**Introduction to JavaScript**

**1.Introduction to JavaScript**

**JavaScript** is a lightweight, cross-platform, and interpreted compiled programming language which is also known as the scripting language for webpages. It is well-known for the development of web pages, many non-browser environments also use it. JavaScript can be used for [**Client-side**](https://www.geeksforgeeks.org/server-side-client-side-programming/) developments as well as [**Server-side**](https://www.geeksforgeeks.org/server-side-client-side-programming/) developments. Javascript is both imperative and declarative type of language. JavaScript contains a standard library of objects, like [**Array**](https://www.geeksforgeeks.org/arrays-in-javascript/), [**Date**](https://www.geeksforgeeks.org/javascript-date-objects/), and [**Math**](https://www.geeksforgeeks.org/javascript-math-object/), and a core set of language elements like [**operators**](https://www.geeksforgeeks.org/javascript-operators/), **control structures**, and [**statements**](https://www.geeksforgeeks.org/javascript-statements/).

* **Client-side:** It supplies objects to control a browser and its [Document Object Model (DOM).](https://www.geeksforgeeks.org/dom-document-object-model/) Like if client-side extensions allow an application to place elements on an HTML form and respond to user events such as **mouse clicks**, **form input**, and **page navigation**. Useful libraries for the client-side are [**AngularJS**](https://www.geeksforgeeks.org/introduction-to-angularjs/), [**ReactJS**](https://www.geeksforgeeks.org/react-js-introduction-working/), [**VueJS**](https://www.geeksforgeeks.org/vue-js/) and so many others.
* **Server-side:** It supplies objects relevant to running JavaScript on a server. Like if the server-side extensions allow an application to communicate with a database, and provide continuity of information from one invocation to another of the application, or perform file manipulations on a server. The useful framework which is the most famous these days is [**node.js**](https://www.geeksforgeeks.org/introduction-to-nodejs/).
* **Declarative programming –**In this type of language we are concern about how it is to be done , basically here logical computation require . Here  main goal is to describe the desired result without direct dictation on how to get it like  arrow function do .

**JavaScript can be added to your HTML file in**[**two ways**](https://www.geeksforgeeks.org/where-to-put-javascript-in-an-html-document/)**:**

* **Internal JS:** We can add JavaScript directly to our HTML file by writing the code inside the <script> tag. The <script> tag can either be placed inside the <head> or the <body> tag according to the requirement.
* [**External JS**](https://www.geeksforgeeks.org/what-is-external-javascript/)**:** We can write JavaScript code in other file having an extension.js and then link this file inside the <head> tag of the HTML file in which we want to add this code.

**Features of JavaScript:** According to a recent survey conducted by **Stack Overflow**, JavaScript is the most popular language on earth.With advances in browser technology and JavaScript having moved into the server with Node.js and other frameworks, JavaScript is capable of so much more. Here are a few things that we can do with JavaScript:

* JavaScript was created in the first place for DOM(document object model) manipulation. Earlier websites were mostly static, after JS was created dynamic Web sites were made.
* Functions in JS are objects. They may have properties and methods just like another object. They can be passed as arguments in other functions.
* Can handle date and time.
* Performs Form Validation although the forms are created using HTML.
* No compiler is needed.

**Applications of JavaScript:**

* **Web Development:** Adding interactivity and behavior to static sites JavaScript was invented to do this in 1995. By using AngularJS that can be achieved so easily.
* **Web Applications:** With technology, browsers have improved to the extent that a language was required to create robust web applications. When we explore a map in Google Maps then we only need to click and drag the mouse. All detailed view is just a click away, and this is possible only because of JavaScript. It uses Application Programming Interfaces(APIs) that provide extra power to the code.
* **Server Applications:** With the help of Node.js, JavaScript made its way from client to server and node.js is the most powerful on the server-side.
* **Games:** Not only in websites, but JavaScript also helps in creating games for leisure. The combination of JavaScript and HTML 5 makes JavaScript popular in game development as well. It provides the EaseJS library which provides solutions for working with rich graphics.
* **Smartwatches:** JavaScript is being used in all possible devices and applications. It provides a library PebbleJS which is used in smartwatch applications. This framework works for applications that require the internet for its functioning.
* **Art:** Artists and designers can create whatever they want using JavaScript to draw on HTML 5 canvas, and make the sound more effective also can be used [**p5.js**](https://www.geeksforgeeks.org/p5-js-introduction/) library.
* **Machine Learning:** This JavaScript ml5.js library can be used in web development by using machine learning.
* **Mobile Applications:**JavaScript can also be used to build an application for non-web contexts. The features and uses of JavaScript make it a powerful tool for creating mobile applications. This is a Framework for building web and mobile apps using JavaScript. Using React Native, we can build mobile applications for different operating systems. We do not require to write code for different systems. Write once use it anywhere!

**Limitations of JavaScript:**

* **Security risks:** JavaScript can be used to fetch data using AJAX or by manipulating tags that load data such as <img>, <object>, <script>. These attacks are called **cross site script attacks**. They inject JS that is not the part of the site into the visitor’s browser thus fetching the details.
* **Performance:**JavaScript does not provide the same level of performance as offered by many traditional languages as a complex program written in JavaScript would be comparatively slow. But as JavaScript is used to perform simple tasks in a browser, so performance is not considered a big restriction in its use.
* **Complexity:**To master a scripting language, programmers must have a thorough knowledge of all the programming concepts, core language objects, client and server-side objects otherwise it would be difficult for them to write advanced scripts using JavaScript.
* **Weak error handling and type checking facilities:**It is weakly typed language as there is no need to specify the data type of the variable. So wrong type checking is not performed by compile.

# Internal & External JavaScript:

You can use JavaScript code in two ways.

1. You can either include the JavaScript code **internally within your HTML document**itself.
2. You can keep the JavaScript code in **a separate external file** and then point to that file from your HTML document.

