

The file **imperative2.html** in the examples package contains an example of how to use `getElementById` and the `innerHTML` field of an element. But you can also try it for yourself by loading **imperative1.html** into Chrome or FireFox, opening the JavaScript Console, and accessing the element with the `id` “heading”. *TODO: Update This Example.*

4.3.2 Changing an Element’s style

Changing an elements’ CSS information can be done by accessing its `style` variable. This variable is also an object, containing all the style information for this Node. The style object has a variable for every possible CSS property.

For example, to make an element red, you could do the following:⁵

```
var e = document.getElementById("myElement");
e.style.color = "red";
```

Notice that in the above example, we have first retrieved the element using `getElementById`, and then stored it in the variable `e`. Then we use the variable `e` to set the `color` of the element.

The only wrinkle in this process is that we run into problems with the dot notation for style properties like `background-color`. The problem is that `e.style.background-color` looks like we are subtracting the variable `color` from the variable `e.style.background`. The solution implemented in

⁵ Some developers consider it best practice to use square bracket notation with the style object (e.g. `e.style["backgroundColor"] = "red"`). It probably doesn’t matter which one you use, but you will often see it written this way.

most browsers is to convert hyphenated-property-names to **camelCasePropertyNames** (e.g. `border-radius-top-left` becomes `borderRadiusTopLeft`).

For example:

```
var e = document.getElementById("myElement");
e.style.backgroundColor = "red";
```

Try It Yourself

1. Create a web page with a single “hello world” paragraph. Create a script on the page after this element that prompts the user for a color, and then sets the color of the “hello world” element according to what they typed. Why do you have to create the script after the `<p>` element? What happens if you place it earlier in the file?
2. Modify the script from question 1 so that it sets other style properties of the element (background color, border, border-radius, width, etc.)

4.3.3 Changing an Element's `class`

Sometimes you may want to change a large number of property-value pairs at once. In this case, a better option might be to define styles for two different classes, and then change the `className` field of the element, like this:

```
document.getElementById("myElement").className = "newClassName";
```

This will automatically update the style of the element to match its new class.

Try it Yourself

In the examples package, you will find two files that show you how to alter class names:

additionQuiz2.html

- Presents a simple addition quiz to the user, and changes the `className` field of an element when giving feedback.

basicFormWithJavaScript2.html

- When the user clicks in the text field, the content and the `className` of the text field element change

TODO: Update these examples

5. Functions and Events

Up to this point, you have only been using JavaScript in an imperative programming style, running scripts as the page loads. This is pretty limited. Usually you want to make changes or interact with the user *after* the document is loaded, not *while* it's loading. To do this, you need to declare **functions** that will be created when the page is loaded, but can be executed at another time in response to specific **events**. That's what this chapter is about.

5.1 Defining a Function

JavaScript functions are a lot like the methods, procedures, and functions found in most other languages.

Here's an example of a function definition. This one has no parameters and no return value.

```
<script type="text/javascript">
  function hello() {
    alert("Hello, world!");
  }
</script>
```

Functions can be defined anywhere on the page, as long as they are inside a `<script>` element, but it's good practice to define your functions at the top of the document, inside the `<head>` element, and it's even better practice to define them in a separate file (usually with a `.js` extension) and then load that file with a `<script>` tag that looks like this:

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

5.1.1 Responding to an Event

No matter where you define a function, it won't execute unless you **call** it. Most JavaScript function calls are triggered in response to **browser events**. An event is an important milestone in the life of a page, such as when an element finishes loading, when the user resizes the browser, when an element is clicked, when the mouse enters or leaves the page, or when a key is pressed.

Java Connection

Methods. JavaScript functions are very similar to Java methods, although as we shall see, there are also some very important differences.

Java Connection

Events. In Java, you define listener classes which contain event handlers (i.e. methods). In JavaScript, you just define the handlers (i.e. functions).

Any HTML element can be the source of the event, and by default most events are ignored. But for every event on every element, you can give the browser a JavaScript statement to execute when the event happens. Here's an example:

```
<p onclick="alert('You clicked me!');"> Click Me </p>
```

The HTML code above creates a paragraph element with the text “Click Me”. The `onclick` attribute specifies a JavaScript statement that should be executed when a **click event** happens (i.e. when the user clicks on the paragraph).

Note that the JavaScript statement has to be enclosed in quotation marks, like any attribute value, and that because of this, the `alert` command has to use single quotes around the text in order to be syntactically correct. Alternatively, since HTML and JavaScript both treat double and single quotes interchangeably, you could have written this:

```
<p onclick='alert("You clicked me! ");'> Click Me </p>
```

You can respond to an event with your own function (like the `hello` function created in the last section), in just the same way.

```
<h1 onclick="hello();">Click Me</h1>
```

Other JavaScript mouse events are listed below.

ondblclick

- Triggered when element is double-clicked

onmousedown / onmouseup

- Triggered when a mouse button is pressed or released over the element)

onmouseover / onmouseout

- Triggered when mouse is moved over / moved out of the bounding box of an element

onmousemove

- Triggered every time the mouse moves (even one pixel) while it is over the bounding box of an element

Try it Yourself

The files **functions1.html** and **functions2.html** in the examples package show the `hello` function in action as above. In `functions1.html` the function is defined on the page, while in `functions2.html`, it is defined in a separate file and loaded using a `<script>` tag.

Some other fun events to try are **onmouseover** and **onmouseout**. Try changing **function1.html** to use these events instead (note that if you don't set the *width* of an element, it will span the whole width of the browser and will cause the event to trigger a lot).

5.1.2 Clicking on Buttons

Clicking on text is all very well, but we should really be clicking on buttons. You can define a button like this:

```
<input type="button" value="Press me"/>
```

To make it respond to clicks, just add an *onclick* attribute, like this:

```
<input type="button" value="Press me" onclick="hello()"/>
```

See **functions3.html** in the examples package for another example of how to use a button.

Try it Yourself

Create a web page that asks the user a true or false question and give them a button to click for each option. Display a success or fail message somewhere on the page in response to their click.

Java Connection

Parameters and Return Values.

JavaScript functions have them, just like Java methods. But because the language is weakly typed, you don't have to define parameter or return types ahead of time.

5.2 Parameters, Global Variables, and Return Values

Just like Java methods, functions can have **parameters** and **return values**. But in JavaScript, there is no need to declare any types for anything.

```
function foo(x, y, z) {  
    return x+y+z;  
}
```

The function above takes three parameters and returns the result of “adding” them together. What gets returned depends on the type of what was passed in. Here are some examples...

```
foo(1,2,3)      → 6  
foo(1,"2",3)    → "123"    // why?  
foo(1,2,"3")    → "33"     // why?
```

Variables declared inside functions or listed as parameters are usually **local** to the function. Variables declared outside of functions are always **global**, which means they are accessible to all functions on the page.

```

var current = 0;                ← current is global
function count(inc) {          ← inc is local
    var newVal = current + inc; ← newVal local
    next = newVal;
    alert(next);
}

```

In the above example, `current` is a global variable while `inc` and `newVal` are local variables. **But watch out!** If you use a variable inside a function without declaring it, as shown below, it will automatically be created with global scope.

```

var current = 0;                ← current is global
function count(inc) {          ← inc is local
    newVal = next + inc;        ← newVal is global now
    next = newVal;
    alert(next);
}

