JavaScript

JavaScript is *an object-based scripting language* that is lightweight and cross-platform.

JavaScript is not compiled but interpreted. The JavaScript interpreter (embedded in browser) is responsible to translate the JavaScript code.

It is untyped , multi-paradigm, functional, event driven.

It is interpreted by browser's java script engine.

Different browser's had different engines.

Disadvantages of JavaScript

- Security. Because the code executes on the users' computer, in some cases it can be exploited for malicious purposes. This is one reason some people choose to disable JavaScript.
- JavaScript is sometimes interpreted differently by different browsers.

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.

Limitations of JavaScript

- 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.

What is JavaScript ?

Javascript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as **LiveScript,** but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name **LiveScript**. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

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

Client-side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.

JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

Advantages of JavaScript

The merits of using JavaScript are –

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

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 – Syntax

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>**.

The script tag takes two important attributes −

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

So your JavaScript segment will look like −

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

Whitespace and Line Breaks

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

Semicolons are Optional

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

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

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

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

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

Case Sensitivity

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

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

3 Places to put JavaScript code

1. Between the body tag of html
2. Between the head tag of html
3. In .js file (external 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 –

1. Script in <head>...</head> section.
2. Script in <body>...</body> section.
3. Script in <body>...</body> and <head>...</head> sections.
4. Script in an external file and then include in <head>...</head> section.

JavaScript in <head>...</head> section

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows –

```
<html>
```

```
 <head>
 
 
   <script type="text/javascript">
     <!--
       function sayHello() {
         alert("Hello World")
       }
     //-->
   </script>
 
 
 </head>
 
 
 <body>
   <input type="button" onclick="sayHello()" value="Say Hello" />
 </body>
 
 
</html>
```

JavaScript in <body>...</body> section

If you need a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document. In this case, you would not have any function defined using JavaScript. Take a look at the following code.

```
<html>
```

```
  <head>
  </head>

  <body>

    <script type="text/javascript">
      <!--
        document.write("Hello World")
      //-->
    </script>

    <p>This is web page body </p>

  </body>
</html
```

JavaScript in <body> and <head> Sections

You can put your JavaScript code in <head> and <body> section altogether as follows –

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

  <body>
    <script type="text/javascript">
      <!--
        document.write("Hello World")
      //-->
    </script>

    <input type="button" onclick="sayHello()" value="Say Hello" />

  </body>
</html>
```

JavaScript in External File

As you begin to work more extensively with JavaScript, you will be likely to find that there are cases where you are reusing identical JavaScript code on multiple pages of a site.

You are not restricted to be maintaining identical code in multiple HTML files. The **script** tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.

Here is an example to show how you can include an external JavaScript file in your HTML code using **script** tag and its **src** attribute.

```
<html>
```

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

  <body>
    .......
  </body>
</html>
```

To use JavaScript from an external file source, you need to write all your JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above.

For example, you can keep the following content in **filename.js** file and then you can use **sayHello** function in your HTML file after including the filename.js file.

```
function sayHello() {
  alert("Hello World")
}
```

Note:-

Java script code must be inserted between script tags <script></script> .

Script can be placed anywhere in HTML document body or head.

Eg:- <script type=”text/javascript”></script>

The type attribute is not required because java script is default scripting language html.

If you write a script in head tag than firstly so that script will execute first after that html code executed.

Basics Of Javascript

Java script is programming language

Java script statements are seprated by semicolons.

Java script statements are composed of: Values,Operators,Expressions,Keywords and Comments.
Absolutely constants are called literals.

Example:-
245
2.45
“hello”

JavaScript Display

JavaScript can "display" data in different ways:

Writing into an HTML element, using **innerHTML**.

Writing into the HTML output using **document.write()**.

Writing into an alert box, using **window.alert()**.

Writing into the browser console, using **console.log()**.

Using innerHTML

To access an HTML element, JavaScript can use the **document.getElementById(id)** method.

Example:-

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

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

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

Using document.write()

Example:-

```
<script>
document.write(5 + 6);
</script>
```

Note:-

Using document.write() after an HTML document is fully loaded, will delete all existing HTML:

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<button onclick="document.write(5 + 6)">Try it</button>

</body>
</html>
```

Using window.alert()

Example:-

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
```

```
</script>

</body>
</html>
```

Using console.log()

you can use the **console.log()** method to display data.

Example:-

```
<!DOCTYPE html>
<html>
<body>

<script>
console.log(5 + 6);
</script>

</body>
</html>
```

JavaScript comments

The **JavaScript comments** are meaningful way to deliver message. It is used to add information about the code, warnings or suggestions so that end user can easily interpret the code.

The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser.

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.

SERVER SIDE JAVA SCRIPT

Yes, since the advent of **Node.js** and its frameworks you can use Javascript in both Client side and server side. If you ever heard of MEAN Stack (trending among web developers) then it is the same stack which is completely written in Javascript.

Warning for Non-JavaScript Browsers

If you have to do something important using JavaScript, then you can display a warning message to the user using **<noscript>** tags.

Example:-

```
<noscript>
   Sorry...JavaScript is needed to go ahead.
</noscript>
```

if the user's browser does not support JavaScript or JavaScript is not enabled, then the message from </noscript> will be displayed on the screen.

Note:-

A **computer program** is a list of "instructions" to be "executed" by the computer.

In a programming language, these program instructions are called **statements**.

Comments in JavaScript

JavaScript supports both C-style and C++-style comments, Thus –

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

JavaScript Variable

A **JavaScript variable** is simply a name of storage location.

Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword .

Note:-

Var is a keyword.

Used to store data temporarily.

Variables do not have specific data type (we use variant keyword for variable declaration).

Note: -

Concept of garbage value is not exist in java script like c,c++ so you can decalare a variable like this without assigning a value.

eg:- var x;

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.

Note:-

concatenation between two operands there must be one operand is string.

Eg:-

var a=”2”+4+5;

var name=”hello”+” “+”rajat”;

JavaScript local variable

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only. For example:

```
<script>
function abc(){
var x=10;//local variable
}
</script>
```

JavaScript global variable

A **JavaScript global variable** is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable. For example:

Example:-

```
<script>
var data=200;//gloabal variable
function a(){
document.writeln(data);
}
function b(){
document.writeln(data);
}
a();//calling JavaScript function
b();
</script>
```

Declaring JavaScript global variable within function

To declare JavaScript global variables inside function, you need to use **window object**. For example:

```
function m(){
window.value=100;//declaring global variable by window object
}
function n(){
alert(window.value);//accessing global variable from other function
}
```

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:

```
var value=50;
function a(){
alert(window.value);//accessing global variable
}
```

Javascript Data Types

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

Note

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

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.

Note:-

JavaScript is a dynamic type language, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use var here to specify the data type. It can hold any type of values such as numbers, strings etc.

