JavaScript

Client-Side Programming

- HTML is good for developing *static* pages
  - can specify text/image layout, presentation, links
  - Web page looks the same each time it is accessed

  In order to develop interactive/reactive pages, must integrate programming in some form or another

- client-side programming
  - programs are written in a separate programming (or scripting) language

    e.g., JavaScript, JScript, VBScript
  - programs are embedded in the HTML of a Web page, with (HTML) tags to identify the program component

    e.g., `<script type="text/javascript"> … </script>`
  - the browser executes the program as it loads the page, integrating the dynamic output of the program with the static content of HTML
  - could also allow the user (client) to input information and process it, might be used to validate input before it's submitted to a remote server

Scripts vs. Programs

- a scripting language is a simple, interpreted programming language
  - scripts are embedded as plain text, interpreted by application
  - *simpler execution model:* don't need compiler or development environment
  - *saves bandwidth:* source code is downloaded, not compiled executable
  - *platform-independence:* code interpreted by any script-enabled browser

  but: slower than compiled code, not as powerful/full-featured

JavaScript: the first Web scripting language, developed by Netscape in 1995
syntactic similarities to Java/C++, but simpler, more flexible in some respects, limited in other (loose typing, dynamic variables, simple objects)

JScript: Microsoft version of JavaScript, introduced in 1996 same core language, but some browser-specific differences fortunately, IE, Netscape, Firefox, etc. can (mostly) handle both JavaScript & JScript

JavaScript 1.5 & JScript 5.0 cores both conform to ECMAScript standard

VBScript: client-side scripting version of Microsoft Visual Basic

Common Scripting Tasks

- adding dynamic features to Web pages
  - validation of form data (probably the most commonly used application)
  - image rollovers
  - time-sensitive or random page elements
  - handling cookies
- defining programs with Web interfaces
  - utilize buttons, text boxes, clickable images, prompts, etc

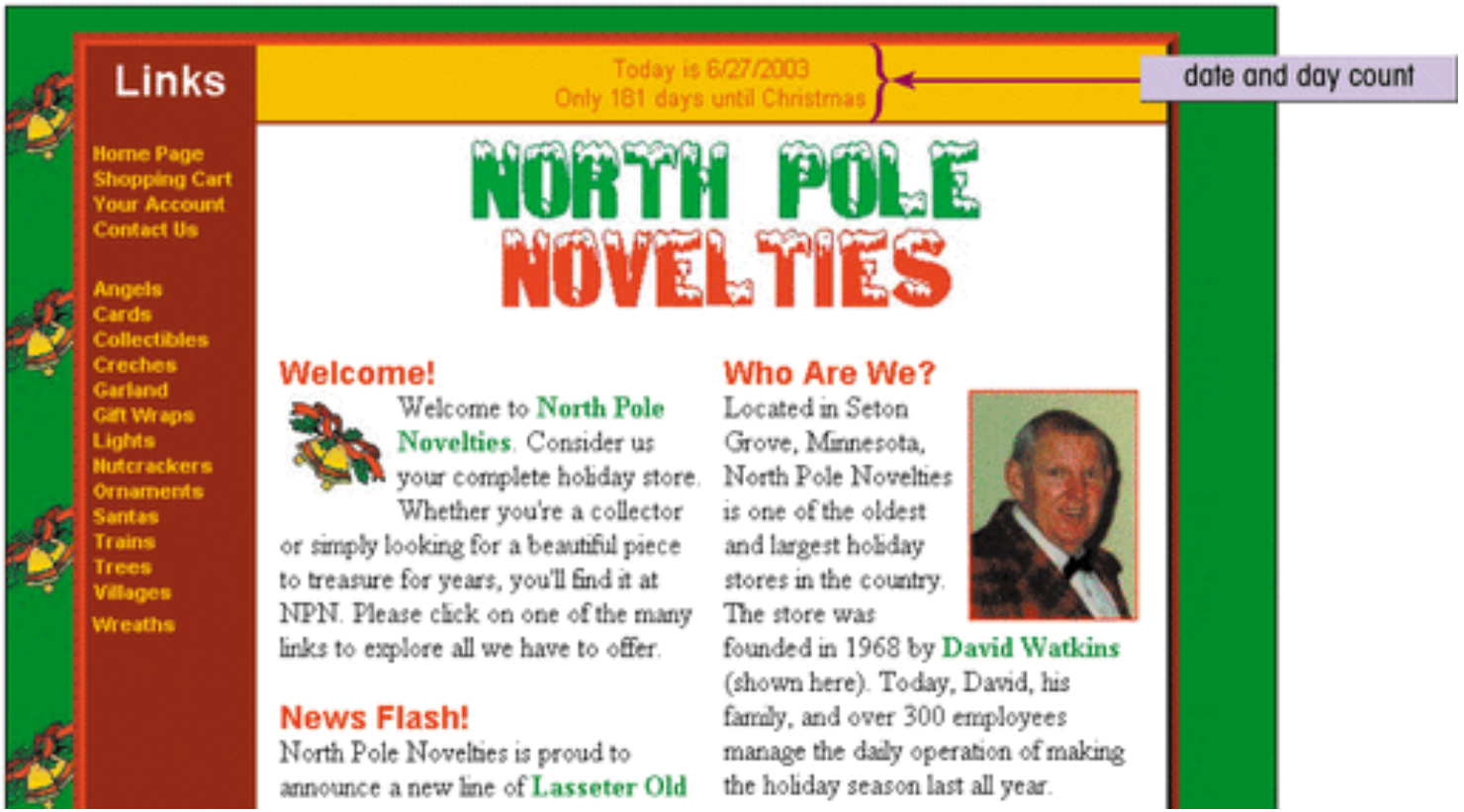
- limitations of client-side scripting
  - since script code is embedded in the page, it is viewable to the world
  - for security reasons, scripts are limited in what they can do
    e.g., can't access the client's hard drive
  - since they are designed to run on any machine platform, scripts do not contain platform specific commands
  - script languages are not full-featured
    e.g., JavaScript objects are very crude, not good for large project development