```

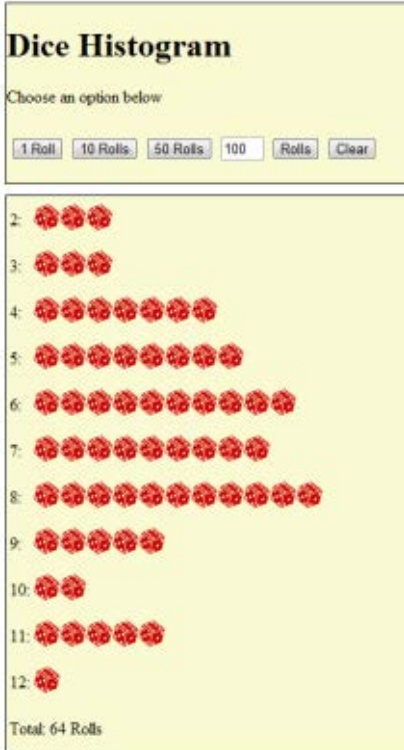
This can cause confusion, not to mention bugs. But the problem goes away if you **always declare all variables using the `var` keyword**. The file `functions4.html` in `Chapter6Examples.zip` contains an example of a function with a parameter that accesses a global variable.

Try it Yourself

Create an HTML page that gets a message from the user (using a `prompt` dialog), stores it in a global variable, then gives the user a button to display the message once, and another button to display it 10 times. Write a JavaScript function with one parameter `n` that displays the message the user entered `n` times inside a `<p>` element on the page. You should leave messages there once you have written them, so if they press the “10 times” button 3 times, the page should show 30 copies of the message.

Extension: Number each occurrence of the message starting at 1.

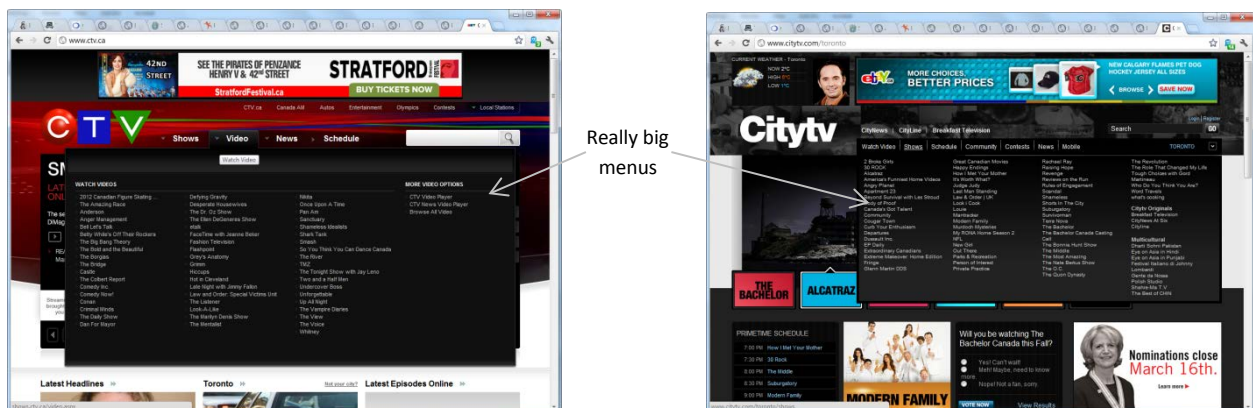
Mini-Project



1. Create a web page with one button. When the user presses the button, it should “roll” two dice (i.e. generate random numbers from 1 to 6) and display them on the page. (Hint: Use `Math.floor(Math.random() * 6 + 1)` to simulate each die.)
2. Build a histogram for the user like the one at left, showing the number of times each roll comes up. You can use the die image provided for this in the examples package. (Hint: Use a table, alter the `innerHTML` of the appropriate `<td>` element for each roll).
3. Add a “clear” button.
4. Add “roll 10 times” and “roll 50 times” button.
5. **For the adventurous:** Add a text input field with another button so that they can choose how many times to roll. (We haven’t covered this yet, but you can get info in the HTML section of w3schools under “forms”, and also in the JavaScript section under JS Validation).

5.3 Application: Drop-Down Menus

You see drop down menus all over the web, on sites designed for both desktop and mobile viewing. Lately the trend on desktop sites seems to be towards “mega-menus” that are rich with structured information and links, as shown below:



To see how this is done, go to a site with a menu like this, right click and “inspect element”. You’ll find it’s just a `div` element, positioned cleverly, and then shown and hidden using the CSS visibility

property from JavaScript. Sometimes the style of the button also changes to match the drop down menu style and indicate which menu is being displayed.

The file **dropDownMenu.html** in the examples package shows an example of drop down menus in action. Note that this file will not display properly on Internet Explorer, but there are two “IE Safe” versions that should look fine (these will be discussed later).

Things to note about dropDownMenu.html:

- CSS and JavaScript are stored in separate files
- All div elements have `absolute` position with `left` at `0px`, but with the `margin` property to set to `auto`. This keeps them centered even when the screen is resized.
- Both the buttons and the menus have `onmouseover` and `onmouseout` properties (why?)

Try it Yourself

The file **sideMenus.html** in the examples package contains some starter code. When you view this file you will see what the page is supposed to look like when the user mouses over menu 1. See if you can add JavaScript and make other changes so that when the page starts, the menus are not highlighted, but menu 1 pops up when the user mouses over the button and hides itself again when the user leaves the menu.

If you have time, try to add the other two menus as well. The content of these menus is not important.

6. Arrays in JavaScript

Let's take stock. You now have all the tools you need to write a simple web app. In a web app, the HTML and CSS define an **interface** and the JavaScript functions provide **functionality**, usually in response to various events during the life cycle of the page. For richer user interaction, you really need to make use of HTML form elements. That's the subject of Chapter 7. But making use of form elements often involves simple array processing. So before we go there, I need to take you on a slight detour to talk about arrays in JavaScript.

Like most languages, JavaScript also contains support for arrays. In many cases the code for processing arrays will look very familiar. For example, any AlmostNovice Java, C, or C++ programmer ought to be able to describe the effect of this JavaScript code on the array `a`:

```
for(var i = 0; i < a.length; i++)  
    a[i] = a[i] * 2;
```

Things are only slightly different in JavaScript because of the consequences of the language being weakly typed.

Java Connection

Arrays. JavaScript arrays are a lot like Java arrays, except: 1. You don't need to specify the size; 2. You can mix data types in the same array; and 3. The array indices don't have to be contiguous.

6.1 Consequences of Weak Typing

The first consequence is that **you don't have to declare a type for an array**. The following pieces of code show two different ways of creating a generic empty array.

```
var a = [];  
var a = new Array();
```

You can also create an array with some initial contents, like this:

```
var initializedArray = [43, "hello", -2.5, true];
```

Notice that this array contains elements of three different types (Number, String, and Boolean). This brings us to the second consequence of weak typing: **you can mix elements of different types in a single array**.

In fact, **you can even have arrays as elements of arrays**, like this:

```
var strangeArray = [6, [5, 3, -2], "JavaScript"];
```

The `strangeArray` has three elements. `strangeArray[0]` is 6, `strangeArray[1]` is the array `[5, 3, -2]`, `strangeArray[2]` is "JavaScript", `strangeArray[1][0]` is 5, and so on.

