Java Script

JavaScript

JavaScript is the programming language of the HTML and the Web. JavaScript is the premier client-side scripting language used today on the Web. It's widely used in tasks ranging from the validation of form data to the creation of complex user interfaces. 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 is one of the **3 languages** all web developers **must** learn:

1. **HTML** to define the content of web pages
2. **CSS** to specify the layout of web pages
3. **JavaScript** to program the behavior of web pages

Client side Scripting

Client side programming has mostly to do with the user interface, with which the user interacts. In web developing it's the browser, in the user's machine, that runs this code, and is mainly done in javascript, flash, etc. This code must run in a variety of browsers. Its main tasks are validating input animation manipulating UI elements applying styles some calculations are done when you don't want the page to refresh so often. In Client side scripting, Script file usually download on client system and client browser compiled this script file and run it.

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. 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.

Server side Scripting

Server side programming has to do with generating dynamic content. It runs on servers. Many of these servers are "headless". Most web pages are not static, they search a database in order to show the user updated personalized information. This sides interacts with the back end, like say, the database. This programming can be done in a lot of languages: PHP Java and jsp asp Perl Python Ruby on Rails, etc. This code has to do which: Querying the database Encode the data into html Insert and update information onto the database Business rules and calculations

In server side scripting, when user request page for display then script run on server and generate dynamic HTML file and send this HTML file to user.

Syntax of JavaScript

There is a flexibility given to include JavaScript code anywhere in an HTML document. However the most preferred ways to include JavaScript in an HTML file are as follows:

- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in an external file and then include in <head>...</head> section.

JavaScript in <head>...</head> Section

```
<html>
<head>
<script type="text/javascript">
<!--
    document.write("Hello World Wide Web");
}
//-->
</script>
</head>
<body>
</body>
</html>
```

JavaScript in <body>...</body> Section

```
<html>
<head>
</head>
<body>
<script type="text/javascript"> <!--
    document.write("Hello World Wide Web");
//-->
</script>
<p>This is web page body </p>
</body>
</html>
```

JavaScript in External File

```
<html>
<head>
        <script type="text/javascript" src="filename.js" ></script>
</head>
<body>
.......
</body>
</html>
```

The script tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files. The JavaScript code is now in the filename.js file

JavaScript 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 also defines two trivial data types, **null** and **undefined,** each of which defines only a single value. 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 andJavaScript takes care of it automatically

Variables are declared with the **var** keyword

```
<script type="text/javascript">
<!--
        var name = "Ali";
        var money;
        money = 500.75;
//-->
</script>
```

Operators

JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Arithmetic Operator	Comparison operator	Logical operator	Assignment operator	Conditional operator
+ Add	== equal to	&& AND	== assignment	? :
- Subtraction	!= not equal to	|| OR	+= add assign	
* Multiply	< less than	! NOT	-+ sub assign	
/ Division	> Greater than		*= multiple assign	
% Modulus	< less/equal to		/= divide assign	
++ increment	>greater/equal			
-- decrement				

Example operators

```
<html>

<body>

<script type="text/javascript"> <!--

        var a = 33;

        var b = 10;

        var c = "Test";

        var linebreak = "<br />";
```

```
            document.write("a + b = ");
            result = a + b;
            document.write(result);
            document.write(linebreak);

            document.write("a - b = ");
            result = a - b;
            document.write(result);
             document.write(linebreak);


            document.write("(a == b) => ");
             result = (a == b);
            document.write(result);
            document.write(linebreak);

            document.write("(a > b) => ");
             result = (a > b);
            document.write(result);
            document.write(linebreak);

            document.write("(a != b) => ");
             result = (a != b);
            document.write(result);
            document.write(linebreak);

            document.write ("((a > b) ? 100 : 200) => ");
             result = (a > b) ? 100 : 200;
            document.write(result);
            document.write(linebreak);

    </script>


            <p>Set the variables to different values and then try...</p>

    </body>

    </html>
```

Output

a + b = 43

a - b = 23

(a == b) => false

(a > b) => true

(a != b) => false

((a > b) ? 100 : 200) => 200

typeof Operator

The **typeof** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand. The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation

Conditional Statements

JavaScript supports the following conditional statements

- if statement
- if...else statement
- if...else if... statement
- switch case

Syntax

```
if (expression){
    Statement(s) to be executed if expression is true
}

if (expression){
    Statement(s) to be executed if expression is true
}else{
    Statement(s) to be executed if expression is false
}


if (expression 1){
    Statement(s) to be executed if expression 1 is true
}else if (expression 2){
    Statement(s) to be executed if expression 2 is true
}else if (expression 3){
    Statement(s) to be executed if expression 3 is true
```

```
}else{
    Statement(s) to be executed if no expression is true
}

switch (expression)
{
   case condition 1: statement(s)
                          break;
   case condition 2: statement(s)
                          break;

   case condition n: statement(s)
                          break;
   default: statement(s)
}
```

Example

```
<script type="text/javascript"> <!--
var age = 15;

if( age > 18 ){
    document.write("<b>Qualifies for driving</b>"); }else{
    document.write("<b>Does not qualify for driving</b>");
}
//--> </script>
```

Looping Structure

JavaScript supports the following necessary loops for ease of the programming.

- **While loop**
  - ***The most basic loop in JavaScript is the while loop. The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates***
- **Do while loop**

- ***The do...while loop is similar to the while loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false.***

- **For loop**
  - ***The 'for' loop is the most compact form of looping. It includes the following three important parts:***
    - The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
    - The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
    - The **iteration statement** where you can increase or decrease your counter.
- **For in loop**
  - ***The for...in loop is used to loop through an object's properties. In each iteration, one property from object is assigned to variablename and this loop continues till all the properties of the object are exhausted*.**

Syntax

```
while (expression){
    Statement(s) to be executed if expression is true
}

do{
    Statement(s) to be executed;
} while (expression);

for (initialization; test condition; iteration statement){
        Statement(s) to be executed if test condition is true
}
for (variablename in object){
  statement or block to execute
```

```
}
```

Example

```
<script type="text/javascript">
<!--
        var count = 0;
        document.write("Starting Loop "+ "<br />");
        while (count < 10){
           document.write("Current Count : " + count + "<br />");
           count++;
        }
        document.write("Loop stopped!" + "<br />");
//-->

</script>
```

Output

Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!

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

Function Definition

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

```
function functionname(parameter-list)
{
        statements
}
```

Example

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

This defines a function named sayHello

Function call

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

```
<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>
```

This piece of code outputs a button and when it is clicked the **sayHello function** gets invoked displaying the text "**Hello there**!"on the screen.

Function Parameters

When a function is called parameters can be passed to the called function. The procedure for passing parameters is described in the following example

```
<html>
<head>
        <script type="text/javascript"> function
        concatenate(first, last)
        {
            var full;
            full = first + last;
            return full;
        }
        function secondFunction()
        {
            var result;
            result = concatenate('bilal', 'ahmad');
            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>
</html>
```

In this example button is displayed in the browser, when that is clicked the function secondFunction() gets called, inside this function concatenate() function is called with two arguments . it returns the concatenated string of the two passed arguments. It also shows that to return values from the function the keyword return is used. It can also be used to return from the function at any time without even returning any value.

JavaScript Objects

JavaScript is an Object Oriented Programming (OOP) language. Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object, otherwise the attribute is considered a property.

Object Properties

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The syntax for adding a property to an object is:

```
objectName.objectProperty = propertyValue
```

Object Methods

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a function and a method – at a function is a standalone unit of statements and a method is attached to an object and can be referenced by the **this** keyword. Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

User defined Objects

All user-defined objects and built-in objects are descendants of an object called **Object**. To define an object in JavaScript require the use of keyword **new operator**

new operator

The **new** operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method

```
var employee = new Object();

var books = new Array("C++", "Perl", "Java");

var day = new Date("August 15, 1947");
```

Three objects are defined employee, books, and day, of type Object, Array, Date. the constructor methods are Object(), Array(), and Date(). These constructors are built-in JavaScript functions.

Object constructor

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called **Object()** to build the object. The return value of the **Object()** constructor is assigned to a variable. The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the **var** keyword.

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

    var book = new Object(); // Create the object
    book.subject = "JavaScript"; // Assign properties to the object
    book.author = Thomas Powell";
</script>

<script type="text/javascript">

   document.write("Book name is : "+book.subject + "<br>");
   document.write("Book author is : "+book.author + "<br>");
</script>
```

In this example a single object named book is created using the Object constructor. Two properties are added to the object on the fly, and these properties can be the accessed.

User defined object constructor

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

        function book(title, author){
        // this function is used to create objects
            this.title = title;
            this.author = author;
        }
</script>

<script type="text/javascript">
    var myBook = new book("Java", "Herbert"); //object is created
    document.write("Book title is : " + myBook.title + "<br>");
    document.write("Book author is : " + myBook.author + "<br>");
</script>
```

The function book serves as the constructor function to define an object.
There is other way of defining an object in JavaScript.

