What is JavaScript?

- JavaScript was designed to add interactivity to HTML pages
- JavaScript is a scripting language
- A scripting language is a lightweight programming language
- JavaScript is usually embedded directly into HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
- Everyone can use JavaScript without purchasing a license

What can a JavaScript do?

- **JavaScript gives HTML designers a programming tool -** HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages
- **JavaScript can put dynamic text into an HTML page -** A JavaScript statement like this: document.write("<h1>" + name + "</h1>") can write a variable text into an HTML page
- **JavaScript can react to events -** A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- **JavaScript can read and write HTML elements -** A JavaScript can read and change the content of an HTML element
- **JavaScript can be used to validate data -** A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing
- **JavaScript can be used to detect the visitor's browser** - A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser
- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer

The Real Name is ECMAScript

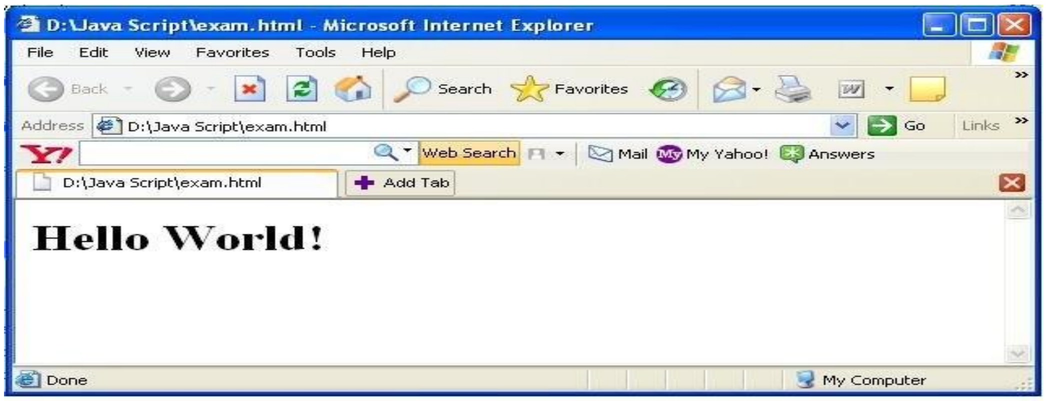
- JavaScript's official name is ECMAScript.
- ECMAScript is developed and maintained by the ECMA organization.
- ECMA-262 is the official JavaScript standard.
- The language was invented by Brendan Eich at Netscape (with Navigator 2.0), and has appeared in all Netscape and Microsoft browsers since 1996.
- The development of ECMA-262 started in 1996, and the first edition of was adopted by the ECMA General Assembly in June 1997.
- The standard was approved as an international ISO (ISO/IEC 16262) standard in

1998.

Example:

```
<html>
    <body>
            <script type="text/javascript">
                  document.write("<h1>Hello World!</h1>");
            </script>
      </body>
</html>
```

Output:


D:\Java Script\exam.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address D:\Java Script\exam.html Go Links
Web Search Mail My Yahoo! Answers
D:\Java Script\exam.html Add Tab
Done
My Computer

Hello World!

- To insert a JavaScript into an HTML page, we use the <script> tag. Inside the <script> tag we use the type attribute to define the scripting language.
- So, the <script type="text/javascript"> and </script> tells where the JavaScript starts and ends:
- The **document.write** command is a standard JavaScript command for writing output to a page.
- By entering the document.write command between the <script> and </script> tags, the browser will recognize it as a JavaScript command and execute the code line. In this case the browser will write Hello World! to the page:

Where to Put the JavaScript

- JavaScripts in a page will be executed immediately while the page loads into the browser. This is not always what we want. Sometimes we want to execute a script when a page loads, or at a later event, such as when a user clicks a button. When this is the case we put the script inside a function.

Scripts in <head>

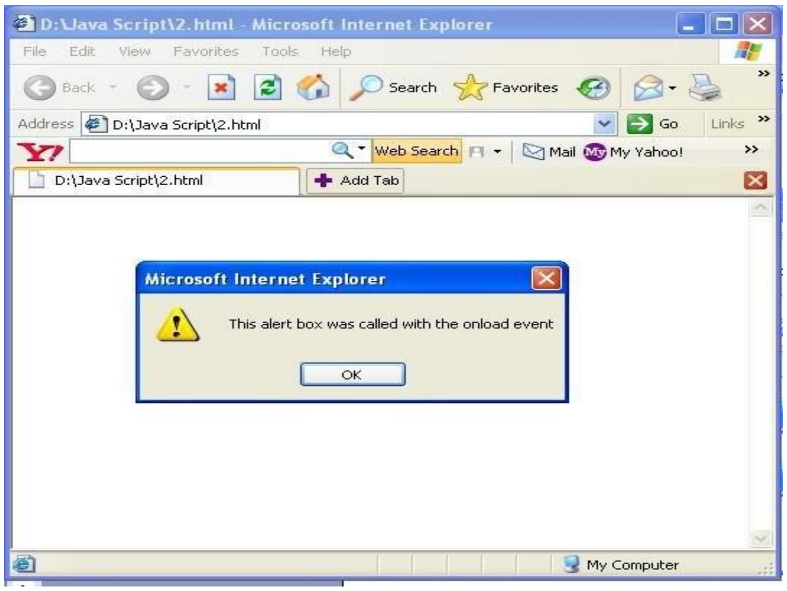
- Scripts to be executed when they are called, or when an event is triggered, are placed in functions.

- Put your functions in the head section, this way they are all in one place, and they do not interfere with page content.

Example

```
<html>
<head>
        <script type="text/javascript">
                function message()
                {
                    alert("This alert box was called with the onload event");
                 }
           </script>
</head>
 <body onload="message()">
</body>
</html>
```

Output:


D:\Java Script\2.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address D:\Java Script\2.html Go Links
Web Search Mail My Yahoo!
D:\Java Script\2.html Add Tab
Microsoft Internet Explorer
OK
My Computer

This alert box was called with the onload event

Scripts in <body>

- If you don't want your script to be placed inside a function, or if your script should write page content, it should be placed in the body section.

Example

```
<html>
<head></head>
```

```
        <body>
                    <script type="text/javascript">
                     document.write("This message is written by JavaScript");
                  </script>
        </body>
    </html>
```

Scripts in <head> and <body>

- You can place an unlimited number of scripts in your document, so you can have scripts in both the body and the head section.

Example

```
<html>
      <head>
             <script type="text/javascript">
                 function message()
                 {
                         alert("This alert box was called with the onload event");
                  }
              </script>
      </head>
      <body onload="message()">
              <script type="text/javascript">
                     document.write("This message is written by JavaScript");
                </script>
       </body>
</html>
```

Using an External JavaScript

- If you want to run the same JavaScript on several pages, without having to write the same script on every page, you can write a JavaScript in an external file.
- Save the external JavaScript file with a .js file extension.

  Note: The external script cannot contain the <script></script> tags!
- To use the external script, point to the .js file in the "src" attribute of the <script> tag.

