JAVASCRIPT NOTES

Index

What is JavaScript

JavaScript is a lightweight, interpreted **programming** language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. **JavaScript** is very easy to implement because it is integrated with HTML. It is open and cross-platform. JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document. JavaScript has no connectivity with Java programming language. In addition to web browsers, databases such as CouchDB and MongoDB uses JavaScript as their scripting and query language.

Features of JavaScript

There are following features of JavaScript:

1. All popular web browsers support JavaScript as they provide built-in execution environments.
2. JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
3. JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).
4. JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
5. It is a light-weighted and interpreted language.
6. It is a case-sensitive language.
7. JavaScript is supportable in several operating systems including, Windows, macOS, etc.
8. It provides good control to the users over the web browsers.

Application of JavaScript

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

- Client-side validation,
- Dynamic drop-down menus,
- Displaying date and time,
- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
- Displaying clocks etc.

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.

Limitations of JavaScript

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

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

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

JavaScript - Syntax

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

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

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script.

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>
```

Let us start with the very simple example of JavaScript “Hello You”

```
<html>
  <body>
    <script language = "javascript" type = "text/javascript">
      <!--
        document.write("Hello You!")
      //-->
    </script>
  </body>
</html>
```

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>
```

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.

Comments in JavaScript

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

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

Example

The following example shows how to use comments in JavaScript.

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

    /*
    * This is a multi-line comment in JavaScript
    * It is very similar to comments in C Programming
    */
  //-->
</script>
```

JavaScript Output

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.

The id attribute defines the HTML element. The innerHTML property defines the HTML content:

```
<!DOCTYPE html>
<html>
<body>
<h2>My First Web Page</h2>
<p>My First Paragraph.</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
</body>
</html>
```

Using document.write()

For testing purposes, it is convenient to use document.write():

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

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

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

</body>
</html>
```

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

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

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

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

</body>
</html>
```

The document.write() method should only be used for testing.

Using window.alert()

You can use an alert box to display data:

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

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

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

</body>
</html>
```

You can skip the window keyword.

In JavaScript, the window object is the global scope object, that means that variables, properties, and methods by default belong to the window object. This also means that specifying the window keyword is optional:

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

<h2>My First Web Page</h2>
<p>My first paragraph.</p>

<script>
alert(5 + 6);
</script>

</body>
</html>
```

Using console.log()

For debugging purposes, you can call the console.log() method in the browser to display data.

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

<h2>Activate Debugging</h2>

<p>F12 on your keyboard will activate debugging.</p>
<p>Then select "Console" in the debugger menu.</p>
<p>Then click Run again.</p>

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

</body>
</html>
```

JavaScript Print

JavaScript does not have any print object or print methods. You cannot access output devices from JavaScript. The only exception is that you can call the window.print() method in the browser to print the content of the current window.

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

<h2>The window.print() Method</h2>

<p>Click the button to print the current page.</p>

<button onclick="window.print()">Print this page</button>

</body>
</html>
```

Enabling JavaScript in Browsers

All the modern browsers come with built-in support for JavaScript. Frequently, you may need to enable or disable this support manually. This chapter explains the procedure of enabling and disabling JavaScript support in your browsers: Internet Explorer, Firefox, chrome, and Opera.

JavaScript in Internet Explorer

Here are simple steps to turn on or turn off JavaScript in your Internet Explorer −

- Follow **Tools** → **Internet Options** from the menu.
- Select **Security** tab from the dialog box.
- Click the **Custom Level** button.
- Scroll down till you find **Scripting** option.
- Select *Enable* radio button under **Active scripting**.
- Finally click OK and come out

To disable JavaScript support in your Internet Explorer, you need to select **Disable** radio button under **Active scripting**.

JavaScript in Firefox

Here are the steps to turn on or turn off JavaScript in Firefox −

- Open a new tab → type **about: config** in the address bar.
- Then you will find the warning dialog. Select **I'll be careful, I promise!**
- Then you will find the list of **configure options** in the browser.
- In the search bar, type **javascript.enabled**.
- There you will find the option to enable or disable javascript by right-clicking on the value of that option → **select toggle**.

If javascript.enabled is true; it converts to false upon clicking **toogle**. If javascript is disabled; it gets enabled upon clicking toggle.

JavaScript in Chrome

Here are the steps to turn on or turn off JavaScript in Chrome −

- Click the Chrome menu at the top right hand corner of your browser.
- Select **Settings**.
- Click **Show advanced settings** at the end of the page.
- Under the **Privacy** section, click the Content settings button.

- In the "Javascript" section, select "Do not allow any site to run JavaScript" or "Allow all sites to run JavaScript (recommended)".

JavaScript in Opera

Here are the steps to turn on or turn off JavaScript in Opera −

- Follow **Tools** → **Preferences** from the menu.
- Select **Advanced** option from the dialog box.
- Select **Content** from the listed items.
- Select **Enable JavaScript** checkbox.
- Finally click OK and come out.

To disable JavaScript support in your Opera, you should not select the **Enable JavaScript checkbox**.

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.

You can add a **noscript** block immediately after the script block as follows −

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

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

Now, 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.

JavaScript - Placement in HTML File

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

In the following section, we will see how we can place JavaScript in an HTML file in different ways.

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>
```

This code will produce the following results −

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>
```

This code will produce the following results −

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>
```