Java vs. JavaScript

- Requires the JDK to create the applet
- Requires a Java virtual machine to run the applet
- Applet files are distinct from the XHTML code
- Source code is hidden from the user
- Programs must be saved as separate files and compiled before they can be run
- Programs run on the server side

- Requires a text editor
- Required a browser that can interpret JavaScript code
- JavaScript can be placed within HTML and XHTML
- Source code is made accessible to the user
- Programs cannot write content to the hard disk
- Programs run on the client side

Example of Web Site using JavaScript



Why you want to study JavaScript?

JavaScript is one of **3** languages all web developers **MUST** learn:

- **HTML** to define the content of web pages
- **CSS** to specify the layout of web pages
- **JavaScript** to program the behavior of web pages

What can we do with JavaScript?

- To create interactive user interface in a web page (e.g., menu, pop-up alert, windows, etc.)
- Manipulating web content dynamically
  - Change the content and style of an element
  - Replace images on a page without page reload
  - Hide/Show contents
- Form validation

What is JavaScript ?

- JavaScript is a lightweight, interpreted programming language
- Designed for creating network-centric applications
- Complementary to and integrated with Java
- Complementary to and integrated with HTML
- Open and cross-platform
- JavaScript is the most popular programming language in the world.
- It is the language for HTML, for the Web, for computers, servers, laptops, tablets, smart phones, and more.

JavaScript Syntax

- A JavaScript consists of JavaScript statements that are placed within the <script>... </script> HTML tags in a web page.

- You can place the <script> tag containing your JavaScript anywhere within you web page but it is preferred way to keep it within the <head> tags.

- The <script> tag alert the browser program to begin interpreting all the text between these tags as a script. So simple syntax of your JavaScript will be as follows

```
<script ...>
JavaScript code
</script>
```

The script tag takes two important attributes:

- **language:** This attribute specifies what scripting language you are using. Typically, its value will be *javascript*. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.