Example

```
<html>
           <head>
                <script type="text/javascript" src="xxx.js"></script>
          </head> <body></body>
</html>
```

JavaScript is Case Sensitive: Unlike HTML, JavaScript is case.

JavaScript Statements: A JavaScript statement is a command to a browser. The purpose of the command is to tell the browser what to do.

JavaScript Comments:

- Comments can be added to explain the JavaScript, or to make the code more readable.
- Single line comments start with //.

JavaScript Multi-Line Comments: Multi line comments start with /* and end with */.

Do You Remember Algebra From School?

- Do you remember algebra from school? x=5, y=6, z=x+y
- Do you remember that a letter (like x) could be used to hold a value (like 5), and that you could use the information above to calculate the value of z to be 11?
- These letters are called **variables**, and variables can be used to hold values (x=5) or expressions (z=x+y).

JavaScript Variables

- As with algebra, JavaScript variables are used to hold values or expressions.
- A variable can have a short name, like x, or a more descriptive name, like carname.
- Rules for JavaScript variable names:
  - Variable names are case sensitive (y and Y are two different variables)
  - Variable names must begin with a letter or the underscore character

  Note: Because JavaScript is case-sensitive, variable names are case-sensitive.

Declaring (Creating) JavaScript Variables

- Creating variables in JavaScript is most often referred to as "declaring" variables.
- You can declare JavaScript variables with the **var** keyword:

```
var x;
var carname;
```

- After the declaration shown above, the variables are empty (they have no values yet).
- However, you can also assign values to the variables when you declare them:

```
var x=5;
var carname="Volvo";
```

- After the execution of the statements above, the variable **x** will hold the value **5**, and **carname** will hold the value **Volvo**.

  Note: When you assign a text value to a variable, use quotes around the value.

Assigning Values to Undeclared JavaScript Variables

- If you assign values to variables that have not yet been declared, the variables will automatically be declared.
- These statements:

  -
    ```
    x=5;
    carname="Volvo";
    ```

  have the same effect as:

```
var x=5;
var carname="Volvo";
```

JavaScript Arithmetic Operators

- Arithmetic operators are used to perform arithmetic between variables and/or values.

  Given that **y=5**, the table below explains the arithmetic operators:

Operator	Description	Example	Result
+	Addition	x=y+2	x=7
-	Subtraction	x=y-2	x=3
*	Multiplication	x=y*2	x=10
/	Division	x=y/2	x=2.5
%	Modulus (division remainder)	x=y%2	x=1
++	Increment	x=++y	x=6
--	Decrement	x=--y	x=4

JavaScript Assignment Operators

- Assignment operators are used to assign values to JavaScript variables.

  Given that **x=10** and **y=5**, the table below explains the assignment operators:

Operator	**Example**	**Same As**	**Result**
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

The + Operator Used on Strings

- The + operator can also be used to add string variables or text values together.
- To add two or more string variables together, use the + operator.

```
txt1="What a very";
txt2="nice day";
txt3=txt1+txt2;
```

JavaScript Comparison and Logical Operators

- Comparison and Logical operators are used to test for true or false.

Comparison Operators

- Comparison operators are used in logical statements to determine equality or difference between variables or values.

  Given that **x=5**, the table below explains the comparison operators:

Operator	**Description**	**Example**
==	is equal to	x==8 is false
===	is exactly equal to (value and type)	x===5 is true x==="5" is false
!=	is not equal	x!=8 is true
>	is greater than	x>8 is false
<	is less than	x<8 is true
>=	is greater than or equal to	x>=8 is false
<=	is less than or equal to	x<=8 is true

Logical Operators

- Logical operators are used to determine the logic between variables or values.

  Given that **x=6 and y=3**, the table below explains the logical operators:

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

Conditional Operator

- JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

```
variablename=(condition)?value1:value2
```

Example

```
greeting=(visitor=="PRES")?"Dear President ":"Dear ";
```

- If the variable **visitor** has the value of "PRES", then the variable **greeting** will be assigned the value "Dear President " else it will be assigned "Dear".

Control Structures

If Statement

- Use the if statement to execute some code only if a specified condition is true.

Syntax

```
if (condition)
  {
  code to be executed if condition is true
  }
```

If...else Statement

- Use the if. ..else statement to execute some code if a condition is true and another code if the condition is not true.

Syntax

```
if (condition)
  {
  code to be executed if condition is true
```

.

```
 }
else
 {
 code to be executed if condition is not true
 }
```

If...else if...else Statement

- Use the if....else if...else statement to select one of several blocks of code to be executed.

Syntax

```
if (condition1)
 {
 code to be executed if condition1 is true
 }
else if (condition2)
 {
 code to be executed if condition2 is true
 }
else
 {
 code to be executed if condition1 and condition2 are not true
 }
```

The JavaScript Switch Statement

- Use the switch statement to select one of many blocks of code to be executed.

Syntax

```
switch(n)
{
case 1:
  execute code block 1
  break;
case 2:
  execute code block 2
  break;
default:
  code to be executed if n is different from case 1 and 2
}
```

- This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed.

- Use **break** to prevent the code from running into the next case automatically.

JavaScript Popup Boxes

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

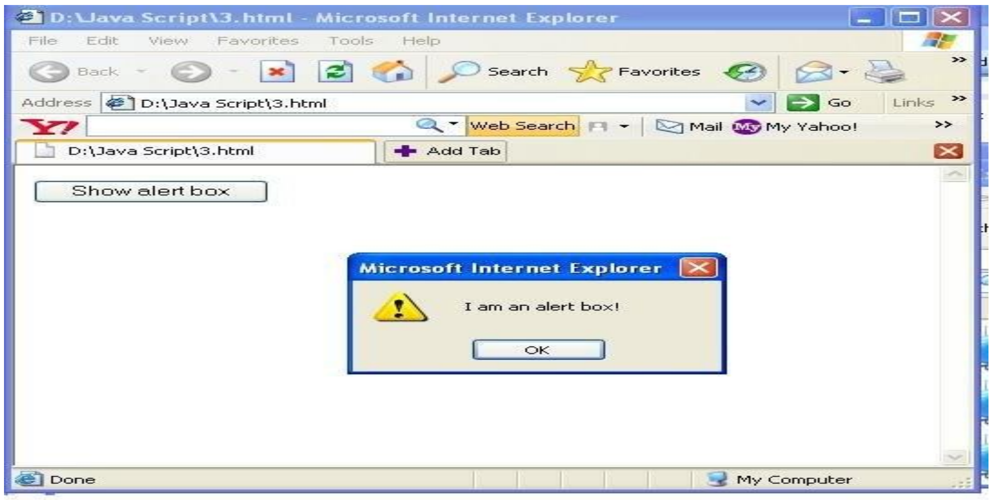
Syntax