This code will produce the following result −

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")
}
```

JavaScript - Variables

The JavaScript syntax defines two types of values:

- Fixed values called Literals
- Variable values called Variables

JavaScript Literals

The two most important syntax rules for fixed values are:

1. **Numbers** are written with or without decimals: 10, 10.20

2. **Strings** are text, written within double or single quotes:"Richa" or'Richa'

JavaScript Variables

Variables are containers for storing data (storing data values). There are four different ways to use variables in JavaScript.

- Using var
- Using let
- Using const
- Using nothing

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

<h1>JavaScript Variables</h1>

<p>In this example, x, y, and z are variables.</p>

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

<script>
var x = 5;
var y = 6;
var z = x + y;
document.getElementById("demo").innerHTML =
"The value of z is: " + z;
</script>

</body>
</html>
```

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

<h2>JavaScript Variables</h2>
```

```
<p>In this example, x, y, and z are variables.</p>

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

<script>
let x = 5;
let y = 6;
let z = x + y;
document.getElementById("demo").innerHTML =
"The value of z is: " + z;
</script>

</body>
</html>
```

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

<h1>JavaScript Variables</h1>

<p>In this example, x, y, and z are undeclared variables.</p>

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

<script>
x = 5;
y = 6;
z = x + y;
document.getElementById("demo").innerHTML =
"The value of z is: " + z;
</script>

</body>
</html>
```

When to Use JavaScript var?

Always declare JavaScript variables with var,let, orconst.

The var keyword is used in all JavaScript code from 1995 to 2015.

The let and const keywords were added to JavaScript in 2015.

If you want your code to run in older browser, you must use var.

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- **Global Variables** − A global variable has global scope which means it can be defined anywhere in your JavaScript code.
- **Local Variables** − A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

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

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

Variables created **without** a declaration keyword (var, let, or const) are always global, even if they are created inside a function.

When to Use JavaScript const?

If you want a general rule: always declare variables with const.

If you think the value of the variable can change, use let.

In this example, price1, price2, and total, are variables:

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

<h1>JavaScript Variables</h1>

<p>In this example, price, price2, and total are variables.</p>

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

<script>
const price1 = 5;
const price2 = 6;
let total = price1 + price2;
document.getElementById("demo").innerHTML =
"The total is: " + total;
</script>

</body>
</html>
```

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _ (but we will not use it in this tutorial)
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

JavaScript Keywords

JavaScript **keywords** are used to identify actions to be performed.

The `let` keyword tells the browser to create variables:

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

<h2>The <b>let</b> Keyword Creates Variables</h2>

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

<script>
let x, y;
x = 5 + 6;
y = x * 10;
document.getElementById("demo").innerHTML = y;
</script>

</body>
</html>
```

One of many JavaScript HTML methods is getElementById().

The example below "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello JavaScript":

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

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change HTML content.</p>

<button type="button" onclick='document.getElementById("demo").innerHTML = "Hello
JavaScript!"'>Click Me!</button>

</body>
</html>
```

JavaScript Statements

A **computer program** is a list of "instructions" to be "executed" by a computer. In a programming language, these programming instructions are called **statements**. A **JavaScript program** is a list of programming **statements**.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Statements</h2>
<p>A <b>JavaScript program</b> is a list of <b>statements</b> to be executed by a computer.</p>
<p id="demo"></p>
<script>
let x, y, z;  // Statement 1
x = 5;        // Statement 2
y = 6;        // Statement 3
z = x + y;    // Statement 4
document.getElementById("demo").innerHTML =
"The value of z is " + z + ".";
</script>
</body>
</html>
```

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

This statement tells the browser to write "Hello Richa." inside an HTML element with id="demo":

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

<h2>JavaScript Statements</h2>

<p>In HTML, JavaScript statements are executed by the browser.</p>

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

<script>
document.getElementById("demo").innerHTML = "Hello Richa.";
</script>

</body>
</html>
```

Most JavaScript programs contain many JavaScript statements. The statements are executed, one by one, in the same order as they are written.

JavaScript programs (and JavaScript statements) are often called JavaScript code.

JavaScript White Space

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

The following lines are equivalent:

```
let person = "Hege";
let person="Hege";
```

JavaScript Line Length and Line Breaks

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

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

<h2>JavaScript Statements</h2>

<p>
The best place to break a code line is after an operator or a comma.
</p>

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

<script>
document.getElementById("demo").innerHTML =
"Hello Dolly!";
</script>

</body>
</html>
```

JavaScript Code Blocks

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.

The purpose of code blocks is to define statements to be executed together.

One place you will find statements grouped together in blocks, is in JavaScript functions:

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

<h2>JavaScript Statements</h2>

<p>JavaScript code blocks are written between { and }</p>

<button type="button" onclick="myFunction()">Click Me!</button>
```

```
<p id="demo1"></p>
<p id="demo2"></p>
<script>
function myFunction() {
  document.getElementById("demo1").innerHTML = "Hello Dolly!";
  document.getElementById("demo2").innerHTML = "How are you?";
}
</script>
</body>
</html>
```

JavaScript - Operators

Let us take a simple expression **4 + 5 is equal to 9**. Here 4 and 5 are called **operands** and '+' is called the **operator**. JavaScript supports the following types of operators.

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

Lets have a look on all operators one by one.

Arithmetic Operators

JavaScript supports the following arithmetic operators −

Assume variable A holds 10 and variable B holds 20, then −

Sr.No.	Operator & Description
1	**+ (Addition)** Adds two operands **Ex:** A + B will give 30
2	**- (Subtraction)** Subtracts the second operand from the first **Ex:** A - B will give -10
3	*** (Multiplication)** Multiply both operands **Ex:** A * B will give 200
4	**/ (Division)** Divide the numerator by the denominator **Ex:** B / A will give 2
5	**% (Modulus)** Outputs the remainder of an integer division **Ex:** B % A will give 0

6	**++ (Increment)** Increases an integer value by one **Ex:** A++ will give 11
7	**-- (Decrement)** Decreases an integer value by one **Ex:** A-- will give 9

Note − Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

Example

The following code shows how to use arithmetic operators in JavaScript.

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

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

        b = --b;
        document.write("--b = ");
        result = --b;
        document.write(result);
        document.write(linebreak);
      //-->
    </script>

    Set the variables to different values and then try...
  </body>
</html>
```

Comparison Operators

JavaScript supports the following comparison operators −

Assume variable A holds 10 and variable B holds 20, then −

Sr.No.	Operator & Description
1	**= = (Equal)** Checks if the value of two operands are equal or not, if yes, then the condition becomes true. **Ex:** (A == B) is not true.
2	**!= (Not Equal)** Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true. **Ex:** (A != B) is true.
3	**> (Greater than)** Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A > B) is not true.
4	**< (Less than)**

	Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A < B) is true.
5	**>= (Greater than or Equal to)** Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true. **Ex:** (A >= B) is not true.
6	**<= (Less than or Equal to)** Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true. **Ex:** (A <= B) is true.

Example

The following code shows how to use comparison operators in JavaScript.

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var a = 10;
        var b = 20;
        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) => ");
            result = (a <= b);
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      Set the variables to different values and different operators and then try...
   </body>
</html>
```

Logical Operators

JavaScript supports the following logical operators −

Assume variable A holds 10 and variable B holds 20, then −

Sr.No.	Operator & Description
1	**&& (Logical AND)** If both the operands are non-zero, then the condition becomes true. **Ex:** (A && B) is true.
2	**|| (Logical OR)** If any of the two operands are non-zero, then the condition becomes true. **Ex:** (A || B) is true.
3	**! (Logical NOT)** Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. **Ex:** ! (A && B) is false.

Example

Try the following code to learn how to implement Logical Operators in JavaScript.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = true;
            var b = false;
            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);
         //-->
      </script>
      <p>Set the variables to different values and different operators and then try...</p>
   </body>
</html>
```

Bitwise Operators

JavaScript supports the following bitwise operators −

Assume variable A holds 2 and variable B holds 3, then −

Sr.No.	Operator & Description
1	**& (Bitwise AND)** It performs a Boolean AND operation on each bit of its integer arguments. **Ex:** (A & B) is 2.
2	**| (BitWise OR)** It performs a Boolean OR operation on each bit of its integer arguments. **Ex:** (A | B) is 3.
3	**^ (Bitwise XOR)** It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both. **Ex:** (A ^ B) is 1.

4	**~ (Bitwise Not)** It is a unary operator and operates by reversing all the bits in the operand. **Ex:** (~B) is -4.
5	**<< (Left Shift)** It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on. **Ex:** (A << 1) is 4.
6	**>> (Right Shift)** Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand. **Ex:** (A >> 1) is 1.
7	**>>> (Right shift with Zero)** This operator is just like the >> operator, except that the bits shifted in on the left are always zero. **Ex:** (A >>> 1) is 1.

Example

Try the following code to implement Bitwise operator in JavaScript.

```
<html>
  <body>
    <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>
     <p>Set the variables to different values and different operators and then try...</p>
   </body>
</html>
```

Assignment Operators

JavaScript supports the following assignment operators −

Sr.No.	Operator & Description
1	**= (Simple Assignment)** Assigns values from the right side operand to the left side operand **Ex:** C = A + B will assign the value of A + B into C
2	**+= (Add and Assignment)** It adds the right operand to the left operand and assigns the result to the left operand. **Ex:** C += A is equivalent to C = C + A
3	**−= (Subtract and Assignment)** It subtracts the right operand from the left operand and assigns the result to the left operand. **Ex:** C -= A is equivalent to C = C - A
4	***= (Multiply and Assignment)**

	It multiplies the right operand with the left operand and assigns the result to the left operand. **Ex:** C *= A is equivalent to C = C * A
5	**/= (Divide and Assignment)** It divides the left operand with the right operand and assigns the result to the left operand. **Ex:** C /= A is equivalent to C = C / A
6	**%= (Modules and Assignment)** It takes modulus using two operands and assigns the result to the left operand. **Ex:** C %= A is equivalent to C = C % A

Note − Same logic applies to Bitwise operators so they will become like <<=, >>=, >>=, &=, |= and ^=.

Example

Try the following code to implement assignment operator in JavaScript.

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var a = 33;
        var b = 10;
        var linebreak = "<br />";

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

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

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

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

        document.write("Value of a => (a /= b) => ");
```

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

            document.write("Value of a => (a %= b) => ");
            result = (a %= b);
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      <p>Set the variables to different values and different operators and then try...</p>
   </body>
</html>
```

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

Try the following code to understand how the Conditional Operator works in JavaScript.

```
<html>
   <body>
      <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>
      <p>Set the variables to different values and different operators and then try...</p>
   </body>
</html>
```

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

The following code shows how to implement **typeof** operator.

