Javascript Tutorial

JavaScript Tutorial

Javascript--what the heck is it? Is it a really difficult programming language that casual web designers should be afraid of? What is it used for? Hopefully we'll be able to answer these questions for you and more in this tutorial

JavaScript has been around for several years now, in many different flavors. The main benefit of Javascript is to add additional interaction between the website and its visitors with just a little extra work by the web developer. Javascript allows industrious web masters to get more out of their website than HTML and CSS can provide.

By definition, JavaScript is a client-side scripting language. This means the web surfer's browser will be running the script. The opposite of client-side is server-side, which occurs in a language like PHP. PHP scripts are run by the web hosting server.

There are many uses (and abuses!) for the powerful JavaScript language. Here are a few things that you may or may not have seen in your web surfing days:

Clocks
Mouse Trailers (an animation that follows your mouse when you surf a site)
Drop Down Menus
Alert Messages
Popup Windows
HTML Form Data Validation

How To Write JavaScript

If you have ever used CSS before, you will find the whole part about including JavaScript will be a lot simpler to grasp. Here are Tizag's three important steps you should always follow when creating or using someone else's JavaScript code:

1. Use the script tag to tell the browser you are using JavaScript.
2. Write or download some JavaScript
3. Test the script!

There are so many different things that can go wrong with a script, be it human error, browser compatibility issues, or operating system differences. So, when using JavaScript, be sure that you test your script out on a wide variety of systems and most importantly, on different web browsers.

Your First JavaScript Script

To follow the classic examples of many programming tutorials, let's use JavaScript to print out "Hello World" to the browser. I know this isn't very interesting, but it will be a good way to explain all the overhead required to do something in JavaScript.

HTML & JavaScript Code:

```
<html>
<body>
<script type="text/JavaScript">
<!--
document.write("Hello World!")
//-->
</script>
</body>
</html>
```

Display:

Hello World!

Our first step was to tell the browser we were using a script with the <script> tag. Next we set the type of script equal to "text/JavaScript". You may notice that doing this is similar to the way you specify CSS, which is "text/css".

Next, we added an optional HTML comment that surrounds our JavaScript code. If a browser does not support JavaScript, it will not display our code in plain text to the user! The comment was ended with a "//-->" because "//" signifies a comment in JavaScript. We add that to prevent a browser from reading the end of the HTML comment in as a piece of JavaScript code.

JavaScript document.write

The final step of our script was to use a function called *document.write*, which writes a string into our HTML document. *document.write* can be used to write text, HTML, or a little of both. We passed the famous string of text to the function to spell out "Hello World!" which it printed to the screen.

Do not worry if you don't completely understand how *document.write* works, as we will be discussing functions in a later lesson.

Syntax

Looking at our JavaScript code above, notice that there is no semicolon at the end of the statement "document.write(Hello World!)". Why? JavaScript does not require that you use semicolons to signify the end of each statement.

If you are an experienced programmer and prefer to use semicolons, feel free to do so. JavaScript will not malfunction from ending semicolons. The only time it is necessary to use a semicolon is when you choose to smash two statements onto one line(i.e. two *document.write* statements on one line).

JavaScript - Is it Enabled?

This lesson will first teach you how to enable JavaScript in Internet Explorer, Firefox, and Opera, then show you how you can write a very simple script to separate website visitors who don't have JavaScript enabled from those who do.

Enable JavaScript - Internet Explorer

In Internet Explorer 6/7 (download Internet Explorer), you can check to see if JavaScript is enabled by navigating to the custom security settings that are somewhat buried (don't worry; we'll help you find it).

1. Click on the **Tools** menu
2. Choose **Internet Options...** from the menu
3. Click the **Security** tab on the **Internet Options** pop up
4. Click the **Custom Level...** button to access your security settings
5. Scroll almost all the way down to the **Scripting** section
6. Select the Enable button for **Active scripting**
7. Click **OK** to finish the process
8. Click **Yes** when asked to confirm

Enable JavaScript - Firefox

In Firefox 2 (download Firefox) you can check to see if JavaScript is enabled by navigating to the **Content** settings under **Options**.

1. Click on the **Tools** menu
2. Choose **Options...** from the menu
3. Click the **Content** tab in the **Options** pop up
4. Make sure that **Enable JavaScript** is checked
5. Click **OK** to finish the process

Enable JavaScript - Opera

In Opera (download Opera) you can check to see if JavaScript is enabled by navigating to the **Content** settings under **Preferences**.

1. Click on the **Tools** menu
2. Choose **Preferences...** from the menu
3. Click the **Advanced** tab in the **Preferences** pop up
4. Select **Content** from the list of items on the left
5. Make sure that **Enable JavaScript** is checked
6. Click **OK** to finish the process

JavaScript Detection

These days, it's basically impossible to navigate the web without a JavaScript-enabled browser, so checking whether or not a user has JavaScript enabled is not all that important. Chances are, the only way it be disabled is if the company's IT staff has decided to disable JavaScript for some reason. However, if you still want to be sure your users are JavaScript enabled, this script will get it done.

The only sure fire way to separate users who don't have JavaScript from those who do is to use a simple redirect script that will **only** work for those with JavaScript enabled. If a person's browser does not have JavaScript enabled, the script will not run, and they will remain on the same page.

JavaScript Code:

```
<script type="text/JavaScript">
window.location = "http://www.example.com/JavaScript-enabled.html"
</script>
```

Replace the example URL with the webpage of your choice.

Where to Place JavaScript

There are three general areas that JavaScript can be placed for use in a webpage.

1. Inside the head tag
2. Within the body tag (like our example in the previous lesson)
3. In an external file (we'll talk about this next lesson)

The location choice of *head* or *body* is very simple. If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the *head*. If you want the script to run when the page loads, like our "Hello World!" example in the previous lesson, then you will want to place the script within the *body* tag.

External JavaScript files and their uses will be discussed in the next lesson.

An Example Head Script

Since we have already seen the kind of script that goes in the *body*, how about we write a script that takes place when some event occurs? Let's have an alert show up when a user click on a button.

HTML & JavaScript Code:

```
<html>
<head>
<script type="text/JavaScript">
<!--
function popup() {
alert("Hello World")
}
//-->
</script>
</head>
<body>
<input type="button" onclick="popup()" value="popup">
</body>
</html>
```

Display:

We created a function called *popup* and placed it in the *head* of the HTML document. Now every time someone clicks on the button (this is an event), an alert will pop up with "Hello World!". We will go into more depth on functions and events in a later lesson.

External JavaScript Files

Having already discussed placing JavaScript in the *head* and *body* of your HTML document, let us now explore the third possible location type -- an external file. If you have ever used external CSS before, this lesson will be a cinch.

Importing an External JavaScript File

Importing an external file is relatively painless. First, the file you are importing must be valid JavaScript, and **only** JavaScript. Second, the file must have the file extension ".js". Lastly, you must know the location of the file.

Let us assume we have a file "myjs.js" that contains a one line Hello World alert function. Also, let us assume that the file is the same directory as the HTML file we are going to code up. To import the file you would do the following in your HTML document.

File myjs.js Contents:

```
function popup() {
alert("Hello World")
}
```

HTML & JavaScript Code:

```
<html>
<head>
<script src="myjs.js">
</script>
</head>
<body>
<input type="button" onclick="popup()" value="Click Me!">
</body>
</html>
```

Display:

Great JavaScript Repositories

There is a ton of great stuff you can do with JavaScript, if you know how to code like Paul Allen and Bill Gates, but for the rest of us, it is nice to get incredible JavaScript tools without having to write them ourselves. Below are some of the better JavaScript resources on the web these days.

JavaFile.com
Java-Scripts.com
Drop Down JavaScript Menus

External File Tips & Recap

Use external JavaScript files when you want to use the same script on many pages, but don't want to have to rewrite the code on every page!
Use external JavaScript files for including both types of scripts: the type that you place in the *head* (functions) and the type you place in the *body* (scripts you want to run when the page loads).
Be sure that your JavaScript files (.js) do not include the <script> tag. They should only contain HTML commenting and JavaScript code -- nothing more!

JavaScript Operators

Operators in JavaScript are very similar to operators that appear in other programming languages. The definition of an operator is a symbol that is used to perform an operation. Most often these operations are arithmetic (addition, subtraction, etc), but not always.

You will want to bookmark this page for future reference.

JavaScript Arithmetic Operator Chart

Operator	English	Example
+	Addition	2 + 4
-	Subtraction	6 - 2
*	Multiplication	5 * 3
/	Division	15 / 3
%	Modulus	43 % 10

Modulus **%** may be a new operation to you, but it's just a special way of saying "finding the remainder". When you perform a division like 15/3 you get 5, exactly. However, if you do 43/10 you get an answer with a decimal, 4.3. 10 goes into 40 four times and then there is a leftover. This leftover is what is returned by the modulus operator. 43 % 10 would equal 3.

JavaScript Operator Example with Variables

Performing operations on variables that contain values is very common and easy to do. Below is a simple script that performs all the basic arithmetic operations.

HTML & JavaScript Code:

```
<body>
<script type="text/JavaScript">
<!--
var two = 2
var ten = 10
var linebreak = "<br />"

document.write("two plus ten = ")
var result = two + ten
document.write(result)
document.write(linebreak)


document.write("ten * ten = ")
result = ten * ten
document.write(result)
document.write(linebreak)

document.write("ten / two = ")
result = ten / two
document.write(result)
//-->
</script>
</body>
```

Display:

two plus ten = 12
ten * ten = 100
ten / two = 5

Comparison Operators

Comparisons are used to check the relationship between variables and/or values. A single equal sign sets a value while a double equal sign (==) compares two values. Comparison operators are used inside conditional statements and evaluate to either *true* or *false*. We will talk more about conditional statements in the upcoming lessons.

Operator	English	Example	Result
==	Equal To	x == y	false
!=	Not Equal To	x != y	true
<	Less Than	x < y	true
>	Greater Than	x > y	false
<=	Less Than or Equal To	x <= y	true
>=	Greater Than or Equal To	x >= y	false

JavaScript Variables

Variables in JavaScript behave the same as variables in most popular programming languages (C, C++, etc) do, but in JavaScript you don't have to declare variables before you use them. If you don't know what declaring is, don't worry about it. It isn't important!

JavaScript Using Variables

A variable's purpose is to store information so that it can be used later. A variable is a symbolic name that represents some data that you set. To think of a variable name in real world terms, picture that the name is a grocery bag and the data it represents are the groceries. The name wraps up the data so you can move it around a lot easier, but the name is not the data!

A Variable Example

When using a variable for the first time it is not necessary to use "var" before the variable name, but it is a good programming practice to make it crystal clear when a variable is being used for the first time in the program. Here we are showing how the same variable can take on different values throughout a script.