```
alert("sometext");
```

Example

```
<html>
      <head>
            <script type="text/javascript">
                  function show_alert()
                  {
                        alert("I am an alert box!");
                  }
            </script>
     </head>
    <body>
                <input type="button" onclick="show_alert()" value="Show alert box" />
     </body>
</html>
```

Output:


D:\Java Script\3.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address D:\Java Script\3.html Go Links
Web Search Mail My Yahoo!
D:\Java Script\3.html Add Tab
Show alert box
Microsoft Internet Explorer
OK
Done My Computer

I am an alert box!

Confirm Box

- A confirm box is often used if you want the user to verify or accept something.
- When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
- If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax

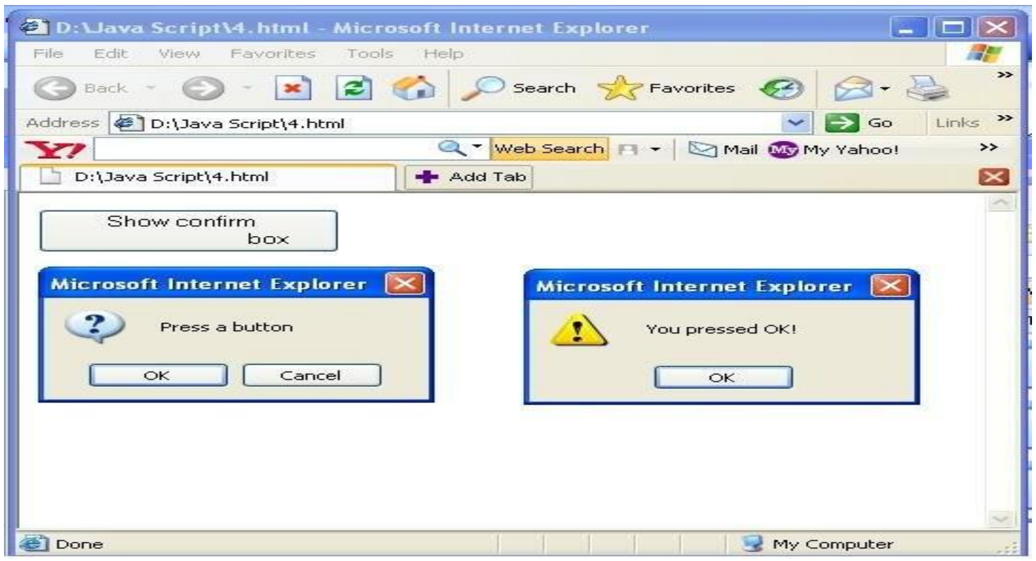
```
confirm("sometext");
```

Example

```
<html>
    <head>
            <script type="text/javascript">
                  function show_confirm()
                  {
                          var r=confirm("Press a button");
                          if (r==true)
                          {
                                  alert("You pressed OK!");
                          }
                          else
                          {
                                alert("You pressed Cancel!");
                           }
                  }
            </script>
   </head>
    <body>
              <input type="button" onclick="show_confirm()" value="Show confirm
                   box" />
    </body>
</html>
```

Output:

.


D:\Java Script\4.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address D:\Java Script\4.html
Go
Links
Web Search
Mail
My Yahoo!
D:\Java Script\4.html
Add Tab
Show confirm box
Microsoft Internet Explorer
Press a button
OK
Cancel
Microsoft Internet Explorer
You pressed OK!
OK
Done
My Computer

Prompt Box

- A prompt box is often used if you want the user to input a value before entering a page.
- When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
- If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

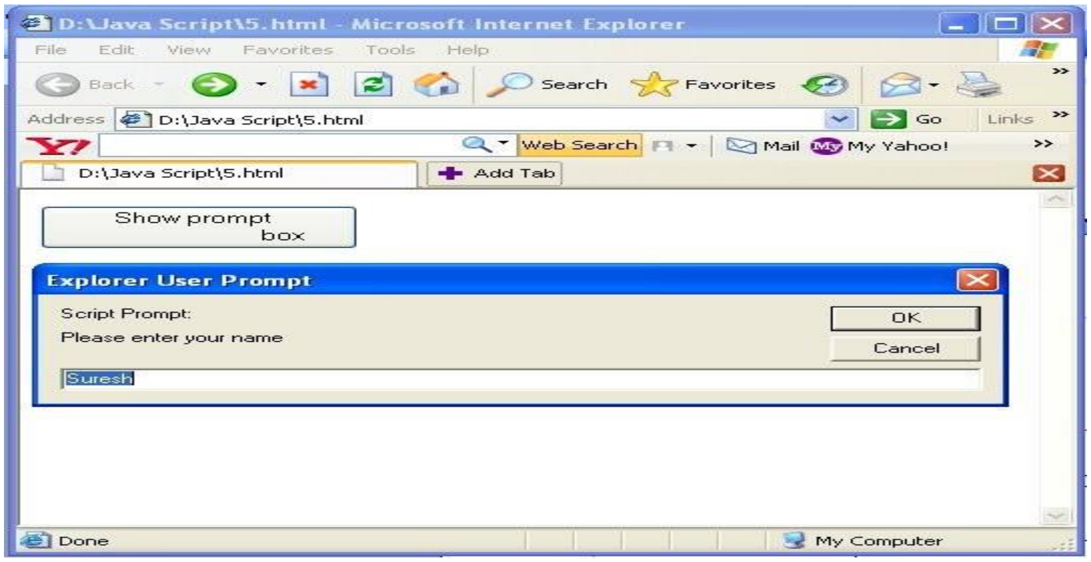
Syntax

```
prompt("sometext","defaultvalue");
```

Example

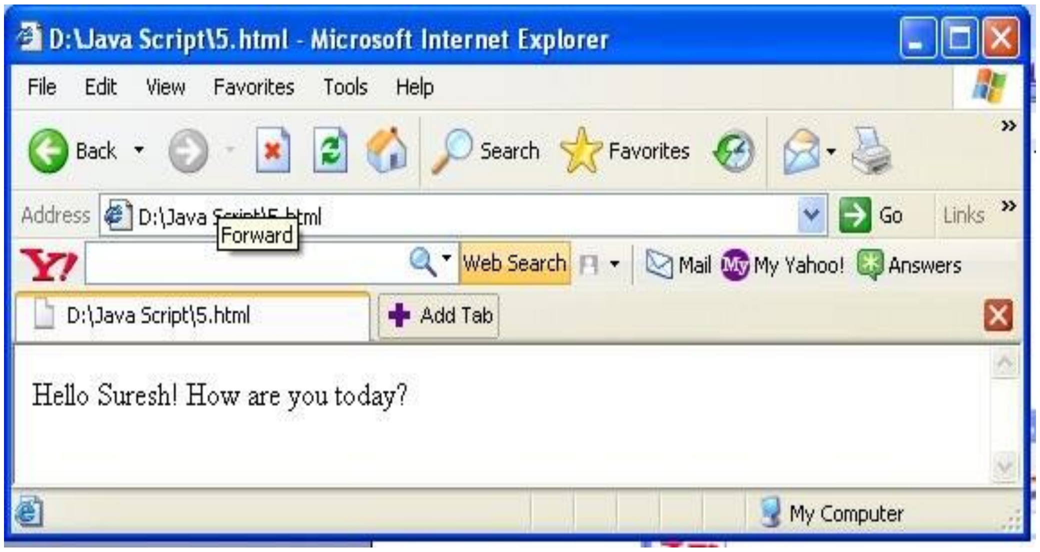
```
<html>
        <head>
                <script type="text/javascript">
                function show_prompt()
                {
                        var name=prompt("Please enter your name","Harry Potter");
                        if (name!=null && name!="")
                        {
                                document.write("Hello " + name + "! How are you
today?");
                         }
                  }
                </script>
        </head>
        <body>
                <input type="button" onclick="show_prompt()" value="Show prompt
                    box" />
        </body>
</html>
```

Output:


D:\Java Script\5.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address D:\Java Script\5.html
Show prompt box
Explorer User Prompt
OK
Cancel
Done
My Computer

Script Prompt:
Please enter your name

Suresh


D:\Java Script\5.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address D:\Java Script\5.html
Forward
My Computer

Hello Suresh! How are you today?

.

JavaScript Loops

- Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.
- In JavaScript, there are two different kind of loops:
  - **for** - loops through a block of code a specified number of times
  - **while** - loops through a block of code while a specified condition is true

The for Loop

- The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (var=startvalue;var<=endvalue;var=var+increment)
{
code to be executed
}
```

