# JavaScript Primer



**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

So what is JavaScript? Is it Java? What does it do? Is it difficult to learn? These questions will all be answered in this tutorial, and more. Just to get us started, JavaScript is basically a scripting language that helps kick HTML into **overdrive**. With it, elements in a document can be programmatically accessed and manipulated, bringing a dull web page to life. If you are content with using simply html to create web pages, you are ready to head out to our special tutorial , how to make fire with sticks; the rest of us, lets rock.

In this tutorial, we'll be looking at the following topics:

-Tutorial introduction
-FAQs about this language.
-Getting Started: Setting Up your code.
-Introducing objects-what JavaScript's made of
-Using the document object to explain objects.
-Functions and creating your own functions

**FAQs about this language**



FAQs about this language.

**What is JavaScript?**
So what exactly is JavaScript? Well, it's is a scripting language developed by Netscape to add interactivity and power to web documents. Examples of JavaScript include live clocks, rollover effects, scrollers, form validations, and so on. JavaScript differs from most other programming languages in that it is relatively easy to master, even for people who have absolutely no programming experiences whatsoever.

**Why learn JavaScript?**
The first few words that come to mind are: "Freedom baby, freedom!" With html, you are restricted to creating static, non interactive webpages. This, in today's internet standards, is unacceptable. With JavaScript, you can change that. Imagine being able to break free and allow your creativity to dictate what you put on your webpage, instead of the other way round. And the best part is, JavaScript can be learned by anyone-yes, I said anyone!

**What's the difference between Java and JavaScript?**
Java is completely different from JavaScript-It's a lot more powerful, more complex, and unfortunately, a lot harder to master. It belongs in the same league as C, C++, and other more complex languages. Also, you need to compile a Java program before you can run it, whereas with JavaScript, no compilation is needed-simply open up a text editor, type it, save it, and your browser is ready to run it!

**Can my JavaScript programs run on both Netscape and Internet Explorer browsers?**
Unfortunately, not necessarily. JavaScript was created by Netscape, so it is most compatible with Netscape. Internet Explorer 4.x supports 99% of what JavaScript has to offer, although IE 3.x is not quite as adorable. A good rule to follow is to always test your codes using both browsers before uploading it onto the internet. You will be surprised how many websites fail to do this, annoying surfers and not even realizing that their scripts are going haywire behind their backs! (this might pertain to me too)

-Tutorial introduction
-FAQs about this language.
-Getting Started: Setting Up your code.
-Introducing objects-what JavaScript's made of
-Using the document object to explain objects.
-Functions and creating your own functions

**Getting Started: Setting up your code**

**WA Help Forum**
Having trouble with anything? Visit our help forum to get the answers you need.

**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

**Home / JavaScript Tutorials / JavaScript Primer**



## Getting Started: Setting Up your code.

Where do your JavaScript codes go? Well, basically anywhere inside the <html> tags of your page. The beginning of your code begins with <script> and ends with </script>

```
<html>
<head><title>This is an example page</title></head>
<body>
Welcome to the JavaScript course!
<script language="JavaScript">
<!--
document.write("Hi there. This text is written using JavaScript!")
//-->
</script>
</body>
</html>
```

**Output:** Hi there. This text is written using JavaScript!

As you can see, we began our script with the tag <script language="JavaScript"> The part in orange is purely optional, but you should include them to remind yourself-and others that you are using JavaScript now. The second and next to last lines of the above example are <!-- and //-->, which are the comment tags. These tags should ALWAYS be included to help hide your code against older browsers of both Netscape and IE. If you don't include them, and someone is using an old browser, the browser will just "dump" all your code as text onto the screen, in other words, not a pretty sight! The only "functional part" of this script is the document.write(".....") part. It basically writes to the page whatever you put inside the quotation marks. Don't worry so much about why this is so yet, we will discuss this in detail later. We end this entire code with </script> This terminates your script, and brings you back to html.

Like html, you can insert comments in your JavaScript codes. Comments are ignored by the browser, and only used as reminder or documentation for your code. To basic syntax of inserting comments is either:

```
//
```

for single-lined comments, or

```
/*

    */
```

for multiple ones.

For example:

```
<script language="JavaScript">
<!--
//this script does nothing and is useless!
/*Hay, don't involve me
in this!*/
//-->
</script>
```

Ok, we are now ready to proceed to some real programming!

JavaScript, like many programming languages, relies on objects, functions, and event handlers to create workable programs. If you know absolutely *nothing* about these technical terms and how to use them, don't worry. By the time we're through with this tutorial, you *will*.

-Tutorial introduction
-FAQs about this language.
-Getting Started: Setting Up your code.
-Introducing objects-what JavaScript's made of
-Using the document object to explain objects.
-Functions and creating your own functions

**Introducing objects- what JavaScript's made of** 

## Introducing objects-what JavaScript's made of



Eventually we're going to have to face it, so lets just get it over with. I will now explain to you the fundamental structure of ALL JavaScript codes, because in order to understand JavaScript, you'll have to understand the structure of it. JavaScript is a language that revolves around the "object" concept, meaning that the majority of what you do with JavaScript involves merely picking of one JavaScript's pre-made code and accessing it. Uh? You say. Don't worry, I'll clear it up.



**Using the document object to explain objects**

## Home / JavaScript Tutorials / JavaScript Primer

### Using the document object to explain objects.



The document object is one of the most important objects of JavaScript. Lets have a look at a very simple JavaScript code. The below script writes out a line of text onto the web page:

```
document.write("Hi there. This text is written using javascript!")
```

- "document" is the object in the above example.
- "write" is the method of this object. (Think of it as the arm and legs of this object that allows it to do something-anything.

JavaScript is a language of objects, and all objects (95%) of them have both methods and properties. "Document" is just one of the many objects that essentially make up JavaScript as a language-learn these objects, and you are a JavaScript programmer! It is the object that controls the layout of a webpage-background color, text, images etc. Now, the word "write" is a method of the document object. Most objects have more than one method and property (You'll see what property is very soon), and this is true for the document object as well. Lets have a look at some of the others that the document object possesses.

**Document Object**

| Properties | Methods |
|---|---|
| **bgColor** (changes bgcolor) | **write** (writes something) |
| **lastModified** (gives date of document's last change) | **writeln** (writes in new line) |
| **referrer** (gives URL of the page that linked here) | |
| **fgColor** (changes foreground color (text)) | |

The column on the left are properties of the document object. They are static attributes of the object. Lets say you want to write out the date and time of the last modification to your page onto the screen. Here's what you would do:

```
<script>
var example
example=document.lastModified
document.write("This page was last modified: "+example)
</script>
```

**Output: This page was last modified: 10/30/97 03:40:56**

Whenever you update your page and save it, this script will display the date and time of this occurrence. Something you can add to the end of your page right now! In this example, "document.lastModified" is a property of

"document." Its a static attribute of the current document. In order to show this property, we used the non-static method to write this information out. Notice that we used "+" to put together "This page was last modified:" and example. The "+" sign is used to "combine" strings into one-similar to math, in a sense.

As you can see, as long as you know the object name and what methods and properties it contains, you know how to use it! The majority of commands in JavaScript are referenced by first specifying the object name, than a dot (.), plus the method/property.

Lets try playing around with another document method then. Looking at the above chart, lets use "referrer", which gives the URL of the page that you came from to get here. Lets see what this might be useful for:

```
<script>
document.writeln("Please thank this site for adding a link to me!")
document.write(document.referrer)
</script>
```

Ok, maybe not so useful like this, but you get the point.

As you learn more about JavaScript, you will bump into more objects. All objects' properties and methods are accessed in the same manner as above- first by specifying the object name, then a dot (.), then the method or property you want to use from it.

Remember, JavaScript is a language of objects- 95% of what you'll want to do with JavaScript will involve simply picking the right object and using it!

-Tutorial introduction
-FAQs about this language.
-Getting Started: Setting Up your code.
-Introducing objects-what JavaScript's made of
-Using the document object to explain objects.
-Functions and creating your own functions

**Functions and creating your own functions** 

**Home / JavaScript Tutorials / JavaScript Primer**



## Functions and creating your own functions

Ok, what are functions? Well, functions are chunks of code that together achieve a more complex task, and are not executed until you call upon them. Its like your trusted car-It can take you places, but it only does that when you drive it-it does not start driving by itself. Unlike regular lines of codes, the execution of them are deferred until you want them to.

The basic syntax of a function is:

```
function whatever_name()
{
function codes are entered here
}
```

Lets do a basic example:

```
function test()
{
document.write("Hello there!")
}
```

We've just created a simple function. Note that if only this were within your <script></script> tags, you will not see "Hello there" on your screen. Like the car you own, it does not drive by itself. To "drive" it, you have to call it. Take a look:

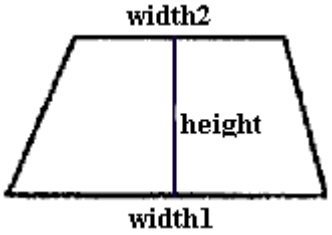
```
function test()
{
document.write("Hello there!")
}
test()
```

Now the function is "summoned", and you will see the words "Hello there!"

## Functions with Parameters

The beauty of functions is that it can receive data from the "outside" world and process it. The term parameter is used as a definition for "what goes into the function." You can supply different data into the function each time. What this means is that you can create one function, and use it over and over again. An example should clear up this. Lets do an example that calculates the area of a trapezoid. The formula is : (width1+width2)*height/2

```
function area(w1,w2,h)
{
var area=(w1+w2)*h/2
alert(area+" sq ft")
}
```



```
area(2,3,7)
area(5,7,4)
area(3,2,1)
```

Click here for output>>

- w1, w2, and h are what the function use to store the input it receives from outside the function. They are the parameters. In the first function call, w1=2, w2=3, and h=7
- You can have as many parameters as you like; of course, you have to specify that in your function. In this case, there are three: w1, w2, and h.
- Another reminder: Look at: alert(area+" sq ft"). How can we add characters? Well, in JavaScript, we could, and the result is the combination of the two strings into one. For example: alert ("Hi there"+" What is your age?") is the same as: alert("Hi there What is your age?")

You are not limited to passing actual values into a function that accepts parameters. You can also pass in variables can contain these values. This adds flexibility to your functions. Lets look at an example of such that'll demonstrate this concept:

Click for example>>

```
<script>
var x=prompt("Please enter your age")
function calsecs(age)
{
var temp=age*365*24*60*60
alert("You have lived "+temp+" seconds!")
}
calsecs(x)
</script>
```

Notice that we passed in x into the function, which can change each time a person enters a different age. One important thing to take notice is that the actual variable name that's passed in, "x", is not, and does not have to be the same name represented in the function. For example, instead of using "x", we used the name "age" in the declaration of the function . The name "age" is simply a placeholder that "holds" the actual variable that gets passed in. Simply put, the two names do not have to be the same, despite the fact they represent the same "thing". Lets say I call the function above another three times:

calsecs(z) //this time, "age" holds the variable "z"

calsecs(w) //this time, "age" holds the variable "w"

calsecs(m) //this time, "age" holds the variable "m"!

If the two names have to be the same, then you are restricted to passing only that variable in, instead of "z", "m", "w", or whatever you want!

## Functions that return a value

Ok, I know you're sick and tired by now of all the function talk, but this will be the last one, and a very important one indeed. One thing to realize is that a function itself can return a value. Lets see what I mean:

```
function diameter(x)
{
temp=2*x
return temp
}
```

Look at the part in red. This function will take in a radius of a circle, and return the diameter. Lets see how this function may be used in a script:

```
<script>
var d=diameter(5) //d will contain 10
var a=3.14*diameter(5) //a will contain 31.4
</script>
```

See, by setting a function to return a value, the function itself becomes, in a sense, a variable that will store what the function itself returned. Is, and why is this useful, you ask? Because now you can have a function process something, return the "processed goods", and have the script continue on manipulating that variable. Normally, once a function processes something, the processing of that "something" ends there. BTW, if you didn't get a thing I just said in the last sentence, don't worry about it! Just know how to return a value from a function, and sooner or later, you will wonder how you lived without this knowledge for so long. One thing to take note: A function can only return one value, just like a variable can only contain one value at a time. For example, the following is illegal:

```
function illegal(x)
{
temp=2*x
temp2=2*2*x
return temp
return temp2
}
```

-Tutorial introduction
-FAQs about this language.

-Getting Started: Setting Up your code.
-Introducing objects-what JavaScript's made of
-Using the document object to explain objects.
-Functions and creating your own functions

**End Of Tutorial**



# Creating interactive alert, confirm, and prompt boxes using JavaScript

**WA Help Forum**
Having trouble with anything? Visit our help forum to get the answers you need.

**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

You've undoubtedly seen them, used them, and gotten annoyed by them while surfing the net. Now its time to learn how to create them- interactive dialog boxes, that is. With JavaScript, you can pump into your pages some life by using JavaScript controlled boxes. For example, you can have a box pop up asking surfers to type in his/her name and then display it accordingly. You can also have a confirmation box that gives users a choice whether or not to proceed with a specified action, such as enter a restricted site. These are among the endless possibilities that are out there; fortunately, there are only three methods (commands) you'll need to learn to be able to achieve all of them.