## **What is Internal JavaScript?**

**Find today program using internal JavaScript**

**Today.html**

**<html>**

**<head>**

**<title>My First JavaScript code!!!</title>**

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

**// Create a Date Object**

**var day = new Date();**

**// Use getDay function to obtain todays Day.**

**// getDay() method returns the day of the week as a number like 0 for Sunday, 1 for Monday,., 6**

**// This value is stored in today variable**

**var today = day.getDay();**

**// To get the name of the day as Sunday, Monday or Saturday, we have created an array named weekday and stored the values**

**var weekday = new Array(7);**

**weekday[0]="Sunday";**

**weekday[1]="Monday";**

**weekday[2]="Tuesday";**

**weekday[3]="Wednesday";**

**weekday[4]="Thursday";**

**weekday[5]="Friday";**

**weekday[6]="Saturday";**

**// weekday[today] will return the day of the week as we want**

**document.write("Today is " + weekday[today] + ".");**

**</script>**

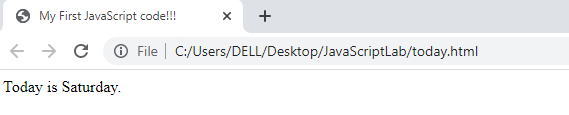
**</head>**

**<body>**

**</body>**

**</html>**

**Output:**

****

## **2. Applying JavaScript (internal and external)**

## **What is External JavaScript?**

You plan to display the current date and time in all your web pages. Suppose you wrote the code and copied into all your web pages (say 100). But later, you want to change the format in which the date or time is displayed. In this case, you will have to make changes in all the 100 web pages. This will be a very time consuming and difficult task.

So, save the JavaScript code in a new file with the extension .js. Then, add a line of code in all your web pages to point to your .js file like this:

<script type="text/javascript" src="currentdetails.js">

**Note**: It is assumed that the .js file and all your web pages are in the same folder. If the external.js file is in a different folder, you need to specify the full path to your file in the src attribute.

**How to link external JavaScript**

**dateAndTime.js**

var currentDate = new Date();

var day = currentDate.getDate();

var month = currentDate.getMonth() + 1;

var monthName;

var hours = currentDate.getHours();

var mins = currentDate.getMinutes();

var secs = currentDate.getSeconds();

var strToAppend;

if (hours >12 )

{

hours1 = "0" + (hours - 12);

strToAppend = "PM";

}

else if (hours <12)

{

hours1 = "0" + hours;

strToAppend = "AM";

}

else

{

hours1 = hours;

strToAppend = "PM";

}

if(mins<10)

mins = "0" + mins;

if (secs<10)

secs = "0" + secs;

switch (month)

{

case 1:

monthName = "January";

break;

case 2:

monthName = "February";

break;

case 3:

monthName = "March";

break;

case 4:

monthName = "April";

break;

case 5:

monthName = "May";

break;

case 6:

monthName = "June";

break;

case 7:

monthName = "July";

break;

case 8:

monthName = "August";

break;

case 9:

monthName = "September";

break;

case 10:

monthName = "October";

break;

case 11:

monthName = "November";

break;

case 12:

monthName = "December";

break;

}

var year = currentDate.getFullYear();

var myString;

myString = "Today is " + day + " - " + monthName + " - " + year + ".<br />Current time is " + hours1 + ":" + mins + ":" + secs + " " + strToAppend + ".";

document.write(myString);

**dateAndTime.html**

<html>

<head>

<title>My External JavaScript Code!!!</title>

<script type="text/javascript" src="currentdetails.js">

</script>

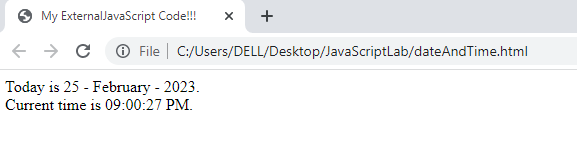
</head>

<body>

</body>

</html>

**Output:**



### **Or**

**Syntax:**

**InternalJavaScript.html**

<!DOCTYPE html>

<html>

<head>

<title>Internal JS</title>

</head>

<body>

<h2>Welcome to Guntur Engineering College</h2>

<script>

<!Internal Javascript>

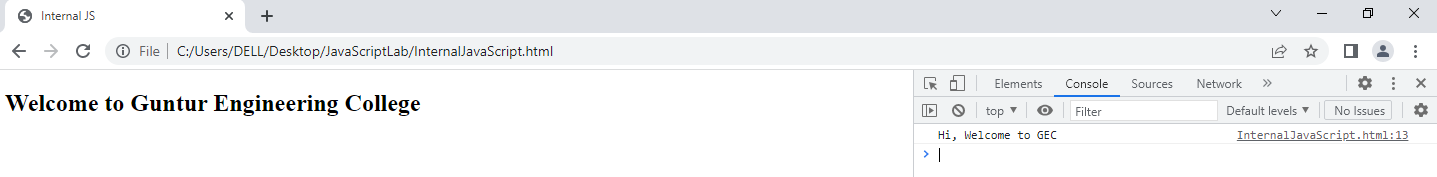
console.log("Hi Geeks, Welcome to GEC");

</script>

</body>

</html>

**Output: For console press F12 on Browser**



**ExternalJavaScript.html**

<!DOCTYPE html>

<html>

<head>

<title>External JS</title>

</head>

<body>

<h2>Welcome to Guntur Engineering College</h2>

<script src="GEC.js"></script> <!External Javascript >

</body>

</html>

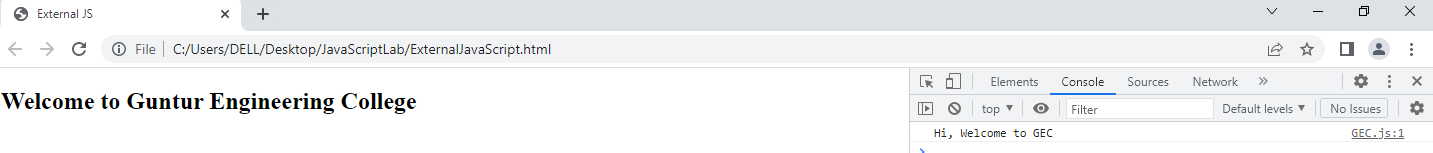
**GEC.js**

console.log("Hi, Welcome to GEC");

**Output:** The output will display on the console.

**Welcome to Guntur Engineering College**

**For console press F12 on Browser**

****

### **When to Use Internal and External JavaScript Code?**

If you have only a few lines of code that is specific to a particular webpage, then it is better to keep your JavaScript code internally within your HTML document.