Example

- The example below defines a loop that starts with i=0. The loop will continue to run as long as **i** is less than, or equal to 5. **i** will increase by 1 each time the loop runs.

  Note: The increment parameter could also be negative, and the <= could be any comparing statement.

Example

```
<html>
        <body>
                <script type="text/javascript">
                        var i=0;
                        for (i=0;i<=5;i++)
                        {
                                document.write("The number is " + i);
                                document.write("<br />");
                        }
                </script>
        </body>
</html>
```

The while Loop

- The while loop loops through a block of code while a specified condition is true.

.

Syntax

```
while (var<=endvalue)
 {
 code to be executed
 }
```

Example

```
<html>
<body>
        <script type="text/javascript">
         var i=0;
        while (i<=5)
         {
                 document.write("The number is " + i);
                document.write("<br />");
                  i++;
          }
        </script>
    </body>
 </html>
```

The do...while Loop

- The do...while loop is a variant of the while loop. This loop will execute the block of code ONCE, and then it will repeat the loop as long as the specified condition is true.

Syntax

```
do
 {
 code to be executed
 }
while (var<=endvalue);
```

Example

```
<html>
    <body>
        <script type="text/javascript">
         var i=0;
         do
         {
                  document.write("The number is " + i);
                   document.write("<br />");
                    i++;
          }while (i<=5);
```

.

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

The break Statement

- The break statement will break the loop and continue executing the code that follows after the loop (if any).

Example

```
<html>
        <body>
              <script type="text/javascript">
                var i=0;
                for (i=0;i<=10;i++)
                {
                         if (i==3)
                         {
                                    break;
                          }
                         document.write("The number is " + i);
                          document.write("<br />");
                 }
              </script>
        </body>
</html>
```

The continue Statement

- The continue statement will break the current loop and continue with the next value.

Example

```
  <html>
     <body>
           <script type="text/javascript">
            var i=0
            for (i=0;i<=10;i++)
            {
                  if (i==3)
                    {
                            continue;
                     }
                     document.write("The number is " + i);
                      document.write("<br />");
              }</script></body>

    </html>
```

JavaScript for...in Statement

- The for...in statement loops through the elements of an array or through the properties of an object.

Syntax

```
for (variable in object)
  {
  code to be executed
  }
```

Note: The code in the body of the for...in loop is executed once for each element/property.

Note: The variable argument can be a named variable, an array element, or a property of an object.

Example

```
<html>
    <body>
            <script type="text/javascript">
             var x;
             var mycars = new Array();
             mycars[0] = "Saab";
             mycars[1] = "Volvo";
             mycars[2] = "BMW";

              for (x in mycars)
              {
                      document.write(mycars[x] + "<br />");
                }
            </script>
        </body>
</html>
```

JavaScript Functions

- A function contains code that will be executed by an event or by a call to the function.
- You may call a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file).
- Functions can be defined both in the <head> and in the <body> section of a document. However, to assure that a function is read/loaded by the browser before it is called, it could be wise to put functions in the <head> section.

How to Define a Function

Syntax

```
function functionname(var1,var2,...,varX)
{
some code
}
```

- The parameters var1, var2, etc. are variables or values passed into the function. The { and the } defines the start and end of the function.

  Note: A function with no parameters must include the parentheses () after the function name.

  Note: Do not forget about the importance of capitals in JavaScript! The word function must be written in lowercase letters, otherwise a JavaScript error occurs! Also note that you must call a function with the exact same capitals as in the function name.

Example

```
<html>
        <head>
                <script type="text/javascript">
                    function displaymessage()
                    {
                                alert("Hello World!");
                     }
                 </script>
          </head>
      <body>
            <form>
                   <input type="button" value="Click me!" onclick="displaymessage()" />
            </form>
         </body>
</html>
```

The return Statement

- The return statement is used to specify the value that is returned from the function.
- So, functions that are going to return a value must use the return statement.

Example

- The example below returns the product of two numbers (a and b):

```
  <html>
       <head>
              <script type="text/javascript">
                   function product(a,b)
                   {
                           return a*b;
                   }
             </script>
      </head>
      <body>
           <script type="text/javascript">
                       document.write(product(4,3));
           </script>
       </body>
</html>
```

String object

- The String object is used to manipulate a stored piece of text.
- String objects are created with new String().

String Object Properties

Property	Description
constructor	Returns the function that created the String object's prototype
length	Returns the length of a string

String Object Methods

Method	Description
charAt()	Returns the character at the specified index
charCodeAt()	Returns the Unicode of the character at the specified index
concat()	Joins two or more strings, and returns a copy of the joined strings
indexOf()	Returns the position of the first found occurrence of a specified value in a string
lastIndexOf()	Returns the position of the last found occurrence of a specified value in a string
match()	Searches for a match between a regular expression and a string, and returns the matches
replace()	Searches for a match between a substring (or regular expression) and a string, and replaces the matched substring with a new substring
search()	Searches for a match between a regular expression and a string, and returns the position of the match
slice()	Extracts a part of a string and returns a new string
substr()	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character
substring()	Extracts the characters from a string, between two specified indices
toLowerCase()	Converts a string to lowercase letters
toUpperCase()	Converts a string to uppercase letters

Example: To find the length of the string

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

Example: **The toLowerCase() and toUpperCase() methods**

```
<html>
        <body>
                <script type="text/javascript">
                        var txt="Hello World!";
                        document.write(txt.toLowerCase() + "<br />");
                        document.write(txt.toUpperCase());
                </script>
        </body>
</html>
```

Example: **The match() method --- How to search for a specified value within a string.**

```
<html>
        <body>
                <script type="text/javascript">
                        var str="Hello world!";
                        document.write(str.match("world")  +  "<br />");
                        document.write(str.match("World") + "<br />");
                        document.write(str.match("worlld") + "<br />");
                        document.write(str.match("world!"));
                </script>
        </body>
</html>
```

Example: **Replace characters in a string - replace()**

```
        <html>
                <body>
                        <script type="text/javascript">
                                var str="Visit Microsoft!";
                                document.write(str.replace("Microsoft","W3Schools"));
                        </script>
                </body>
</html>
```

Example: **The indexOf() method**

```
<html>
        <body>
                <script type="text/javascript">
                        var str="Hello world!";
                        document.write(str.indexOf("d") + "<br />");
                        document.write(str.indexOf("WORLD") + "<br />");
                        document.write(str.indexOf("world"));
                </script></body></html>
```

Math Object

- The Math object allows you to perform mathematical tasks.
- Math is not a constructor. All properties/methods of Math can be called by using Math as an object, without creating it.