- **type:** This attribute is what is now recommended to indicate the scripting language in use and its value should be set to *"text/javascript"*.

- So your JavaScript segment will look like:

```
<script language="javascript" type="text/javascript">
JavaScript code
  </script>
```

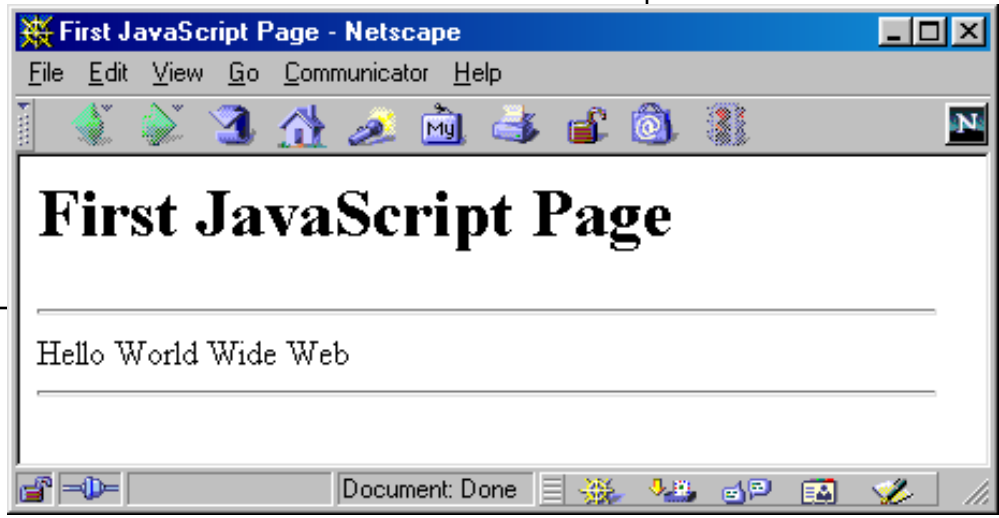
Your First JavaScript Program: Example to print "Hello World".

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

Above code will display following result:
Hello World!

A Simple Script

```
<html>
<head><title>First JavaScript Page</title></head>
<body>
<h1>First JavaScript Page</h1>
<script type="text/javascript">
  document.write("<hr>");
  document.write("Hello World Wide Web");
  document.write("<hr>");
</script>
</body>
</html>
```



Embedding JavaScript

```
<html>
<head><title>First JavaScript Program</title></head>
<body>
<script type="text/javascript"
        src="your_source_file.js"></script>
</body>
</html>
```

Inside your_source_file.js

```
document.write("<hr>");
document.write("Hello World Wide Web");
document.write("<hr>");
```

- Use the **`src`** attribute to include JavaScript codes from an external file.
- The included code is inserted in place.

Embedding JavaScript

- The scripts inside an HTML document is interpreted in the order they appear in the document.
  - Scripts in a function is interpreted when the function is called.

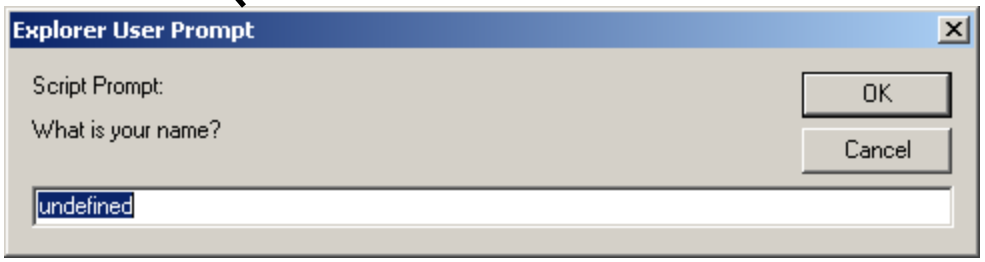
- So where you place the <script> tag matters.