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

            result = (typeof b == "string" ? "B is String" : "B is Numeric");
            document.write("Result => ");
            document.write(result);
            document.write(linebreak);

            result = (typeof a == "string" ? "A is String" : "A is Numeric");
            document.write("Result => ");
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      <p>Set the variables to different values and different operators and then try...</p>
   </body>
</html>
```

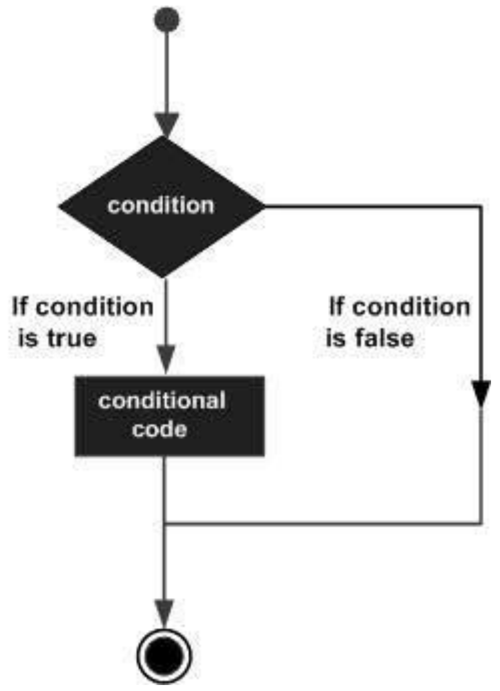
JavaScript-Control Structures

if...else Statement

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.



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

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

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>
```

if...else statement

The **'if...else'** statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

Syntax

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

Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the 'if' block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

Example

Try the following code to learn how to implement an if-else statement in JavaScript.

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

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
}
```

There is nothing special about this code. It is just a series of **if** statements, where each **if** is a part of the **else** clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the **else** block is executed.

Example

Try the following code to learn how to implement an if-else-if statement in JavaScript.

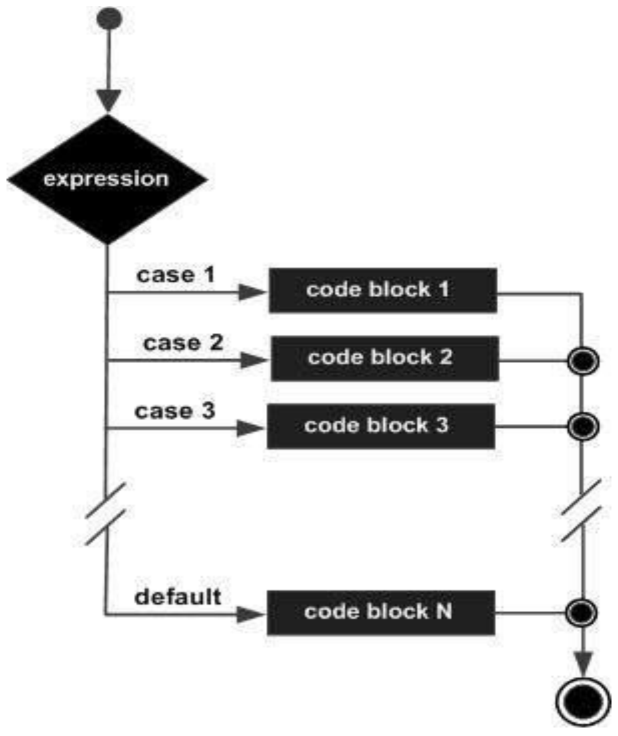
```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var book = "maths";
            if( book == "history" ) {
               document.write("<b>History Book</b>");
            } else if( book == "maths" ) {
               document.write("<b>Maths Book</b>");
            } else if( book == "economics" ) {
               document.write("<b>Economics Book</b>");
            } else {
               document.write("<b>Unknown Book</b>");
            }
         //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
<html>
```

Switch Case

Starting with JavaScript 1.2, you can use a **switch** statement which handles exactly this situation, and it does so more efficiently than repeated **if...else if** statements.

Flow Chart

The following flow chart explains a switch-case statement works.



Syntax

The objective of a **switch** statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each **case** against the value of the expression until a match is found. If nothing matches, a **default** condition will be used.

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

  case condition 2: statement(s)
  break;
  ...

  case condition n: statement(s)
  break;

  default: statement(s)
}
```

The **break** statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

We will explain **break** statement in **Loop Control** chapter.

Example

Try the following example to implement switch-case statement.

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var grade = 'A';
        document.write("Entering switch block<br />");
        switch (grade) {
          case 'A': document.write("Good job<br />");
          break;

          case 'B': document.write("Pretty good<br />");
          break;

          case 'C': document.write("Passed<br />");
          break;

          case 'D': document.write("Not so good<br />");
          break;

          case 'F': document.write("Failed<br />");
          break;

          default:  document.write("Unknown grade<br />")
        }
        document.write("Exiting switch block");
      //-->
    </script>
    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Break statements play a major role in switch-case statements. Try the following code that uses switch-case statement without any break statement.

