**JAVASCRIPT BASICS TUTORIAL**

* **JavaScript – Home:**
* **What is JavaScript?**

JavaScript was introduced in **1995** as a way to add programs to webpages in the Netscape Navigator browser. The language has since been adopted by all other major graphical web browsers. It has made modern web applications possible—applications with which you can interact directly without doing a page reload for every action. Java Script is also used in more traditional websites to provide various forms of interactivity and cleverness.

It is important to note that JavaScript has almost nothing to do with the programming language named Java. The similar name was inspired by marketing considerations rather than good judgment. When JavaScript was being introduced, the Java language was being heavily marketed and was gaining popularity. Someone thought it was a good idea to try to ride along on this success. Now we are stuck with the name.

After its adoption outside of Netscape, a standard document was written to describe the way the JavaScript language should work so that the various pieces of software that claimed to support JavaScript were actually talking about the same language. This is called the ECMAScript standard, after the Ecma International organization that did the standardization. In practice, the terms ECMAScript and JavaScript can be used interchangeably—they are two names for the same language.

* **Key Features of JavaScript**
* JavaScript is a lightweight, interpreted **programming** language.
* JavaScript is the programing language of **HTML** and **web.**
* 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**.
* **Did You Know?**

**JavaScript** and [**Java**](https://www.w3schools.com/java/default.asp)are completely **different** languages, both in concept and design. JavaScript was first known as **LiveScript,** but Netscape changed its name to JavaScript. JavaScript was **invented by Brendan Eich in 1995**, and became an ECMA standard in 1997. **ECMA-262** is the **official name** of the standard. **ECMAScript** is the official name of the language

## **Why Study JavaScript?**

* JavaScript is one of the **3 languages** all web developers **must** learn:

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

* Web pages are not the only place where JavaScript is used. Many desktop and server programs use JavaScript. Node.js is the best known. Some databases, like MongoDB and CouchDB, also use JavaScript as their programming language.
* JavaScript usage has now extended to mobile app development, desktop app development, and game development. This opens many opportunities for you as JavaScript Programmer.
* JavaScript is everywhere, it comes installed on every modern web browser and so to learn JavaScript you really do not need any special environment setup. For example Chrome, Safari, Mozilla Firefox and every browser you know as of today, supports JavaScript.
* JavaScript is the most popular **programming language** in the world and that makes it a programmer’s great choice. Once you learnt JavaScript, it helps you developing great front-end as well as back-end softwares using different JavaScript based frameworks like jQuery, Node.JS etc.
* JavaScript helps you create really beautiful and crazy fast websites. You can develop your website with a console like look and feel and give your users the best Graphical User Experience.
* Great thing about JavaScript is that you will find tons of frameworks and Libraries already developed which can be used directly in your software development to reduce your time to market.
* Due to high demand, there is tons of job growth and high pay for those who know JavaScript. You can navigate over to different job sites to see what having JavaScript skills looks like in the job market.

There are many useful **JavaScript frameworks** and libraries available:

* Angular
* React
* jQuery
* Vue.js
* Ext.js
* Ember.js
* Meteor
* Mithril
* Node.js
* Polymer
* Aurelia
* Backbone.js

It is really impossible to give a complete list of all the available JavaScript frameworks and libraries. The JavaScript world is just too large and too much new is happening.

|  |
| --- |
| **\*\*Prerequisites** For better understanding of JavaScript, learner must have a prior knowledge of HTML coding. It would help if the learner had some prior exposure to object-oriented programming concepts and a general idea on creating online applications. |

## **Applications of JavaScript Programming**

As mentioned before, **JavaScript** is one of the most widely used **programming languages** (Front-end as well as Back-end). It has its presence in almost every area of software development. I'm going to list few of them here:

* **User Notifications** - You can use JavaScript to raise dynamic pop-ups on the webpages to give different types of notifications to your website visitors.
* **Manipulating HTML Pages** - JavaScript helps in manipulating HTML page on the fly. This helps in adding and deleting any HTML tag very easily using JavaScript and modify your HTML to change its look and feel based on different devices and requirements.
* **Presentations** - JavaScript also provides the facility of creating presentations which gives website look and feel. JavaScript provides RevealJS and BespokeJS libraries to build a web-based slide presentations.
* **Client side validation** - This is really important to verify any user input before submitting it to the server and JavaScript plays an important role in validating those inputs at front-end itself.
* **Server Applications** - Node JS is built on Chrome's JavaScript runtime for building fast and scalable network applications. This is an event based library which helps in developing very sophisticated server applications including Web Servers.
* **Back-end Data Loading** - JavaScript provides Ajax library which helps in loading back-end data while you are doing some other processing. This really gives an amazing experience to your website visitors.
* **JavaScript – Overview:**

## **Advantages of JavaScript**

The merits of using JavaScript are:–

* **Increased interactivity** − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
* **Immediate feedback to the visitors** − They don't have to wait for a page reload to see if they have forgotten to enter something.
* **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.
* **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.

## **Limitations of JavaScript**

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

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

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

## **What is 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.

## **JavaScript Development Tools**

One of major strengths of JavaScript is that it does not require expensive development tools. You can start with a simple text editor such as Notepad. Since it is an interpreted language inside the context of a web browser, you don't even need to buy a compiler.

To make our life simpler, various vendors have come up with very nice JavaScript editing tools. Some of them are listed here –

* **Macromedia HomeSite 5** − HomeSite 5 is a well-liked HTML and JavaScript editor from Macromedia that can be used to manage personal websites effectively.
* **Microsoft FrontPage** − Microsoft has developed a popular HTML editor called FrontPage. FrontPage also provides web developers with a number of JavaScript tools to assist in the creation of interactive websites.
* **Macromedia Dreamweaver MX** − Macromedia Dreamweaver MX is a very popular HTML and JavaScript editor in the professional web development crowd. It provides several handy prebuilt JavaScript components, integrates well with databases, and conforms to new standards such as XHTML and XML.

## **Where is JavaScript Today?**

The ECMAScript Edition 5 standard will be the first update to be released in over four years. JavaScript 2.0 conforms to Edition 5 of the ECMAScript standard, and the difference between the two is extremely minor.

The specification for JavaScript 2.0 can be found on the following site: <http://www.ecmascript.org/>

Today, Netscape's JavaScript and Microsoft's JScript conform to the ECMAScript standard, although both the languages still support the features that are not a part of the standard.

* **JavaScript – Syntax:**

## **The <script> Tag**

## In HTML, JavaScript code is inserted between <script> and </script> tags. 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. You can place any number of scripts in an HTML document.

<script ...>

JavaScript code

</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>

* **Do you know?**
* **Old Javascript** Examples may use a language attribute and type attribute:

**<script language = “javascript” type = “text/javascript”>**

The **type attribute is not** required. Javascript is the **default scripting language** in HTML.

* We can write <script> tag within <head> tag **But** placing <script> tag at the bottom of the **<body>** tag **improves the display speed**, because script interpretation slows down the display.
* External scripts **cannot** contain <script> tag.

## **Your First JavaScript Code**

**Program :**

### <!DOCTYPE html>

<html>

<body>

<script>

<!--

document.write("Hello World!")

//-->

</script>

</body>

</html>

**Output :**

Hello World!

**Explanation :**

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "//-->". Here "//" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code. Next, we call a function **document.write** which writes a string into our HTML document.

## **JavaScript Display Possibilities: 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:

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

\*\*Changing the innerHTML property of an HTML element is a common way to display data in HTML

## **Using document.write()**

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

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

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

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

## **Using window.alert()**

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

## **Using console.log()**

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

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

## **Semicolons ;**

Semicolons separate JavaScript statements. Add a semicolon at the end of each executable statement:

\*\*On the web, you might see examples without semicolons. Ending statements with semicolon is not required but highly recommended.

## **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.

* **Comments in JavaScript**

JavaScript comments can be used to explain JavaScript code, and to make it more readable.

JavaScript comments can also be used to prevent execution, when testing alternative code.

JavaScript supports both C-style and C++-style comments :

* 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 //-->.

The following example shows how to use comments in JavaScript.

**Example :**

<script>

<!--

// 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 – Enabling:**

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 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 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 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 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:

### <!DOCTYPE html>

<html>

<body>

<script>

<!--

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:**

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 <body>...</body> section.
* Script in <head>...</head> 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 <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.

**Program:**

### <!DOCTYPE html>

<html>

<head>

</head>

<body>

<script>

<!--

document.write("Hello World");

//-->

</script>

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

</body>

</html>

**Output:**

Hello World

This is web page body

We can write <script> tag within <head> tag **But** placing <script> tag at the bottom of the **<body>** tag **improves the display speed**, because script interpretation slows down the display.

## **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 –

**Program:**

### <!DOCTYPE html>

<html>

<head>

<script>

<!--

function sayHello() {

alert("Hello World");

}

//-->

</script>

</head>

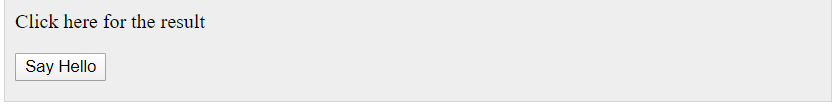
<body>

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

</body>

</html>

**Output:**



## **JavaScript in <body> and <head> Sections**

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

**Program:**

### <!DOCTYPE html>

<html>

<head>

<script>

<!--

function sayHello() {

alert("Hello World");

}

//-->

</script>

</head>

<body>

<script>

<!--

document.write("Hello World")

//-->

</script>

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

</body>

</html>

**Output:**

## **External JavaScript**

Scripts can also be placed in external files. 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.

**External file : myScript.js**

function myFunction() {  
 document.getElementById("demo").innerHTML = "Paragraph changed.";  
}

External scripts are practical when the same code is used in many different web pages. JavaScript files have the file extension **.js**. To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:

<script src="myScript.js"></script>

You can place an external script reference in <head> or <body> as you like. The script will behave as if it was located exactly where the <script> tag is located.

External scripts cannot contain <script> tag

* **JavaScript – Variables:**

## **Datatypes in JavaScript:**

Datatypes 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** e.g. 123, 120.50 etc.
      * **Strings** of text e.g. "This text string" etc.
      * **Boolean** e.g. true or false.

JavaScript also defines two trivial data types, **null** and **undefined,** each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as **object**.

**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 represents numbers using the 64-bit floating-point format defined by the IEEE 754 standard.

## **Variables in JavaScript:**

JavaScript variables are containers for storing data values. 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 as follows:

<script>

<!--

var money;

var name;

//-->

</script>

You can also declare multiple variables with the same **var** keyword as follows –

<script>

<!--

var money, name;

//-->

</script>

Storing a value in a variable is called **variable initialization**. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named **money** and assign the value 100.25 to it later. For another variable, you can assign a value at the time of initialization as follows.

<script>

<!--

var name = "Ali";

var money;

money = 100.25;

//-->

</script>

**Note** − Use the **var** keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.

* **Do you know?**

JavaScript is **untyped** language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

## **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.

**Program:**

### <!DOCTYPE html>

<html>

<body onload = checkscope();>

<script>

<!--

var myVar = "global"; // Declare a global variable

function checkscope( ) {

var myVar = "local"; // Declare a local variable

document.write(myVar);

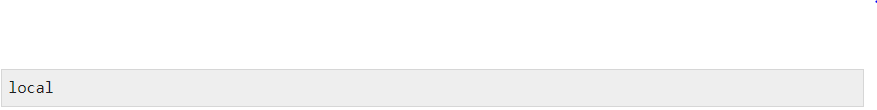
}

//-->

</script>

</body>

</html>

**Output:**

## **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 \_
* Reserved words (like JavaScript keywords) cannot be used as names
* You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, **break** or **boolean** variable names are not valid.
* JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, **123test** is an invalid variable name but **\_123test** is a valid one.

Javascript identifiers are case-sensitive.  For example, **Name** and **name** are two different variables.

## **JavaScript Dollar Sign $**

Remember that JavaScript identifiers (names) must begin with:

* + A letter (A-Z or a-z)
  + A dollar sign ($)
  + Or an underscore (\_)

Since JavaScript treats a dollar sign as a letter, identifiers containing $ are valid variable names

var $$$ = "Hello World";  
 var $ = 2;  
 var $myMoney = 5;

* **Do you know?**

Using the dollar sign is not very common in JavaScript, but professional programmers often use it as an alias for the main function in a JavaScript library.

In the JavaScript library jQuery, for instance, the main function $ is used to select HTML elements. In jQuery $("p"); means "select all p elements".

‘\_’ by professional programmers is to use it as an alias for "private (hidden)" variables.

If you re-declare a JavaScript variable, it will not lose its value.

## **JavaScript Reserved Words**

A list of all the reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names. There are 59 reserved keywords in Javascript.

|  |
| --- |
|  |
| abstract | else | instanceof | switch |
| boolean | enum | int | synchronized |
| break | export | interface | this |
| byte | extends | long | throw |
| case | false | native | throws |
| catch | final | new | transient |
| char | finally | null | true |
| class | float | package | try |
| const | for | private | typeof |
| continue | function | protected | var |
| debugger | goto | public | void |
| default | if | return | volatile |
| delete | implements | short | while |
| do | import | static | with |
| double | in | super |  |

* **JavaScript – Operators:**

## **What is an Operator?**

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

Let’s 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** | **Operator Name** | **Description** | **Example** |
| 1] | **+** | Addition | Adds two or more operands | A + B will give 30 |
| 2] | **-** | Subtraction | Subtracts the second operand from the first | A - B will give -10 |
| 3] | **\*** | Multiplication | Multiply both operands | A \* B will give 200 |
| 4] | **/** | Division | Divide the numerator by the denominator | B / A will give 2 |
| 5] | **%** | Modulus | Outputs the remainder of an integer division | B % A will give 0 |
| 6] | **++** | Increment | Increases an integer value by one | A++ will give 11 |
| 7] | -- | Decrement | Decreases an integer value by one | A-- will give 9 |

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

**Program:**

### <!DOCTYPE html>

<html>

<body>

<script>

<!--

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>

### Output:

### 

## **Comparison Operators**

JavaScript supports the following comparison operators. Assume variable A holds 10 and variable B holds 20, then −

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No.** | **Operator** | **Operator Name** | **Description** | **Example** |
| 1] | **==** | Equal | Checks if the value of two operands are equal or not, if yes, then the condition becomes true. | (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. | (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. | (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. | (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. | (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. | (A <= B) is true |

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

**Program:**

### <!DOCTYPE html>

<html>

<body>

<script>

<!--

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);

//-->

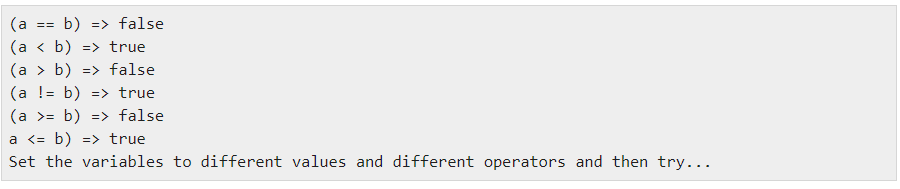
</script>

Set the variables to different values and different operators and then try...

</body>

</html>

**Output:**



## **Logical Operators**

JavaScript supports the following logical operators. Assume variable A holds 10 and variable B holds 20, then –

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No.** | **Operator** | **Operator Name** | **Description** | **Example** |
| 1] | **&&** | Logical AND | If both the operands are non-zero, then the condition becomes true | (A && B) is true. |
| 2] | **||** | Logical OR | Subtracts the second operand from the first | (A || B) is true. |
| 3] | **!** | Logical NOT | Multiply both operands | ! (A && B) is false. |

The following code to learn how to implement Logical Operators in JavaScript.