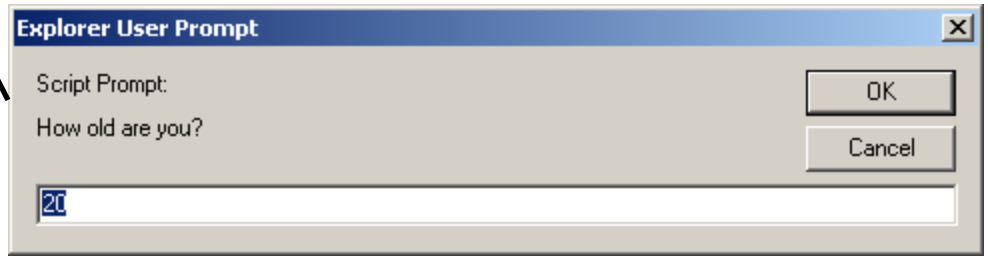
Hiding JavaScript from Incompatible Browsers

```
<script type="text/javascript">
<!-
  document.writeln("Hello, WWW");
// -->
</script>
<noscript>
  Your browser does not support JavaScript.
</noscript>
```

alert(), confirm(), and prompt()

```
<script type="text/javascript">
alert("This is an Alert method");
confirm("Are you OK?");
prompt("What is your name?");
prompt("How old are you?","20");
</script>
```





Whitespace and Line Breaks

- JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs.
- Because you can use spaces, tabs, and newlines freely in your program so you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

Semicolons are Optional

- Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if your statements are each placed on a separate line.
- For example, the following code could be written without semicolons

```
<script language="javascript" type="text/javascript">
   var1 = 10
   var2 = 20
  </script>
```

But when formatted in a single line as follows, the semicolons are required:

```
<script language="javascript" type="text/javascript">
 var1 = 10; var2 = 20;
</script>
```

Note: It is a good programming practice to use semicolons.

Case Sensitivity

- JavaScript is a case-sensitive language. This means that language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

- So identifiers *Time*, *TIme* and *TIME* will have different meanings in JavaScript.

NOTE: Care should be taken while writing your variable and function names in JavaScript.

Comments in JavaScript

JavaScript supports both C-style and C++-style comments, Thus:

- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

JavaScript Placement in HTML File

There is a flexibility given to include JavaScript code anywhere in an HTML document. But there are following most preferred ways to include JavaScript in your HTML file.

- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in <body>...</body> and <head>...</head> sections.
- Script in and external file and then include in <head>...</head> section.

JavaScript DataTypes

JavaScript allows you to work with three primitive data types:

- Numbers eg. 123, 120.50 etc.
- Strings of text e.g. "This text string" etc.
- Boolean e.g. true or false.
- JavaScript also defines two trivial data types, *null* and *undefined*, each of which defines only a single value.

Data Types

- Primitive data types
  - Number: integer & floating-point numbers
  - Boolean: true or false
  - String: a sequence of alphanumeric characters

- Composite data types (or Complex data types)
  - Object: a named collection of data
  - Array: a sequence of values (an array is actually a predefined object)

- Special data types
  - Null: the only value is "null" – to represent nothing.
  - Undefined: the only value is "undefined" – to represent the value of an uninitialized variable

JavaScript Variables

- Like many other programming languages, JavaScript has variables.
- Variables can be thought of as named containers.
- You can place data into these containers and then refer to the data simply by naming the container.
- Before you use a variable in a JavaScript program, you must declare it.
- Variables are declared with the **var** keyword as follows:

```
<script type="text/javascript">
        var money; var name;
</script>
```

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variable will have only two scopes.

- **Global Variables:** A global variable has global scope which means it is defined everywhere in your JavaScript code.

