JavaScript

What You Can do with **JavaScript**

- **JavaScript Can Create HTML Elements .**
- **JavaScript Can Change HTML Content.**
- **JavaScript Can Change HTML Attributes.**
- **JavaScript Can Change HTML Styles (CSS).**
- **JavaScript Can Validate Data.**

JavaScript - Where To place

- **The <script> Tag**
- In HTML, JavaScript code must be inserted between <script> and </script> tags.
- **JavaScript in <head> or <body>**
- You can place any number of scripts in an HTML document.
- Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

- **External JavaScript**
- Scripts can also be placed in external files.
- External scripts are practical when the same code is used in many different web pages.
- JavaScript files have the **file extension .js**.
- To use an external script, put the name of the script file in the src (source) attribute of the <script> tag:

JavaScript Output

JavaScript Display Possibilities

- Writing into an alert box, using **window.alert()**.
- Writing into the HTML output using **document.write()**.
- Writing into an HTML element, using **innerHTML**.
- Writing into the browser console, using **console.log()**

JavaScript Data Types

- String,
- Number,
- Boolean,
- Null
- Undefined.

Java Script POP up Boxes

- Alert Box
- Confirm Box
- Prompt Box

JavaScript Variables

- In a programming language, **variables** are used to **store** data values.
- JavaScript uses the **var** keyword to **define** variables.
- An **equal sign** is used to **assign values** to variables.
- In this example, x is defined as a variable. Then, x is assigned (given) the value 6:

Java Script Objects

- What is An Object
- It is an element which is having certain properties
- Mobile, TV, Pen , Car , Bus , Student …. Etc.
- In JavaScript Every Thing is an Object .
- Example : Car model , color , make all are properties of an Object

JavaScript Objects

- **Accessing Object Properties**
- You can access object properties in two ways
  - *objectName.propertyName*
  - *objectName[propertyName]*
- **Accessing Object Methods**
- You access an object method with the following syntax:
  - *objectName.methodName()*

Java Script - Arrays

Java Script Arrays Are Collection of the Objects or values . Each Object will be having Similar kind of its own properties as we have seen in Early Slides .

Java Script Arrays will be surrounded with “[ ]” .

The Data Inside this Square Brackets are related to this Array .

Example : states=[];

The “States” is an Array . It don’t have any values . It is an Empty Array .

Let us add one value for that Array .

States=[“Karnataka ”];

How many Values we have In States Array : 1

JavaScript Arrays

- **Creating an Array**
  - var *array-name* = [*item1*, *item2*, ...];

  Access the Elements of an Array

  var name = cars[0];

  cars[0] = "Opel";

  Array Properties and Methods

  The **length** property of an array returns the length of an array (the number of array elements)

Adding Array Elements

- var fruits = ["Banana", "Orange", "Apple", "Mango"];
  fruits.push("Lemon");
- var fruits = ["Banana", "Orange", "Apple", "Mango"];
  fruits[fruits.length] = "Lemon";
- var fruits = ["Banana", "Orange", "Apple", "Mango"];
  fruits[10] = "Lemon";

Accessing The Arrays

```
States=[“Karnataka ”,”Andhra”,”Telangana”];
console.log(States[0]);
console.log(States[1]);
console.log(States[2]);
```

Array Index Positons are started from 0,1,2,3

Note : No of Values are different and Index positions are different .

JavaScript Array Methods

- **Converting Arrays to Strings**

- var fruits = ["Banana", "Orange", "Apple", "Mango"];
  document.getElementById("demo").innerHTML = fruits.valueOf();

- var fruits = ["Banana", "Orange", "Apple", "Mango"];
  document.getElementById("demo").innerHTML = fruits.toString();

Poping and Pushing

- When you work with arrays, it is easy to remove elements and add new elements.
- This is what popping and pushing is: Popping items out of an array, or pushing items into an array.
- The **pop()** method removes the last element from an array