On the other hand, if your JavaScript code is used in many web pages, then you should consider keeping your code in a separate file. In that case, if you wish to make some changes to your code. you just have to change only one file which makes code maintenance easy. If your code is too long, then also it is better to keep it in a separate file. This helps in easy debugging.

## **3.Understanding the JavaScript Syntax**

The syntax of JavaScript is the set of rules that define a correctly structured JavaScript program.

A JavaScript consists of JavaScript statements that are placed within the <script></script> HTML tags in a web page, or within the external JavaScript file having **.js** extension.

var x = 5;

var y = 10;

var sum = x + y;

document.write(sum);

## **JavaScript Comments**

A comment is simply a line of text that is completely ignored by the JavaScript interpreter. Comments are usually added with the purpose of providing extra information pertaining to source code. It will not only help you understand your code when you look after a period of time but also others who are working with you on the same project.

JavaScript support single-line as well as multi-line comments. Single-line comments begin with a double forward slash (//), followed by the comment text. Here's an example:

// This is my first JavaScript program

document.write("Hello World!");

Whereas, a multi-line comment begins with a slash and an asterisk (/\*) and ends with an asterisk and slash (\*/). Here's an example of a multi-line comment.

/\* This is my first program

in JavaScript \*/

**4.Introduction to Document and Window Object**

**Introduction to Document and Window Object**

# HTML DOM Documents

## **The Document Object**

When an HTML document is loaded into a web browser, it becomes a **document object**.

The **document object** is the root node of the HTML document.

The **document object** is a property of the **window object**.

The **document object** is accessed with:

window.document or just document

### **Examples**

let url = window.document.URL;

let url = document.URL;

## **Document Object Properties and Methods**

The following properties and methods can be used on HTML documents:

|  |  |
| --- | --- |
| Property / Method | Description |
| [activeElement](https://www.w3schools.com/jsref/prop_document_activeelement.asp) | Returns the currently focused element in the document |
| [addEventListener()](https://www.w3schools.com/jsref/met_document_addeventlistener.asp) | Attaches an event handler to the document |
| [adoptNode()](https://www.w3schools.com/jsref/met_document_adoptnode.asp) | Adopts a node from another document |
| [anchors](https://www.w3schools.com/jsref/coll_doc_anchors.asp) | [Deprecated](https://www.w3schools.com/jsref/coll_doc_anchors.asp) |
| [applets](https://www.w3schools.com/jsref/coll_doc_applets.asp) | [Deprecated](https://www.w3schools.com/jsref/coll_doc_applets.asp) |
| [baseURI](https://www.w3schools.com/jsref/prop_doc_baseuri.asp) | Returns the absolute base URI of a document |
| [body](https://www.w3schools.com/jsref/prop_doc_body.asp) | Sets or returns the document's body (the <body> element) |
| [charset](https://www.w3schools.com/jsref/prop_document_charset.asp) | [Deprecated](https://www.w3schools.com/jsref/prop_document_charset.asp) |
| [characterSet](https://www.w3schools.com/jsref/prop_document_characterset.asp) | Returns the character encoding for the document |
| [close()](https://www.w3schools.com/jsref/met_doc_close.asp) | Closes the output stream previously opened with document.open() |
| [cookie](https://www.w3schools.com/jsref/prop_doc_cookie.asp) | Returns all name/value pairs of cookies in the document |
| [createAttribute()](https://www.w3schools.com/jsref/met_document_createattribute.asp) | Creates an attribute node |
| [createComment()](https://www.w3schools.com/jsref/met_document_createcomment.asp) | Creates a Comment node with the specified text |
| [createDocumentFragment()](https://www.w3schools.com/jsref/met_document_createdocumentfragment.asp) | Creates an empty DocumentFragment node |
| [createElement()](https://www.w3schools.com/jsref/met_document_createelement.asp) | Creates an Element node |
| [createEvent()](https://www.w3schools.com/jsref/event_createevent.asp) | Creates a new event |
| [createTextNode()](https://www.w3schools.com/jsref/met_document_createtextnode.asp) | Creates a Text node |
| [defaultView](https://www.w3schools.com/jsref/prop_document_defaultview.asp) | Returns the window object associated with a document, or null if none is available. |
| [designMode](https://www.w3schools.com/jsref/prop_document_designmode.asp) | Controls whether the entire document should be editable or not. |
| [doctype](https://www.w3schools.com/jsref/prop_document_doctype.asp) | Returns the Document Type Declaration associated with the document |
| [documentElement](https://www.w3schools.com/jsref/prop_document_documentelement.asp) | Returns the Document Element of the document (the <html> element) |
| [documentMode](https://www.w3schools.com/jsref/prop_doc_documentmode.asp) | [Deprecated](https://www.w3schools.com/jsref/prop_doc_documentmode.asp) |
| [documentURI](https://www.w3schools.com/jsref/prop_document_documenturi.asp) | Sets or returns the location of the document |
| [domain](https://www.w3schools.com/jsref/prop_doc_domain.asp) | Returns the domain name of the server that loaded the document |
| [domConfig](https://www.w3schools.com/jsref/prop_document_domconfig.asp) | [Deprecated](https://www.w3schools.com/jsref/prop_document_domconfig.asp) |
| [embeds](https://www.w3schools.com/jsref/coll_doc_embeds.asp) | Returns a collection of all <embed> elements the document |
| [execCommand()](https://www.w3schools.com/jsref/met_document_execcommand.asp) | [Deprecated](https://www.w3schools.com/jsref/met_document_execcommand.asp) |
| [forms](https://www.w3schools.com/jsref/coll_doc_forms.asp) | Returns a collection of all <form> elements in the document |
| [getElementById()](https://www.w3schools.com/jsref/met_document_getelementbyid.asp) | Returns the element that has the ID attribute with the specified value |
| [getElementsByClassName()](https://www.w3schools.com/jsref/met_document_getelementsbyclassname.asp) | Returns an [HTMLCollection](https://www.w3schools.com/jsref/dom_obj_htmlcollection.asp) containing all elements with the specified class name |
| [getElementsByName()](https://www.w3schools.com/jsref/met_doc_getelementsbyname.asp) | Returns an live [NodeList](https://www.w3schools.com/jsref/dom_obj_html_nodelist.asp) containing all elements with the specified name |
| [getElementsByTagName()](https://www.w3schools.com/jsref/met_document_getelementsbytagname.asp) | Returns an [HTMLCollection](https://www.w3schools.com/jsref/dom_obj_htmlcollection.asp) containing all elements with the specified tag name |
| [hasFocus()](https://www.w3schools.com/jsref/met_document_hasfocus.asp) | Returns a Boolean value indicating whether the document has focus |
| [head](https://www.w3schools.com/jsref/prop_doc_head.asp) | Returns the <head> element of the document |
| [images](https://www.w3schools.com/jsref/coll_doc_images.asp) | Returns a collection of all <img> elements in the document |
| [implementation](https://www.w3schools.com/jsref/prop_document_implementation.asp) | Returns the DOMImplementation object that handles this document |
| [importNode()](https://www.w3schools.com/jsref/met_document_importnode.asp) | Imports a node from another document |
| [inputEncoding](https://www.w3schools.com/jsref/prop_document_inputencoding.asp) | [Deprecated](https://www.w3schools.com/jsref/prop_document_inputencoding.asp) |
| [lastModified](https://www.w3schools.com/jsref/prop_doc_lastmodified.asp) | Returns the date and time the document was last modified |
| [links](https://www.w3schools.com/jsref/coll_doc_links.asp) | Returns a collection of all <a> and <area> elements in the document that have a href attribute |
| [normalize()](https://www.w3schools.com/jsref/met_document_normalize.asp) | Removes empty Text nodes, and joins adjacent nodes |
| [normalizeDocument()](https://www.w3schools.com/jsref/met_document_normalizedocument.asp) | [Deprecated](https://www.w3schools.com/jsref/met_document_normalizedocument.asp) |
| [open()](https://www.w3schools.com/jsref/met_doc_open.asp) | Opens an HTML output stream to collect output from document.write() |
| [querySelector()](https://www.w3schools.com/jsref/met_document_queryselector.asp) | Returns the first element that matches a specified CSS selector(s) in the document |
| [querySelectorAll()](https://www.w3schools.com/jsref/met_document_queryselectorall.asp) | Returns a static [NodeList](https://www.w3schools.com/jsref/dom_obj_html_nodelist.asp) containing all elements that matches a specified CSS selector(s) in the document |
| [readyState](https://www.w3schools.com/jsref/prop_doc_readystate.asp) | Returns the (loading) status of the document |
| [referrer](https://www.w3schools.com/jsref/prop_doc_referrer.asp) | Returns the URL of the document that loaded the current document |
| [removeEventListener()](https://www.w3schools.com/jsref/met_document_removeeventlistener.asp) | Removes an event handler from the document (that has been attached with the [addEventListener()](https://www.w3schools.com/jsref/met_document_addeventlistener.asp) method) |
| [renameNode()](https://www.w3schools.com/jsref/met_document_renamenode.asp) | [Deprecated](https://www.w3schools.com/jsref/met_document_renamenode.asp) |
| [scripts](https://www.w3schools.com/jsref/coll_doc_scripts.asp) | Returns a collection of <script> elements in the document |
| [strictErrorChecking](https://www.w3schools.com/jsref/prop_document_stricterrorchecking.asp) | [Deprecated](https://www.w3schools.com/jsref/prop_document_stricterrorchecking.asp) |
| [title](https://www.w3schools.com/jsref/prop_doc_title.asp) | Sets or returns the title of the document |
| [URL](https://www.w3schools.com/jsref/prop_doc_url.asp) | Returns the full URL of the HTML document |
| [write()](https://www.w3schools.com/jsref/met_doc_write.asp) | Writes HTML expressions or JavaScript code to a document |
| [writeln()](https://www.w3schools.com/jsref/met_doc_writeln.asp) | Same as write(), but adds a newline character after each statement |

# The Window Object

## **Window Object**

The window object represents an open window in a browser.

If a document contain frames (<iframe> tags), the browser creates one window object for the HTML document, and one additional window object for each frame.