And a final, perhaps not so obvious, consequence of weak typing is that you can arbitrarily extend the length of an array, like this:

```
var a = [];  
a[0] = -23;  
a[1] = 45;  
a[9] = -1;
```

After the code above has executed, we have an array with indices ranging from 0 to 9, but not all indices are defined. This is a powerful use of arrays, but it can get messy.

Fortunately, `undefined` for an array element is treated the same as `false`. So if you are ever not sure whether an element has been defined, you can test it in an `if` statement like this:

```
if (a[3])  
    alert("element 3 defined");
```

Try it Yourself

Create a page with a script that defines an array and puts the values "Hello", "World", and "!" into random array locations between 1 and 10. Then prompt the user for three integers, and check to see if they have found the array locations. Give them a score out of 3 depending on how many they found and show them the values they uncovered.

TODO: Expand 6.1 with more examples and exercises

6.2 Arrays and the DOM

Now that you know about arrays in JavaScript, you can use several other powerful `document` functions, including `getElementsByClassName` and `getElementsByTagName`. These methods behave just like `getElementById`, except that they can return more than one object, because there can be multiple elements with the same class name or of the same tag type in the document. In both cases, a (possibly empty) array of objects is returned.

Try it Yourself

Go to your favorite Wikipedia page, open the JavaScript console of your browser, and execute the following two statements:

```
var a = document.getElementsByTagName("p");  
for (var i=0; i<a.length; i++)  
    a[i].innerHTML="JavaScript Rules!";
```

Did the code do what you expected? Identify another common element on the page (in Chrome, you can right click and choose “inspect element” to find out what type it is), and try making them all red.

7. Form Input

A web application, whatever job it is designed to do, needs to be able to have a dialog with the user. In previous chapters you saw at some simple ways of interacting with the user on the client side using JavaScript popups, HTML buttons, and events like `onclick`.

HTML forms not only provide a richer set of tools for getting user input, but they allow the information collected to be used entirely on the client side (in a stand-alone JavaScript application) or sent back to the server to be stored and/or used to create custom content for the user (with PHP or other server-side scripts).

You may not have been introduced to forms in your first HTML course, so this chapter introduces the basics of HTML Forms, and then show you how to use JavaScript both to validate form data before sending it to the server, and also to create web applications that operate entirely on the client side.

7.1 Form Basics

Any web page can include one or more `<form>` elements. Each `<form>` element can contain any number of `<input>` elements along with other elements for formatting (e.g. a `<form>` element might contain a `<table>` so that the form is laid out nicely, as shown in the screen capture below). If you are going to access the form in a JavaScript function (see section 7.2 below) you will need to specify a *name* attribute for the `<form>` element as well.

```
<form name="myFirstForm">
  <!-- your form goes here -->
</form>
```

The image on the right shows a simple HTML form. All the input elements were created using the `<input>` tag. Each input element must have a `type` attribute and most should have a `name` attribute (used to access the data typed by the user).



Here's the code for the text input element in the above example, shown along with other surrounding tags for context:





```
<tr><td>User Name:</td><td><input type="text" name="username"/></td></tr>
```


Try it Yourself

You can find the full code for this example in the file **BasicForm.html** in the examples package. Take a moment to load it up and make sure you understand how it was put together.

7.1.1 Possible <input> Types

Here's an almost exhaustive list of the input types along with some notes on how to use them.

Type Name	Appearance & Notes
<code>type="text"</code>	<div></div> <ul style="list-style-type: none">- You can use the <code>value</code> attribute to fill in a default value, like this: <pre><input type="text" name="name" value="Choose User Name"/></pre>
<code>type="password"</code>	Looks the same as a text field, but hides what the user types.
<code>type="radio"</code>	<div></div> <ul style="list-style-type: none">- Radio buttons are typically grouped to allow the user to check at most one at a time.- Radio buttons that belong to the same group should have the same <code>name</code> attribute.- If you want a radio button to be checked by default, use the <code>checked</code> attribute, as shown below. Note that <code>checked</code> does not require a value. <pre><input type="radio" name="gender" checked/> Female
 <input type="radio" name="gender"/> Male
</pre>
<code>type="checkbox"</code>	<div></div> <ul style="list-style-type: none">- Checkboxes can be grouped, but unlike radio buttons, the user can check always allow multiple boxes in the same group.- Comments for radio buttons also apply here.
<code>type="reset"</code>	<div></div> <ul style="list-style-type: none">- This button will clear the form. It usually does not need a <code>name</code> attribute.- To change the text on the button, use the <code>value</code> attribute, like this <pre><input type="reset" value="Clear"/></pre>

Type Name	Appearance & Notes
<code>type="submit"</code>	 <ul style="list-style-type: none"> - This button will cause the information on the form to be submitted to the server as an HTTP request. We will learn how to use this effectively later in the course. At the moment the button will just trigger a reload of the page (which is why when you press it the form seems to clear). - You can change the text on the button in the same way as for the <code>reset</code> button.

- Use this to create a generic button element with text set by the `value` attribute.
- Use this element to trigger JavaScript in situations not covered by `submit` or `reset`.
- **Note:** The `<button>` element was introduced in Chapter 6, but this element is not for use in forms because its behavior varies across browsers. Use this `button` input element instead.

TODO: HTML5 form types

Try it Yourself

More information and interactive examples of these and other form elements can be found on **w3schools**. Below are some quick exercises to do with form layout. In the next section, we will use JavaScript to add functionality to each of the forms you create.

1. Currency Converter

- Create a simple form to allow the user to convert Canadian dollars to another currency of your choice.

2. Shopping Cart

In a web application, the goal is always to do as much processing as possible on the client side. For example, on a shopping cart confirmation page, the user can change quantities of items, choose options and then click a “recalculate” button to get their total before submitting the information to the server. The recalculate step should be handled by on the client side with JavaScript.

Create a shopping cart confirmation page (this does NOT have to be fancy, and you should not worry about style for this, but if you like you could try modifying your shopping cart for the first assignment).

Here's what the shopping cart should do:

- Present 2 items along with their prices, with text fields to show the current quantities.
- Allow them to enter a promotional code in a text field.
- Allow them to choose standard delivery (\$4.99) or expedited delivery (\$19.99)
- Allow them to separately select gift wrapping (\$9.99), insurance (10% of the purchase price before tax and discounts), and rewards membership (free).
- Present the tax (13%) and total at the bottom.
- Give them 3 buttons: reset, recalculate, and submit

7.2 Forms with JavaScript

Any form element can cause JavaScript commands to be executed in response to events. The most common events for use with forms are:

<code>onclick</code>	Triggered when the user clicks the element
<code>onchange</code>	Triggered when the value of the element changes
<code>oninput</code>	Triggered whenever a text area or text field is typed into
<code>onmouseover</code>	Triggered when the mouse pointer moves over the element.
<code>onmouseout</code>	Triggered when the mouse pointer leaves the element.
<code>onfocus</code>	Triggered when the user clicks or tabs to an element. For example, when the cursor is in a text box, it means the textbox has focus. Typically only form elements and hyperlinks can have the focus, and only one element at a time can have it.
<code>onblur</code>	Triggered when the element loses focus.
<code>onsubmit</code>	Triggered when the submit button is pressed, or when enter is pressed. Sends an HTTP request to the server with the contents of the form.