Syntax

```
var obj = { }
```

Example

```
var box = {
        height: 300
        width : 200
        length: 500
}
```

Predefined Objects in JavaScript

JavaScript has a number of built-in objects that are highly useful.

- **Array**
- **Boolean**
- **Date**
- **Math**
- **Number**
- **String**
- **regExp**

Array Object

The **Array** object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Syntax

var fruits = new Array("apple", "orange", "mango");

Example

```
<script type="text/javascript">
    var arr = new Array(water, air, food);
    for(str in arr)
    {
     Document.write(str);
    }
}
```

Some of the most important properties Array Object are length: number of elements in array and index: gives the index of the element if it is in array

Example

```
<script type="text/javascript">
        var arr = new Array( 10, 20, 30 );
         document.write("arr.length is:" + arr.length);
</script>
```

Common methods associated with array Object are as

Method	Description
concat()	Returns a new array comprised of this array joined with other array(s) and/or value(s)
forEach()	Calls a function for each element in the array.
indexOf()	Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
join()	Joins all elements of an array into a string.
pop()	Removes the last element from an array and returns that element.
push()	Adds one or more elements to the end of anarray and returns the new length of the array.
reverse()	Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
sort()	Sorts the elements of an array.
toString()	Returns a string representing the array and its elements.

Example

```
<script type="text/javascript">
        var alpha = ["a", "b", "c"];
        var numeric = [1, 2, 3];
        var alphaNumeric = alpha.concat(numeric);
```

```
        document.write("alphaNumeric : " + alphaNumeric );
</script>

<script type="text/javascript">
        var arr = new Array("First","Second","Third");
        var str = arr.join(); document.write("str : " + str );
</script>

<script type="text/javascript">
      var numbers = [1, 4, 9];
      var element = numbers.pop();
      document.write("element is : " + element );
      var element = numbers.pop();
      document.write("<br />element is : " + element );
</script>

<script type="text/javascript">
        var arr = [0, 1, 2, 3].reverse();
        document.write("Reversed array is : " + arr );
</script>

<script type="text/javascript">
        var arr = new Array("orange", "mango", "banana", "sugar");
        var sorted = arr.sort();
        document.write("Sorted string is : " + sorted );
</script>
```

Output

alphaNumeric : a,b,c,1,2,3
str : First,Second,Third
element is : 9
element is : 4
Reversed array is : 3,2,1,0
Sorted string is : banana,mango,orange,sugar

Date Object

The Date object is a datatype built into the JavaScript language. Date objects are created with the **new Date()**.Once a Date object is created, a number of methods allow you to operate on it.

Most methods simply allow you to get and set the year, month, day, hour, minute, second, and millisecond fields of the object.

Syntax

new Date( )

new Date(milliseconds)

new Date(datestring)

- **No Argument:** With no arguments, the Date() constructor creates a Date object set to the current date and time.
- **milliseconds:** When one numeric argument is passed, it is taken as the internal numeric representation of the date in milliseconds, as returned by the getTime() method. For example, passing the argument 5000 creates a date that represents five seconds past midnight on 1/1/70.
- **datestring:** When one string argument is passed, it is a string representation of a date, in the format accepted by the **Date.parse()** method.

Common Date Methods

Method	Description
Date()	Returns today's date and time
getDate()	Returns the day of the month for the specified date
getDay()	Returns the day of the week for the specified date
getFullYear()	Returns the year of the specified date
getHours()	Returns the hour in the specified date
getMinutes()	Returns the minutes in the specified date
getMilliseconds()	Returns the milliseconds in the specified date
getSeconds()	Returns the seconds in the specified date
getTime()	Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC.
setDate()	Sets the day of the month
setTime()	Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.

