Javascript

BIT-301

IT Methodologies

By:- Inderjeet Singh

Introduction

- JavaScript is a lightweight, interpreted programming language.
- It is designed for creating network-centric applications.
- It is complimentary to and integrated with Java.
- JavaScript is very easy to implement because it is integrated with HTML.
- It is open and cross-platform.

Advantages

- **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 drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Javascript Syntax- Internal

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
    document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>
<h1>JavaScript in Head</h1>
<p id="demo">A Paragraph.</p>
<button type="button" onclick="myFunction()">Try it</button>
</body>
</html>
```

Javascript Syntax- External

```
<head>
<script src=“myscript.js” language=“javascript” type=“text/javascript”></script>
</head>
```

```
//myscript.js
alert("I am an alert box!");
```

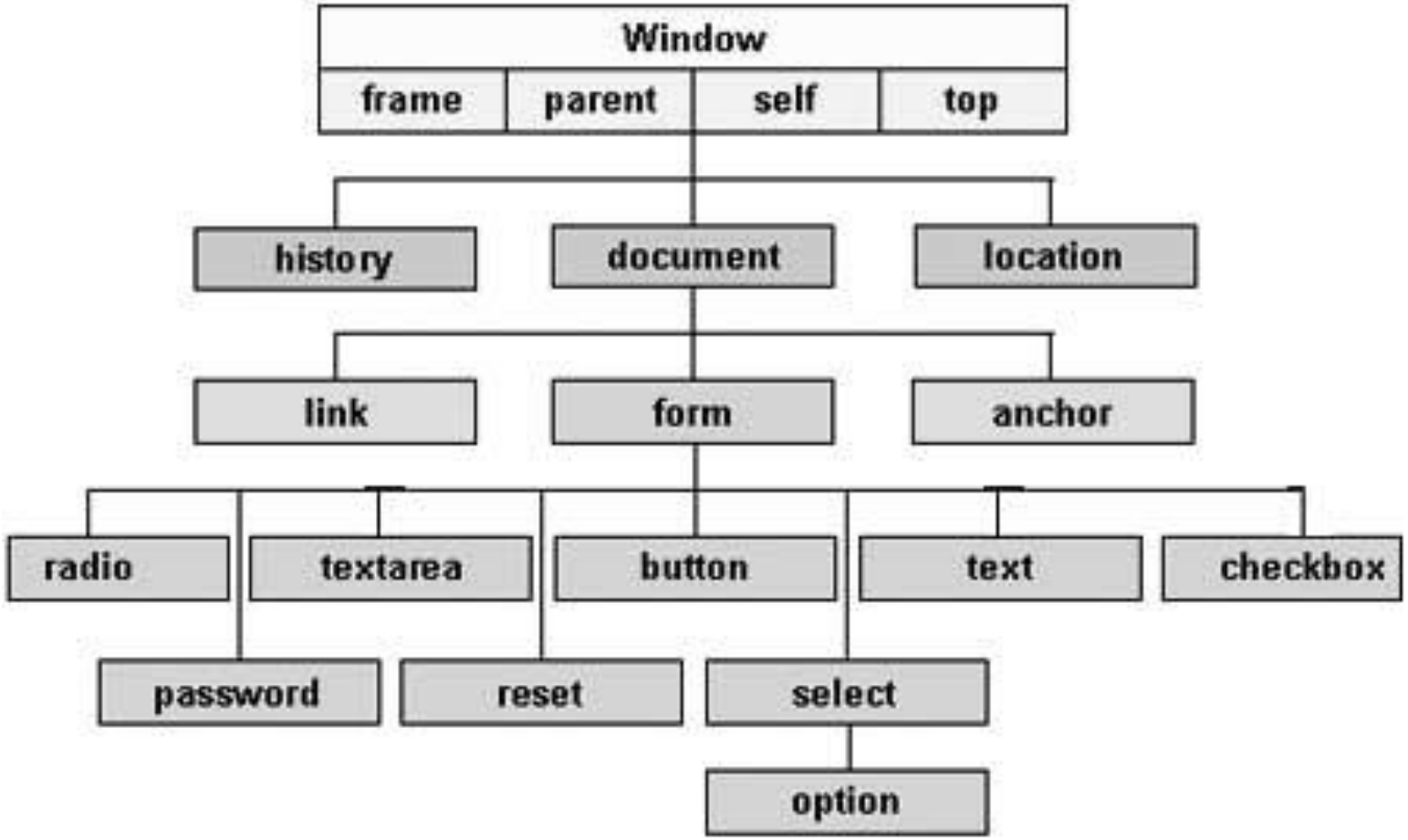
Some Conventions

- Semicolons are Optional
- Case Sensitivity
- Comments in JavaScript using // (single line) /*…*/ (multiline)
- <noscript>…</noscript> in head section executes code for javascript disabled browsers

Javascript DOM

- **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.

Javascript DOM



The HTML DOM Document Object

Finding HTML Elements

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

Changing HTML Elements

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

The HTML DOM Document Object

Adding and Deleting HTML Elements

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

Adding Event Handlers

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

Finding HTML Objects

Property	Description	DOM
document.anchors	Returns all <a> elements that have a name attribute	1
document.applets	Returns all <applet> elements (Deprecated in HTML5)	1
document.baseURI	Returns the absolute base URI of the document	3
document.body	Returns the <body> element	1
document.cookie	Returns the document's cookie	1
document.doctype	Returns the document's doctype	3
document.documentElement	Returns the <html> element	3
document.documentMode	Returns the mode used by the browser	3
document.documentURI	Returns the URI of the document	3
document.domain	Returns the domain name of the document server	1
document.domConfig	Obsolete. Returns the DOM configuration	3
document.embeds	Returns all <embed> elements	3

Finding HTML Objects

document.forms	Returns all <form> elements	1
document.head	Returns the <head> element	3
document.images	Returns all <img> elements	1
document.implementation	Returns the DOM implementation	3
document.inputEncoding	Returns the document's encoding (character set)	3
document.lastModified	Returns the date and time the document was updated	3
document.links	Returns all <area> and <a> elements that have a href attribute	1
document.readyState	Returns the (loading) status of the document	3
document.referrer	Returns the URI of the referrer (the linking document)	1
document.scripts	Returns all <script> elements	3
document.strictErrorChecking	Returns if error checking is enforced	3
document.title	Returns the <title> element	1
document.URL	Returns the complete URL of the document	1

Javascript Datatypes & Variables

- JavaScript allows you to work with three primitive data types –
  - **Numbers,** eg. 123, 120.50 etc.
  - **Strings** of text e.g. "This text string" etc.
  - **Boolean** e.g. true or false.
- JavaScript Variables