## **Window Object Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [closed](https://www.w3schools.com/jsref/prop_win_closed.asp) | Returns a boolean true if a window is closed. |
| [console](https://www.w3schools.com/jsref/prop_win_console.asp) | Returns the Console Object for the window. See also [The Console Object](https://www.w3schools.com/jsref/obj_console.asp). |
| [defaultStatus](https://www.w3schools.com/jsref/prop_win_defaultstatus.asp) | Deprecated. |
| [document](https://www.w3schools.com/jsref/prop_win_document.asp) | Returns the Document object for the window. See also [The Document Object](https://www.w3schools.com/jsref/dom_obj_document.asp). |
| [frameElement](https://www.w3schools.com/jsref/prop_win_frameelement.asp) | Returns the frame in which the window runs. |
| [frames](https://www.w3schools.com/jsref/prop_win_frames.asp) | Returns all window objects running in the window. |
| [history](https://www.w3schools.com/jsref/prop_win_history.asp) | Returns the History object for the window. See also [The History Object](https://www.w3schools.com/jsref/obj_history.asp). |
| [innerHeight](https://www.w3schools.com/jsref/prop_win_innerheight.asp) | Returns the height of the window's content area (viewport) including scrollbars |
| [innerWidth](https://www.w3schools.com/jsref/prop_win_innerwidth.asp) | Returns the width of a window's content area (viewport) including scrollbars |
| [length](https://www.w3schools.com/jsref/prop_win_length.asp) | Returns the number of <iframe> elements in the current window |
| [localStorage](https://www.w3schools.com/jsref/prop_win_localstorage.asp) | Allows to save key/value pairs in a web browser. Stores the data with no expiration date |
| [location](https://www.w3schools.com/jsref/prop_win_location.asp) | Returns the Location object for the window. See also the [The Location Object](https://www.w3schools.com/jsref/obj_location.asp). |
| [name](https://www.w3schools.com/jsref/prop_win_name.asp) | Sets or returns the name of a window |
| [navigator](https://www.w3schools.com/jsref/prop_win_navigator.asp) | Returns the Navigator object for the window. See also [The Navigator object](https://www.w3schools.com/jsref/obj_navigator.asp). |
| [opener](https://www.w3schools.com/jsref/prop_win_opener.asp) | Returns a reference to the window that created the window |
| [outerHeight](https://www.w3schools.com/jsref/prop_win_outerheight.asp) | Returns the height of the browser window, including toolbars/scrollbars |
| [outerWidth](https://www.w3schools.com/jsref/prop_win_outerwidth.asp) | Returns the width of the browser window, including toolbars/scrollbars |
| [pageXOffset](https://www.w3schools.com/jsref/prop_win_pagexoffset.asp) | Returns the pixels the current document has been scrolled (horizontally) from the upper left corner of the window |
| [pageYOffset](https://www.w3schools.com/jsref/prop_win_pagexoffset.asp) | Returns the pixels the current document has been scrolled (vertically) from the upper left corner of the window |
| [parent](https://www.w3schools.com/jsref/prop_win_parent.asp) | Returns the parent window of the current window |
| [screen](https://www.w3schools.com/jsref/prop_win_screen.asp) | Returns the Screen object for the window See also [The Screen object](https://www.w3schools.com/jsref/obj_screen.asp) |
| [screenLeft](https://www.w3schools.com/jsref/prop_win_screenleft.asp) | Returns the horizontal coordinate of the window relative to the screen |
| [screenTop](https://www.w3schools.com/jsref/prop_win_screentop.asp) | Returns the vertical coordinate of the window relative to the screen |
| [screenX](https://www.w3schools.com/jsref/prop_win_screenx.asp) | Returns the horizontal coordinate of the window relative to the screen |
| [screenY](https://www.w3schools.com/jsref/prop_win_screeny.asp) | Returns the vertical coordinate of the window relative to the screen |
| [sessionStorage](https://www.w3schools.com/jsref/prop_win_sessionstorage.asp) | Allows to save key/value pairs in a web browser. Stores the data for one session |
| [scrollX](https://www.w3schools.com/jsref/prop_win_scrollx.asp) | An alias of [pageXOffset](https://www.w3schools.com/jsref/prop_win_pagexoffset.asp) |
| [scrollY](https://www.w3schools.com/jsref/prop_win_scrolly.asp) | An alias of [pageYOffset](https://www.w3schools.com/jsref/prop_win_pagexoffset.asp) |
| [self](https://www.w3schools.com/jsref/prop_win_self.asp) | Returns the current window |
| [status](https://www.w3schools.com/jsref/prop_win_status.asp) | Deprecated. Avoid using it. |
| [top](https://www.w3schools.com/jsref/prop_win_top.asp) | Returns the topmost browser window |

## **Window Object Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [addEventListener()](https://www.w3schools.com/jsref/met_win_addeventlistener.asp) | Attaches an event handler to the window |
| [alert()](https://www.w3schools.com/jsref/met_win_alert.asp) | Displays an alert box with a message and an OK button |
| [atob()](https://www.w3schools.com/jsref/met_win_atob.asp) | Decodes a base-64 encoded string |
| [blur()](https://www.w3schools.com/jsref/met_win_blur.asp) | Removes focus from the current window |
| [btoa()](https://www.w3schools.com/jsref/met_win_btoa.asp) | Encodes a string in base-64 |
| [clearInterval()](https://www.w3schools.com/jsref/met_win_clearinterval.asp) | Clears a timer set with setInterval() |
| [clearTimeout()](https://www.w3schools.com/jsref/met_win_cleartimeout.asp) | Clears a timer set with setTimeout() |
| [close()](https://www.w3schools.com/jsref/met_win_close.asp) | Closes the current window |
| [confirm()](https://www.w3schools.com/jsref/met_win_confirm.asp) | Displays a dialog box with a message and an OK and a Cancel button |
| [focus()](https://www.w3schools.com/jsref/met_win_focus.asp) | Sets focus to the current window |
| [getComputedStyle()](https://www.w3schools.com/jsref/jsref_getcomputedstyle.asp) | Gets the current computed CSS styles applied to an element |
| getSelection() | Returns a Selection object representing the range of text selected by the user |
| [matchMedia()](https://www.w3schools.com/jsref/met_win_matchmedia.asp) | Returns a MediaQueryList object representing the specified CSS media query string |
| [moveBy()](https://www.w3schools.com/jsref/met_win_moveby.asp) | Moves a window relative to its current position |
| [moveTo()](https://www.w3schools.com/jsref/met_win_moveto.asp) | Moves a window to the specified position |
| [open()](https://www.w3schools.com/jsref/met_win_open.asp) | Opens a new browser window |
| [print()](https://www.w3schools.com/jsref/met_win_print.asp) | Prints the content of the current window |
| [prompt()](https://www.w3schools.com/jsref/met_win_prompt.asp) | Displays a dialog box that prompts the visitor for input |
| [removeEventListener()](https://www.w3schools.com/jsref/met_win_removeeventlistener.asp) | Removes an event handler from the window |
| requestAnimationFrame() | Requests the browser to call a function to update an animation before the next repaint |
| [resizeBy()](https://www.w3schools.com/jsref/met_win_resizeby.asp) | Resizes the window by the specified pixels |
| [resizeTo()](https://www.w3schools.com/jsref/met_win_resizeto.asp) | Resizes the window to the specified width and height |
| scroll() | Deprecated. This method has been replaced by the [scrollTo()](https://www.w3schools.com/jsref/met_win_scrollto.asp) method. |
| [scrollBy()](https://www.w3schools.com/jsref/met_win_scrollby.asp) | Scrolls the document by the specified number of pixels |
| [scrollTo()](https://www.w3schools.com/jsref/met_win_scrollto.asp) | Scrolls the document to the specified coordinates |
| [setInterval()](https://www.w3schools.com/jsref/met_win_setinterval.asp) | Calls a function or evaluates an expression at specified intervals (in milliseconds) |
| [setTimeout()](https://www.w3schools.com/jsref/met_win_settimeout.asp) | Calls a function or evaluates an expression after a specified number of milliseconds |
| [stop()](https://www.w3schools.com/jsref/met_win_stop.asp) | Stops the window from loading |

# Window Console Object

## **The Console Object**

The **console object** provides access to the browser's debugging console.

The **console object** is a property of the **window object**.

The **console object** is accessed with:

window.console or just console

### **Examples**

window.console.error("You made a mistake");

console.error("You made a mistake");

## **Console Object Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [assert()](https://www.w3schools.com/jsref/met_console_assert.asp) | Writes an error message to the console if a assertion is false |
| [clear()](https://www.w3schools.com/jsref/met_console_clear.asp) | Clears the console |
| [count()](https://www.w3schools.com/jsref/met_console_count.asp) | Logs the number of times that this particular call to count() has been called |
| [error()](https://www.w3schools.com/jsref/met_console_error.asp) | Outputs an error message to the console |
| [group()](https://www.w3schools.com/jsref/met_console_group.asp) | Creates a new inline group in the console. This indents following console messages by an additional level, until console.groupEnd() is called |
| [groupCollapsed()](https://www.w3schools.com/jsref/met_console_groupcollapsed.asp) | Creates a new inline group in the console. However, the new group is created collapsed. The user will need to use the disclosure button to expand it |
| [groupEnd()](https://www.w3schools.com/jsref/met_console_groupend.asp) | Exits the current inline group in the console |
| [info()](https://www.w3schools.com/jsref/met_console_info.asp) | Outputs an informational message to the console |
| [log()](https://www.w3schools.com/jsref/met_console_log.asp) | Outputs a message to the console |
| [table()](https://www.w3schools.com/jsref/met_console_table.asp) | Displays tabular data as a table |
| [time()](https://www.w3schools.com/jsref/met_console_time.asp) | Starts a timer (can track how long an operation takes) |
| [timeEnd()](https://www.w3schools.com/jsref/met_console_timeend.asp) | Stops a timer that was previously started by console.time() |
| [trace()](https://www.w3schools.com/jsref/met_console_trace.asp) | Outputs a stack trace to the console |
| [warn()](https://www.w3schools.com/jsref/met_console_warn.asp) | Outputs a warning message to the console |

# Window History

## **The Window History Object**

The **history object** contains the URLs visited by the user (in the browser window).

The **history object** is a property of the **window object**.

The **history object** is accessed with:

window.history or just history:

### **Examples**

let length = window.history.length;

let length = history.length;

## **History Object Properties and Methods**

|  |  |
| --- | --- |
| **Property/Method** | **Description** |
| [back()](https://www.w3schools.com/jsref/met_his_back.asp) | Loads the previous URL (page) in the history list |
| [forward()](https://www.w3schools.com/jsref/met_his_forward.asp) | Loads the next URL (page) in the history list |
| [go()](https://www.w3schools.com/jsref/met_his_go.asp) | Loads a specific URL (page) from the history list |
| [length](https://www.w3schools.com/jsref/prop_his_length.asp) | Returns the number of URLs (pages) in the history list |

# 5.Variables and Operators

# JavaScript Variables

4 Ways to Declare a JavaScript Variable:

1.Using var, 2.Using let, 3.Using const, 4.Using nothing

## **What are Variables?**

Variables are containers for storing data (storing data values).

In this example, x, y, and z, are variables, declared with the var keyword:

### **Example**

var x = 5;

var y = 6;

var z = x + y;

In this example, x, y, and z, are variables, declared with the let keyword:

### **Example**

let x = 5;

let y = 6;

let z = x + y;

In this example, x, y, and z, are undeclared variables:

### **Example**

x = 5;

y = 6;

z = x + y;

From all the examples above, you can guess:

* x stores the value 5
* y stores the value 6
* z stores the value 11

**When to Use JavaScript var?**

Always declare JavaScript variables with **var, let**, or **const**.

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

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

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

## **When to Use JavaScript const?**

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

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

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

### **Example**

const price1 = 5;

const price2 = 6;

let total = price1 + price2;

The two variables price1 and price2 are declared with the const keyword.

These are constant values and cannot be changed.

The variable total is declared with the let keyword.

This is a value that can be changed.

## **Just Like Algebra**

Just like in algebra, variables hold values:

let x = 5;

let y = 6;

Just like in algebra, variables are used in expressions:

let z = x + y;

From the example above, you can guess that the total is calculated to be 11.

## **Note**

Variables are containers for storing values.

## **JavaScript Identifiers**

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

These unique names are called **identifiers**.

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

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

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

## **Note**

JavaScript identifiers are case-sensitive.

## **The Assignment Operator**

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.

This is different from algebra. The following does not make sense in algebra:

x = x + 5

In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x.

(It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

## **Note**

The "equal to" operator is written like == in JavaScript.

**6.Data Types and Num Type Conversion**

## **JavaScript Data Types**

JavaScript variables can hold numbers like 100 and text values like "John Doe".

In programming, text values are called text strings.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes.

If you put a number in quotes, it will be treated as a text string.

### **Example**

const pi = 3.14;

let person = "John Doe";

let answer = 'Yes I am!';

## **Declaring a JavaScript Variable**

Creating a variable in JavaScript is called "declaring" a variable.

You declare a JavaScript variable with the **var**or the **let** keyword:

var carName;

**or**

let carName;

After the declaration, the variable has no value (technically it is undefined).

To **assign** a value to the variable, use the equal sign:

carName = "Volvo";

You can also assign a value to the variable when you declare it:

let carName = "Volvo";

In the example below, we create a variable called carName and assign the value "Volvo" to it.

Then we "output" the value inside an HTML paragraph with id="demo":

### **Example**

<p id="demo"></p>

<script>

let carName = "Volvo";

document.getElementById("demo").innerHTML = carName;

</script>

## **Note**

It's a good programming practice to declare all variables at the beginning of a script.

## **One Statement, Many Variables**

You can declare many variables in one statement.

Start the statement with let and separate the variables by **comma**:

### **Example**

let person = "John Doe", carName = "Volvo", price = 200;

A declaration can span multiple lines:

### **Example**

let person = "John Doe",

carName = "Volvo",

price = 200;

## **Value = undefined**

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

A variable declared without a value will have the value undefined.

The variable carName will have the value undefined after the execution of this statement:

### **Example**

let carName;

## **Re-Declaring JavaScript Variables**

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

The variable carName will still have the value "Volvo" after the execution of these statements:

### **Example**

var carName = "Volvo";

var carName;

## **Note**

You cannot re-declare a variable declared with let or const.

This will not work:

let carName = "Volvo";

let carName;

**6.Data Types and Num Type Conversion**

In programming, type conversion is the process of converting data of one type to another. For example: converting String data to Number.

There are two types of type conversion in JavaScript.

Implicit Conversion - automatic type conversion

Explicit Conversion - manual type conversion

## JavaScript Implicit Conversion

In certain situations, JavaScript automatically converts one data type to another (to the right type). This is known as implicit conversion.

### **Example 1: Implicit Conversion to String**

// numeric string used with + gives string type

let result;

result = '3' + 2;

console.log(result) // "32"

result = '3' + true;

console.log(result); // "3true"

result = '3' + undefined;

console.log(result); // "3undefined"

result = '3' + null;

console.log(result); // "3null"

[Run Code](https://www.programiz.com/javascript/online-compiler)

Note: When a number is added to a string, JavaScript converts the number to a string before concatenation.

**Example 2: Implicit Conversion to Number**

// numeric string used with - , / , \* results number type

let result;

result = '4' - '2';

console.log(result); // 2

result = '4' - 2;

console.log(result); // 2

result = '4' \* 2;

console.log(result); // 8

result = '4' / 2;

console.log(result); // 2

[Run Code](https://www.programiz.com/javascript/online-compiler)

**Example 3: Non-numeric String Results to NaN**

// non-numeric string used with - , / , \* results to NaN

let result;

result = 'hello' - 'world';

console.log(result); // NaN

result = '4' - 'hello';

console.log(result); // NaN

[Run Code](https://www.programiz.com/javascript/online-compiler)

**Example 4: Implicit Boolean Conversion to Number**

// if boolean is used, true is 1, false is 0

let result;

result = '4' - true;

console.log(result); // 3

result = 4 + true;

console.log(result); // 5

result = 4 + false;

console.log(result); // 4

[Run Code](https://www.programiz.com/javascript/online-compiler)

Note: JavaScript considers 0 as false and all non-zero number as true. And, if true is converted to a number, the result is always 1.

**Example 5: null Conversion to Number**

// null is 0 when used with number

let result;

result = 4 + null;

console.log(result); // 4

result = 4 - null;

console.log(result); // 4

[Run Code](https://www.programiz.com/javascript/online-compiler)

**Example 6: undefined used with number, boolean or null**

// Arithmetic operation of undefined with number, boolean or null gives NaN

let result;

result = 4 + undefined;

console.log(result); // NaN

result = 4 - undefined;

console.log(result); // NaN

result = true + undefined;

console.log(result); // NaN

result = null + undefined;

console.log(result); // NaN

[Run Code](https://www.programiz.com/javascript/online-compiler)

**JavaScript Explicit Conversion**

You can also convert one data type to another as per your needs. The type conversion that you do manually is known as explicit type conversion.

In JavaScript, explicit type conversions are done using built-in methods.

Here are some common methods of explicit conversions.

1. Convert to Number Explicitly

To convert numeric strings and boolean values to numbers, you can use Number(). For example,

let result;

// string to number

result = Number('324');

console.log(result); // 324

result = Number('324e-1')

console.log(result); // 32.4

// boolean to number

result = Number(true);

console.log(result); // 1

result = Number(false);

console.log(result); // 0

[Run Code](https://www.programiz.com/javascript/online-compiler)

In JavaScript, empty strings and null values return 0. For example,

let result;

result = Number(null);

console.log(result); // 0

let result = Number(' ')

console.log(result); // 0

**If a string is an invalid number, the result will be NaN. For example,**

let result;

result = Number('hello');

console.log(result); // NaN

result = Number(undefined);

console.log(result); // NaN

result = Number(NaN);

console.log(result); // NaN

Note: You can also generate numbers from strings using parseInt(), parseFloat(), unary operator + and Math.floor(). For example,

let result;

result = parseInt('20.01');

console.log(result); // 20

result = parseFloat('20.01');

console.log(result); // 20.01

result = +'20.01';

console.log(result); // 20.01

result = Math.floor('20.01');

console.log(result); // 20

**2. Convert to String Explicitly**

To convert other data types to strings, you can use either String() or toString(). For example,

//number to string

let result;

result = String(324);

console.log(result); // "324"

result = String(2 + 4);

console.log(result); // "6"

//other data types to string

result = String(null);

console.log(result); // "null"

result = String(undefined);

console.log(result); // "undefined"

result = String(NaN);

console.log(result); // "NaN"

result = String(true);

console.log(result); // "true"

result = String(false);

console.log(result); // "false"

// using toString()

result = (324).toString();

console.log(result); // "324"

result = true.toString();

console.log(result); // "true"

Note: String() takes null and undefined and converts them to string. However, toString() gives error when null are passed.