Syntax

```
var x = Math.PI; // Returns PI
var y = Math.sqrt(16); // Returns the square root of 16
```

Math Object Properties

Property	Description
E	Returns Euler's number (approx. 2.718)
LN2	Returns the natural logarithm of 2 (approx. 0.693)
LN10	Returns the natural logarithm of 10 (approx. 2.302)
LOG2E	Returns the base-2 logarithm of E (approx. 1.442)
LOG10E	Returns the base-10 logarithm of E (approx. 0.434)
PI	Returns PI (approx. 3.14159)
SQRT1_2	Returns the square root of 1/2 (approx. 0.707)
SQRT2	Returns the square root of 2 (approx. 1.414)

Math Object Methods

Method	Description
abs(x)	Returns the absolute value of x
acos(x)	Returns the arccosine of x, in radians
asin(x)	Returns the arcsine of x, in radians
atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians
ceil(x)	Returns x, rounded upwards to the nearest integer
cos(x)	Returns the cosine of x (x is in radians)
exp(x)	Returns the value of E^x
floor(x)	Returns x, rounded downwards to the nearest integer
log(x)	Returns the natural logarithm (base E) of x
max(x,y,z,...,n)	Returns the number with the highest value
min(x,y,z,...,n)	Returns the number with the lowest value
pow(x,y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Rounds x to the nearest integer
sin(x)	Returns the sine of x (x is in radians)

sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent of an angle

Example: abs(x) Returns the absolute value of x

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

document.write(Math.abs(7.25) + "<br />");
document.write(Math.abs(-7.25) + "<br />");
document.write(Math.abs(null) + "<br />");
document.write(Math.abs("Hello") + "<br />");
document.write(Math.abs(7.25-10));

</script>
```

The output of the code above will be:

```
7.25
7.25
0
NaN
2.75
```

Example: acos(x)

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

        document.write(Math.acos(0.64) + "<br />");
        document.write(Math.acos(0) + "<br />");
        document.write(Math.acos(-1) + "<br />");
        document.write(Math.acos(1) + "<br />");
        document.write(Math.acos(2));

</script>
```

The output of the code above will be:

```
0.8762980611683406
1.5707963267948965
3.141592653589793
0
NaN
```

Example: floor()

- The floor() method rounds a number DOWNWARDS to the nearest integer, and returns the result.

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

        document.write(Math.floor(0.60) + "<br />");
        document.write(Math.floor(0.40) + "<br />");
        document.write(Math.floor(5) + "<br />");
        document.write(Math.floor(5.1) + "<br />");
        document.write(Math.floor(-5.1) + "<br />");
        document.write(Math.floor(-5.9));

</script>
```

The output of the code above will be:

```
0
0
5
5
-6
-6
```

JavaScript max() Method

- The max() method returns the number with the highest value.

Syntax

```
Math.max(x,y,z,...,n)
```

Parameter	**Description**
x,y,z,...,n	Optional. One or more numbers. If no arguments are given, the result is -Infinity

Example:

```
<script type="text/javascript">
        document.write(Math.max(5,10) + "<br />");
        document.write(Math.max(0,150,30,20,38) + "<br />");
</script>
```

Arrays

- An array is an ordered set of data elements which can be accessed through a single variable name.
- We can access the data elements either sequentially by reading from the start of the array, or by their index.
- The index is the position of the element in the array (first element begins at position 0 and last at arraylength -1)

Creating Arrays: There are three different ways to create arrays.

1. The easiest way is simply to create a variable and pass it some elements in array format.

   var days=[“Monday”,”Tuesday”,”Wednesday”];

2. The second approach is to create an array object using the key word **new** and a set of elements to store.

   var days=new Array(“Monday”,”Tuesday”,”Wednesday”);

3. Finally, an empty array object which has a space for a number of elements can be created.

   var days=new Array(3);

   Note: JavaScript can hold mixed data types as following examples show:

   var days=[“Monday”,”Tuesday”,23, 45.78, ”Wednesday”];

Array Object Properties

Property	**Description**
constructor	Returns the function that created the Array object's prototype
length	Sets or returns the number of elements in an array

Array Object Methods

Method	**Description**
concat()	Joins two or more arrays, and returns a copy of the joined arrays
join()	Joins all elements of an array into a string
pop()	Removes the last element of an array, and returns that element

push()	Adds new elements to the end of an array, and returns the new length
reverse()	Reverses the order of the elements in an array
shift()	Removes the first element of an array, and returns that element
slice()	Selects a part of an array, and returns the new array
sort()	Sorts the elements of an array
splice()	Adds/Removes elements from an array
toString()	Converts an array to a string, and returns the result
unshift()	Adds new elements to the beginning of an array, and returns the new length
valueOf()	Returns the primitive value of an array

Example: Creating an array

```
        <html>
                <body>
                        <script type="text/javascript">
                                var mycars = new Array();
                                mycars[0] = "Saab";
                                mycars[1] = "Volvo";
                                mycars[2] = "BMW";
                                for (i=0;i<mycars.length;i++)
                                {
                                        document.write(mycars[i] + "<br />");
                                }
                         </script>
                </body>
</html>
```

Example: for..in

```
        <html>
                <body>
                        <script type="text/javascript">
                        var x;
                        var mycars = new Array();
                        mycars[0] = "Saab";
                        mycars[1] = "Volvo";
                        mycars[2] = "BMW";
                        for (x in mycars)
                        {
                                document.write(mycars[x] + "<br />");
                        }
                        </script></body></html>
```

<u>Example:</u>concat() mathod

```
<script type="text/javascript">
        var parents = ["Jani", "Tove"];
        var children = ["Cecilie", "Lone"];
        var family = parents.concat(children);
        document.write(family);
</script>
```

<u>Example:</u> join()

```
<script type="text/javascript">
         var fruits = ["Banana", "Orange", "Apple", "Mango"];
        document.write(fruits.join() + "<br />");
        document.write(fruits.join("+") + "<br />");
        document.write(fruits.join(" and "));
</script>
```

<u>Example:</u> pop() method

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

        var fruits = ["Banana", "Orange", "Apple", "Mango"];
        document.write(fruits.pop() + "<br />");
        document.write(fruits + "<br />");
        document.write(fruits.pop() + "<br />");
        document.write(fruits);

</script>
```

<u>Example:</u>slice() method

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

var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write(fruits.slice(0,1) + "<br />");
document.write(fruits.slice(1) + "<br />");
document.write(fruits.slice(-2) + "<br />");
document.write(fruits);

</script>
```

Example: push()

- The push() method adds new elements to the end of an array, and returns the new length.

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

        var fruits = ["Banana", "Orange", "Apple", "Mango"];
       document.write(fruits.push("Kiwi") + "<br />");
       document.write(fruits.push("Lemon","Pineapple") + "<br />");
       document.write(fruits);

</script>
```

Example: valueOf()

- The valueOf() method returns the primitive value of an array.