7.2.1 Responding to Events

If you want to do some JavaScript processing in response to any of the events, you must include the event name as an attribute, like this:

```
<input type="radio" onclick="alert('clicked!');return true;" />
```

The JavaScript code in bold will be executed when this radio button is clicked. Notice that the string passed to the alert function is written with single quotes instead of double quotes. This is so it can be placed within the double quotes required by the `onclick` attribute without causing a syntax error.

Also, you should notice that the `onclick` code contains two JavaScript statements separated with semicolons. These statements will be executed in the order that they appear when the button is clicked.

In fact, the code for an event can be thought of as a kind of **anonymous function**.

- It's like a **function** because it can contain a block of JavaScript statements, including loops, if statements, etc.
- It's **anonymous** because it has not been given a name.

Like a function, the quoted code can return a value. In this case, the return value will not have an effect unless you return the value `false`. If you do that, it will prevent the input element from taking any other actions that it would normally take (submitting or resetting the form, checking or unchecking a checkbox, etc.)

Here's an example that will prevent the user from ever being able to check or uncheck a radio button.

```
<input type="radio" onclick="alert('Not Allowed');return false;" />
```

Here's another example in which a reset button will only reset the form if the user clicks OK. (If you take away the word `return` the popup will no longer prevent the reset. Why?)

```
<input type="reset" onclick="return confirm('Are you sure?');" />
```

Try it Yourself

The file **basicFormWithJavaScript.html** in the examples package shows how to use `onclick` in this way. In particular, see the code in the `reset` button.

7.2.2 Accessing User Input

In the above examples, we are using pre-defined JavaScript functions to respond to events. But to create a client-side web application, you need to be able to access and process the user input that is stored in the form. You do this by defining your own function and then accessing the form information stored in the DOM.

Try it Yourself

Load up **basicForm.html** from the examples package and try some of these expressions and commands in the JavaScript console of your browser as you read along.

Your browser organizes all your form elements for you in a `document` field called `forms`. The `document.forms` object has a field for every form name (e.g. `document.forms.myForm` will get you the form named "myform"). Each of these forms has a field for every named element in the form (e.g. `document.forms.myForm.myField` will get you the element named "myField" from the form named "myForm").

Recall that when you access an object's fields, you can use the `.` operator or square brackets (i.e. `object.field` or `object["field"]`). It is considered good practice by many JavaScript programmers to use the square bracket notation to access named elements from the DOM. This is because sometimes the names can contain characters that would cause problems using the `.` operator.

So to access a named form element, use the following expression:

```
document.forms["formName"]["elementName"]
```

In the above, "formName" is the value of the *name* attribute of the <form> element you want to access, and "elementName" is the value of the *name* attribute of the <input> element.

If the <input> element is of type *text* or *password* you can access its *value* attribute directly (it's a string).

```
if(document.forms["form1"]["userName"].value == "Sam") // check value
    alert("Hi Sam!");
```

```
document.forms["form1"]["userName"].value = "Joe"; // set value
```

If the form element is of type *radio* or *checkbox*, it's a little more complicated because what you get back from `document.forms["formID"]["elementName"]` is an array of elements. You have to specify the index number of the particular element in the group.⁶ The first radio button in the group will be index 0, then 1, etc. You can access the checked field to get a Boolean value (*true* if the button is checked, *false* otherwise).

Java Connection

Objects. There are two ways of accessing an object's fields in JavaScript. Like Java, you can write `objectName.fieldName`. Unlike Java, you can also write `objectName["fieldname"]`.

```
if(document.forms["form1"]["interests"][0].checked) // check value
    alert("I knew you liked gardening!");
```

```
document.forms["form1"]["interests"][0].checked = true; // set value
```

Try it Yourself

The file **basicFormWithJavaScript.html** in the examples package shows how to use `document.forms` in this way. In particular, see the code for the functions `checkSkiing`, `clearName`, and `fillName`.

The `checkSkiing` function is used to nag the user about whether they really want to uncheck skiing as an interest. It returns a value that allows or prevents the unchecking of this option (but it stays silent if it sees that they are checking it).

The `clearName` and `fillName` functions are used to remove/replace the "Choose a User Name" text in the User Name Text box when appropriate. (It would be nice to also grey-out the "Choose a User Name" text somehow but we will leave this to Chapter 8.)

⁶ Or you can define unique *id* attributes for your radio buttons and checkboxes and access them using these attributes.

7.2.3 Building Client-Side Web Applications

Take a look at the **additionQuiz.html** file in the examples package.

This program is an example of a simple but complete client-side web application. It contains two JavaScript functions. The first function creates and presents a new question to the user and clears out any old answers or feedback, while the second checks the user's answer and gives feedback. Each of these functions is called by a different button press.

Notice the use of the `` elements with unique ids. They allow the values for the two operands to be set in a JavaScript function without rewriting the entire paragraph element that contains them.

Notice also the use of the `onload` event in the `<body>` element. This event is triggered after the `<body>` is finished loading, and calls the function to set the first question.

Try it Yourself

1. Get the currency converter working.

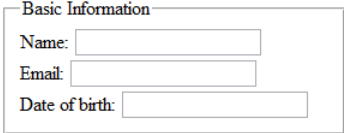
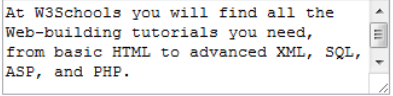
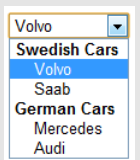
- When the user clicks "convert" it should display on the page the result of the conversion (you can look up the rate on line or just make something up if you like).

2. Get the shopping cart working.

- When the user clicks "reset", they should be warned that the form is about to clear and get a chance to stop it.
- When the user clicks "recalculate", a function should be called that checks the quantities of the products and the various options the user has chosen, then displays the new total price with tax. Don't forget to use `Math` methods to round to two decimal places.

7.3 Advanced Form Features

The "HTML Forms" entry on w3schools highlights most of the basic elements above, but it also contains links (if you scroll all the way down) to a number of other elements that can be used to make your forms better looking and more functional. Here's a list of these "advanced" form features. You are encouraged to look them up on your own and use them in your assignments and exercises.

Element	Appearance & Notes
<code><fieldset></code> and <code><legend></code>	 <ul style="list-style-type: none"> - Use these elements to visually group different areas of your form as shown above. - You can also apply different styles to the different field sets.
<code><label></code>	<ul style="list-style-type: none"> - Used with checkboxes and radio buttons so that the user can select the button by clicking on the text label for the button, not just on the button itself on the label. - This simple addition can make your form much more user friendly.
<code><textarea></code>	
<code><select></code> , <code><option></code> and <code><optgroup></code>	
<code><input type="image"></code>	<ul style="list-style-type: none"> - Creates a submit button using an image. - Note that any image can be made into a button by adding an <i>onclick</i> attribute to the <code></code> tag. But submit is a special button so it requires this special method.

Try it Yourself

The **FormDemo.html** file in the examples package shows these elements and some new HTML5 elements in action along with simple JavaScript functions for accessing their contents.

You should also see the file `additionQuizHTML5Output.html` as well as `w3schools` for the use of the new HTML5 `<output>` element.