**Program:**

### <!DOCTYPE html>

<html>

<body>

<script>

<!--

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);

//-->

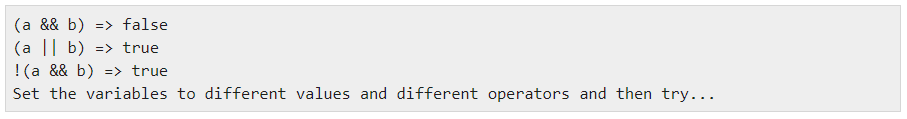
</script>

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

</body>

</html>

**Output:**



## **Bitwise Operators**

JavaScript supports the following bitwise operators. Assume variable A holds 2 and variable B holds 3, then −

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No.** | **Operator** | **Operator Name** | **Description** | **Example** |
| 1] | **&** | Bitwise AND | It performs a Boolean AND operation on each bit of its integer arguments. | (A & B) is 2 |
| 2] | **|** | BitWise OR | It performs a Boolean OR operation on each bit of its integer arguments. | (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. | (A ^ B) is 1 |
| 4] | **~** | Bitwise Not | It is a unary operator and operates by reversing all the bits in the operand. | (~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. | (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. | (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. | (A >>> 1) is 1 |

The following code to implement Bitwise operator in JavaScript

**Program:**

### <!DOCTYPE html>

<html>

<body>

<script>

<!--

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);

//-->

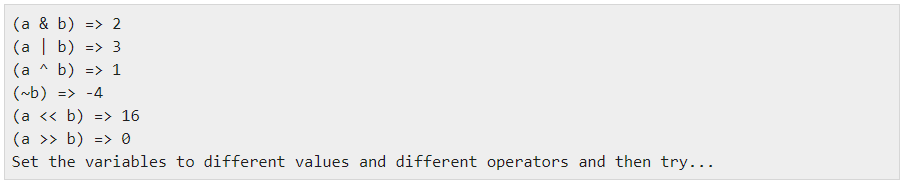
</script>

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

</body>

</html>

**Output:**



## **Assignment Operators**

JavaScript supports the following assignment operators –

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No.** | **Operator** | **Operator Name** | **Description** | **Example** |
| 1] | **=** | Simple Assignment | Assigns values from the right side operand to the left side operand | 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. | 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 | 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. | 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. | 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. | C %= A is equivalent to C = C % A |

 The following code to implement assignment operator in JavaScript.

**Program:**

### <!DOCTYPE html>

<html>

<body>

<script>

<!--

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);

//-->

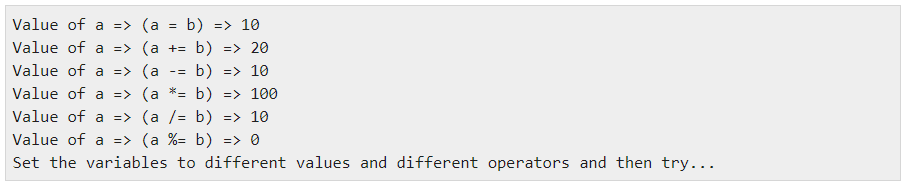
</script>

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

</body>

</html>

**Output:**



## **Conditional** **Operators**

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** | **Operator Name** | **Description** |
| 1] | **?:** | Conditional | If Condition is true? Then value X : Otherwise value Y |

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

**Program:**

### <!DOCTYPE html>

<html>

<body>

<script>

<!--

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);

//-->

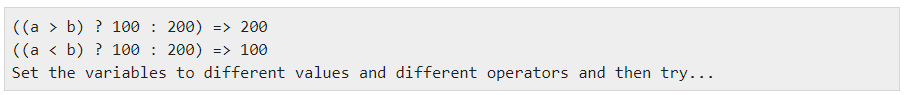
</script>

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

</body>

</html>

**Output:**



## **typedef** **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" |

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

**Program:**

### <!DOCTYPE html>

<html>

<body>

<script>

<!--

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);

//-->

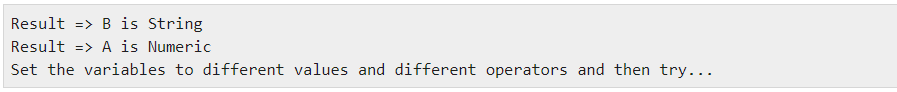
</script>

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

</body>

</html>

**Output:**



* **Do you know?**

## **JavaScript String Operators**

* The + operator can also be used to add (concatenate) strings.

var txt1 = "John";  
 var txt2 = "Doe";  
 var txt3 = txt1 + " " + txt2;

**Result:**



* The += assignment operator can also be used to add (concatenate) strings

var txt1 = "What a very ";  
 txt1 += "nice day";

**Result:**



* **JavaScript – If...Else:**

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.

In JavaScript we have the following conditional statements:

* Use if to specify a block of code to be executed, if a specified condition is true
* Use if..else to specify a block of code to be executed, if the same condition is false
* Use else if to specify a new condition to test, if the first condition is false
* Use switch to specify many alternative blocks of code to be executed (will see in switch block)

## **Flow Chart of if-else**



## **if statement**

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

### Syntax:

if (*condition*) {  
  //  block of code to be executed if the condition is true}

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

**Program:**

### <!DOCTYPE html>

<html>

<body>

<script>

<!--

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>

**Output:**

**Qualifies for driving**

Set the variable to different value and then try...

## **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 (*condition*) {  
  //  block of code to be executed if the condition is true} else {  
  //  block of code to be executed if the condition is false}

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

**Program:**

### <!DOCTYPE html>

<html>

<body>

<script>

<!--

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>

**Output:**

**Does not qualify for driving**

Set the variable to different value and then try...

## **The 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:

if (*condition1*) {  
  //  block of code to be executed if condition1 is true} else if (*condition2*) {  
  //  block of code to be executed if the condition1 is false and condition2 is true  
} else {  
  //  block of code to be executed if the condition1 is false and condition2 is false}

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.

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

**Program:**

### <!DOCTYPE html>

<html>

<body>

<script>

<!--

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>

### Output:

**Maths Book**

Set the variable to different value and then try...

* **JavaScript – Switch Case:**

You can use multiple **if...else…if** statements, as in the previous chapter, to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

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.

**Syntax:**

switch(*expression*) {  
  case *x*:  
*// code block*    break;  
  case *y*:  
*// code block*    break;  
  default:  
    // *code block*  
}

This is how it works:

* The switch expression is evaluated once.
* The value of the expression is compared with the values of each case.
* If there is a match, the associated block of code is executed.
* If there is no match, the default code block is executed.

The following example to implement switch-case statement.

**Program:**

### <!DOCTYPE html>

<html>

<body>

<script>

<!--

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>

### Output:

Entering switch block

Good job

Exiting switch block

Set the variable to different value and then try...

## **The break Keyword**

When JavaScript reaches a break keyword, it breaks out of the switch block.This will stop the execution of inside the block.It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

If you omit the break statement, the next case will be evaluated even if the evaluation does not match the case.

## **Switching Details**

If multiple cases matches a case value, the **first** case is selected.

If no matching cases are found, the program continues to the **default** label.

If no default label is found, the program continues to the statement(s) **after the switch**.

* **JavaScript – While Loop:**

## **Loops in JavaScript**

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.

JavaScript supports all the necessary loops to ease down the pressure of programming.

## **Different Kinds of Loops**

JavaScript supports different kinds of loops:

* + for - loops through a block of code a number of times
  + for/in - loops through the properties of an object
  + for/of - loops through the values of an iterable object
  + while - loops through a block of code while a specified condition is true
  + do/while - also loops through a block of code while a specified condition is true

## **The while Loop**

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 (condition) {  
*// code block to be executed*  
}

The following example to implement while loop.

**Program:**

### <!DOCTYPE html>

<html>

<body>

<script>

<!--

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>

### Output:

Starting Loop

Current Count : 0

Current Count : 1

Current Count : 2

Current Count : 3

Current Count : 4

Current Count : 5

Current Count : 6

Current Count : 7

Current Count : 8

Current Count : 9

Loop stopped!

Set the variable to different value and then try...

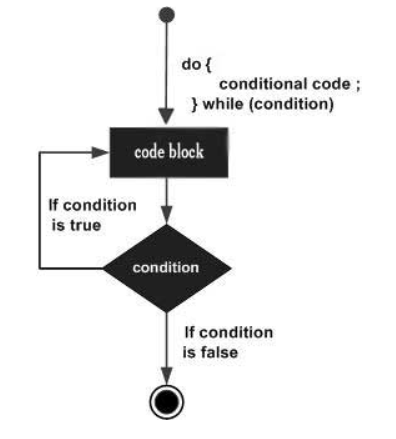
If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser

## **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 {  
*// code block to be executed*}  
while (condition);

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

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

**Program:**

### <!DOCTYPE html>

<html>

<body>

<script>

<!--

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>

### Output:

Starting Loop

Current Count : 0

Current Count : 1

Current Count : 2

Current Count : 3

Current Count : 4

Loop Stopped!

Set the variable to different value and then try...

* **JavaScript – 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.

**syntax:**

for (*statement 1*;*statement 2*;*statement 3*) {  
  // *code block to be executed*  
}

**Statement 1** is executed (one time) before the execution of the code block.

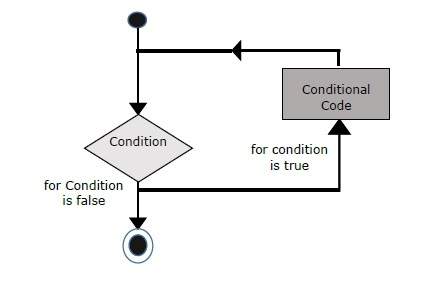
**Statement 2** defines the condition for executing the code block.

**Statement 3** is executed (every time) after the code block has been executed.

* you can omit statement 1 (like when your values are set before the loop starts):
* If you omit statement 2, you must provide a break inside a loop. Otherwise the loop will never end. This will crash your browser.

## **Flow Chart**

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



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

**Program:**

### <!DOCTYPE html>

<html>

<body>

<script>

<!--

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>

### Output:

Starting Loop

Current Count : 0

Current Count : 1

Current Count : 2

Current Count : 3

Current Count : 4

Current Count : 5

Current Count : 6

Current Count : 7

Current Count : 8

Current Count : 9

Loop stopped!

Set the variable to different value and then try...

* **JavaScript – For...in:**

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.

**Program:**

### <!DOCTYPE html>

<html>

<body>

<h2>JavaScript For/In Loop</h2>

<p>The for/in statement loops through the properties of an object.</p>

<script>

var txt = "";

var person = {fname:"John", lname:"Doe", age:25};

var x;

for (x in person) {

txt += person[x] + " ";

}

document.write(txt);

</script>

</body>

</html>

**Output:**

## **JavaScript For/In Loop**

The for/in statement loops through the properties of an object.

John Doe 25

## **The For/Of Loop**

The JavaScript for/of statement loops through the values of an iterable objects for/of lets you loop over data structures that are iterable such as Arrays, Strings, Maps, NodeLists, and more.

The for/of loop has the following **syntax**:

for (*variable* of *iterable*) {  
  // *code block to be executed*  
}

variable - For every iteration the value of the next property is assigned to the variable. Variable can be declared with const, let, or var.

iterable - An object that has iterable properties.

### Looping over an Array

### Program:

### <!DOCTYPE html>

### <html>

### <body>

### <h2>JavaScript For/Of Loop</h2>

### <p>The for/of statement loops through the values of an iterable object.</p>

### <script>

### var cars = ['BMW', 'Volvo', 'Mini'];

### var x;

### for (x of cars) {

### document.write(x + "<br >");

### }

### </script>

### </body>

### </html>

**Output:**

## **JavaScript For/Of Loop**

The for/of statement loops through the values of an iterable object.

BMW  
Volvo  
Mini

### Looping over a String

**Program:**

### <!DOCTYPE html>

<html>

<body>

<h2>JavaScript For/Of Loop</h2>

<p>The for/of statement loops through the values of an iterable object.</p>

<script>

var txt = 'JavaScript';

var x;

for (x of txt) {

document.write(x + "<br >");

}

</script>

</body>

</html>

**Output:**

## **JavaScript For/Of Loop**

The for/of statement loops through the values of an iterable object.

J  
a  
v  
a  
S  
c  
r  
i  
p  
t

* **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 "**jumps out**" of a loop.

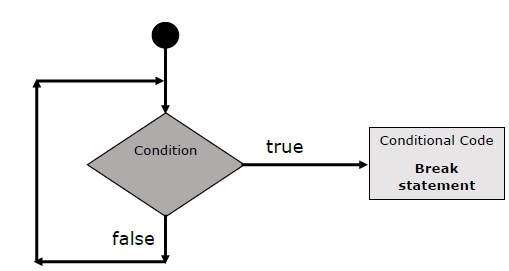
The continue statement "**jumps over**" one iteration in the loop.

## **The Break Statement**

The break statement breaks the loop and continues executing the code after the loop (if any).

* **Flow Chart**

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



**Program:**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Loops</h2>

<p>A loop with a <b>break</b> statement.</p>

<script>

var text = "";

var i;

for (i = 0; i < 10; i++) {

if (i === 3) { break; }

text += "The number is " + i + "<br>";

}

document.write(text);

</script>

</body>

</html>

**Output:**

## **JavaScript Loops**

A loop with a **break** statement.

The number is 0  
The number is 1  
The number is 2

## **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.

**Program:**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Loops</h2>

<p>A loop with a <b>continue</b> statement.</p>

<p>A loop which will skip the step where i = 3.</p>

<script>

var text = "";

var i;

for (i = 0; i < 10; i++) {

if (i === 3) { continue; }

text += "The number is " + i + "<br>";

}

document.write(text);

</script>

</body>

</html>

**Output:**

## **JavaScript Loops**

A loop with a **continue** statement.

A loop which will skip the step where i = 3.

The number is 0  
The number is 1  
The number is 2  
The number is 4  
The number is 5  
The number is 6  
The number is 7  
The number is 8  
The number is 9

## **JavaScript Labels**

To label JavaScript statements you precede the statements with a label name and a colon:

label:  
statements

The break and the continue statements are the only JavaScript statements that can "jump out of" a code block.

**Syntax:**

break labelname;  
continue labelname;

The continue statement (with or without a label reference) can only be used to **skip one loop iteration**.

The break statement, without a label reference, can only be used to **jump out of a loop or a switch**.

With a label reference, the break statement can be used to **jump out of any code block**:

**Program:**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript break</h2>

<script>

var cars = ["BMW", "Volvo", "Saab", "Ford"];

var text = "";

list: {

text += cars[0] + "<br>";

text += cars[1] + "<br>";

break list;

text += cars[2] + "<br>";

text += cars[3] + "<br>";

}

document.write(text);

</script>

</body>

</html>

**Output:**

## **JavaScript break**

BMW  
Volvo

* **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.

Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions.

JavaScript allows us to write our own functions as well. This section explains how to write your own functions in JavaScript.

## **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.

## **JavaScript Function Syntax**

A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses **()**.Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:  
 **(parameter1, parameter2, ...)**The code to be executed, by the function, is placed inside curly brackets: **{}**

function name(parameter list) {  
  // code to be executed  
}

Function **parameters** are listed inside the parentheses () in the function definition. Function **arguments** are the **values** received by the function when it is invoked. Inside the function, the arguments (the parameters) behave as local variables.

## **Function Invocation**

The code inside the function will execute when "something" **invokes** (calls) the function:

* When an event occurs (when a user clicks a button)
* When it is invoked (called) from JavaScript code
* Automatically (self invoked)

## **Function Return**

When JavaScript reaches a return statement, the function will stop executing. If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement. This statement should be the last statement in a function. Functions often compute a **return value**. The return value is "returned" back to the "caller":

**Program:**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Functions</h2>

<p>This example calls a function which performs a calculation and returns the result:</p>

<script>

var x = myFunction(4, 3);

document.write(x);

function myFunction(a, b) {

return a \* b;

}

</script>

</body>

</html>

**Output:**

## **JavaScript Functions**

This example calls a function which performs a calculation and returns the result.

12

## **Why Functions?**

You can reuse code: Define the code once, and use it many times. You can use the same code many times with different arguments, to produce different results.

## **Functions Used as Variable Values**

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

## **Local Variables**

Variables declared within a JavaScript function, become **LOCAL** to the function. Local variables can only be accessed from within the function. Since local variables are only recognized inside their functions, variables with the same name can be used in different functions. Local variables are created when a function starts, and deleted when the function is completed.

* **JavaScript – Events:**

HTML events are **"things"** that happen to HTML elements. When JavaScript is used in HTML pages, JavaScript can **"react"** on these events

* **HTML Events**

An HTML event can be something the browser does, or something a user does. Here are some examples of HTML events:

* + An HTML web page has finished loading
  + An HTML input field was changed
  + An HTML button was clicked

Often, when events happen, you may want to do something. JavaScript lets you execute code when events are detected. HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

## **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.

## **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.

### Program:

<!DOCTYPE html>

<html>

<head>

<script>

<!--

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>

### Output:

### 

## **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.

### Program:

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.

<!DOCTYPE html>

<html>

<head>

<script>

<!--

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.

<!DOCTYPE html>

<html>

<head>

<script>

<!--

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>

## **What can JavaScript Do?**

Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

* + Things that should be done every time a page loads
  + Things that should be done when the page is closed
  + Action that should be performed when a user clicks a button
  + Content that should be verified when a user inputs data
  + And more ...

Many different methods can be used to let JavaScript work with events:

* + HTML event attributes can execute JavaScript code directly
  + HTML event attributes can call JavaScript functions
  + You can assign your own event handler functions to HTML elements
  + You can prevent events from being sent or being handled
  + And more ...

## **Common HTML Events**

Here is a list of some common HTML events:

|  |  |
| --- | --- |
| **Event** | **Description** |
| onchange | An HTML element has been changed |
| onclick | The user clicks an HTML element |
| onmouseover | The user moves the mouse over an HTML element |
| onmouseout | The user moves the mouse away from an HTML element |
| onkeydown | The user pushes a keyboard key |
| onload | The browser has finished loading the page |

* **JavaScript – Cookies:**

## **What are Cookies?**

Web Browsers and Servers use HTTP protocol to communicate and HTTP is a stateless protocol. But for a commercial website, it is required to maintain session information among different pages. For example, one user registration ends after completing many pages. But how to maintain users' session information across all the web pages.

In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

## **How It Works?**

Your server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now, when the visitor arrives at another page on your site, the browser sends the same cookie to the server for retrieval. Once retrieved, your server knows/remembers what was stored earlier.

Cookies are a plain text data record of 5 variable-length fields −

* **Expires** − The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
* **Domain** − The domain name of your site.
* **Path** − The path to the directory or web page that set the cookie. This may be blank if you want to retrieve the cookie from any directory or page.
* **Secure** − If this field contains the word "secure", then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
* **Name=Value** − Cookies are set and retrieved in the form of key-value pairs

Cookies were originally designed for CGI programming. The data contained in a cookie is automatically transmitted between the web browser and the web server, so CGI scripts on the server can read and write cookie values that are stored on the client.

JavaScript can also manipulate cookies using the **cookie** property of the **Document** object. JavaScript can read, create, modify, and delete the cookies that apply to the current web page.

## **Storing Cookies**

The simplest way to create a cookie is to assign a string value to the document.cookie object, which looks like this.

document.cookie = "key1 = value1;key2 = value2;expires = date";

Here the **expires** attribute is optional. If you provide this attribute with a valid date or time, then the cookie will expire on a given date or time and thereafter, the cookies' value will not be accessible.

**Note** − Cookie values may not include semicolons, commas, or whitespace. For this reason, you may want to use the JavaScript **escape()** function to encode the value before storing it in the cookie. If you do this, you will also have to use the corresponding **unescape()** function when you read the cookie value.

### Program:

<!DOCTYPE html>

<html>

<head>

<script>

<!--

function WriteCookie() {

if( document.myform.customer.value == "" ) {

alert("Enter some value!");

return;

}

cookievalue = escape(document.myform.customer.value) + ";";

document.cookie = "name=" + cookievalue;

document.write ("Setting Cookies : " + "name=" + cookievalue );

}

//-->

</script>

</head>

<body>

<form name = "myform" action = "">

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

<input type = "button" value = "Set Cookie" onclick = "WriteCookie();"/>

</form>

</body>

</html>

### Output:

### 

### After set cookie (entered name swap)

### 

Now your machine has a cookie called **name**. You can set multiple cookies using multiple key = value pairs separated by comma.

## **Reading Cookies**

Reading a cookie is just as simple as writing one, because the value of the document.cookie object is the cookie. So you can use this string whenever you want to access the cookie. The document.cookie string will keep a list of name=value pairs separated by semicolons, where **name** is the name of a cookie and value is its string value.

You can use strings' **split()** function to break a string into key and values as follows:

### Program:

<!DOCTYPE html>

<html>

<head>

<script>

<!--

function ReadCookie() {

var allcookies = document.cookie;

document.write ("All Cookies : " + allcookies );

// Get all the cookies pairs in an array

cookiearray = allcookies.split(';');

// Now take key value pair out of this array

for(var i=0; i<cookiearray.length; i++) {

name = cookiearray[i].split('=')[0];

value = cookiearray[i].split('=')[1];

document.write ("Key is : " + name + " and Value is : " + value);

}

}

//-->

</script>

</head>

<body>

<form name = "myform" action = "">

<p> click the following button and see the result:</p>

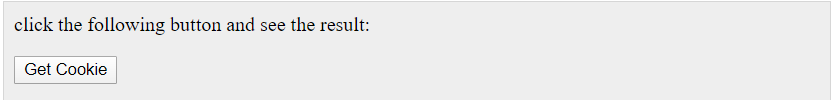
<input type = "button" value = "Get Cookie" onclick = "ReadCookie()"/>

</form>

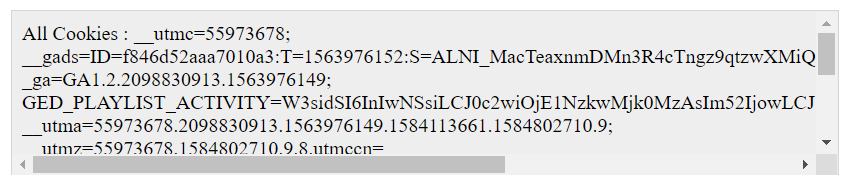
</body>

</html>

**Output:**



After **set cookie**



**Note** − Here **length** is a method of **Array** class which returns the length of an array. We will discuss Arrays in a separate chapter. By that time, please try to digest it.

**Note** − There may be some other cookies already set on your machine. The above code will display all the cookies set on your machine.

## **Setting Cookies Expiry Date**

You can extend the life of a cookie beyond the current browser session by setting an expiration date and saving the expiry date within the cookie. This can be done by setting the **‘expires’** attribute to a date and time.

### Program:

Try the following example. It illustrates how to extend the expiry date of a cookie by 1 Month.

<!DOCTYPE html>

<html>

<head>

<script>

<!--

function WriteCookie() {

var now = new Date();

now.setMonth( now.getMonth() + 1 );

cookievalue = escape(document.myform.customer.value) + ";"

document.cookie = "name=" + cookievalue;

document.cookie = "expires=" + now.toUTCString() + ";"

document.write ("Setting Cookies : " + "name=" + cookievalue );

}

//-->

</script>

</head>

<body>

<form name = "myform" action = "">

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

<input type = "button" value = "Set Cookie" onclick = "WriteCookie()"/>

</form>

</body>

</html>

### Output:

### 

### After set cookie (entered name swap)

### 

## **Deleting a Cookie**

Sometimes you will want to delete a cookie so that subsequent attempts to read the cookie return nothing. To do this, you just need to set the expiry date to a time in the past.

### Program:

Try the following example. It illustrates how to delete a cookie by setting its expiry date to one month behind the current date.

<!DOCTYPE html>

<html>

<head>

<script>

<!--

function WriteCookie() {

var now = new Date();

now.setMonth( now.getMonth() - 1 );

cookievalue = escape(document.myform.customer.value) + ";"

document.cookie = "name=" + cookievalue;

document.cookie = "expires=" + now.toUTCString() + ";"

document.write("Setting Cookies : " + "name=" + cookievalue );

}

//-->

</script>

</head>

<body>

<form name = "myform" action = "">

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

<input type = "button" value = "Set Cookie" onclick = "WriteCookie()"/>

</form>

</body>

</html>

### Output:

### 

### After set cookie

### 

* **JavaScript – Page Redirect:**

## **What is Page Redirection?**

You might have encountered a situation where you clicked a URL to reach a page X but internally you were directed to another page Y. It happens due to **page redirection**. This concept is different from [JavaScript Page Refresh](https://www.tutorialspoint.com/javascript/javascript_page_refresh.htm).

There could be various reasons why you would like to redirect a user from the original page. We are listing down a few of the reasons −

* You did not like the name of your domain and you are moving to a new one. In such a scenario, you may want to direct all your visitors to the new site. Here you can maintain your old domain but put a single page with a page redirection such that all your old domain visitors can come to your new domain.
* You have built-up various pages based on browser versions or their names or may be based on different countries, then instead of using your server-side page redirection, you can use client-side page redirection to land your users on the appropriate page.
* The Search Engines may have already indexed your pages. But while moving to another domain, you would not like to lose your visitors coming through search engines. So you can use client-side page redirection. But keep in mind this should not be done to fool the search engine, it could lead your site to get banned.

## **How Page Re-direction Works?**

The implementations of Page-Redirection are as follows.

### Program:

It is quite simple to do a page redirect using JavaScript at client side. To redirect your site visitors to a new page, you just need to add a line in your head section as follows.

<!DOCTYPE html>

<html>

<head>

<script>

<!--

function Redirect() {

window.location = "https://www.tutorialspoint.com";

}

//-->

</script>

</head>

<body>

<p>Click the following button, you will be redirected to home page.</p>

<form>

<input type = "button" value = "Redirect Me" onclick = "Redirect();" />

</form>

</body>

</html>

### Output:

### 

### Program:

You can show an appropriate message to your site visitors before redirecting them to a new page. This would need a bit time delay to load a new page. The following example shows how to implement the same. Here **setTimeout()** is a built-in JavaScript function which can be used to execute another function after a given time interval.

<!DOCTYPE html>

<html>

<head>

<script>

<!--

function Redirect() {

window.location = "https://www.tutorialspoint.com";

}

document.write("You will be redirected to main page in 10 sec.");

setTimeout('Redirect()', 10000);

//-->

</script>

</head>

<body>

</body>

</html>

### Output:

### 

You will be redirected to tutorialspoint.com main page in 10 seconds!

### Program:

The following example shows how to redirect your site visitors onto a different page based on their browsers.

<!DOCTYPE html>

<html>

<head>

<script>

<!--

var browsername = navigator.appName;

if( browsername == "Netscape" ) {

window.location = "http://www.location.com/ns.htm";

} else if ( browsername =="Microsoft Internet Explorer") {

window.location = "http://www.location.com/ie.htm";

} else {

window.location = "http://www.location.com/other.htm";

}

//-->

</script>

</head>

<body>

</body>

</html>

* **JavaScript – Dialog Boxes:**

JavaScript supports three important types of dialog boxes. These dialog boxes can be used to raise and alert, or to get confirmation on any input or to have a kind of input from the users

## **Alert Dialog Box**

An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.

Nonetheless, an alert box can still be used for friendlier messages. Alert box gives only one button “OK” to select and proceed.

**Program:**

<!DOCTYPE html>

<html>

<head>

<script>

<!--

function Warn() {

alert ("This is a warning message!");

document.write ("This is a warning message!");

}

//-->

</script>

</head>

<body>

<p>Click the following button to see the result: </p>

<form>

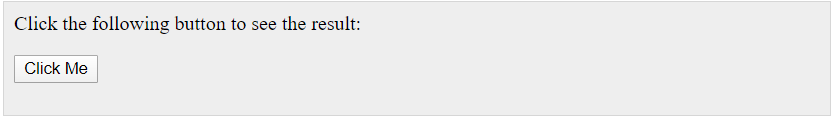
<input type = "button" value = "Click Me" onclick = "Warn();" />

</form>

</body>

</html>

Output:



Dialog box comes after clicking Click Me





## **Confirmation Dialog Box**

A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: OK and Cancel.

If the user clicks on the OK button, the window method confirm() will return true. If the user clicks on the Cancel button, then confirm() returns false. You can use a confirmation dialog box as follows.

### Program:

<!DOCTYPE html>

<html>

<head>

<script>

<!--

function getConfirmation() {

var retVal = confirm("Do you want to continue ?");

if( retVal == true ) {

document.write ("User wants to continue!");

return true;

} else {

document.write ("User does not want to continue!");

return false;

}

}

//-->

</script>

</head>

<body>

<p>Click the following button to see the result: </p>

<form>

<input type = "button" value = "Click Me" onclick = "getConfirmation();" />

</form>

</body>

</html>

### Output:

### 

Dialog box comes after clicking **Click Me**

### 

### 

## **Prompt Dialog Box**

The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK.

This dialog box is displayed using a method called prompt() which takes two parameters: (i) a label which you want to display in the text box and (ii) a default string to display in the text box.

This dialog box has two buttons: OK and Cancel. If the user clicks the OK button, the window method prompt() will return the entered value from the text box. If the user clicks the Cancel button, the window method prompt() returns null.

### Program:

The following example shows how to use a prompt dialog box −

<!DOCTYPE html>

<html>

<head>

<script>

<!--

function getValue() {

var retVal = prompt("Enter your name : ", "your name here");

document.write("You have entered : " + retVal);

}

//-->

</script>

</head>

<body>

<p>Click the following button to see the result: </p>

<form>

<input type = "button" value = "Click Me" onclick = "getValue();" />

</form>

</body>

</html>

### Output

### 

Dialog box comes after clicking **Click Me**

### 

### After entering name:

### 

* **JavaScript – Void Keyword:**

**void** is an important keyword in JavaScript which can be used as a unary operator that appears before its single operand, which may be of any type. This operator specifies an expression to be evaluated without returning a value.

## **Syntax**

The syntax of **void** can be either of the following two −

<head>

<script>

<!--

void func()

javascript:void func()

**or:**

void(func())

javascript:void(func())

//-->

</script>

</head>

### Program:

The most common use of this operator is in a client-side *javascript:* URL, where it allows you to evaluate an expression for its side-effects without the browser displaying the value of the evaluated expression.

Here the expression **alert ('Warning!!!')** is evaluated but it is not loaded back into the current document −

<!DOCTYPE html>

<html>

<head>

<script>

<!--

//-->

</script>

</head>

<body>

<p>Click the following, This won't react at all...</p>

<a href = "javascript:void(alert('Warning!!!'))">Click me!</a>

</body>

</html>

### Output:

### 

Dialog box comes after clicking **Click Me**

### 

### Program:

Take a look at the following example. The following link does nothing because the expression "0" has no effect in JavaScript. Here the expression "0" is evaluated, but it is not loaded back into the current document.

<!DOCTYPE html>

<html>

<head>

<script>

<!--

//-->

</script>

</head>

<body>

<p>Click the following, This won't react at all...</p>

<a href = "javascript:void(0)">Click me!</a>

</body>

</html>

### Output:

### 

### Program:

Another use of **void** is to purposely generate the **undefined** value as follows.

<!DOCTYPE html>

<html>

<head>

<script>

<!--

function getValue() {

var a,b,c;

a = void ( b = 5, c = 7 );

document.write('a = ' + a + ' b = ' + b +' c = ' + c );

}

//-->

</script>

</head>

<body>

<p>Click the following to see the result:</p>

<form>

<input type = "button" value = "Click Me" onclick = "getValue();" />

</form>

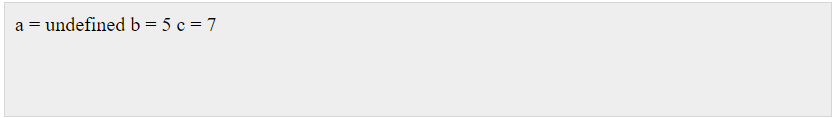
</body>

</html>

### Output



After clicking **Click Me**



* **JavaScript – Page Printing:**

Many times you would like to place a button on your webpage to print the content of that web page via an actual printer. JavaScript helps you to implement this functionality using the **print** function of **window** object.

The JavaScript print function **window.print()** prints the current web page when executed. You can call this function directly using the **onclick** event as shown in the following example.

### Program:

Try the following example.

<!DOCTYPE html>

<html>

<head>

<script type = "text/javascript">

<!--

//-->

</script>

</head>

<body>

<form>

<input type = "button" value = "Print" onclick = "window.print()" />

</form>

</body>

<html>

### Output:

### 

### After clicking on print

### 

Although it serves the purpose of getting a printout, it is not a recommended way. A printer friendly page is really just a page with text, no images, graphics, or advertising.

You can make a page printer friendly in the following ways −

* Make a copy of the page and leave out unwanted text and graphics, then link to that printer friendly page from the original.
* If you do not want to keep an extra copy of a page, then you can mark your printable text using proper comments like <!-- PRINT STARTS HERE -->..... <!-- PRINT ENDS HERE --> and then you can use PERL or any other script in the background to purge printable text and display for final printing.
* **How to Print a Page?**

If you don’t find the above facilities on a web page, then you can use the browser's standard toolbar to get print the web page. Follow the link as follows.

File → Print → Click OK button.

**JAVASCRIPT OBJECTS**

* **JavaScript – Objects:**

In JavaScript, almost "everything" is an object.

* + - Booleans can be objects (if defined with the new keyword)
    - Numbers can be objects (if defined with the new keyword)
    - Strings can be objects (if defined with the new keyword)
    - Dates are always objects
    - Maths are always objects
    - Regular expressions are always objects
    - Arrays are always objects
    - Functions are always objects
    - Objects are always objects

All JavaScript values, except primitives, are objects.

## **JavaScript Primitives**

A **primitive value** is a value that has no properties or methods. A **primitive data type** is data that has a primitive value.

JavaScript defines 5 types of primitive data types:

* + - String
    - number
    - boolean
    - null
    - undefined

Primitive values are immutable (they are hardcoded and therefore cannot be changed).

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++", "Perl", "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.

### Program:

This program demonstrates how to create an Object.

<!DOCTYPE html>

<html>

<head>

<title>User-defined objects</title>

<script type = "text/javascript">

var book = new Object(); // Create the object

book.subject = "Perl"; // Assign properties to the object

book.author = "Mohtashim";

</script>

</head>

<body>

<script>

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

document.write("Book author is : " + book.author + "<br>");

</script>

</body>

</html>

### Output:

Book name is : Perl

Book author is : Mohtashim

### Program:

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.

<!DOCTYPE html>

<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>

var myBook = new book("Perl", "Mohtashim");

document.write("Book title is : " + myBook.title + "<br>");

document.write("Book author is : " + myBook.author + "<br>");

</script>

</body>

</html>

### Output:

Book title is : Perl

Book author is : Mohtashim

## **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.

### Program:

This program demonstrates how to add a function along with an object.

<!DOCTYPE html>

<html>

<head>

<title>User-defined objects</title>

<script>

// 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("Perl", "Mohtashim");

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 : Perl

Book author is : Mohtashim

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

}

### Program:

Try the following example.

<!DOCTYPE html>

<html>

<head>

<title>User-defined objects</title>

<script>

// 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>

var myBook = new book("Perl", "Mohtashim");

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 : Perl

Book author is : Mohtashim

Book price is : 100

## **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](https://www.tutorialspoint.com/javascript/javascript_number_object.htm)
* [JavaScript Boolean Object](https://www.tutorialspoint.com/javascript/javascript_boolean_object.htm)
* [JavaScript String Object](https://www.tutorialspoint.com/javascript/javascript_strings_object.htm)
* [JavaScript Array Object](https://www.tutorialspoint.com/javascript/javascript_arrays_object.htm)
* [JavaScript Date Object](https://www.tutorialspoint.com/javascript/javascript_date_object.htm)
* [JavaScript Math Object](https://www.tutorialspoint.com/javascript/javascript_math_object.htm)
* [JavaScript RegExp Object](https://www.tutorialspoint.com/javascript/javascript_regexp_object.htm)

Avoid **String, Number**, and **Boolean**objects. They complicate your code and slow down execution speed.

* **JavaScript – Number:**

JavaScript has only one type of number. Numbers can be written with or without decimals.

## **JavaScript Numbers are Always 64-bit Floating Point**

Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.

JavaScript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard.  
  
 This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63:

|  |  |  |
| --- | --- | --- |
| **Value (aka Fraction/Mantissa)** | **Exponent** | **Sign** |
| 52 bits (0 - 51) | 11 bits (52 - 62) | 1 bit (63) |

## **Precision**

Integers (numbers without a period or exponent notation) are accurate up to 15 digits.

## **Numeric Strings**

JavaScript strings can have numeric content:

var x = 100;         // x is a number  
  
 var y = "100";       // y is a string

## **NaN - Not a Number**

NaN is a JavaScript reserved word indicating that a number is not a legal number. Trying to do arithmetic with a non-numeric string will result in NaN (Not a Number):

**Example**

var x = 100 / "Apple";  // x will be NaN (Not a Number)

NaN is a number: typeof NaN returns number:

**Example**

typeof NaN;            // returns "number"

## **Infinity**

Infinity (or –Infiniti) is the value JavaScript will return if you calculate a number outside the largest possible number.

**Example**

var myNumber = 2;  
while (myNumber != Infinity) {   // Execute until Infinity  
  myNumber = myNumber \* myNumber;  
}

Infinity is a number: typeof Infinity returns number.

**Example**

typeof Infinity;     // returns "number"

## **Hexadecimal**

JavaScript interprets numeric constants as hexadecimal if they are preceded by 0x.

**Example**

var x = 0xFF;        // x will be 255

Never write a number with a leading zero (like 07). Some JavaScript versions interpret numbers as octal if they are written with a leading zero.By default, JavaScript displays numbers as **base 10** decimals. But you can use the toString() method to output numbers from **base 2** to **base 36**. Hexadecimal is **base 16**. Decimal is **base 10**. Octal is **base 8**. Binary is **base 2**.

## **Numbers Can be Objects**

The **Number** object represents numerical date, either integers or floating-point numbers. In general, you do not need to worry about **Number** objects because the browser automatically converts number literals to instances of the number class.

### Syntax

The syntax for creating a **number** object is as follows −

var val = new Number(number);

In the place of number, if you provide any non-number argument, then the argument cannot be converted into a number, it returns **NaN** (Not-a-Number).

Normally JavaScript numbers are primitive values created from literals:

var x = 123;

But numbers can also be defined as objects with the keyword new:

var y = new Number(123);

**Example**

var x = 123;  
var y = new Number(123);  
  
// typeof x returns number  
// typeof y returns object

**Do not** create Number objects. It **slows down execution speed.**

The **new keyword complicates the code**. This can produce some unexpected results.

Note the difference between **(x==y)** and **(x===y).**

Comparing two JavaScript objects will always return **false.**

## **Number Properties**

Here is a list of each property and their description.

|  |  |
| --- | --- |
| **Sr.No.** | **Property & Description** |
| **1** | [MAX\_VALUE](https://www.tutorialspoint.com/javascript/number_max_value.htm)  The largest possible value a number in JavaScript can have 1.7976931348623157E+308 |
| **2** | [MIN\_VALUE](https://www.tutorialspoint.com/javascript/number_min_value.htm)  The smallest possible value a number in JavaScript can have 5E-324 |
| **3** | [NaN](https://www.tutorialspoint.com/javascript/number_nan.htm)  Equal to a value that is not a number. |
| **4** | [NEGATIVE\_INFINITY](https://www.tutorialspoint.com/javascript/number_negative_infinity.htm)  A value that is less than MIN\_VALUE. |
| **5** | [POSITIVE\_INFINITY](https://www.tutorialspoint.com/javascript/number_positive_infinity.htm)  A value that is greater than MAX\_VALUE |
| **6** | [prototype](https://www.tutorialspoint.com/javascript/object_prototype.htm)  A static property of the Number object. Use the prototype property to assign new properties and methods to the Number object in the current document |
| **7** | [constructor](https://www.tutorialspoint.com/javascript/number_constructor.htm)  Returns the function that created this object's instance. By default this is the Number object. |

## **Number Methods**

The Number object contains only the default methods that are a part of every object's definition.

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| **1** | [toExponential()](https://www.tutorialspoint.com/javascript/number_toexponential.htm)  Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation. |
| **2** | [toFixed()](https://www.tutorialspoint.com/javascript/number_tofixed.htm)  Formats a number with a specific number of digits to the right of the decimal. |
| **3** | [toLocaleString()](https://www.tutorialspoint.com/javascript/number_tolocalestring.htm)  Returns a string value version of the current number in a format that may vary according to a browser's local settings. |
| **4** | [toPrecision()](https://www.tutorialspoint.com/javascript/number_toprecision.htm)  Defines how many total digits (including digits to the left and right of the decimal) to display of a number. |
| **5** | [toString()](https://www.tutorialspoint.com/javascript/number_tostring.htm)  Returns the string representation of the number's value |
| **6** | [valueOf()](https://www.tutorialspoint.com/javascript/number_valueof.htm)  Returns the number's value. |

* **JavaScript – Boolean:**

A JavaScript Boolean represents one of two values: **true** or **false**.

* **Boolean Values**

Very often, in programming, you will need a data type that can only have one of two values, like

* + YES / NO
  + ON / OFF
  + TRUE / FALSE

For this, JavaScript has a **Boolean** data type. It can only take the values **true** or **false**.

The Boolean value of an expression is the basis for all JavaScript comparisons and conditions.

## **The Boolean() Function**

You can use the **Boolean()** function to find out if an expression (or a variable) is true:

**Example**

Boolean(10 > 9)        // returns true

## **Everything Without a "Value" is False**

* The Boolean value of 0 (zero) is false:

var x = 0;  
Boolean(x);       // returns false

* The Boolean value of **-0** (minus zero) is **false**:

var x = -0;  
Boolean(x);       // returns false

* The Boolean value of **""**(empty string) is **false**:

var x = "";  
Boolean(x);       // returns false

* The Boolean value of **undefined** is **false**:

var x;  
Boolean(x);       // returns false

* The Boolean value of **null** is **false**:

var x = null;  
Boolean(x);       // returns false

* The Boolean value of **false** is (you guessed it) **false**:

var x = false;  
Boolean(x);       // returns false

* The Boolean value of **NaN** is **false**:

var x = 10 / "H";  
Boolean(x);       // returns false

## **Booleans Can be Objects**

Normally JavaScript booleans are primitive values created from literals.

var x = false;

### Syntax

Use the following syntax to create a **boolean** object.

var val = new Boolean(value);

But booleans can also be defined as objects with the keyword **new**:

var y = new Boolean(false);

**Example**

var x = false;  
var y = new Boolean(false);  
  
// typeof x returns boolean  
// typeof y returns object

## **Boolean Properties**

Here is a list of the properties of Boolean object −

|  |  |
| --- | --- |
| **Sr.No.** | **Property & Description** |
| 1 | [constructor](https://www.tutorialspoint.com/javascript/boolean_constructor.htm)  Returns a reference to the Boolean function that created the object. |
| 2 | [prototype](https://www.tutorialspoint.com/javascript/object_prototype.htm)  The prototype property allows you to add properties and methods to an object. |

## **Boolean Methods**

Here is a list of the methods of Boolean object and their description.

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | [toSource()](https://www.tutorialspoint.com/javascript/boolean_tosource.htm)  Returns a string containing the source of the Boolean object; you can use this string to create an equivalent object. |
| 2 | [toString()](https://www.tutorialspoint.com/javascript/boolean_tostring.htm)  Returns a string of either "true" or "false" depending upon the value of the object. |
| 3 | [valueOf()](https://www.tutorialspoint.com/javascript/boolean_valueof.htm)  Returns the primitive value of the Boolean object |

**Do not** create **Boolean objects**. It slows down execution speed.

The new keyword **complicates the code**. This can produce some unexpected results:

* **JavaScript – Strings:**

JavaScript strings are used for storing and manipulating text.The **String** object lets you work with a series of characters; it wraps Javascript's string primitive data type with a number of helper methods.

As JavaScript automatically converts between string primitives and String objects, you can call any of the helper methods of the String object on a string primitive.

## **String Length**

To find the length of a string, use the built-in length property:

**Example**

var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
var sln = txt.length;

## **Escape Character**

Because strings must be written within quotes, JavaScript will misunderstand this string:

var x = "We are the so-called "Vikings" from the north.";

The string will be chopped to "We are the so-called ".The solution to avoid this problem, is to use the **backslash escape character**.The backslash (\) escape character turns special characters into string characters:

|  |  |  |
| --- | --- | --- |
| **Code** | **Result** | **Description** |
| \' | ' | Single quote |
| \" | " | Double quote |
| \\ | \ | Backslash ' |

Six other escape sequences are valid in JavaScript:

Code Result

\b Backspace

\f Form Feed

\n New Line

\r Carriage Return

\t Horizontal Tabulator

\v Vertical Tabulator

The 6 escape characters above were originally designed to control typewriters, teletypes, and fax machines. They do not make any sense in HTML.

* **Strings Can be Objects**

Normally, JavaScript strings are primitive values, created from literals:

var firstName = "John";

But strings can also be defined as objects with the keyword new:

var firstName = new String("John");

### Syntax

### Use the following syntax to create a String object −

var val = new String(string);

The **String** parameter is a series of characters that has been properly encoded.

* **String Properties**

Here is a list of the properties of String object and their description.

|  |  |
| --- | --- |
| **Sr.No.** | **Property & Description** |
| 1 | constructor  Returns a reference to the String function that created the object. |
| 2 | length  Returns the length of the string. |
| 3 | prototype  The prototype property allows you to add properties and methods to an object. |

## **String Methods**

Here is a list of the methods available in String object along with their description.

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | charAt()  Returns the character at the specified index. |
| 2 | charCodeAt()  Returns a number indicating the Unicode value of the character at the given index. |
| 3 | concat()  Combines the text of two strings and returns a new string. |
| 4 | indexOf()  Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found. |
| 5 | lastIndexOf()  Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found. |
| 6 | localeCompare()  Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order. |
| 7 | match()  Used to match a regular expression against a string. |
| 8 | replace()  Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring. |
| 9 | search()  Executes the search for a match between a regular expression and a specified string. |
| 10 | slice()  Extracts a section of a string and returns a new string. |
| 11 | split()  Splits a String object into an array of strings by separating the string into substrings. |
| 12 | substr()  Returns the characters in a string beginning at the specified location through the specified number of characters. |
| 13 | substring()  Returns the characters in a string between two indexes into the string. |
| 14 | toLocaleLowerCase()  The characters within a string are converted to lower case while respecting the current locale. |
| 15 | toLocaleUpperCase()  The characters within a string are converted to upper case while respecting the current locale. |
| 16 | toLowerCase()  Returns the calling string value converted to lower case. |
| 17 | toString()  Returns a string representing the specified object. |
| 18 | toUpperCase()  Returns the calling string value converted to uppercase. |
| 19 | valueOf()  Returns the primitive value of the specified object. |

## **String HTML Wrappers**

Here is a list of the methods that return a copy of the string wrapped inside an appropriate HTML tag.

|  |  |
| --- | --- |
| **Sr. No.** | **Method & Description** |
| 1 | anchor()  Creates an HTML anchor that is used as a hypertext target. |
| 2 | big()  Creates a string to be displayed in a big font as if it were in a <big> tag. |
| 3 | blink()  Creates a string to blink as if it were in a <blink> tag. |
| 4 | bold()  Creates a string to be displayed as bold as if it were in a <b> tag. |
| 5 | fixed()  Causes a string to be displayed in fixed-pitch font as if it were in a <tt> tag |
| 6 | fontcolor()  Causes a string to be displayed in the specified color as if it were in a <font color="color"> tag. |
| 7 | fontsize()  Causes a string to be displayed in the specified font size as if it were in a <font size="size"> tag. |
| 8 | italics()  Causes a string to be italic, as if it were in an <i> tag. |
| 9 | link()  Creates an HTML hypertext link that requests another URL. |
| 10 | small()  Causes a string to be displayed in a small font, as if it were in a <small> tag. |
| 11 | strike()  Causes a string to be displayed as struck-out text, as if it were in a <strike> tag. |
| 12 | sub()  Causes a string to be displayed as a subscript, as if it were in a <sub> tag |
| 13 | sup()  Causes a string to be displayed as a superscript, as if it were in a <sup> tag |

* **JavaScript – Arrays:**

JavaScript arrays are used to store multiple values in a single variable.

## **What is an Array?**

An array is a special variable, which can hold more than one value at a time. 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.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

var car1 = "Saab";  
 var car2 = "Volvo";  
 var car3 = "BMW";

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

## **Creating an Array**

Using an array literal is the easiest way to create a JavaScript Array.

**Syntax:**

var array\_name = [item1, item2, ...];

**Example**

var cars = ["Saab", "Volvo", "BMW"];

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

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

## **Access the Elements of an Array**

You access an array element by referring to the **index number**.

**Note:** Array indexes start with 0.

[0] is the first element. [1] is the second element.

## **Arrays are Objects**

Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.

But, JavaScript arrays are best described as arrays.

## **Array Elements Can Be Objects**

JavaScript variables can be objects. Arrays are special kinds of objects. Because of this, you can have variables of different types in the same Array. You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

myArray[0] = Date.now;  
 myArray[1] = myFunction;  
 myArray[2] = myCars;

## **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. |

## **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. |

* **JavaScript – Date:**

The Date object is a datatype built into the JavaScript language. Date objects are created with the **new Date( )** as shown below.

Once a Date object is created, a number of methods allow you to operate on it. Most methods simply allow you to get and set the year, month, day, hour, minute, second, and millisecond fields of the object, using either local time or UTC (universal, or GMT) time.

The ECMAScript standard requires the Date object to be able to represent any date and time, to millisecond precision, within 100 million days before or after 1/1/1970. This is a range of plus or minus 273,785 years, so JavaScript can represent date and time till the year 275755.

### Syntax

You can use any of the following syntaxes to create a Date object using Date() constructor.

new Date( )

new Date(milliseconds)

new Date(datestring)

new Date(year,month,date[,hour,minute,second,millisecond ])

**Note** − Parameters in the brackets are always optional.

Here is a description of the parameters −

* **No Argument** − With no arguments, the Date() constructor creates a Date object set to the current date and time.
* **milliseconds** − When one numeric argument is passed, it is taken as the internal numeric representation of the date in milliseconds, as returned by the getTime() method. For example, passing the argument 5000 creates a date that represents five seconds past midnight on 1/1/70.
* **datestring** − When one string argument is passed, it is a string representation of a date, in the format accepted by the **Date.parse()** method.
* **7 agruments** − To use the last form of the constructor shown above. Here is a description of each argument −
* **year** − Integer value representing the year. For compatibility (in order to avoid the Y2K problem), you should always specify the year in full; use 1998, rather than 98.
* **month** − Integer value representing the month, beginning with 0 for January to 11 for December.
* **date** − Integer value representing the day of the month.
* **hour** − Integer value representing the hour of the day (24-hour scale).
* **minute** − Integer value representing the minute segment of a time reading.
* **second** − Integer value representing the second segment of a time reading.
* **millisecond** − Integer value representing the millisecond segment of a time reading.

## **Date Properties**

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

|  |  |
| --- | --- |
| **Sr.No.** | **Property & Description** |
| 1 | constructor  Specifies the function that creates an object's prototype. |
| 2 | prototype  The prototype property allows you to add properties and methods to an object |

## **Date Methods**

Here is a list of the methods used with **Date** and their description.

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | Date()  Returns today's date and time |
| 2 | getDate()  Returns the day of the month for the specified date according to local time. |
| 3 | getDay()  Returns the day of the week for the specified date according to local time. |
| 4 | getFullYear()  Returns the year of the specified date according to local time. |
| 5 | getHours()  Returns the hour in the specified date according to local time. |
| 6 | getMilliseconds()  Returns the milliseconds in the specified date according to local time. |
| 7 | getMinutes()  Returns the minutes in the specified date according to local time. |
| 8 | getMonth()  Returns the month in the specified date according to local time. |
| 9 | getSeconds()  Returns the seconds in the specified date according to local time. |
| 10 | getTime()  Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC. |
| 11 | getTimezoneOffset()  Returns the time-zone offset in minutes for the current locale. |
| 12 | getUTCDate()  Returns the day (date) of the month in the specified date according to universal time. |
| 13 | getUTCDay()  Returns the day of the week in the specified date according to universal time. |
| 14 | getUTCFullYear()  Returns the year in the specified date according to universal time. |
| 15 | getUTCHours()  Returns the hours in the specified date according to universal time. |
| 16 | getUTCMilliseconds()  Returns the milliseconds in the specified date according to universal time. |
| 17 | getUTCMinutes()  Returns the minutes in the specified date according to universal time. |
| 18 | getUTCMonth()  Returns the month in the specified date according to universal time. |
| 19 | getUTCSeconds()  Returns the seconds in the specified date according to universal time. |
| 20 | getYear()  Deprecated - Returns the year in the specified date according to local time. Use getFullYear instead. |
| 21 | setDate()  Sets the day of the month for a specified date according to local time. |
| 22 | setFullYear()  Sets the full year for a specified date according to local time. |
| 23 | setHours()  Sets the hours for a specified date according to local time. |
| 24 | setMilliseconds()  Sets the milliseconds for a specified date according to local time. |
| 25 | setMinutes()  Sets the minutes for a specified date according to local time. |
| 26 | setMonth()  Sets the month for a specified date according to local time. |
| 27 | setSeconds()  Sets the seconds for a specified date according to local time. |
| 28 | setTime()  Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC. |
| 29 | setUTCDate()  Sets the day of the month for a specified date according to universal time. |
| 30 | setUTCFullYear()  Sets the full year for a specified date according to universal time. |
| 31 | setUTCHours()  Sets the hour for a specified date according to universal time. |
| 32 | setUTCMilliseconds()  Sets the milliseconds for a specified date according to universal time. |
| 33 | setUTCMinutes()  Sets the minutes for a specified date according to universal time. |
| 34 | setUTCMonth()  Sets the month for a specified date according to universal time. |
| 35 | setUTCSeconds()  Sets the seconds for a specified date according to universal time. |
| 36 | setYear()  Deprecated - Sets the year for a specified date according to local time. Use setFullYear instead. |
| 37 | toDateString()  Returns the "date" portion of the Date as a human-readable string. |
| 38 | toGMTString()  Deprecated - Converts a date to a string, using the Internet GMT conventions. Use toUTCString instead. |
| 39 | toLocaleDateString()  Returns the "date" portion of the Date as a string, using the current locale's conventions. |
| 40 | toLocaleFormat()  Converts a date to a string, using a format string. |
| 41 | toLocaleString()  Converts a date to a string, using the current locale's conventions. |
| 42 | toLocaleTimeString()  Returns the "time" portion of the Date as a string, using the current locale's conventions. |
| 43 | toSource()  Returns a string representing the source for an equivalent Date object; you can use this value to create a new object. |
| 44 | toString()  Returns a string representing the specified Date object. |
| 45 | toTimeString()  Returns the "time" portion of the Date as a human-readable string. |
| 46 | toUTCString()  Converts a date to a string, using the universal time convention. |
| 47 | valueOf()  Returns the primitive value of a Date object. |

* **JavaScript – Math:**

The JavaScript Math object allows you to perform mathematical tasks on numbers. The **math** object provides you properties and methods for mathematical constants and functions. Unlike other global objects, **Math** is not a constructor. All the properties and methods of **Math** are static and can be called by using Math as an object without creating it.

Thus, you refer to the constant **pi** as **Math.PI** and you call the *sine* function as **Math.sin(x)**, where x is the method's argument.

### Syntax

The syntax to call the properties and methods of Math are as follows

var pi\_val = Math.PI;

var sine\_val = Math.sin(30);

## **Math.round()**

Math.round(x) returns the value of x rounded to its nearest integer:

**Example**

Math.round(4.7);    // returns 5  
Math.round(4.4);    // returns 4

## **Math.pow()**

Math.pow(x, y) returns the value of x to the power of y:

**Example**

Math.pow(8, 2);      // returns 64

## **Math.sqrt()**

Math.sqrt(x) returns the square root of x:

**Example**

Math.sqrt(64);      // returns 8

## **Math.abs()**

Math.abs(x) returns the absolute (positive) value of x:

**Example**

Math.abs(-4.7);     // returns 4.7

## **Math.ceil()**

Math.ceil(x) returns the value of x rounded **up** to its nearest integer:

**Example**

Math.ceil(4.4);     // returns 5

## **Math.floor()**

Math.floor(x) returns the value of x rounded **down** to its nearest integer:

**Example**

Math.floor(4.7);    // returns 4

## **Math.sin()**

Math.sin(x) returns the sine (a value between -1 and 1) of the angle x (given in radians).

If you want to use degrees instead of radians, you have to convert degrees to radians:

Angle in radians = Angle in degrees x PI / 180.

**Example**

Math.sin(90 \* Math.PI / 180);     // returns 1 (the sine of 90 degrees)

## **Math.cos()**

Math.cos(x) returns the cosine (a value between -1 and 1) of the angle x (given in radians).

If you want to use degrees instead of radians, you have to convert degrees to radians:

Angle in radians = Angle in degrees x PI / 180.

**Example**

Math.cos(0 \* Math.PI / 180);     // returns 1 (the cos of 0 degrees)

## **Math.min() and Math.max()**

Math.min() and Math.max() can be used to find the lowest or highest value in a list of arguments:

**Example**

Math.min(0, 150, 30, 20, -8, -200);  // returns -200

### Example

Math.max(0, 150, 30, 20, -8, -200);  // returns 150

## **Math.random()**

Math.random() returns a random number between 0 (inclusive), and 1 (exclusive):

**Example**

Math.random();     // returns a random number

## **Math Properties (Constants)**

JavaScript provides 8 mathematical constants that can be accessed with the Math object:

**Example**

Math.E        // returns Euler's number  
Math.PI       // returns PI  
Math.SQRT2    // returns the square root of 2  
Math.SQRT1\_2  // returns the square root of 1/2  
Math.LN2      // returns the natural logarithm of 2  
Math.LN10     // returns the natural logarithm of 10  
Math.LOG2E    // returns base 2 logarithm of E  
Math.LOG10E   // returns base 10 logarithm of E

## **Math Constructor**

Unlike other global objects, the Math object has no constructor. Methods and properties are static.

All methods and properties (constants) can be used without creating a Math object first.

## **Math Object Methods**

|  |  |
| --- | --- |
| Method | Description |
| abs(x) | Returns the absolute value of x |
| acos(x) | Returns the arccosine of x, in radians |
| acosh(x) | Returns the hyperbolic arccosine of x |
| asin(x) | Returns the arcsine of x, in radians |
| asinh(x) | Returns the hyperbolic arcsine of x |
| atan(x) | Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians |
| atan2(y, x) | Returns the arctangent of the quotient of its arguments |
| atanh(x) | Returns the hyperbolic arctangent of x |
| cbrt(x) | Returns the cubic root of x |
| ceil(x) | Returns x, rounded upwards to the nearest integer |
| cos(x) | Returns the cosine of x (x is in radians) |
| cosh(x) | Returns the hyperbolic cosine of x |
| exp(x) | Returns the value of Ex |
| floor(x) | Returns x, rounded downwards to the nearest integer |
| log(x) | Returns the natural logarithm (base E) of x |
| max(x, y, z, ..., n) | Returns the number with the highest value |
| min(x, y, z, ..., n) | Returns the number with the lowest value |
| pow(x, y) | Returns the value of x to the power of y |
| random() | Returns a random number between 0 and 1 |
| round(x) | Rounds x to the nearest integer |
| sin(x) | Returns the sine of x (x is in radians) |
| sinh(x) | Returns the hyperbolic sine of x |
| sqrt(x) | Returns the square root of x |
| tan(x) | Returns the tangent of an angle |
| tanh(x) | Returns the hyperbolic tangent of a number |
| trunc(x) | Returns the integer part of a number (x) |

* **JavaScript – RegExp:**

A regular expression is an object that describes a pattern of characters.

The JavaScript **RegExp** class represents regular expressions, and both String a nd **RegExp** define methods that use regular expressions to perform powerful pattern- matching and search-and-replace functions on text.

### Syntax

A regular expression could be defined with the **RegExp ()** constructor, as follows −

var pattern = new RegExp(pattern, attributes);

or simply

var pattern = /pattern/attributes;

Here is the description of the parameters −

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

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

|  |  |
| --- | --- |
| **Sr.No.** | **Expression & Description** |
| 1 | **[...]**  Any one character between the brackets. |
| 2 | **[^...]**  Any one character not between the brackets. |
| 3 | **[0-9]**  It matches any decimal digit from 0 through 9. |
| 4 | **[a-z]**  It matches any character from lowercase **a**through lowercase **z**. |
| 5 | **[A-Z]**  It matches any character from uppercase **A** through uppercase **Z**. |
| 6 | **[a-Z]**  It matches any character from lowercase **a** through uppercase **Z**. |

The ranges shown above are general; you could also use the range [0-3] to match any decimal digit ranging from 0 through 3, or the range [b-v] to match any lowercase character ranging from **b** through **v**.

## **Quantifiers**

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

|  |  |
| --- | --- |
| **Sr.No.** | **Expression & Description** |
| 1 | **p+**  It matches any string containing one or more p's. |
| 2 | **p\***  It matches any string containing zero or more p's. |
| 3 | **p?**  It matches any string containing at most one p. |
| 4 | **p{N}**  It matches any string containing a sequence of **N** p's |
| 5 | **p{2,3}**  It matches any string containing a sequence of two or three p's. |
| 6 | **p{2, }**  It matches any string containing a sequence of at least two p's. |
| 7 | **p$**  It matches any string with p at the end of it. |
| 8 | **^p**  It matches any string with p at the beginning of it. |

### Examples

Following examples explain more about matching characters.

|  |  |
| --- | --- |
| **Sr.No.** | **Expression & Description** |
| 1 | **[^a-zA-Z]**  It matches any string not containing any of the characters ranging from **a** through **z** and **A** through Z. |
| 2 | **p.p**  It matches any string containing **p,** followed by any character, in turn followed by another **p**. |
| 3 | **^.{2}$**  It matches any string containing exactly two characters. |
| 4 | **<b>(.\*)</b>**  It matches any string enclosed within <b> and </b>. |
| 5 | **p(hp)\***  It matches any string containing a **p** followed by zero or more instances of the sequence **hp**. |

## **Literal characters**

|  |  |
| --- | --- |
| **Sr.No.** | **Character & Description** |
| 1 | **Alphanumeric**  Itself |
| 2 | **\0**  The NUL character (\u0000) |
| 3 | **\t**  Tab (\u0009 |
| 4 | **\n**  Newline (\u000A) |
| 5 | **\v**  Vertical tab (\u000B) |
| 6 | **\f**  Form feed (\u000C) |
| 7 | **\r**  Carriage return (\u000D) |
| 8 | **\xnn**  The Latin character specified by the hexadecimal number nn; for example, \x0A is the same as \n |
| 9 | **\uxxxx**  The Unicode character specified by the hexadecimal number xxxx; for example, \u0009 is the same as \t |
| 10 | **\cX**  The control character ^X; for example, \cJ is equivalent to the newline character \n |

## **Metacharacters**

A metacharacter is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

For instance, you can search for a large sum of money using the '\d' metacharacter: **/([\d]+)000/**, Here **\d** will search for any string of numerical character.

The following table lists a set of metacharacters which can be used in PERL Style Regular Expressions.

|  |  |
| --- | --- |
| **Sr.No.** | **Character & Description** |
| 1 | **.**  a single character |
| 2 | **\s**  a whitespace character (space, tab, newline) |
| 3 | **\S**  non-whitespace character |
| 4 | **\d**  a digit (0-9) |
| 5 | **\D**  a non-digit |
| 6 | **\w**  a word character (a-z, A-Z, 0-9, \_) |
| 7 | **\W**  a non-word character |
| 8 | **[\b]**  a literal backspace (special case). |
| 9 | **[aeiou]**  matches a single character in the given set |
| 10 | **[^aeiou]**  matches a single character outside the given set |
| 11 | **(foo|bar|baz)**  matches any of the alternatives specified |

## **Modifiers**

Several modifiers are available that can simplify the way you work with **regexps,** like case sensitivity, searching in multiple lines, etc.

|  |  |
| --- | --- |
| **Sr.No.** | **Modifier & Description** |
| 1 | **i**  Perform case-insensitive matching. |
| 2 | **m**  Specifies that if the string has newline or carriage return characters, the ^ and $ operators will now match against a newline boundary, instead of a string boundary |
| 3 | **g**  Performs a global matchthat is, find all matches rather than stopping after the first match. |

## **RegExp Properties**

Here is a list of the properties associated with RegExp and their description.

|  |  |
| --- | --- |
| **Sr.No.** | **Property & Description** |
| 1 | [constructor](https://www.tutorialspoint.com/javascript/regexp_constructor.htm)  Specifies the function that creates an object's prototype. |
| 2 | [global](https://www.tutorialspoint.com/javascript/regexp_global.htm)  Specifies if the "g" modifier is set. |
| 3 | [ignoreCase](https://www.tutorialspoint.com/javascript/regexp_ignorecase.htm)  Specifies if the "i" modifier is set. |
| 4 | [lastIndex](https://www.tutorialspoint.com/javascript/regexp_lastindex.htm)  The index at which to start the next match. |
| 5 | [multiline](https://www.tutorialspoint.com/javascript/regexp_multiline.htm)  Specifies if the "m" modifier is set. |
| 6 | [source](https://www.tutorialspoint.com/javascript/regexp_source.htm)  The text of the pattern. |

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

* **RegExp Methods**

Here is a list of the methods associated with RegExp along with their description.

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | [exec()](https://www.tutorialspoint.com/javascript/regexp_exec.htm)  Executes a search for a match in its string parameter. |
| 2 | [test()](https://www.tutorialspoint.com/javascript/regexp_test.htm)  Tests for a match in its string parameter. |
| 3 | [toSource()](https://www.tutorialspoint.com/javascript/regexp_tosource.htm)  Returns an object literal representing the specified object; you can use this value to create a new object. |
| 4 | [toString()](https://www.tutorialspoint.com/javascript/regexp_tostring.htm)  Returns a string representing the specified object. |

* **JavaScript – HTML DOM:**

Every web page resides inside a browser window which can be considered as an object.

A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

* **Window object** − Top of the hierarchy. It is the outmost element of the object hierarchy.
* **Document object** − Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
* **Form object** − Everything enclosed in the <form>...</form> tags sets the form object.
* **Form control elements** − The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

Here is a simple hierarchy of a few important objects −



There are several DOMs in existence. The following sections explain each of these DOMs in detail and describe how you can use them to access and modify document content.

* [The Legacy DOM](https://www.tutorialspoint.com/javascript/javascript_legacy_dom.htm) − This is the model which was introduced in early versions of JavaScript language. It is well supported by all browsers, but allows access only to certain key portions of documents, such as forms, form elements, and images.
* [The W3C DOM](https://www.tutorialspoint.com/javascript/javascript_w3c_dom.htm) − This document object model allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C). This model is supported by almost all the modern browsers.
* [The IE4 DOM](https://www.tutorialspoint.com/javascript/javascript_ie4_dom.htm) − This document object model was introduced in Version 4 of Microsoft's Internet Explorer browser. IE 5 and later versions include support for most basic W3C DOM features.

## **DOM compatibility**

If you want to write a script with the flexibility to use either W3C DOM or IE 4 DOM depending on their availability, then you can use a capability-testing approach that first checks for the existence of a method or property to determine whether the browser has the capability you desire. For example −

if (document.getElementById) {

// If the W3C method exists, use it

} else if (document.all) {

// If the all[] array exists, use it

} else {

// Otherwise use the legacy DOM

}

**JAVASCRIPT ADVANCED**

* **JavaScript – Error Handling:**

There are three types of errors in programming: (a) Syntax Errors, (b) Runtime Errors, and (c) Logical Errors.

## **Syntax Errors**

Syntax errors, also called **parsing errors,** occur at compile time in traditional programming languages and at interpret time in JavaScript.

For example, the following line causes a syntax error because it is missing a closing parenthesis.

<script>

<!--

window.print(;

//-->

</script>

When a syntax error occurs in JavaScript, only the code contained within the same thread as the syntax error is affected and the rest of the code in other threads gets executed assuming nothing in them depends on the code containing the error.

## **Runtime Errors**

Runtime errors, also called **exceptions,** occur during execution (after compilation/interpretation).

For example, the following line causes a runtime error because here the syntax is correct, but at runtime, it is trying to call a method that does not exist.

<script>

<!--

window.printme();

//-->

</script>

Exceptions also affect the thread in which they occur, allowing other JavaScript threads to continue normal execution.

## **Logical Errors**

Logic errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.

You cannot catch those errors, because it depends on your business requirement what type of logic you want to put in your program.

## **The try...catch...finally Statement**

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the **try...catch...finally** construct as well as the **throw** operator to handle exceptions.

You can **catch** programmer-generated and **runtime** exceptions, but you cannot **catch** JavaScript syntax errors.

Here is the **try...catch...finally** block syntax −

<script>

<!--

try {

// Code to run

[break;]

}

catch ( e ) {

// Code to run if an exception occurs

[break;]

}

[ finally {

// Code that is always executed regardless of

// an exception occurring

}]

//-->

</script>

The **try** block must be followed by either exactly one **catch** block or one **finally** block (or one of both). When an exception occurs in the **try** block, the exception is placed in **e** and the **catch** block is executed. The optional **finally** block executes unconditionally after try/catch.

### Program:

Here is an example where we are trying to call a non-existing function which in turn is raising an exception. Let us see how it behaves without **try...catch**−

<!DOCTYPE html>

<html>

<head>

<script>

<!--

function myFunc() {

var a = 100;

alert("Value of variable a is : " + a );

}

//-->

</script>

</head>

<body>

<p>Click the following to see the result:</p>

<form>

<input type = "button" value = "Click Me" onclick = "myFunc();" />

</form>

</body>

</html>

### Output

### 

### After clicking on click me

### 

### Program:

Now let us try to catch this exception using **try...catch** and display a user-friendly message. You can also suppress this message, if you want to hide this error from a user.

<!DOCTYPE html>

<html>

<head>

<script>

<!--

function myFunc() {

var a = 100;

try {

alert("Value of variable a is : " + a );

}

catch ( e ) {

alert("Error: " + e.description );

}

}

//-->

</script>

</head>

<body>

<p>Click the following to see the result:</p>

<form>

<input type = "button" value = "Click Me" onclick = "myFunc();" />

</form>

</body>

</html>

### Output

### 

### After clicking on click me

### 

### Program:

You can use **finally** block which will always execute unconditionally after the try/catch. Here is an example.

<!DOCTYPE html>

<html>

<head>

<script>

<!--

function myFunc() {

var a = 100;

try {

alert("Value of variable a is : " + a );

}

catch ( e ) {

alert("Error: " + e.description );

}

finally {

alert("Finally block will always execute!" );

}

}

//-->

</script>

</head>

<body>

<p>Click the following to see the result:</p>

<form>

<input type = "button" value = "Click Me" onclick = "myFunc();" />

</form>

</body>

</html>

### Output

### 

### After clicking on click me

### 

## **The throw Statement**

You can use **throw** statement to raise your built-in exceptions or your customized exceptions. Later these exceptions can be captured and you can take an appropriate action.

### Program:

The following example demonstrates how to use a **throw** statement.

<!DOCTYPE html>

<html>

<head>

<script>

<!--

function myFunc() {

var a = 100;

var b = 0;

try {

if ( b == 0 ) {

throw( "Divide by zero error." );

} else {

var c = a / b;

}

}

catch ( e ) {

alert("Error: " + e );

}

}

//-->

</script>

</head>

<body>

<p>Click the following to see the result:</p>

<form>

<input type = "button" value = "Click Me" onclick = "myFunc();" />

</form>

</body>

</html>

### Output

### 

### After clicking on click me

### 

You can raise an exception in one function using a string, integer, Boolean, or an object and then you can capture that exception either in the same function as we did above, or in another function using a **try...catch** block.

## **The onerror() Method**

### Program:

The **onerror** event handler was the first feature to facilitate error handling in JavaScript. The **error** event is fired on the window object whenever an exception occurs on the page.

<!DOCTYPE html>

<html>

<head>

<script>

<!--

window.onerror = function () {

alert("An error occurred.");

}

//-->

</script>

</head>

<body>

<p>Click the following to see the result:</p>

<form>

<input type = "button" value = "Click Me" onclick = "myFunc();" />

</form>

</body>

</html>

### Output

### 

### After clicking on click me

### 

The **onerror** event handler provides three pieces of information to identify the exact nature of the error −

* **Error message** − The same message that the browser would display for the given error
* **URL** − The file in which the error occurred
* **Line number**− The line number in the given URL that caused the error

Here is the example to show how to extract this information.

### Program:

<!DOCTYPE html>

<html>

<head>

<script>

<!--

window.onerror = function (msg, url, line) {

alert("Message : " + msg );

alert("url : " + url );

alert("Line number : " + line );

}

//-->

</script>

</head>

<body>

<p>Click the following to see the result:</p>

<form>

<input type = "button" value = "Click Me" onclick = "myFunc();" />

</form>

</body>

</html>

### Output

### 

### After clicking on click me

### 

You can display extracted information in whatever way you think it is better.

You can use an **onerror** method, as shown below, to display an error message in case there is any problem in loading an image.

<img src="myimage.gif" onerror="alert('An error occurred loading the image.')" />

You can use **onerror** with many HTML tags to display appropriate messages in case of errors.

* **JavaScript – Validations:**

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

* **Basic Validation** − First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
* **Data Format Validation** − Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

### Program

We will take an example to understand the process of validation. Here is a simple form in html format.

<!DOCUMENT html>

<html>

<head>

<title>Form Validation</title>

<script type = "text/javascript">

<!--

// Form validation code will come here.

//-->

</script>

</head>

<body>

<form action = "/cgi-bin/test.cgi" name = "myForm" onsubmit = "return(validate());">

<table cellspacing = "2" cellpadding = "2" border = "1">

<tr>

<td align = "right">Name</td>

<td><input type = "text" name = "Name" /></td>

</tr>

<tr>

<td align = "right">EMail</td>

<td><input type = "text" name = "EMail" /></td>

</tr>

<tr>

<td align = "right">Zip Code</td>

<td><input type = "text" name = "Zip" /></td>

</tr>

<tr>

<td align = "right">Country</td>

<td>

<select name = "Country">

<option value = "-1" selected>[choose yours]</option>

<option value = "1">USA</option>

<option value = "2">UK</option>

<option value = "3">INDIA</option>

</select>

</td>

</tr>

<tr>

<td align = "right"></td>

<td><input type = "submit" value = "Submit" /></td>

</tr>

</table>

</form>

</body>

</html>

### Output

### 

## **Basic Form Validation**

First let us see how to do a basic form validation. In the above form, we are calling **validate()** to validate data when **onsubmit** event is occurring. The following code shows the implementation of this validate() function.

<script type = "text/javascript">

<!--

// Form validation code will come here.

function validate() {

if( document.myForm.Name.value == "" ) {

alert( "Please provide your name!" );

document.myForm.Name.focus() ;

return false;

}

if( document.myForm.EMail.value == "" ) {

alert( "Please provide your Email!" );

document.myForm.EMail.focus() ;

return false;

}

if( document.myForm.Zip.value == "" || isNaN( document.myForm.Zip.value ) ||

document.myForm.Zip.value.length != 5 ) {

alert( "Please provide a zip in the format #####." );

document.myForm.Zip.focus() ;

return false;

}

if( document.myForm.Country.value == "-1" ) {

alert( "Please provide your country!" );

return false;

}

return( true );

}

//-->

</script>

## **Data Format Validation**

Now we will see how we can validate our entered form data before submitting it to the web server.

The following example shows how to validate an entered email address. An email address must contain at least a ‘@’ sign and a dot (.). Also, the ‘@’ must not be the first character of the email address, and the last dot must at least be one character after the ‘@’ sign.

### Program

Try the following code for email validation.

<script type = "text/javascript">

<!--

function validateEmail() {

var emailID = document.myForm.EMail.value;

atpos = emailID.indexOf("@");

dotpos = emailID.lastIndexOf(".");

if (atpos < 1 || ( dotpos - atpos < 2 )) {

alert("Please enter correct email ID")

document.myForm.EMail.focus() ;

return false;

}

return( true );

}

//-->

</script>

* **JavaScript – Animation:**

You can use JavaScript to create a complex animation having, but not limited to, the following elements −

* + Fireworks
  + Fade Effect
  + Roll-in or Roll-out
  + Page-in or Page-out
  + Object movements

You might be interested in existing JavaScript based animation library: [Script.Aculo.us](https://www.tutorialspoint.com/script.aculo.us/scriptaculous_effects.htm).

This tutorial provides a basic understanding of how to use JavaScript to create an animation.

JavaScript can be used to move a number of DOM elements (<img />, <div> or any other HTML element) around the page according to some sort of pattern determined by a logical equation or function.

JavaScript provides the following two functions to be frequently used in animation programs.

* **setTimeout( function, duration)** − This function calls **function** after **duration** milliseconds from now.
* **setInterval(function, duration)** − This function calls **function** after every **duration** milliseconds.
* **clearTimeout(setTimeout\_variable)** − This function calls clears any timer set by the setTimeout() functions.

JavaScript can also set a number of attributes of a DOM object including its position on the screen. You can set *top* and left attribute of an object to position it anywhere on the screen. Here is its syntax.

// Set distance from left edge of the screen.

object.style.left = distance in pixels or points;

or

// Set distance from top edge of the screen.

object.style.top = distance in pixels or points;

## **Manual Animation**

So let's implement one simple animation using DOM object properties and JavaScript functions as follows. The following list contains different DOM methods.

* We are using the JavaScript function **getElementById()** to get a DOM object and then assigning it to a global variable **imgObj**.
* We have defined an initialization function **init()** to initialize **imgObj** where we have set its **position** and **left** attributes.
* We are calling initialization function at the time of window load.
* Finally, we are calling **moveRight()** function to increase the left distance by 10 pixels. You could also set it to a negative value to move it to the left side.

### Program

Try the following example.

<!DOCTYPE html>

<html>

<head>

<title>JavaScript Animation</title>

<script>

<!--

var imgObj = null;

function init() {

imgObj = document.getElementById('myImage');

imgObj.style.position= 'relative';

imgObj.style.left = '0px';

}

function moveRight() {

imgObj.style.left = parseInt(imgObj.style.left) + 10 + 'px';

}

window.onload = init;

//-->

</script>

</head>

<body>

<form>

<img id = "myImage" src = "/images/html.gif" />

<p>Click button below to move the image to right</p>

<input type = "button" value = "Click Me" onclick = "moveRight();" />

</form>

</body>

</html>

### Output

### 

## **Automated Animation**

In the above example, we saw how an image moves to right with every click. We can automate this process by using the JavaScript function **setTimeout()** as follows −

Here we have added more methods. So let's see what is new here −

* The **moveRight()** function is calling **setTimeout()** function to set the position of *imgObj*.
* We have added a new function **stop()** to clear the timer set by **setTimeout()** function and to set the object at its initial position.

### Program

Try the following example code.

<!DOCTYPE html>

<html>

<head>

<title>JavaScript Animation</title>

<script>

<!--

var imgObj = null;

var animate ;

function init() {

imgObj = document.getElementById('myImage');

imgObj.style.position= 'relative';

imgObj.style.left = '0px';

}

function moveRight() {

imgObj.style.left = parseInt(imgObj.style.left) + 10 + 'px';

animate = setTimeout(moveRight,20); // call moveRight in 20msec

}

function stop() {

clearTimeout(animate);

imgObj.style.left = '0px';

}

window.onload = init;

//-->

</script>

</head>

<body>

<form>

<img id = "myImage" src = "/images/html.gif" />

<p>Click the buttons below to handle animation</p>

<input type = "button" value = "Start" onclick = "moveRight();" />

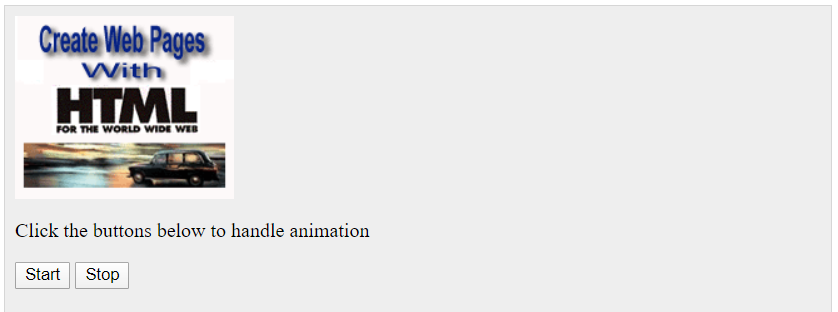
<input type = "button" value = "Stop" onclick = "stop();" />

</form>

</body>

</html>

**Output**



## **Rollover with a Mouse Event**

Here is a simple example showing image rollover with a mouse event.

Let's see what we are using in the following example −

* At the time of loading this page, the ‘if’ statement checks for the existence of the image object. If the image object is unavailable, this block will not be executed.
* The **Image()** constructor creates and preloads a new image object called **image1**.
* The src property is assigned the name of the external image file called /images/html.gif.
* Similarly, we have created **image2** object and assigned /images/http.gif in this object.
* The # (hash mark) disables the link so that the browser does not try to go to a URL when clicked. This link is an image.
* The **onMouseOver** event handler is triggered when the user's mouse moves onto the link, and the **onMouseOut** event handler is triggered when the user's mouse moves away from the link (image).
* When the mouse moves over the image, the HTTP image changes from the first image to the second one. When the mouse is moved away from the image, the original image is displayed.
* When the mouse is moved away from the link, the initial image html.gif will reappear on the screen.

<!DOCTYPE html>

<html>

<head>

<title>Rollover with a Mouse Events</title>

<script type = "text/javascript">

<!--

if(document.images) {

var image1 = new Image(); // Preload an image

image1.src = "/images/html.gif";

var image2 = new Image(); // Preload second image

image2.src = "/images/http.gif";

}

//-->

</script>

</head>

<body>

<p>Move your mouse over the image to see the result</p>

<a href = "#" onMouseOver = "document.myImage.src = image2.src;"

onMouseOut = "document.myImage.src = image1.src;">

<img name = "myImage" src = "/images/html.gif" />

</a>

</body>

</html>

* **JavaScript – Multimedia:**

The JavaScript **navigator** object includes a child object called **plugins**. This object is an array, with one entry for each plug-in installed on the browser. The navigator.plugins object is supported only by Netscape, Firefox, and Mozilla only.

### Program

Here is an example that shows how to list down all the plug-on installed in your browser −

<!DOCTYPE html>

<html>

<head>

<title>List of Plug-Ins</title>

</head>

<body>

<table border = "1">

<tr>

<th>Plug-in Name</th>

<th>Filename</th>

<th>Description</th>

</tr>

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

for (i = 0; i<navigator.plugins.length; i++) {

document.write("<tr><td>");

document.write(navigator.plugins[i].name);

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

document.write(navigator.plugins[i].filename);

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

document.write(navigator.plugins[i].description);

document.write("</td></tr>");

}

</script>

</table>

</body>

</html>

### Output

### 

## **Checking for Plug-Ins**

Each plug-in has an entry in the array. Each entry has the following properties −

* **name** − is the name of the plug-in.
* **filename** − is the executable file that was loaded to install the plug-in.
* **description** − is a description of the plug-in, supplied by the developer.
* **mimeTypes** − is an array with one entry for each MIME type supported by the plug-in.

You can use these properties in a script to find out the installed plug-ins, and then using JavaScript, you can play appropriate multimedia file. Take a look at the following example.

<!DOCTYPE html>

<html>

<head>

<title>Using Plug-Ins</title>

</head>

<body>

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

media = navigator.mimeTypes["video/quicktime"];

if (media) {

document.write("<embed src = 'quick.mov' height = 100 width = 100>");

} else {

document.write("<img src = 'quick.gif' height = 100 width = 100>");

}

</script>

</body>

</html>

### Output

### 

**NOTE** − Here we are using HTML <embed> tag to embed a multimedia file.

## **Controlling Multimedia**

Let us take one real example which works in almost all the browsers −

<!DOCTYPE html>

<html>

<head>

<title>Using Embeded Object</title>

<script type = "text/javascript">

<!--

function play() {

if (!document.demo.IsPlaying()) {

document.demo.Play();

}

}

function stop() {

if (document.demo.IsPlaying()) {

document.demo.StopPlay();

}

}

function rewind() {

if (document.demo.IsPlaying()) {

document.demo.StopPlay();

}

document.demo.Rewind();

}

//-->

</script>

</head>

<body>

<embed id = "demo" name = "demo"

src = "http://www.amrood.com/games/kumite.swf"

width = "318" height = "300" play = "false" loop = "false"

pluginspage = "http://www.macromedia.com/go/getflashplayer"

swliveconnect = "true">

<form name = "form" id = "form" action = "#" method = "get">

<input type = "button" value = "Start" onclick = "play();" />

<input type = "button" value = "Stop" onclick = "stop();" />

<input type = "button" value = "Rewind" onclick = "rewind();" />

</form>

</body>

</html>

\*\*Note: This program will work on Mozilla, Firefox or Netscape

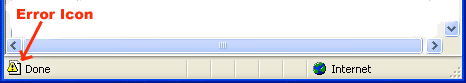
* **JavaScript – Debugging:**

Every now and then, developers commit mistakes while coding. A mistake in a program or a script is referred to as a **bug**.

The process of finding and fixing bugs is called **debugging** and is a normal part of the development process. This section covers tools and techniques that can help you with debugging tasks..

## **Error Messages in IE**

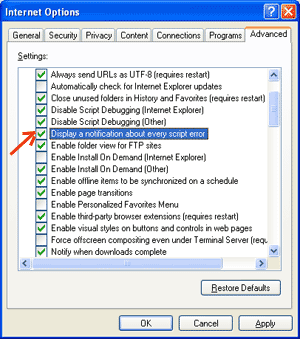
The most basic way to track down errors is by turning on error information in your browser. By default, Internet Explorer shows an error icon in the status bar when an error occurs on the page.



Double-clicking this icon takes you to a dialog box showing information about the specific error that occurred.

Since this icon is easy to overlook, Internet Explorer gives you the option to automatically show the Error dialog box whenever an error occurs.

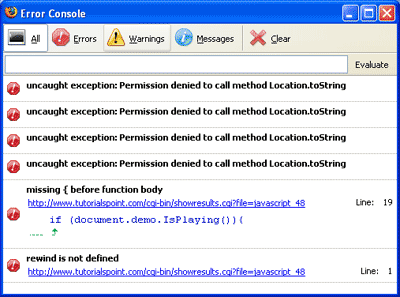
To enable this option, select **Tools → Internet Options → Advanced tab.** and then finally check the **"Display a Notification About Every Script Error"** box option as shown below −



## **Error Messages in Firefox or Mozilla**

Other browsers like Firefox, Netscape, and Mozilla send error messages to a special window called the **JavaScript Console** or **Error Consol**. To view the console, select **Tools → Error Consol or Web Development**.

Unfortunately, since these browsers give no visual indication when an error occurs, you must keep the Console open and watch for errors as your script executes.



## **Error Notifications**

Error notifications that show up on Console or through Internet Explorer dialog boxes are the result of both syntax and runtime errors. These error notification include the line number at which the error occurred.

If you are using Firefox, then you can click on the error available in the error console to go to the exact line in the script having error.

## **How to debug a Script**

There are various ways to debug your JavaScript −

### Use a JavaScript Validator

One way to check your JavaScript code for strange bugs is to run it through a program that checks it to make sure it is valid and that it follows the official syntax rules of the language. These programs are called **validating parsers** or just **validators** for short, and often come with commercial HTML and JavaScript editors.

The most convenient validator for JavaScript is Douglas Crockford's JavaScript Lint, which is available for free at [Douglas Crockford's JavaScript Lint](http://www.jslint.com/).

Simply visit that web page, paste your JavaScript (Only JavaScript) code into the text area provided, and click the jslint button. This program will parse through your JavaScript code, ensuring that all the variable and function definitions follow the correct syntax. It will also check JavaScript statements, such as if and **while,** to ensure they too follow the correct format

### Add Debugging Code to Your Programs

You can use the **alert()** or **document.write()** methods in your program to debug your code. For example, you might write something as follows −

var debugging = true;

var whichImage = "widget";

if( debugging )

alert( "Calls swapImage() with argument: " + whichImage );

var swapStatus = swapImage( whichImage );

if( debugging )

alert( "Exits swapImage() with swapStatus=" + swapStatus );

By examining the content and order of the **alert()** as they appear, you can examine the health of your program very easily.

### Use a JavaScript Debugger

A debugger is an application that places all aspects of script execution under the control of the programmer. Debuggers provide fine-grained control over the state of the script through an interface that allows you to examine and set values as well as control the flow of execution.

Once a script has been loaded into a debugger, it can be run one line at a time or instructed to halt at certain breakpoints. Once execution is halted, the programmer can examine the state of the script and its variables in order to determine if something is amiss. You can also watch variables for changes in their values.

The latest version of the Mozilla JavaScript Debugger (code-named Venkman) for both Mozilla and Netscape browsers can be downloaded at <http://www.hacksrus.com/~ginda/venkman>

## **Useful Tips for Developers**

You can keep the following tips in mind to reduce the number of errors in your scripts and simplify the debugging process −

* Use plenty of **comments**. Comments enable you to explain why you wrote the script the way you did and to explain particularly difficult sections of code.
* Always use **indentation** to make your code easy to read. Indenting statements also makes it easier for you to match up beginning and ending tags, curly braces, and other HTML and script elements.
* Write **modular code**. Whenever possible, group your statements into functions. Functions let you group related statements, and test and reuse portions of code with minimal effort.
* Be consistent in the way you name your variables and functions. Try using names that are long enough to be meaningful and that describe the contents of the variable or the purpose of the function.
* Use consistent syntax when naming variables and functions. In other words, keep them all lowercase or all uppercase; if you prefer Camel-Back notation, use it consistently.
* **Test long scripts** in a modular fashion. In other words, do not try to write the entire script before testing any portion of it. Write a piece and get it to work before adding the next portion of code.
* Use **descriptive variable and function names** and avoid using single-character names.
* **Watch your quotation marks**. Remember that quotation marks are used in pairs around strings and that both quotation marks must be of the same style (either single or double).
* **Watch your equal signs**. You should not used a single = for comparison purpose.
* Declare **variables explicitly** using the **var** keyword.
* **JavaScript – Image Map:**

You can use JavaScript to create client-side image map. Client-side image maps are enabled by the **usemap** attribute for the <img /> tag and defined by special <map> and <area> extension tags.

The image that is going to form the map is inserted into the page using the <img /> element as normal, except that it carries an extra attribute called **usemap**. The value of the usemap attribute is the value of the name attribute on the <map> element, which you are about to meet, preceded by a pound or hash sign.

The <map> element actually creates the map for the image and usually follows directly after the <img /> element. It acts as a container for the <area /> elements that actually define the clickable hotspots. The <map> element carries only one attribute, the **name** attribute, which is the name that identifies the map. This is how the <img /> element knows which <map> element to use.

The <area> element specifies the shape and the coordinates that define the boundaries of each clickable hotspot.

The following code combines imagemaps and JavaScript to produce a message in a text box when the mouse is moved over different parts of an image.

<!DOCTYPE HTML>

<html>

<head>

<title>Using JavaScript Image Map</title>

<script>

<!--

function showTutorial(name) {

document.myform.stage.value = name

}

//-->

</script>

</head>

<body>

<form name = "myform">

<input type = "text" name = "stage" size = "20" />

</form>

<!-- Create Mappings -->

<img src = "/images/usemap.gif" alt = "HTML Map" border = "0" usemap = "#tutorials"/>

<map name = "tutorials">

<area shape="poly"

coords = "74,0,113,29,98,72,52,72,38,27"

href = "/perl/index.htm" alt = "Perl Tutorial"

target = "\_self"

onMouseOver = "showTutorial('perl')"

onMouseOut = "showTutorial('')"/>

<area shape = "rect"

coords = "22,83,126,125"

href = "/html/index.htm" alt = "HTML Tutorial"

target = "\_self"

onMouseOver = "showTutorial('html')"

onMouseOut = "showTutorial('')"/>

<area shape = "circle"

coords = "73,168,32"

href = "/php/index.htm" alt = "PHP Tutorial"

target = "\_self"

onMouseOver = "showTutorial('php')"

onMouseOut = "showTutorial('')"/>

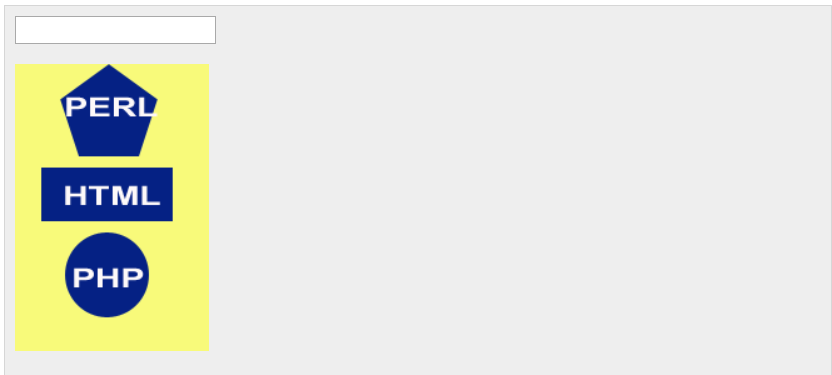
</map>

</body>

</html>

## **Output**

You can feel the map concept by placing the mouse cursor on the image object.



* **JavaScript – Browsers:**

It is important to understand the differences between different browsers in order to handle each in the way it is expected. So it is important to know which browser your web page is running in.

To get information about the browser your webpage is currently running in, use the built- in **navigator** object.

## **Navigator Properties**

There are several Navigator related properties that you can use in your Web page. The following is a list of the names and descriptions of each.

|  |  |
| --- | --- |
| **Sr.No.** | **Property & Description** |
| 1 | **appCodeName**  This property is a string that contains the code name of the browser, Netscape for Netscape and Microsoft Internet Explorer for Internet Explorer. |
| 2 | **appVersion**  This property is a string that contains the version of the browser as well as other useful information such as its language and compatibility. |
| 3 | **language**  This property contains the two-letter abbreviation for the language that is used by the browser. Netscape only. |
| 4 | **mimTypes[]**  This property is an array that contains all MIME types supported by the client. Netscape only. |
| 5 | **platform[]**  This property is a string that contains the platform for which the browser was compiled."Win32" for 32-bit Windows operating systems |
| 6 | **plugins[]**  This property is an array containing all the plug-ins that have been installed on the client. Netscape only. |
| 7 | **userAgent[]**  This property is a string that contains the code name and version of the browser. This value is sent to the originating server to identify the client. |

## **Navigator Methods**

There are several Navigator-specific methods. Here is a list of their names and descriptions.

|  |  |
| --- | --- |
| **Sr.No.** | **Description** |
| 1 | **javaEnabled()**  This method determines if JavaScript is enabled in the client. If JavaScript is enabled, this method returns true; otherwise, it returns false. |
| 2 | **plugings.refresh**  This method makes newly installed plug-ins available and populates the plugins array with all new plug-in names. Netscape only. |
| 3 | **preference(name,value)**  This method allows a signed script to get and set some Netscape preferences. If the second parameter is omitted, this method will return the value of the specified preference; otherwise, it sets the value. Netscape only. |
| 4 | **taintEnabled()**  This method returns true if data tainting is enabled; false otherwise. |

## **Browser Detection**

There is a simple JavaScript which can be used to find out the name of a browser and then accordingly an HTML page can be served to the user.

<!DOCTYPE html>

<html>

<head>

<title>Browser Detection Example</title>

</head>

<body>

<script>

<!--

var userAgent = navigator.userAgent;

var opera = (userAgent.indexOf('Opera') != -1);

var ie = (userAgent.indexOf('MSIE') != -1);

var gecko = (userAgent.indexOf('Gecko') != -1);

var netscape = (userAgent.indexOf('Mozilla') != -1);

var version = navigator.appVersion;

if (opera) {

document.write("Opera based browser");

// Keep your opera specific URL here.

} else if (gecko) {

document.write("Mozilla based browser");

// Keep your gecko specific URL here.

} else if (ie) {

document.write("IE based browser");

// Keep your IE specific URL here.

} else if (netscape) {

document.write("Netscape based browser");

// Keep your Netscape specific URL here.

} else {

document.write("Unknown browser");

}

// You can include version to along with any above condition.

document.write("<br /> Browser version info : " + version );

//-->

</script>

</body>

</html>

### Output

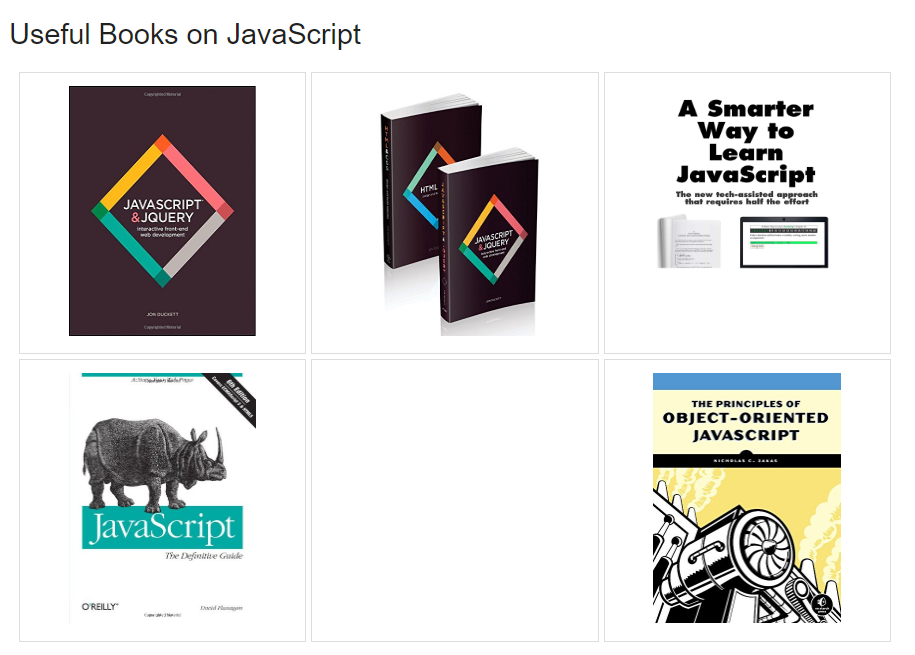
### 

**JAVASCRIPT USEFUL RESOURCES**

The following resources contain additional information on JavaScript. Please use them to get more in-depth knowledge on this topic.

## **Useful Links on JavaScript**

* [JavaScript Wiki](https://en.wikipedia.org/wiki/JavaScript) − Wiki Page for JavaScript
* [JavaScript](https://developer.mozilla.org/en-US/Learn/Getting_started_with_the_web/JavaScript_basics) − JavaScript basics Tutorial



**JAVASCRIPT – INTERVIEW QUESTIONS & ANSWERS**

Dear readers, these **JavaScript Interview Questions** have been designed specially to get you acquainted with the nature of questions you may encounter during your interview for the subject of **JavaScript**. Also good interviewers hardly plan to ask any particular question during your interview, normally questions start with some basic concept of the subject and later they continue based on further discussion and what you answer:

**What is JavaScript?**

JavaScript is a lightweight, interpreted programming language with object-oriented capabilities that allows you to build interactivity into otherwise static HTML pages.

The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

**Name some of the JavaScript features.**

Following are the features of JavaScript −

* JavaScript is a lightweight, interpreted programming language.
* JavaScript is designed for creating network-centric applications.
* JavaScript is complementary to and integrated with Java.
* JavaScript is complementary to and integrated with HTML.
* JavaScript is open and cross-platform.

**What are the advantages of using JavaScript?**

Following are the advantages of using JavaScript −

* **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.

**What are disadvantages of using 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 multithreading or multiprocess capabilities.

**Is JavaScript a case-sensitive language?**

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

**How can you create an Object in JavaScript?**

JavaScript supports Object concept very well. You can create an object using the object literal as follows −

var emp = {

name: "Zara",

age: 10

};

**How can you read properties of an Object in JavaScript?**

You can write and read properties of an object using the dot notation as follows −

// Getting object properties

emp.name // ==> Zara

emp.age // ==> 10

// Setting object properties

emp.name = "Daisy" // <== Daisy

emp.age = 20 // <== 20

**How can you create an Array in JavaScript?**

You can define arrays using the array literal as follows −

var x = [];

var y = [1, 2, 3, 4, 5];

**How to read elements of an array in JavaScript?**

An array has a length property that is useful for iteration. We can read elements of an array as follows −

var x = [1, 2, 3, 4, 5];

for (var i = 0; i < x.length; i++) {

// Do something with x[i]

}

**What is a named function in JavaScript? How to define a named function?**

A named function has a name when it is defined. A named function can be defined using function keyword as follows −

function named(){

// do some stuff here

}

**How many types of functions JavaScript supports?**

A function in JavaScript can be either named or anonymous.

**How to define a anonymous function?**

An anonymous function can be defined in similar way as a normal function but it would not have any name.

**Can you assign a anonymous function to a variable?**

Yes! An anonymous function can be assigned to a variable.

**Can you pass a anonymous function as an argument to another function?**

Yes! An anonymous function can be passed as an argument to another function.

**What is arguments object in JavaScript?**

JavaScript variable arguments represents the arguments passed to a function.

**How can you get the type of arguments passed to a function?**

Using typeof operator, we can get the type of arguments passed to a function. For example −

function func(x){

console.log(typeof x, arguments.length);

}

func(); //==> "undefined", 0

func(1); //==> "number", 1

func("1", "2", "3"); //==> "string", 3

**How can you get the total number of arguments passed to a function?**

Using arguments.length property, we can get the total number of arguments passed to a function. For example −

function func(x){

console.log(typeof x, arguments.length);

}

func(); //==> "undefined", 0

func(1); //==> "number", 1

func("1", "2", "3"); //==> "string", 3

**How can you get the reference of a caller function inside a function?**

The arguments object has a callee property, which refers to the function you're inside of. For example −

function func() {

return arguments.callee;

}

func(); // ==> func

**What is the purpose of 'this' operator in JavaScript?**

JavaScript famous keyword this always refers to the current context.

**What are the valid scopes of a variable in JavaScript?**

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

* **Global Variables −** A global variable has global scope which means it is visible everywhere 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.

**Which type of variable among global and local, takes precedence over other if names are same?**

A local variable takes precedence over a global variable with the same name.

**What is callback?**

A callback is a plain JavaScript function passed to some method as an argument or option. Some callbacks are just events, called to give the user a chance to react when a certain state is triggered.

**What is closure?**

Closures are created whenever a variable that is defined outside the current scope is accessed from within some inner scope.

**Give an example of closure?**

Following example shows how the variable counter is visible within the create, increment, and print functions, but not outside of them −

function create() {

var counter = 0;

return {

increment: function() {

counter++;

},

print: function() {

console.log(counter);

}

}

}

var c = create();

c.increment();

c.print(); // ==> 1

**Which built-in method returns the character at the specified index?**

charAt() method returns the character at the specified index.

**Which built-in method combines the text of two strings and returns a new string?**

concat() method returns the character at the specified index.

**Which built-in method calls a function for each element in the array?**

forEach() method calls a function for each element in the array.

**Which built-in method returns the index within the calling String object of the first occurrence of the specified value?**

indexOf() method returns the index within the calling String object of the first occurrence of the specified value, or −1 if not found.

**Which built-in method returns the length of the string?**

length() method returns the length of the string.

**Which built-in method removes the last element from an array and returns that element?**

pop() method removes the last element from an array and returns that element.

**Which built-in method adds one or more elements to the end of an array and returns the new length of the array?**

push() method adds one or more elements to the end of an array and returns the new length of the array.

**Which built-in method reverses the order of the elements of an array?**

reverse() method reverses the order of the elements of an array −− the first becomes the last, and the last becomes the first.

**Which built-in method sorts the elements of an array?**

sort() method sorts the elements of an array.

**Which built-in method returns the characters in a string beginning at the specified location?**

substr() method returns the characters in a string beginning at the specified location through the specified number of characters.

**Which built-in method returns the calling string value converted to lower case?**

toLowerCase() method returns the calling string value converted to lower case.

**Which built-in method returns the calling string value converted to upper case?**

toUpperCase() method returns the calling string value converted to upper case.

**Which built-in method returns the string representation of the number's value?**

toString() method returns the string representation of the number's value.

**What are the variable naming conventions in JavaScript?**

While naming your variables in JavaScript keep following rules in mind.

You should not use any of the JavaScript reserved keyword as variable name. These keywords are mentioned in the next section. For example, break or boolean variable names are not valid.

JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or the underscore character. For example, 123test is an invalid variable name but \_123test is a valid one.

JavaScript variable names are case sensitive. For example, Name and name are two different variables.

**How typeof operator works?**

The typeof 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.

**What typeof returns for a null value?**

It returns "object".

**Can you access Cookie using javascript?**

JavaScript can also manipulate cookies using the cookie property of the Document object. JavaScript can read, create, modify, and delete the cookie or cookies that apply to the current web page.

**How to create a Cookie using JavaScript?**

The simplest way to create a cookie is to assign a string value to the document.cookie object, which looks like this −

Syntax −

document.cookie = "key1 = value1; key2 = value2; expires = date";

Here expires attribute is option. If you provide this attribute with a valid date or time then cookie will expire at the given date or time and after that cookies' value will not be accessible.

**How to read a Cookie using JavaScript?**

Reading a cookie is just as simple as writing one, because the value of the document.cookie object is the cookie. So you can use this string whenever you want to access the cookie.

The document.cookie string will keep a list of name = value pairs separated by semicolons, where name is the name of a cookie and value is its string value.

You can use strings' split() function to break the string into key and values.

**How to delete a Cookie using JavaScript?**

Sometimes you will want to delete a cookie so that subsequent attempts to read the cookie return nothing. To do this, you just need to set the expiration date to a time in the past.

**How to redirect a url using JavaScript?**

his is very simple to do a page redirect using JavaScript at client side. To redirect your site visitors to a new page, you just need to add a line in your head section as follows −

<head>

<script type="text/javascript">

<!--

window.location="http://www.newlocation.com";

//-->

</script>

</head>

**How to print a web page using javascript?**

JavaScript helps you to implement this functionality using print function of window object. The JavaScript print function window.print() will print the current web page when executed.

**What is Date object in JavaScript?**

The Date object is a datatype built into the JavaScript language. Date objects are created with the new Date( ).

Once a Date object is created, a number of methods allow you to operate on it. Most methods simply allow you to get and set the year, month, day, hour, minute, second, and millisecond fields of the object, using either local time or UTC (universal, or GMT) time.

**What is Number object in JavaScript?**

he Number object represents numerical date, either integers or floating-point numbers. In general, you do not need to worry about Number objects because the browser automatically converts number literals to instances of the number class.

Syntax −

Creating a number object −

var val = new Number(number);

If the argument cannot be converted into a number, it returns NaN (Not-a-Number).

**How to handle exceptions in JavaScript?**

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the try...catch...finally construct as well as the throw operator to handle exceptions.

You can catch programmer-generated and runtime exceptions, but you cannot catch JavaScript syntax errors.

**What is purpose of onError event handler in JavaScript?**

The onerror event handler was the first feature to facilitate error handling for JavaScript. The error event is fired on the window object whenever an exception occurs on the page.

The onerror event handler provides three pieces of information to identify the exact nature of the error −

* **Error message −** The same message that the browser would display for the given error.
* **URL −** The file in which the error occurred.
* **Line number −** The line number in the given URL that caused the error.

## **What is Next?**

Further you can go through your past assignments you have done with the subject and make sure you are able to speak confidently on them. If you are fresher then interviewer does not expect you will answer very complex questions, rather you have to make your basics concepts very strong.

Second it really doesn't matter much if you could not answer few questions but it matters that whatever you answered, you must have answered with confidence. So just feel confident during your interview. All the very best for your future endeavor. Cheers :-)