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

var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write(fruits.valueOf());

</script>
```

Output: Banana,Orange,Apple,Mango

Example:shift()

- The shift() method removes the first element of an array, and returns that element.

```
<script type="text/javascript">
        var fruits = ["Banana", "Orange", "Apple", "Mango"];
       document.write(fruits.shift() + "<br />");
       document.write(fruits + "<br />");
       document.write(fruits.shift() + "<br />");
       document.write(fruits);
</script>
```

Output:

```
Banana
Orange,Apple,Mango
Orange
Apple,Mango
```

Date Object

- The Date object is used to work with dates and times.
- Date objects are created with new Date().
- There are four ways of instantiating a date:
    1. var d = new Date(); - creates an empty date object
    2. var d = new Date(milliseconds); - constructs a new date object upon the number of milliseconds which have elapsed since 00:00:00 on 01/01/1970.
    3. var d = new Date(dateString); - create a date object based upon the contents of a text string. The string must be in the format which is created by the Date.parse() function.
    4. var d = new Date(year, month, day, hours, minutes, seconds, milliseconds); - create a new date object based upon numerical values for the year, month and day. January is represented by the integer value 0, December by 11.
- Parse(String) – returns the number of milliseconds since midnight on 01/01/1970 which the string represents. The string must be the following format:

  Mon, 9 April 2001 14:02:35

Date Object Methods

Method	**Description**
getDate()	Returns the day of the month (from 1-31)
getDay()	Returns the day of the week (from 0-6)
getFullYear()	Returns the year (four digits)
getHours()	Returns the hour (from 0-23)
getMilliseconds()	Returns the milliseconds (from 0-999)
getMinutes()	Returns the minutes (from 0-59)
getMonth()	Returns the month (from 0-11)
getSeconds()	Returns the seconds (from 0-59)
getTime()	Returns the number of milliseconds since midnight Jan 1, 1970
setDate()	Sets the day of the month (from 1-31)
setFullYear()	Sets the year (four digits)
setHours()	Sets the hour (from 0-23)
setMilliseconds()	Sets the milliseconds (from 0-999)
setMinutes()	Set the minutes (from 0-59)
setMonth()	Sets the month (from 0-11)
setSeconds()	Sets the seconds (from 0-59)
setTime()	Sets a date and time by adding or subtracting a specified number of milliseconds to/from midnight January 1, 1970

setYear()	Deprecated. Use the setFullYear() method instead
toGMTString()	Deprecated. Use the toUTCString() method instead
toString()	Converts a Date object to a string

Example:

```
<html>
        <body>
                <script type="text/javascript">
                        var d=new Date();
                document.write(d);
                </script>
        </body>
</html>
```

Output: Sat Dec 25 20:08:07 UTC+0530 2010

Example: getTime()

- Returns the number of milliseconds since midnight Jan 1, 1970

```
<html>
        <body>

        <script type="text/javascript">
                var d=new Date();
                document.write(d.getTime() + " milliseconds since 1970/01/01");
        </script>

        </body>
</html>
```

Output: 1293288029234 milliseconds since 1970/01/01

Example: setFullYear()

```
     <html>
        <body>

                <script type="text/javascript">

                        var d = new Date();
                        d.setFullYear(1992,10,3);
                        document.write(d);
                </script>
        </body>
        </html>
```

Output: Tue Nov 3 20:12:01 UTC+0530 1992

Example:

```
<html>
        <body onload="datefun()">
                <script type="text/javascript">
                        function datefun()
                        {
                                var today=new Date();
                                var yesterday=new Date();
                                var  diff=today.getDate()-1;
                                yesterday.setDate(diff);
                                document.write("<h3>The Date is:"+today+"</h3>");
                                document.write("<h3>The Yesterday Date
is:"+yesterday+"</h3>");
                        }
                </script>
        </body>
</html>
```

Output:

```
The Date is:Sat Dec 25 20:22:43 UTC+0530 2010
The Yesterday Date is:Fri Dec 24 20:22:43 UTC+0530 2010
```

JavaScript - Catching Errors

- When browsing Web pages on the internet, we all have seen a JavaScript alert box telling us there is a runtime error and asking "Do you wish to debug?".
- Error message like this may be useful for developers but not for users. When users see errors, they often leave the Web page.

The try...catch Statement

- The try...catch statement allows you to test a block of code for errors. The try block contains the code to be run, and the catch block contains the code to be executed if an error occurs.

Syntax

```
try
  {
        //Run some code here
  }
catch(err)
  {
        //Handle errors here
  }
```

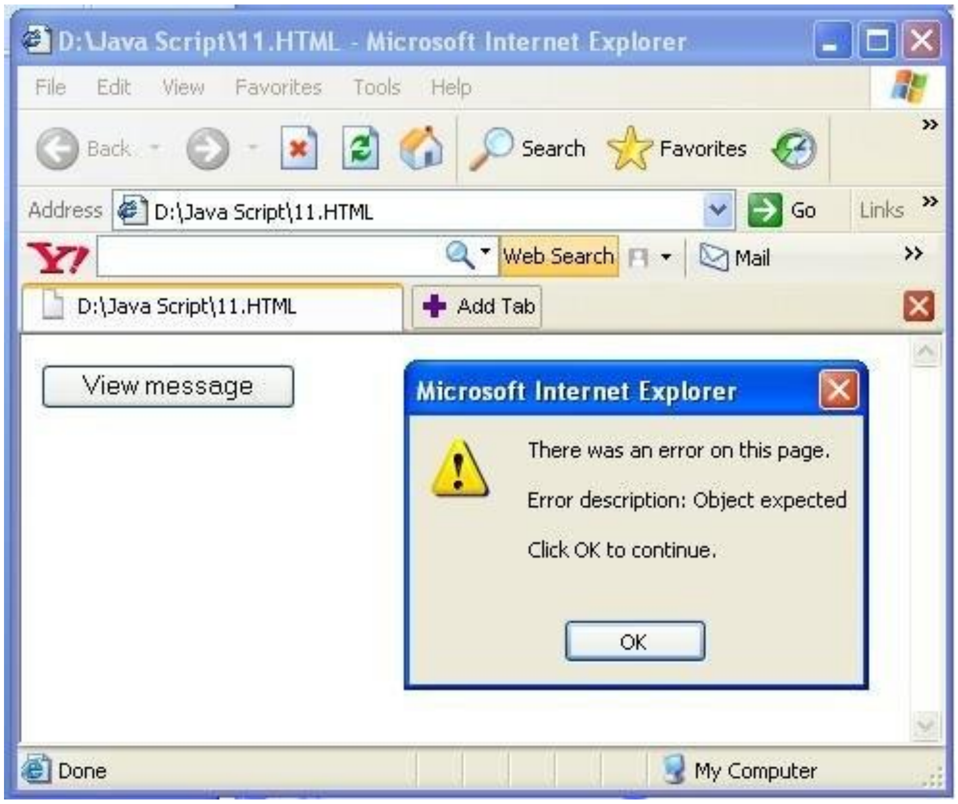
Note: note that try...catch is written in lowercase letters. Using uppercase letters will generate a JavaScript error!

Example

- The example below is supposed to alert "Welcome guest!" when the button is clicked. However, there's a typo in the message() function. alert() is misspelled as adddlert(). A JavaScript error occurs. The catch block catches the error and executes a custom code to handle it. The code displays a custom error message informing the user what happened.