Try it Yourself

1. Make the currency converter better.

- Use fieldsets, and images to improve the look of the currency converter.
- Add drop down lists to allow the user to pick the “from” and “to” currency, then perform different calculations depending on what they choose.
- Make it robust so that if they enter a non-numeric value it recovers gracefully.

2. Make the shopping cart better.

- Use fieldsets, labels, images and/or drop down menus to improve the look and feel of the shopping cart confirmation page.
- Add a text area for shipping instructions.
- Make it robust so that if they enter a non-numeric value it recovers gracefully.

3. Finish the dice histogram from previous chapters.

- Now you can add a text field to let them roll as many times as they want in one go.

Dice Histogram

Choose an option below

1 Roll 10 Rolls 50 Rolls 100 Rolls Clear

2- [three dice icons]

3- [three dice icons]

8. Autonomous Client-Side Web Apps

You almost have all the tools now to create a true client-side web app. You define an interface in HTML and CSS, then add functionality to it in JavaScript. But there are two things that make these apps less useful than apps written in, say, Java.

1. You can't store any information about the user to access on their next visit.
2. If the user loses their connection or moves their mobile device outside the service area, your app is no longer accessible.

But HTML5 is bringing with it two very important new tools for building client-side apps that, once loaded for the first time, can offer rich functionality and user interactions without ever bothering the server again. This is great news, particularly for developers of mobile web apps where bandwidth and limited connectivity have in the past been barriers to delivering apps entirely in a browser.

The new **web storage** system in HTML5 replaces the user of cookies as a flexible way to store data on the client, and the new **app cache** allows a user to “reload” a web app even while off line. These two new capabilities combined allow us to create fully functional apps in a web browser that do not require an active Internet connection. That's what this chapter is about.

8.1 Web Storage

Web developers often want their sites to remember users, maintain their settings between page loads, and even keep local copies of the data they entered for the next time they visit. In the past, this was done through by setting “cookies” on the user's hard drive.

Cookies are small pieces of data (maximum 4KB) that are stored on the client side, linked to a particular URL. They are accessible through JavaScript on the client side, and are sent to the server along with every HTTP request. They are typically used to store a unique identifier for each user or session, and then the bulk of a user's data is stored on the client side using this unique identifier. This is how Facebook recognizes you when you return to its site.

As an alternative to cookies, HTML5 offers two new JavaScript objects: `localStorage` and `sessionStorage`. They have the following properties:

- Each domain name has its own `localStorage` object.
- Each domain name has one `sessionStorage` object for each open tab.
- These objects can store up to 10 MB of data.
- The data is not automatically sent to the server.
- The data in `sessionStorage` is automatically deleted when the browser tab is closed.
- The data in `localStorage` is never automatically deleted.

Try it Yourself

If you want to look at the `localStorage` and `sessionStorage` for any given site, open the JavaScript Console and type `localStorage` or `sessionStorage`. Or alternatively, go to the “Resources” tab of the developer tools on chrome and you can see the contents of these objects there.

As you continue to read this section, try out the commands in the JavaScript console.

All data stored in `sessionStorage` and `localStorage` must be String data. Each item of data is stored under a defined “key”, and is accessed the same way as any other object, with square bracket notation or dot notation.

For example, the following stores two pieces of string data in local storage:

```
localStorage[ "name" ]="Bob" ;  
localStorage[ "number" ]="22342" ;
```

These two pieces of data will both continue to be available to the domain that set them, even if the user restarts the machine. They can be accessed like this:

```
localStorage[ "name" ]           ← returns "Bob"  
localStorage[ "number" ]        ← returns "22342"
```

And they can be changed any time, or they can be deleted individual like this:

```
localStorage.clear("name");    ← clears the "name" field
```

You can also clear the entire local storage area for the entire domain like this:

```
localStorage.clear();          ← clears everything
```

The `sessionStorage` object works in exactly the same way, except that the data is only accessible in one tab and will automatically clear when the user leaves the page.

8.1.1 Initializing Web Storage

Suppose you are counting how many times a user visits your site. On the user’s first visit, you will have to create the counter item in `localStorage` and set it to 1. On subsequent visits, you should not re-create the “number”, rather you should access it and increment it. You need a way to figure out if the user has been here before.

The code below accomplishes this by taking advantage of the fact JavaScript treats `undefined` the same as `false` in a Boolean expression.

```
if (localStorage["counter"]) {  
    localStorage["counter"] = parseInt(localStorage["counter"]) + 1;  
} else {  
    localStorage["counter"] = 1;  
}
```

Try it Yourself

Load up **webStorageDemo.html** in the examples package to see the above code in action. Note the use of the `onload` event in the `<body>` element. This makes sure that the `count ()` function will run once on each page load, but not until after the entire `<body>` is loaded.

8.1.2 Advice on Sharing Web Storage

The one drawback to `localStorage` is that if your site is on a domain that has multiple users (like `mobile.sheridanc.on.ca`), somebody else's web app can access, clear, and overwrite your `localStorage` data. There is nothing that can be done to avoid this, but it might be a good idea to name your data in a way that is likely to be unique.

For example...

```
localStorage["App2938475SamScott_vehicle"]="Aveo";
```

... is less likely to be tampered with than...

```
localStorage["vehicle"]="Aveo";
```

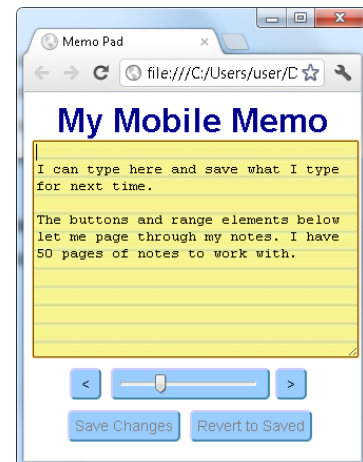
Try it Yourself

1. Experiment in the developer console.
 - Load up a web page (e.g. google) and go to the developer console.
 - Store some data in `localStorage` and `sessionStorage` as shown above.
 - Surf around the site, and check to make sure your data is still there
 - In the same browser tab, go to a new site (e.g. facebook) and check to see if the data is there.
 - Go back to the first site and check again.
 - Open a new tab, go to the same site as the first and check again.
 - Close the browser, re-open it, go to the same site and check again.
 - Clear the local storage object.
2. Write a simple standalone web app with two pages.
 - On page one, have the user fill in a form with their name and the color shirt they are wearing. If they have been here before, you should fill in their name automatically. Store their name in `localStorage` and their shirt color in `sessionStorage`. Provide a link to page 2.
 - On page 2, you should greet the user by name and tell them whether you like their shirt or not, based on the color they entered last time (e.g. write an if statement that only says you like their shirt if they say it was red). Provide a link back to page 1.

Mini-Project

Write a mobile “memopad” application. This application should allow the user to type into a text area. The text should be stored in `localStorage` and displayed when they return for them to read or modify. Make it look professional (like a real app you might see on a mobile phone).

As an added challenge, allow 50 pages of memo storage with buttons or other form elements to allow them to flip through the pages.



8.2 The App Cache

With `localStorage` you can create a standalone web app like the mobile memo pad above. This can sometimes be preferable to creating a native app. Native apps have to be created and maintained for each phone type, but mobile apps can be written and maintained once in JavaScript and then run on any device with a browser.

