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!

For a complete reference of the events recognized by JavaScript, go to our complete JavaScript reference.

---

onLoad and onUnload

The onLoad and onUnload events are triggered when the user enters or leaves the page.

The onLoad event is often used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

Both the onLoad and onUnload events are also often used to deal with cookies that should be set when a user enters or leaves a page. For example, you could have a popup asking for the user's name upon his first arrival to your page. The name is then stored in a cookie. Next time the visitor arrives at your page, you could have another popup saying something like: "Welcome John Doe!".

---

onFocus, onBlur and onChange

The onFocus, onBlur and onChange events are often used in combination with validation of form fields.

Below is an example of how to use the onChange event. The checkEmail() function will be called whenever the user changes the content of the field:

```
<input type="text" size="30" id="email" onchange="checkEmail()">
```

---

onSubmit

The onSubmit event is used to validate ALL form fields before submitting it.

Below is an example of how to use the onSubmit event. The checkForm() function will be called when the user clicks the submit button in the form. If the field values are not accepted, the submit should be cancelled. The function checkForm() returns either true or false. If it returns true the form will be submitted, otherwise the submit will be cancelled:

```
<form method="post" action="xxx.htm" onsubmit="return checkForm()">
```

---

onMouseOver and onMouseOut

onMouseOver and onMouseOut are often used to create "animated" buttons.

Below is an example of an onMouseOver event. An alert box appears when an onMouseOver event is detected:

```
<a href="http://www.w3schools.com" onmouseover="alert('An onMouseOver
event');return false"><img src="w3s.gif" alt="W3Schools" /></a>
```

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.

This chapter will teach you how to catch and handle JavaScript error messages, so you don't lose your audience.

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

Examples

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:

Example

```
<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>
```

The next example uses a confirm box to display a custom message telling users they can click OK to continue viewing the page or click Cancel to go to the homepage. If the confirm method returns false, the user clicked Cancel, and the code redirects the user. If the confirm method returns true, the code does nothing:

Example

```
<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+="Click OK to continue viewing this page,\n";
  txt+="or Cancel to return to the home page.\n\n";
  if(!confirm(txt))
    {
    document.location.href="http://www.w3schools.com/";
    }
  }
}
</script>
</head>

<body>
<input type="button" value="View message" onclick="message()" />
</body>

</html>
```

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:

Example

```
<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>
```

In JavaScript you can add special characters to a text string by using the backslash sign.

Insert Special Characters

The backslash (\) is used to insert apostrophes, new lines, quotes, and other special characters into a text string.

Look at the following JavaScript code:

```
var txt="We are the so-called "Vikings" from the north.";
document.write(txt);
```

In JavaScript, a string is started and stopped with either single or double quotes. This means that the string above will be chopped to: We are the so-called

To solve this problem, you must place a backslash (\) before each double quote in "Viking". This turns each double quote into a string literal:

```
var txt="We are the so-called \"Vikings\" from the north.";
document.write(txt);
```

JavaScript will now output the proper text string: We are the so-called "Vikings" from the north.

Here is another example:

```
document.write ("You \& I are singing!");
```

The example above will produce the following output:

```
You & I are singing!
```

The table below lists other special characters that can be added to a text string with the backslash sign:

Code	Outputs
\'	single quote
\"	double quote
\&	ampersand
\\	backslash
\n	new line
\r	carriage return
\t	tab
\b	backspace
\f	form feed

JavaScript is Case Sensitive

A function named "myfunction" is not the same as "myFunction" and a variable named "myVar" is not the same as "myvar".

JavaScript is case sensitive - therefore watch your capitalization closely when you create or call variables, objects and functions.

White Space

JavaScript ignores extra spaces. You can add white space to your script to make it more readable. The following lines are equivalent:

```
name="Hege";
name = "Hege";
```

Break up a Code Line

You can break up a code line **within a text string** with a backslash. The example below will be displayed properly:

```
document.write("Hello \
World!");
```

However, you cannot break up a code line like this:

```
document.write \
("Hello World!");
```

JavaScript is an Object Oriented Programming (OOP) language.

An OOP language allows you to define your own objects and make your own variable types.

Object Oriented Programming

JavaScript is an Object Oriented Programming (OOP) language. An OOP language allows you to define your own objects and make your own variable types.