```
<script>
        var n1, n2;
        var person = "John Doe", carName = "Volvo", price = 200;
        n1=100;
        var x = 5;
        var y = 6 //Semicolon not necessary
        var z = x + y;
</script>
```

Javascript Identifier Rules

- All JavaScript **variables** must be **identified** with **unique names**.
- These unique names are called **identifiers**.
- Identifiers can be short names (like x and y), or more descriptive names (age, sum, totalVolume).
- The general rules for constructing names for variables (unique identifiers) are:
  - Names can contain letters, digits, underscores, and dollar signs.
  - Names must begin with a letter
  - Names can also begin with $ and _ (but we will not use it in this tutorial)
  - Names are case sensitive (y and Y are different variables)
  - Reserved words (like JavaScript keywords) cannot be used as names

Javascript Operators

- Let us take a simple expression **4 + 5 is equal to 9**. Here 4 and 5 are called **operands** and ‘+’ is called the **operator**. JavaScript supports the following types of operators.
  - Arithmetic Operators
  - Comparision Operators
  - Logical (or Relational) Operators
  - Assignment Operators
  - Conditional (or ternary) Operators

Arithmetic Operators

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

Assignment Operators

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

Comparison Operators

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

Logical Operators

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

Assignment Operators

```
<!DOCTYPE html>
<html>
<body>
<h1>The %= Operator</h1>
<p id="demo"></p>
<script>
var x = 10;
x %= 5;
document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

Conditional or Ternary Operators

```
<!DOCTYPE html>
<html><body>
<p>Input your age and click the button:</p>
<input id="age" value="18" />
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
    var age, voteable;
    age = document.getElementById("age").value;
    voteable = (age < 18) ? "Too young":"Old enough";
    document.getElementById("demo").innerHTML = voteable + " to vote.";
}
</script></body></html>
```

Javascript PopBoxes

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

Alert Box

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to display an alert box:</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
    alert(“Hello! This is an alert box");
}
</script>
</body>
</html>
```

Confirm Box

- A confirm box is often used if you want the user to verify or accept something.

```
<!DOCTYPE html>
<html><body>
<p>Click the button to display a confirm box.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
    var x;
    if (confirm("Press a button!") == true) {  x = "You pressed OK!";
    } else { x = "You pressed Cancel!";
    }
    document.getElementById("demo").innerHTML = x;
}
</script>
</body></html>
```

Prompt Box

- A prompt box is often used if you want the user to input a value before entering a page.