This could be highly convenient, except for one thing. The user has to connect to the Internet and send an HTTP Request in order to user the app. It would be much better if the app could be stored locally like a native app. And that’s exactly what the HTML5 app cache does.

To use the app cache, you need two things:

1. An app cache “manifest” file.
2. A `manifest` attribute on the `<html>` tag of the site’s front page.

8.2.1 The `manifest` Attribute

The `manifest` attribute is part of the HTML tag. It specifies the location of a manifest file, like so:

```
<html manifest="myManifest.appcache">
...
</html>
```

The name of the manifest file doesn’t matter, but the `.appcache` file extension is recommended so it can be easily identified. You can also include a full file path.

The presence of a `manifest` attribute in an HTML file will cause that file to be automatically stored locally to be accessed when the user is offline. The contents of the manifest file specify which other files should also be stored locally.

8.2.2 The Manifest File

Here is a simple manifest file:

```
CACHE MANIFEST
#May 29, 2012, 6:34 pm
js/memo.js
css/memo.css
images/116_2.jpg
```

The first line is required. The line that starts with a # character is just a comment. The remaining lines specify which files on the site are to be cached (along with the HTML file that loaded this manifest). These are file paths relative to the current location.

You can also include a `NETWORK` section. This section specifies files that must be loaded fresh from the server every time. And you can also include a `FALLBACK` section that specifies a file to be loaded if a page is not accessible. You can read more about these options on w3schools.

Try it Yourself

You can only play with the app cache if you have access to a web server. If so, upload the files in the **appCacheExperiments** section of the examples package and try them out. Try accessing them each while on line, then sever your network connection and try again. Try to figure out why each one loads or does not load.

8.2.3 Updating the Manifest

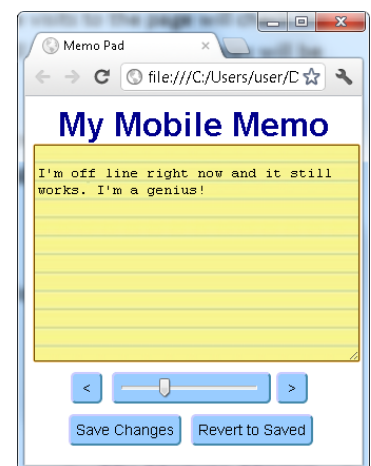
If your site uses the app cache, it will only be loaded one time. Future visits to the page will check the `appcache` file first. If it has not changed, none of the cached files will be loaded. Local copies will be used instead.

This is why when you update your site, it is crucial that you update your manifest file as well, even if it means just changing one comment line. If you don't do this, returning users will not see the updated site.

Try it Yourself

Finally, we can create fully autonomous client-side web-based apps for mobile and other users.

1. Update your two-page app from section 8.1 so that it still works if the user is off line.
2. Update your memopad app so it works off line as well.



9. The jQuery Library

Now you're writing some really powerful, standalone web apps. At this point you arguably have everything you need to do this. But there are easier ways to program in JavaScript. This chapter is about **jQuery**: a public domain JavaScript library that has caught on like wildfire and is changing the way JavaScript code is written.

I will give you the basic tools you need to use jQuery effectively, but I will leave a lot of the details for you to explore yourself. In section 9.2, I will also reveal some surprising facts about the true nature of JavaScript functions.

The jQuery library is publicly available for download at <http://www.jquery.com>.

The API for jQuery can be found at <http://api.jquery.com>. W3Schools also has some useful jQuery pages.

9.1 Selectors and Effects

The jQuery library brings the power of CSS **selectors** to JavaScript. A CSS selector is the first component of a CSS rule – the part that specifies which elements the rule applies to.

```
p, a:hover, #myDiv, span.movie {
    color:red;
    width:100%;
}
```

The CSS rule above uses the selector "p, a:hover, #myDiv, span.movie" to specify that it applies simultaneously to all elements of type <p>, any <a> element that the mouse is hovering over, the unique element with its id attribute set to "myDiv", and any element with a class attribute that includes "movie".

In plain old JavaScript, if you wanted to do make same changes to this diverse collection of elements (e.g. turn them all green) you would need to write expressions using `document.getElementById()`, `document.getElementsByTagName()`, and `document.getElementsByClassName()`, and you would need to write loops to apply the changes to the results of these expressions. But with jQuery you can use CSS-style selectors to do it all in one line, like this:

```
$("p,a:hover,#myDiv,span.movie").css("color","green");
```

9.1.1 The jQuery Sandbox

The following sections make reference to the **jQuery SandBox**, which you can find in the **examples** package. When you take a look at this site, you will notice that the jQuery library is included in the `js` folder, and that the `index.html` file loads the jQuery library using the following line:

```
<script type="text/javascript" src="js/jquery-1.7.2.js"></script>
```

This line causes all the code in the jQuery library to be included in this page. If you want to use jQuery in your own pages, you have to do two things: 1. download the library and put it somewhere in your project; and 2. Link to it in a `<script>` element as above.

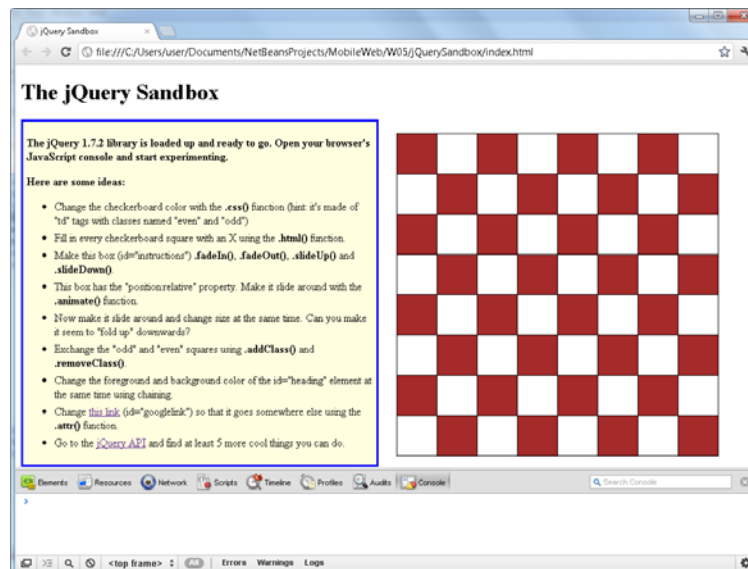
At the time of writing, 1.7.2 was the latest version of jQuery, but that might have changed by now. The latest version is always freely available at <http://www.jquery.com>, where you can also choose the “minimized” version of the library to save bandwidth.

Java Connection

Importing Libraries. Just like in Java, where you often have to import classes that are extensions to the base language, jQuery also has to be imported.

Try it Yourself

To use the sandbox, load it into a browser and open the browser’s JavaScript console. Because the `index.html` file loads the jQuery library, you can play with jQuery commands in this window.



9.1.1 The `$` Function

Everything you will do with the jQuery library will be based on the new `$` function it provides. The syntax of the `$` function is:

```
$("selector")
```

The jQuery **selector** works like a CSS selector, with a few extensions unique to jQuery. If you use the `$` function on its own, it will return an object of type **jQuery** that looks and behaves very much like an array of DOM elements.