Example

```
<script type="text/javascript">
  var dt = Date();
```

```
  document.write("Date and Time : " + dt );
</script>

<script type="text/javascript">
  var dt = new Date("October 15, 2016 23:15:00");
  document.write("getDate() : " + dt.getDate()+"<br />" );
  document.write("getDay() : " + dt.getDay()+"<br />");
  document.write("getFullYear() : " + dt.getFullYear()+"<br />" );
  document.write("getHours() : " + dt.getHours()+"<br />" );
  document.write("getMinutes() : " + dt.getMinutes()+"<br />" );
  document.write("getMonth() : " + dt.getMonth()+"<br />" );
</script>
```

Output

Date and Time : Wed Oct 20 2016 15:00:57 GMT+0530 (India Standard Time)

getDate() : 25
getDay: 6
getFullYear: 2016
getHours: 23
getMinutes: 15
getMonth: 9

Math Object

The **math** object provides you properties and methods for mathematical constants and functions. **Math** is not a constructor. All the properties and methods of Math are static and can be called by using **Math** as an object without creating it.

Syntax

var pi = Math.PI;

var sin = Math.sin(30);

Most commonly used Math properties

Property	Description
E	Euler's constant and the base of natural logarithms, approximately 2.718.
LOG2E	Base 2 logarithm of E, approximately 1.442.
LOG10E	Base 10 logarithm of E, approximately 0.434.
LN2	Natural logarithm of 2, approximately 0.693.
LN10	Natural logarithm of 10, approximately 2.302.
SQRT2	Square root of 2, approximately 1.414.
PI	Ratio of the circumference of a circle to its diameter, approximately 3.14159.

Math Object has a large number of Methods only a few are listed her

Method	Description
abs()	Returns the absolute value of a number.
sqrt()	Returns the square root of a number.
ceil()	Returns the smallest integer greater than or equal to a
floor()	Returns the largest integer less than or equal to a number
log()	Returns the natural logarithm (base E) of a number.
pow()	Returns base to the exponent power, that is, base exponent
random()	Returns a pseudo-random number between 0 and 1.
sin()	Returns the sine of a number.
cos()	Returns the cos of a number.
tan()	Returns the tangent of a number.

Example

```
    <script type="text/javascript">
      var value = Math.E;
      document.write("Euler Value is :" + value+"<br/>");

      value = Math.PI;
      document.write("PI Value is : " + value+"<br/>");

      value = Math.SQRT2;
      document.write("Sqrt of 2 is : " + value+"<br/>");

    </script>

 <script type="text/javascript">
      var value;

      value = Math.abs(-1);
      document.write("Absolute : " + value +"<br/>");

      value = Math.abs(20);
      document.write("Absolute : " + value +"<br/>");

      value = Math.pow(7, 2);
      document.write("power of 7^2 : " + value +"<br/>");

      value = Math.random( );
      document.write("Random value : " + value +"<br/>");


      value = Math.sqrt( 81 );
      document.write("Square Root 81 : " + value+"<br/>" );

 </script>
```

Output

Euler Value is :2.718281828459045

PI Value is : 3.141592653589793

Sqrt of 2 is : 1.4142135623730951

Absolute : 1

Absolute : 20

Power of 7^2 : 49

Random value : 0.8729205637

Square Root 81: 9

String

The **String** object lets you work with a series of characters; it wraps Javascript's string primitive data type with a number of helper methods.

Syntax

var str = new String(string);

The length of the string can bec checked with the length property of the string object.

Example

```
<script type="text/javascript">
      var str = new String( "This is string" );
      document.write("str.length is:" + str.length);
</script>
```

There are a number of methods available in String object. Some are listed here

Method	Description
charAt()	Returns the character at the specified index.
concat()	Combines the text of two strings and returns a new string.
replace()	Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
search()	Executes the search for a match between a regular expression and a specified string.
split()	Splits a String object into an array of strings by separating the string into substrings
substr()	Returns the characters in a string beginning at the specified location through the specified number of characters.
toLowerCase()	Returns the calling string value converted to lower case.
toUpperCase()	Returns the calling string value converted to uppercase.
toString()	Returns a string representing the specified object.
match()	Used to match a regular expression against a string.
slice()	Extracts a section of a string and returns a new string.