For example:

```
var a=40;//holding number
var b="Rahul";//holding string
```

Note – JavaScript does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values.

JavaScript Operators

JavaScript operators are symbols that are used to perform operations on operands. For example:

```
var sum=10+20;
```

There are following types of operators in JavaScript.

1. Arithmetic Operators
2. Comparison Operators
3. Logical Operators
4. Assignment Operators
5. Special Operators
6. Bitwise Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands.

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

Post increment or post decrement operator follow this concept a++ means there is two statements a=a; a=a+1;

Pre increment or pre decrement operator follow this concept to expand the ++a statement to two statements a=a+1 ; a=a;

JavaScript Comparison Operators

The JavaScript comparison operator compares the two operands.

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10===20 = false
!=	Not equal to	10!=20 = true

!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

JavaScript Logical Operators

The following operators are known as JavaScript logical operators.

The NOT operator (!) returns true for false statements and false for true statements.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

JavaScript Assignment Operators

Operator	Description	Example
=	Assign	10+10 = 20

+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

Miscellaneous Operator

We will discuss two operators here that are quite useful in JavaScript: the **conditional operator** (? :) and the **typeof operator**.

Conditional Operator (? :)

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Sr.No	Operator and Description
1	**? : (Conditional)** If Condition is true? Then value X : Otherwise value Y

Example:-

```
<script type="text/javascript">
      <!--
        var a = 10;
        var b = 20;
        var linebreak = "<br />";

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

        document.write ("((a < b) ? 100 : 200) => ");
        result = (a < b) ? 100 : 200;
        document.write(result);
        document.write(linebreak);
      //-->
    </script>
```

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.

Here is a list of the return values for the **typeof** Operator.

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

Example:-

```
  <script type="text/javascript">
      <!--
         var a = 10;
         var b = "String";
         var linebreak = "<br />";

         result = (typeof b == "string" ? "B is String" : "B is Numeric");
```

JavaScript Bitwise Operators

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	Zero fill left shift	5 << 1	0101 << 1	1010	10
>>>	Zero fill right shift	5 >>> 1	0101 >>> 1	0010	2

Example

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

```
var a = 2; // Bit presentation 10

var b = 3; // Bit presentation 11

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("(~b) => ");

result = (~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);
      //-->
    </script>
```

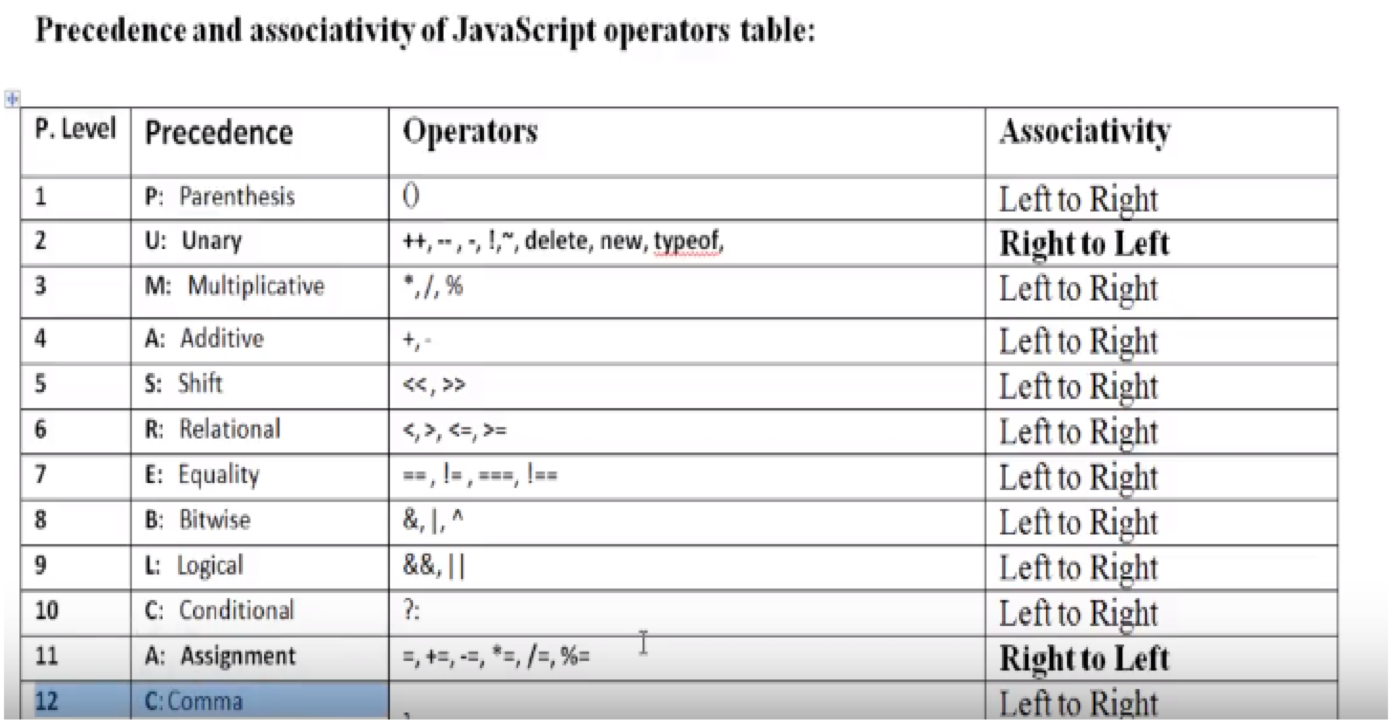
Precedence And Associativity Of Java Script Operators

while evaluating a given expression ; which operator should be operated first before the other?

Which operator needs to be given higher precedence than the other?

Associativity

if the same operator appears one or more times in an expression, in which direction the specific operator need to be evaluated.



Precedence and associativity of JavaScript operators table:

P. Level	Precedence	Operators	Associativity
1	P: Parenthesis	()	Left to Right
2	U: Unary	++, -- , -, !,~, delete, new, typeof,	**Right to Left**
3	M: Multiplicative	*, /, %	Left to Right
4	A: Additive	+, -	Left to Right
5	S: Shift	<<, >>	Left to Right
6	R: Relational	<, >, <=, >=	Left to Right
7	E: Equality	== , != , ===, !==	Left to Right
8	B: Bitwise	&, |, ^	Left to Right
9	L: Logical	&&, ||	Left to Right
10	C: Conditional	?:	Left to Right
11	A: Assignment	=, +=, -=, *=, /=, %=	**Right to Left**
12	C: Comma	,	Left to Right