```
<!DOCTYPE html>
<html><body>
<p>Click the button to demonstrate the prompt box.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
    var person = prompt("Please enter your name", “Ronald");
    if (person != null) {
        document.getElementById("demo").innerHTML = "Hello " + person + "! How are you today?";
    }
}
</script>
</body></html>
```

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
  }

Javascript Functions

- 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 Functions

```
<!DOCTYPE html>
<html>
<body>
<p>This example calls a function to convert from Fahrenheit to Celsius:</p>
<p id="demo"></p>
<script>
function toCelsius(f) {
    return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
</script>
</body>
</html>
```

Javascript Arrays

- JavaScript arrays are used to store multiple values in a single variable.
- An array is a special variable, which can hold more than one value at a time.
- If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:
- Syntax: var *array-name* = [*item1*, *item2*, ...];
- You refer to an array element by referring to the **index number** like array-name[0]

Javascript Arrays

```
<!DOCTYPE html>
<html><body>
<p id="demo"></p>
<script>
var cars = [
   “Wolkswagen",
   "Volvo",
   "BMW"
];
document.getElementById("demo").innerHTML = cars[0];
</script>
</body></html>
```

Arrays- Objects, Property

- Arrays are a special type of objects. The **typeof** operator in JavaScript returns "object" for arrays.
- The real strength of JavaScript arrays are the built-in array properties and methods:

```
var x = cars.length;        // The length property returns the number of elements in cars
var y = cars.sort();
```

Arrays- Adding Elements

```
<!DOCTYPE html>
<html><body>
<p>The push method appends a new element to an array.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;
function myFunction() {
   fruits.push("Lemon")
   document.getElementById("demo").innerHTML = fruits;
}
</script></body>
</html>
```

Looping Array Elements

```
<!DOCTYPE html>
<html><body>
<p>The best way to loop through an array is using a standard for loop:</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
    var index;
    var text = "<ul>";
    var fruits = ["Banana", "Orange", "Apple", "Mango"];
    for (index = 0; index < fruits.length; index++) {
        text += "<li>" + fruits[index] + "</li>";
    }
    text += "</ul>";
    document.getElementById("demo").innerHTML = text;
}
</script></body></html>
```

Sorting Array Elements

```
<!DOCTYPE html>
<html><body>
<p>Click the button to sort the array.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;
function myFunction() {
    fruits.sort();
    document.getElementById("demo").innerHTML = fruits;
}
</script></body></html>
```

Javascript Condition Statement

```
<!DOCTYPE html>
<html><body>
<p>Click the button to get a time-based greeting:</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
   var greeting;
   var time = new Date().getHours();
   if (time < 10) {
      greeting = "Good morning";
   } else if (time < 20) {
      greeting = "Good day";
   } else {
      greeting = "Good evening";
   }
document.getElementById("demo").innerHTML = greeting;
}
</script></body></html>
```

Javascript Switch

```
<!DOCTYPE html>
<html><body>
<p id="demo"></p>
<script>
var day;
switch (new Date().getDay()) {
   case 0:
      day = "Sunday";
      break;
   case 1:
      day = "Monday";
      break;
  default:
        day= “Tuesday”;
}
document.getElementById("demo").innerHTML = "Today is " + day;
</script></body></html>
```

Javascript For Loop

```
<!DOCTYPE html>
<html><body>
<p>Click the button to loop through a block of code five times.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
    var text = "";
    var i;
    for (i = 0; i < 5; i++) {
        text += "The number is " + i + "<br>";
    }
    document.getElementById("demo").innerHTML = text;
}
</script></body></html>
```

Javascript While Loop

```
<!DOCTYPE html>
<html><body>
<p>Click the button to loop through a block of code as long as i is less than 10.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
   var text = "";
   var i = 0;
   while (i < 10) {
      text += "<br>The number is " + i;
      i++;
   }
   document.getElementById("demo").innerHTML = text;
}
</script></body></html>
```

Javascript do-while Loop

```
<!DOCTYPE html>
<html><body>
<p>Click the button to loop through a block of code as long as i is less than 10.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
   var text = ""
   var i = 0;
   do {
      text += "<br>The number is " + i;
      i++;
   }
   while (i < 10)
   document.getElementById("demo").innerHTML = text;
}
</script></body></html>
```

Javascript Events

- An HTML event can be something the browser does, or something a user does.
- Here are some examples of HTML events:
  - An HTML web page has finished loading
  - An HTML input field was changed
  - An HTML button was clicked

Javascript Events

- Syntax:
  - *<some-HTML-element some-event='**some JavaScript**'>*

```
<!DOCTYPE html>
<html>
<body>
<button onclick="getElementById('demo').innerHTML=Date()">The time is?</button>
<p id="demo"></p>
</body>
</html>
```

Javascript 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

Form Validations

```
<!DOCTYPE html>
<html><head>
<script>
function validateForm() {
    var x = document.forms["myForm"]["fname"].value;
    if (x == null || x == "") {
        alert("Name must be filled out");
        return false;  }
}
</script>
</head>
<body>
<form name="myForm" action="demo_form.asp"
onsubmit="return validateForm()" method="post">
Name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
</body></html>
```

Limitations

- 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.