For example, if you execute this command in the sandbox, you will get back a jQuery object containing all the `<a>` elements in the document.

```
$("a");
```

That object that is returned to you should look something like this:

```
[ <a id="googlelink" href="http://www.google.com" target="googlewindow">this link</a>,  
  <a href="http://api.jquery.com/category/effects/" target="bob">jQuery API</a> ]
```

Try it Yourself

Try the following in the jQuery SandBox and try to make sense of the results.

- Get all `<td>` elements with the `class` attribute set to "odd"
- Get all `<h1>` elements
- Get all `` elements
- Get the element with the `id` set to "instructions" as well as every `<tr>` element.
- Execute the jQuery command `$("*")`. What elements are in the resulting object?

The object you get back from the `$` function can be treated a lot like any an array. For example, this command will change the border style of the checkerboard:

```
$( "#checkerboard" )[ 0 ].style[ "border-style" ]="dashed";
```

The above statement is equivalent to the following:

```
document.getElementById( "checkerboard" ).style[ "border-style" ]="dashed";
```

Why is this here?

Similarly, you could put an X into every even checkerboard square with the following:

```
var squares=$( "td.even" );  
for (var i=0; i<squares.length; i++)  
    squares[i].innerHTML = "X";
```

9.1.2 Inner HTML, Attributes, and Styles

So far so good. But where jQuery really gets its power is in the **chaining** of other jQuery functions onto the end of the `$` function. Three such functions are `.html()`, `.attr()`, and `.css()`. When chained at the end of the `$` function using the `.` operator, these functions can be used to set the inner HTML, attribute values, and style properties of an entire set of selected elements all at once.

Here are some examples:

```
$( "#instructions" ).html( "No more instructions!" );  
$( "td.odd" ).css( "backgroundColor", "blue" );  
$( "a" ).attr( "href", "http://www.facebook.com" );
```

The above statements will delete the instructions text, change all odd checkerboard cells to blue, and change all the links in the SandBox to point to FaceBook.

You can also use each of these functions with one less parameter, and they will return the innerHTML, style property, or attribute of the first element in the matching set.

```
$( "#instructions" ).html();           ← returns innerHTML of first match
```

```
$( "td.odd" ).css( "backgroundColor" ); ← returns backgroundColor of first match
$( "a" ).attr( "href" );                ← returns href attribute of first match
```

You can also change the classes an element belongs to with the `.addClass()` and `.removeClass()` functions, and you can find out if it belongs to a particular class with the `.hasClass()` function.

The jQuery functions above all have counterparts in ordinary JavaScript (`.style`, `.innerHTML`, `.className`, etc.). But in many cases the jQuery versions are more powerful. For example, the JavaScript command below returns an empty string because the color property of this element is not explicitly set in the CSS file of the SandBox.

```
document.getElementById( "instructions" ).style[ "color" ]
```

However, the equivalent jQuery command, shown below, searches the hierarchy of CSS styles and returns the correct color `"rgb(0,0,0)"` which it retrieves from the default browser style.

```
$( "#instructions" ).css( "color" )
```

And it doesn't stop there! Each chained jQuery function returns a copy of the original jQuery object. This means they can be chained one after another as many times as you want.

So if you want to change both the foreground color, background color, and inner HTML of the SandBox instructions, you can do it in one line, like this:

```
$( "#instructions" ).css( "backgroundColor", "red" ).css( "color", "rgb(255,255,255,0.5)" ).html( "Hello, World!" );
```

There is no limit to how many functions you can chain together in this way.

Try it Yourself

Try each of these commands in the console window of a browser displaying the SandBox. Note that you can always reload the page to get it back to its initial state between commands.

- Use the `.css()` function to change the checkerboard border style to dashed.
- Use the `.html()` function to put an "X" in every even checkerboard square.
- Use the `.addClass()` and `.removeClass()` functions to change the odd squares to even.
- Use the `.html()` function to change every link on the page to read "this is a link"
- Change the contents and color of all `<td>` elements in one line.
- Change the class of even to odd and odd to even in two lines. Can you do it in one line?

The following require changes to the `index.html` file of the sandbox.

- Put a button on the sandbox page that, when clicked, toggles the checkerboard from white and red squares to brown and black squares, and back again.
- Put a second button on the sandbox page that prompts the user for two numbers, then moves the instructions panel to those coordinates.

9.1.3 Effects and Animations

The jQuery library also contains functions to make various effects and animations easy. These functions can be chained after the `$` function just like the ones discussed in the previous section. For example:

```
$("td.odd").fadeOut();
```

The above will fade out all the odd checkerboard squares in the SandBox, and then remove them from the table by setting their `display` property to `"none"`. (Note that to be at its most effective, the `.fadeOut()` function should really only be applied to elements with absolute positioning).

I will leave it to you to explore most of these effects. You can find information on all of them at <http://api.jquery.com> in the “effects” section. Read the “basics”, “fading” and “sliding” sections. Click on each function name for an explanation. You can also go to w3schools and look in the Learn jQuery section for more help.

Note that most of these functions have optional parameters. For example, the page for the `.fadeOut()` function has the following header:

```
.fadeOut( [duration] [, callback] )
```

The square brackets mean that the duration and callback parameters are optional. If you call this function with no parameters, it will use default values. You can also set the duration if you call it with one parameter, or set both duration and callback if you call it with two parameters. (I will explain more about the callback in the next section).

There is also a generic animation function called `.animate()`, which is explained in the “custom” section of “effects”. This function needs at least one parameter, specifying a set of CSS properties and values to “move towards”. Any of the specified properties that can be animated (usually numeric properties) will be animated.

```
$("#instructions").animate({left:"0px", width:"20%"});
```

The parameter on the above instruction is a little odd if you’ve never seen it before. The curly brackets indicate that it is actually a specification of a JavaScript object in JavaScript Object Notation (JSON). The contents of the curly brackets consist of a list of field names (unquoted) and values (quoted).

So `{left:"0px", width:"20%"}` creates a new object with the fields `left` and `right` set to the strings `"0px"` and `"20%"` respectively. Then this object is passed as the first parameter to `.animate()`. I will have a lot more to say about creating JavaScript objects in a later chapter.

Try it Yourself

1. Complete the sandbox exercises as laid out in the instructions panel.
2. Explore the jQuery API a bit more. Take a look at the “Attributes”, “CSS”, “Dimensions”, “Effects” and “Manipulation” sections. Find and try out at *least* 3 more interesting functions.

3. Go back to the memo pad exercise from previous chapters and tweak it so it makes effective use of jQuery.
4. Go back to the exercise in which you created menus that appear and disappear when buttons were clicked. Modify this to make good use of jQuery selectors and chained functions.
5. Go back to the “Catch the Rabbit” exercise from previous chapters (you can use your own solution or the sample solution). Using jQuery, change the game so that when you mouse over the rabbit, it slides to a new, randomly-generated location on the screen.
6. Take the **sliding tiles** starter code from the **exercises** package and add JavaScript (using jQuery) to implement a sliding tiles puzzle. When the user clicks a tile (i.e. an `<a>` element) that is beside the blank tile, it should slide smoothly into its new position using the jQuery `.animate()` function. Note that this code can be very tricky to write if you have never done anything like this before.

9.2 Events in jQuery and the Truth About JavaScript Functions

Up until now, you have probably been adding event handlers to elements by placing JavaScript code into the corresponding HTML attributes, like this:

```
<input type="button" onclick="myFunction()">
```

This is not considered best practice by most JavaScript programmers for at least three reasons.