JAVASCRIPT CONTROL STATEMENTS

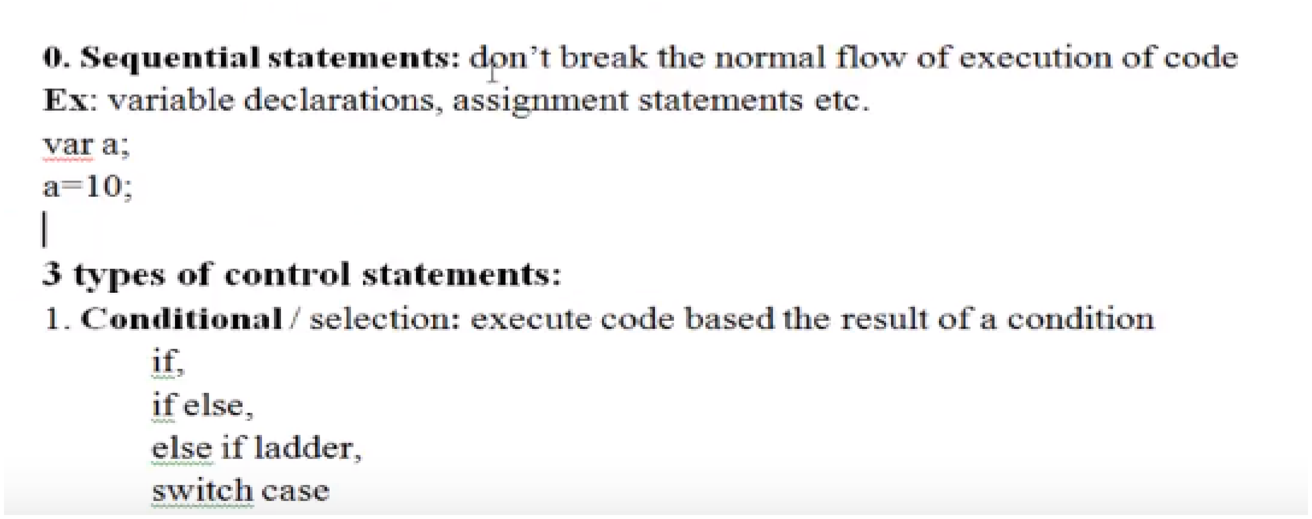
JavaScript supports the following forms of **if..else** statement –

- if statement
- if...else statement
- if...else if... statement.

JavaScript Control Statements

Note: Control statements help us to get the control over flow of execution of a code.

Note: We can solve any real world problem by writing a program containing four programming language statements: **sequential, conditional, control transfer** and **iterative** statements.

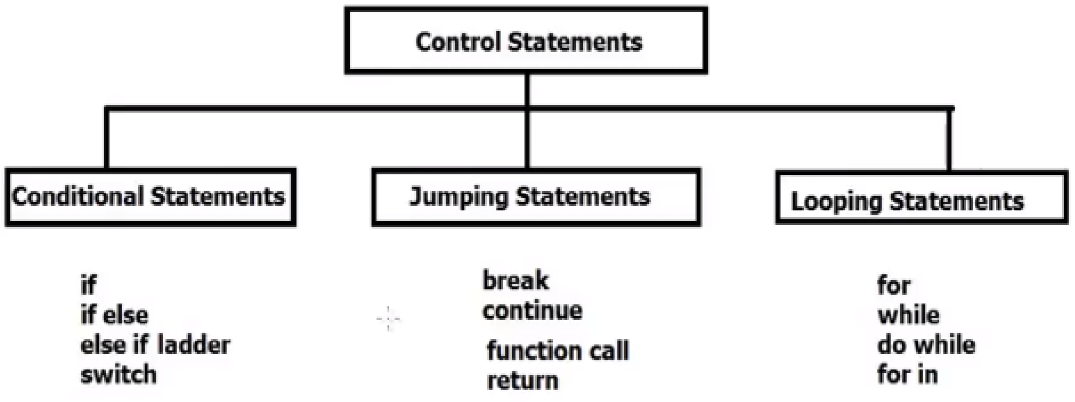


0. Sequential statements: don't break the normal flow of execution of code
Ex: variable declarations, assignment statements etc.

var a;
a=10;

3 types of control statements:

1. **Conditional** / selection: execute code based the result of a condition
    - if,
    - if else,
    - else if ladder,
    - switch case



if statement

The **if** statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax

The syntax for a basic if statement is as follows –

```
if (expression){

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

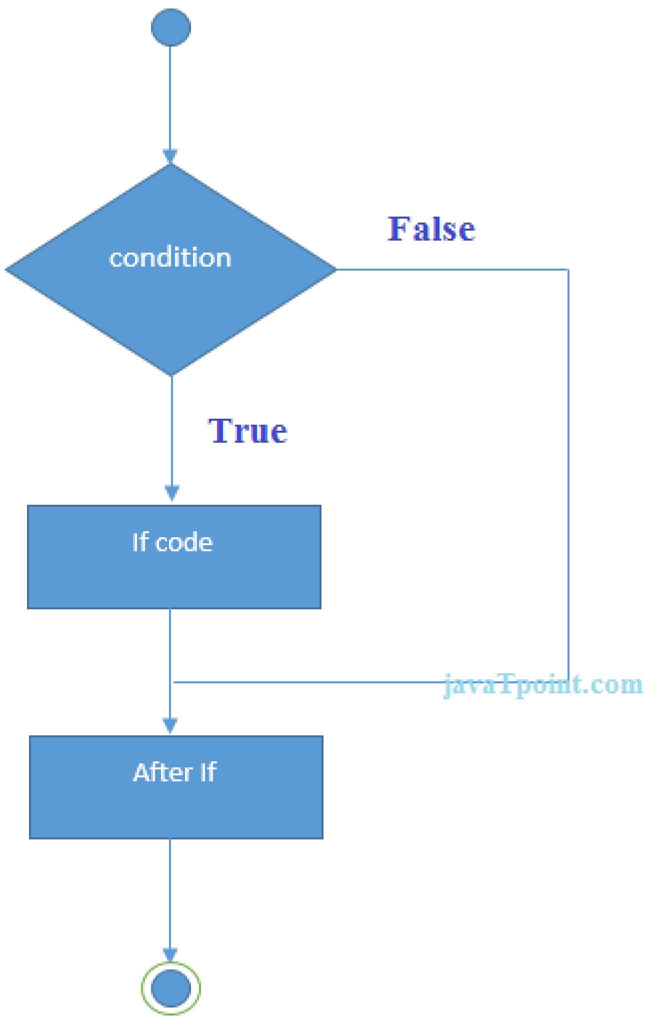
Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

Example

Try the following example to understand how the **if** statement works.

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var age = 20;

        if( age > 18 ){
          document.write("<b>Qualifies for driving</b>");
        }
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Flowchart of JavaScript If statement



JavaScript - if...else Statement

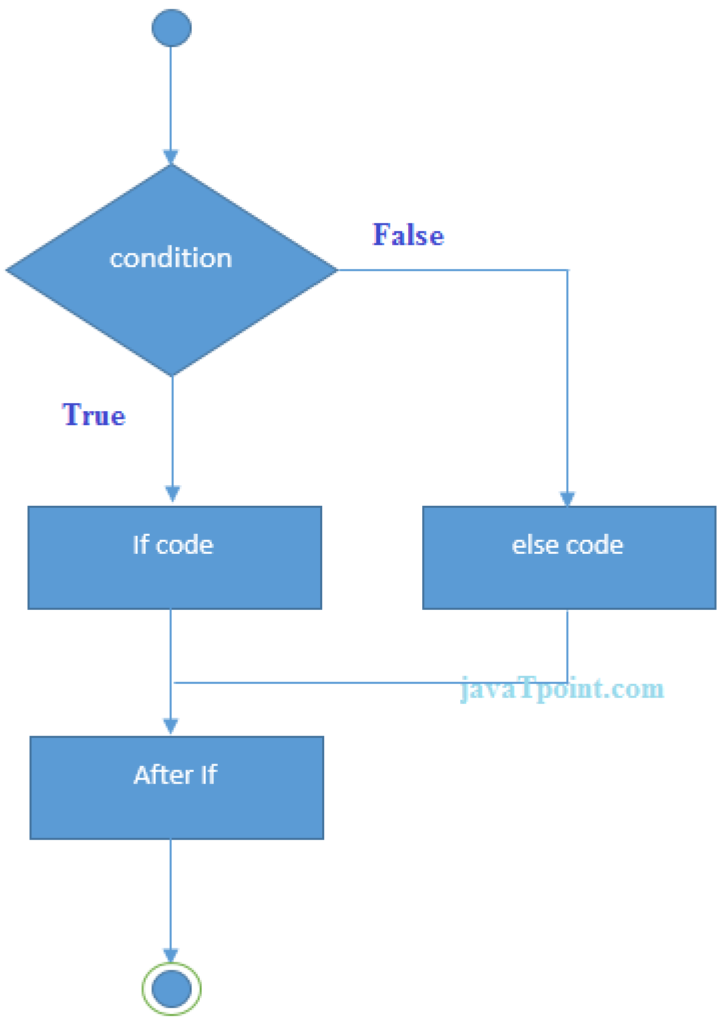
While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use conditional statements that allow your program to make correct decisions and perform right actions.

JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain the **if..else**statement

Flow Chart of if-else

The following flow chart shows how the if-else statement works.