Example

```
<script type="text/javascript">
    var str = new String( "This is string" );
    document.writeln("str.charAt(0) is:" + str.charAt(0)  + “<br />”);
    document.writeln("str.charAt(1) is:" + str.charAt(1)   + “<br />”);
</script>

<script type="text/javascript">
     var re = /apples/gi;
     var str = "Apples are round, and apples are juicy.";

     if ( str.search(re) == -1 ){
           document.write("Does not contain Apples" );
     }else{
           document.write("Contains Apples" );
     }
</script>

<script type="text/javascript">

     var str = "Apples are round, and apples are juicy.";
     document.write("(1,2): " + str.substring(1,2) +"<br/>");
     document.write("(0,10): " + str.substring(0, 10) +"<br/>");
     document.write("(5): " + str.substring(5) +"<br/>");

</script>
```

Output

str.charAt(0) is:T
str.charAt(1) is:h

Contains Apples

(1,2): p
(0,10): Apples are
(5): s are round, and apples are juicy.

regExp

regExp short for regular expression is an object that describes a pattern of characters. The JavaScript **RegExp** class represents regular expressions, and both String and **RegExp** define methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.

A regular expression could be defined with the **RegExp()** constructor

Syntax

var pattern = new RegExp(pattern, attributes);

or

var pattern = /pattern/attributes;

- **pattern:** A string that specifies the pattern of the regular expression or another regular expression.
- **attributes:** An optional string containing any of the "g", "i", and "m" attributes that specify global, case-insensitive, and multiline matches, respectively

Brackets

Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

Expression	Description
[...]	Any one character between the brackets.
[^...]	Any one character not between the brackets.
[0-9]	It matches any decimal digit from 0 through 9.
[a-z]	It matches any character from lowercase a through lowercase z
[A-Z]	It matches any character from uppercase A through uppercase Z
[a-Z]	It matches any character from lowercase a through uppercase Z

Quantifiers

The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character has a specific connotation. The +, *, ?, and $ flags all follow a character sequence.

Expression	Description
p+	It matches any string containing at least one p.
p*	It matches any string containing zero or more p's.
p?	It matches any string containing one or more p's.
p{N}	It matches any string containing a sequence of N p's
p{2,3}	It matches any string containing a sequence of two or three p's.
p{2, }	It matches any string containing a sequence of at least two p's.
p$	It matches any string with p at the end of it.
^p	It matches any string with p at the beginning of it.

Examples of RegExp

Expression	Description
[^a-zA-Z]	It matches any string not containing any of the characters ranging from a through z and A through Z.
p.p	It matches any string containing p, followed by any character, in turn followed by another p.
^.{2}$	It matches any string containing exactly two characters.
(.*)	It matches any string enclosed within and .
p(hp)*	It matches any string containing a p followed by zero or more instances of the sequence hp

RegExp has two main methods

exec() : it searches for a match in the string parameter, and returns the string if found, null if not found

test(): it simply test and return true or false

Examples

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

    var str = "Javascript is an interesting scripting language";

    var re = new RegExp( "script", "g" );
// checks for script for multiple occurances
    var result = re.exec(str);
    document.write("Test 1 - returned value : " +  result);