HTML & JavaScript Code:

```
<body>
<script type="text/JavaScript">
<!--
var linebreak = "<br />"
var my_var = "Hello World!"

document.write(my_var)
document.write(linebreak)

my_var = "I am learning JavaScript!"
document.write(my_var)
document.write(linebreak)

my_var = "Script is Finishing up..."
document.write(my_var)
//-->
</script>
</body>
```

Display:

Hello World!
I am learning JavaScript!
Script is Finishing up...

We made two variables in this example--one to hold the HTML for a line break and the other for a dynamic variable that had a total of three different values throughout the script.

To assign a value to a variable, you use the equal sign (=) with the variable on the left and the value to be assigned on the right. If you swap the order, your script will not work correctly! In English, the JavaScript "myVar = 'Hello World!'" would be: myVar equals 'Hello World!'.

The first time we used a variable, we placed *var* in front to signify its first use. This is an easy way to organize the variables in your code and see when they came into existence. In subsequent assignments of the same variable, we did not need the var.

JavaScript Variable Naming Conventions

When choosing a variable name, you must first be sure that you do not use any of the JavaScript reserved names Found Here. Another good practice is choosing variable names that are descriptive of what the variable holds. If you have a variable that holds the size of a shoe, then name it "shoe_size" to make your JavaScript more readable.

Finally, JavaScript variable names may not start with a numeral (0-9). These variable names would be illegal: 7lucky, 99bottlesofbeer, and 3zacharm.

A good rule of thumb is to have your variable names start with a lowercase letter (a-z) and use underscores to separate a name with multiple words (i.e. my_var, strong_man, happy_coder, etc).

JavaScript Functions

If you have any programming experience, you do not need to spend much time on this lesson. Functions in JavaScript behave similarly to numerous programming languages (C, C++, PHP, etc). If this is your first time learning about functions, then be sure to go through this lesson very thoroughly. A solid understanding of functions will make the rest of this tutorial much easier to follow.

What's a Function?

A function is a piece of code that sits dormant until it is referenced or called upon to do its "function". In addition to controllable execution, functions are also a great time saver for doing repetitive tasks.

Instead of having to type out the code every time you want something done, you can simply call the function multiple times to get the same effect. This benefit is also known as "code reusability".

Example Function in JavaScript

A function that does not execute when a page loads should be placed inside the *head* of your HTML document. Creating a function is really quite easy. All you have to do is tell the browser you're making a function, give the function a name, and then write the JavaScript like normal. Below is the example alert function from the previous lesson.

HTML & JavaScript Code:

```
<html>
<head>
<script type="text/javascript">
<!--
function popup() {
    alert("Hello World")
}
//-->
</script>
</head>
<body>
<input type="button" onclick="popup()" value="popup">
</body>
</html>
```

We first told the browser we were going to be using a function by typing "function". Next, we gave our function a name, so that we could use it later. Since we are making a pop up alert, we called our function "popup".

The curly braces "{,}" define the boundaries of our function code. All popup function code must be contained within the curly braces.

Something that might be slightly confusing is that within our "popup" function, we use **another** function called "alert," which brings up a popup box with the text that we supply it. It is perfectly OK to use functions within functions, like we have done here. Furthermore, this is one of the great things about using functions!

What we didn't talk about was how we got this function to execute when the button is clicked. The click is called an event, and we will be talking about how to make functions execute from various types of events in the next lesson.

Events in JavaScript

The absolute coolest thing about JavaScript is its ability to help you create dynamic webpages that increase user interaction, making the visitor feel like the website is almost coming alive right before her eyes.

The building blocks of an interactive web page is the JavaScript event system. An event in JavaScript is something that happens with or on the webpage. A few example of events:

A mouse click
The webpage loading
Mousing over a hot spot on the webpage, also known as hovering
Selecting an input box in an HTML form
A keystroke

A Couple of Examples Using Events

JavaScript has predefined names that cover numerous events that can occur, including the ones listed above. To capture an event and make something happen when that event occurs, you must specify the event, the HTML element that will be waiting for the event, and the function(s) that will be run when the event occurs.

We have used a JavaScript event in a previous lesson, where we had an alert popup when the button was clicked. This was an "onclick" JavaScript event. We will do that same example again, as well as the *mouseover* and *mouseout* events.

HTML & JavaScript Code:

```
<html>
<head>
<script type="text/javascript">
<!--
function popup() {
                alert("Hello World")
}
//-->
</script>
</head>
<body>

<input type="button" value="Click Me!" onclick="popup()"><br />
<a href="#" onmouseover="" onMouseout="popup()">
Hover Me!</a>
```

```
</body>
</html>
```

Display:

Hover Me!

With the button we used the event *onClick* event as our desired action and told it to call our popup function that is defined in our header. To call a function you must give the function name followed up with parenthesis "()".

Our *mouseover* and *mouseout* events were combined on one HTML element--a link. We wanted to do nothing when someone put their mouse on the link, but when the mouse moves off the link (*onMouseout*), we displayed a popup.

JavaScript Statements

All the JavaScript code that you will write will, for the most part, be comprised of many separate statements. A statement can set a variable equal to a value. A statement can also be a function call, i.e. *document.write()*. Statements define what the script will do and how it will be done.

Typical Ending of a Statement

In typical programming languages like C and PHP, the end of a statement is marked with a semicolon(;), but we have seen that the semicolon is optional in JavaScript. In JavaScript, the end of a statement is most often marked by pressing return and starting a new line.

Categories of Statements

In addition to standard statements like changing a variable's value, assigning a new value, or calling a function, there are groups of statements that are distinct in their purpose. We will provide a brief overview of each of these categories in this lesson and cover them in greater detail later in the tutorial. These distinct groups of statements include:

Conditional Statements
Loop Statements
Object Manipulation Statements
Comment Statements
Exception Handling Statements

Conditional Statements

If you were to win a $100 million lottery, you would probably quit your job. That last statement is a conditional if/then statement that is used a great deal in programming. If some condition (winning the lottery) is true, then something happens (you quit your job). If the condition is false (you didn't win the lottery), then you probably will not take that action (quit your job).

Conditional statements are used to control your scripts so that different actions can be taken depending on the situation. You may want to display a special image on your home page during the holidays. This condition would depend on what day it was, and if it was a holiday, then a special holiday image would be displayed to your visitors. Without proper knowledge of the conditional statement, your scripts will not be as lively or dynamic as they could possibly be.

Loop Statements

Have you ever had to send out marriage announcements? If not, this is how it goes. You take the invitation, place it in the envelope, lick the envelope, seal the envelope, then send it off. Then you take the next invitation off the stack of 99 remaining invitations, place it in an envelope, lick the envelope, seal... You get the idea! It is a boring and repetitive task!

Wouldn't it be great if there was an easier way? Well, in programming and in JavaScript there is! The process is called "looping." With the right planning, looping will turn your cute little scripts into massive workhorses.

A loop statement checks to see if some condition is true, and if that condition is true, it executes a chunk of code. After the code is executed, the condition is checked again. If it is true, the process starts over again; if it is false, the loop stops and the rest of the code continues along. If we think about our wedding invitation example as a loop, we would first check if there are any invitations left. If there are, we would stuff, lick, and seal the next envelope. If there are no envelopes left, we would stop.

Believe me when I say this is something you want to learn more about!

Object Manipulation Statements

These are statements that are used to take advantage of the object model to get tasks done. If you do not know about the object model at this time, do not worry. We will be talking about it later.

Comment Statements

Comment statements are used to prevent the browser from executing certain parts of code that you designate as non-code. Why would you want to do this? There are many reasons. By disallowing the block of text from being be read, you can then place in comments for yourself, much like HTML comments. You can also block out segments of your code for whatever reason you may have.

The single line comment is just two slashes (//) and the multiple line comment starts with (/*) and ends with (*/). We will talk about comments in greater depth in a later lesson.

Exception Handling Statements

Sometimes when you are programming you do not know for sure if the file that you will be writing to, the input stream you are reading from, or the connection you have established will be usable for the code that you want to execute. There is a way to program safety mechanisms, so that your code handles common problems that may arise (maybe the input stream you are reading from suddenly disappears).

The *try...catch* statement tries to execute a piece of code and if it fails, the *catch* should handle the error gracefully. This is an advanced programming subject that is interesting, but not necessary for the majority of JavaScript programmers.

JavaScript If Statement

As your JavaScript programs get more sophisticated, you will need to make use of conditional statements that allow your program to make decisions. Nearly all other programming languages use conditionals, and JavaScript is no exception.

The "If Statement" is a way to make decisions based on a variable or some other type of data. For example, you might have a variable that stores the date. With this tiny bit of information, you can easily program a small script to print out, "Today is my Birthday!" whenever the day and month were equal to your birthday.

This lesson will teach you the basics of using an "If Statement" in JavaScript.

JavaScript If Statement Syntax

There are two major parts to an If Statement: the conditional statement and the code to be executed.

The conditional statement is a statement that will evaluate to be either True or False. The most common type of conditional statement used checks to see if something equals a value. An example would be checking if a date equals your birthday.

Below is a segment of JavaScript code that will be executed only if the If Statement's conditional statement is true. In this simple *If Statement* example, we print out a message if the variable we are checking is equal to 7.

JavaScript Code:

```
<script type="text/javascript">
<!--
var myNum = 7;

if(myNum == 7){
		document.write("Lucky 7!");
}
//-->
</script>
```

Display:

Lucky 7!

This simple example created *myNum* and set it to 7. We then checked to see if *myNum* was equal to 7 ("myNum == 7") in the *If Statement's* conditional statement, evaluated to *True*.

Because the conditional statement was *True* the block of code associated with our If Statement ("document.write...") was executed, as you can see in the Display.

JavaScript If Statement: Else

We already taught you how to execute code if a given condition is *True*, but what if you want to execute *another* piece of code if something is *False*? The answer is to use an extension to the *If Statement*; the *Else* clause.

The *Else* clause is executed when the conditional statement is *False*. Let's take our example from above, add an *Else* clause, and change the value of *myNum* so that our conditional statement is False.

JavaScript Code:

```
<script type="text/javascript">
<!--
var myNum = 10;

if(myNum == 7){
		document.write("Lucky 7!");
}else{
		document.write("You're not very lucky today...");
}
//-->
</script>
```

Display:

You're not very lucky today...

JavaScript Else If Statement

In the previous lesson, you learned how to create a basic If Statement in JavaScript, which is good enough for most programming situations. However, sometimes it is helpful to have the ability to check for more than one condition in a single If Statement block.

The *Else If* statement is an extension to the If Statement that allows you to create as many checks (conditional statements) as you want in one big block of If Statement code.