```
if(expression){

//content to be evaluated if condition is true

}

else{

//content to be evaluated if condition is false

}
```



Let’s see the example of if-else statement in JavaScript to find out the even or odd number.

```
<script>

var a=20;

if(a%2==0){

document.write("a is even number");

}
```

```
else{

document.write("a is odd number");

}

</script>
```

if...else if... statement

The **if...else if...** statement is an advanced form of **if…else** that allows JavaScript to make a correct decision out of several conditions.

Syntax

The syntax of an if-else-if statement is as follows –

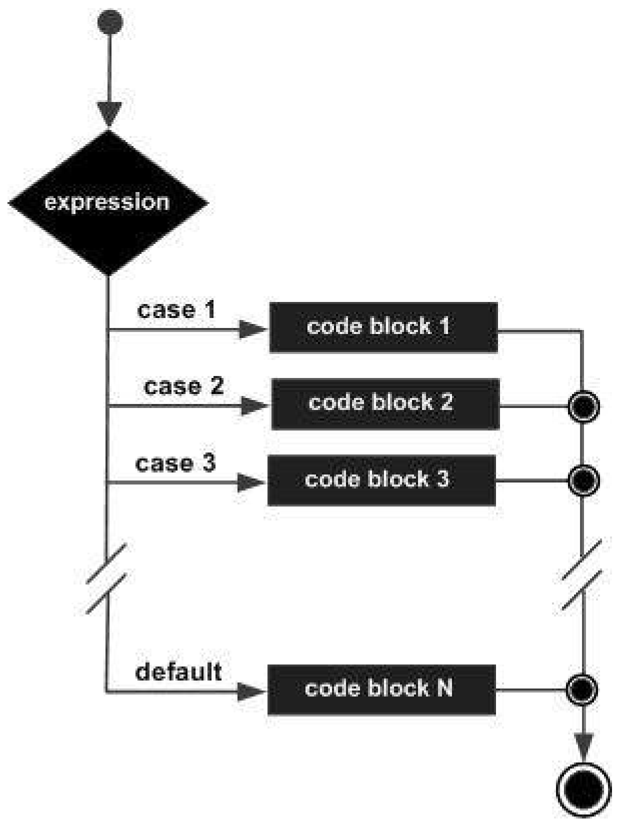
```
if (expression 1){

  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

}
```

JavaScript Switch

The **JavaScript switch statement** is used *to execute one code from multiple expressions*. It is just like else if statement that we have learned in previous page. But it is convenient than *if..else..if* because it can be used with numbers, characters etc.

you can use a **switch** statement which handles exactly this situation, and it does so more efficiently than repeated **if...else if**statements.



```
switch(expression){
case value1:
 code to be executed;
 break;
case value2:
 code to be executed;
 break;
......

default:
 code to be executed if above values are not matched;
}
```

JavaScript Loops

While writing a program, you may encounter a situation where you need to perform an action over and over again. In such situations, you would need to write loop statements to reduce the number of lines.

The **JavaScript loops** are used *to iterate the piece of code* using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

1. for loop
2. while loop
3. do-while loop

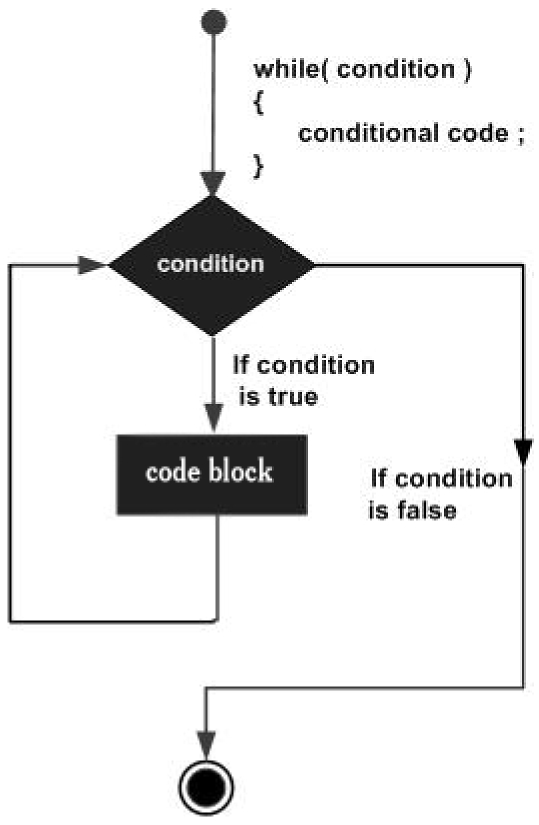
JavaScript while loop

The **JavaScript while loop** *iterates the elements for the infinite number of times*. It should be used if number of iteration is not known. The syntax of while loop is given below.

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.

Flow Chart

The flow chart of **while loop** looks as follows –



Syntax

The syntax of **while loop** in JavaScript is as follows −