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var grade = 'A';
        document.write("Entering switch block<br />");
        switch (grade) {
          case 'A': document.write("Good job<br />");
          case 'B': document.write("Pretty good<br />");
          case 'C': document.write("Passed<br />");
          case 'D': document.write("Not so good<br />");
          case 'F': document.write("Failed<br />");
```

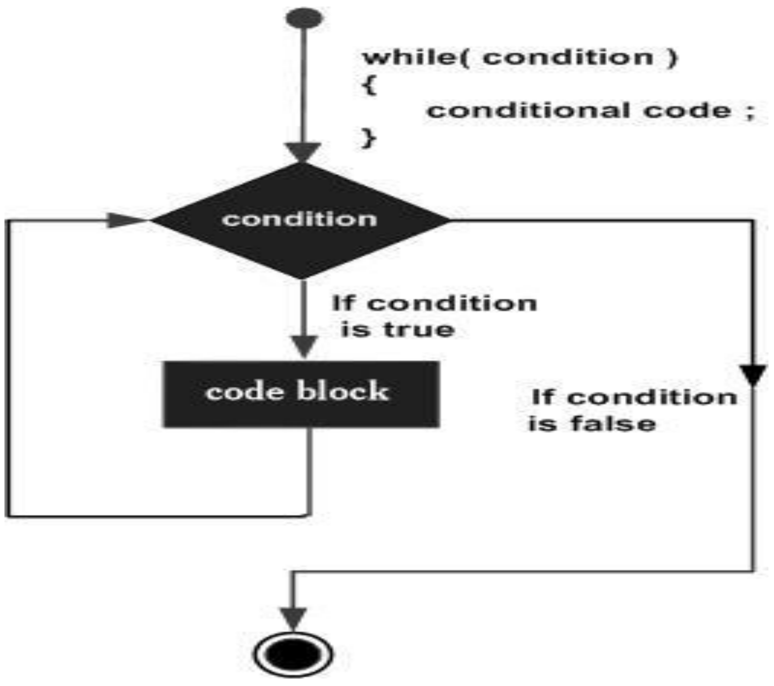
```
            default: document.write("Unknown grade<br />")
          }
          document.write("Exiting switch block");
        //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

While Loops

The most basic loop in JavaScript is the **while** loop which would be discussed in this chapter. 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
}
```

Example

Try the following example to implement while loop.

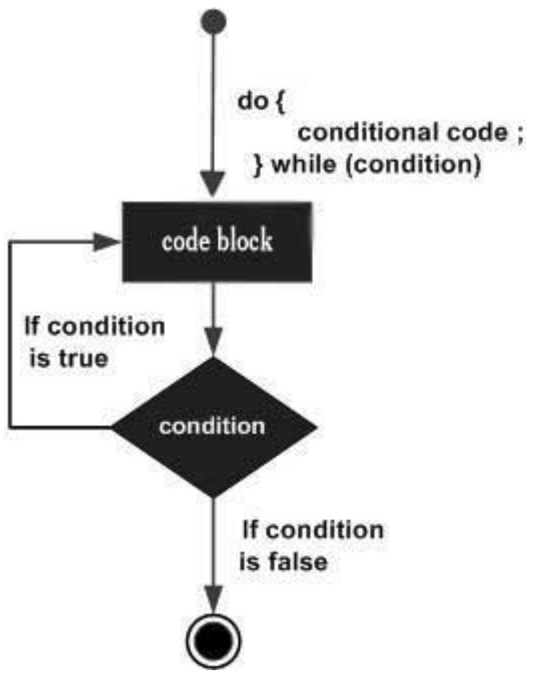
```
<html>
   <body>

      <script type = "text/javascript">
      <!--
        var count = 0;
        document.write("Starting Loop ");

        while (count < 10) {
          document.write("Current Count : " + count + "<br />");
          count++;
        }

        document.write("Loop stopped!");
      //-->
    </script>

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

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 –



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.

Example

Try the following example to learn how to implement a **do-while** loop in JavaScript.

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var count = 0;

        document.write("Starting Loop" + "<br />");
        do {
          document.write("Current Count : " + count + "<br />");
          count++;
        }

        while (count < 5);
        document.write ("Loop stopped!");
      //-->
    </script>
    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

For Loop

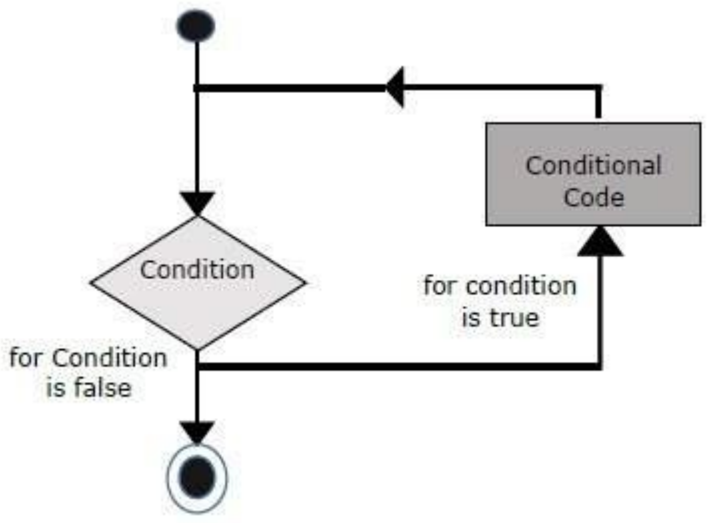
The '**for**' loop is the most compact form of looping. It includes the following three important parts −

- 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.