```
<html>
        <head>
                <script type="text/javascript">
                        var txt="";
                        function message()
                        {
                                try
                                {
                                        adddlert("Welcome guest!");
                                  }
                                catch(err)
                                {
                                        txt="There was an error on this page.\n\n";
                        txt+="Error description: " + err.description + "\n\n";
                        txt+="Click OK to continue.\n\n";
                        alert(txt);
                    }
                }
                </script>
        </head>
        <body>
                <input type="button" value="View message" onclick="message()" />
        </body>
</html>
```

Output:


D:\Java Script\11.HTML - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
D:\Java Script\11.HTML
Go
Links
Web Search
Mail
Add Tab
View message
Microsoft Internet Explorer
OK
Done
My Computer

```
There was an error on this page.

Error description: Object expected

Click OK to continue.
```

The Throw Statement

- The throw statement allows you to create an exception. If you use this statement together with the try...catch statement, you can control program flow and generate accurate error messages.

.

Syntax:

```
throw(exception)
```

- The exception can be a string, integer, Boolean or an object.
- Note that throw is written in lowercase letters. Using uppercase letters will generate a JavaScript error!

Example

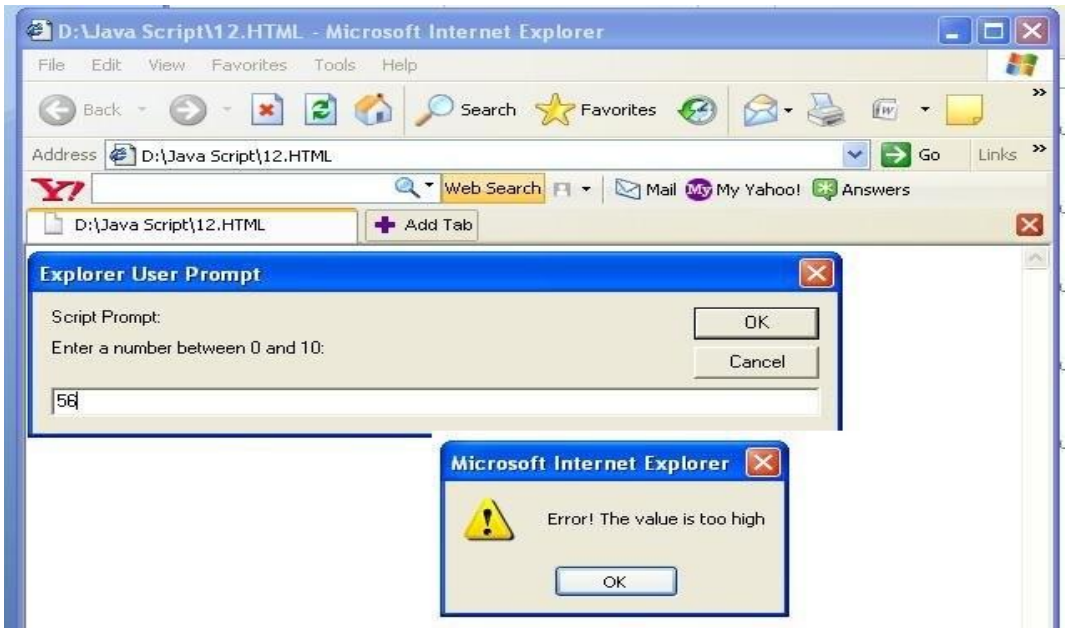
The example below determines the value of a variable called x. If the value of x is higher than 10, lower than 0, or not a number, we are going to throw an error. The error is then caught by the catch argument and the proper error message is displayed:

```
<html>
        <body>
                <script type="text/javascript">
                var x=prompt("Enter a number between 0 and 10:","");
                try
                {
                         if(x>10)
                         {
                                 throw "Err1";
                        }
                        else if(x<0)
                         {
                                  throw "Err2";
                         }
                         else if(isNaN(x))
                         {
                                 throw "Err3";
                        }
                }
                catch(er)
                {
                        if(er=="Err1")
                        {
                                alert("Error! The value is too high");
                         }
                        if(er=="Err2")
                        {
                                alert("Error! The value is too low");
                        }
                        if(er=="Err3")
                        {
```

.

```
                                        alert("Error! The value is not a number");
                              }
                     }
                    </script>
          </body>
</html>
```

Output


D:\Java Script\12.HTML - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address D:\Java Script\12.HTML Go Links
Web Search Mail My Yahoo! Answers
D:\Java Script\12.HTML Add Tab

Explorer User Prompt

Script Prompt:
Enter a number between 0 and 10:

56

OK Cancel

Microsoft Internet Explorer

Error! The value is too high

OK

Built in Objects

The Document Object

- A document is a Web page that is being either displayed or created. The document has a number of properties that can be accessed by JavaScript programs and used to manipulate the content of the page.
- Some of these properties can be used to create HTML pages from with in JavaScript while others may be used to change the operation of the current page.

Methods:

write/writeln : HTML pages can be created on the fly using JavaScript.

bgcolor, fgcolor: These are the same properties that can be set in the <BODY> tag. The difference here is that the values can be set from within a JavaScript.

The methods accept either hexadecimal values or common names for colors.

Example:

```
document.bgcolor=”coral”;
document.fgcolor=”coral”;
```

anchors/links: The anchors property is an array of these names in the order in which they appear in the HTML document.

Example: document.anchors[0];

forms: This is an array in the order of the document. This one contains all of the HTML forms. By combining this array with the individual form objects each form item can be accessed.

close(): The document isn’t completely written until the close() method has been called. If you don’t use this method then the browser will keep waiting for more data even if there is none.

The Form Object

- Two aspects of the form can be manipulated through JavaScript.
  1. First, most commonly and probably most usefully, the data that is entered onto your form can be checked at submission.
  2. Second you can actually build forms through JavaScript.
- The elements of the form are held in an array.

Example:

```
<html>

<head>
```

```
        <script type="text/javascript">
                function validate()
                {
                        var method=document.forms[0].method;
                        var action=document.forms[0].action;
                        var value=document.forms[0].elements[0].value;
                        if(value=="suresh")
                        {
                                alert("Hi "+value);
                        }
                        else
                        {
                                document.forms[0].reset();
                        }