Shift ,Unshift and Splice

- **Shifting Elements**
- Shifting is equivalent to popping, working on the first element instead of the last.
- The **shift()** method removes the first element of an array, and "shifts" all other elements one place down
- The **unshift()** method adds a new element to an array (at the beginning), and "unshifts" older elements:
- **Deleting Elements**
- Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator **delete**
- **Splicing an Array**
- The **splice()** method can be used to add new items to an array:
- **Using splice() to Remove Elements**
- With clever parameter setting, you can use splice() to remove elements without leaving "holes" in the array:

Sort ,Reverse and Slice

- **Sorting an Array**
- The **sort()** method sorts an array alphabetically:
- **Reversing an Array**
- The **reverse()** method reverses the elements in an array.
- You can use it to sort an array in descending order:
- **Slicing an Array**
- The **slice()** method slices out a piece of an array into a new array:

JavaScript Functions

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when "something" invokes it (calls it)
  - function *name*(*parameter1, parameter2, parameter3*) {
    code to be executed
    }

Different ways of declaring functions

- function A(){}; // function declaration
- var B = function(){}; // function expression
- var C = (function(){}); // function expression with grouping operators
- var D = function foo(){}; // named function expression
- var E = (function(){ // IIFE that returns a function
- return function(){}
- })();
- var F = new Function(); // Function constructor
- var G = new function(){}; // special case: object constructor

Function Invocation

- The code inside the function will execute when "something" **invokes** (calls) the function:
  - When an event occurs (when a user clicks a button)
  - When it is invoked (called) from JavaScript code
  - Automatically (self invoked)

Function Return

- When JavaScript reaches a **return statement**, the function will stop executing.
- If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
- Functions often compute a **return value**. The return value is "returned" back to the "caller":

JavaScript Loops

- Loops are handy, if you want to run the same code over and over again, each time with a different value.
- Often this is the case when working with arrays:

Different Kinds of Loops

JavaScript supports different kinds of loops:

- **for** - loops through a block of code a number of times
- **for/in** - loops through the properties of an object
- **while** - loops through a block of code while a specified condition is true
- **do/while** - also loops through a block of code while a specified condition is true

Conditional Statements

- **JavaScript If...Else Statements**
- Use **if** to specify a block of code to be executed, if a specified condition is true
- Use **else** to specify a block of code to be executed, if the same condition is false
- Use **else if** to specify a new condition to test, if the first condition is false
- Use **switch** to specify many alternative blocks of code to be executed
- **Syntax**
- if (*condition*) {
  block of code to be executed if the condition is true
  }

JavaScript Switch Statement

- Use the switch statement to select one of many blocks of code to be executed.
- **Syntax**
- switch(*expression*) {
  case *n*:
  code block
  break;
  case *n*:
  code block
  break;
  default:
  default code block
  }
- This is how it works:
- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.

JavaScript Operators

- JavaScript uses an **assignment operator** (=) to **assign** values to variables:
- **JavaScript Keywords**
- JavaScript **keywords** are used to identify actions to be performed.
- The **var** keyword tells the browser to create a new variable:
- **JavaScript Comments**
- Not all JavaScript statements are "executed".
- Code after double slashes **//** or between **/*** and ***/** is treated as a **comment**.
- Comments are ignored, and will not be executed:

JavaScript Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers (literals or variables).

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

The **addition** operator (+) adds numbers:

JavaScript Assignment Operators

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

Operator	Example	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

The **assignment** operator (=) assigns a value to a variable.

Comparison Operators

Operator	Description	Comparing	Returns	Try it
==	equal to	x == 8	false	Try it »
		x == 5	true	Try it »
===	equal value and equal type	x === "5"	false	Try it »
		x === 5	true	Try it »
!=	not equal	x != 8	true	Try it »
!==	not equal value or not equal type	x !== "5"	true	Try it »
		x !== 5	false	Try it »
>	greater than	x > 8	false	Try it »
<	less than	x < 8	true	Try it »
>=	greater than or equal to	x >= 8	false	Try it »
<=	less than or equal to	x <= 8	*true*	Try it »

JavaScript Events

- HTML events are **"things"** that happen to HTML elements.
- When JavaScript is used in HTML pages, JavaScript can **"react"** on these events
- JavaScript lets you execute code when events are detected.
- HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.
- With single quotes:
- *<some-HTML-element some-event=**'some JavaScript'**>*
- With double quotes:
- *<some-HTML-element some-event=**"some JavaScript"**>*

Common HTML Events

Here is a list of some common HTML events:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

JavaScript Scope

- In JavaScript, objects and functions are also variables.
- **In JavaScript, scope is the set of variables, objects, and functions you have access to.**

  JavaScript has function scope: The scope changes inside functions
- **Local JavaScript Variables**

  Variables declared within a JavaScript function, become **LOCAL** to the function.