However, creating your own objects will be explained later, in the Advanced JavaScript section. We will start by looking at the built-in JavaScript objects, and how they are used. The next pages will explain each built-in JavaScript object in detail.

Note that an object is just a special kind of data. An object has properties and methods.

Properties

Properties are the values associated with an object.

In the following example we are using the length property of the String object to return the number of characters in a string:

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

The output of the code above will be:

```
12
```

Methods

Methods are the actions that can be performed on objects.

In the following example we are using the toUpperCase() method of the String object to display a text in uppercase letters:

```
<script type="text/javascript">
var str="Hello world!";
document.write(str.toUpperCase());
</script>
```

The output of the code above will be:

```
HELLO WORLD!
```

Number Object

The Number object is an object wrapper for primitive numeric values.

Number objects are created with new Number().

Syntax

```
var num = new Number(value);
```

Note: If the value parameter cannot be converted into a number, it returns NaN (Not-a-Number).

Number Object Properties

Property	Description
constructor	Returns the function that created the Number object's prototype
MAX_VALUE	Returns the largest number possible in JavaScript
MIN_VALUE	Returns the smallest number possible in JavaScript
NEGATIVE_INFINITY	Represents negative infinity (returned on overflow)
POSITIVE_INFINITY	Represents infinity (returned on overflow)
prototype	Allows you to add properties and methods to an object

Number Object Methods

Method	Description
toExponential(x)	Converts a number into an exponential notation
toFixed(x)	Formats a number with x numbers of digits after the decimal point
toPrecision(x)	Formats a number to x length
toString()	Converts a Number object to a string
valueOf()	Returns the primitive value of a Number object

The JavaScript global properties and functions can be used with all the built-in JavaScript objects.

JavaScript Global Properties

Property	Description
Infinity	A numeric value that represents positive/negative infinity
NaN	"Not-a-Number" value
undefined	Indicates that a variable has not been assigned a value

JavaScript Global Functions

Function	Description
decodeURI()	Decodes a URI
decodeURIComponent()	Decodes a URI component
encodeURI()	Encodes a URI
encodeURIComponent()	Encodes a URI component
escape()	Encodes a string
eval()	Evaluates a string and executes it as if it was script code
isFinite()	Determines whether a value is a finite, legal number
isNaN()	Determines whether a value is an illegal number
Number()	Converts an object's value to a number
parseFloat()	Parses a string and returns a floating point number
parseInt()	Parses a string and returns an integer
String()	Converts an object's value to a string
unescape()	Decodes an encoded string

Return the length of a string
How to return the length of a string.

Style strings
How to style strings.

The toLowerCase() and toUpperCase() methods
How to convert a string to lowercase or uppercase letters.

The match() method
How to search for a specified value within a string.

Replace characters in a string - replace()
How to replace a specified value with another value in a string.

The indexOf() method
How to return the position of the first found occurrence of a specified value in a string.

Complete String Object Reference

For a complete reference of all the properties and methods that can be used with the String object, go to our complete String object reference.

The reference contains a brief description and examples of use for each property and method!

String object

The String object is used to manipulate a stored piece of text.

Examples of use:

The following example uses the length property of the String object to find the length of a string:

```
var txt="Hello world!";
document.write(txt.length);
```

The code above will result in the following output:

```
12
```

The following example uses the toUpperCase() method of the String object to convert a string to uppercase letters:

```
var txt="Hello world!";
document.write(txt.toUpperCase());
```

The code above will result in the following output:

```
HELLO WORLD!
```

String Object

The String object is used to manipulate a stored piece of text.

String objects are created with new String().

Syntax

```
var txt = new String(string);

or more simply:

var txt = string;
```

For a tutorial about the String object, read our JavaScript String Object tutorial.

String Object Properties

Property	Description
constructor	Returns the function that created the String object's prototype
length	Returns the length of a string
prototype	Allows you to add properties and methods to an object

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
fromCharCode()	Converts Unicode values to characters
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
split()	Splits a string into an array of substrings
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
valueOf()	Returns the primitive value of a String object

String HTML Wrapper Methods

The HTML wrapper methods return the string wrapped inside the appropriate HTML tag.