                }
         </script>
        </head>
        <body>
                <form method="suresh@gmail.com" method="post">
                        <input type="text" name="User" size="10">
                        <input type="submit" value="Press Me!" onClick="validate()">
                </form>
        </body>
</html>
```

<u>**Events on the Form Object:**</u>

1. **onClick=”method”**

   This can be applied to all form elements. The event is triggered when the user clicks on that element.

2. **onSubmit=”method”**

   This even can only be triggered by the form itself and occurs when a form is submitted.

3. **onReset=”method”**

   Like the previous one this is a form-only event and is triggered when a form is reset by the user.

.

Events

- By using JavaScript, we have the ability to create dynamic web pages. Events are actions that can be detected by JavaScript.
- Every element on a web page has certain events which can trigger a JavaScript. For example, we can use the onClick event of a button element to indicate that a function will run when a user clicks on the button. We define the events in the HTML tags.
- Examples of events:

  A mouse click
  A web page or an image loading
  Mousing over a hot spot on the web page
  Selecting an input field in an HTML form
  Submitting an HTML form
  A keystroke

  Note: Events are normally used in combination with functions, and the function will not be executed before the event occurs!

<body> and <frameset> Events

The two attributes below can only be used in <body> or <frameset>:

Attribute	Description
onLoad	Script to be run when a document load
onUnload	Script to be run when a document unload

Form Events

The attributes below can be used in form elements:

Attribute	Description
onBlur	Script to be run when an element loses focus
onChange	Script to be run when an element change
onFocus	Script to be run when an element gets focus
onReset	Script to be run when a form is reset
onSelect	Script to be run when an element is selected
onSubmit	Script to be run when a form is submitted

.

Image Events

The attribute below can be used with the img element:

Attribute	Description
onAbort	Script to be run when loading of an image is interrupted

Keyboard Events

Attribute	Description
onKeyDown	Script to be run when a key is pressed
onKeyPress	Script to be run when a key is pressed and released
OnKeyUp	Script to be run when a key is released

Mouse Events

Attribute	Description
onClick	Script to be run on a mouse click
onDblClick	Script to be run on a mouse double-click
onMouseDown	Script to be run when mouse button is pressed
onMouseMove	Script to be run when mouse pointer moves
onMouseOut	Script to be run when mouse pointer moves out of an element
onMouseOver	Script to be run when mouse pointer moves over an element
onMouseUp	Script to be run when mouse button is released

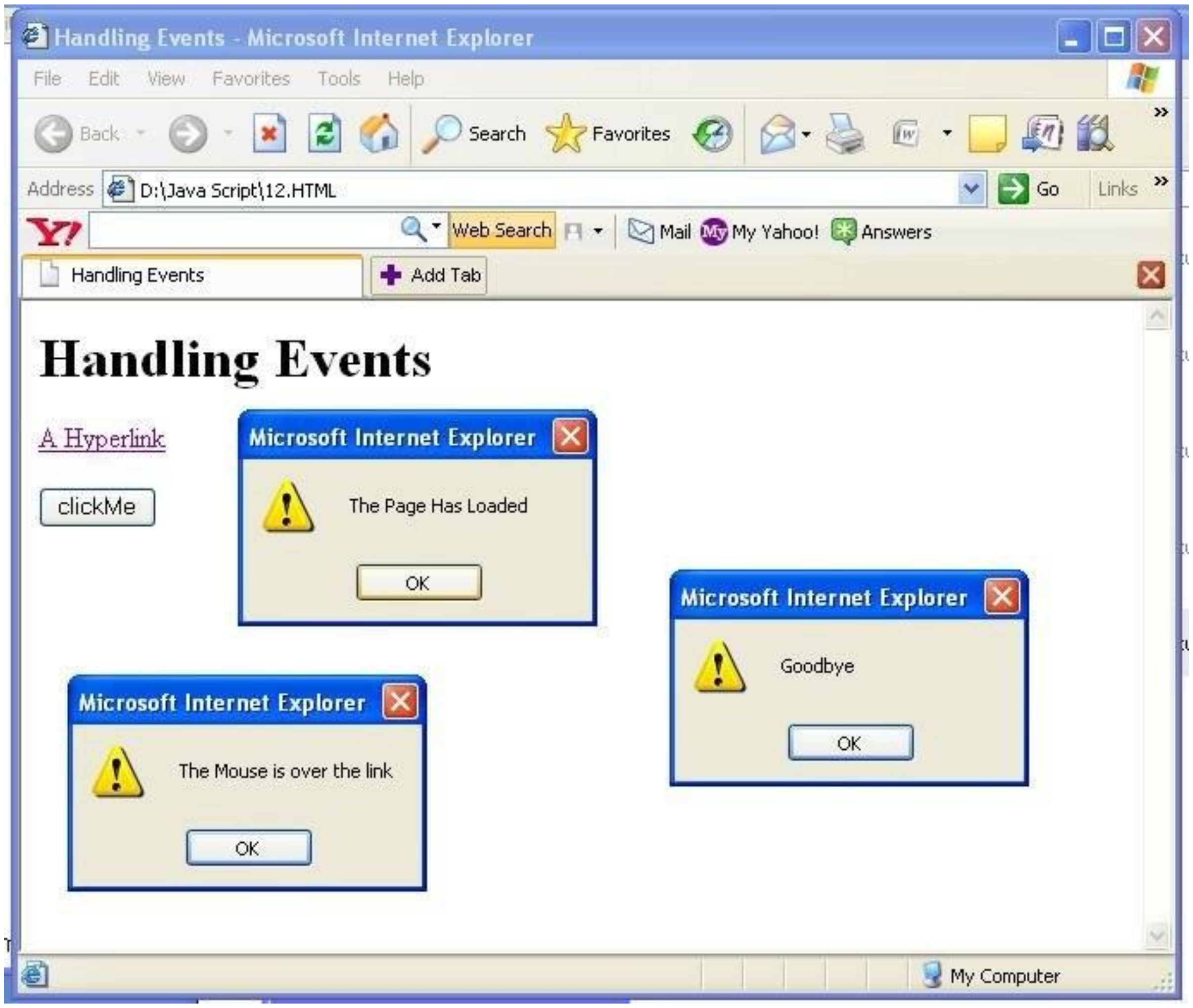
JavaScript Objects and Event Handlers

Object	Event Handlers
Window	onload onunload onblur onfocus
Link	onclick onmouseout onmouseover onmouseout
Image	onabort onerror onload
text textarea password	onblur onchange onfocus onselect

.

button reset submit radio checkbox	onclick
select	onblur onchange onfocus

Example:


Handling Events - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address D:\Java Script\12.HTML Go Links
Web Search Mail My Yahoo! Answers
Handling Events Add Tab
My Computer

Handling Events

A Hyperlink

clickMe

Microsoft Internet Explorer
The Page Has Loaded
OK

Microsoft Internet Explorer
Goodbye
OK

Microsoft Internet Explorer
The Mouse is over the link
OK

.