  Local variables have **local scope**: They can only be accessed within the function.
- **Global JavaScript Variables**

  A variable declared outside a function, becomes **GLOBAL**.

  A global variable has **global scope**: All scripts and functions on a web page can access it

JavaScript Strings Properties

String 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

JavaScript String Methods

- **Finding a String in a String**
- The **indexOf()** method returns the index of (the position of) the **first** occurrence of a specified text in a string:
- var str = "Please locate where 'locate' occurs!";
  var pos = str.indexOf("locate");
- The **lastIndexOf()** method returns the index of the **last** occurrence of a specified text in a string:
- var str = "Please locate where 'locate' occurs!";
  var pos = str.lastIndexOf("locate");

- **Searching for a String in a String**

  The **search()** method searches a string for a specified value and returns the position of the match:

  ```
  var str = "Please locate where 'locate' occurs!";
  var pos = str.search("locate");
  ```

- **Extracting String Parts**

  slice(start, end)

  substring(start, end)

  substr(start, length)

- **Replacing String Content**

  The **replace()** method replaces a specified value with another value in a string

Using String Methods

- In JavaScript, regular expressions are often used with the two **string methods**: search() and replace().
  - **The search() method** uses an expression to search for a match, and returns the position of the match.
  - **The replace() method** returns a modified string where the pattern is replaced
- **Using String search() With a Regular Expression**
- **Example**

Use a regular expression to do a case-insensitive search for "w3schools" in a string:

```
var str = "Visit W3Schools";
var n = str.search(/w3schools/i);
```

The result in n will be: 6

JavaScript Numbers

Property	Description
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)
NaN	Represents a "Not-a-Number" value
POSITIVE_INFINITY	Represents infinity (returned on overflow)

JavaScript Number Methods

Global Methods

JavaScript global functions can be used on all JavaScript data types.

These are the most relevant methods, when working with numbers:

Method	Description
Number()	Returns a number, converted from its argument.
parseFloat()	Parses its argument and returns a floating point number
parseInt()	Parses its argument and returns an integer

Number Methods

JavaScript number methods are methods that can be used on numbers:

Method	Description
toString()	Returns a number as a string
toExponential()	Returns a string, with a number rounded and written using exponential notation.
toFixed()	Returns a string, with a number rounded and written with a specified number of decimals.
toPrecision()	Returns a string, with a number written with a specified length
valueOf()	Returns a number as a number

JavaScript Dates

- **JavaScript Date Formats**
- **Displaying Dates**
- new Date()
  new Date(milliseconds)
  new Date(dateString)
  new Date(year, month, day, hours, minutes, seconds, milliseconds)
- **Displaying Dates**
- When you display a date object in HTML, it is automatically converted to a string, with the **toString()** method.
- The **toUTCString()** method converts a date to a UTC string (a date display standard)
- The **toDateString()** method converts a date to a more readable format

JavaScript Date Methods

Date Get Methods

Get methods are used for getting a part of a date. Here are the most common (alphabetically):

Method	Description
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

JavaScript Regular Expressions

- A regular expression is a sequence of characters that forms a **search pattern**.
- When you search for data in a text, you can use this search pattern to describe what you are searching for.
- A regular expression can be a single character, or a more complicated pattern.
- Regular expressions can be used to perform all types of **text search** and **text replace** operations
- **Syntax**
- */pattern/modifiers;*

JavaScript Strings & Regular Expressions

Regular expressions can be used with the following string methods

1. match()
2. replace()
3. split()
4. search()