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
}
```

Example

Try the following example to learn how a **for** loop works in JavaScript.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var count;
            document.write("Starting Loop" + "<br />");

            for(count = 0; count < 10; count++) {
               document.write("Current Count : " + count );
               document.write("<br />");
            }
            document.write("Loop stopped!");
         //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

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

The syntax of ‘for..in’ loop is –

```
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.

Example

Try the following example to implement ‘for-in’ loop. It prints the web browser’s **Navigator** object.

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var aProperty;
        document.write("Navigator Object Properties<br /> ");
        for (aProperty in navigator) {
          document.write(aProperty);
          document.write("<br />");
        }
        document.write ("Exiting from the loop!");
      //-->
    </script>
    <p>Set the variable to different object and then try...</p>
  </body>
</html>
```

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.

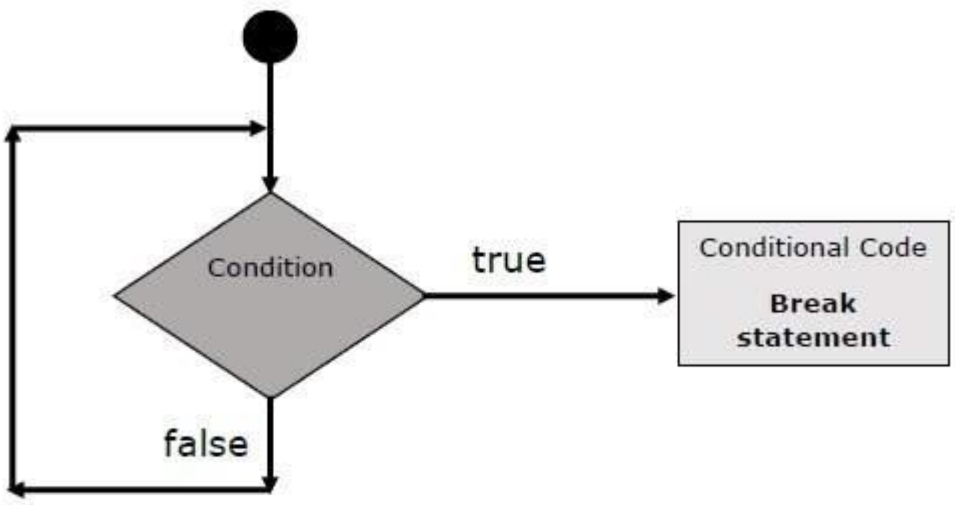
To handle all such situations, JavaScript provides **break** and **continue** statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

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>
```

Output

```
Entering the loop
2
3
4
5
Exiting the loop!
Set the variable to different value and then try...
```

We already have seen the usage of **break** statement inside **a switch** statement.

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>
```

Output

```
Entering the loop
2
3
4
6
7
8
9
10
Exiting the loop!
Set the variable to different value and then try...
```

Using Labels to Control the Flow

Starting from JavaScript 1.2, a label can be used with **break** and **continue** to control the flow more precisely. A **label** is simply an identifier followed by a colon (:) that is applied to a statement or a block of code. We will see two different examples to understand how to use labels with break and continue.

Note − Line breaks are not allowed between the **'continue'** or **'break'** statement and its label name. Also, there should not be any other statement in between a label name and associated loop.

Try the following two examples for a better understanding of Labels.

Example 1

The following example shows how to implement Label with a break statement.

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        document.write("Entering the loop!<br /> ");
        outerloop:        // This is the label name
        for (var i = 0; i < 5; i++) {
          document.write("Outerloop: " + i + "<br />");
          innerloop:
          for (var j = 0; j < 5; j++) {
            if (j > 3 ) break ;           // Quit the innermost loop
            if (i == 2) break innerloop;  // Do the same thing
            if (i == 4) break outerloop;  // Quit the outer loop
            document.write("Innerloop: " + j + " <br />");
          }
        }
        document.write("Exiting the loop!<br /> ");
      //-->
    </script>
  </body>
</html>
```

Output

```
Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 2
Outerloop: 3
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 4
Exiting the loop!
```

Example 2

```
<html>
  <body>

    <script type = "text/javascript">
      <!--
      document.write("Entering the loop!<br /> ");
      outerloop:     // This is the label name

      for (var i = 0; i < 3; i++) {
        document.write("Outerloop: " + i + "<br />");
        for (var j = 0; j < 5; j++) {
          if (j == 3) {
            continue outerloop;
          }
          document.write("Innerloop: " + j + "<br />");
        }
      }

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

  </body>
</html>
```

JavaScript - Events

What is an 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.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

Please go through this small tutorial for a better understanding HTML Event Reference. Here we will see a few examples to understand a relation between Event and JavaScript −

onclick Event Type

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

Example

Try the following example.