**The alert, confirm, and prompt boxes**

## Home / JavaScript Tutorials / Creating interactive boxes



### The alert, confirm, and prompt boxes

The three "commands" involved in creating alert, confirm, and prompt boxes are:

- window.alert()
- window.confirm()
- window.prompt()

Lets look at them in detail:

The first one is:

window.alert()

This command pops up a message box displaying whatever you put in it. For example:

```
<body>
<script>
window.alert("My name is George. Welcome!")
</script>
</body>
```

Click here for output>>

As you can see, whatever you put inside the quotation marks, it will display it.

The second one is:

window.confirm()

Confirm is used to confirm a user about certain action, and decide between two choices depending on what the user chooses.

Click here for output: Click here>>

```
<script>
var x=window.confirm("Are you sure you are ok?")
if (x)
window.alert("Good!")
else
window.alert("Too bad")
```

```
</script>
```

There are several concepts that are new here, and I'll go over them. First of all, "var x=" is a variable declaration; it declares a variable ("x" in this case) that will store the result of the confirm box. All variables are created this way. x will get the result, namely, "true" or "false". Then we use a "if else" statement to give the script the ability to choose between two paths, depending on this result. If the result is true (the user clicked "ok"), "good" is alerted. If the result is false (the user clicked "cancel"), "Too bad" is alerted instead. (For all those interested, variable x is called a Boolean variable, since it can only contain either a value of "true" or "false").

The third one is:

window.prompt()

Prompt is used to allow a user to enter something, and do something with that info:

Click here for output: Click here>>

```
<script>
var y=window.prompt("please enter your name")
window.alert(y)
</script>
```

The concept here is very similar. Store whatever the user typed in the prompt box using y. Then display it.



**A more complex example of interactive dialog boxes** 

## Home / JavaScript Tutorials / Creating interactive boxes

### A more complex example of interactive dialog boxes



Using these three basic methods (window.alert(), window.confirm(), and window.prompt()), we can actually create some more "useful" scripts.The below illustrates how to add a confirmation box to a web page that asks a user whether he/she really wants to enter the page. If not, the surfer is taken to Yahoo instead:

```
<html>
<head>
<script>
var stay=confirm("The following site contains appalling material suitable only for webmasters. Please 'ok' to enter, 'cancel' to exit immediately!")
if (!stay)
window.location="http://www.yahoo.com"
</script>
</head>
<body>
Appalling material here
</body>
</html>
```

In the above example, clicking "cancel" in the confirmation box will take the surfer to yahoo, and clicking ok will continue with the loading of the "appalling" document.

There are two lines of code above that require explaining:

```
if (!stay)
window.location="http://www.yahoo.com"
```

Translating the above into pure English, it reads "if the surfer pressed "cancel", take him to Yahoo. Recall that confirmation boxes always return either a value of "true" or "false" to a variable. If the user pressed cancel, a value of "false" is returned. In JavaScript, we can detect this false value by using the exclamation mark (!). Then by using the prebuilt window.location property, we can change the contents of the document to the specified url accordingly.

As you learn more about JavaScript, you will be able to build more complex JavaScript boxes that will irritate your surfers even more.

Ok, having looked at the three methods (commands), let me introduce you to an alternate way of writing the methods that save you some time.

In all our examples above, we wrote the three methods as follows:

- window.alert()
- window.confirm()
- window.prompt()

Actually, we could ALWAYS simply write the following instead:

- alert()
- confirm()
- prompt()

For example:

```
<script>
alert("this is really good on my wrist!")
</script>
```

The three commands we have just gone through all have practical (and annoying sometimes) uses that I'm sure you've seen on the web. Try them out, play around with them, but don't get too close to them!

-Tutorial introduction
-The alert, confirm, and prompt boxes
-A more complex example of interactive dialog boxes

**End of Tutorial**

# Understanding "event handlers" in JavaScript



**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

So, what are event handlers? Very powerful and useful! They are *JavaScript code that are not added inside the <script> tags, but rather, inside the html tags*, that execute JavaScript when something happens, such as pressing a button, moving your mouse over a link, submitting a form etc. The basic syntax of these event handlers is:

name_of_handler="JavaScript code here"

For example:

onClick="alert('hello!')"

**Event Handlers:**

| | |
|---|---|
| **onclick:** | Use this to invoke JavaScript upon clicking (a link, or form boxes) |
| **onload:** | Use this to invoke JavaScript after the page or an image has finished loading. |
| **onmouseover:** | Use this to invoke JavaScript if the mouse passes by some link |
| **onmouseout:** | Use this to invoke JavaScript if the mouse goes pass some link |
| **onunload:** | Use this to invoke JavaScript right after someone leaves this page. |



**onClick event handler** 

## Home / JavaScript Tutorials / Understanding Event Handlers



**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

### onClick Event handler

Ok, lets see how the onclick event-handler can help us. **Remember, any event handlers are added inside html tags, not inside <script></script> ( there is an alternate way, which will not be discussed in this section).** First of all, does that mean we can simply dump event handlers anywhere within any html tag? Noooo! onClick handlers execute something only when users *click* on form buttons, check boxes, etc, or text links, therefore they can only be inserted in these tags, for example, <a>,<input type=..>. Lets see an example of an onClick event handler:
.

Click here for output: Click me

```
<script>
function inform()
{
alert("You have activated me by clicking the grey button! Note that the event handler is added within the event that it handles, in this case, the form button event tag")
}
</script>

<form>
<input type="button" name="test" value="Click me" onclick="inform()">
</form>
```

The function inform() is invoked when the user clicks the button.

All examples you have seen up to now use this handler, the onClick handler, to illustrate the examples. Ok, let me show you another example that will make use of the checkbox element.

Try this: (it will change the background color of a document interactively)

```
<form name="go">
<input type="checkbox" name="C1"
onclick="document.bgColor='lightblue'">
<input type="checkbox" name="C2"
onclick="document.bgColor='lightyellow'">
input type="checkbox" name="C3"
onclick="document.bgColor='lightgreen'">
</form>
```

I've put the actual demo of this example onto another window. Click the button to see it.

Click here for output: Click here>>

- We used the onclick handler to change the background color. Notice that we just wrote in plain English the name of the bgcolor...you can do that, for most colors.
- This is a crudely implemented bgcolor changer. Whenever you select additional checkboxes, the previous one stays checked. You can use buttons or radio boxes to alter that.



**onLoad event hander** 

.

## Home / JavaScript Tutorials / Understanding Event Handlers

### onLoad Event handlers

The onload event handler is used to call the execution of JavaScript after a page /frameset /image has completely loaded. It is added like this:

<body onload="inform()"> //Execution of code //will begin after the page has loaded.

<frameset onload="inform()"> //Execution of code //will begin after the current frame has loaded.

<img src="whatever.gif" onload="inform()"> //Execution of code will begin after the image //has loaded.

Lets see an example of an onload handler:

```
<html>
<head><title>Body onload example</title>
</head>
<body onload="alert('This page has finished loading!')">
Welcome to my page
</body>
</html>
```

As soon as the page has finished loading, it will alert you saying so.



**onMouseover, onMouseout event handler**



**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

# Home / JavaScript Tutorials / Understanding Event Handlers



## onMouseover,onMouseout

These handlers are used exclusively with links.. The following example writes something to the status bar (at the bottom of your screen) whenever a mouse cursor hovers over the link, and deletes it when the mouse moves away.

Output: Dont't Click Here

```
<a href="blabla.htm" onmouseover="status='Do not click here, its empty!';return true" onmouseout="status=' '">Don't Click Here</a>
```

Several new concepts arise here, so I'll go over each one of them.

- the "status" refers to window.status, which is how you write to the status bar.
- Note that instead of calling a function, we wrote directly the JavaScript code within the handler :"status='Do not click here, its empty!';return true" This is ok, but it is important that you separate statements with ;. You could have, alternatively, written everything up until "return true" as a function and then calling it:.
  function writestatus()
  {
  status="Do not click here, its empty!"
  }
  and then: onmouseover="writestatus();return true"
- So you're thinking, "what is return true?" Good question. You need to add this line of code to set the status property with the mouseover effect. Uh? I know, don't worry so much now, it really isn't important. Just remember you need this to "activate" the status onmouseover effect.
- onmouseout="status=' '" clears the status after the mouse leaves the link. Whenever the mouse moves away from the link, the status bar is "reset" again. If you don't insert this code, the status bar will still have those words you entered into it even after taking away the cursor.
- Whenever we have nested quotations, the inner ones are always singled. Ie:
  onclick="document.write('hello')"



**onUnload event handler**

## Home / JavaScript Tutorials / Understanding Event Handlers

### onUnload event handler

onunload executes JavaScript *immediately after* someone leaves the page. A common use (though not that great) is to thank someone as that person leaves your page for coming and visiting.

<body onunload="alert('Thank you. Please come back to this site and visit us soon, ok?')">

There are other event handlers, many belonging to forms. These event handlers are discussed in the tutorial **Accessing and validating forms using Javascript.**

For convenient reference, here is a complete list of all the event handlers in JavaScript:

| Event Handlers | Can be used with these tags: |
|---|---|
| onAbort | images |
| onBlur | windows, all form elements, frames |
| onClick | buttons, radio buttons, checkboxes, submit buttons, reset buttons, links |
| onChange | text fields, textareas, select lists |
| onError | windows, images |
| onFocus | windows, frames, and all form elements |
| onLoad | body, images |
| onMouseover | areas, links |
| onMouseout | links |
| onReset | forms |
| onSelect | text fields, textareas |
| onSubmit | submit button |
| onUnload | body |





**End of Tutorial**

# Creating time-dependent scripts using JavaScript



**Jump to...**


If you were like me, before I learned JavaScript, one of the most eager things I wanted to know was how to access the time using JavaScript-displaying it, working with it and so on. Well, it was, in fact very easy, and I'll tell you why. JavaScript already has a built in object that gets the date and time for you -all you have to do is use it! In this tutorial, we'll take a look at:



**The date object** 

## The Date Object

To use this date object, you need to first create an *instance* of it. Uh? Instant noodles? Well, not quite. What we have to do is tell JavaScript to activate this object:

To activate a Date Object so you can use it, do this:

```
var mydate= new Date()
```

Whenever you want to create an instance of the date object, use this important word: **new** followed by the object name(). Now, I'm going to list some of the methods of the date object that you can use:

**Some methods for the date object:** (uses the time displayed on your cmpt)

| | |
|---|---|
| **getDate()** | returns the day of the month |
| **getDay()** | Returns the day of the week |
| **getHours()** | returns the hour **(Starts from 0-23)!** |
| **getMinutes()** | returns the minutes |
| **getSeconds()** | returns the seconds |
| **getMonth()** | returns the month. **(Starts from 0-11)!** |
| **getYear()** | returns the year |
| **getTime()** | returns the complete time (in really weird format) |

Ok, lets say we want to write out today's date:

```
var today_date= new Date()
var myyear=today_date.getYear()
var mymonth=today_date.getMonth()+1
var mytoday=today_date.getDate()

document.write("Today's date is: ")
document.write(myyear+"/"+mymonth+"/"+
mytoday)
```

output:

- Come back tomorrow, and the date shown will be different.
- Lets have a look at the codes in red. "today_date" is the instance of the date object- you could name it anything you want. After creating an instance, that instance can use all the methods. So:
  var myyear=today_date.getYear()
  means "get the year and put it in variable myyear."
- We added "1" to the month, since in JavaScript, it counts months starting from "0", not "1".

**Dynamically writing out HTML codes** 

## Dynamically writing out html codes

With the above knowledge, and a little HTML, you can display different images according to the date/time. Here is a common example: Displaying a different image depending on whether it is mourning or night.

Lets first get two images: (One for mourning, one for night.)


```
var current= new Date()
var day_night=current.getHours()
if (day_night<=12)
document.write("<img src='day.gif'>")
else
document.write("<img src='night.gif'>")
```

Output:

There is one very important concept here:

- Take a look at the part in red. What we have done here is *dynamically* written out the html code required to display an image *(*not to be mistaken with dynamic html). Here is where JavaScript becomes amazing. You don't have to hard code your html tags into your webpage- allow JavaScript to think and write it out for you. **Notice that we used the "write" method to do so, with the html code enclosed in " " This is how we write html in JavaScript.**
- So basically, if its mourning (before noon), "day.gif" will be shown, and at night, "night.gif."
- Don't you get mesmerized looking at that sunset picture?