- **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

JavaScript Variable Names

- While naming your variables in JavaScript keep following rules in mind.
- Not use any of the JavaScript reserved keyword as variable name. For example, *break* or *boolean* variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or the underscore character. For example, *123test* is an invalid variable name but *_123test* is a valid one.
- JavaScript variable names are case sensitive. For example, *Name* and *name* are two different variables.

```
<script type="text/javascript">
 <!–
var myVar = "global"; // Declare a global variable

function checkscope( )
{
    var myVar = "local"; // Declare a local variable
    document.write(myVar);
}
//-->
</script>
```

This produces the following result:

local

The *typeof* Operator

- The *typeof* is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.
- The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

```
<html>
<body>

<script type="text/javascript">
<!--
var a = 10;
var b = "String";
var linebreak = "<br />";

result = (typeof b == "string" ? "B is String" : "B is Numeric");
document.write("Result => ");
document.write(result);
document.write(linebreak);

result = (typeof a == "string" ? "A is String" : "A is Numeric");
document.write("Result => ");
document.write(result);
document.write(linebreak);

//-->
</script>

<p>Set the variables to different values and different operators and then try...</p>

</body>
</html>
```

JavaScript Operators

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 y=2 x+y	4
-	Subtraction	x=5 y=2 x-y	3
*	Multiplication	x=5 y=4 x*y	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

JavaScript Operators – 2

Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

JavaScript Operators - 3

Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
===	is equal to (checks for both value and type)	x=5 y="5" x==y returns true x===y returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

JavaScript Operators - 4

Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

if statement

The **if** statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax:

```
if (expression)
{
  Statement(s) to be executed if expression is true
}
```

Example

```
<script type="text/javascript">
 <!–
var age = 20;
 if( age > 18 )
{ document.write("<b>Qualifies for driving</b>");
}
//-->
 </script>
```

This will produce following result:

Qualifies for driving

if...else statement

The **if...else** statement is the next form of control statement that allows JavaScript to execute statements in more controlled way.

Syntax:

```
if (expression)
{
Statement(s) to be executed if expression is true
}
else{
Statement(s) to be executed if expression is false
 }
```

Example

```
  <script type="text/javascript">
<!-- var age = 15;
 if( age > 18 )
{
 document.write("<b>Qualifies for driving</b>");
 }
else{
 document.write("<b>Does not qualify for
    driving</b>");
 }
//-->
 </script>
```

if...else if... statement

Syntax:

```
if (expression 1)
{
Statement(s) to be executed if expression 1 is true
}else if (expression 2){
Statement(s) to be executed if expression 2 is true
 }else if (expression 3){
Statement(s) to be executed if expression 3 is true
}else{
Statement(s) to be executed if no expression is true
 }
```

Example

```
   <script type="text/javascript">
 <!–
      var book = "maths";
if( book == "history" )
{ document.write("<b>History Book</b>");
}else if( book == "maths" ){
document.write("<b>Maths Book</b>");
}else if( book == "economics" ){
document.write("<b>Economics Book</b>");
}else{
document.write("<b>Unknown Book</b>");
}
//-->
</script>
```

switch statement

```
switch (expression)
{ case condition 1:
    statement(s)
    break;
 case condition 2:
    statement(s)
    break; ...
case condition n:
    statement(s)
    break;
default:
    statement(s)
 }
```

```
<script type="text/javascript">
<!–
      var grade='A';
      document.write("Entering switch block<br />");
switch (grade)
 {
case 'A': document.write("Good job<br />");
            break;
case 'B': document.write("Pretty good<br />");
                break;
case 'C': document.write("Passed<br />");
                break;
 case 'D': document.write("Not so good<br />");
                break;
case 'F': document.write("Failed<br />");
break;
default: document.write("Unknown grade<br />")
}
document.write("Exiting switch block");
//-->
</script>
```

```
<script type="text/javascript">
<!–
      var grade='A';
      document.write("Entering switch block<br />");
switch (grade)
 {
case 'A': document.write("Good job<br />");
case 'B': document.write("Pretty good<br />");
case 'C': document.write("Passed<br />");
 case 'D': document.write("Not so good<br />");
case 'F': document.write("Failed<br />");
default: document.write("Unknown grade<br />")
}
document.write("Exiting switch block");
//-->
</script>
```

This will produce following result:

Entering switch block
Good job
Pretty good
Passed
Not so good
Failed Unknown grade
Exiting switch block