Modifiers can be used with regular expressions to specify the kind of search

g	Global search
i	Case-insensitive search
m	Multiline search

Using regular expression with match() method

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";

document.write(string.match(/\d+/g));
```

Output : 1011011010,35,8912398912,45

JavaScript Strings & Regular Expressions

Using regular expression with replace() method

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";

document.write(string.replace(/\d+/g, "XXX"));
```

↓

Tom contact number is XXX. His age is XXX.
Mark contact number is XXX. His age is XXX

Using regular expression with split() method

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";

document.write(string.split(/\d+/))
```

↓

Tom contact number is ,. His age is ,
.Mark contact number is ,. His age is ,

JavaScript Strings & Regular Expressions

Using regular expression with search() method

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";

document.write(string.search(/\d+/))
```

↓

22

Global case insensitive match using a regular expression

```
var string = "TOM contact number is 1011011010. tom is 35";
document.write(string.match(/tom/gi));
```

↓

TOM,tom

JavaScript RegExp object

There are 2 ways to create a regular expression in JavaScript

Using a regular expression literal

```
var regex = /\d+/g;
```

Using the constructor function of the RegExp object

```
var regexp = new RegExp("\\d+", "g");
```

Commonly used RegExp object properties

global	returns true if the global modifier (g) is set, otherwise false
ignoreCase	returns true if the case-insensitive modifier (i) is set, otherwise false
multiline	returns true if the multi-line modifier (m) is set, otherwise false
source	Returns the text of the regular expression

Example :

```
var regexp = new RegExp("\\d+", "gi");

document.write("g = " + regexp.global + "<br/>");
document.write("i = " + regexp.ignoreCase + "<br/>");
document.write("m = " + regexp.multiline + "<br/>");
document.write("source = " + regexp.source + "<br/>");
```

→

g = true
i = true
m = false
source = \d+

JavaScript RegExp object

Commonly used RegExp object methods

exec()	Tests for a match in a given string and returns the first match if found otherwise null.
test()	Tests for a match in a gievn string and returns true or false
toString()	Returns the string value of the regular expression

exec() method returns the first match

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";

var regexp = new RegExp("\\d+");
document.write(regexp.exec(string));
```

→ 1011011010

To get all the matches call .exec() method repeatedly with the g flag

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";

var regexp = new RegExp("\\d+", "g");
var result;

while ((result = regexp.exec(string)) != null)
{
    document.write(result[0] + "<br/>");
}
```

→ 1011011010
35
8912398912
45

JavaScript RegExp object

The following example calls test() method of the RegExp object to check if the string contains numbers

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";

var regexp = new RegExp("\\d+", "g");
document.write("String contain numbers - " + regexp.test(string))
```

↓

```
String contain numbers - true
```

You can also call exec() and test() methods of the RegExp object as shown below

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";

var regexp = /\d+/g;
document.write("String contain numbers - " + regexp.test(string))
```

OR

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";

