UNIT V
CLIENT SIDE SCRIPTING

JavaScript:

Javascript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities. JavaScript was first known as LiveScript, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name LiveScript. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers. The ECMA-262 Specification defined a standard version of the core JavaScript language.

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform.

Client-Side JavaScript:

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser. It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content. The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field. The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server. JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

Advantages of JavaScript:

The merits of using JavaScript are: □

- Less server interaction: You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- Immediate feedback to the visitors: They don't have to wait for a page reload to see if they have forgotten to enter something.
- Increased interactivity: You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- Richer interfaces: You can use JavaScript to include such items as dragand-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of JavaScript:

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features:

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities.

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.

You can place the **<script>** tags, containing your JavaScript, anywhere within you web page, but it is normally recommended that you should keep it within the **<head>** tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

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

The script tag takes two important attributes:

- **Language:** This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **Type:** This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So your JavaScript syntax will look as follows.

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

EXAMPLE CODE:

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "//->". Here "//" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code. Next, we call a function document.write which writes a string into our HTML document.

```
<html>
<body>
<script language="javascript" type="text/javascript">
```

```
<!--
   document.write ("Hello World!")
//-->
</script>
</body>
</html>
```

This code will produce the following result:
Hello World!

Whitespace and Line Breaks

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

Semicolons are Optional

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

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

But when formatted in a single line as follows, you must use semicolons:

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

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

Case Sensitivity

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.
So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

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

Comments in JavaScript

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

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

  EXAMPLE:
  The following example shows how to use comments in JavaScript.

```
<script language="javascript" type="text/javascript">
<!--
// This is a comment. It is similar to comments in C++
/*
 * This is a multiline comment in JavaScript
 * It is very similar to comments in C Programming
 */
//-->
</script>
```

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

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

  You can also declare multiple variables with the same var keyword as follows:

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

```
//-->
</script>
```

Storing a value in a variable is called variable initialization. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable. For instance, you might create a variable named money and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

```
<script type="text/javascript">
<!--
var name = "Ali";
var money;
money = 2000.50;
//-->
</script>
```

Note: Use the var keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice. JavaScript is untyped language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes. □ Global Variables: A global variable has global scope which means it can be defined anywhere in your JavaScript code. □ Local Variables: A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

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

```
//-->
</script>
```

It will produce the following result:
Local

JavaScript Variable Names

While naming your variables in JavaScript, keep the following rules in mind.

- You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, break or boolean variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, 123test is an invalid variable name but _123test is a valid one.
- JavaScript variable names are case-sensitive. For example, Name and name are two different variables.

FUNCTIONS

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions. Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. You must have seen functions like alert() and write() in the earlier chapters. We were using these functions again and again, but they had been written in core JavaScript only once. JavaScript allows us to write our own functions as well

Function Definition

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the function keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.
Syntax The basic syntax is shown here.

```
<script type="text/javascript">
<!--
function functionname(parameter-list)
{
  statements
}
//-->
</script>
```

Example Try the following example. It defines a function called sayHello that takes no parameters:

```
<script type="text/javascript">
<!--
function sayHello()
{
   alert("Hello there");
}
//-->
</script>
```

Calling a Function

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<html>
<head>
<script type="text/javascript">
function sayHello()
{
  document.write ("Hello there!");
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello()" value="Say Hello">
</form>
<p>Use different text in write method and then try...</p>
</body>
</html>
```

Output

Click the following button to call the function

Say Hello

Function Parameters

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Example Try the following example. We have modified our sayHello function here. Now it takes two parameters.

```
<html>
<head>
<script type="text/javascript">
function sayHello(name, age)
{
   document.write (name + " is " + age + " years old.");
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Output

Click the following button to call the function

Say Hello

Use different parameters inside the function and then try...