1. It spreads your JavaScript code throughout the HTML file, making the code harder to maintain. It would be much better if you could add all your handler code in a single place.
2. It violates the clean separation between the structure of the interface (the job of HTML), the style of the interface (the job of CSS) and the functionality of your app (the job of JavaScript).
3. It causes the browser to create a new function with the code you put in the attribute as its contents. This is unnecessary and wasteful.

It is widely considered best practice to add event handlers in JavaScript after the document has loaded (e.g. in response to the `load` event of the `<body>`). But to be able to do that, you need to understand how functions work in JavaScript in a little more depth. Once you have this understanding, you will also be able to use the new jQuery `ready` event, as well as the `callback` parameters of the jQuery effects functions.

9.2.1 The Truth About Functions

The truth about JavaScript functions is that they are actually values, just like Strings, numbers, Booleans, and objects. They can be stored in variables, passed as parameters to other functions, and returned as return values from functions.

When you use a **function declaration** like this:

```
function foo (a, b, c) {  
    alert(a+b+c);  
}
```

What you are really doing is creating a variable named `foo`, and assigning a **function value** to it.

The following shows the assignment of a **function expression** to a variable. It is legal JavaScript code, and is more or less equivalent⁷ to the function declaration above:

```
var foo = function(a, b, c) {  
    alert(a+b+c);  
};
```

The above statement creates a variable named `foo`, and assigns to it an unnamed function with 3 parameters. Note that since this is an assignment statement, we put a semicolon at the end of it.

Java Connection

Methods vs. Functions. In Java, methods and variables are completely different entities (how would you describe the difference?) In JavaScript, function names are actually variable names, and functions are data.

Try it Yourself

1. No matter whether you use function declaration or function expressions, if you refer to `foo` in your program, you are referring to a variable. If you don't believe me, try the following either in a `<script>` element or in the JavaScript console of your browser:

```
function foo () {alert("hi");}  
foo;  
foo();  
alert(foo);  
alert(foo());
```

- ← Test your understanding!
- ← What is the difference between these four
- ← lines? Can you explain the results?
- ← The answer is in the footnote.⁸

```
foo = 5;  
alert(foo);
```

```
foo = function() {alert("hi");};  
foo;  
foo();  
alert(foo);
```

```
foo = "hi";  
alert (foo);
```

⁷ The main difference is that you can only use function declaration in certain contexts, but you can use function expressions in any context. Because it is limited and adds nothing to the language, many programmers avoid function declaration altogether and always use the function expression notation.

⁸ In the console window, `foo` returns the value of the variable. It's a function. But `foo()` calls the function so you will see the popup. On the other hand, `alert(foo)` will display a popup whose contents is the value of the variable `foo`. Finally, `alert(foo())` will first call the `foo` function, displaying a popup. Then it will create another popup to display the return value of `foo()`. Since there is nothing returned, it will display "undefined".

2. Load the file **eventHandlerTest.html** from the **examples** package into a browser. In the JavaScript console, type: `document.getElementById("test1").onclick`. What is the result? What do you think it means?

9.2.2 Adding Event Handlers

If you want to add event handlers to an element after it is created, you can do it by getting the element out of the DOM, creating a function, and assigning it to the attribute for the event handler, like this:

```
document.getElementById("test").onclick = function(event) {  
    alert("hello!");  
};
```

Or you can create the function (either through function declaration or function expressions) first, and then assign it, like this:

```
var clickHandler = function (event) {  
    alert("hello! ");  
}  
document.getElementById("test1").onclick = clickHandler;
```

← You could also use function declaration
← but only in a global context

Note that the functions above all contain an `event` parameter. This is a special object that represents the event that fired to trigger this function. You can use it to access information about the event, such as which DOM was involved (`event.target`), which mouse button was pressed (`event.button`), the location of the mouse (`event.clientX` and `event.clientY`), etc.

For more information on the JavaScript Event object, go to w3schools.com and click on **HTML DOM**, then **HTML DOM Objects**, then **HTML Events**.

Try it Yourself

1. Load the file **eventHandlerTest.html** from the **examples** package into a browser. Take a look at how the handler has been added to `test2`. When is the handler added? Could it be added sooner than this? Why or why not?
2. Add two handler functions to the message element using JavaScript (not HTML attributes). This function should respond to the `onmouseover` and `onmouseout` events. It should cause one of the other boxes on the screen to change color.

9.2.3 Event Handlers in jQuery

The jQuery library contains its own event management system that extends the regular HTML and JavaScript system, making it more robust and useful. In jQuery, event handlers are added to elements by chaining on the `$` selector function, like this

```
$(selector).event( handler );
```

In the above, `selector` is a normal jQuery selector, `event` is an event name (load, click, mousedown, etc.), and `handler` is a function. Here's an example of how to add the `clickHandler` function from 9.2.2 to the all `<div>` elements in the document.

```
$( "div" ).click(clickHandler);
```

The correct place to add handlers is either in the `load` event of the `<body>` element, or in the new jQuery `ready` event. Here's what this would look like.

1. Using the `load` event of the `<body>`.

```
$( "body" ).load( function() {  
    $( "div" ).click(clickHandler);  
    ...  
});
```

2. Using the `ready` event of the document object.

```
$(document).ready( function() {  
    $( "div" ).click(clickHandler);  
    ...  
});
```

Note that `document` is unquoted here. You're passing the DOM object to the `$` function, not a string.

The difference between the above two options is that the `load` event will not fire until all resources (images, etc) of the `<body>` are loaded. But the jQuery `ready` event will fire as soon as the DOM is ready. So unless you want to access images and other external resources that may not be loaded yet, the `ready` event is the one you should use.

Go to the jQuery API and click on **events** for more information on the events supported by jQuery. The most important categories of events are keyboard, mouse, and form events, but you should take a look at the others as well.

Try it Yourself

1. Load the file **eventHandlerTest.html** from the **examples** package into a browser. Take a look at how the handler has been added to the table cell elements. Change the file so that the handler is added on the `load` event of the body.
2. Add a `<textarea>` element to the HTML. Using jQuery, add a handler that responds to the `input` event, checks the element's `value` attribute for certain "secret words" and changes the colors of other elements on the screen in response.

9.2.4 jQuery Callback functions

Many jQuery effects have parameters for callback functions. Now that you understand a bit more about functions, you can use these effectively as well. For example, the `.slideDown()` function has parameters for `duration` (in milliseconds) and `callback`. The callback parameter should be a function that is to be called after the slide operation has finished.

For example, if you wanted to slide an element down and then slide up another, you could do it like this:

```
$("#div1").slideDown(500, function() {  
    $("#div2").slideUp(500);  
});
```

You can see something like this in action in the file **callBackTest.html** from the **examples** package.

Try it Yourself

1. Add to callBackTest.html so that when the user mouses over div2, it moves down 100 pixels and then changes red.
2. Add to callBackTest.html so that there is a third div. When it is clicked, it should start sliding around the screen randomly and not stop until it is caught. (This is a bit tricky.)

11. Using AJAX with jQuery

Not Written Yet.

12. Gaming on the HTML5 Canvas

Not Written Yet.

13. Object-Oriented JavaScript

Not Written Yet.