    re = new RegExp( "pushing", "g" );
//  checks for pushing for multiple occurances
    var result = re.exec(str);
    document.write("<br />Test 2 - returned value : " + result);
 </script>
```

Output

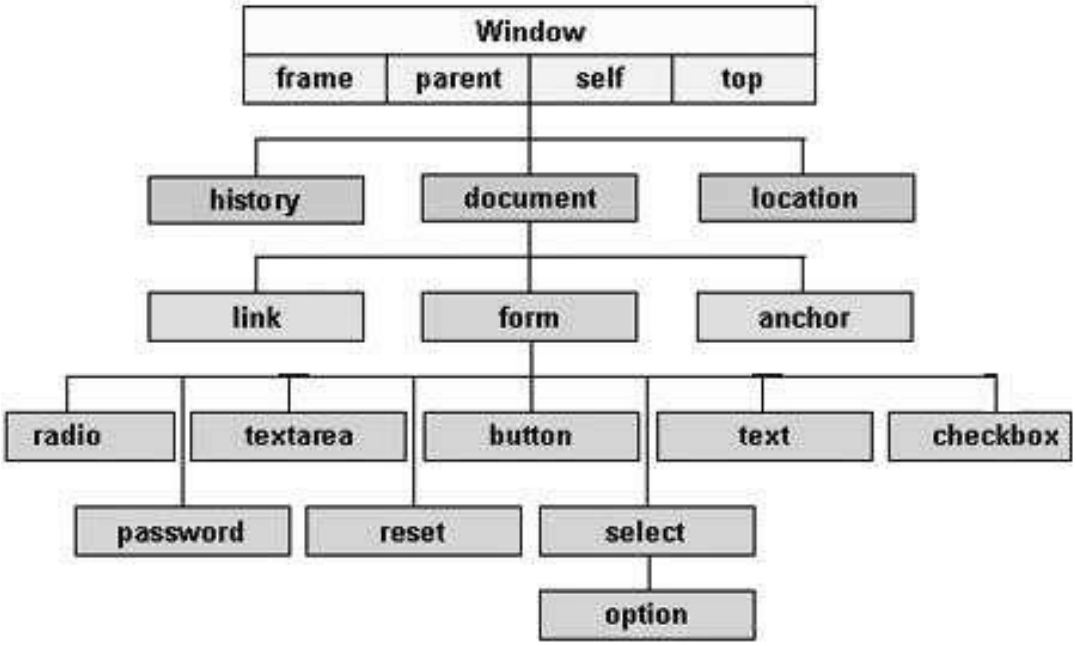
Test 1 - returned value : script
Test 2 - returned value : null

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.

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.



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.

Document Properties in W3C DOM

This model supports all the properties available in Legacy DOM with additional document properties which can be accessed using W3C DOM

Property	Description
anchors[]	An array of Anchor objects, one for each anchor that appears in the document Ex: document.anchors[0], document.anchors[1]
applets[]	An array of Applet objects, one for each applet that appears in the document Ex: document.applets[0], document.applets[1] and so on
Cookie	A string valued property with special behavior that allows the cookies associated with this document to be queried and set. Ex: document.cookie
Domain	A string that specifies the Internet domain the document is from. Ex: document.domain
forms[]	An array of Form objects, one for each HTML form that appears in the document. Ex: document.forms[0], document.forms[1] and so on
images[]	An array of Image objects, one for each image that is embedded in the document with the HTML <img> tag. Ex: document.images[0], document.images[1] and so on
links	It is a document link array.
referrer	A read-only string that contains the URL of the document, if any, from which the current document was linked Ex: document.referrer
URL	A read-only string that specifies the URL of the document. Ex: document.URL
Body	A reference to the Element object that represents the <body> tag of this document Ex: document.body

Document Methods in W3C DOM

This model supports all the methods available in Legacy DOM with additional document properties which can be accessed using W3C DOM

Method	Description
getElementById(id)	Returns the Element of this document that has the specified value for its id attribute, or null if no such Element exists in the document. Ex: document.getElementById(id)
getElement ByName(name)	Returns an array of nodes of all elements in the document that have a specified value for their name attribute. If no such elements are found, returns a zero-length array Ex: document.getElementsByName(name)

Example

```
<html>
<head>
<title> Document Title </title>
<script type="text/javascript"> <!--

      function myFunc()
      {
         document.getElementsById("heading1").style.color = 'red';
      }
//-->
</script>
</head>
<body>

<h1 id="heading1">This is Main Heading</h1>
 <p>Click the following to see the result:</p>

<form id="form1" name="FirstForm">
      <input type="button" value="Click Me" onclick="myFunc();" />
</form>

</body>
</html>
```

This example displays a single button in the browser with “Click Me” on it. When that button is pressed it changes the color of the heading “This is Main Heading”.

Event

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.

Some of the most commonly used event types are as

- **onClick Event**
- **onSubmit Event**
- **onMouseOver Event**
- **onMouseOut Event**

onClick Event

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>
```

This example simply display a button on the screen and when that is clicked “Hello World” gets displayed in the browser. The onClicked event is triggered in this case on the button with text “Say Hello”.

onsubmit Event

onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.

Example

```
<html>
<head>
<script type="text/javascript">
<!--
      function validation()
      {
         all validation goes here
         .........
         return either true or false
      }
//-->
</script>
</head>

<body>
<form method="POST" onsubmit="return validate()">


     <input type="submit" value="Submit" />
</form>
</body>
</html>
```

In this example when submit button is clicked validate function is called before submitting the form data to the server. Submission of form data to server is dependent on the validate function.

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

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> TESTING MOUSE</h2>
</div>

</body>
</html>
```

In this example when the mouse moves over the text " TESTING MOUSE" the function gets called and the text Mouse Over is displayed in the browser similarly with the when mouse leaves the text. In real life development effects to the text or to page will happen.