JavaScript Else If Example

Imagine that you want to have a small "student" script that will print out a customized message depending who is accessing the webpage. If you have more than two custom messages, you could use the *Else If* extension to solve this programming problem.

JavaScript Code:

```
<script type="text/javascript">
<!--
var visitor = "principal";

if(visitor == "teacher"){
		document.write("My dog ate my homework...");
}else if(visitor == "principal"){
		document.write("What stink bombs?");
} else {
		document.write("How do you do?");
}
//-->
</script>
```

Display:

What stink bombs?

There are two important things to note about the *Else If* extension:

1. You must have a normal If Statement before you can use the *Else If* statement. This is because the *Else If* statement is an addon to the If Statement.
2. You can have multiple *Else If* add-ons. In our example, we only used one *Else If* extension, but you can add as many as you require.

JavaScript While Loop

The while loop is an advanced programming technique that allows you to do something over and over *while* a conditional statement is true. Although the general uses of the *while loop* are usually a bit complex, this lesson will teach you the basics of how to create a *while loop* in JavaScript.

JavaScript While Loop Explained

There are two key parts to a JavaScript *while loop*:

1. The conditional statement which must be *True* for the *while loop's* code to be executed.
2. The *while loop's* code that is contained in curly braces "{ and }" will be executed if the condition is *True*.

When a *while loop* begins, the JavaScript interpreter checks if the condition statement is true. If it is, the code between the curly braces is executed. At the end of the code segment "}", the *while loop* loops back to the condition statement and begins again.

If the condition statement is always *True*, then you will never exit the *while loop*, so be **very careful** when using while loops!

Creating a Simple While Loop

This example shows how to create a basic *while loop* that will execute a *document.write* 10 times and then exit the loop statement.

JavaScript Code:

```
<script type="text/javascript">
<!--
var myCounter = 0;
var linebreak = "<br />";
document.write("While loop is beginning");
document.write(linebreak);

while(myCounter < 10){
        document.write("myCounter = " + myCounter);
        document.write(linebreak);
        myCounter++;
}

document.write("While loop is finished!");
</script>
```

Display:

```
While loop is beginning
myCounter = 0
myCounter = 1
myCounter = 2
myCounter = 3
myCounter = 4
myCounter = 5
myCounter = 6
myCounter = 7
myCounter = 8
myCounter = 9
While loop is finished!
```

Our variable *myCounter* started off at 0, which is less than 10, so our while loop executed its code. The value 0 was printed to the browser and then *myCounter* was incremented by 1 and the while loop started over again.

1 was less than 10 so the while loop's code was executed... and the process repeats itself a few more times until...

myCounter was 10 which was not less than 10 so the while loop's code did not execute. You can see this in the *Display:* because the last value to be printed out was 9.

Note: Advanced programmers may recognize that a *for loop* would be a better solution for this example, but we hope you can ignore this for our needs to create an easy example!

JavaScript For Loop

The JavaScript *For Loop* resembles the for loop you may have seen in many other programming languages. It is used when you need to do a set of operations many times, with an increment of some kind after each run through the block of code.

If you have read the previous lesson JavaScript While Lesson then this should be a cinch.

JavaScript For Loop Explained

There are four important aspects of a JavaScript *for loop*:

1. The *counter* variable is something that is created and usually used only in the *for loop* to count how many times the *for loop* has *looped*. i is the normal label for this counter variable and what we will be using.
2. The conditional statement. It is what decides whether the *for loop* continues executing or not. This check usually includes the *counter* variable in some way.
3. The counter variable is incremented after every loop in the *increment* section of the *for loop*.
4. The code that is executed for each loop through the *for loop*.

This may seem strange, but 1-3 all occur on the same line of code. This is because the *for loop* is such a standardized programming practice that the designers felt they might as well save some space and clutter when creating the *for loop*.

JavaScript For Loop Example

This example will show you how to create a simple *for loop* that prints out the value of our counter until the counter reaches 5. Pay special close attention to the three different items that are on the first line of the for loop code. These are the important *for loop* parts 1-3 that we talked about earlier.

JavaScript Code:

```
<script type="text/javascript">
<!--
var linebreak = "<br />";
document.write("For loop code is beginning");
document.write(linebreak);

for(i = 0; i < 5; i++){
                document.write("Counter i = " + i);
                document.write(linebreak);
}

document.write("For loop code is finished!");
</script>
```

Display:

```
For loop code is beginning
Counter i = 0
Counter i = 1
Counter i = 2
Counter i = 3
Counter i = 4
For loop code is finished!
```

The counter variable name *i* may seem a little strange, but it has been used for years now! No matter the language, *i* is the default name for a loop counter. Other common variable names are *j, k, x, y* and *z*.

In this example, our counter was initially set to 0 with "i = 0;" and then the conditional statement "i < 5;" was executed. Our counter was indeed smaller than 5, so the *for loop*'s code was executed.

After the loop's code is executed **then** the increment "i++" happens, making the counter *i* equal to 1. The *for loop* then checked that *i* was less than 5, which it was, causing the *for loop's* code to be executed again.

This looping happened a couple more times until *i* was equal to 5, which is not less than 5, causing the *for loop*'s code to stop executing.

For loops may seem very confusing at first, but let me assure you, they are quite useful and should be studied thoroughly by anyone who wishes to become an intermediate programmer.

JavaScript Comments

Have you ever written a script or a program in the past only to look at it six months later with no idea what's going on in the code? You probably forgot to do what all programmers tend to forget to do: write comments!

When writing code you may have some complex logic that is confusing, this is a perfect opportunity to include some comments in the code that will explain what is going on. Not only will this help you remember it later on, but if you someone else views your code, they will also be able to understand the code (hopefully)!

Another great thing about comments is the ability for comments to remove bits of code from execution when you are debugging your scripts. This lesson will teach you how to create two types of comments in JavaScript: single line comments and multi-line comments.

Creating Single Line Comments

To create a single line comment in JavaScript, you place two slashes "//" in front of the code or text you wish to have the JavaScript interpreter ignore. When you place these two slashes, all text to the right of them will be ignored, until the next line.

These types of comments are great for commenting out single lines of code and writing small notes.

JavaScript Code:

```
<script type="text/javascript">
<!--
// This is a single line JavaScript comment

document.write("I have comments in my JavaScript code!");
//document.write("You can't see this!");
//-->
</script>
```

Display:

```
I have comments in my JavaScript code!
```

Each line of code that is colored red is commented out and will not be interpreted by the JavaScript engine.

Creating Multi-line Comments

Although a single line comment is quite useful, it can sometimes be burdensome to use when disabling long segments of code or inserting long-winded comments. For this large comments you can use JavaScript's multi-line comment that begins with /* and ends with */.

JavaScript Code:

```
<script type="text/javascript">
<!--
document.write("I have multi-line comments!");
/*document.write("You can't see this!");
document.write("You can't see this!");
document.write("You can't see this!");
document.write("You can't see this!");
document.write("You can't see this!");
document.write("You can't see this!");*/
//-->
</script>
```

Display:

I have multi-line comments!

Quite often text editors have the ability to comment out many lines of code with a simple key stroke or option in the menu. If you are using a specialized text editor for programming, be sure that you check and see if it has an option to easily comment out many lines of code!

JavaScript Array

An array is a variable that can store many variables within it. Many programmers have seen arrays in other languages, and they aren't that different in JavaScript.

The following points should always be remembered when using arrays in JavaScript:

The array is a special type of variable.
Values are stored into an array by using the array name and by stating the location in the array you wish to store the value in brackets. Example: myArray[2] = "Hello World";
Values in an array are accessed by the array name and location of the value. Example: myArray[2];
JavaScript has built-in functions for arrays, so check out these built-in array functions before writing the code yourself!

Creating a JavaScript Array

Creating an array is slightly different from creating a normal variable. Because JavaScript has variables and properties associated with arrays, you have to use a special function to create a new array. This example shows how you would create a simple array, store values to it, and access these values.

JavaScript Code:

```
<script type="text/javascript">
<!--
var myArray = new Array();

myArray[0] = "Football";
myArray[1] = "Baseball";
myArray[2] = "Cricket";

document.write(myArray[0] + myArray[1] + myArray[2]);
//-->
</script>
```

Display:

FootballBaseballCricket

Notice that you set values and get values from an array by specifying the position, in brackets, of the value you want to use.

JavaScript Array Sorting

Imagine that you wanted to sort an array alphabetically before you wrote the array to the browser. Well, this code has already been written and can be accessed by using the Array's *sort* method.

JavaScript Code:

```
<script type="text/javascript">
<!--
var myArray2= new Array();

myArray2[0] = "Football";
myArray2[1] = "Baseball";
myArray2[2] = "Cricket";

myArray2.sort();

document.write(myArray2[0] + myArray2[1] + myArray2[2]);
//-->
</script>
```

Display:

BaseballCricketFootball

JavaScript Array: Other Resources

Arrays are a very complex area of programming, especially in JavaScript. This lesson cannot possibly cover all the areas of JavaScript arrays, but we have compiled some useful resources for you to check out!

JavaScript Array Object - General information about the JavaScript Array Object.
Dev Guru Array Methods - A large collection of all the things you shouldn't program for an array because they have all been done for you!

JavaScript Alert - What is it?

If you do not have JavaScript enabled on your web browser, then you may have been able to avoid "alerts" in your internet adventures. The JavaScript alert is a dialogue box that pops up and takes the focus away from the current window and forces the web browser to read the message. View an alert message.

You may have noticed that you didn't get a JavaScript alert popup when you came to this page. That is because doing so would be in bad taste for a web designer. You see, alerts should be very, very rarely used and even then these following guidelines should be considered when using them.

When to Use Popups / Alerts

JavaScript alerts are ideal for the following situations:

If you want to be absolutely sure they see a message before doing anything on the website.
You would like to warn the user about something. For example "the following page contains humor not suitable for those under the age of 14."
An error has occurred and you want to inform the user of the problem.
When asking users for confirmation of some action. For example, if they have just agreed to sign over the deed to their house and you want to ask them again if they are absolutely positive they want to go through with this decision!

Even though the above situations would all be valid times to use the alert function, you could also skip the alert popup and just have the error message, confirmation, etc displayed in plain HTML. More and more bigger sites are opting to lose JavaScript alerts and instead keep everything in HTML.

Coding a Simple JavaScript Alert

Just for fun, let's suppose that we are making an alert for some website that asks people to hand over the deed to their house. We need to add an alert to be sure these people are in agreement. The following code will add an alert by using an HTML button and the *onClick* event.

HTML & JavaScript Code:

```
<form>
<input type="button" onclick=
"alert('Are you sure you want to give us the deed to your house?')"
value="Confirmation Alert">
</form>
```

Display:

The string that appears between the single quotes is what will be printed inside the alert box when the user clicks on the button. If the HTML Forms are confusing to you, be sure to brush up on our HTML Forms Lesson. Continue the tutorial to learn more about the different kinds of JavaScript pop ups that are at your disposal.

JavaScript Confirm

The JavaScript *confirm* function is very similar to the JavaScript *alert* function. A small dialogue box pops up and appears in front of the web page currently in focus. The confirm box is different from the alert box. It supplies the user with a choice; they can either press OK to confirm the popup's message or they can press cancel and not agree to the popup's request.

Confirmation are most often used to *confirm* an important actions that are taking place on a website. For example, they may be used to confirm an order submission or notify visitors that a link they clicked will take them to another website.

JavaScript Confirm Example

Below is an example of how you would use a confirm dialogue box to warn users about something, giving them the option to either continue on or stay put.

HTML & JavaScript Code:

```
<html>
<head>
<script type="text/javascript">
<!--
function confirmation() {
        var answer = confirm("Leave tizag.com?")
        if (answer){
                alert("Bye bye!")
                window.location = "http://www.google.com/";
        }
        else{
                alert("Thanks for sticking around!")
        }
}
//-->
</script>
</head>
<body>
<form>
<input type="button" onclick="confirmation()" value="Leave Tizag.com">
</form>
</body>
</html>
```

Note the part in red. This is where all the magic happens. We call the confirm function with the message, "Leave Tizag?". JavaScript then makes a popup window with two choices and will return a value to our script code depending on which button the user clicks.

If the user clicks OK, a value of 1 is returned. If a user clicks cancel, a value of 0 is returned.. We store this value in *answer* by setting it equal to the *confirm* function call.

After *answer* has stored the value, we then use *answer* as a conditional statement. If *answer* is anything but zero, then we will send the user away from Tizag.com. If *answer* is equal to zero, we will keep the user at Tizag.com because they clicked the Cancel button.

In either case, we have a JavaScript alert box that appears to inform the user on what is going to happen. It will say, "Bye bye!" if they choose to leave and, "Thanks for sticking around!" if they choose to stay.

In this lesson, we also used the *window.location* property for the first time. Whatever we set *window.location* to will be where the browser is redirected to. In this example, we chose Google.com. We will discuss redirection in greater detail later on.

JavaScript Confirm

The JavaScript *confirm* function is very similar to the JavaScript *alert* function. A small dialogue box pops up and appears in front of the web page currently in focus. The confirm box is different from the alert box. It supplies the user with a choice; they can either press OK to confirm the popup's message or they can press cancel and not agree to the popup's request.

Confirmation are most often used to *confirm* an important actions that are taking place on a website. For example, they may be used to confirm an order submission or notify visitors that a link they clicked will take them to another website.

JavaScript Confirm Example

Below is an example of how you would use a confirm dialogue box to warn users about something, giving them the option to either continue on or stay put.

HTML & JavaScript Code:

```
<html>
<head>
<script type="text/javascript">
<!--
function confirmation() {
		var answer = confirm("Leave tizag.com?")
		if (answer){
			alert("Bye bye!")
			window.location = "http://www.google.com/";
		}
		else{
			alert("Thanks for sticking around!")
		}
}
//-->
</script>
</head>
<body>
<form>
<input type="button" onclick="confirmation()" value="Leave Tizag.com">
</form>
</body>
</html>
```

Note the part in red. This is where all the magic happens. We call the confirm function with the message, "Leave Tizag?". JavaScript then makes a popup window with two choices and will return a value to our script code depending on which button the user clicks.

If the user clicks OK, a value of 1 is returned. If a user clicks cancel, a value of 0 is returned.. We store this value in *answer* by setting it equal to the *confirm* function call.

After *answer* has stored the value, we then use *answer* as a conditional statement. If *answer* is anything but zero, then we will send the user away from Tizag.com. If *answer* is equal to zero, we will keep the user at Tizag.com because they clicked the Cancel button.

In either case, we have a JavaScript alert box that appears to inform the user on what is going to happen. It will say, "Bye bye!" if they choose to leave and, "Thanks for sticking around!" if they choose to stay.

In this lesson, we also used the *window.location* property for the first time. Whatever we set *window.location* to will be where the browser is redirected to. In this example, we chose Google.com. We will discuss redirection in greater detail later on.

JavaScript Prompt

The JavaScript prompt is a relic from the 1990's that you seldom see being used in modern day websites. The point of the JavaScript prompt is to gather information from the user so that the information can be used throughout the site to give the visitor a personalized feel.

Back in the day, you'd often see prompts on personal webpages asking for your name. After you typed in the information, you would be greeted with a page that had a welcome message, such as, "Welcome to My Personal WebPage John Schmieger!" (If your name just so happened to be John Schmieger).

The JavaScript prompt is not very useful and many find it slightly annoying, but hey, this tutorial is here to educate you, so let's learn how to make that prompt!

Simple JavaScript Prompt

You can use a prompt for a wide variety of useless tasks, but below we use it for an exceptionally silly task. Our prompt is used to gather the user's name to be displayed in our alert dialogue box.

HTML & JavaScript Code:

```
<head>
<script type="text/javascript">
<!--
function prompter() {
var reply = prompt("Hey there, good looking stranger!  What's your name?", "")
alert ( "Nice to see you around these parts " + reply + "!")
}
//-->
</script>
</head>
<body>
<input type="button" onclick="prompter()" value="Say my name!">
</body>
```

Display:

Recap on JavaScript Prompt

It sure is a quick way to gather some information, but it is not as reliable an information gatherer as other options available to you. If you want to find out someone's name and information, the best way to request this information would be through the use of HTML Forms. And if you want to use the information you collected in your website, you might use some PHP to get that job done in a more sophisticated manner.

JavaScript Print Function

The JavaScript print function performs the same operation as the print option that you see at the top of your browser window or in your browser's "File" menu. The JavaScript print function will send the contents of the webpage to the user's printer.

Many believe this function to be worthless, but there are many computer users who do not know their way around a computer well, and helpful features such as this function can somtimes create a more user-friendly environment.

JavaScript Print Script - window.print()

The JavaScript print function *window.print()* will print the current webpage when executed. In this example script, we will be placing the function on a JavaScript button that will perform the print operation when the *onClick* event occurs.

HTML & JavaScript Code:

```
<form>
<input type="button" value="Print This Page" onClick="window.print()" />
</form>
```

Display:

If you click this button you should be prompted with whatever application your computer uses to handle its print functionality.

JavaScript Redirect

You're moving to a new domain name. You have a time-delay placeholder on your download site. You have a list of external web servers that are helping to mirror your site. What will help you deal with and/or take advantage of these situations? JavaScript redirects will.

When your webpage is moved, you'd probably want to notify your visitors of the change. One good way is to place a "redirect page" at the old location which, after a timed delay, will forward visitors to the new location of your webpage. You can do just this with a JavaScript redirection.

JavaScript Window.Location

Control over what page is loaded into the browser rests in the JavaScript property *window.location*. By setting *window.location* equal to a new URL, you will in turn change the current webpage to the one that is specified. If you wanted to redirect all your visitors to www.google.com when they arrived at your site, you would just need the script below:

HTML & JavaScript Code:

```
<script type="text/javascript">
<!--
window.location = "http://www.google.com/"
//-->
</script>
```

JavaScript Time Delay

Implementing a timed delay in JavaScript is useful in the following situations:

- Showing an "Update your bookmark" page when you have to change URLs or page locations
- For download sites that wish to have a timed delay before the download starts
- To refresh a webpage once every specified number of seconds

The code for this timed delay is slightly involved and is beyond the scope of this tutorial. However, we have tested it and it seems to function properly.

HTML & JavaScript Code:

```
<html>
<head>
<script type="text/javascript">
<!--
function delayer(){
   window.location = "../javascriptredirect.htm"
}
//-->
</script>
</head>
<body onLoad="setTimeout('delayer()', 5000)">
<h2>Prepare to be redirected!</h2>
<p>This page is a time delay redirect, please update your bookmarks to our new
location!</p>

</body>
</html>
```

View Page:

Time Delay Redirect

The most important part of getting the delay to work is being sure to use the JavaScript function *setTimeout*. We want the *delayer()* function to be used after 5 seconds or 5000 milliseconds (5 seconds), so we pass the *setTimeout()* two arguments.

'delayer()' - The function we want *setTimeout()* to execute after the specified delay.
5000 - the number of millisecods we want *setTimeout()* to wait before executing our function. 1000 miliseconds = 1 second.

Web Page Redirection

Do use JavaScript for redirections when you change your website's URL or move a file to a new location. Don't use redirections when they can easily be replaced with a normal HTML hyperlink.

JavaScript Popups

Chances are, if you are reading this webpage, then you have experienced hundreds of JavaScript popup windows throughout your web surfing lifetime. Want to dish out some pain of your own creation onto unsuspecting visitors? I hope not! Because websites with irrelevant popups are bad!

However, we will show you how to make windows popup for reasonable occasions, like when you want to display extra information, or when you want to have something open a new window that isn't an HTML anchor tag (i.e. a hyperlink).

JavaScript window.open Function

The *window.open()* function creates a new browser window, customized to your specifications, without the use of an HTML anchor tag. In this example, we will be making a function that utilizes the *window.open()* function.

HTML & JavaScript Code:

```
<head>
<script type="text/javascript">
<!--
function myPopup() {
window.open( "http://www.google.com/" )
}
//-->
</script>
</head>
<body>
<form>
<input type="button" onClick="myPopup()" value="POP!">
</form>
<p onClick="myPopup()">CLICK ME TOO!</p>
</body>
```

Display:

CLICK ME TOO!

This works with pretty much any tag that can be clicked on, so please go ahead and experiment with this fun little tool. Afterwards, read on to learn more about the different ways you can customize the JavaScript window that **pops** up.

JavaScript Window.Open Arguments

There are three arguments that the *window.open* function takes:

1. The relative or absolute URL of the webpage to be opened.
2. The text name for the window.
3. A long string that contains all the different properties of the window.

Naming a window is very useful if you want to manipulate it later with JavaScript. However, this is beyond the scope of this lesson, and we will instead be focusing on the different properties you can set with your brand spanking new JavaScript window. Below are some of the more important properties:

dependent - Subwindow closes if the parent window (the window that opened it) closes
fullscreen - Display browser in fullscreen mode
height - The height of the new window, in pixels
width - The width of the new window, in pixels
left - Pixel offset from the left side of the screen
top - Pixel offset from the top of the screen
resizable - Allow the user to resize the window or prevent the user from resizing, currently broken in Firefox.
status - Display or don't display the status bar

Dependent, fullscreen, resizable, and status are all examples of ON/OFF properties. You can either set them equal to zero to turn them off, or set them to one to turn them ON. There is no inbetween setting for these types of properties.

Upgraded JavaScript Popup Window!

Now that we have the tools, let's make a sophisticated JavaScript popup window that we can be proud of!

HTML & JavaScript Code:

```
<head>
<script type="text/javascript">
<!--
function myPopup2() {
window.open( "http://www.google.com/", "myWindow",
"status = 1, height = 300, width = 300, resizable = 0" )
}
//-->
</script>
</head>
<body>
<form>
<input type="button" onClick="myPopup2()" value="POP2!">
</form>
<p onClick="myPopup2()">CLICK ME TOO!</p>
</body>
```

Display:

CLICK ME TOO!

Now, that is a prime example of a worthless popup! When you make your own, try to have them relate to your content, like a small popup with no navigation that just gives the definition or explanation of a word, sentence, or picture!

JavaScript Date and Time Object

The Date object is useful when you want to display a date or use a timestamp in some sort of calculation. In Java, you can either make a Date object by supplying the date of your choice, or you can let JavaScript create a Date object based on your visitor's system clock. It is usually best to let JavaScript simply use the system clock.

When creating a Date object based on the computer's (not web server's!) internal clock, it is important to note that if someone's clock is off by a few hours or they are in a different time zone, then the Date object will create a different times from the one created on your own computer.

JavaScript Date Today (Current)

To warm up our JavaScript Date object skills, let's do something easy. If you do not supply any arguments to the Date constructor (this makes the Date object) then it will create a Date object based on the visitor's internal clock.

HTML & JavaScript Code:

```
<h4>It is now
<script type="text/javascript">
<!--
var currentTime = new Date()
//-->
</script>
</h4>
```

Display:

It is now

Nothing shows up! That's because we still don't know the methods of the Date object that let us get the information we need (i.e. Day, Month, Hour, etc).

Get the JavaScript Time

The Date object has been created, and now we have a variable that holds the current date! To get the information we need to print out, we have to utilize some or all of the following functions:

getTime() - Number of milliseconds since 1/1/1970 @ 12:00 AM
getSeconds() - Number of seconds (0-59)
getMinutes() - Number of minutes (0-59)
getHours() - Number of hours (0-23)
getDay() - Day of the week(0-6). 0 = Sunday, ... , 6 = Saturday
getDate() - Day of the month (0-31)
getMonth() - Number of month (0-11)
getFullYear() - The four digit year (1970-9999)

Now we can print out the date information. We will be using the *getDate*, *getMonth*, and *getFullYear* methods in this example.

HTML & JavaScript Code:

```
<h4>It is now
<script type="text/javascript">
<!--
var currentTime = new Date()
var month = currentTime.getMonth() + 1
var day = currentTime.getDate()
var year = currentTime.getFullYear()
document.write(month + "/" + day + "/" + year)
//-->
</script>
</h4>
```

Display:

It is now 3/6/2010 !

Notice that we added 1 to the *month* variable to correct the problem with January being 0 and December being 11. After adding 1, January will be 1, and December will be 12.

JavaScript Current Time Clock

Now, instead of displaying the date we, will display the format you might see on a typical digital clock -- HH:MM AM/PM (H = Hour, M = Minute).

HTML & JavaScript Code:

```
<h4>It is now
<script type="text/javascript">
<!--
var currentTime = new Date()
var hours = currentTime.getHours()
var minutes = currentTime.getMinutes()
if (minutes < 10){
minutes = "0" + minutes
}
document.write(hours + ":" + minutes + " ")
if(hours > 11){
document.write("PM")
} else {
document.write("AM")
}
//-->
</script>
</h4>
```

Display:

It is now 14:33 PM

Above, we check to see if either the *hours* or *minutes* variable is less than 10. If it is, then we need to add a zero to the beginning of minutes. This is not necessary, but if it is 1:01 AM, our clock would output "1:1 AM", which doesn't look very nice at all!

JavaScript Form Validation

There's nothing more troublesome than receiving orders, guestbook entries, or other form submitted data that are incomplete in some way. You can avoid these headaches once and for all with JavaScript's amazing way to combat bad form data with a technique called "form validation".

The idea behind JavaScript form validation is to provide a method to check the user entered information before they can even submit it. JavaScript also lets you display helpful alerts to inform the user what information they have entered incorrectly and how they can fix it. In this lesson we will be reviewing some basic form validation, showing you how to check for the following:

- If a text input is empty or not
- If a text input is all numbers
- If a text input is all letters
- If a text input is all alphanumeric characters (numbers & letters)
- If a text input has the correct number of characters in it (useful when restricting the length of a username and/or password)
- If a selection has been made from an HTML select input (the drop down selector)
- If an email address is valid
- How to check all above when the user has completed filling out the form

This lesson is a little long, but knowing how to implement these form validation techniques is definitely worth the effort on your part. Remember to check out Tizag's HTML forms lesson if you need to brush up on your form knowledge.

Form Validation - Checking for Non-Empty

This has to be the most common type of form validation. You want to be sure that your visitors enter data into the HTML fields you have "required" for a valid submission. Below is the JavaScript code to perform this basic check to see if a given HTML input is empty or not.

JavaScript Code:

```
// If the length of the element's string is 0 then display helper message
function notEmpty(elem, helperMsg){
        if(elem.value.length == 0){
                alert(helperMsg);
                elem.focus(); // set the focus to this input
                return false;
        }
        return true;
}
```

The function *notEmpty* will check to see that the HTML input that we send it has something in it. *elem* is a HTML text input that we send this function. JavaScriptstrings have built in properties, one of which is the *length* property which returns the length of the string. The chunk of code *elem.value* will grab the string inside the input and by adding on length *elem.value.length* we can see how long the string is.

As long as *elem.value.length* isn't 0 then it's not empty and we return true, otherwise we send an alert to the user with a *helperMsg* to inform them of their error and return false.

Working Example:

```
<script type='text/javascript'>
function notEmpty(elem, helperMsg){
        if(elem.value.length == 0){
                alert(helperMsg);
                elem.focus();
                return false;
        }
        return true;
}
</script>
<form>
Required Field: <input type='text' id='req1'/>
<input type='button'
        onclick="notEmpty(document.getElementById('req1'), 'Please Enter a Value')"
        value='Check Field' />
</form>
```

Display:

Required Field:

Form Validation - Checking for All Numbers

If someone is entering a credit card, phone number, zip code, similar information you want to be able to ensure that the input is all numbers. The quickest way to check if an input's string value is all numbers is to use a regular expression /^[0-9]+$/ that will only *match* if the string is all numbers and is at least one character long.

JavaScript Code:

```
// If the element's string matches the regular expression it is all numbers
function isNumeric(elem, helperMsg){
        var numericExpression = /^[0-9]+$/;
        if(elem.value.match(numericExpression)){
                return true;
        }else{
                alert(helperMsg);
                elem.focus();
                return false;
        }
}
```

What we're doing here is using JavaScript existing framework to have it do all the hard work for us. Inside each string is a function called *match* that you can use to see if the string matches a certain regular expression. We accessed this function like so: elem.value.match(expressionhere).

We wanted to see if the input's string was all numbers so we made a regular expression to check for numbers [0-9] and stored it as *numericExpression*.

We then used the *match* function with our regular expression. If it is numeric then *match* will return true, making our if statement pass the test and our function *isNumeric* will also return true. However, if the expression fails because there is a letter or other character in our input's string then we'll display our *helperMsg* and return false.

Working Example:

```
<script type='text/javascript'>
function isNumeric(elem, helperMsg){
		var numericExpression = /^[0-9]+$/;
		if(elem.value.match(numericExpression)){
			return true;
		}else{
			alert(helperMsg);
			elem.focus();
			return false;
		}
}
</script>
<form>
Numbers Only: <input type='text' id='numbers'/>
<input type='button'
		onclick="isNumeric(document.getElementById('numbers'), 'Numbers Only Please')"
		value='Check Field' />
</form>
```

Display:

Numbers Only:

Form Validation - Checking for All Letters

This function will be identical to *isNumeric* except for the change to the regular expression we use inside the *match* function. Instead of checking for numbers we will want to check for all letters.

If we wanted to see if a string contained only letters we need to specify an expression that allows for both lowercase and uppercase letters: /^[a-zA-Z]+$/ .

JavaScript Code:

```
// If the element's string matches the regular expression it is all letters
function isAlphabet(elem, helperMsg){
		var alphaExp = /^[a-zA-Z]+$/;
		if(elem.value.match(alphaExp)){
			return true;
		}else{
			alert(helperMsg);
			elem.focus();
			return false;
		}
}
```

Working Example:

```
<script type='text/javascript'>
function isAlphabet(elem, helperMsg){
		var alphaExp = /^[a-zA-Z]+$/;
		if(elem.value.match(alphaExp)){
			return true;
		}else{
			alert(helperMsg);
			elem.focus();
			return false;
		}
}
</script>
<form>
Letters Only: <input type='text' id='letters'/>
<input type='button'
		onclick="isAlphabet(document.getElementById('letters'), 'Letters Only Please')"
		value='Check Field' />
</form>
```

Display:

Letters Only:

Form Validation - Checking for Numbers and Letters

By combining both the *isAlphabet* and *isNumeric* functions into one we can check to see if a text input contains only letters and numbers.

JavaScript Code:

```
// If the element's string matches the regular expression it is numbers and letters
function isAlphanumeric(elem, helperMsg){
		var alphaExp = /^[0-9a-zA-Z]+$/;
		if(elem.value.match(alphaExp)){
			return true;
		}else{
			alert(helperMsg);
			elem.focus();
			return false;
		}
}
```

Form Validation - Restricting the Length

Being able to restrict the number of characters a user can enter into a field is one of the best ways to prevent bad data. For example, if you know that the zip code field should only be 5 numbers you know that 2 numbers is not sufficient.

Below we have created a *lengthRestriction* function that takes a text field and two numbers. The first number is the minimum number of characters and the second is the maximum number of a characters the input can be. If you just want to specify an exact number then send the same number for both minimum and maximum.

JavaScript Code:

```
function lengthRestriction(elem, min, max){
	var uInput = elem.value;
	if(uInput.length >= min && uInput.length <= max){
		return true;
	}else{
		alert("Please enter between " +min+ " and " +max+ " characters");
		elem.focus();
		return false;
	}
}
```

Here's an example of this function for a field that requires 6 to 8 characters for a valid username.

Working Example:

```
<script type='text/javascript'>
function lengthRestriction(elem, min, max){
	var uInput = elem.value;
	if(uInput.length >= min && uInput.length <= max){
		return true;
	}else{
		alert("Please enter between " +min+ " and " +max+ " characters");
		elem.focus();
		return false;
	}
}
</script>
<form>
Username(6-8 characters): <input type='text' id='restrict'/>
<input type='button'
	onclick="lengthRestriction(document.getElementById('restrict'), 6, 8)"
	value='Check Field' />
</form>
```

Display:

Username(6-8 characters):

Form Validation - Selection Made

To be sure that someone has actually selected a choice from an HTML select input you can use a simple trick of making the first option as helpful prompt to the user and a red flag to you for your validation code.

By making the first option of your select input something like "Please Choose" you can spur the user to both make a selection and allow you to check to see if the default option "Please Choose" is still selected when the submit the form.

JavaScript Code:

```
function madeSelection(elem, helperMsg){
	if(elem.value == "Please Choose"){
		alert(helperMsg);
		elem.focus();
		return false;
	}else{
		return true;
	}
}
```

Working Example:

```
<script type='text/javascript'>
function madeSelection(elem, helperMsg){
	if(elem.value == "Please Choose"){
		alert(helperMsg);
		elem.focus();
		return false;
	}else{
		return true;
	}
}
</script>
<form>
Selection: <select id='selection'>
<option>Please Choose</option>
<option>CA</option>
<option>WI</option>
<option>XX</option>
</select>
<input type='button'
	onclick="madeSelection(document.getElementById('selection'), 'Please Choose Something')"
	value='Check Field' />
</form>
```

Display:

Selection: Please Choose

Form Validation - Email Validation

And for our grand finale we will be showing you how to check to see if a user's email address is valid. Every email is made up for 5 parts:

1. A combination of letters, numbers, periods, hyphens, plus signs, and/or underscores
2. The at symbol @
3. A combination of letters, numbers, hyphens, and/or periods
4. A period
5. The top level domain (com, net, org, us, gov, ...)

Valid Examples:

bobby.jo@filltank.net
jack+jill@hill.com
the-stand@steven.king.com

Invalid Examples:

@deleted.net - no characters before the @
free!dom@bravehe.art - invalid character !
shoes@need_shining.com - underscores are not allowed in the domain name

The regular expression to check for all of this is a little overkill and beyond the scope of this tutorial to explain thoroughly. However, test it out and you'll see that it gets the job done.

JavaScript Code:

```
function emailValidator(elem, helperMsg){
		var emailExp = /^[\w\-\.\+]+\@[a-zA-Z0-9\.\-]+\.[a-zA-z0-9]{2,4}$/;
		if(elem.value.match(emailExp)){
			return true;
		}else{
			alert(helperMsg);
			elem.focus();
			return false;
		}
}
```

Working Example:

```
<script type='text/javascript'>
function emailValidator(elem, helperMsg){
		var emailExp = /^[\w\-\.\+]+\@[a-zA-Z0-9\.\-]+\.[a-zA-z0-9]{2,4}$/;
		if(elem.value.match(emailExp)){
			return true;
		}else{
			alert(helperMsg);
			elem.focus();
			return false;
		}
}
</script>
<form>
Email: <input type='text' id='emailer'/>
<input type='button'
		onclick="emailValidator1(document.getElementById('emailer'), 'Not a Valid Email')"
		value='Check Field' />
</form>
```

Display:

Email:

Validating a Form - All at Once

If you've made it this far I commend you, but we're not done yet! The final step is to be able to perform all of these validation steps when the user is ready to submit their data.

Each form has a JavaScript event called *onSubmit* that is triggered when its *submit* button is clicked. If this even returns 0 or false then a form cannot be submitted, and if it returns 1 or true it will always be submitted. Wouldn't it be perfect if we could somehow make an if statement that said "If the form is valid submit it (1) else don't submit it (0)"? Well with a master *formValidator* function we can do just that.

formValidator will be somewhat like a list of checks that we want to do before a form is submitted. But before we can decide what we want to check for, we need to have our form!

HTML Form Code:

```
<form onsubmit='return formValidator()' >
First Name: <input type='text' id='firstname' /><br />
Address: <input type='text' id='addr' /><br />
Zip Code: <input type='text' id='zip' /><br />
State: <select id='state'>
		<option>Please Choose</option>
		<option>AL</option>
		<option>CA</option>
		<option>TX</option>
		<option>WI</option>
</select><br />
Username(6-8 characters): <input type='text' id='username' /><br />
Email: <input type='text' id='email' /><br />
<input type='submit' value='Check Form' /><br />
</form>
```

That's a lot of data to verify and the first thing we would probably want to check is that each field was at least filled out. To check for completion we will ensure no fields are empty and that the *SELECT* field has a selection. Here are the starting pieces of our master validation function *formValidator*.

JavaScript Code:

```
function formValidator(){
		// Make quick references to our fields
		var firstname = document.getElementById('firstname');
		var addr = document.getElementById('addr');
		var zip = document.getElementById('zip');
		var state = document.getElementById('state');
		var username = document.getElementById('username');
		var email = document.getElementById('email');

		// Check each input in the order that it appears in the form!
		if(isAlphabet(firstname, "Please enter only letters for your name")){
			if(isAlphanumeric(addr, "Numbers and Letters Only for Address")){
				if(isNumeric(zip, "Please enter a valid zip code")){
					if(madeSelection(state, "Please Choose a State")){
						if(lengthRestriction(username, 6, 8)){
							if(emailValidator(email,
"Please enter a valid email address")){
								return
true;
							}
						}
```

```
                                        }
                                }
                        }
                }

                return false;
}
```

The first part of this function is where we create easy references to our HTML inputs using the *getElementById* function. These quick references will make our next block of code much easier to read!

The second part uses a bunch of embedded if statements to see whether or not each field has the correct type of data. If every single one of those fields we check validates, then we'll return true and the form will be submitted successfully.

However, if just one of those if statements fails then the *return false* at the end of the function is reached and prevents the form for being submitted.

As you can see this function really does do quite a lot, definitely earning the title of *formValidator*. Notice how this one function references all of the functions we have covered in this lesson. By placing all of these checks in a central location you make your code easier to read and easier to change around in the future.

Now let's put all the necessary and HTML together and try it out!

All Together Now

Below we have taken the HTML form code and the new function *formValidator* and plugged in all the other form validation functions taught in this lesson that are referenced in *formValidator*.

HTML & JavaScript Code:

```
<script type='text/javascript'>

function formValidator(){
        // Make quick references to our fields
        var firstname = document.getElementById('firstname');
        var addr = document.getElementById('addr');
        var zip = document.getElementById('zip');
        var state = document.getElementById('state');
        var username = document.getElementById('username');
        var email = document.getElementById('email');

        // Check each input in the order that it appears in the form!
        if(isAlphabet(firstname, "Please enter only letters for your name")){
                if(isAlphanumeric(addr, "Numbers and Letters Only for Address")){
                        if(isNumeric(zip, "Please enter a valid zip code")){
                                if(madeSelection(state, "Please Choose a State")){
                                        if(lengthRestriction(username, 6, 8)){
                                                if(emailValidator(email,
"Please enter a valid email address")){
                                                        return
true;
                                                }
                                        }
```



```
                                        }
                                }
                        }
                }

                return false;

}

function notEmpty(elem, helperMsg){
        if(elem.value.length == 0){
                alert(helperMsg);
                elem.focus(); // set the focus to this input
                return false;
        }
        return true;
}

function isNumeric(elem, helperMsg){
        var numericExpression = /^[0-9]+$/;
        if(elem.value.match(numericExpression)){
                return true;
        }else{
                alert(helperMsg);
                elem.focus();
                return false;
        }
}

function isAlphabet(elem, helperMsg){
        var alphaExp = /^[a-zA-Z]+$/;
        if(elem.value.match(alphaExp)){
                return true;
        }else{
                alert(helperMsg);
                elem.focus();
                return false;
        }
}

function isAlphanumeric(elem, helperMsg){
        var alphaExp = /^[0-9a-zA-Z]+$/;
        if(elem.value.match(alphaExp)){
                return true;
        }else{
                alert(helperMsg);
                elem.focus();
                return false;
        }
}

function lengthRestriction(elem, min, max){
        var uInput = elem.value;
        if(uInput.length >= min && uInput.length <= max){
                return true;
        }else{
                alert("Please enter between " +min+ " and " +max+ " characters");
                elem.focus();
                return false;
        }
}

function madeSelection(elem, helperMsg){
```

```
                if(elem.value == "Please Choose"){
                        alert(helperMsg);
                        elem.focus();
                        return false;
                }else{
                        return true;
                }
}

function emailValidator(elem, helperMsg){
                var emailExp = /^[\w\-\.\+]+\@[a-zA-Z0-9\.\-]+\.[a-zA-z0-9]{2,4}$/;
                if(elem.value.match(emailExp)){
                        return true;
                }else{
                        alert(helperMsg);
                        elem.focus();
                        return false;
                }
}
</script>

<form onsubmit='return formValidator()' >
First Name: <input type='text' id='firstname' /><br />
Address: <input type='text' id='addr' /><br />
Zip Code: <input type='text' id='zip' /><br />
State: <select id='state'>
                <option>Please Choose</option>
                <option>AL</option>
                <option>CA</option>
                <option>TX</option>
                <option>WI</option>
</select><br />
Username(6-8 characters): <input type='text' id='username' /><br />
Email: <input type='text' id='email' /><br />
<input type='submit' value='Check Form' />
</form>
```

Display:

First Name:

Address:

Zip Code:

State: Please Choose

Username(6-8 characters):

Email:

Check Form

JavaScript Void 0

Hyperlinks like this one entice visitors to click because they know clicking it will lead them to a new page. However, sometimes when you are making a script, you would like to add functionality to your website that lets a hyperlink to be clicked and perform a useful action like update the sums on the webpage, without loading a new page.

It's these types of programming solutions that will utilize the JavaScript *Void 0* programming tool. This lesson will teach you some of the reasons to use the JavaScript *Void 0*programming strategy in your scripts.

Directly Executing JavaScript in a Browser

Web browsers allow you to execute JavaScript statements directly by entering JavaScript code into the browser's URL text field. All you need to do is place a *JavaScript:* before your code to inform the browser you wish to run JavaScript. You can play around with this right now by typing something like

```
JavaScript:alert("I'm learning at Tizag.com")
```

into the browser's URL text field and pressing Enter.

This is useful to you, the JavaScript scripter, because you can now set your hyperlinks's *href* attribute equal to a JavaScript statement! This means you can remove the hyperlink's ability to load a new page and reprogram it to do your "complete some actions directly on this page" bidding.

This practice can be seen in services like Gmail (Google Email) which does a great deal of interaction with hyperlinks, but has very few new pages loading. Here is an example link that does not load a new webpage.

JavaScript Code:

```
<a href="javascript: alert('News Flash!')">News Flash</a>
```

Display:

News Flash

This is interesting to learn, but it isn't much more than a gimmick. The true power of direct URL JavaScript statements is only unleashed when you use it to return a value. This is where *void 0* comes into play.

JavaScript Void 0 Explanation

Web browsers will try and take whatever is used as a URL and load it. The only reason we can use a JavaScript Alert statement without loading a new page is because *alert* is a function that returns a *null* value. This means that when the browser attempts to load a new page it sees *null* and has **nothing** to load.

The important thing to notice here is that if you ever do use a JavaScript statement as the URL that returns a value, the browser **will** attempt to load a page. To prevent this unwanted action, you need to use the *void* function on such statement, which will always return *null* and never load a new page.

Simple JavaScript Void 0 Simple Example

void is an operator that is used to return a *null* value so the browser will not be able to load a new page. An important thing to note about the *void* operator is that it requires a value and cannot be used by itself. Here is a simple way to use *void* to cancel out the page load.

JavaScript Code:

```
<a href="javascript: void(0)">I am a useless link</a>
```

Display:

I am a useless link

Simple JavaScript Void 0 Useful Example

This example shows how you would return a value using the void operator. *myNum* is a variable that we set to the value 10. We then use the same variable *myNum* in an *alert*operation.

JavaScript Code:

```
<a href="javascript: void(myNum=10);alert('myNum = '+myNum)">
Set myNum Please</a>
```

Display:

Set myNum Please

JavaScript String Functions

This may be old news to you, but inside every JavaScript string are several functions that are just waiting to do your bidding. This is because strings in JavaScript are actually objects with a bunch of properties and functions (also called methods) that can be accessed in the following general way:

Pseudo JavaScript Code:

```
<script type="text/javascript">
var myString = "Hello Thur!";

//This is how you would access a property
var myProperty = myString.property;

//This is how you would access a function
var myFunctionResult = myString.function(argument1, argument2);
</script>
```

As you can see, the most important part to accessing string properties and functions is to first create them. In this case, *myString* was our guinea pig.

What's a String Property?

A property is just some basic information about the object. For example, a string object has a *length* property which stores the number of characters in the string.

What's a String Function?

The string's functions are useful for finding out more about your string. For example, the string function split lets you take a string and chop it into pieces whenever characters that you supply, appear.

It is important to note that these functions do not actually change the string itself. Rather, they return new strings that you can store for use elsewhere. In our example, we stored the result of our make-believe function into *myFunctionResult*.

String Functions and Properties

The following lessons will teach you how to manipulate, count, reorder, replace, search and do just about anything else to your JavaScript strings. These advanced topics will open up a whole new world of choices for the dynamic webpages you have yet to build!

JavaScript String Length

Advanced scripters will often need to know how long a JavaScript string is. For example, if a webdeveloper was creating a submission form that required the username to be no longer than 20 characters, then she would need to check the length of the string before allowing the user to submit data.

String Length Property

The *length* property returns the number of characters that are in a string, using an integer. Below is the basic code for accessing this property.

JavaScript Code:

```
<script type="text/javascript">
var myString = "123456";

var length = myString.length;

document.write("The string is this long: " + length);

// Same thing, but using the property inside the write function
document.write("<br />The string is this long: " + myString.length);

</script>
```

Display:

The string is this long: 6
The string is this long: 6

String Changed? Length Might Change

If you were to reference the *length* property after concatenating (adding) some characters to the string, then the length property will reflect these changes. Think of this as a friendly reminder to only check the length of the string after you are sure it isn't going to be changed.

JavaScript Code:

```
<script type="text/javascript">
var myString = "123456";
document.write("The string is this long: " + myString.length);

myString = myString + "7890";
document.write("<br />The string is now this long: " + myString.length);

</script>
```

Display:

```
The string is this long: 6
The string is now this long: 10
```

JavaScript String Split Function

The ability to split up a string into separate chunks has been supported in many programming languages, and it is available in JavaScript as well. If you have a long string like "Bobby Susan Tracy Jack Phil Yannis" and want to store each name separately, you can specify the space character " " and have the *split* function create a new chunk every time it sees a space.

Split Function: Delimiter

The space character " " we mentioned will be our *delimiter* and it is used by the split function as a way of breaking up the string. Every time it sees the *delimiter* we specified, it will create a new element in an array. The first argument of the split function is the *delimiter*.

Simple Split Function Example

Let's start off with a little example that takes a string of numbers and splits when it sees the number 5. That means the *delimiter* for this example is 5. Notice that the *split* function returns an array that we store into *mySplitResult*.

JavaScript Code:

```
<script type="text/javascript">
var myString = "123456789";

var mySplitResult = myString.split("5");

document.write("The first element is " + mySplitResult[0]);
document.write("<br /> The second element is  " + mySplitResult[1]);
</script>
```

Display:

```
The first element is 1234
The second element is 6789
```

Make sure you realize that because we chose the 5 to be our *delimiter*, it is not in our result. This is because the *delimiter* is removed from the string and the remaining characters are separated by the chasm of space that the 5 used to occupy.

Larger Split Function Example

Below we have created a *split* example to illustrate how this function works with many splits. We have created a string with numbered words zero through four. The *delimiter* in this example will be the space character " ".

JavaScript Code:

```
<script type="text/javascript">
var myString = "zero one two three four";

var mySplitResult = myString.split(" ");

for(i = 0; i < mySplitResult.length; i++){
                document.write("<br /> Element " + i + " = " + mySplitResult[i]);
}
</script>
```

Display:

```
Element 0 = zero
Element 1 = one
Element 2 = two
Element 3 = three
Element 4 = four
```

JavaScript String Search

Knowing if something is or isn't in a string can be very important. If you have an online forum and don't want people to be able to create usernames that include swear words, you can use the search function to find bad words in usernames and reject them if any were found.

String Search Function

This string function takes a regular expression and then examines that string to see if there are any matches for that expression. If there is a match , it will return the position in the string where the match was found. If there isn't a match, it will return -1. We won't be going into great depth about regular expressions, but we will show you how to search for words in a string.

Search Function Regular Expression

The most important thing to remember when creating a regular expression is that it must be surrounded with slashes */regular expression/*. With that knowledge let's search a string to see if a common name "Alex" is inside it.

JavaScript Code:

```
<script type="text/javascript">
var myRegExp = /Alex/;
var string1 = "Today John went to the store and talked with Alex.";
var matchPos1 = string1.search(myRegExp);

if(matchPos1 != -1)
		document.write("There was a match at position " + matchPos1);
else
		document.write("There was no match in the first string");


</script>
```

Display:

```
There was a match at position 45
```

Notice that our regular expression was just the name "Alex". The search function then used this name to see if "Alex" existed in *string1*. A match was found, and the position of the match (45), was returned.

String Search Function: Alternative Searches

Another basic tool for regular expressions is the pipe character "|" (it's below the Backspace key on standard keyboards) which allows you to search for alternative words /RegExp1|RegExp2/. Instead of just searching for just one word, we can now use the pipe character to search for multiple words.

JavaScript Code:

```
<script type="text/javascript">
var myRegExp = /Alex|John/;
var string1 = "Today John went to the store and talked with Alex.";
var matchPos1 = string1.search(myRegExp);

if(matchPos1 != -1)
		document.write("There was a match at position " + matchPos1);
else
		document.write("There was no match in the first string");


</script>
```

Display:

```
There was a match at position 6
```

Notice that our regular expression had two names: Alex and John. The search function then used these names to try to find the first occurrence in the string *string1*. John came before Alex in our string, so its position (6), was returned.

Let's look at a couple more advanced examples.

Advanced Search Function Examples

The following examples play around with the names a little so you can clearly see how the search function operates.

JavaScript Code:

```
<script type="text/javascript">
var myRegExp1 = /Tom|Jan|Alex/;
var string1 = "John went to the store and talked with Alexandra today.";
var matchPos1 = string1.search(myRegExp1);

if(matchPos1 != -1)
		document.write("The first string found a match at " + matchPos1);
else
		document.write("No match was found in the first string");

var myRegExp2 = /Tom|Jan|Alex /;
var string2 = "John went to the store and talked with Alexandra today.";
var matchPos2 = string2.search(myRegExp2);
if(matchPos2 != -1)
		document.write("<br />The second string found a match at " + matchPos2);
else
		document.write("<br />No match was found in the second string");

var myRegExp3 = /Tom|Jan|Alexandra/;
var string3 = "John went to the store and talked with Alexandra today.";
var matchPos3 = string3.search(myRegExp3);
if(matchPos3 != -1)
		document.write("<br />The third string found a match at " + matchPos3);
else
		document.write("<br />No match was found in the third string");

var myRegExp4 = /Tom|Jan|Alexandra/;
var string4 = "John went to the store and talked with Alex today.";
var matchPos4 = string4.search(myRegExp4);
if(matchPos4 != -1)
		document.write("<br />The fourth string found a match at " + matchPos4);
else
		document.write("<br />No match was found in the fourth string");
</script>
```

Display:

```
The first string found a match at 39
No match was found in the second string
The third string found a match at 39
No match was found in the fourth string
```

In the first search, a match was found. This is because our search didn't specify that the name had to be exactly Alex, and because the name Alexandra contains "Alex", a match was found.

In the second search, we fixed this by adding a space after the name Alex to make our search for "Alex ", and no match was found.

In the third search, we changed our expression to include "Alexandra" and found a match.

In the fourth and final search, we changed our string to include "Alex" and we didn't find a match.

JavaScript String Replace

JavaScript's String Object has a handy function that lets you replace words that occur within the string. This comes in handy if you have a form letter with a default reference of "username". With the replace function you could grab get the person's name with an HTML form or JavaScript prompt and then replace all occurrences of "username" with the value that they entered.

String Replace Function

The string *replace* function has two arguments:

1. SearchFor - What word is going to be replaced. This can be a string or a regular expression.
2. ReplaceText - What the word will be replaced with. This needs to be a string.

replace returns the new string with the replaces, but if there weren't any words to replace, then the original string is returned.

Replace Function: String Replace

To start off with, let's just search for a string and replace it with the visitor's name. The first argument is what we are searching for, and the second argument is what we are going to replace.

JavaScript Code:

```
<script type="text/javascript">
var visitorName = "Chuck";
var myOldString = "Hello username! I hope you enjoy your stay username.";
var myNewString = myOldString.replace("username", visitorName);

document.write("Old string =  " + myOldString);
document.write("<br />New string = " + myNewString);

</script>
```

Display:

```
Old string = Hello username! I hope you enjoy your stay username.
New string = Hello Chuck! I hope you enjoy your stay username.
```

Notice that only the first occurrence of "username" was replaced. This is the drawback to using a string as your *searchFor* argument. But don't worry you can replace all occurrences of "username" if you decide to use regular expressions.

Replace Function: Regular Expression

Let's do the exact same replace, except this time, we'll use a a regular expression instead of a string. The only change we need to make is to surround our *searchFor* word with slashes / / instead of quotations " ".

JavaScript Code:

```
<script type="text/javascript">
var visitorName = "Chuck";
var myOldString = "Hello username! I hope you enjoy your stay username.";
var myNewString = myOldString.replace(/username/, visitorName);

document.write("Old string =  " + myOldString);
document.write("<br />New string = " + myNewString);

</script>
```

Display:

```
Old string = Hello username! I hope you enjoy your stay username.
New string = Hello Chuck! I hope you enjoy your stay username.
```

Darn! No luck. We've still only replaced the first "username" with Chuck instead of all of them. What's the solution to our problem? The answer is enabling the global property for our regular expression.

Replace Function: Global Regular Expression

By enabling the global property of our regular expression, we can go from replacing one match at a time to replacing all matches at once. To enable the global property, just put a "g" at the end of the regular expression.

JavaScript Code:

```
<script type="text/javascript">
var visitorName = "Chuck";
var myOldString = "Hello username! I hope you enjoy your stay username.";
var myNewString = myOldString.replace(/username/g, visitorName);

document.write("Old string =  " + myOldString);
document.write("<br />New string = " + myNewString);

</script>
```

Display:

```
Old string = Hello username! I hope you enjoy your stay username.
New string = Hello Chuck! I hope you enjoy your stay Chuck.
```

Finally we've succeeded! Remember that if you just want to replace one word, you should use a string or normal regular expression as your *searchFor* parameter. However, if you want to replace everything, be sure to write a regular expression and append a little g at the end!

JavaScript String indexOf

Ever want to find something in one of your JavaScript strings? Maybe even start your search halfway through the string? Well you're in luck! The string function *indexOf* lets you supply one required argument (the search query) and one optional argument (the offset) for all your simple searching needs.

If you are looking for a more powerful search feature, you should check out JavaScript's string search function, which has support for regular expressions, but not offsets.

String indexOf Function

As we mentioned above, *indexOf* has two arguments, with the second one being optional:

1. SearchString - What you would like to search for.
2. Offset (optional) - How far into the string you would like the search to begin. If you want to search the whole string, omit this argument.

indexOf Function Example

To start off with, we will be finding the position of the "www" in a typical URL string. The *indexOf* function will return the location of the first match that it encounters.

JavaScript Code:

```
<script type="text/javascript">
var aURL = "http://www.tizag.com/";
var aPosition = aURL.indexOf("www");

document.write("The position of www  =  " + aPosition);
</script>
```

Display:

The position of www = 7

If you'd like to follow along with how this function does it's counting, here's a complete breakdown:

h - 0 - no match
t - 1 - no match
t - 2 - no match
p - 3 - no match
: - 4 - no match
/ - 5 - no match
/ - 6 - no match
w - 7 - maybe match
w - 8 - maybe match
w - 9 - It's a match, return the position of the start of the match (7).

Let's see what happens when we add another www to the URL string to see if this function gets confused or not.

JavaScript Code:

```
<script type="text/javascript">
var aURL = "http://www.tizag.com/www.html";
var aPosition = aURL.indexOf("www");

document.write("The position of www  =  " + aPosition);

</script>
```

Display:

The position of www = 7

Looks like it returned the position of the first "www", just as we expected, so there are no surprises in this function. However, what would we do if we wanted to find the second "www"? Well, we'd have to use an offset, for starters.

indexOf Function Offset Example

If we use the *indexOf* function to find the first "www", then we can use that position + 1 as a way to skip the first "www" and find the second.

JavaScript Code:

```
<script type="text/javascript">
var aURL = "http://www.tizag.com/www.html";
var aPosition = aURL.indexOf("www");
var secondPos = aURL.indexOf("www", aPosition + 1);

document.write("The position of www  =  " + aPosition);
document.write("<br />The position of the second www  =  " + secondPos);

</script>
```

Display:

The position of www = 7
The position of the second www = 21

By using an offset of 8 (7 + 1), we were able to skip the first "www" to instead find the second. In case you're interested in what the function was actually looking at after we gave an offset, we have it below:

ww.tizag.com/www.html

There are only 2 w's left at the beginning of the string, due to the offset, making it impossible for a match to be found at the first set of "www".

JavaScript String Compare

Comparing strings in JavaScript is quite easy, as long as you know about the equals operator and the JavaScript If Statement. This is all you need to know to find out if two strings of your choosing are equal.

Comparing one String to another String

Below we have created a fake authentication system and use an if statement to see if the user's name will grant them access to a special message.

JavaScript Code:

```
<script type="text/javascript">
var username = "Agent006";
if(username == "Agent007")
		document.write("Welcome special agent 007");
else
		document.write("Access Denied!");
document.write("<br /><br />Would you like to try again?<br /><br />");

// User enters a different name
username = "Agent007";
if(username == "Agent007")
		document.write("Welcome special agent 007");
else
		document.write("Access Denied!");

</script>
```

Display:

Access Denied!

Would you like to try again?

Welcome special agent 007

Be sure you realize that when you are comparing one string to another, you use two equals operators "==" instead of just one "=". When you use two equals operators, it means you are comparing two values.

In this case, the English translation of our program would be: "If username is equal to Agent007, then print out a welcome message; otherwise, access is denied."

Case Insensitive String Compare

Above, we used a case sensitive compare, meaning that if the capitalization wasn't exactly the same in both strings, the if statement would fail. If you would like to just check that a certain word is entered, without worrying about the capitalization, use the *toLowerCase* function.

JavaScript Code:

```
<script type="text/javascript">
var username = "someAgent";
if(username == "SomeAgent")
		document.write("Welcome special agent");
else
		document.write("Access Denied!");

// Now as case insensitive
document.write("<br /><br />Let's try it with toLowerCase<br /><br />");
if(username.toLowerCase() == "SomeAgent".toLowerCase())
		document.write("Welcome special agent");
else
		document.write("Access Denied!");
</script>
```

Display:

Access Denied!

Let's try it with toLowerCase

Welcome special agent

By converting both strings to lowercase, we were able to remove the problem of it failing to find a match when the capitalization was slightly off. In this case, the "s" was capitalized in the username, but not capitalized in our if statement check.

JavaScript getElementById

Have you ever tried to use JavaScript to do some form validation? Did you have any trouble using JavaScript to grab the value of your text field? There's an easy way to access any HTML element, and it's through the use of *id* attributes and the *getElementById* function.

JavaScript document.getElementById

If you want to quickly access the value of an HTML input give it an *id* to make your life a lot easier. This small script below will check to see if there is any text in the text field "myText". The argument that *getElementById* requires is the *id* of the HTML element you wish to utilize.

JavaScript Code:

```
<script type="text/javascript">
function notEmpty(){
		var myTextField = document.getElementById('myText');
		if(myTextField.value != "")
				alert("You entered: " + myTextField.value)
		else
				alert("Would you please enter some text?")
}
</script>
<input type='text' id='myText' />
<input type='button' onclick='notEmpty()' value='Form Checker' />
```

Display:

document.getElementById returned a reference to our HTML element *myText*. We stored this reference into a variable, *myTextField*, and then used the *value* property that all input elements have to use to grab the value the user enters.

There are other ways to accomplish what the above script does, but this is definitely a straight-forward and browser-compatible approach.

]Things to Remember About getElementById

When using the *getElementById* function, you need to remember a few things to ensure that everything goes smoothly. You always need to remember that *getElementById* is a method (or function) of the *document* object. This means you can only access it by using *document.getElementById*.

Also, be sure that you set your HTML elements' id attributes if you want to be able to use this function. Without an id, you'll be dead in the water.

If you want to access the text within a non-input HTML element, then you are going to have to use the *innerHTML* property instead of *value*. The next lesson goes into more detail about the uses of *innerHTML*.

JavaScript innerHTML

Ever wonder how you could change the contents of an HTML element? Maybe you'd like to replace the text in a paragraph to reflect what a visitor has just selected from a drop down box. By manipulating an element's *innerHtml* you'll be able to change your text and HTML as much as you like.

Changing Text with innerHTML

Each HTML element has an *innerHTML* property that defines both the HTML code and the text that occurs between that element's opening and closing tag. By changing an element's *innerHTML* after some user interaction, you can make much more interactive pages.

However, using *innerHTML* requires some preparation if you want to be able to use it easily and reliably. First, you must give the element you wish to change an id. With that *id* in place you will be able to use the *getElementById* function, which works on all browsers.

After you have that set up you can now manipulate the text of an element. To start off, let's try changing the text inside a bold tag.

JavaScript Code:

```
<script type="text/javascript">
function changeText(){
		document.getElementById('boldStuff').innerHTML = 'Fred Flinstone';
}
</script>
<p>Welcome to the site <b id='boldStuff'>dude</b> </p>
<input type='button' onclick='changeText()' value='Change Text'/>
```

Display:

Welcome to the site **dude**

You now know how to change the text in any HTML element, but what about changing the text in an element based on user input? Well, if we combine the above knowledge with a text input...

Updating Text Based on User Input

By adding a Text Input, we can take to updating our bold text with whatever the user types into the text input. Note: We updated the function a bit and set the id to *boldStuff2*.

JavaScript Code:

```
<script type="text/javascript">
function changeText2(){
		var userInput = document.getElementById('userInput').value;
		document.getElementById('boldStuff2').innerHTML = userInput;
}
</script>
<p>Welcome to the site <b id='boldStuff2'>dude</b> </p>
<input type='text' id='userInput' value='Enter Text Here' />
<input type='button' onclick='changeText2()' value='Change Text'/>
```

Display:

Welcome to the site **dude**

Enter Text Here

Changing HTML with innerHTML

You can also insert HTML into your elements in the exact same way. Let's say we didn't like the text that was displayed in our paragraph and wanted to updated it with some color. The following code will take the old black text and make it bright white. The only thing we're doing different here is inserting the html element *span* to change the color.

JavaScript Code:

```
<script type="text/javascript">
function changeText3(){
		var oldHTML = document.getElementById('para').innerHTML;
		var newHTML = "<span style='color:#ffffff'>" + oldHTML + "</span>";
		document.getElementById('para').innerHTML = newHTML;
}
</script>
<p id='para'>Welcome to the site <b id='boldStuff3'>dude</b> </p>
<input type='button' onclick='changeText3()' value='Change Text'/>
```

Display:

Welcome to the site **dude**

This was a pretty simple example for changing the HTML of an element. All we did was take the old text that was in the *paragraph* tag and surround it in a *span* tag to change the color. However, there are many more things you can do by changing an element's HTML, so don't forget this useful tool!