```
while (expression){

  Statement(s) to be executed if expression is true

}
```

```
while (condition)
{
  code to be executed
}
```

JavaScript do while loop

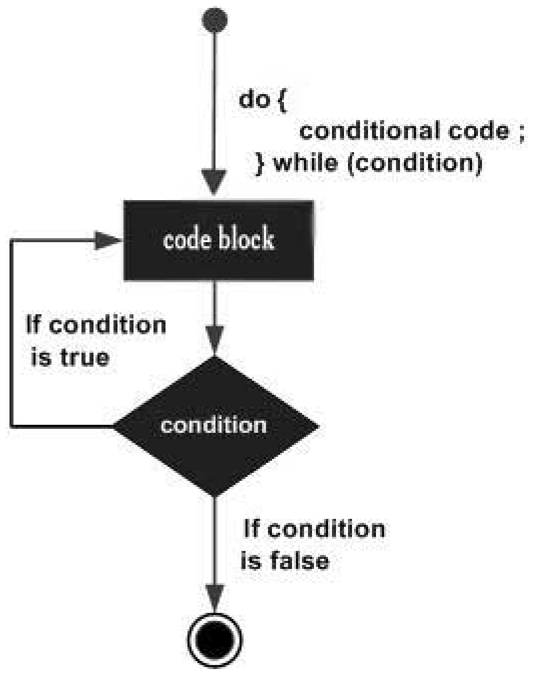
The **JavaScript do while loop** *iterates the elements for the infinite number of times* like while loop. But, code is *executed at least*once whether condition is true or false. The syntax of do while loop is given below.

The 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**.

Flow Chart

The flow chart of a **do-while** loop would be as follows –



```
do{
   code to be executed
}while (condition);
```

Syntax

The syntax for **do-while** loop in JavaScript is as follows –

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

Note – Don’t miss the semicolon used at the end of the do...while loop.

JavaScript - For Loop

The **JavaScript for loop** *iterates the elements for the fixed number of times*. It should be used if number of iteration is known. The syntax of for loop is given below.

```
for (initialization; condition; increment or update st)
{
   code to be executed
}
```

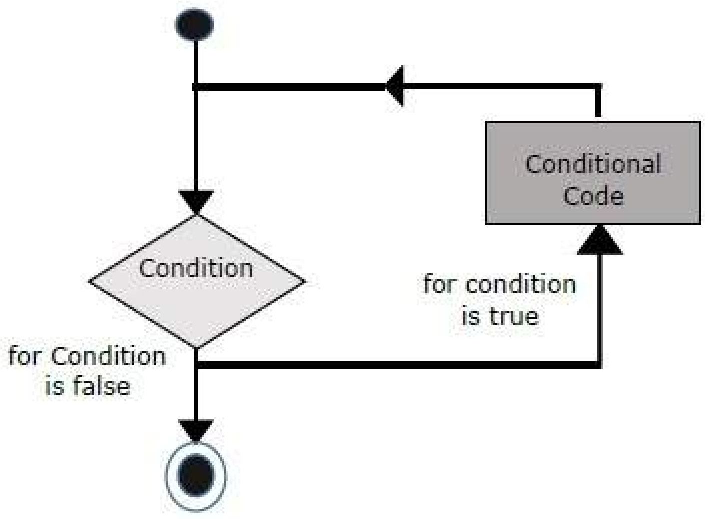
The '**for**' loop is the most compact form of looping. It includes the following three important parts –

- 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.

You can put all the three parts in a single line separated by semicolons.

Flow Chart

The flow chart of a for loop in JavaScript would be as follows –



Syntax

The syntax of for loop is JavaScript is as follows –

```
for (initialization; test condition; iteration statement){
  Statement(s) to be executed if test condition is true
}
```

JavaScript *for...in* loop

The **for...in** loop is used to loop through an object's properties. As we have not discussed Objects yet, you may not feel comfortable with this loop. But once you understand how objects behave in JavaScript, you will find this loop very useful.

Syntax

```
for (variablename in object){
  statement or block to execute
}
```

In each iteration, one property from **object** is assigned to **variablename** and this loop continues till all the properties of the object are exhausted.

JavaScript - Loop Control

JavaScript provides full control to handle loops and switch statements. There may be a situation when you need to come out of a loop without reaching its bottom. There may also be a situation when you want to skip a part of your code block and start the next iteration of the loop.

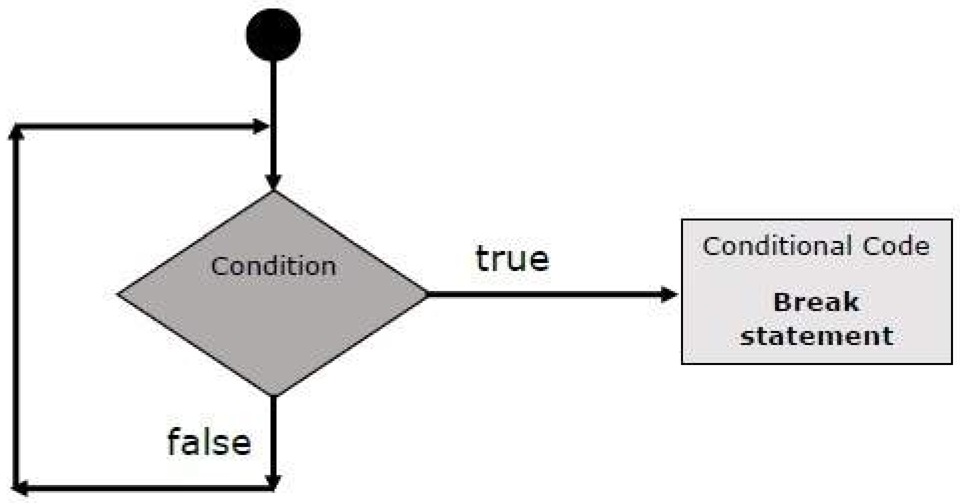
To handle all such situations, JavaScript provides **break** and **continue** statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

The break Statement

The **break** statement, which was briefly introduced with the *switch* statement, is used to exit a loop early, breaking out of the enclosing curly braces.

Flow Chart

The flow chart of a break statement would look as follows –



Example

The following example illustrates the use of a **break** statement with a while loop. Notice how the loop breaks out early once **x** reaches 5 and reaches to **document.write (..)** statement just below to the closing curly brace –