The return Statement

A JavaScript function can have an optional return statement. This is required if you want to return a value from a function. This statement should be the last statement in a function. For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Example

Try the following example. It defines a function that takes two parameters and concatenates them before returning the resultant in the calling program.

```
<html>
<head>
<script type="text/javascript">
function concatenate(first, last)
{
   var full;
```

```
   full = first + last;
   return  full;
}
function secondFunction()
{
   var result;
   result = concatenate('Zara', 'Ali');
   document.write (result );
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Output

Click the following button to call the function

Call Function

Use different parameters inside the function and then try...

Nested Functions

Prior to JavaScript 1.2, function definition was allowed only in top level global code, but JavaScript 1.2 allows function definitions to be nested within other functions as well. Still there is a restriction that function definitions may not appear within loops or conditionals. These restrictions on function definitions apply only to function declarations with the function statement.

Example

Try the following example to learn how to implement nested functions.

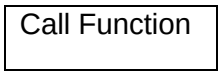
```
<html>
<head>
<script type="text/javascript">
<!--
function hypotenuse(a, b) {
   function square(x) { return x*x; }

   return Math.sqrt(square(a) + square(b));
}
```

```
function secondFunction(){
   var result;
   result = hypotenuse(1,2);
   document.write ( result );
}
//-->
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Output

Click the following button to call the function



Use different parameters inside the function and then try...

Function () Constructor

The function statement is not the only way to define a new function; you can define your function dynamically using Function() constructor along with the new operator.
Note: Constructor is a terminology from Object Oriented Programming. You may not feel comfortable for the first time, which is OK.
Syntax
Following is the syntax to create a function using Function() constructor along with the new operator.

```
<script type="text/javascript">
<!--
var variablename = new Function(Arg1, Arg2..., "Function Body");
//-->
</script>
```

The Function() constructor expects any number of string arguments. The last argument is the body of the function – it can contain arbitrary JavaScript statements, separated from each other by semicolons.

Notice that the Function() constructor is not passed any argument that specifies a name for the function it creates. The unnamed functions created with the Function() constructor are called anonymous functions.

Example

Try the following example.

```
<html>
<head>
<script type="text/javascript">
<!--
var func = new Function("x", "y", "return x*y;");

function secondFunction(){
   var result;
   result = func(10,20);
document.write ( result );
}
//-->
</script>
</head>

<body>
<p>Click the following button to call the function</p>

<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>

<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Output

Click the following button to call the function

Call Function

Use different parameters inside the function and then try...

Function Literals

JavaScript 1.2 introduces the concept of function literals which is another new way of defining functions. A function literal is an expression that defines an unnamed function.

Syntax

The syntax for a function literal is much like a function statement, except that it is used as an expression rather than a statement and no function name is required.

```
<script type="text/javascript">
<!--
var variablename = function(Argument List){
                    Function Body
  };
//-->
</script>
```

Syntactically, you can specify a function name while creating a literal function as follows.

```
<script type="text/javascript">
<!--
var variablename = function FunctionName(Argument List){
                    Function Body
     };
//-->
</script>
```

But this name does not have any significance, so it is not worthwhile.

Example

Try the following example. It shows the usage of function literals.

```
<html>
<head>
<script type="text/javascript">
<!--
var func = function(x,y){ return x*y };

function secondFunction(){
  var result;
  result = func(10,20);
  document.write ( result );
}
//-->
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
```

```
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Output

Click the following button to call the function

Call Function

Use different parameters inside the function and then try...

EVENTS

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page. When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc. Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable. Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code. Please go through this small tutorial for a better understanding HTML Event Reference. Here we will see a few examples to understand the relation between Event and JavaScript.

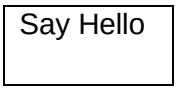
onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

Example

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
   document.write ("Hello World")
}
//-->
</script>
</head>
<body>
<p> Click the following button and see result</p>
<input type="button" onclick="sayHello()" value="Say Hello" />
```

```
</body>
</html>
```
Output
Click the following button and see result



onsubmit Event Type

onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.
Example
The following example shows how to use onsubmit. Here we are calling a validate() function before submitting a form data to the webserver. If validate() function returns true, the form will be submitted, otherwise it will not submit the data.
Try the following example.

```
<html>
<head>
<script type="text/javascript">
<!--
function validation() {
   all validation goes here
   .........
   return either true or false
}
//-->
</script>
</head>
<body>
<form method="POST" action="t.cgi" onsubmit="return validate()">
.......
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

onmouseover and onmouseout

These two event types will help you create nice effects with images or even with text as well. The onmouseover event triggers when you bring your mouse over any element and the onmouseout triggers when you move your mouse out from that element. Try the following example.

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

```
<!--
function over() {
  document.write ("Mouse Over");
}
function out() {
  document.write ("Mouse Out");
}
//-->
</script>
</head>
<body>
<p>Bring your mouse inside the division to see the result:</p>
<div onmouseover="over()" onmouseout="out()">
<h2> This is inside the division </h2>
</div>
</body>
</html>
```

Output

Bring your mouse inside the division to see the result:

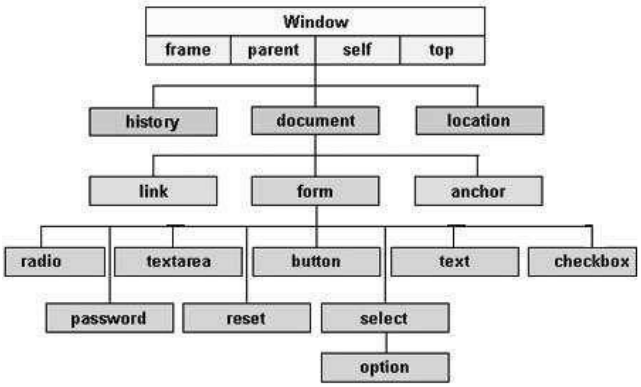
This is inside the division

DOM

Every web page resides inside a browser window which can be considered as an object.
A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.
The way a document content is accessed and modified is called the Document Object Model, or DOM. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- Window object: Top of the hierarchy. It is the outmost element of the object hierarchy.
- Document object: Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- Form object: Everything enclosed in the <form>...</form> tags sets the form object.
- Form control elements: The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

Here is a simple hierarchy of a few important objects:



There are several DOMs in existence. The following sections explain each of these DOMs in detail and describe how you can use them to access and modify document content.

- The Legacy DOM: This is the model which was introduced in early versions of JavaScript language. It is well supported by all browsers, but allows access only to certain key portions of documents, such as forms, form elements, and images.
- The W3C DOM: This document object model allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C). This model is supported by almost all the modern browsers.
- The IE4 DOM: This document object model was introduced in Version 4 of Microsoft's Internet Explorer browser. IE 5 and later versions include support for most basic W3C DOM features.

FORM VALIDATION

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

- Basic Validation - First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
- Data Format Validation - Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

Example

We will take an example to understand the process of validation. Here is a simple form in html format.

```
<html>
<head>
<title>Form Validation</title>
```

```
<script type="text/javascript">
<!--
// Form validation code will come here.
//-->
</script>
</head>
<body>
 <form action="/cgi-bin/test.cgi" name="myForm"
        onsubmit="return(validate());">
 <table cellspacing="2" cellpadding="2" border="1">
 <tr>
   <td align="right">Name</td>
<td><input type="text" name="Name" /></td>
 </tr>
 <tr>
   <td align="right">EMail</td>
   <td><input type="text" name="EMail" /></td>
 </tr>
 <tr>
   <td align="right">Zip Code</td>
   <td><input type="text" name="Zip" /></td>
 </tr>
 <tr>
 <td align="right">Country</td>
 <td>
 <select name="Country">
   <option value="-1" selected>[choose yours]</option>
   <option value="1">USA</option>
   <option value="2">UK</option>
   <option value="3">INDIA</option>
 </select>
 </td>
 </tr>
 <tr>
   <td align="right"></td>
   <td><input type="submit" value="Submit" /></td>
 </tr>
 </table>
 </form>
 </body>
 </html>
```

Basic Form Validation

First let us see how to do a basic form validation. In the above form, we are calling validate() to validate data when onsubmit event is occurring. The following code shows the implementation of this validate() function.

```
<script type="text/javascript">
<!--
// Form validation code will come here.
function validate()
{

   if( document.myForm.Name.value == "" )
   {
     alert( "Please provide your name!" );
     document.myForm.Name.focus() ;
     return false;
   }
   if( document.myForm.EMail.value == "" )
   {
     alert( "Please provide your Email!" );
     document.myForm.EMail.focus() ;
     return false;
}
   if( document.myForm.Zip.value == "" ||
         isNaN( document.myForm.Zip.value ) ||
         document.myForm.Zip.value.length != 5 )
   {
     alert( "Please provide a zip in the format #####." );
     document.myForm.Zip.focus() ;
     return false;
   }
   if( document.myForm.Country.value == "-1" )
   {
     alert( "Please provide your country!" );
     return false;
   }
   return( true );
}
//-->
</script>
```

Data Format Validation

Now we will see how we can validate our entered form data before submitting it to the web server.

The following example shows how to validate an entered email address. An email address must contain at least a ‘@’ sign and a dot (.). Also, the ‘@’ must not be the first character of the email address, and the last dot must at least be one character after the ‘@’ sign.

Example
Try the following code for email validation.

```
<script type="text/javascript">
<!--
function validateEmail()
{
   var emailID = document.myForm.EMail.value;
   atpos = emailID.indexOf("@");
   dotpos = emailID.lastIndexOf(".");
   if (atpos < 1 || ( dotpos - atpos < 2 ))
   {
      alert("Please enter correct email ID")
      document.myForm.EMail.focus() ;
      return false;
   }
   return( true );
}
//-->
</script>
```

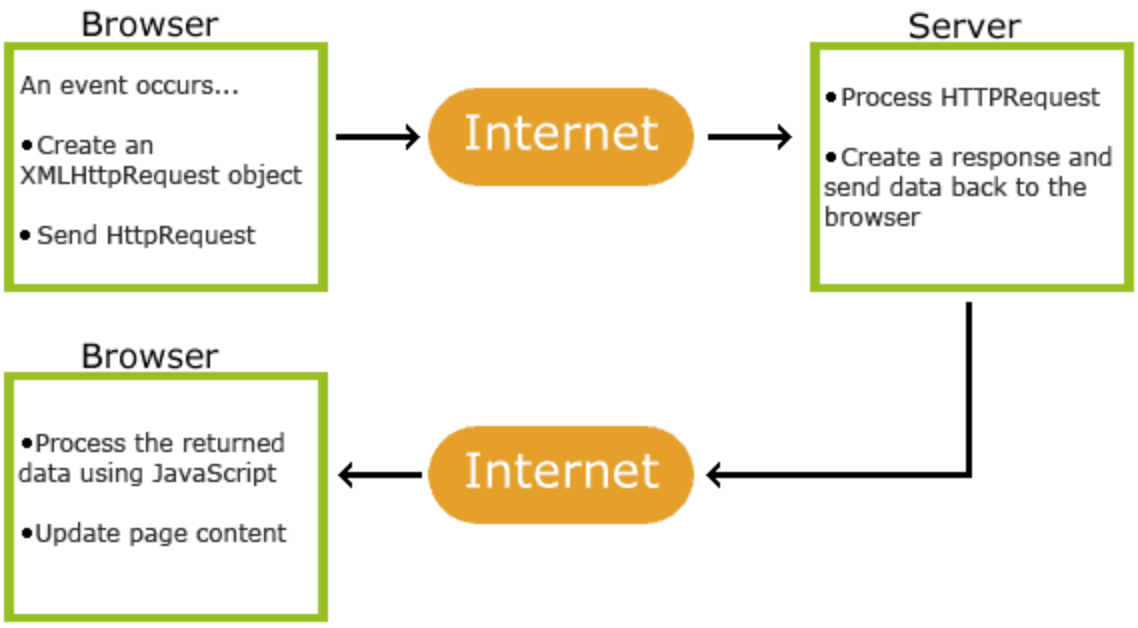
What is AJAX?

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.

AJAX is not a programming language.

AJAX just uses a combination of:

- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)



- 1. An event occurs in a web page (the page is loaded, a button is clicked)
- 2. An XMLHttpRequest object is created by JavaScript
- 3. The XMLHttpRequest object sends a request to a web server
- 4. The server processes the request
- 5. The server sends a response back to the web page
- 6. The response is read by JavaScript
- 7. Proper action (like page update) is performed by JavaScript

Create an XMLHttpRequest Object

All modern browsers (Chrome, IE7+, Firefox, Safari, and Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

variable = new XMLHttpRequest();

XMLHttpRequest Object Methods

Method	Description

new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(*method,url,async,user,psw*)	Specifies the request *method*: the request type GET or POST *url*: the file location *async*: true (asynchronous) or false (synchronous) *user*: optional user name *psw*: optional password
send()	Sends the request to the server Used for GET requests
send(*string*)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

Send a Request To a Server

To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object:

```
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

Method	Description
open(*method, url, async*)	Specifies the type of request *method*: the type of request: GET or POST *url*: the server (file) location *async*: true (asynchronous) or false (synchronous)
send()	Sends the request to the server (used for GET)
send(*string*)	Sends the request to the server (used for POST)