What else can you do with this object? Plenty. How about a live clock? Well, that is really fairly easy to do, provided we know this very important function:



**The setTimeout function**

**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

## setTimeout function

setTimeout, a method of the window object, basically delays the execution of a function or statement until the specified time has passed. The basic syntax of this function is:

setTimeout("expression", delaytime)

"expression" is the function/statement you want delayed, and delaytime is the delay time, **in milliseconds.**

For example, this will output "Three seconds have gone!" after 3 seconds.


```
document.three.hi.value=""
//simply clears the text box
function talk()
{
setTimeout("document.three.hi.value='Three seconds have gone!'",3000)
}

<form name="three">
<input type="text" size="28">
<input type="button" value="Start" onClick="talk()">
</form>
```

Let's do a example that, by placing the setTimeout in a special position, continuously writes to a box, incrementally itself every second.


```
var c=0
document.go.timer.value=""
function count()
{
document.go.timer.value=c
c=c+1
setTimeout("count()",1000)
}
```

```
<form name="go">
<input type="text" name="timer" size="12">
<input type="button" value="Start" onClick="count()">
</form>
```

- A very important point is that the setTimeout method executes count() after one second, but look at the place where setTimeout is placed-inside count()! This will execute everything within function count(), including setTimeout itself, *every* one second. Basically, this forces the function to loop back and start at the beginning each time it encounters this method. If you don't get it, study it a little harder-you'll see it!
- The above function will run forever, until you reload or leave. This is exactly the function we need to create a clock, since we need a function that will continuously update itself. This function is so important, because whenever you write something that needs continuous refreshment, all you have to do is put a setTimeout function inside another function and call it. Now, a little thought will tell us that we can control this function, if we want to, so that it does not run forever- namely, by putting it in an if-else-statement. Note that to get "1" second, we used "1000". (1 millisecond*1000) equals 1 second.

Using setTimeout, we can implement a live clock. Ok, now that we have all the tools, lets do an example of that:

And, like always, here's the complete date object:

**Date object**

| **Methods** |
|---|

**getDate**
**getDay**
**getHours**
**getMinutes**
**getMonth**
**getSeconds**
**getTime**
**getTimezoneOffset**
**getYear**
**parse**
**setDate**
**setHours**
**setMinutes**
**setMonth**
**setSeconds**
**setTime**
**setYear**
**toGMTString**
**toLocaleString**
**toString**
**UTC**
**valueOf**

Lets attempt to make a live clock then, shall we?

We will now attempt to show an example of each of the two concepts we had learned in the last section-using the date object, and dynamically writing out html.

-Tutorial introduction
-The date object
-Dynamically writing out html codes
-The setTimeout function
-Adding a live clock with the help of a form
-Example: To watermark or not to watermark?

**Adding a live clock with the help of a form** 

## Adding a live clock with the help of a form

Adding a live clock using forms is quite simple...the basic concept is to continuously write to the form every 1 second, using the updated time from your own computer. Last section, we already discussed the essence of how to achieve calling a function continuously (by using the setTimeout function in a special manner) -now all we need to do is use that knowledge. You may want to cut and paste the below code and play around with it:

//clock

8:22:39 PM

```
<form name="Tick">
<input type="text" size="12" name="Clock">
</form>

<script>
function show()
{
var Digital=new Date()
var hours=Digital.getHours()
var minutes=Digital.getMinutes()
var seconds=Digital.getSeconds()
var dn="AM"
if (hours>12)
{
dn="PM"
hours=hours-12
//this is so the hours written out is
//in 12-hour format, instead of the default //24-hour format.
}
if (hours==0)
hours=12
//this is so the hours written out
//when hours=0 (meaning 12a.m) is 12
if (minutes<=9)
minutes="0"+minutes
if (seconds<=9)
seconds="0"+seconds
document.Tick.Clock.value=
hours+":"+minutes+":"+seconds+" "+dn
setTimeout("show()",1000)
}
show()
```

```
</script>
```

- Believe it or not, that's all there really is to it!
- Look at the part in blue. We first set "dn=AM". If the hours is > 12, meaning its afternoon, we set "dn=PM". If minutes/seconds is greater than 9, we added a "0" to it...why? because we want to pad it, just to make it look better. That's the basic structure of function show()
- The most important part, the setTimeout method, was already discussed prior to this page. Basically, this script is run every one second, which gives the illusion that the clock is actually ticking.
- **Remember, the format of hours in JavaScript is from 0~23, not 1~24! That's the reason why we set it to display "12" whenever hours=0.**

You, by no means, have to use a form when implementing your clock, but a form has many advantages: it is easy to add to your code, and more importantly, since all it does it "write" to a form, there is virtually no delay in time every second as it writes a new value to the text form. It is possible to implement a live clock that uses images. For an example of such, click here.

-Tutorial introduction
-The date object
-Dynamically writing out html codes
-The setTimeout function
-Adding a live clock with the help of a form
-Example: To watermark or not to watermark?

**Example: To watermark or not to wartermark?** 

## Dynamically writing out html



**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

Dynamically writing out html using JavaScript is very important-and powerful. With it, html turn dynamic. Instead of hard-coding lines of html onto your webpage, we can use JavaScript to intelligently write it out, changing the appearance of a html document dynamically. The basic concept of dynamically writing out html is amazingly simple:

- Chose any html line (ie: <img src="mygif.gif"> ) that you want to write out dynamically.
- Use the document.write() method of JavaScript to selectively write it out, instead of hard coding it into your document.

The concept may seem simple, and it is, but please realize that it is very useful and powerful! What you may have not realized is that, using JavaScript, you can dynamically write out any html code, yes, even the <html> tag itself!. All you need to know is html itself, and the document.write method of JavaScript to accomplish amazing things...

Lets have a look at an example that involves writing out the body tag dynamically. If you are using Microsoft Internet Explorer, you should be familiar with the concept "watermark" background. In essence, it is a background image of a document that does not tile itself beyond the first screen, and stays static when the rest of the document scrolls up and down. Using watermarked backgrounds, you can have one big image that sits behind the background, and will not repeat itself to fit the entire document, creating a very unique and often beautiful look. However, many people are reluctant to use watermark backgrounds, since when Netscape users drops by, they will see one large image that is eager to reproduce itself-very ugly and unprofessional. Lets see an example

Before **(Please be patient as the huge background image downloads)**

Now, as you can see if you're using Netscape, the background tiles, which is really not that suitable, since its such a large "single" image. By using JavaScript, however, we can easily control that, by dynamically using different backgrounds, depending on the browser type. Before we do so, however, lets see the syntax of adding basic backgrounds, and watermarked backgrounds:

```
<body background="regular.gif">
```

```
//"regular" backgounds

<body background="watermark.gif" bgproperties="fixed">
//watermarked background. Works with Internet Explorer.
```

Ok, now that we know the syntax, we can easily use the "document.write" method to dynamically write out the background html code, depending on browser type:

After

```
<html>
<head><title>watermark improved</title></head>
<script>
document.write("<body background=")
if (navigator.appName=="Microsoft Internet Explorer")
document.write("'watermark.gif" bgproperties="fixed">')
else
document.write("'regular.gif">')
</script>
"
"
text here
"
"
</body>
</html>
```

Don't worry so much about how we tested for browser type, that's talked about in Lesson 10. Simply concentrate on how we dynamically wrote out the <body> tag itself, changing the way a document implements a background, depending on browser type.

No one is convincing you that you should now start to use watermarked backgrounds-I don't like to do so myself, since a watermarked background has to be almost as large as an entire screen to achieve its effects, which translates to long downloading time. The point here is that JavaScript adds a lot of versatility to your html document-the hell with static html tags, JavaScript's taking over!

-Tutorial introduction
-The date object
-Dynamically writing out html codes
-The setTimeout function
-Adding a live clock with the help of a form
-Example: To watermark or not to watermark?

.

**End of tutorial**

## Windows and JavaScript



Using JavaScript, we can manipulate windows-open them, close them, reload them, load another page into them-just name it.

-Tutorial introduction
-opening a new window using JavaScript
-Hiding/showing toolbars, menu bars, status bus etc upon opening a window
-reloading, closing, loading new content into a window using JavaScript
-Accessing objects of a window via another

### Opening a new window using JavaScript 

## Opening a new window using JavaScript



**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

With JavaScript, you can easily manipulate windows. One of the most asked questions when it comes to windows is how to open more than one window. Ok, then, lets first talk about that. To open a window, simply use the open() method of JavaScript. (window.open())

```
window.open("URL")
```

Lets say you just want to open a window containing "page2.htm"

Click here to see

```
<form>
<input type="button" value="Click here to see">
onclick="window.open('page2.htm')"
</form>
```

-Tutorial introduction
-opening a new window using JavaScript
-Hiding/showing toolbars, menu bars, status bus etc upon opening a window
-reloading, closing, loading new content into a window using JavaScript
-Accessing objects of a window via another

**Hiding/showing toolbars, menu bars, status bars etc upon opening a window**

## Hiding/showing toolbars, menu bars, status bus etc upon opening a window

Lets talk about some of the attributes we can add to the above script to control not only the size of the opened window, but also, what is to be shown: toolbar, menu bar etc. To add attributes when a window is opened, we still would use the window.open() method, but with a little addition:

```
open("URL","name","attributes")
```

As you can see, we first have to add in a name (this name has nothing to do with adding attributes-it is the window name used in the TARGET attribute of a <a> and <form> tag-don't worry about it, it is not commonly used). Moving on, here are the complete list of attributes you can add:

| | | |
|---|---|---|
| **width** | **height** | **toolbar** |
| **location** | **directories** | **status** |
| **scrollbars** | **resizable** | **menubar** |

Here is an example that opens a window that's not only smaller in size, but with ONLY the menubar turned on:

Click here to see

```
<form>
<input type="button" value="Click here to see">
onclick="window.open('page2.htm',
'win1','width=200,height=200,menubar')"
</form>
```

Here is another example with no attributes turned on, except the size changed:

Click here to see

```
<form>
<input type="button" value="Click here to see">
onclick="window.open('page2.htm',
'win1','width=200,height=200')"
</form>
```

Play around with them, if you like, but I'll end it here.

-Tutorial introduction
-opening a new window using JavaScript
-Hiding/showing toolbars, menu bars, status bus etc upon opening a window
-reloading, closing, loading new content into a window using JavaScript
-Accessing objects of a window via another

**Reloading, closeing, loading new content into a window using JavaScript** 

## Reloading, closing, loading new content into a window using JavaScript

With JavaScript, we can write code that replaces the "reload" and "close" button of our browser.

To reload a window, use this method:

```
window.location.reload()
```

Here's an example:

Click to reload!

To close a window, use this method:

```
window.close()
```

I was almost tempted to provide an example!

Ok, we can also load new content into a window using JavaScript-this is similar to the <a> tag in html, only now, using JavaScript, we have more control over this process.

The basic syntax when loading new content into a window is:

```
window.location="yoururl.com"
```

This is exactly the same as

```
<a href="yoururl.com></a> //if we use html
```

Lets provide an example, where a confirm box will allow users to choose between going to two places:

Click for example>>

```
<script>
<!--
function warp()
{
var ok=confirm('Click "OK" to go to geocities, "CANCEL" to go to CNN')
if (ok)
location="http://www.geocities.com"
```

```
else
location="http://www.cnn.com"
}
//-->
</script>
```

-Tutorial introduction
-opening a new window using JavaScript
-Hiding/showing toolbars, menu bars, status bus etc upon opening a window
-reloading, closing, loading new content into a window using JavaScript
-Accessing objects of a window via another

**Accessing objects of a window via another** 

## Accessing objects of a window via another



After you open a secondary window, there is a connection between the original window and this newly opened one that allows you to cross-access objects/variables, properties of each window. But before we discuss how this is done, we need to first discuss window names, a very different kind from the one mentioned in the beginning of this section.

Look at below:

```
var hello=
open('page2.htm','hi','width=200,height=200')
```

By giving this window a name using the above method, for example, "hello", it will give you access to anything that's inside this window from other windows. Whenever we want to access anything that's inside this newly opened window, for example, to write to this window, we would do this:
`hello.document.write("hi!")`
Ok, lets say you want to, from the current window, open a secondary window, and FROM HERE, change the background color of this newly opened window.

Click here first>>

**Now, click the below buttons:**

**Change background color: ...lightgreen: ...lightyellow: ...pink:**

The radio buttons are here, but we have changed the bgcolor of another window...lets see the core code:

```
//for button
onClick="win1=open('page2.htm','winname',
'width=200,height=200')"

//for radio button3
onClick="win1.document.bgColor=
'lightgreen';win1.focus()"
//for radio button2
onClick="win1.document.bgColor=
'lightyellow';win1.focus()"
//for radio button3
onClick="win1.document.bgColor=
'pink';win1.focus()"
```

The most important part is: `win1.document.bgColor`, which is what caused the bgcolor to change in the secondary window, instead of the one that contains the script. Also notice that we used another method, focus(), to bring focus to the second window every time it changes.

Here's the complete listing of the window object we touched upon in this tutorial:

**windows object**

| properties | methods |
|---|---|
| closed | alert |
| defaultStatus | blur |
| document | clearTimeOut |
| Frame | close |
| frames | confirm |
| history | focus |
| length | open |
| location | prompt |
| name | setTimeOut |
| opener | |
| parent | |

-Tutorial introduction
-opening a new window using JavaScript
-Hiding/showing toolbars, menu bars, status bus etc upon opening a window
-reloading, closing, loading new content into a window using JavaScript
-Accessing objects of a window via another

**End of tutorial**

## Loading two frames with one link





I must have received over 1000 mails asking me this question: "How do I load two frames with one link"? Well, here you are.