```
<html>
  <body>

    <script type="text/javascript">
      <!--
      var x = 1;
      document.write("Entering the loop<br /> ");

      while (x < 20)
      {
        if (x == 5){
          break; // breaks out of loop completely
        }
        x = x + 1;
        document.write( x + "<br />");
      }

      document.write("Exiting the loop!<br /> ");
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

The continue Statement

The **continue** statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a **continue**statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

Example

This example illustrates the use of a **continue** statement with a while loop. Notice how the **continue** statement is used to skip printing when the index held in variable **x** reaches 5 .

```
<html>
  <body>
```

```
    <script type="text/javascript">
      <!--
        var x = 1;
        document.write("Entering the loop<br /> ");

        while (x < 10)
        {
          x = x + 1;

          if (x == 5){
            continue; // skip rest of the loop body
          }
          document.write( x + "<br />");
        }

        document.write("Exiting the loop!<br /> ");
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

JavaScript – 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.

Advantage of JavaScript function

There are mainly two advantages of JavaScript functions.

1. **Code reusability**: We can call a function several times so it save coding.
2. **Less coding**: It makes our program compact. We don't need to write many lines of code each time to perform a common task.

JavaScript Function Syntax

The syntax of declaring function is given below.

```
function functionName([arg1, arg2, ...argN]){
 //code to be executed
}
```

Note:- JavaScript Functions can have 0 or more arguments.

Example:-

```
<script>
function msg(){
alert("hello! this is message");
}
</script>
<input type="button" onclick="msg()" value="call function"/>
```

When we are creating a function we focus on three steps.

1)FUNCTION DECLARATION

Here we focus on four things

1. Function type(return type)
2. Function name
3. Parameter list(Arguments)
4. Terminating semicolon

2)FUNCTION CALL
3)FUNCTION DEFINITION

Function Definition

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

Syntax

The basic syntax is shown here.

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

Calling a Function

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

```
<html>
  <head>

    <script type="text/javascript">
      function  sayHello()
      {
        document.write ("Hello there!");
      }
    </script>

  </head>
  <body>
```

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


      <form>
        <input type="button" onclick="sayHello()" value="Say Hello">
      </form>



    </body>
  </html
```

Function Parameters

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

Example:-

it takes two parameters.

```
  <html>
    <head>

      <script type="text/javascript">
        function sayHello(name, age)
        {
```

```
        document.write (name + " is " + age + " years old.");
      }
    </script>

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

    <form>
      <input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
    </form>

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

The return Statement

A JavaScript function can have an optional **return** statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Example:-

```
<html>
  <head>

    <script type="text/javascript">
      function concatenate(first, last)
      {
        var full;
        full = first + last;
        return full;
      }

      function secondFunction()
      {
        var result;
        result = concatenate('Zara', 'Ali');
        document.write (result );
      }
    </script>

  </head>

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

    <form>
```

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

      </form>




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


  </body>

</html>
```

Converting Arrays to Strings

The JavaScript method **toString()** converts an array to a string of (comma separated) array values.

Join()

The **join()** method also joins all array elements into a string.

It behaves just like toString(), but in addition you can specify the separator.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

Popping and Pushing

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is:

Popping items **out** of an array, or pushing items **into** an array.

Popping

The **pop()** method removes the last element from an array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();
```

Pushing

The **push()** method adds a new element to an array (at the end):

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");      //  Adds a new element ("Kiwi") to fruits
```

Shifting Elements

Shifting is equivalent to popping, working on the first element instead of the last.

The **shift()** method removes the first array element and "shifts" all other elements to a lower index.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();          // Removes the first element "Banana" from fruits
```

The shift() method returns the string that was "shifted out":

The **unshift()** method adds a new element to an array (at the beginning), and "unshifts" older elements:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");    // Adds a new element "Lemon" to fruits
```

The unshift() method returns the new array length.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");    // Returns 5
```

Deleting Elements

Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator **delete**:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
```

```
delete fruits[0];          // Changes the first element in fruits to undefined
```

Using **delete** may leave undefined holes in the array. Use pop() or shift() instead.

JavaScript Math Object

The **JavaScript math** object provides several constants and methods to perform mathematical operation. Unlike date object, it doesn't have constructors.

Math.sqrt(n)

The JavaScript math.sqrt(n) method returns the square root of the given number.

Math.random()

The JavaScript math.random() method returns the random number between 0 to 1.

Math.pow(m,n)

The JavaScript math.pow(m,n) method returns the m to the power of n that is m^n.

Math.floor(n)

The JavaScript math.floor(n) method returns the lowest integer for the given number. For example 3 for 3.7, 5 for 5.9 etc.

Math.ceil(n)

The JavaScript math.ceil(n) method returns the largest integer for the given number. For example 4 for 3.7, 6 for 5.9 etc.

Math.round(n)

The JavaScript math.round(n) method returns the rounded integer nearest for the given number. If fractional part is equal or greater than 0.5, it goes to upper value 1 otherwise lower value 0. For example 4 for 3.7, 3 for 3.3, 6 for 5.9 etc.

Math.abs(n)

The JavaScript math.abs(n) method returns the absolute value for the given number. For example 4 for -4, 6.6 for -6.6 etc.

JavaScript Number Object

In general, you do not need to worry about **Number** objects because the browser automatically converts number literals to instances of the number class.

The **JavaScript number** object *enables you to represent a numeric value*. It may be integer or floating-point. JavaScript number object follows IEEE standard to represent the floating-point numbers.

By the help of Number() constructor, you can create number object in JavaScript. For example:

1. var n=new Number(value);

If value can't be converted to number You can direct assign a number to a variable also. For example:

```
var x=102;//integer value

var y=102.7;//floating point value

var z=13e4;//exponent value, output: 130000

var n=new Number(16);//integer value by number object
```

JavaScript Number Methods

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

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.

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

```
  var num=77.1234;
  var val = num.toExponential();
  document.write("num.toExponential() is : " + val );
  document.write("<br />");

  val = num.toExponential(4);
  document.write("num.toExponential(4) is : " + val );
  document.write("<br />");

  val = num.toExponential(2);
  document.write("num.toExponential(2) is : " + val);
  document.write("<br />");

  val = 77.1234.toExponential();
  document.write("77.1234.toExponential()is : " + val );
  document.write("<br />");

  val = 77.1234.toExponential();
  document.write("77 .toExponential() is : " + val);
</script>
```

```
<script type="text/javascript">
  var num=177.1234;
  document.write("num.toFixed() is : " + num.toFixed());
```

```
        document.write("<br />");

        document.write("num.toFixed(6) is : " + num.toFixed(6));
        document.write("<br />");

        document.write("num.toFixed(1) is : " + num.toFixed(1));
        document.write("<br />");

        document.write("(1.23e+20).toFixed(2) is:" + (1.23e+20).toFixed(2));
        document.write("<br />");

        document.write("(1.23e-10).toFixed(2) is : " + (1.23e-10).toFixed(2));
      </script>
```

Description

This method converts a **number** object into a human readable string representing the number using the locale of the environment.

Syntax

Its syntax is as follows –

```
number.toLocaleString()
```

```
<script type="text/javascript">
        var num = new Number(177.1234);
        document.write( num.toLocaleString());
    </script>
```

JavaScript Number - toPrecision()

Description

This method returns a string representing the **number** object to the specified precision.

Syntax

Its syntax is as follows –

```
number.toPrecision( [ precision ] )
```

```
<script type="text/javascript">
        var num = new Number(7.123456);
        document.write("num.toPrecision() is " + num.toPrecision());
        document.write("<br />");

        document.write("num.toPrecision(4) is " + num.toPrecision(4));
        document.write("<br />");

        document.write("num.toPrecision(2) is " + num.toPrecision(2));
        document.write("<br />");

        document.write("num.toPrecision(1) is " + num.toPrecision(1));
     </script>
```

Description

This method returns a string representing the specified object. The **toString()**method parses its first argument, and attempts to return a string representation in the specified radix (base).

Syntax

Its syntax is as follows –

```
number.toString( [radix] )
```

Parameter Details

radix – An integer between 2 and 36 specifying the base to use for representing numeric values.

Return Value

Returns a string representing the specified Number object.

Example

Try the following example.

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

   num = new Number(15);

   document.write("num.toString() is " + num.toString());

   document.write("<br />");


   document.write("num.toString(2) is " + num.toString(2));

   document.write("<br />");


   document.write("num.toString(4) is " + num.toString(4));

   document.write("<br />");

  </script>
```

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

Description

This method returns a string of either "true" or "false" depending upon the value of the object.

Syntax

Its syntax is as follows –

```
boolean.toString()
```

```
<script type="text/javascript">
   var flag = new Boolean(false);
   document.write( "flag.toString is : " + flag.toString() );
  </script>
```

Description

Javascript boolean **valueOf()** method returns the primitive value of the specified **boolean** object.

Syntax

Its syntax is as follows –

```
boolean.valueOf()
```

```
 <script type="text/javascript">
   var flag = new Boolean(false);
   document.write( "flag.valueOf is : " + flag.valueOf() );
  </script>
```

JavaScript Form Validation

It is important to validate the form submitted by the user because it can have inappropriate values. So validation is must.

The JavaScript provides you the facility the validate the form on the client side so processing will be fast than server-side validation. So, most of the web developers prefer JavaScript form validation.

Through JavaScript, we can validate name, password, email, date, mobile number etc fields.

JavaScript form validation example

In this example, we are going to validate the name and password. The name can’t be empty and password can’t be less than 6 characters long.

Here, we are validating the form on form submit. The user will not be forwarded to the next page until given values are correct.

```
<script>

function validateform(){

var name=document.myform.name.value;

var password=document.myform.password.value;



if (name==null || name==""){

alert("Name can't be blank");

return false;

}else if(password.length<6){

alert("Password must be at least 6 characters long.");

return false;

}

}

</script>
```

JavaScript Retype Password Validation

```
<script type="text/javascript">
function matchpass(){
var firstpassword=document.f1.password.value;
var secondpassword=document.f1.password2.value;

if(firstpassword==secondpassword){
return true;
}
else{
alert("password must be same!");
return false;
}
}
</script>
```

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. For example:

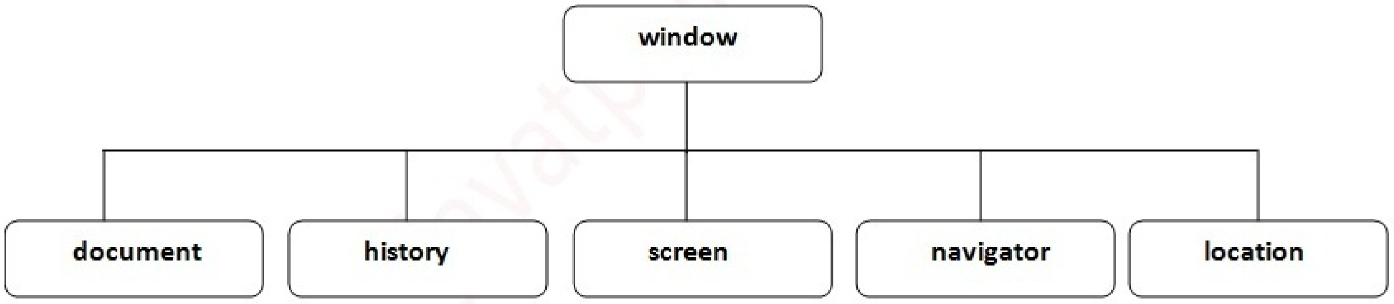
```
1. window.alert("hello javatpoint");
```

is same as:

```
1. alert("hello javatpoint");
```

You can use a lot of properties (other objects) defined underneath the window object like document, history, screen, navigator, location, innerHeight, innerWidth,

Note: The document object represents an html document. It forms DOM (Document Object Model).



Visit the next page to learn about window object fully with example.

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.

Note: if html document contains frame or iframe, browser creates additional window objects for each frame.

Methods of window object

The important methods of window object are as follows:

Method	Description
alert()	displays the alert box containing message with ok button.
confirm()	displays the confirm dialog box containing message with ok and cancel button.
prompt()	displays a dialog box to get input from the user.
open()	opens the new window.
close()	closes the current window.
setTimeout()	performs action after specified time like calling function, evaluating expressions etc.

Example of open() in javascript

It displays the content in a new window.

```
<script type="text/javascript">
function msg(){
open("http://www.javatpoint.com");
}
</script>
```

```
<input type="button" value="javatpoint" onclick="msg()"/>
```

Example of setTimeout() in javascript

It performs its task after the given milliseconds.

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

function msg(){

setTimeout(

function(){

alert("Welcome to   after 2 seconds")

},2000);


}

</script>


<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.

The history object is the window property, so it can be accessed by:

1. window.history

Or,

1. history

Property of JavaScript history object

There are only 1 property of 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 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.

```
<script>
document.writeln("<br/>screen.width: "+screen.width);
document.writeln("<br/>screen.height: "+screen.height);
document.writeln("<br/>screen.availWidth: "+screen.availWidth);
document.writeln("<br/>screen.availHeight: "+screen.availHeight);
document.writeln("<br/>screen.colorDepth: "+screen.colorDepth);
document.writeln("<br/>screen.pixelDepth: "+screen.pixelDepth);
</script>
```

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.

As mentioned earlier, it is the object of window. So

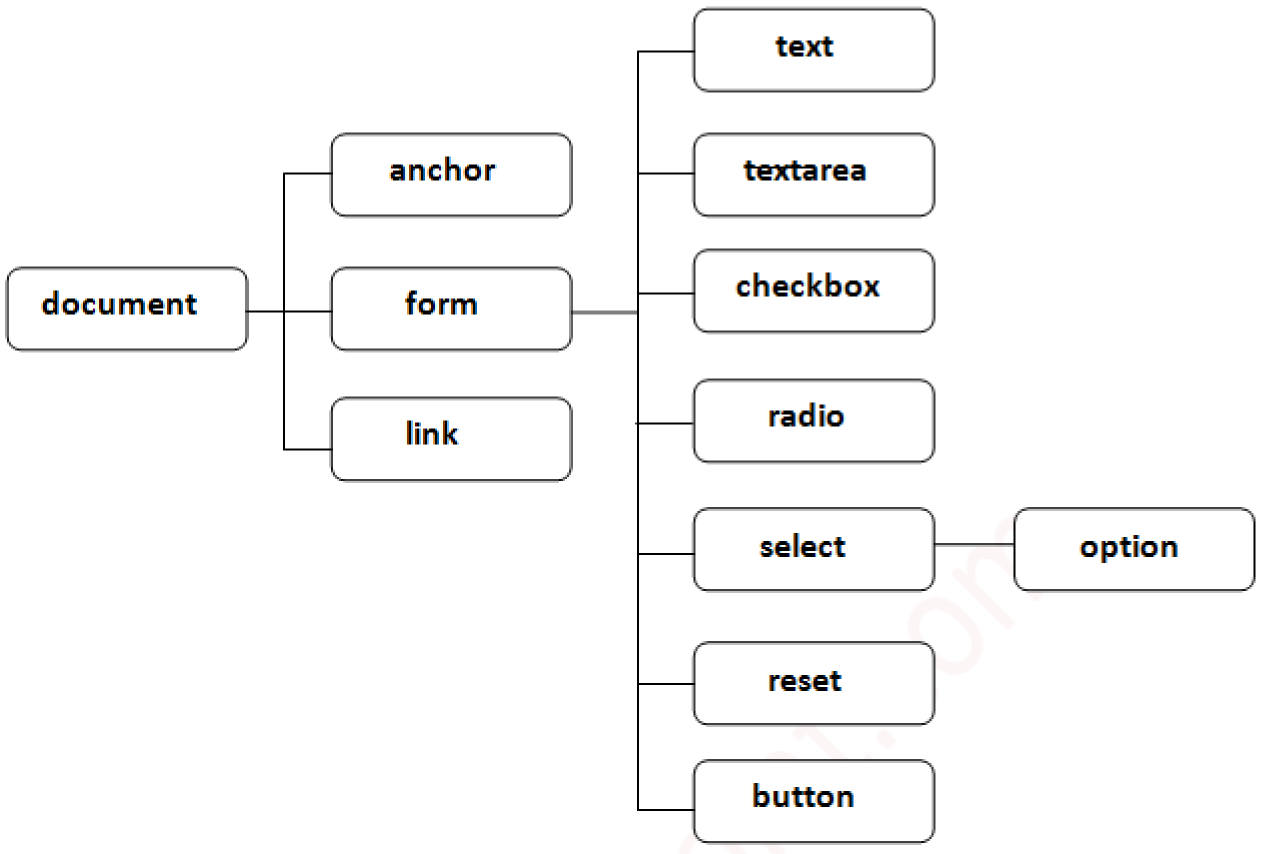
1. window.document

Is same as

1. document

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.

Accessing field value by document object

In this example, we are going to get the value of input text by user. Here, we are using **document.form1.name.value** to get the value of name field.

Here, **document** is the root element that represents the html document.

form1 is the name of the form.

name is the attribute name of the input text.

value is the property, that returns the value of the input text.

Let's see the simple example of document object that prints name with welcome message.

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

function printvalue(){

var name=document.form1.name.value;

alert("Welcome: "+name);

}

</script>


<form name="form1">

Enter Name:<input type="text" name="name"/>

<input type="button" onclick="printvalue()" value="print name"/>

</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.

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

function getcube(){

var number=document.getElementById("number").value;

alert(number*number*number);

}

</script>

<form>

Enter No:<input type="text" id="number" name="number"/><br/>

<input type="button" value="cube" onclick="getcube()"/>

</form>
```

Javascript-document.getElementsByName() method

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

The syntax of the getElementsByName() method is given below:

1. document.getElementsByName("name")

   Here, name is required.

Example of document.getElementsByName() method

In this example, we going to count total number of genders. Here, we are using getElementsByName() method to get all the genders.

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

function totalelements()

{

var allgenders=document.getElementsByName("gender");

alert("Total Genders:"+allgenders.length);

}

</script>

<form>

Male:<input type="radio" name="gender" value="male">

Female:<input type="radio" name="gender" value="female">


<input type="button" onclick="totalelements()" value="Total Genders">

</form>
```

Javascript - document.getElementsByTagName() method

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

The syntax of the getElementsByTagName() method is given below:

1. document.getElementsByTagName("name")

Here, name is required.

Example of document.getElementsByTagName() method

In this example, we going to count total number of paragraphs used in the document. To do this, we have called the document.getElementsByTagName("p") method that returns the total paragraphs.

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

function countpara(){

var totalpara=document.getElementsByTagName("p");

alert("total p tags are: "+totalpara.length);



}

</script>

<p>This is a pragraph</p>

<p>Here we are going to count total number of paragraphs by getElementByTagName() method.
</p>

<p>Let's see the simple example</p>

<button onclick="countpara()">count paragraph</button>
```

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 going to create the html form when user clicks on the button.

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.

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

function showcommentform() {

var data="Name:<input type='text' name='name'><br>Comment:<br><textarea rows='5' cols='80'></tex
tarea>

<br><input type='submit' value='Post Comment'>";

document.getElementById('mylocation').innerHTML=data;

}

</script>

<form name="myForm">

<input type="button" value="comment" onclick="showcommentform()">

<div id="mylocation"></div>

</form>
```

Javascript - innerText

The **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.

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

function validate() {

var msg;

if(document.myForm.userPass.value.length>5){

msg="good";

}

else{

msg="poor";

}

document.getElementById('mylocation').innerText=msg;

 }


</script>

<form name="myForm">

<input type="password" value="" name="userPass" onkeyup="validate()">

Strength:<span id="mylocation">no strength</span>

</form>
```

Javascript date **getTime()** method returns the numeric value corresponding to the time for the specified date according to universal time. The value returned by the **getTime** method is the number of milliseconds since 1 January 1970 00:00:00.

You can use this method to help assign a date and time to another Date object.

HTML/DOM events for JavaScript

HTML or DOM events are widely used in JavaScript code. JavaScript code is executed with HTML/DOM events. So before learning JavaScript, let's have some idea about events.

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.