The *while* Loop

```
while (expression)
{
 Statement(s) to be executed if expression is
    true
}
```

```
<script type="text/javascript">
<!–
var count = 0;
document.write("Starting Loop" + "<br />");
while (count < 10)
{
document.write("Current Count : " + count + "<br />"); count++; }
      document.write("Loop stopped!");
 //-->
</script>
```

This will produce following result:

```
Starting Loop
Current Count :0
Current Count : 1
Current Count : 2
Current Count : 3
 Current Count : 4
Current Count : 5
 Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

for...in loop

```
<script type="text/javascript">
 <!–
     var aProperty;
     document.write("Navigator Object Properties<br /> ");
    for (aProperty in navigator)
    { document.write(aProperty); document.write("<br />");
     } document.write("Exiting from the loop!");
//-->
</script>
```

This will produce following result:

```
Navigator Object Properties
 appCodeName
 appName
appMinorVersion
 cpuClass platform
plugins
opsProfile
 userProfile
systemLanguage
 userLanguage
appVersion
 userAgent
onLine
cookieEnabled
mimeTypes
 Exiting from the loop!
```

The *break* Statement

```
<script type="text/javascript">
<!–
var x = 1;
 document.write("Entering the loop<br /> ");
while (x < 20)
 {
if (x == 5)
{
break; // breaks out of loop completely
}
 x = x + 1;
 document.write( x + "<br />");
} document.write("Exiting the loop!<br /> ");
 //-->
 </script>
```

This will produce following result:

Entering the loop

2

3

4

5

Exiting the loop!

The *do...while* Loop

Syntax:

```
do
{
Statement(s) to be executed;
} while (expression);
```

```
<script type="text/javascript">
<!–
var count = 0;
document.write("Starting Loop" + "<br />");
 do
{
document.write("Current Count : " + count + "<br />");
 count++;
}while (count < 0);
 document.write("Loop stopped!");
//-->
</script>
```

This will produce following result:

```
Starting Loop
Current Count : 0
 Loop stopped!
```

The *for* Loop

Syntax:

for (initialization; test condition; iteration statement)

{

Statement(s) to be executed if test condition is true

}

```
<script type="text/javascript">
 <!—
 var count;
document.write("Starting Loop" + "<br />");
for(count = 0; count < 10; count++)
{ document.write("Current Count : " + count );
 document.write("<br />");
 }
document.write("Loop stopped!");
 //-->
 </script>
```

This will produce following result which is similar to **while** loop:

Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!

```
<html>
<head>
<title>Title of the Page</title>
<script language="JavaScript">

function goodbye(){
alert("Goodbye!")
}

</script>
</head>
<body onload="goodbye()">
Now you are leaving this page <a href="page2.html">for another</a>.
</body>
</html>
```

```
<html>
<head>
<title>Title of the Page</title>
<script language="JavaScript">

function goodbye(){
alert("Goodbye!")
}

</script>
</head>
<body onUnload="goodbye()">
Now you are leaving this page <a href="page2.html">for another</a>.
</body>
</html>
```

onchange event

```
<html>
<body>

<p>Modify the text in the input field, then click outside the field to fire the onchange event.</p>

Enter some text: <input type="text" name="txt" value="Hello" onchange="myFunction(this.value)">

<script>
function myFunction(val) {
   alert("The input value has changed. The new value is: " + val);
}
</script>

</body>
</html>
```

```
<HTML>
<HEAD>

<SCRIPT LANGUAGE="JavaScript">
<!-- Beginning of JavaScript -
function changecolor(code) {
document.bgColor=code }
// - End of JavaScript - -->
</SCRIPT>

</HEAD>

<BODY>
<form>
<input type="button" name="Button1" value="RED" onclick="changecolor('red')">
<input type="button" name="Button2" value="GREEN" onclick="changecolor('green')">
<input type="button" name="Button3" value="BLUE" onclick="changecolor('blue')">
<input type="button" name="Button4" value="WHITE" onclick="changecolor('white')">
</form>
</BODY>
</HTML>
```