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

   <body>
      <p>Click the following button and see result</p>
      <form>
         <input type = "button" onclick = "sayHello()" value =
"Say Hello" />
      </form>
   </body>
</html>
```

onsubmit Event Type

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

Example

The following example shows how to use onsubmit. Here we are calling a **validate()** function before submitting a form data to the webserver. If **validate()** function returns true, the form will be submitted, otherwise it will not submit the data.

Try the following example.

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

   <body>
      <form method = "POST" action = "t.cgi" onsubmit = "return
validate()">
         .......
         <input type = "submit" value = "Submit" />
      </form>
   </body>
</html>
```

onmouseover and onmouseout

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

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function over() {
               document.write ("Mouse Over");
            }
            function out() {
               document.write ("Mouse Out");
            }
         //-->
      </script>
   </head>

   <body>
      <p>Bring your mouse inside the division to see the
result:</p>
      <div onmouseover = "over()" onmouseout = "out()">
         <h2> This is inside the division </h2>
      </div>
   </body>
</html>
```

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.

Function Definition

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

Syntax

The basic syntax is shown here.

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

Example

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

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

Calling a Function

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

Live Demo

```
<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>
```

Function Parameters

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

Example

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

Live Demo

```
<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

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

Live Demo

```
<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>
```

Function Invocation

The code inside a JavaScript `function` will execute when "something" invokes it.

Invoking a JavaScript Function

The code inside a function is not executed when the function is **defined**. The code inside a function is executed when the function is **invoked**. It is common to use the term "**call a function**" instead of "**invoke a function**".

It is also common to say "call upon a function", "start a function", or "execute a function". In this tutorial, we will use **invoke**, because a JavaScript function can be invoked without being called.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Functions</h2>
<p>The global function (myFunction) returns the product of the arguments (a ,b):</p>
<p id="demo"></p>
<script>
function myFunction(a, b) {
  return a * b;
}
document.getElementById("demo").innerHTML = myFunction(10, 2);
</script>
</body>
</html>
```

The function above does not belong to any object. But in JavaScript there is always a default global object.

In HTML the default global object is the HTML page itself, so the function above "belongs" to the HTML page.

In a browser the page object is the browser window. The function above automatically becomes a window function.

Note

This is a common way to invoke a JavaScript function, but not a very good practice.
Global variables, methods, or functions can easily create name conflicts and bugs in the global object.

myFunction() and window.myFunction() is the same function:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Functions</h2>
<p>Global functions automatically become window methods. Invoking myFunction() is the same as invoking window.myFunction().</p>
<p id="demo"></p>
<script>
function myFunction(a, b) {
  return a * b;
}
document.getElementById("demo").innerHTML = window.myFunction(10, 2);
</script>
</body>
```

```
</html>
```

Function Closures

JavaScript variables can belong to the **local** or **global** scope.

Global variables can be made local (private) with **closures**.

Global Variables

A `function` can access all variables defined **inside** the function, like this:

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

<h2>JavaScript Functions</h2>
<p>A function can access variables defined inside the function:</p>

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

<script>
myFunction();

function myFunction() {
  let a = 4;
  document.getElementById("demo").innerHTML = a * a;
}
</script>

</body>
</html>
```

But a `function` can also access variables defined **outside** the function, like this:

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

<h2>JavaScript Functions</h2>
<p>A function can access variables defined outside the function:</p>

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

<script>
let a = 4;
myFunction();

function myFunction() {
  document.getElementById("demo").innerHTML = a * a;
}
</script>

</body>
</html>
```

In the last example, **a** is a **global** variable.

In a web page, global variables belong to the window object.

Global variables can be used (and changed) by all scripts in the page (and in the window).

In the first example, **a** is a **local** variable.

A local variable can only be used inside the function where it is defined. It is hidden from other functions and other scripting code.

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

<h2>JavaScript Closures</h2>

<p>Counting with a local variable.</p>

<button type="button" onclick="myFunction()">Count!</button>

<p id="demo">0</p>

<script>
const add = (function () {
  let counter = 0;
  return function () {counter += 1; return counter;}
})();

function myFunction(){
  document.getElementById("demo").innerHTML = add();
}
</script>

</body>
</html>
```

The variable `add` is assigned to the return value of a self-invoking function.

The self-invoking function only runs once. It sets the counter to zero (0), and returns a function expression.

This way add becomes a function. The "wonderful" part is that it can access the counter in the parent scope.

This is called a JavaScript **closure.** It makes it possible for a function to have "**private**" variables.

The counter is protected by the scope of the anonymous function, and can only be changed using the add function.

A closure is a function having access to the parent scope, even after the parent function has closed.

JavaScript - Objects

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers −

- **Encapsulation** − the capability to store related information, whether data or methods, together in an object.
- **Aggregation** − the capability to store one object inside another object.
- **Inheritance** − the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
- **Polymorphism** − the capability to write one function or method that works in a variety of different ways.

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;
```

For example − The following code gets the document title using the **"title"** property of the **document** object.

```
var str = document.title;
```

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.

For example − Following is a simple example to show how to use the **write()** method of document object to write any content on the document.

```
document.write("This is test");
```

User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called **Object**.

The 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.

In the following example, the constructor methods are Object(), Array(), and Date(). These constructors are built-in JavaScript functions.

```
var employee = new Object();
var books = new Array("C++", "  JS", "Java");
var day = new Date("August 15, 1947");
```

The 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.