document.write("String contain numbers - " + /\d+/g.test(string))
```

JavaScript Errors Throw and Try to Catch

- The **try** statement lets you test a block of code for errors.
- The **catch** statement lets you handle the error.
- The **throw** statement lets you create custom errors.
- The **finally** statement lets you execute code, after try and catch, regardless of the result
- **JavaScript try and catch**
- The **try** statement allows you to define a block of code to be tested for errors while it is being executed.
- The **catch** statement allows you to define a block of code to be executed, if an error occurs in the try block.
- The JavaScript statements **try** and **catch** come in pairs:
- try {
  Block of code to try
  }
  catch(err) {
  Block of code to handle errors
  }

JavaScript HTML DOM

- With the object model, JavaScript gets all the power it needs to create dynamic HTML:
- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

Finding HTML Elements

Method	Description
document.getElementById()	Find an element by element id
document.getElementsByTagName()	Find elements by tag name
document.getElementsByClassName()	Find elements by class name

Changing HTML Elements

Method	Description
element.innerHTML=	Change the inner HTML of an element
element.attribute=	Change the attribute of an HTML element
element.setAttribute*(attribute,value)*	Change the attribute of an HTML element
element.style.*property*=	Change the style of an HTML element

Adding and Deleting Elements

Method	Description
document.createElement()	Create an HTML element
document.removeChild()	Remove an HTML element
document.appendChild()	Add an HTML element
document.replaceChild()	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Adding Events Handlers

Method	Description
document.getElementById(*id*).onclick=function(){*code*}	Adding event handler code to an onclick event

- The following HTML objects (and object collections) are also accessible:
- document.anchors
- document.body
- document.documentElement
- document.embeds
- document.forms
- document.head
- document.images
- document.links
- document.scripts
- document.title

JavaScript HTML DOM - Changing HTML

- **Changing the HTML Output Stream**
- **Changing HTML Content**
- document.getElementById(*id*).innerHTML = *new HTML*
- **Changing the Value of an Attribute**
- document.getElementById(*id*).*attribute*=*new value*

JavaScript HTML DOM - Changing CSS

- **Changing HTML Style**
- To change the style of an HTML element, use this syntax:
- document.getElementById(*id*).style.*property=new style*
- **Using Events**
- The HTML DOM allows you to execute code when an event occurs.
- Events are generated by the browser when "things happen" to HTML elements:
- An element is clicked on
- The page has loaded
- Input fields are changed

JavaScript HTML DOM Events

- **Reacting to Events**
- A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.
- To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:
- onclick=*JavaScript*
- Examples of HTML events:
- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

JavaScript HTML DOM Navigation

- **DOM Nodes**
- According to the W3C HTML DOM standard, everything in an HTML document is a node:
- The entire document is a document node
- Every HTML element is an element node
- The text inside HTML elements are text nodes
- Every HTML attribute is an attribute node
- All comments are comment nodes

JavaScript HTML DOM EventListener

- The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.
- You can add many event handlers of the same type to one element, i.e two "click" events.
- You can add event listeners to any DOM object not only HTML elements. i.e the window object.
- The addEventListener() method makes it easier to control how the event reacts to bubbling.
- When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.
- You can easily remove an event listener by using the removeEventListener() method.
- **Syntax**

  element.addEventListener(*event, function, useCapture*);

Navigating Between Nodes

- You can use the following node properties to navigate between nodes with JavaScript:
  - parentNode
  - childNodes[*nodenumber*]
  - firstChild
  - lastChild
  - nextSibling
  - previousSibling

JavaScript Window - The Browser Object Model

- **The Browser Object Model (BOM)**

  There are no official standards for the **B**rowser **O**bject **M**odel (BOM).

  Since modern browsers have implemented (almost) the same methods and properties for JavaScript interactivity, it is often referred to, as methods and properties of the BOM

- **The Window Object**
  - The **window** object is supported by all browsers. It represents the browser's window.
  - All global JavaScript objects, functions, and variables automatically become members of the window object.
  - Global variables are properties of the window object.
  - Global functions are methods of the window object.
  - Even the document object (of the HTML DOM) is a property of the window object:
  - window.document.getElementById("header");
  - is the same as:
  - document.getElementById("header");

Window Size

- Three different properties can be used to determine the size of the browser window (the browser viewport, NOT including toolbars and scrollbars).
- For Internet Explorer, Chrome, Firefox, Opera, and Safari:
  - window.innerHeight - the inner height of the browser window
  - window.innerWidth - the inner width of the browser window
- For Internet Explorer 8, 7, 6, 5:
  - document.documentElement.clientHeight
  - document.documentElement.clientWidth
  - or
  - document.body.clientHeight
  - document.body.clientWidth

Other Window Methods

- Some other methods:
  - window.open() - open a new window
  - window.close() - close the current window
  - window.moveTo() -move the current window
  - window.resizeTo() -resize the current window

Window Screen