**How to access objects in one frame from another**

..

http://www.wsabstract.com

## How to access objects in one frame from another



Ok, the title may be a little ambiguous, but lets do an example to see what I mean by "cross-accessing" objects. Lets say we have a basic two-columned frame:


Further more, lets assume this is what's in page1.htm:

```
//page1.htm
<html>
<body>
<script>
function testing()
{
var x="this is page1!"
alert(x)
}
</script>
</body>
</html>
```

Ok, so we've created a little script in page1, that when run, will alert something. Normally, you can only call this function in page1, by doing something like this:

```
<script>
testing()
</script>
```

If you put the above code in page2, however, you will get an error message saying this function is not defined, and of course, that's true. Frames contain *separate* pages that are not connected in any other way other than the fact that they are stuck together by their master page (kinda like brothers and sisters). But wait..fortunately, JavaScript has provided a way to cross-access objects and variables from one frame via another...and the syntax is really easy:

parent.framename.(object/property)

That's all there is to it. Notice we have to first give each frame a name before proceeding (as mentioned in the previous section when adding specific types of links). For example, lets create the above master page, and add in some names so we can access page1's function from page2!

```
<html>
<frameset cols="30%,70%">
<frame src="page1.htm" name="foody">
<frame src="page2.htm" name="goody">
</frameset>
</html>
```

Now, to access page1's lame little function, this is what we will do in page2:

```
//page2.htm
<html>
<body>
<script>
parent.foody.testing()
</script>
</body>
</html>
```

Knowing this syntax, we can access any objects/properties/variables stored in one frame from another. The example above will cause the function to be run in page1, although we called it in page2.

-Tutorial introduction
-How to access objects in one frame from another
-How to use JavaScript to load multiple frames
-How frames can actually help JavaScript!

**How to use JavaScript to load multiple frame** 

..

## Home / JavaScript Tutorials / Loading two frames with one link

### Using JavaScript to load multiple frames



So what good is it to cross-access objects, you ask? Well, one of the most popular uses of such is to load and change the content of more than one frame at one. Without JavaScript, there isn't a prayer. Lets say we have the following example:

Click here for demo>>

```
<html>
<frameset cols="30%,70%">
<frame src="page1.htm" name="frame1">
<frame src="page2.htm" name="frame2">
</frameset>
</html>
```

We want to add a link in the first frame (page1.htm is in it) that will change the contents of not only page1, but page2 too. Ok, here we go: The below is the html file for page1:

Click here for demo>>

```
//page1.htm
<html>
<body>
<h1>This is page 1 </h1>
<a href="3.htm" onClick="parent.frame2.location='4.htm'">Click Here</a>
</body>
</html>
```

As you can see, this additional code will cause one link to perform two loading of pages.

Just for the hell of it, lets write a function that'll load 5 frames at once:

```
<script>
function muchoframes()
{
parent.frame1.location='6.htm
parent.frame2.location='7.htm
parent.frame3.location='8.htm
parent.frame4.location='9.htm
parent.frame5.location='10.htm
```

```
}
</script>
```

Nothing can stop you now!

-Tutorial introduction
-How to access objects in one frame from another
-How to use JavaScript to load multiple frames
-How frames can actually help JavaScript!

**How frames can actually help JavaScript** 

..

## Home / JavaScript Tutorials / Loading two frames with one link



**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

### Frames can actually help JavaScript too!

With JavaScript helping others all the time, its nice to know that someone gives it a hand once in a while too. Frames can be very helpful to JavaScript when creating JavaScript driven applications, and the reason requires a little explanation.

By default, all variables and functions created in JavaScript in one page is lost when you leave that particular page. For example, if I make a declaration in this page:
var x=500, this x only exists in this page-as soon as you leave, x is erased, gone. This is great, in most cases, since if the browser retains every variable, not only will it overload your memory, but also, if two pages use the same variable, there will be a small riot starting in your browser. Having said that, in special cases, what if you *want* to retain that variable temporarily, even if a new page is loaded? Well, the key is by using frames. You can store all variables you want to retain for further processing in one frame that is not being changed,while the other frames can change all they want. Since that particular frame is not being loaded with new content, the variable is retained, until you leave that whole document, of course.

-Tutorial introduction
-How to access objects in one frame from another
-How to use JavaScript to load multiple frames
-How frames can actually help JavaScript!

..

**End of tutorial**

# Understanding the various types of links



In this tutorial, we'll revisit the part of HTML responsible for spinning the web into what it is today: links. Links are among the very first things all webmasters learn, but many did not know that there is life after the href and mailto link.

A standard looking link in a document looks something like this:

<a href="page1.htm">Click here</a>

Clicking on the above will take us to "page1.htm". While most authors are content with using a link to simply link to a page or as a mailto link, its important to realize that it has other functions too. The below lists these functions:

| Types of links | Functions |
|---|---|
| **http:** | **Takes us to another html document** |
| **mailto:** | **Opens up the default email program** |
| **javascript:** | **Executes a JavaScript code** |
| **view-source:** | **Views the source of the specified document** |
| **about:** | **Views information about the browser, such as cache, version etc.** |

Lets explain in detail the later three:

-Tutorial introduction
-JavaScript:
-view-source:
-about:

**JavaScript:** 

# Home / JavaScript Tutorials / Understanding the various types of links

## JavaScript:



**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

A link can execute a JavaScript command upon clicking (instead of loading a page). Simply use the keyword **JavaScript:,** along with the code to execute:

Click to reload!

```
<a href="JavaScript:window.location.reload()">Click to reload!</a>
```

-Tutorial introduction
-JavaScript:
-view-source:
-about:

**view-source:**

# Home / JavaScript Tutorials / Understanding the various types of links

## view-source:

**WA Help Forum**
Having trouble with anything? Visit our help forum to get the answers you need.

**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

By default, whenever we want to view the source of a document, we would reach for "view" of the toolbar of our browser. This "view" function can actually be simulated using a special kind of link- the view-source link. The standard syntax required is:

<a href="view-source:complete file path">Click to view the source!</a>

The path has to be complete, not simply the name of the file. Here are a couple of examples:

```
<a href=
"view-source:file:///C|/web/wa/index.htm">
Click to view the source!</a>
```

```
<a href=
"view-source:http://www.wsabstract.com/index.htm">
Click to view the source!</a>
```

Here's where JavaScript comes in. Due to the fact that a html document can be viewed both online and offline (downloaded onto a harddrive), hard coding the complete path of a file to view source will produce a bad link in one of the two cases above. You may have entered http://www.wsabstract.com/javatutors/links.htm in the view-source command while the surfer is viewing the page offline (file:///C|/web/javatutors/links.htm). To solve this little problem, simply use the window.location property of JavaScript to dynamically determine the current path of the html document:

```
<script>
function viewthesource(){
window.location="view-source:"+window.location
}
</script>
<a href="javascript:viewthesource()">
Click to view source!</a>
```

Click to view source!



**about:**

## about:



"about:" allows a link, when clicked on, to directly display information we would normally use features of the toolbar to display, such as cache or plugin information. **The "about" command does NOT work with Internet Explorer browsers.** Lets see the available information displayable using the **about:** command:

| Syntax | Functions |
|---|---|
| **about:** | **Simulates "About Netscape" in the "Help" menu of Netscape** |
| **about:cache** | **Displays disk cache information (this feature is disabled if your browser does not permit the viewing of cache).** |
| **about:plugins** | **Simulates "Plug-ins" in the "Help" menu of Netscape** |

Putting it all together:

Click to view browser information

Click to view cache information

Click to view plugins information

```
<a href="about:">Click to view browser information</a>

<a href="about:cache">Click to view cache information</a>

<a href="about:plugins">Click to view plugins information</a>
```

Now that you're armed with more link knowledge, go out and fortify your position on the WWW!

-Tutorial introduction
-JavaScript:
-view-source:
-about:

.

**End of tutorial**

## The language attribute of the JavaScript



In the tutorial JavaScript primer, I briefly mentioned the language attribute of JavaScript, and how it is used more or less as simply a way of telling others that the containing code is written in JavaScript. Well, back then, that is the case, but as JavaScript evolved with the advent of next generation browsers, the language attribute has gained significant importance in the JavaScript language, and may quickly crawl itself out of the status as simply being an "optional" declaration. In this tutorial, I'll discus both how to use the language attribute, and its merits to your code.



**Using the language attribute of JavaScript**

# Home / General JavaScript Tutorials / The language attribute of JavaScript

## Using the language attribute of JavaScript



The basic form of the language attribute is very easy to implement. Simply shove in the keyword "language=", followed by "JavaScript", into your <script> tag:

```
<script language="JavaScript">
alert("hi!")
</script>
```

The above declaration does more than simply tell you and I that the containing code is written in JavaScript- *it tells the browser as well.* This is very important to realize, although up until now, it was also optional. Due to the dominance of JavaScript over other languages from the start, all browsers since NS 2.x default to the language "JavaScript" when no language attribute is explicitly defined. Therefore, your browser treats:

```
<script>
alert("hi!")
</script>
```

exactly the same as the first code- as a JavaScript code. Having said that, you may begin to ask: "Could there be that there exist other languages other than JavaScript for my web browser?" Your suspicion is correct. More on this later.



**Variations of the language attribute**

## Variations of the language attribute



The variations of the language attribute that have emerged as NS 3, and now NS 4 and IE 4, have shipped is what makes this attribute more significant when writing scripts. Apart from the standard "JavaScript" value, there are now three additional choices- "JavaScript1.1", and "JavaScript1.2", and "JavaScript1.3. For example:

```
<script language="JavaScript1.1">
alert("hi!")
</script>
```

Lets now explain what these values are, and do:

**Values of the language attribute**

| | |
|---|---|
| **JavaScript** | Specifies that the containing code is written in JavaScript. |
| **JavaScript1.1** | Specifies the containing code is written in version 1.1 of JavaScript. Browsers lower than NS 3 and IE 4 will ignore the code (ie: NS 2, IE 3) |
| **JavaScript1.2** | Specifies the containing code is written in version 1.2 of JavaScript. Browsers lower than NS 4 and IE 4 will ignore the code (ie: NS 3) |
| **JavaScript1.3** | Specifies the containing code is written in JavaScript 1.3, intended for NS 4.5 and up. All other browsers will ignore the code. |

Now you know that the language attribute is more than just simple decoration for your <script> tag- it actually affects how the browser will treat the containing code.

Moving on, I'll demonstrate just how to use the language attribute to your advantage when writing scripts.

-Tutorial introduction
-Using the language attribute of JavaScript
-Variations of the language attribute
-Using the language attribute in the real world
-Other Scripting languages on the web

**Using the language attribute in the real world**

## Using the language attribute in the real world

For your convenience, lets first list again the language attribute table:

**Values of the language attribute**

| | |
|---|---|
| **JavaScript** | Specifies that the containing code is written in JavaScript. |
| **JavaScript1.1** | Specifies the containing code is written in version 1.1 of JavaScript. Browsers lower than NS 3 and IE 4 will ignore the code (ie: NS 2, IE 3) |
| **JavaScript1.2** | Specifies the containing code is written in the latest version of JavaScript, version 1.2. Browsers lower than NS 4 and IE 4 will ignore the code (ie: NS 3) |
| **JavaScript1.3** | Specifies the containing code is written in JavaScript 1.3, intended for NS 4.5 and up. All other browsers will ignore the code. |

Using the language attribute, we can in essence write "conditional" JavaScript without using a single "if" statement. If the code in question is to be executed only by NS 3 and up, we use the "JavaScript1.1" value inside the <script> tag. If its a script that utilizes the latest features of JavaScript1.2, then that value should be used instead.

The below is a simple script that preloads two images. The basic code looks like below:

```
<script>
<!--
image01= new Image(42,42)
image01.src="1.gif"
image02= new Image(42,42)
image02.src="3.gif"
//-->
</script>
```

Is there anything wrong with the above code? Well, that depends who you talk to. Talk to someone who's using NS 3, and she'll say its all dandy. Talk to an IE 3 user, however, and she'll complain of JavaScript errors when the page is loading. This is due to the fact that IE 3 does NOT support the image object, and preloading images involves manipulating the image object. The solution? Very simple. Add in a language attribute of JavaScript1.1 to the above code, and everyone's happy:

```
<script language="JavaScript1.1">
<!--
```

**WA Help Forum**
Having trouble with anything? Visit our help forum to get the answers you need.

Want to print this page? Use this version instead.

**Jump to...**
-Tutorial Index
-Free JavaScripts
-Free Java Applets
-Web Tutorials
-Frontpage Tutorials



**Link to us!**
We always welcome and appreciate links back to Website Abstraction. Click here for details.

```
image01= new Image(42,42)
image01.src="1.gif"
image02= new Image(42,42)
image02.src="3.gif"
//-->
</script>
```

By doing the above, IE 3 ignores the code completely (its not even loaded into cache).

The "JavaScript1.2" value operates in a similar manner as "JavaScript1.1". It tells all browsers other than NS 4+ and IE 4+ to ignore the containing code. This value is becoming increasingly useful as more and more JavaScript1.2 scripts are appearing. The below alerts the dimensions of the surfer's screen, which is only supported in NS 4+ and IE 4+: The JavaScript1.2 value is used inside the <script> tag:

```
<script language="JavaScript1.2">
alert(screen.width+" "+screen.height)
</script>
```

The "JavaScript1.3" value allows programmers to write code intended specifically for NS 4.5 and up. NS 4.5 supports a small set of new features not found in JavaScript 1.2, such as unicode support, and improvements to many existing objects. For a summary of these features, **click here**.

The language attribute not only allows us to easily create backwards compatible scripts, but efficient ones as well. Browsers that do not support the script version indicated by the language attribute do not even load the script into cache, saving your surfers a few bytes of cache space.

## Other scripting languages on the web

JavaScript has become so dominant over its short life span that many authors believe its the only scripting language on the web. Well, not quite. There exist at least two other fairly popular ones, namely, JScript, and Vbscript. Both of these languages are supported only by IE, and the easiest way to make sure that only IE browsers interpret code written in these two languages is by using the language attribute. The below demonstrates this:

```
<script language="VBscript">
msgBox("This text is only alerted for IE users!")
</script>
```

Without the language attribute, NS will choke on the above line of code.

**End of Tutorial**

## Home / JavaScript Tutorials / Image And JavaScript

# Images and JavaScript- Apples and Apples



In order to create rollover effects, we have to first understand how to access images in general using JavaScript. All images in a web page are treated as an "image object" to JavaScript. The image object is created for you whenever you insert images onto your webpage. It does so automatically, even if there's not one line of JavaScript code on your page, as opposed to the Date object, for example. In this tutorial, we will look at everything you'll need to create your own roll-over effects!



**Accessing images using JavaScript** 

## Accessing images using JavaScript

The first order of business is to learn how to access images in general using JavaScript. Assuming you have three images on your webpage, as follows:

**Image 1:** **Image 2:** **Image 3:**



If you want to refer to the three images using JavaScript, here's how:

```
document.images[0] //first image
document.images[1] //second image
document.images[2] //third image
```

As can be seen, images to JavaScript are an array of images, so you use "[ ]" to access each image, "[0]" being the first one. If you are not familiar with arrays, they are discussed here.. There's an alternative to using integers to access the images, and that is using the name of each image. (Uh? what name?) Well, the name is something you can choose if you want to add to an image, which makes referring to them a lot easier. The name is inserted in the html tag. Lets use the first image as an example:

```
<html>
<body>
<img name="myimage01" src="whatever.gif" width="100" height="50">
</html>
```

Now, to access this image, this is what you'd do instead:

```
document.images["myimage01"] //first image
```

Once you can access images, you can create animations or mouse over/out, even mouse down/up effects. For example, this is a typical example of a mouseover/mouseout effect (works with Netscape 3.x and 4.x) We will later see how we can create this effect for IE4.x browsers too.

This can be done with the onmouseover, onmouseout event handlers-along with the help of *preloading images*. So lets first talk about images being preloaded.

-Preloading images
-onMouseover/onMouseout effects script
-Presenting the final version of the onMouseover effects with the filter added in
-Presenting an improved filter that is friendly to IE4.0

**Preloading images** 

**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

## Preloading images

What is it? It is basically loading an image into cache before being used, so whenever we want to use it, bam! it appears instantaneously. Usually, when we load a page, all image are not preloaded-that's why we see these image becoming visible bit by bit. Why would we want to preload an image? Because it is the solution to avoiding having delays in between switches of images when using the onmouseover/onmousedown effect-among other things. How do we preload an image? **By creating an instance of the image object, and inserting it in the <head></head>tags.** Ok, don't worry, I'll explain it below: Since the object contains simply a few properties/methods, I'll list them ALL here:

**The Image object**

| properties | methods |
|---|---|
| src | onLoad |
| height,width | onError |
| lowsrc | onAbort |
| border | |
| vspace,hspace | |

Ok, lets preload two images to be used for the onmouseover/onmouseout effect. These are the two images we shall preload:


**"1.gif"**

**"3.gif"**

```
<head>
<script>
<!--
image01= new Image(42,42)
image01.src="1.gif"
image02= new Image(42,42)
image02.src="3.gif"
//-->
</script>
</head>
```

- Notice that the preloading of the image takes place within in <head></head> section of your webpage.

- Recall what you've learned-we are creating an instance of the image object (we did something similar for the date object). By using the keyword **"new"**, we have done that. The image object happens to accept two parameters-the width and the height of the image you are putting into the object. That, in this case, happens to be 42 by 42. Next we need to tell it what exactly to contain in it, by using the method src and pointing that to the actual path of the image.

You may be thinking to yourself :"What is this? This doesn't make any sense-why are we doing all this?" The answer is quite simple. In html, there simply is no way you can load an image without showing it. All image tags go within the body tag, which immediately displays it. But in JavaScript, you can-you can first load the image and defer until the right time to wam boom bala boom. So, that's why we are going through all this mess to do something like this. Don't worry, this is about all you need to know about the image object; the other properties are virtually not used at all.

Now that we've preloaded the images, and, recalling the way we accessed images (on the very top) using JavaScript, we can get down to business. The basic idea is to have JavaScript alter the path of the image depending on whether its onmouseover or onmouseout. Change the path, change the image!

-Tutorial introduction
-Accessing images using JavaScript
-Preloading images
-onMouseover/onMouseout effects script
-Presenting the final version of the onMouseover effects with the filter added in
-Presenting an improved filter that is friendly to IE4.0

**onMouseover/ onMouseout effect script** 

## Home / JavaScript Tutorials / Image And JavaScript

### onMouseover/onMouseout effects script



```
<html>
<head>
<script language="JavaScript">
<!--
image01= new Image(42,42)
image01.src="1.gif"
image02= new Image(42,42)
image02.src="3.gif"
//-->
</script>
</head>

<body>
<a href="whatever.htm" onmouseover=
"document.images['example'].src=image02.src" onmouseout=
"document.images['example'].src=image01.src">
<img src="1.gif" name="example"></a>
</body>

</html>
```

- One very important point: the "onmouseover, onmouseout handler is placed inside the link tag, not the image tag, although we are changing the image. Why? First of all, the image object does not support the onmouseover/onmouseout handler! Placing it in there will do nothing. Instead, we place it in the link tag, which will do the trick, because the link tag and the image tag are "connected."
- Note in both the onmouseover, onmouseout reference, we refer to the same image name-"example." We want it to load whatever image the two image objects contain OVER the existing one, not on Mars or Pluto!

The above will work and produce the effect...there is one problem though: It works only with NS3.x and 4.x. It will not work with any versions of IE, and the problem is more serious than that. It will, in fact, produce error messages for IE users, either upon loading or after putting their mouse over the image. What we need here is a "filter" code that will filter out browsers that don't support the onmouseover code. So why didn't I simply implement a filter above? Because I wanted to show you exactly how the process of swapping images occurs. The concept is very important, and if I simply gave you the final version with the filter added, you may not see this. Anyhow, moving on, we will now try to implement a filter that will take care of the problem.

-Presenting the final version of the onMouseover effects with the filter added in
-Presenting an improved filter that is friendly to IE4.0

**Presentin the final version of the onMouseover effects with the filter added in** 

## Home / JavaScript Tutorials / Image And JavaScript



### Presenting the final version of the onMouseover effects with the filter added in.

In order to add a filter to distinguish between the different versions of Netscape/IE, we need to look at a prebuilt object that helps us do that:

**Navigator Object-Some properties of it.**

| | |
|---|---|
| **appName** | This will provide you with the browser name of your browser: "Netscape" or "Microsoft Internet Explorer", for example. |
| **appversion** | This will give you the version number of a browser. To make it harder on you, it doesnt return only the version number, but also the platform and country (U/I). ie: "3.01 (Win95;I)." |

```
<html>
<head>
<script>
<!--
image01= new Image(42,42)
image01.src="1.gif"
image02= new Image(42,42)
image02.src="3.gif"
//-->
</script>

<script language="javascript">
<!--
function filter(imagename,objectsrc)
{
if ((navigator.appName=="Netscape")&&
(parseInt(navigator.appVersion)>=3))
document.images[imagename].src=eval(objectsrc+".src")
}
//-->
</script>
</head>

<body>
<a href="whatever.htm"
onmouseover="filter('example','image02')"
onmouseout="filter('example','image01')">
<img src="1.gif" name="example"></a>
</body>
</html>
```

Ok, by adding the filter, we have successfully prevented MS IE users from receiving error messages when surfing to your page. Lets discuss the details here:

- Lets take a look at "function filter(imagename,objectsrc)" This is the actual filter that will help filter out those IE browsers that will generate errors. Notice that this function will accept two parameters-the name of the image that is inside the <img> tag, and the src of the image object that will be used to replace that one. We tested not only for the browser type, but the browser version. (parseInt(navigator.appVersion)>=3))basically says: if the version is greater than "3". Look at "parseInt(.....)" This is a function that will attempt to extract a number from a string. For example, parseInt(23jhhkku) will give back "23" We need this to extract the "number" part of the version of your browser.
- Now the following line is what will actually assign a new path to the image, changing the image if it passes the browser test:
  document.images[imagename].src=
  eval(objectsrc+".src")
  Notice that when assigning the new image path to the image, eval(objectsrc+".src")was used instead of simply objectsrc+".src" We need to use this function, because if we didn't, it will not work. Let's see why. Lets say we have something like this:

```
x="2+6+9"
alert(x)
```

Click here for output>>

as opposed to this:

```
x=eval("2+6+9")
alert(x)
```

Click here for output>>

See the difference? Eval treats a string as if it were a statement, and we need this so it treats the path of the image: objectsrc+".src" as a path, not simply a bunch of letters strung together.

After defining this function, we could use it over and over again with different images, by simply passing in a different image name and object path. This is where we should begin to realize that a little careful thought in programming can save us a lot of time afterwards. If we were to use the original coding for many images, even, assuming it is compatible with IE browsers, we would still be doing a lot more work than using a function, like above.

-Tutorial introduction
-Accessing images using JavaScript
-Preloading images
-onMouseover/onMouseout effects script
-Presenting the final version of the onMouseover effects with the filter added in

-Presenting an improved filter that is friendly to IE4.0

**Presenting an improved filter that is friendly to IE 4.0** 

## Home / JavaScript Tutorials / Image And JavaScript



### Presenting the final version of the onMouseover effects with the filter added in.

In order to add a filter to distinguish between the different versions of Netscape/IE, we need to look at a prebuilt object that helps us do that:

**Navigator Object-Some properties of it.**

| | |
|---|---|
| **appName** | This will provide you with the browser name of your browser: "Netscape" or "Microsoft Internet Explorer", for example. |
| **appversion** | This will give you the version number of a browser. To make it harder on you, it doesnt return only the version number, but also the platform and country (U/I). ie: "3.01 (Win95;I)." |

```
<html>
<head>
<script>
<!--
image01= new Image(42,42)
image01.src="1.gif"
image02= new Image(42,42)
image02.src="3.gif"
//-->
</script>

<script language="javascript">
<!--
function filter(imagename,objectsrc)
{
if ((navigator.appName=="Netscape")&&
(parseInt(navigator.appVersion)>=3))
document.images[imagename].src=eval(objectsrc+".src")
}
//-->
</script>
</head>

<body>
<a href="whatever.htm"
onmouseover="filter('example','image02')"
onmouseout="filter('example','image01')">
<img src="1.gif" name="example"></a>
</body>
</html>
```

Ok, by adding the filter, we have successfully prevented MS IE users from receiving error messages when surfing to your page. Lets discuss the details here:

- Lets take a look at "function filter(imagename,objectsrc)" This is the actual filter that will help filter out those IE browsers that will generate errors. Notice that this function will accept two parameters-the name of the image that is inside the <img> tag, and the src of the image object that will be used to replace that one. We tested not only for the browser type, but the browser version. (parseInt(navigator.appVersion)>=3))basically says: if the version is greater than "3". Look at "parseInt(.....)" This is a function that will attempt to extract a number from a string. For example, parseInt(23jhhkku) will give back "23" We need this to extract the "number" part of the version of your browser.
- Now the following line is what will actually assign a new path to the image, changing the image if it passes the browser test:
  document.images[imagename].src=
  eval(objectsrc+".src")
  Notice that when assigning the new image path to the image, eval(objectsrc+".src")was used instead of simply objectsrc+".src" We need to use this function, because if we didn't, it will not work. Let's see why. Lets say we have something like this:

```
x="2+6+9"
alert(x)
```

Click here for output>>

as opposed to this:

```
x=eval("2+6+9")
alert(x)
```

Click here for output>>

See the difference? Eval treats a string as if it were a statement, and we need this so it treats the path of the image: objectsrc+".src" as a path, not simply a bunch of letters strung together.

After defining this function, we could use it over and over again with different images, by simply passing in a different image name and object path. This is where we should begin to realize that a little careful thought in programming can save us a lot of time afterwards. If we were to use the original coding for many images, even, assuming it is compatible with IE browsers, we would still be doing a lot more work than using a function, like above.

**image object**

| properties | methods |
|---|---|

| | |
|---|---|
| **border** | **onAbort** |
| **complete** | **onError** |
| **height** | **onLoad** |
| **hspace** | |
| **lowsrc** | |
| **name** | |
| **prototype** | |
| **src** | |
| **vspace** | |
| **width** | |

And here's the complete navigator object:

**Navigator object**

| properties | methods |
|---|---|
| **appCodeName** | **no methods** |
| **appName** | |
| **appVersion** | |
| **mimeTypes** | |
| **plugins** | |
| **userAgent** | |

Be sure to check out the onmouseover whipper, which automates the process of creating rollover effects for you!

-Tutorial introduction
-Accessing images using JavaScript
-Preloading images
-onMouseover/onMouseout effects script
-Presenting the final version of the onMouseover effects with the filter added in
-Presenting an improved filter that is friendly to IE4.0

.

**End of tutorial**

## Home / Web Building Tutorials / Creating an image slide show

# Creating an image slide show (Two parts)

**WA Help Forum**
Having trouble with anything? Visit our help forum to get the answers you need.

**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

Apart from roll-over effects, this must be one of the most common questions I get asked regarding images: "How do I create an image slide show?" Well, there are many ways, the simplest and most efficient using JavaScript. In this tutorial, we will discuss all the necessary information you'll need to put together a cool image slide show using JavaScript. Note that NS 2.x and IE 3.x does NOT support the image object required to power the slideshow, so these browsers will see no sliding images.

Notice that the above slide show has the added functionality of changing URLs as the image changes. This tutorial consists of two parts; the first part will discuss how to implement a slide show in general, and the second will build on the first to discuss how to change the URLs as the image "slides"

-Tutorial introduction
-How to create a basic slide show
-How to create a clickable slide show

**Go on to part 2**

## How to create a basic slide show

### Step 1: Get some images!

The first step, needless to say, is to first fetch the images you want to include in the slideshow. For this tutorial, we'll be using these three images:


**"firstcar.gif"**


**"secondcar.gif"**


**"thirdcar.gif"**

Imagine yourself as a car salesmen, and these three cars are the ones you are selling!

### Step 2: Preload the images using JavaScript.

The term "preload" in JavaScript refers to the loading of images into cache prior to displaying them. Preloading images is "necessary" in a slide show, since the switching between images have to be instantaneous, without any delay:

```
<head>
<script language="JavaScript1.1">
<!--
var image1=new Image()
image1.src="firstcar.gif"
var image2=new Image()
image2.src="secondcar"
var image3=new Image()
image3.src="thirdcar.gif"
//-->
</script>
</head>
```

We've created three instances of the image object, each referring to one of the images that make up the slide show. By doing so, the images are loaded into cache, standing by for us to display at anytime. Notice that the entire code is inserted in the <head> section of the page (Detailed discussion of preloading images can be found here).

### Step 3: Add in the html codes necessary to display the first image of the slide show.

```
<body>
<img src="firstcar.gif" name="slide" width=100 height=56>
</body>
```

All we've done above is insert the first image of the slide show using HTML. Notice how we gave the image a "name" attribute. By naming the image with an arbitrary name, it enables JavaScript to access and manipulate the image, which we will see next.

Step 4: With everything in place, all we need now is a script that accesses the above image and changes the src of the image periodically, creating a slide show. The below lists the complete script:

```
<script>
<!--
//variable that will increment through the images
var step=1
function slideit(){
//if browser does not support the image object, exit.
if (!document.images)
return
document.images.slide.src=eval("image"+step+".src")
if (step<3)
step++
else
step=1
//call function "slideit()" every 2.5 seconds
setTimeout("slideit()",2500)
}
slideit()
//-->
</script>
```

The core of this script is the part in red:

```
document.images.slide.src=eval("image"+step+".src")
```

The left handside accesses the path of image "slide" using the image object, then the name of the image, then the keyword "src". The path of any image in a html document can be accessed this way. The right handside dynamically assigns a new src (path) to the image, thus changing the image. The three possible paths the image may receive are:

```
image1.src  //"firstcar.gif"
image2.src  //"secondcar.gif"
image3.src  //"thirdcar.gif"
```

in that order.

Step 5: Putting it all together.

We've preloaded the images, inserted the first image of the slideshow, and created the function necessary to change the path associated with the image. All we have to do now is put it all together into one page, and we've got ourselves a slideshow:

```
<html>
<head>
<script language="JavaScript1.1">
<!--
```

```
var image1=new Image()
image1.src="firstcar.gif"
var image2=new Image()
image2.src="secondcar"
var image3=new Image()
image3.src="thirdcar.gif"
//-->
</script>
</head>
<body>
<img src="firstcar.gif" name="slide" width=100 height=56>
<script>
<!--
//variable that will increment through the images
var step=1
function slideit(){
//if browser does not support the image object, exit.
if (!document.images)
return
document.images.slide.src=eval("image"+step+".src")
if (step<3)
step++
else
step=1
//call function "slideit()" every 2.5 seconds
setTimeout("slideit()",2500)
}
slideit()
//-->
</script>
</body>
</html>
```

-Tutorial introduction
-How to create a basic slide show
-How to create a clickable slide show

**How to create a clickable slide show** 

**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

## How to create a clickable slide show

The slideshow we've just created is almost identical to the one in the beginning of this tutorial, except that its not clickable. We will now see how to extend this example to enable it to not only be "hyperlinked", but hyperlinked with a different url, depending on the image displayed.

A slideshow that's not clickable is essentially the same as a fancy animated gif. A slideshow that is, however, becomes an interactive billboard. In this section, we will enhance the old slideshow to make it into just that!

Don't panic- the road to interactivity does not require us to alter the original code in anyway. All that's needed is the addition of a <a> tag surrounding the <img> tag, and a function that manipulates this <a> tag to match different links to each image in the slideshow.

Step 1: Surround the existing <img> tag with a <a> tag. Use a JavaScript url in place of a standard url to allow programmatic access to it:

```
<a href="javascript:slidelink()"><img src="firstcar.gif" name="slide" width=100 height=56></a>
```

Notice the code

```
javascript:slidelink()
```

The above is called a JavaScript url, and when clicked on, will call and execute a JavaScript function, instead of load a document. By throwing out the standard link and replacing it with a JavaScript url, a url turns dynamic. Now, there really is nothing mysterious about a JavaScript url- it simply executes a JavaScript function in place of loading a html document. In the above case, the function that will be executed is slidelink(), which we will actually implement in our next step

Step 2: Declare and implement function slidelink() inside the original slideshow script.

The "meat" of this tutorial, slidelink() is the function that will dynamically change the url of the slideshow to match the image that's currently being displayed in the slideshow. The below lists the original slideshow script, with new additions highlighted in red:

```
<script>
<!--
var step=1
//a variable that will keep track of the image currently being displayed.
```

```
var whichimage=1
function slideit(){
if (!document.images)
return
document.images.slide.src=eval("image"+step+".src")
whichimage=step
if (step<3)
step++
else
step=1
setTimeout("slideit()",1800)
}
slideit()
function slidelink(){
if (whichimage==1)
window.location="link1.htm"
else if (whichimage==2)
window.location="link2.htm"
else if (whichimage==3)
window.location="link3.htm"
}
//-->
</script>
```

We declared a new variable called "whichimage", which will be used to keep track of the image currently in display. In other words, variable "whichimage" keeps track of whether the image currently in display is the first, second, or third image in the slideshow. How does this variable achieve that? By retrieving the number stored inside variable "step" just before it gets incremented during each image rotation. Once we are able to keep track of this information, we can write code that loads a different url, depending on the image displayed. This is realized through function slidelink().

## Step 3: Throw everything together.

All that's left now is to toss everything into one bowl, and we have an interactive billboard that takes us to a different url depending on the image shown:

```
<html>
<head>
<script language="JavaScript1.1">
<!--
//preload images
var image1=new Image()
image1.src="firstcar.gif"
var image2=new Image()
image2.src="secondcar"
var image3=new Image()
image3.src="thirdcar.gif"
//-->
</script>
</head>
<body>
<a href="javascript:slidelink()"><img src="firstcar.gif" name="slide" border=0
width=100 height=56></a>
```

```
<script>
<!--
var step=1
var whichimage=1
function slideit(){
if (!document.images)
return
document.images.slide.src=eval("image"+step+".src")
whichimage=step
if (step<3)
step++
else
step=1
setTimeout("slideit()",1800)
}
slideit()
function slidelink(){
if (whichimage==1)
window.location="link1.htm"
else if (whichimage==2)
window.location="link2.htm"
else if (whichimage==3)
window.location="link3.htm"
}
//-->
</script>
</body>
</html>
```

In this tutorial, we've created a slideshow that cycles through three images; it could **easily** be modified to accommodate any number of images.

-Tutorial introduction
-How to create a basic slide show
-How to create a clickable slide show

/

**End of Tutorial**

# Accessing and validating forms using Javascript



**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

Before JavaScript was invented, validating forms means diving into a thick CGI programming book and spending hours wasting away an otherwise perfectly good life. With the introduction of JavaScript, it has gotten a LOT simpler. In fact, by the time you're done reading this tutorial, you'll be able to validate forms.



**The object model of JavaScript** 

## Home / JavaScript Tutorials / Accessing And Validating Forms

### The object model of JavaScript

**WA Help Forum**
Having trouble with anything? Visit our help forum to get the answers you need.

**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

Before we start diving in head first, lets just a close look at the object concept of JavaScript, which will all tie in with accessing forms.

JavaScript is a language of objects, and objects are something you should not be afraid of. A simple example of an object is the document object, which allows us to access various aspects of a document, such as background color, referrer etc. JavaScript contains many objects, each representing one component of the web page.

Think of an object as a "tool" that contains properties and methods, allowing it to do things. Lets use a typical example. In real life, a car would be the object, the leather seats, power steering would be the properties, and the car's ability to move around is the object's method. In JavaScript, its just like that. However, properties of one object may itself be another object, which in turn contains other properties. This is important, and I will talk a little more about this. The below is the JavaScript Object tree, which lists, in order, each object, and also, the precedence of the objects:

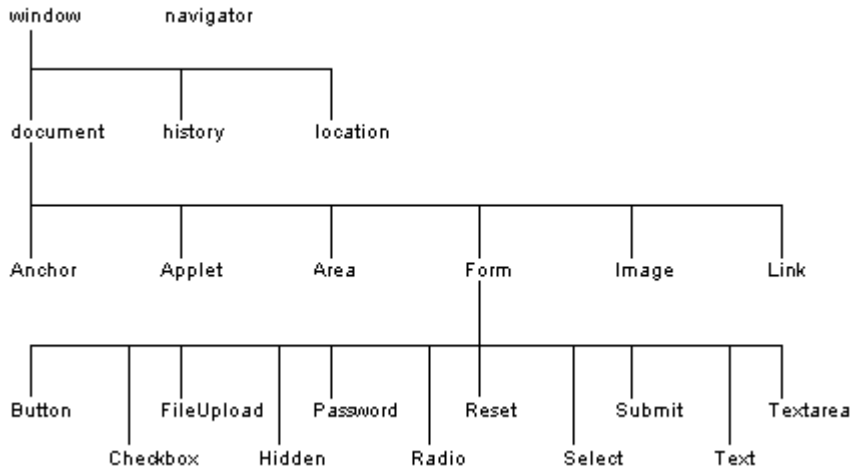


**The JavaScript object tree** ▶

## Home / JavaScript Tutorials / Accessing And Validating Forms



### The JavaScript Object Tree




**IMPORTANT: THIS IMAGE WAS TAKEN FROM THE NETSCAPE JAVASCRIPT AUTHORING GUIDE. FULL CREDIT FOR THE ABOVE IMAGE GOES THERE.**

If you want to access something further down below the tree from the object "window", you use the (dot) operator to specify your choice. For example, if you want to write something to the current document, you would do something like the following: window.document.write("Hola") However, its important to understand that the "window" part is omitted for most references. The reason why you could write document.write("Hola")instead of window.document.write("Hola")is because JavaScript assumes you are talking about the current window. Lets have a look at how form objects are accessed: =



**Accessing forms using JavaScript**




## Accessing forms using JavaScript



To access a form within a page, do this:
document.formname.elementname.value

For example, say you have a form like this:

```
<form name="test">
<input type="text" size="10" value="good" name="boxes">
</form>
```

good

**Notice that we gave the form and the element a name. This is so JavaScript can use this information to gain access to them .** Ok, now we want JavaScript to alert us the value inside this box.

```
<input type="button" value="Click Here>>"
onclick="alert(document.test.boxes.value)">
```

Click here for output: Click here>>

Note how we got to the value of the element of the form object-by going down the JavaScript object tree. Try altering the text in the text box, and try clicking above again. You will see that it will alert the new entry each time.

Ok, so whenever you create a form, always remember to give both the form and the element a distinct name in which JavaScript can then acquire to access the element. Is this the only way? No. You could use arrays, but I will not talk about that in this section; this is generally the better way to access forms in that it will not confuse you by asking you to keep track of each form using its index number.

Lets do an illustration with multiple boxes.