Method	Description
anchor()	Creates an anchor
big()	Displays a string using a big font
blink()	Displays a blinking string
bold()	Displays a string in bold
fixed()	Displays a string using a fixed-pitch font
fontcolor()	Displays a string using a specified color
fontsize()	Displays a string using a specified size
italics()	Displays a string in italic
link()	Displays a string as a hyperlink
small()	Displays a string using a small font
strike()	Displays a string with a strikethrough
sub()	Displays a string as subscript text
sup()	Displays a string as superscript text

Return today's date and time
How to use the Date() method to get today's date.

getTime()
Use getTime() to calculate the years since 1970.

setFullYear()
How to use setFullYear() to set a specific date.

toUTCString()
How to use toUTCString() to convert today's date (according to UTC) to a string.

getDay()
Use getDay() and an array to write a weekday, and not just a number.

Display a clock
How to display a clock on your web page.

Complete Date Object Reference

For a complete reference of all the properties and methods that can be used with the Date object, go to our complete Date object reference.

The reference contains a brief description and examples of use for each property and method!

Create a Date Object

The Date object is used to work with dates and times.

Date objects are created with the Date() constructor.

There are four ways of instantiating a date:

```
new Date() // current date and time
new Date(milliseconds) //milliseconds since 1970/01/01
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds, milliseconds)
```

Most parameters above are optional. Not specifying, causes 0 to be passed in.

Once a Date object is created, a number of methods allow you to operate on it. Most methods allow you to get and set the year, month, day, hour, minute, second, and milliseconds of the object, using either local time or UTC (universal, or GMT) time.

All dates are calculated in milliseconds from 01 January, 1970 00:00:00 Universal Time (UTC) with a day containing 86,400,000 milliseconds.

Some examples of instantiating a date:

```
today = new Date()
d1 = new Date("October 13, 1975 11:13:00")
d2 = new Date(79,5,24)
d3 = new Date(79,5,24,11,33,0)
```

Set Dates

We can easily manipulate the date by using the methods available for the Date object.

In the example below we set a Date object to a specific date (14th January 2010):

```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
```

And in the following example we set a Date object to be 5 days into the future:

```
var myDate=new Date();
myDate.setDate(myDate.getDate()+5);
```

Note: If adding five days to a date shifts the month or year, the changes are handled automatically by the Date object itself!

Compare Two Dates

The Date object is also used to compare two dates.

The following example compares today's date with the 14th January 2010:

```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
var today = new Date();

if (myDate>today)
  {
  alert("Today is before 14th January 2010");
  }
else
  {
  alert("Today is after 14th January 2010");
  }
```

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:

```
var d = new Date();
var d = new Date(milliseconds);
var d = new Date(dateString);
var d = new Date(year, month, day, hours, minutes, seconds, milliseconds);
```

Date Object Properties

Property	Description
constructor	Returns the function that created the Date object's prototype
prototype	Allows you to add properties and methods to an object

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
getTimezoneOffset()	Returns the time difference between GMT and local time, in minutes
getUTCDate()	Returns the day of the month, according to universal time (from 1-31)
getUTCDay()	Returns the day of the week, according to universal time (from 0-6)
getUTCFullYear()	Returns the year, according to universal time (four digits)
getUTCHours()	Returns the hour, according to universal time (from 0-23)
getUTCMilliseconds()	Returns the milliseconds, according to universal time (from 0-999)
getUTCMinutes()	Returns the minutes, according to universal time (from 0-59)
getUTCMonth()	Returns the month, according to universal time (from 0-11)
getUTCSeconds()	Returns the seconds, according to universal time (from 0-59)
getYear()	Deprecated. Use the getFullYear() method instead
parse()	Parses a date string and returns the number of milliseconds since midnight of January 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
setUTCDate()	Sets the day of the month, according to universal time (from 1-31)
setUTCFullYear()	Sets the year, according to universal time (four digits)
setUTCHours()	Sets the hour, according to universal time (from 0-23)
setUTCMilliseconds()	Sets the milliseconds, according to universal time (from 0-999)
setUTCMinutes()	Set the minutes, according to universal time (from 0-59)
setUTCMonth()	Sets the month, according to universal time (from 0-11)
setUTCSeconds()	Set the seconds, according to universal time (from 0-59)
setYear()	Deprecated. Use the setFullYear() method instead
toDateString()	Converts the date portion of a Date object into a readable string
toGMTString()	Deprecated. Use the toUTCString() method instead