Example 1

Try the following example; it demonstrates how to create an Object.

```
<html>
  <head>
    <title>User-defined objects</title>
    <script type = "text/javascript">
      var book = new Object();   // Create the object
      book.subject = "  JS";     // Assign properties to the object
      book.author  = "Richa Arora";
    </script>
  </head>

  <body>
    <script type = "text/javascript">
      document.write("Book name is : " + book.subject + "<br>");
      document.write("Book author is : " + book.author + "<br>");
    </script>
  </body>
</html>
```

Output

```
Book name is :   JS
Book author is : Richa Arora
```

Example 2

This example demonstrates how to create an object with a User-Defined Function. Here **this** keyword is used to refer to the object that has been passed to a function.

```
<html>
  <head>
  <title>User-defined objects</title>
    <script type = "text/javascript">
      function book(title, author) {
        this.title = title;
        this.author  = author;
      }
    </script>
  </head>

  <body>
    <script type = "text/javascript">
      var myBook = new book(“JS", "Richa arora");
      document.write("Book title is : " + myBook.title + "<br>");
      document.write("Book author is : " + myBook.author + "<br>");
    </script>
  </body>
</html>
```

Output

```
Book title is : JS
Book author is : Richa Arora
```

Defining Methods for an Object

The previous examples demonstrate how the constructor creates the object and assigns properties. But we need to complete the definition of an object by assigning methods to it.

Example

Try the following example; it shows how to add a function along with an object.

```
<html>

  <head>
  <title>User-defined objects</title>
    <script type = "text/javascript">
      // Define a function which will work as a method
      function addPrice(amount) {
        this.price = amount;
      }

      function book(title, author) {
        this.title = title;
        this.author  = author;
        this.addPrice = addPrice;  // Assign that method as property.
      }
    </script>
  </head>

  <body>
    <script type = "text/javascript">
      var myBook = new book("  JS", "Richa Arora");
      myBook.addPrice(100);

      document.write("Book title is : " + myBook.title + "<br>");
      document.write("Book author is : " + myBook.author + "<br>");
      document.write("Book price is : " + myBook.price + "<br>");
    </script>
  </body>
```

```
</html>
```

Output

```
Book title is :   JS
Book author is : Richa Arora
Book price is : 100
```

The 'with' Keyword

The **'with'** keyword is used as a kind of shorthand for referencing an object's properties or methods.

The object specified as an argument to **with** becomes the default object for the duration of the block that follows. The properties and methods for the object can be used without naming the object.

Syntax

The syntax for with object is as follows −

```
with (object) {
  properties used without the object name and dot
}
```

Example

Try the following example.

```
<html>
  <head>
  <title>User-defined objects</title>
    <script type = "text/javascript">
      // Define a function which will work as a method
      function addPrice(amount) {
        with(this) {
          price = amount;
        }
      }
      function book(title, author) {
        this.title = title;
        this.author = author;
        this.price = 0;
        this.addPrice = addPrice;  // Assign that method as property.
      }
    </script>
  </head>

  <body>
    <script type = "text/javascript">
      var myBook = new book("  JS", "Richa Arora");
      myBook.addPrice(100);

      document.write("Book title is : " + myBook.title + "<br>");
      document.write("Book author is : " + myBook.author + "<br>");
      document.write("Book price is : " + myBook.price + "<br>");
    </script>
  </body>
</html>
```

JavaScript Native Objects

JavaScript has several built-in or native objects. These objects are accessible anywhere in your program and will work the same way in any browser running in any operating system.

Here is the list of all important JavaScript Native Objects −

- JavaScript Number Object
- JavaScript Boolean Object
- JavaScript String Object
- JavaScript Array Object
- JavaScript Date Object
- JavaScript Math Object
- JavaScript RegExp Object

JavaScript - The Arrays 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

Use the following syntax to create an **Array** object −

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

The **Array** parameter is a list of strings or integers. When you specify a single numeric parameter with the Array constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295.

You can create array by simply assigning values as follows −

```
var fruits = [ "apple", "orange", "mango" ];
```

You will use ordinal numbers to access and to set values inside an array as follows.

```
fruits[0] is the first element
fruits[1] is the second element
fruits[2] is the third element
```

Array Properties

Here is a list of the properties of the Array object along with their description.

Sr.No.	Property & Description
1	constructor Returns a reference to the array function that created the object.
2	**index** The property represents the zero-based index of the match in the string
3	**input** This property is only present in arrays created by regular expression matches.
4	length Reflects the number of elements in an array.
5	prototype The prototype property allows you to add properties and methods to an object.

In the following sections, we will have a few examples to illustrate the usage of Array properties.

Array Methods

Here is a list of the methods of the Array object along with their description.

Sr.No.	Method & Description
1	concat() Returns a new array comprised of this array joined with other array(s) and/or value(s).
2	every() Returns true if every element in this array satisfies the provided testing function.
3	filter() Creates a new array with all of the elements of this array for which the provided filtering function returns true.
4	forEach() Calls a function for each element in the array.
5	indexOf() Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
6	join() Joins all elements of an array into a string.
7	lastIndexOf() Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
8	map() Creates a new array with the results of calling a provided function on every element in this array.
9	pop() Removes the last element from an array and returns that element.

10	push() Adds one or more elements to the end of an array and returns the new length of the array.
11	reduce() Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
12	reduceRight() Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.
13	reverse() Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
14	shift() Removes the first element from an array and returns that element.
15	slice() Extracts a section of an array and returns a new array.
16	some() Returns true if at least one element in this array satisfies the provided testing function.
17	toSource() Represents the source code of an object
18	sort() Sorts the elements of an array
19	splice() Adds and/or removes elements from an array.
20	toString() Returns a string representing the array and its elements.
21	unshift() Adds one or more elements to the front of an array and returns the new length of the array.