**Basic Calculator-Enter a math expression in the first box, and than use the calculate button to get the answer. Ex: 25*(6-5)**

Calculate Reset

**Answer:**

```
<form name="example">
<input type="text" size="20" name="calculator">

<input type="button" name="B1" value="Calculate" onclick="cal()">

<input type="reset" name="B2" value="Reset">

Answer:<input type="text" size="20" name="answer">
</form>

<script>
function cal()
{
document.example.answer.value=
eval(document.example.calculator.value)
}
</script>
```

The key here is the code within the script tags:

```
document.example.answer.value=eval(document.example.calculator.value)
```

Basically, we are saying: put whatever is in the right-hand-side, and stuff it into the left hand side. Hay, but wait! What is eval(.....) in the above? It is a built in function that "evaluates" what's in the first box before stuffing it into the second one. eval(*something)* is a built in function that "makes sense" of any thing you put in the function and than outputs it. If we simply do this, without using the eval function:

```
document.example.answer.value=document.example.calculator.value
```

If we type: (3*5)+7
You would get: (3*5)+7
As opposed to: 22

Don't expect to understand fully the eval function right now...just understand its role in this example. We will come back to this function, because its such an important one; in the mean time, don't worry about it. Now, there is one small problem with this example: Try inputting letters (ie: l*we) into it, and you will get an error message. That's because, of course, you cannot calculate letters, but your code does not know that! It attempts to, and out pops JavaScript errors-we shall discuss validating forms in the next page.

Finally, lets do a more complex example that has a little more practicality to it. You may have seen the below example used on the net:

**Move your mouse over the links:**

**Web Resources**
.
**Web Heaven**
.
.

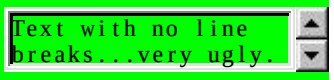
Here's the "core" code for the first link, assuming we have given the form the name "myform", and the textarea the name "mytextarea":

```
<a href="resources.htm"
onMouseover="document.myform.mytextarea.value=
'blablabla...'
onMouseout="document.myform.mytextarea.value="">Web Resources</a>
```

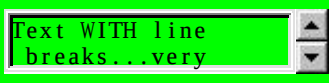
As you can see, we used the event handers we had learned before to give the textarea box different text, depending on the state of the mouse. When the mouse moves over it, we supply the box with the text we want it to display, and when it moves out, we reset it. Note that resetting it is by giving it a value of null, or "

**NOTE:**
**When we want to "line break" text using JavaScript in a textarea, we use "\n" If you don't line break text when it exceeds the width of the textarea box, scrollbars will appear to accommodate the new text. For example:**

document.a.b.value="Text with no line breaks...very ugly."




document.a.b.value="Text WITH line\n breaks...very good!."




**Note: IE 4.x will automatically line-break text in either case, so that might be why you're saying to yourself: "I don't see no difference in the two!"**



**Introduction to validating forms** ▶

## Introduction to validating forms



**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

One of the most practical uses of JavaScript is validating forms. Lets first define what is meant by validating forms. Well, using JavaScript, you can check what a user enters into a form, intercept a form from being submitted and ask the user to retype or fill in the unfilled entries of a form before re-submission. Before, you would have to do this using CGI, which is both something I don't want to touch, and something you don't want to either...you with me? Anyone?



**Using the onBlur event handler**

## Using the onBlur event handler

We will now introduce two event handlers, ones especially for forms, that are used commonly when doing form validations.

**Additional Event handlers for forms**

| | |
|---|---|
| **onblur:** | Executes JavaScript whenever a user moves with the mouse the focus away from an element within a form. In other words, whenever a person first clicks an element, and then clicks anywhere outside of it. |
| **onsubmit:** | Executes JavaScript whenever someone clicks the "submit" button. |

Ok, you need clarification. *Onblur is used when you want to check each element (ie: text, textarea boxes ) individually, and ask the user to correct the input before moving on to the next box*. The other event, onsubmit, validates the form at the end, as the user presses the submit button. I prefer the later, but its up to you.

Ok, lets see exactly how onblur works. We're going to write a script that checks the first text box , and asks for correction as soon as text has been entered and the user moves the focus onto another box (assuming the person has entered "wrong" data.) Here we go:

**Enter your email address:** (Try entering something without the "@", and you'll get a alert telling you to re-enter the data as you proceed to the Feedback box.)

**Feedback please:**

Submit

```
<script>
function emailcheck()
{
var string1=document.example.email.value
if (string1.indexOf("@")==-1)
 {
 alert("Please input a valid email address!")
 document.example.email.focus()
```

```
.}
}
</script>

<form name="example"><input type="text" size="20" name="email"
onblur="emailcheck()">
<strong>Feedback please:</strong>
<textarea name="S1" rows="2" cols="20"></textarea>
<input type="submit" name="B1" value="Submit">
</form>
```

Ok, lets have a look at what's inside the <script> tags.

- We defined a function called emailcheck() that'll be called by the onblur event hander when the user clicks elsewhere away from that element.
- We copied the value of the box into "string1". Why? Simply for easier reference, you don't have to do this.
- Here comes the part that requires explaining. What is: "string1.indexOf("@")==-1" in the code? Well, first of all, you need to understand that all data entered into a text, text area box are string variables-variables that are treated as characters, instead of numbers. For ex, "Hello there" or "12343" are strings, 12343 is a number. Now, here is the surprise: string itself is also an object. If that's the case, that means it has methods/properties. One of the methods, you might have guessed it, is *indexOf("the char you are looking for")*. Using this method, we can search every character within a string and look for what we want. If it finds it will return the position of the char within the string. If it doesn't, it will return -1. Therefore, "string1.indexOf("@")==-1" basically means: "if @ is not in the string, do this:
  alert("Please input a valid email address!")
  document.example.email.focus()
  What's the thing in red? Well, focus() is a method of the text box, which basically forces the cursor to be at the specified text box, which is like the user clicking inside the text box with the mouse.Why do we need this? If we didn't, the only thing the user that entered bad data will see is the message "Please input a valid email address!" but THEN still allowed to move on onto the second box and submit the form. That's merely a slap on the face, and will not truly screen out those that are determined to not follow the rules. However, if we use the focus() method, the focus will be set back to the box containing errors, and will only allow users to proceed after it has been corrected.
- This is not a fool proof validation...in fact, you could enter "@sprynet" and get away with it. However, that can always be corrected!

If you've been playing around with the onblur event handler, even with simply the above example, you might have noticed that it can be very annoying at times; the better way is to simply defer the checking until a user presses the submit button, and check for errors all in one scoop. That's where the onsubmit handler comes in.



**Using the onSubmit event handler** 

## Home / JavaScript Tutorials / Accessing And Validating Forms



### onsubmit

The "onsubmit" handler is inserted inside the <form> tag, and not inside any one element, unlike "onblur." For example: <form name="george" onsubmit="what_ever"> Lets do an example:

**Enter Your name (*required)** If you leave the required sections blank, as you click submit, you will be forced to come back and fill this box in.

**Feedback please: (*required)**

**Your home address (*NOT required)**

Submit

```
<script>
<!--
function validate()
{
if ((document.example2.naming.value=="")||
(document.example2.feed.value==""))
 {
 alert ("You must fill in all of the required  fields!")
 return false
 }
}
//-->
</script>

<form name="example2" onsubmit="return validate()">

<input type="text" size="20" name="naming">

<strong>Feedback please: (*required)</strong>
```

```
<textarea name="feed" rows="3" cols="25"></textarea>

<strong>Your home address (*NOT required)</strong>

<input type="text" size="35" name="address">

<input type="submit" name="B1" value="Submit">

</form>
```

Some every important things here:

- document.example2.naming.value==''.What is the quotation in orange? That is used to indicate an empty value-something that contains nothing. It is important that you distinguish between "" and " " The later means "1 empty space", as opposed to "empty value". The later is a char-namely, a space.
- What is "return true", "return false"? This is what's used to actually allow, or stop the form from submitting, respectively. This is how JavaScript controls the submitting of a form. By default, a form will return true. (Submit the form). This is a important point-for example, by using the above knowledge, you can apply it to also stop a link from completing upon clicking. I'll show you an example:
  <a href="normal.htm" onclick="return false">Click here, it won't work!</a>
  Click here, it won't work!
  By returning false, we prohibit the action from completing!

- Now, a confusing point may be-onsubmit="return validate()" why return validate()? Wouldn't that be like a double return? No. Function validate only returns "true/false". You need "return true/fast" to actually manipulate whether a form submits or not. That's why we have to return validate(), as opposed to just validate().

Ok, in this tutorial, we ran into the string object, so lets list all its properties and methods for quick glance:

| properties | Methods |
|---|---|

| length | anchor |
| --- | --- |
| | big |
| | blink |
| | bold |
| | fixed |
| | fontcolor |
| | italics |
| | small |
| | strike |
| | sub |
| | sup |
| | charAt |
| | indexOf, lastIndexOf |
| | link |
| | split |
| | substring |
| | toLowerCase |
| | toUpperCase |
| | |

-Tutorial introduction
-The object model of JavaScript
-The JavaScript object tree
-Accessing forms using JavaScript
-Introduction to validating forms
-Using the onBlur event handler
-Using the onSubmit event handler

..

**Emd of tutorial**

## Home / JavaScript Tutorials / I'll have a double combo please

# I'll have a double combo please

Double combo coming up!.In this tutorial, we'll look at how to create combo link boxes using JavaScript.



**Working with selection lists** 

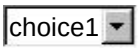
# Home / JavaScript Tutorials / I'll have a double combo please

## Working with selection boxes



Selection lists are quite commonly used in forms-with some help from JavaScript, it can be very useful sometimes.

Example of selection list:




The fundamental way to access forms is through its name. Accessing selection lists are no different. Here's an example:

```
<form name="George">
<p><select name="example" size="1">
<option>choice1</option>
<option>choice2</option>
<option>choice3</option>
<option>choice4</option>
</select></p>
</form>
```

To access anything within this selection list, we would write:

document.George.example

Accessing selection lists in JavaScript can be quite confusing. Where's the confusion, you ask? Well, the above line will access only the selection list itself, and not the individual elements. We have to go deeper to do that. Lets discuss how this is done.

-Tutorial introduction
-Working with selection lists
-Using the options array to access elements within the list
-Properties of the options array itself
-Example: Creating "combo link boxes"

**Using the options array to access elements within the list**

## Using the options array to access elements within the list

Using the above list as an example, with values added in for each element:

```
<form name="George">
<p><select name="example" size="1">
<option value="1">choice1</option>
<option value="2">choice2</option>
<option value="3">choice3</option>
<option value="4">choice4</option>
</select></p>
</form>
```

JavaScript gives us access to the *individual* elements (choice1, choice 2, choice 3 etc) by using the options array. An array is a series of variables all with the same name, but with different index numbers. (More on this in the future). This is, however, all you're need to know about arrays to access selections.


Ok, lets assume we want to access the first element of this selection list:

```
document.George.example.options[0]
//gives us access to element "choice1"
```

This will give us direct access to the first element of the selection list.

To access the third element, we would do this:

```
document.George.example.options[2]
//gives us access to element "choice3"
```

And to access the final element, we would do this:

```
document.George.example.options[3]
//gives us access to element "choice4"
```

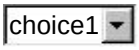
It doesn't take a rocket scientist to realize that the number inside the [ ] are always "1" minus the actual position of the element in the list. ALL ARRAYS BEGIN WITH INDEX NUMBER "0", NOT "1"

So, now that we know the way to get to these little rats, lets see some properties we can use with them:

**Properties of the elements of the options array**

| properties | description |
|---|---|
| text | returns the actual text (name) of the element |
| value | returns the value of the element |

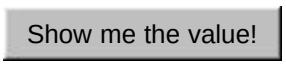
Lets say we want to alert the name of the first element (in this case, the name is "choice1"):





onClick=
"alert(document.George.example.options[0].text)"

And lets say we want to know the value that's associated with this element:




onClick=
"alert(document.George.example.options[0].value)"

I know selection lists are a mess, but that's the way they are accessed...

We now know how to access each of the elements within the selection list...but only by hard-coding the index number into the script. For example, we still have no way of telling our script to alert the value of each element, depending on what he/she has selected. We will now proceed to learn how we can have scripts automatically update itself whenever a different selection is made by the user, and see some practical uses of this.

-Tutorial introduction
-Working with selection lists
-Using the options array to access elements within the list
-Properties of the options array itself
-Example: Creating "combo link boxes"

**Properties of the options array itself** ▶

**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

## Properties of the options array itself

The keyword here is "itself". In the last page, *we looked at properties of elements of the options array*, NOT the options array itself.

Just to refresh our minds, here are the properties we looked at **last** section:

**Properties of the elements of the options array**

| properties | description |
|---|---|
| text | returns the actual text (name) of the element |
| value | returns the value of the element |

We will now look at properties of the options array itself:

**Properties of the options array itself**

| properties | description |
|---|---|
| length | tells us the number of elements within this selection list |
| selectedIndex | tells us the index number of the selected option |

Using the above table, to use the properties, we would do the following:

document.formname.selectionname.options.property

Lets see this in action:

choice1

```
<form name="George">
<p><select name="example" size="1">
<option value="1">choice1</option>
<option value="2">choice2</option>
<option value="3">choice3</option>
</select></p>
</form>
```

Using the length property, we can see how many elements are in the selection list:

Click here to see

alert(document.George.example.options.length)

As you can see, it alerts "3", since there are three elements in the list.

The selectedIndex property informs the index number of the *element selected, and updates itself whenever a new selection is made.*

For example:

Make a selection:


onClick=
"alert(document.George.example.options.selectedIndex)"

The selectedIndex property updates itself if you change elements, so if you had selected "choice1", it alerts "0" (since array indexes start at 0), if you than changed and selected "choice2", it will alert "1" instead. The example above may not seem like much, but it paves the way for many useful scripts.