- The **window.screen** object can be written without the window prefix.
- Properties:
  - screen.width
  - screen.height
  - screen.availWidth
  - screen.availHeight
  - screen.colorDepth
  - screen.pixelDepth

Window Location

- The **window.location** object can be written without the window prefix.
- Some examples:
  - window.location.href returns the href (URL) of the current page
  - window.location.hostname returns the domain name of the web host
  - window.location.pathname returns the path and filename of the current page
  - window.location.protocol returns the web protocol used (http:// or https://)
  - window.location.assign loads a new document
- **Window Location Href**

  The **window.location.href** property returns the URL of the current page
- **Window Location Pathname**

  The **window.location.pathname** property returns the pathname of the current page.
- **Window Location Hostname**

  The **window.location.hostname** property returns the name of the internet host (of the current page)
- **Window Location Protocol**

  The **window.location.protocol** property returns the web protocol of the page
- **Window Location Assign**

  The **window.location.assign()** method loads a new document

JavaScript Window History

- The **window.history** object can be written without the window prefix.
- To protect the privacy of the users, there are limitations to how JavaScript can access this object.
- Some methods:

  history.back() - same as clicking back in the browser

  history.forward() - same as clicking forward in the browser

Window Navigator

- The **window.navigator** object can be written without the window prefix.
- Some examples:
  - navigator.appName
  - navigator.appCodeName
  - navigator.platform
- **Navigator Cookie Enabled**

  The property cookieEnabled returns true if cookies are enabled, otherwise false
- **The Browser Names**

  The properties **appName** and **appCodeName** return the name of the browser
- **The Browser Engine**

  The property **product** returns the engine name of the browser
- **The Browser Version I**

  The property **appVersion** returns version information about the browser
- **The Browser Version II**

  The property **userAgent also** returns version information about the browser:
- **The Browser Platform**

  The property **platform** returns the browser platform (operating system)
- **The Browser Language**

  The property **language** returns the browser's language:

JavaScript Timing Events

- **JavaScript Timing Events**
- With JavaScript, it is possible to execute some code at specified time-intervals. This is called timing events.
- It's very easy to time events in JavaScript. The two key methods that are used are:
- setInterval() - executes a function, over and over again, at specified time intervals
- setTimeout() - executes a function, once, after waiting a specified number of milliseconds
- **Syntax**
- window.setInterval("*javascript function*", *milliseconds*);
- The **window.setInterval()** method can be written without the window prefix.
- The first parameter of setInterval() should be a function.
- The second parameter indicates the length of the time-intervals between each execution.
- **Note:** There are 1000 milliseconds in one second.

How to Stop the Execution?

- The clearInterval() method is used to stop further executions of the function specified in the setInterval() method.
- **Syntax**
  - window.clearInterval(*intervalVariable*)
- The **window.clearInterval()** method can be written without the window prefix.
- To be able to use the clearInterval() method, you must use a global variable when creating the interval method:
- myVar=setInterval("*javascript function*", *milliseconds*);
- Then you will be able to stop the execution by calling the clearInterval() method.

- The clearTimeout() method is used to stop the execution of the function specified in the setTimeout() method.
- **Syntax**
- window.clearTimeout(*timeoutVariable*)
- The **window.clearTimeout()** method can be written without the window prefix.
- To be able to use the clearTimeout() method, you must use a global variable when creating the timeout method:
- myVar=setTimeout("*javascript function*", *milliseconds*);
- Then, if the function has not already been executed, you will be able to stop the execution by calling the clearTimeout() method.

JavaScript Cookies

- **What are Cookies?**
- Cookies are data, stored in small text files, on your computer.
- When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.
- Cookies were invented to solve the problem "how to remember information about the user":
- When a user visits a web page, his name can be stored in a cookie.
- Next time the user visits the page, the cookie "remembers" his name.
- Cookies are saved in name-value pairs like:
- username=John Doe

- JavaScript can create, read, and delete cookies with the **document.cookie** property.
- With JavaScript, a cookie can be created like this:
- document.cookie="username=John Doe";
- You can also add an expiry date (in UTC time). By default, the cookie is deleted when the browser is closed:
- document.cookie="username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC";
- With a path parameter, you can tell the browser what path the cookie belongs to. By default, the cookie belongs to the current page.
- document.cookie="username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";

Read a Cookie with JavaScript

- With JavaScript, cookies can be read like this:
- var x = document.cookie;
- **Change a Cookie with JavaScript**
- With JavaScript, you can change a cookie the same way as you create it:
- document.cookie="username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/"
- **Delete a Cookie with JavaScript**
- Deleting a cookie is very simple. Just set the expires parameter to a passed date:
- document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC";