1. What is Java Script?

JavaScript is an object-based scripting language that is lightweight and cross-platform which is a solution for client side dynamic pages. JavaScript is not compiled but translated. The JavaScript Translator (embedded in browser) is responsible to translate the JavaScript code.

1. Where JavaScript is used?

JavaScript is used to create interactive websites. It is mainly used for:

* Client-side validation
* Dynamic drop-down menus
* Displaying data and time
* Displaying popup windows and dialog boxes (like alert dialog box, confirm dialog box and prompt dialog box)
* Displaying clocks etc.

## **JavaScript Example**

Javascript example is easy to code. JavaScript provides 3 places to put the JavaScript code: within body tag, within head tag and external JavaScript file.

1. **<h2>**Welcome to JavaScript**</h2>**
2. **<script>**
3. document.write("Hello JavaScript by JavaScript");
4. **</script>**

**JavaScript Example : code between the head tag**

1. **<html>**
2. **<head>**
3. **<script** type="text/javascript"**>**
4. function msg(){
5. alert("Hello Javatpoint");
6. }
7. **</script>**
8. **</head>**
9. **<body>**
10. **<p>**Welcome to JavaScript**</p>**
11. **<form>**
12. **<input** type="button" value="click" onclick="msg()"**/>**
13. **</form>**
14. **</body>**
15. **</html>**

# **External JavaScript file**

**message.js**

1. function msg(){
2. alert("Hello Javatpoint");
3. }

**index.html**

1. **<html>**
2. **<head>**
3. **<script** type="text/javascript" src="message.js"**></script>**
4. **</head>**
5. **<body>**
6. **<p>**Welcome to JavaScript**</p>**
7. **<form>**
8. **<input** type="button" value="click" onclick="msg()"**/>**
9. **</form>**
10. **</body>**
11. **</html>**
12. Advantages of JavaScript comments

There are mainly two advantages of JavaScript comments.

1. **To make code easy to understand** It can be used to elaborate the code so that end user can easily understand the code.
2. **To avoid the unnecessary code** It can also be used to avoid the code being executed. Sometimes, we add the code to perform some action. But after sometime, there may be need to disable the code. In such case, it is better to use comments.

Same as C and Java- Single and multi-line comments

1. **JavaScript Variable**

A **JavaScript variable** is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

1. Name must start with a letter (a to z or A to Z), underscore( \_ ), or dollar( $ ) sign.
2. After first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables.

## **Example of JavaScript variable**

Let’s see a simple example of JavaScript variable.

1. **<script>**
2. var x = 10;
3. var y = 20;
4. var z=x+y;
5. document.write(z);
6. **</script>**

If you declare a variable using window object, you can access it in another function also.

<html>

<body>

<script>

function m(){

window.value=100;//declaring global variable by window object

}

function n(){

alert(window.value);//accessing global variable from other function

}

m();

n();

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

**Internals of global variable in JavaScript**

When you declare a variable outside the function, it is added in the window object internally. You can access it through window object also. For example:

1. var value=50;
2. function a(){
3. alert(window.value);//accessing global variable
4. }

## **JavaScript primitive data types**

|  |  |
| --- | --- |
| String | represents sequence of characters e.g. "hello" |
| Number | represents numeric values e.g. 100 |
| Boolean | represents boolean value either false or true |
| Undefined | represents undefined value |
| Null | represents null i.e. no value at all |

## **JavaScript non-primitive data types**

The non-primitive data types are as follows:

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| Object | represents instance through which we can access members |
| Array | represents group of similar values |
| RegExp | represents regular expression |

1. **JavaScript Objects**

JavaScript is an object-based language. Everything is an object in JavaScript. JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

## **Creating Objects in JavaScript**

There are 3 ways to create objects.

1. By object literal
2. By creating instance of Object directly (using new keyword)
3. By using an object constructor (using new keyword)

## **1) JavaScript Object by object literal**

1. **<script>**
2. emp={id:102,name:"Shyam Kumar",salary:40000}
3. document.write(emp.id+" "+emp.name+" "+emp.salary);
4. **</script>**

## **2) By creating instance of Object**

1. **<script>**
2. var emp=new Object();
3. emp.id=101;
4. emp.name="Ravi Malik";
5. emp.salary=50000;
6. document.write(emp.id+" "+emp.name+" "+emp.salary);
7. **</script>**

## **3) By using an Object constructor**

1. **<script>**
2. function emp(id,name,salary){
3. this.id=id;
4. this.name=name;
5. this.salary=salary;
6. }
7. e=new emp(103,"Vimal Jaiswal",30000);
8. document.write(e.id+" "+e.name+" "+e.salary);  **</script>**

## **Defining method in JavaScript Object**

We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.

1. **<script>**
2. function emp(id,name,salary){
3. this.id=id;
4. this.name=name;
5. this.salary=salary;
7. this.changeSalary=changeSalary;
8. function changeSalary(otherSalary){
9. this.salary=otherSalary;
10. }
11. }
12. e=new emp(103,"Sonoo Jaiswal",30000);
13. document.write(e.id+" "+e.name+" "+e.salary);
14. e.changeSalary(45000);
15. document.write("**<br>**"+e.id+" "+e.name+" "+e.salary);
16. **</script>**

#### Output of the above example

103 SonooJaiswal 30000  
103 Sonoo Jaiswal 45000

# **JavaScript Array**

## **1) JavaScript array literal**

1. **<script>**
2. var emp=["Sonoo","Vimal","Ratan"];
3. for (i=0;i**<emp.length**;i++){
4. document.write(emp[i] + "**<br/>**");
5. }
6. **</script>**

## **2) JavaScript Array directly (new keyword)**

1. **<script>**
2. var i;
3. var emp = new Array();
4. emp[0] = "Arun";
5. emp[1] = "Varun";
6. emp[2] = "John";
8. for (i=0;i**<emp.length**;i++){
9. document.write(emp[i] + "**<br>**");
10. }
11. **</script>**

## **3) JavaScript array constructor (new keyword)**

1. **<script>**
2. var emp=new Array("Jai","Vijay","Smith");
3. for (i=0;i**<emp.length**;i++){
4. document.write(emp[i] + "**<br>**");
5. }
6. **</script>**

# **JavaScript String-2 Ways only**

## **1) By string literal**

1. **<script>**
2. var str="This is string literal";
3. document.write(str);
4. **</script>**

## **2) By string object (using new keyword)**

1. **<script>**
2. var stringname=new String("hello javascript string");
3. document.write(stringname);
4. **</script>**
5. **JavaScript Date Object**

The **JavaScript date** object can be used to get year, month and day. You can display a timer on the webpage by the help of JavaScript date object.

## **Constructor-** You can use 4 variant of Date constructor to create date object.

1. Date()
2. Date(milliseconds)
3. Date(dateString)
4. Date(year, month, day, hours, minutes, seconds, milliseconds)

### **JavaScript Date Example**

1. Current Date and Time: **<span** id="txt"**></span>**
2. **<script>**
3. var today=new Date();
4. document.getElementById('txt').innerHTML=today;
5. **</script>**

Current Date and Time: Mon May 29 2017 11:24:09 GMT+0530 (India Standard Time)

1. **<script>**
2. var date=new Date();
3. var day=date.getDate();
4. var month=date.getMonth()+1;
5. var year=date.getFullYear();
6. document.write("**<br>**Date is: "+day+"/"+month+"/"+year);
7. **</script>**

Output:

Date is: 29/5/2017

### **JavaScript Current Time Example**

1. Current Time: **<span** id="txt"**></span>**
2. **<script>**
3. var today=new Date();
4. var h=today.getHours();
5. var m=today.getMinutes();
6. var s=today.getSeconds();
7. document.getElementById('txt').innerHTML=h+":"+m+":"+s;
8. **</script>**

Output:

Current Time: 11:24:9

### **JavaScript Digital Clock Example**

There are two ways to set interval in JavaScript: by setTimeout() or setInterval() method.

1. Current Time: **<span** id="txt"**></span>**
2. **<script>**
3. window.onload=function(){getTime();}
4. function getTime(){
5. var today=new Date();
6. var h=today.getHours();
7. var m=today.getMinutes();
8. var s=today.getSeconds();
9. // add a zero in front of numbers**<10**
10. m=checkTime(m);
11. s=checkTime(s);
12. document.getElementById('txt').innerHTML=h+":"+m+":"+s;
13. setTimeout(function(){getTime()},1000);
14. }
15. //setInterval("getTime()",1000);//another way
16. function checkTime(i){
17. if (i**<10**){
18. i="0" + i;
19. }
20. return i;
21. }
22. **</script>**

# **JavaScript Number Object**

1. var x=102;//integer value
2. var y=102.7;//floating point value
3. var z=13e4;//exponent value, output: 130000
4. var n=new Number(16);//integer value by number object

## **JavaScript Number Constants**

|  |  |
| --- | --- |
| **Constant** | **Description** |
| MIN\_VALUE | returns the largest minimum value. |
| MAX\_VALUE | returns the largest maximum value. |
| POSITIVE\_INFINITY | returns positive infinity, overflow value. |
| NEGATIVE\_INFINITY | returns negative infinity, overflow value. |
| NaN | represents "Not a Number" value. |

Let's see the list of JavaScript number constants with description.

## **JavaScript Number Methods**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| toExponential(x) | displays exponential value. |
| toFixed(x) | limits the number of digits after decimal value. |
| toPrecision(x) | formats the number with given number of digits. |
| toString() | converts number into string. |
| valueOf() | coverts other type of value into number. |

# **JavaScript Boolean**

**JavaScript Boolean** is an object that represents value in two states: true or false. You can create the JavaScript Boolean object by Boolean() constructor as given below.

1. Boolean b=new Boolean(value);

The default value of JavaScript Boolean object is false.

## **JavaScript Boolean Example**

1. **<script>**
2. document.write(10**<20**);//true
3. document.write(10**<5**);//false
4. **</script>**
5. **Browser Object Model**

The **Browser Object Model** (BOM) is used to interact with the browser.

The default object of browser is window means you can call all the functions of window by specifying window or directly.

# **Window Object**

The **window object** represents a window in browser. An object of window is created automatically by the browser.

Window is the object of browser, **it is not the object of javascript**. The javascript objects are string, array, date etc.

## **Methods of window object**

#### Example of alert() in javascript

#### It displays alert dialog box. It has message and ok button.

1. **<script** type="text/javascript"**>**
2. function msg(){
3. alert("Hello Alert Box");
4. }
5. **</script>**
6. **<input** type="button" value="click" onclick="msg()"**/>**

#### Example of confirm() in javascript

It displays the confirm dialog box. It has message with ok and cancel buttons.

1. **<script** type="text/javascript"**>**
2. function msg(){
3. var v= confirm("Are u sure?");
4. if(v==true){
5. alert("ok");
6. }
7. else{
8. alert("cancel");
9. }
11. }
12. **</script>**
14. **<input** type="button" value="delete record" onclick="msg()"**/>**

**Example of prompt() in javascript**

It displays prompt dialog box for input. It has message and textfield for entering the input with ok and cancel buttons. Once you enter your input in text field and click on OK it will alert a msg in alert box.

1. **<script** type="text/javascript"**>**
2. function msg(){
3. var v= prompt("Who are you?");
4. alert("I am "+v);
6. }
7. **</script>**
9. **<input** type="button" value="click" onclick="msg()"**/>**

**Example of open() in javascript**

It displays the content in a new window.

1. **<script** type="text/javascript"**>**
2. function msg(){
3. open("http://www.javatpoint.com");
4. }
5. **</script>**
6. **<input** type="button" value="javatpoint" onclick="msg()"**/>**

|  |
| --- |
| Example of setTimeout() in javascript It performs its task after the given milliseconds.   1. **<script** type="text/javascript"**>** 2. function msg(){ 3. setTimeout( 4. function(){ 5. alert("Welcome to Javatpoint after 2 seconds") 6. },2000); 8. } 9. **</script>** 11. **<input** type="button" value="click" onclick="msg()"**/>** |

# **JavaScript History Object**

The **JavaScript history object** represents an array of URLs visited by the user. By using this object, you can load previous, forward or any particular page.

## **Property of JavaScript history object**

|  |  |  |
| --- | --- | --- |
| **No.** | **Property** | **Description** |
| 1 | length | returns the length of the history URLs. |

## **Methods of JavaScript history object**

There are only 3 methods of history object.

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | forward() | loads the next page. |
| 2 | back() | loads the previous page. |
| 3 | go() | loads the given page number. |

# **JavaScript Navigator Object**

The **JavaScript navigator object** is used for browser detection. It can be used to get browser information such as appName, appCodeName, userAgent etc.

## **Methods of JavaScript navigator object**

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | javaEnabled() | checks if java is enabled. |
| 2 | taintEnabled() | checks if taint is enabled. It is deprecated since JavaScript 1.2. |

## **Property of JavaScript navigator object**

|  |  |  |
| --- | --- | --- |
| **No.** | **Property** | **Description** |
| 1 | appName | returns the name |
| 2 | appVersion | returns the version |
| 3 | appCodeName | returns the code name |
| 4 | cookieEnabled | returns true if cookie is enabled otherwise false |
| 5 | userAgent | returns the user agent |
| 6 | language | returns the language. It is supported in Netscape and Firefox only. |
| 7 | userLanguage | returns the user language. It is supported in IE only. |
| 8 | plugins | returns the plugins. It is supported in Netscape and Firefox only. |
| 9 | systemLanguage | returns the system language. It is supported in IE only. |
| 10 | mimeTypes[] | returns the array of mime type. It is supported in Netscape and Firefox only. |
| 11 | platform | returns the platform e.g. Win32. |
| 12 | online | returns true if browser is online otherwise false. |

**Example of navigator object**

Let’s see the different usage of history object.

1. **<script>**
2. document.writeln("**<br/>**navigator.appCodeName: "+navigator.appCodeName);
3. document.writeln("**<br/>**navigator.appName: "+navigator.appName);
4. document.writeln("**<br/>**navigator.appVersion: "+navigator.appVersion);
5. document.writeln("**<br/>**navigator.cookieEnabled: "+navigator.cookieEnabled);
6. document.writeln("**<br/>**navigator.language: "+navigator.language);
7. document.writeln("**<br/>**navigator.userAgent: "+navigator.userAgent);
8. document.writeln("**<br/>**navigator.platform: "+navigator.platform);
9. document.writeln("**<br/>**navigator.onLine: "+navigator.onLine);
10. **</script>**

# **JavaScript Screen Object**

The **JavaScript screen object** holds information of browser screen. It can be used to display screen width, height, colorDepth, pixelDepth etc.

## **Property of JavaScript Screen Object**

There are many properties of screen object that returns information of the browser.

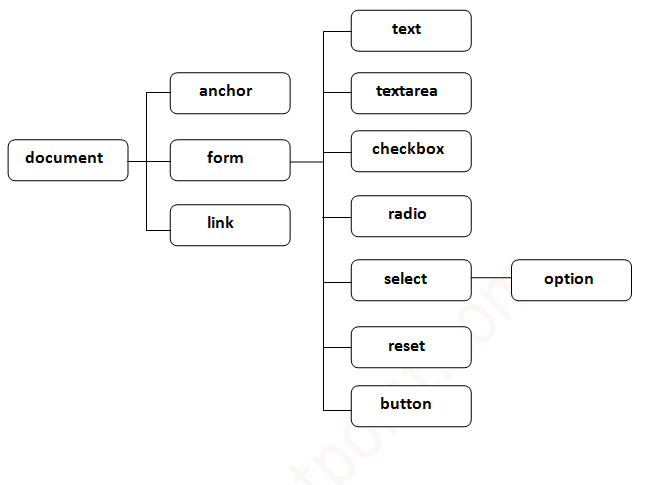
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | | **Property** | **Description** | |
| 1 | | width | returns the width of the screen | |
| 2 | | height | returns the height of the screen | |
| 3 | | availWidth | returns the available width | |
| 4 | | availHeight | returns the available height | |
| 5 | | colorDepth | returns the color depth | |
| 6 | | pixelDepth | returns the pixel depth. | |
| Example of JavaScript Screen Object Let’s see the different usage of screen object.   1. **<script>** 2. document.writeln("**<br/>**screen.width: "+screen.width); 3. document.writeln("**<br/>**screen.height: "+screen.height); 4. document.writeln("**<br/>**screen.availWidth: "+screen.availWidth); 5. document.writeln("**<br/>**screen.availHeight: "+screen.availHeight); 6. document.writeln("**<br/>**screen.colorDepth: "+screen.colorDepth); 7. document.writeln("**<br/>**screen.pixelDepth: "+screen.pixelDepth); 8. **</script>** | | |

1. **Document Object Model**

The **document object** represents the whole html document.

When html document is loaded in the browser, it becomes a document object. It is the **root element** that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.

## **Properties of document object**

Let's see the properties of document object that can be accessed and modified by the document object. 

## **Methods of document object**

We can access and change the contents of document by its methods.

The important methods of document object are as follows:

|  |  |
| --- | --- |
| **Method** | **Description** |
| write("string") | writes the given string on the doucment. |
| writeln("string") | writes the given string on the doucment with newline character at the end. |
| getElementById() | returns the element having the given id value. |
| getElementsByName() | returns all the elements having the given name value. |
| getElementsByTagName() | returns all the elements having the given tag name. |
| getElementsByClassName() | returns all the elements having the given class name. |

### **Accessing field value by document object**

1. **<script** type="text/javascript"**>**
2. function printvalue(){
3. var name=document.form1.name.value;
4. alert("Welcome: "+name);
5. }
6. **</script>**
8. **<form** name="form1"**>**
9. Enter Name:**<input** type="text" name="name"**/>**
10. **<input** type="button" onclick="printvalue()" value="print name"**/>**
11. **</form>**

# **Javascript - document.getElementById() method**

The **document.getElementById()** method returns the element of specified id.

In the previous page, we have used **document.form1.name.value** to get the value of the input value. Instead of this, we can use document.getElementById() method to get value of the input text. But we need to define id for the input field.

1. **<script** type="text/javascript"**>**
2. function getcube(){
3. var number=document.getElementById("number").value;
4. alert(number\*number\*number);
5. }
6. **</script>**
7. **<form>**
8. Enter No:**<input** type="text" id="number" name="number"**/><br/>**
9. **<input** type="button" value="cube" onclick="getcube()"**/>**
10. **</form>**

# **Javascript - document.getElementsByName() method**

The **document.getElementsByName()** method returns all the element of specified name.

1. **<script** type="text/javascript"**>**
2. function totalelements()
3. {
4. var allgenders=document.getElementsByName("gender");
5. alert("Total Genders:"+allgenders.length);
6. }
7. **</script>**
8. **<form>**
9. Male:**<input** type="radio" name="gender" value="male"**>**
10. Female:**<input** type="radio" name="gender" value="female"**>**
12. **<input** type="button" onclick="totalelements()" value="Total Genders"**>**
13. **</form>**

# **Javascript - document.getElementsByTagName() method**

The **document.getElementsByTagName()** method returns all the element of specified tag name.

1. **<script** type="text/javascript"**>**
2. function countpara(){
3. var totalpara=document.getElementsByTagName("p");
4. alert("total p tags are: "+totalpara.length);
6. }
7. **</script>**
8. **<p>**This is a pragraph**</p>**
9. **<p>**Here we are going to count total number of paragraphs by getElementByTagName() method.**</p>**
10. **<p>**Let's see the simple example**</p>**
11. **<button** onclick="countpara()"**>**count paragraph**</button>**

### **Another example of document.getElementsByTagName() method**

1. **<script** type="text/javascript"**>**
2. function counth2(){
3. var totalh2=document.getElementsByTagName("h2");
4. alert("total h2 tags are: "+totalh2.length);
5. }
6. function counth3(){
7. var totalh3=document.getElementsByTagName("h3");
8. alert("total h3 tags are: "+totalh3.length);
9. }
10. **</script>**
11. **<h2>**This is h2 tag**</h2>**
12. **<h2>**This is h2 tag**</h2>**
13. **<h3>**This is h3 tag**</h3>**
14. **<h3>**This is h3 tag**</h3>**
15. **<h3>**This is h3 tag**</h3>**
16. **<button** onclick="counth2()"**>**count h2**</button>**
17. **<button** onclick="counth3()"**>**count h3**</button>**

#### Output of the above example

## This is h2 tag

## This is h2 tag

### This is h3 tag

### This is h3 tag

### This is h3 tag

count h2 count h3

# **Javascript - innerHTML**

The **innerHTML** property can be used to write the dynamic html on the html document. It is used mostly in the web pages to generate the dynamic html such as registration form, comment form, links etc.

### **Example of innerHTML property**

In this example, we are dynamically writing the html form inside the div name having the id mylocation. We are identifing this position by calling the document.getElementById() method.

1. **<script** type="text/javascript" **>**
2. function showcommentform() {
3. var data="Name:**<input** type='text' name='name'**><br>**Comment:**<br><textarea** rows='5' cols='80'**></textarea>**
4. **<br><input** type='submit' value='Post Comment'**>**";
5. document.getElementById('mylocation').innerHTML=data;
6. }
7. **</script>**
8. **<form** name="myForm"**>**
9. **<input** type="button" value="comment" onclick="showcommentform()"**>**
10. **<div** id="mylocation"**></div>**
11. **</form>**

**Javascript - innerText**

he **innerText** property can be used to write the dynamic text on the html document. Here, text will not be interpreted as html text but a normal text.

It is used mostly in the web pages to generate the dynamic content such as writing the validation message, password strength etc.

## **Javascript innerText Example**

In this example, we are going to display the password strength when releases the key after press.

1. **<script** type="text/javascript" **>**
2. function validate() {
3. var msg;
4. if(document.myForm.userPass.value.length**>**5){
5. msg="good";
6. }
7. else{
8. msg="poor";
9. }
10. document.getElementById('mylocation').innerText=msg;
11. }
13. **</script>**
14. **<form** name="myForm"**>**
15. **<input** type="password" value="" name="userPass" onkeyup="validate()"**>**
16. Strength:**<span** id="mylocation"**>**no strength**</span>**
17. **</form>**

# **HTML/DOM events for JavaScript**

|  |  |
| --- | --- |
| **Events** | **Description** |
| onclick | occurs when element is clicked. |
| ondblclick | occurs when element is double-clicked. |
| onfocus | occurs when an element gets focus such as button, input, textarea etc. |
| onblur | occurs when form looses the focus from an element. |
| onsubmit | occurs when form is submitted. |
| onmouseover | occurs when mouse is moved over an element. |
| onmouseout | occurs when mouse is moved out from an element (after moved over). |
| onmousedown | occurs when mouse button is pressed over an element. |
| onmouseup | occurs when mouse is released from an element (after mouse is pressed). |
| onload | occurs when document, object or frameset is loaded. |
| onunload | occurs when body or frameset is unloaded. |
| onscroll | occurs when document is scrolled. |
| onresized | occurs when document is resized. |
| onreset | occurs when form is reset. |
| onkeydown | occurs when key is being pressed. |
| onkeypress | occurs when user presses the key. |
| onkeyup | occurs when key is released. |

**AJAX**

AJAX is an acronym for **Asynchronous JavaScript and XML**. It is a group of inter-related technologies like JavaScript, DOM, XML, HTML, CSS etc.

AJAX allows you to send and receive data asynchronously without reloading the web page. So it is fast.

AJAX allows you to send only important information to the server not the entire page. So only valuable data from the client side is routed to the server side. It makes your application interactive and faster.

There are too many web applications running on the web that are using ajax technology like **gmail**, **facebook**,**twitter**,**google map**, **youtube** etc.

# **Understanding Synchronous vs Asynchronous**

## **Synchronous (Classic Web-Application Model)**

A synchronous request blocks the client until operation completes i.e. browser is not unresponsive. In such case, javascript engine of the browser is blocked.  Full page is refreshed at request time and user is blocked until request completes.

## **Asynchronous (AJAX Web-Application Model)**

An asynchronous request doesn’t block the client i.e. browser is responsive. At that time, user can perform another operations also. In such case, javascript engine of the browser is not blocked. Full page is not refreshed at request time and user gets response from the ajax engine.

# **AJAX Technologies**

As describe earlier, ajax is not a technology but group of inter-related technologies. AJAX technologies includes:

* HTML/XHTML and CSS
* DOM
* XML or JSON
* XMLHttpRequest
* JavaScript

## **HTML/XHTML and CSS**

These technologies are used for displaying content and style. It is mainly used for presentation.

## **DOM**

It is used for dynamic display and interaction with data.

## **XML or JSON**

For carrying data to and from server. JSON (Javascript Object Notation) is like XML but short and faster than XML.

## **XMLHttpRequest**

For asynchronous communication between client and server. For more visit next page.

## **JavaScript**

It is used to bring above technologies together.Independently, it is used mainly for client-side validation.

# **Understanding XMLHttpRequest**

An object of XMLHttpRequest is used for asynchronous communication between client and server.

It performs following operations:

1. Sends data from the client in the background
2. Receives the data from the server
3. Updates the webpage without reloading it.

## **Properties of XMLHttpRequest object**

The common properties of XMLHttpRequest object are as follows:

|  |  |
| --- | --- |
| **Property** | **Description** |
| onReadyStateChange | It is called whenever readystate attribute changes. It must not be used with synchronous requests. |
| readyState | represents the state of the request. It ranges from 0 to 4.  **0** UNOPENED open() is not called.  **1** OPENED open is called but send() is not called.  **2** HEADERS\_RECEIVED send() is called, and headers and status are available.  **3** LOADING Downloading data; responseText holds the data.  **4** DONE The operation is completed fully. |
| reponseText | returns response as text. |
| responseXML | returns response as XML |

## **Methods of XMLHttpRequest object**

The important methods of XMLHttpRequest object are as follows:

|  |  |
| --- | --- |
| **Method** | **Description** |
| void open(method, URL) | opens the request specifying get or post method and url. |
| void open(method, URL, async) | same as above but specifies asynchronous or not. |
| void open(method, URL, async, username, password) | same as above but specifies username and password. |
| void send() | sends get request. |
| void send(string) | send post request. |
| setRequestHeader(header,value) | it adds request headers. |

# **AJAX Interview Questions**

# **What are the advantages of AJAX?**

* Quick Response
* Bandwidth utilization
* User is not blocked until data is retrieved from the server.

# **What are the disadvantages of AJAX?**

* Dependent on JavaScript
* Security issues
* Debugging is difficult

# **What are the security issues with AJAX?**

* AJAX source code is readable
* Attackers can insert script into the system

# **What are the tools for debugging AJAX applications?**

There are two most widely used tools for debugging AJAX applications.

* Firebug for Mozilla Firefox
* Fiddler for IE (Internet Explorer)

# **How can you test the AJAX code?**

JsUnit is the open source unit testing framework for client side JavaScript. It is a part of JUnit.

# **What is the difference between JavaScript and AJAX?**

JavaScript makes a request to the server and waits for the response. It consumes more bandwidth as it reloads the page.

AJAX sends a request to the server and doesn't wait for the response. It doesn't reload the page so consumes less bandwidth.

**JSON**

## **What is JSON**

JSON is an open standard for exchanging data between browser and a server. It supports data structures like object and array. So it is easy to write and read data from JSON.

* JSON stands for JavaScript Object Notation.
* JSON is lightweight data-interchange format.
* JSON is easy to read and write than XML.
* JSON is language independent.
* JSON supports array, object, string, number and values.

## **Exchanging Data**

When exchanging data between a browser and a server, the data can only be text.

JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.

We can also convert any JSON received from the server into JavaScript objects.

This way we can work with the data as JavaScript objects, with no complicated parsing and translations.

## **Sending Data**

If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server:

### **Example**

var myObj = { "name":"John", "age":31, "city":"New York" };  
var myJSON = JSON.stringify(myObj);  
window.location = "demo\_json.php?x=" + myJSON;

## **Receiving Data**

<!DOCTYPE html>

<html>

<body>

<h2>Convert a string written in JSON format, into a JavaScript object.</h2>

<p id="demo"></p>

<script>

var myJSON = '{ "name":"John", "age":31, "city":"New York" }';

var myObj = JSON.parse(myJSON);

document.getElementById("demo").innerHTML = myObj.name;

</script>

</body>

</html>

**Storing and Retrieving data**

<!DOCTYPE html>

<html>

<body>

<h2>Store and retreive data from local storage.</h2>

<p id="demo"></p>

<script>

var myObj, myJSON, text, obj;

//Storing data:

myObj = { "name":"John", "age":31, "city":"New York" };

myJSON = JSON.stringify(myObj);

localStorage.setItem("testJSON", myJSON);

//Retrieving data:

text = localStorage.getItem("testJSON");

obj = JSON.parse(text);

document.getElementById("demo").innerHTML = obj.name;

</script>

</body>

</html>

## **JSON - Evaluates to JavaScript Objects**

The JSON format is almost identical to JavaScript objects.

In JSON, keys must be strings, written with double quotes:

### **JSON**

{ "name":"John" }

In JavaScript, keys can be strings, numbers, or identifier names:

### **JavaScript**

{ name:"John" }

In JSON, string values must be written with double quotes:

### **JSON**

{ "name":"John" }

In JavaScript, you can write string values with double or single quotes:

### **JavaScript**

{ name:'John' }

## **Valid Data Types**

In JSON, values must be one of the following data types:

* a string
* a number
* an object (JSON object)
* an array
* a boolean
* null

JSON values **cannot**be one of the following data types:

* a function
* a date
* *undefined*

## **Looping an Object**

You can loop through object properties by using the for-in loop:

### **Example**

myObj = { "name":"John", "age":30, "car":null };  
for (x in myObj) {  
    document.getElementById("demo").innerHTML += x;  
}

In a for-in loop, use the bracket notation to access the property values:

### **Example**

myObj = { "name":"John", "age":30, "car":null };  
for (x in myObj) {  
    document.getElementById("demo").innerHTML += myObj[x];  
}

## **Nested JSON Objects**

<!DOCTYPE html>

<html><body>

<p>How to access nested JSON objects.</p>

<p id="demo"></p>

<script>

var myObj = {

"name":"John",

"age":30,

"cars": {

"car1":"Ford",

"car2":"BMW",

"car3":"Fiat"

}

}

document.getElementById("demo").innerHTML += myObj.cars.car2 + "<br>";

//or:

document.getElementById("demo").innerHTML += myObj.cars["car2"];

</script>

</body>

</html>

**Modifying values**

<!DOCTYPE html><html><body>

<p>How to modify values in a JSON object.</p>

<p id="demo"></p>

<script>

var myObj, i, x = "";

myObj = {

"name":"John",

"age":30,

"cars": {

"car1":"Ford",

"car2":"BMW",

"car3":"Fiat"

}

}

myObj.cars.car2 = "Mercedes";

for (i in myObj.cars) {

x += myObj.cars[i] + "<br>";

}

document.getElementById("demo").innerHTML = x;

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

Or

myObj.cars["car2"] = "Mercedes";

**Deleting values**

<!DOCTYPE html><html><body>

<p>How to delete properties of a JSON object.</p>

<p id="demo"></p>

<script>

var myObj, i, x = "";

myObj = {

"name":"John",

"age":30,

"cars": {

"car1":"Ford",

"car2":"BMW",

"car3":"Fiat"

}}

delete myObj.cars.car2;

for (i in myObj.cars) {

x += myObj.cars[i] + "<br>";}

document.getElementById("demo").innerHTML = x;

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

## **Arrays in JSON Objects**

Arrays can be values of an object property:

### **Example**

{  
"name":"John",  
"age":30,  
"cars":[ "Ford", "BMW", "Fiat" ]  
}

## **Accessing Array Values**

You access the array values by using the index number:

### **Example**

x = myObj.cars[0];

## **Looping Through an Array**

You can access array values by using a for-in loop:

### **Example**

for (i in myObj.cars) {  
    x += myObj.cars[i];  
}

Or you can use a for loop:

### **Example**

for (i = 0; i < myObj.cars.length; i++) {  
    x += myObj.cars[i];  
}

## **Nested Arrays in JSON Objects**

Values in an array can also be another array, or even another JSON object:

### **Example**

myObj = {  
    "name":"John",  
    "age":30,  
    "cars": [  
        { "name":"Ford", "models":[ "Fiesta", "Focus", "Mustang" ] },  
        { "name":"BMW", "models":[ "320", "X3", "X5" ] },  
        { "name":"Fiat", "models":[ "500", "Panda" ] }  
    ]  
 }

To access arrays inside arrays, use a for-in loop for each array:

### **Example**

for (i in myObj.cars) {  
    x += "<h1>" + myObj.cars[i].name + "</h1>";  
    for (j in myObj.cars[i].models) {  
        x += myObj.cars[i].models[j];  
    }  
}

## **Modify Array Values**

Use the index number to modify an array:

### **Example**

 myObj.cars[1] = "Mercedes";

## **Delete Array Items**

delete myObj.cars[1];

# **JSON.parse()**

A common use of JSON is to exchange data to/from a web server.

When receiving data from a web server, the data is always a string.

Parse the data with JSON.parse(), and the data becomes a JavaScript object.

## **Example - Parsing JSON**

Imagine we received this text from a web server:

'{ "name":"John", "age":30, "city":"New York"}'

Use the JavaScript function JSON.parse() to convert text into a JavaScript object:

var obj = JSON.parse('{ "name":"John", "age":30, "city":"New York"}');

Use the JavaScript object in your page:

### **Example**

<p id="demo"></p>   
  
<script>  
document.getElementById("demo").innerHTML = obj.name + ", " + obj.age;   
</script>

## **JSON From the Server**

You can request JSON from the server by using an AJAX request

As long as the response from the server is written in JSON format, you can parse the string into a JavaScript object.

### **Example**

Use the XMLHttpRequest to get data from the server:

var xmlhttp = new XMLHttpRequest();  
xmlhttp.onreadystatechange = function() {  
    if (this.readyState == 4 && this.status == 200) {  
        myObj = JSON.parse(this.responseText);  
        document.getElementById("demo").innerHTML = myObj.name;  
    }  
};  
xmlhttp.open("GET", "json\_demo.txt", true);  
xmlhttp.send();

## **Array as JSON**

When using the JSON.parse() on a JSON derived from an array, the method will return a JavaScript array, instead of a JavaScript object.

### **Example**

The JSON returned from the server is an array:

var xmlhttp = new XMLHttpRequest();  
xmlhttp.onreadystatechange = function() {  
    if (this.readyState == 4 && this.status == 200) {  
        myArr = JSON.parse(this.responseText);  
        document.getElementById("demo").innerHTML = myArr[0];  
    }  
};  
xmlhttp.open("GET", "json\_demo\_array.txt", true);  
xmlhttp.send();

## **Exceptions**

### **Parsing Dates**

Date objects are not allowed in JSON.

If you need to include a date, write it as a string.

You can convert it back into a date object later:

<!DOCTYPE html>

<html>

<body>

<h2>Convert a string into a date object.</h2>

<p id="demo"></p>

<script>

var text = '{ "name":"John", "birth":"1986-12-14", "city":"New York"}';

var obj = JSON.parse(text);

obj.birth = new Date(obj.birth);

document.getElementById("demo").innerHTML = obj.name + ", " + obj.birth;

</script>

</body>

</html>

Or, you can use the second parameter, of the JSON.parse() function, called reviver.

The reviver parameter is a function that checks each property, before returning the value.

### **Example**

Convert a string into a date, using the reviver function:

var text = '{ "name":"John", "birth":"1986-12-14", "city":"New York"}';  
var obj = JSON.parse(text, function (key, value) {  
    if (key == "birth") {  
        return new Date(value);  
    } else {  
        return value;  
    }});  
document.getElementById("demo").innerHTML = obj.name + ", " + obj.birth;

### **Parsing Functions**

Functions are not allowed in JSON.

If you need to include a function, write it as a string.

You can convert it back into a function later:

<!DOCTYPE html>

<html>

<body>

<h2>Convert a string into a function.</h2>

<p id="demo"></p>

<script>

var text = '{ "name":"John", "age":"function() {return 30;}", "city":"New York"}';

var obj = JSON.parse(text);

obj.age = eval("(" + obj.age + ")");

document.getElementById("demo").innerHTML = obj.name + ", " + obj.age();

</script>

</body>

</html>

# **JSON.stringify()**

A common use of JSON is to exchange data to/from a web server.

When sending data to a web server, the data has to be a string.

Convert a JavaScript object into a string with JSON.stringify().

## **Stringify a JavaScript Object**

Imagine we have this object in JavaScript:

var obj = { "name":"John", "age":30, "city":"New York"};

Use the JavaScript function JSON.stringify() to convert it into a string.

var myJSON = JSON.stringify(obj);

## **Exceptions**

### **Stringify Dates**

In JSON, date objects are not allowed. The JSON.stringify() function will convert any dates into strings.

### **Example**

<!DOCTYPE html><html><body>

<h2>JSON.stringify will convert any date objects into strings.</h2>

<p id="demo"></p>

<script>

var obj = { "name":"John", "today":new Date(), "city":"New York"};

var myJSON = JSON.stringify(obj);

document.getElementById("demo").innerHTML = myJSON;

</script>

</body></html>

Output

JSON.stringify will convert any date objects into strings.

{"name":"John","today":"2017-05-30T05:57:18.583Z","city":"New York"}

### **Stringify Functions**

In JSON, functions are not allowed as object values.

The JSON.stringify() function will remove any functions from a JavaScript object, both the key and the value:

### **Example**

var obj = { "name":"John", "age":function () {return 30;}, "city":"New York"};  
var myJSON = JSON.stringify(obj);  
  
document.getElementById("demo").innerHTML = myJSON;

This can be omitted if you convert your functions into strings before running the JSON.stringify() function.

### **Example**

var obj = { "name":"John", "age":function () {return 30;}, "city":"New York"};  
obj.age = obj.age.toString();  
var myJSON = JSON.stringify(obj);  
  
document.getElementById("demo").innerHTML = myJSON;

## **Features of JSON**

1. Simplicity
2. Openness
3. Self Describing
4. Internationalization
5. Extensibility
6. Interoperability

## **JSON Example**

1. {"employees":[
2. {"name":"Vimal", "email":"vjaiswal1987@gmail.com"},
3. {"name":"Rahul", "email":"rahul12@gmail.com"},
4. {"name":"Jai", "email":"jai87@gmail.com"}  ]}

**XML Example**

1. **<employees>**
2. **<employee>**
3. **<name>**Vimal**</name>**
4. **<email>**vjaiswal1987@gmail.com**</email>**
5. **</employee>**
6. **<employee>**
7. **<name>**Rahul**</name>**
8. **<email>**rahul12@gmail.com**</email>**
9. **</employee>**
10. **<employee>**
11. **<name>**Jai**</name>**
12. **<email>**jai87@gmail.com**</email>**
13. **</employee>**
14. **</employees>**

# **JSON vs XML-**

# Both are interoperable or language-independent.

|  |  |  |
| --- | --- | --- |
| **No.** | **JSON** | **XML** |
| 1) | JSON stands for JavaScript Object Notation. | XML stands for eXtensible Markup Language. |
| 2) | JSON is simple to read and write. | XML is less simple than JSON. |
| 4) | JSON is data-oriented. | XML is document-oriented. |
| 5) | JSON doesn't provide display capabilities. | XML provides the capability to display data because it is a markup language. |
| 6) | JSON supports array. | XML doesn't support array. |
| 7) | JSON is less secured than XML. | XML is more secured. |
| 8) | JSON files are more human readable than XML. | XML files are less human readable. |
| 9) | JSON supports only text and number data type. | XML support many data types such as text, number, images, charts, graphs etc. Moreover, XML offeres options for transferring the format or structure of the data with actual data. |

### **JSON Object Example**

A JSON object contains data in the form of key/value pair. The keys are strings and the values are the JSON types. Keys and values are separated by colon. Each entry (key/value pair) is separated by comma.

The **{**(curly brace) represents the JSON object.

1. {
2. "employee": {
3. "name":       "sonoo",
4. "salary":      56000,
5. "married":    **true**
6. }
7. }

### **JSON Array example**

The **[**(square bracket) represents the JSON array. A JSON array can have values and objects.

Let's see the example of JSON array having values.

1. ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]

Let's see the example of JSON array having objects.

1. [
2. {"name":"Ram", "email":"Ram@gmail.com"},
3. {"name":"Bob", "email":"bob32@gmail.com"}
4. ]

## **JSON Example 2**

1. {"menu": {
2. "id": "file",
3. "value": "File",
4. "popup": {
5. "menuitem": [
6. {"value": "New", "onclick": "CreateDoc()"},
7. {"value": "Open", "onclick": "OpenDoc()"},
8. {"value": "Save", "onclick": "SaveDoc()"}
9. ]
10. }
11. }}

The XML representation of above JSON example is given below.

1. **<menu** id="file" value="File"**>**
2. **<popup>**
3. **<menuitem** value="New" onclick="CreateDoc()" **/>**
4. **<menuitem** value="Open" onclick="OpenDoc()" **/>**
5. **<menuitem** value="Save" onclick="SaveDoc()" **/>**
6. **</popup>**
7. **</menu>**

## **JSON Object with Numbers**

JSON supports numbers in double precision floating-point format. The number can be digits (0-9), fractions (.33, .532 etc) and exponents (e, e+, e-,E, E+, E-).

1. {
2. "integer": 34,
3. "fraction": .2145,
4. "exponent": 6.61789e+0
5. }

## **JSON Object with Booleans**

JSON also supports boolean values true or false.

{

1. "first": **true**,
2. "second": **false**
3. }

## **JSON Nested Object Example**

A JSON object can have another object also. Let's see a simple example of JSON object having another object.

1. {
2. "firstName": "Sonoo",
3. "lastName": "Jaiswal",
4. "age": 27,
5. "address" : {
6. "streetAddress": "Plot-6, Mohan Nagar",
7. "city": "Ghaziabad",
8. "state": "UP",
9. "postalCode": "201007"
10. }
11. }

## **JSON Array of Numbers**

Let's see an example of JSON arrays storing number values.

1. [12, 34, 56, 43, 95]

## **JSON Array of Booleans**

Let's see an example of JSON arrays storing boolean values.

1. [**true**, **true**, **false**, **false**, **true**]

## **JSON Multidimensional Array**

We can store array inside JSON array, it is known as array of arrays or multidimensional array.

1. [
2. [ "a", "b", "c" ],
3. [ "m", "n", "o" ],
4. [ "x", "y", "z" ]
5. ]

# **JSON Comments**

JSON doesn't support comments. It is not a standard.

But you can do some tricks such as adding extra attribute for comment in JSON object, for example:

1. {
2. "employee": {
3. "name":       "Bob",
4. "salary":      56000,
5. "comments":    "He is a nice man"
6. }
7. }

## **1) Java JSON Encode**

Let's see a simple example to encode JSON object in java.

1. **import** org.json.simple.JSONObject;
2. **public** **class** JsonExample1{
3. **public** **static** **void** main(String args[]){
4. JSONObject obj=**new** JSONObject();
5. obj.put("name","sonoo");
6. obj.put("age",**new** Integer(27));
7. obj.put("salary",**new** Double(600000));
8. System.out.print(obj);
9. }}

Output:

{"name":"sonoo","salary":600000.0,"age":27}

## **Java JSON Encode using Map**

Let's see a simple example to encode JSON object using map in java.

1. **import** java.util.HashMap;
2. **import** java.util.Map;
3. **import** org.json.simple.JSONValue;
4. **public** **class** JsonExample2{
5. **public** **static** **void** main(String args[]){
6. Map obj=**new** HashMap();
7. obj.put("name","sonoo");
8. obj.put("age",**new** Integer(27));
9. obj.put("salary",**new** Double(600000));
10. String jsonText = JSONValue.toJSONString(obj);
11. System.out.print(jsonText);
12. }}

Output:

{"name":"sonoo","salary":600000.0,"age":27}

## **Java JSON Array Encode**

Let's see a simple example to encode JSON array in java.

1. **import** org.json.simple.JSONArray;
2. **public** **class** JsonExample1{
3. **public** **static** **void** main(String args[]){
4. JSONArray arr = **new** JSONArray();
5. arr.add("sonoo");
6. arr.add(**new** Integer(27));
7. arr.add(**new** Double(600000));
8. System.out.print(arr);
9. }}

Output:

["sonoo",27,600000.0]

## **Java JSON Array Encode using List**

1. **import** java.util.ArrayList;
2. **import** java.util.List;
3. **import** org.json.simple.JSONValue;
4. **public** **class** JsonExample1{
5. **public** **static** **void** main(String args[]){
6. List arr = **new** ArrayList();
7. arr.add("sonoo");
8. arr.add(**new** Integer(27));
9. arr.add(**new** Double(600000));
10. String jsonText = JSONValue.toJSONString(arr);
11. System.out.print(jsonText);
12. }}

Output:

["sonoo",27,600000.0]

## **2) Java JSON Decode**

Let's see a simple example to decode JSON string in java.

1. **import** org.json.simple.JSONObject;
2. **import** org.json.simple.JSONValue;
3. **public** **class** JsonDecodeExample1 {
4. **public** **static** **void** main(String[] args) {
5. String s="{\"name\":\"sonoo\",\"salary\":600000.0,\"age\":27}";
6. Object obj=JSONValue.parse(s);
7. JSONObject jsonObject = (JSONObject) obj;
9. String name = (String) jsonObject.get("name");
10. **double** salary = (Double) jsonObject.get("salary");
11. **long** age = (Long) jsonObject.get("age");
12. System.out.println(name+" "+salary+" "+age);
13. }
14. }

Output: sonoo 600000.0 27