-Tutorial introduction
-Working with selection lists
-Using the options array to access elements within the list
-Properties of the options array itself
-Example: Creating "combo link boxes"

**Example: Creating "combo link boxes"** 

## Home / JavaScript Tutorials / I'll have a double combo please

.

### Example: Creating "combo link boxes"



A common use of the above knowledge is to create a combo link box, where the selection list acts as a url jumper. Here's an example:

Cnet Go!

Before we plunge right in, lets first put together all our ideas we've learned in Lesson15 and 16. Recall from Lesson 15, that to access the individual elements, we would write:

```
document.George.example.options[x]
```

where "x" is the index number of the element.

Also recall from what we've just mentioned, that the selectedIndex property of the options array returns the index of the element that's currently selected:

```
document.George2.example.options.selectedIndex
```

Now, with the above two pieces of code, we are all set to make our combo box script!

Cnet Go!

```
<form name="George">
<p><select name="example" size="1">
<option value="http://www.cnet.com">choice1</option>
<option value="http://www.cnn.com">choice2</option>
<option value="http://www.geocities.com">choice3</option>
</select></p>

<script language="javascript">
<!--
function go()
{
location=document.George.example.
options[document.George.example.selectedIndex].value
}
//-->
</script>

<input type="button" name="test" value="Go!" onClick="go()">
</form>
```

Pay close attention to how we put together the two pieces of code mentioned earlier:

```
location=
document.George.example.
```

```
options[document.George.example.selectedIndex].value
```

See, the part in red, the script that returns the index of the element selected, is put inside the script that returns the value of that particular element. Now, since the "red" script is a self updating script that automatically changes index numbers whenever a user selects another element, the "blue" script will in turn always get the value (url) of the element the user has selected.

The above only illustrates one of the many uses of selection lists...the fun of it all is using your creativity with JavaScript and "doing your own thing."

-Tutorial introduction
-Working with selection lists
-Using the options array to access elements within the list
-Properties of the options array itself
-Example: Creating "combo link boxes""

**End of Tutorial**

# I'll have a double combo please II



**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

In this second tutorial on combo boxes, we'll tackle and show you how to implement some of the most common add-ons to combo link boxes seen on the net. You've undoubting seen combo boxes that jump to a URL simply upon selecting it, or ones that use an image (instead of the dull form button) as the "go" button. Why settle for a cheese burger when you can have a whopper?



## A quick overview of the syntax of a combo link box

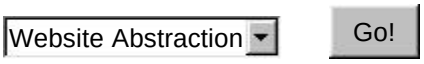
## Home / JavaScript Tutorials / I'll have a double combo please II

### A quick overview of the syntax of a combo link box

**WA Help Forum**
Having trouble with anything? Visit our help forum to get the answers you need.

**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

Before we start diving in, its appropriate to first show the code of the original combo link box we put together in the tutorial I'll have a double combo please. This is the code we will be building upon as we renovate our combo with various add-ons:




```
<form name="mycombo">
<p><select name="example" size="1">
<option value="http://www.cnet.com">Cnet</option>
<option value="http://www.cnn.com">CNN</option>
<option value="http://www.geocities.com">Geocities</option>
</select></p>

<script language="javascript">
<!--
function go()
{
location=
document.mycombo.example.
options[document.mycombo.example.selectedIndex].value
}
//-->
</script>

<input type="button" name="test" value="Go!" onClick="go()">
</form>
```

All set? Good. Lets get started.

-Tutorial introduction
-A quick overview of the syntax of a combo link box
-Creating a combo box that loads the target URL in another frame
-Using an image instead of a form button as the "go" button
-Creating a combo box that jumps to a URL upon selecting

**Creating a combo box that loads the target URL in another frame**

# Home / JavaScript Tutorials / I'll have a double combo please II



## Creating a combo box that loads the target URL in another frame

You've been pounding your head over the keyboard trying to get a combo box to load the selected URL in another frame. Well, its actually very simple, and to accomplish such a task, there are simply two things that needs to be done:

1) Give each frame a name by using the name attribute inside the <frame src='..'> tags
2) Use the frame's name in place of "location" in function go() of our original combo code.

Lets take this one step at a time.

First, give each frame a name. We're use the below simple frames page as an example:

```
<html>
<frameset cols="30%,70%">
<frame src="page1.htm" name="Peter">
<frame src="page2.htm" name="Jane">
</frameset>
</html>
```

The parts in red are each frame's name, and is needed before we can tell JavaScript which frame we want the combo box to load the specified URL in.

Now, having done that, the remaining job is simply to modify our combo code so it targets the desired frame. Assuming the combo code is in page1.htm, and we want it to load the URL into page2.htm. Here's how:

```
<script language="javascript">
<!--
function go()
{
parent.Jane.location=
document.mycombo.example.
options[document.mycombo.example.selectedIndex].value
}
//-->
</script>
```

Notice the keyword "parent". This object represents the parent window in a frames page, and is required as a prefix, since we need to "walk up" one level in order to reach page2.htm, a document parallel in level ti page1.htm. Following "parent" is "Jane", the frame name of page2.htm.

That's all there is to it. Lets see an example of such a combo box:

**Click here to see example.**

-Tutorial introduction
-A quick overview of the syntax of a combo link box
-Creating a combo box that loads the target URL in another frame
-Using an image instead of a form button as the "go" button
-Creating a combo box that jumps to a URL upon selecting

**Using an image instead of a form button as the "go" button** 

# Home / JavaScript Tutorials / I'll have a double combo please II

## Using an image instead of a form button as the "go" button



I don't know who defined the appearance of form buttons as HTML was created, but most would agree that they are quite ugly. Luckily, JavaScript was invented shortly after. In this section, we'll illustrate how to replace the usual (<input type="button") button associated with a combo box with a more eye-catching image.

By default, most combo boxes have the below form button attached to the end of it, which goes to the selected URL upon clicking it:

```
<input type="button" name="test" value="Go!" onClick="go()">
</form>
```

The "meat" of the above button is the code onClick="go()", which tells JavaScript to execute a function ("go()" in this case) if the button is clicked on. With that in mind...

To use an image instead, simply define a hyperlinked image that does the same thing- executes a function when clicked on. There are several methods that will help us achieve that, the simplest being a JavaScript URL:

```
<a href="JavaScript:go()"><img src="combo.gif"></a>
```

The part in red is a JavaScript URL, and whatever we put after the semicolon (:), JavaScript executes it as the link is clicked on. Since we want it to execute function go(), "go()" is put inside it.

Below is  a combo box that uses an image as the "warper"

Cnet 

-Tutorial introduction
-A quick overview of the syntax of a combo link box
-Creating a combo box that loads the target URL in another frame
-Using an image instead of a form button as the "go" button
-Creating a combo box that jumps to a URL upon selecting

**Creating a combo box that jumps to a URL upon selecting** 

# Home / JavaScript Tutorials / I'll have a double combo please II

## Creating a combo box that jumps to a URL upon selecting



In this final section of our tutorial, we'll see how to implement a rather jumpy add-on to a combo box- the ability to go to the specified URL in a combo box simply upon selecting it. The secret to making this happen lies in the onChange event handler. The onChange event handler is inserted inside the select tag, which reacts to the surfer changing the selection inside the combo box. Let's
modify our old combo box so it jumps to a url this way. The revision is highlighted in red:

```
<form name="George">
<p><select name="example" size="1" onChange="go()">
<option value="http://www.cnet.com">Cnet</option>
<option value="http://www.cnn.com">CNN</option>
<option value="http://www.geocities.com">Geocities</option>
</select></p>

<script language="javascript">
<!--
function go()
{
location=
document.George.example.
options[document.George.example.selectedIndex].value
}
//-->
</script>

<input type="button" name="test" value="Go!" onClick="go()">
</form>
```


As you can see, all we really did here is add a onChange event to the select tag, which causes
function go() to react as soon as the selection in the combo link box changes. Now, we still need
the "Go!" button in this case. The onChange event handler only reacts if the selection has changed.
If the user wants to go to the initially selected element, the"Go!" button has to be pressed.

In this tutorial, we've learned how to implement many of the little "gadgets" commonly used with existing combo boxes. Regardless of how you implement a combo box, bear in mind that it is only compatible with NS 3+, and IE 4+. It is NOT compatible with IE 3.x nor NS 2.x, due to the fact that these browsers do not support the options property required to access the individual entries in a <select> tag.

-Tutorial introduction
-A quick overview of the syntax of a combo link box
-Creating a combo box that loads the target URL in another frame
-Using an image instead of a form button as the "go" button
-Creating a combo box that jumps to a URL upon selecting

.

**End of tutorial**

**Home / JavaScript tutorials / Arrays and loops**

# Arrays and loops

**WA Help Forum**
Having trouble with anything? Visit our help forum to get the answers you need.

**Jump to...**
-Free JavaScripts
-JS tutorials
-Free applets
-Web Tutorials
-Freewarejava

In this tutorial, we will explore one of the more dry areas of JavaScript, where instant gratification isn't always the case. Hmmm, you say. Before you reach for the back button, let me assure you that this is something we all will eventually have to learn in able to create complex JavaScripts. So are you all with me? Ok, in this section then, we will look at:

-Tutorial introduction
-Understanding and implementing arrays
-The "for" loop

**Understanding and implementing arrays** 

# Understanding and implementing arrays



So what are arrays? Well, they are an alternative way of defining variables that allow multiple variables to be quickly declared, painlessly. If you're a little confused at this stage, its ok.

Lets first quickly go over the "standard" way of declaring variables. Assuming we want to declare two variables to store the names of your in-laws...here's how it could be done:

```
<script>
var x="Bob"
var y="May"
</script>
```

Pure and simple, right? Now, lets say you're really bored and want to store all of the known relatives in your family tree, all 50 of them, using JavaScript...we could do this:

```
<script>
var x="Sam"
var y="Bob"
var n="Joe"
var v="Peter"
var m="May"
"
"
"
</script>
```

Pure, right? But not very simple. In order to define 50 variables, We had to think of a unique variable name for each one (such as x, y, n, and so on) while keeping track of which variable we're currently on. Well, with arrays, that kind of hassle is non-existent.

Just another friendly reminder: arrays are merely another way of defining variables, and nothing more. Lets first see how an array in general is created. To define an array, do the following:

var arrayname=new Array(# of variables to be created)

For example:

var x=new Array(50)

Now, to store the names into this array, we would do the following instead:

x[0]="Sam"
x[1]="Bob"
x[2]="Joe"
x[3]="Peter"
x[4]="May"
"
"
"
x[49]="George"

Are you starting to see the advantages arrays have over the standard method when declaring large chunks of variables?

Some advantages of arrays:

- No need to provide each variable with a unique name.
- Easier tracking and reference to any of the 50 variables.

As you have seen, individual variables within an array are referenced using brackets [ ] with an integer in there. You may also have noticed that the number in there started at 0, instead of 1. Why, you ask? Well, In the demented world of JavaScript, people like to count starting from 0; hours are counted starting at 0, the month of the year is too, so predictably, so are the numbers inside an array.

The biggest advantage of storing variables in arrays is undoubtedly when it comes to referencing them...you can now literally yell: " tell me who is the 25th person down my family tree!" and you're get an answer easily:

<script>
alert(x[24])
</script>

That kind of versatility is quite difficult without arrays.

-Tutorial introduction
-Understanding and implementing arrays
-The "for" loop

**The "for" loop** 

# The "for" loop



The "for" loop is a JavaScript "method" that allows a certain action (ie: block of code) to be performed continuously in a variably controlled fashion. It is very similar to a "while" loop in which lines of JavaScript codes can be grouped together and repeated until a certain condition. A real life example of a for loop would be as follows:

for length equal to 0 to length equal to 100
run!

Lets see the general syntax of for loops in JavaScript, where y is an arbitrary variable:

```
for (y=0;y<=99;y++){
alert ("hi!")
}
```

The above example will alert "hi!" 100 times. Lets look more closely at the heart of a "for" loop:

```
for (y=0;y<=99;y++)
```

It consists of three components

y=0 //The starting point of a for loop

y<=99 //The ending boundary of a for loop

y++ //How the "for" loop is incremented. y++ means increment it by one step each time until the boundary.

The part that may be confusing is the last part, "y++". Let's see the same above example, only this time, altering that particular part:

```
for (y=0;y<=99;y=y+2){
alert ("hi!")
}
```

How many "hi" will be alerted? Well, 50 will, because we are incrementing the "counter" not by 1, but by 2 this time. The point is, you can determine however you want to increment the for loop by assigning a different statement in the third component of the for loop.

Lets see a practical example of a "for" loop than, where it is used to calculate the sum of 1+2+3+.....all the way up to the specified number:

Click here

Here's the function that performs this calculation:

```
<script>
function cal(){
var y=1
var temp=prompt("Please input a positive integer:")
for (x=2;x<=temp;x++){
y=y+x
}
alert("1+2+...+"+temp+"="+y)
}
</script>
```

The "for" loop is a major part of any programming language, and JavaScript is no exception.

-Tutorial introduction
-Understanding and implementing arrays
-The "for" loop

**End of tutorial**