**3. Convert to Boolean Explicitly**

To convert other data types to a boolean, you can use Boolean().

In JavaScript, undefined, null, 0, NaN, '' converts to false. For example,

let result;

result = Boolean('');

console.log(result); // false

result = Boolean(0);

console.log(result); // false

result = Boolean(undefined);

console.log(result); // false

result = Boolean(null);

console.log(result); // false

result = Boolean(NaN);

console.log(result); // false

All other values give true. For example,

result = Boolean(324);

console.log(result); // true

result = Boolean('hello');

console.log(result); // true

result = Boolean(' ');

console.log(result); // true

[Run Code](https://www.programiz.com/javascript/online-compiler)

### JavaScript Type Conversion Table

The table shows the conversion of different values to String, Number, and Boolean in JavaScript.

|  |  |  |  |
| --- | --- | --- | --- |
| Value | String Conversion | Number Conversion | Boolean Conversion |
| 1 | "1" | 1 | true |
| 0 | "0" | 0 | false |
| "1" | "1" | 1 | true |
| "0" | "0" | 0 | true |
| "ten" | "ten" | NaN | true |
| true | "true" | 1 | true |
| false | "false" | 0 | false |
| null | "null" | 0 | false |
| undefined | "undefined" | NaN | false |
| '' | "" | 0 | false |
| ' ' | " " | 0 | true |

**Converting Values to Numbers:** We can use [**Number() function**](https://www.geeksforgeeks.org/javascript-number-function/)in JavaScript to convert a value to a Number. It can convert any numerical text and boolean value to a Number. In the case of strings of non-numbers, it will convert it to a [**NaN**(Not a Number)](https://www.geeksforgeeks.org/javascript-number-nan-property/)

**Syntax:**

Number(valueToConvert)

**Example:**

**Input:**

var s = "144";

var n = Number(s);

**Output:**

now n contain 144(Number).

**Example 3:** Below code converts a numerical text, dates and boolean values to a number.

* javascript

|  |
| --- |
| // Number and date has been assigned  // to variable v and d respectively  var v = "144";  var d = new Date('1995-12-17T03:24:00');   // Conversion of string to number  console.log(" Number(v) = " + Number(v));   //Conversion of boolean value to number  console.log(" Number(false) = " + Number(false));  console.log(" Number(true) = " + Number(true));   // Conversion of date to number  console.log(" Number(d) = " + Number(d)); |

**Output:**

Number(v) = 144

Number(false) = 0

Number(true) = 1

Number(d) = 819150840000

**Example 4:** If the string is non-number then it converts it to **NaN** and strings of white spaces or empty strings will convert to 0.

* javascript

|  |
| --- |
| // Empty string assigned  var v = "";    // White space assigned  var d = " ";    // Non-number string assigned  var s = "GeeksforGeeks";    // Printing converted values of number  console.log(" Number(v) = " + Number(v));  console.log(" Number(d) = " + Number(d));  console.log(" Number(s) = " + Number(s)); |

**Output:**

Number(v) = 0

Number(d) = 0

Number(s) = NaN

## **JavaScript Type Conversion**

JavaScript variables can be converted to a new variable and another data type:

* By the use of a JavaScript function
* **Automatically** by JavaScript itself

## **Converting Strings to Numbers**

The global method Number() converts a variable (or a value) into a number.

A numeric string (like "3.14") converts to a number (like 3.14).

An empty string (like "") converts to 0.

A non numeric string (like "John") converts to NaN (Not a Number).

### **Examples**

**These will convert:**

Number("3.14")  
Number(Math.PI)  
Number(" ")  
Number("")  
**These will not convert:**

Number("99 88")  
Number("John")

## **Number Methods**

In the chapter [Number Methods](https://www.w3schools.com/js/js_number_methods.asp), you will find more methods that can be used to convert strings to numbers:

|  |  |
| --- | --- |
| **Method** | **Description** |
| Number() | Returns a number, converted from its argument |
|  |  |
| parseFloat() | Parses a string and returns a floating point number |
| parseInt() | Parses a string and returns an integer |

## **The Unary + Operator**

The **unary + operator** can be used to convert a variable to a number:

### **Example**

let y = "5";

// y is a string

let x = + y;

// x is a number

If the variable cannot be converted, it will still become a number, but with the value NaN (Not a Number):

### **Example**

let y = "John";   // y is a string  
let x = + y;      // x is a number (NaN)

## **Converting Numbers to Strings**

The global method String() can convert numbers to strings.

It can be used on any type of numbers, literals, variables, or expressions:

### **Example**

String(x)         // returns a string from a number variable x

String(123)       // returns a string from a number literal 123

String(100 + 23)  // returns a string from a number from an expression

The Number method toString() does the same.

### **Example**

x.toString()  
(123).toString()  
(100 + 23).toString()

## **More Methods**

In the chapter [Number Methods](https://www.w3schools.com/js/js_number_methods.asp), you will find more methods that can be used to convert numbers to strings:

|  |  |
| --- | --- |
| **Method** | **Description** |
| toExponential() | Returns a string, with a number rounded and written using exponential notation. |
| toFixed() | Returns a string, with a number rounded and written with a specified number of decimals. |
| toPrecision() | Returns a string, with a number written with a specified length |

## **Converting Dates to Numbers**

The global method Number() can be used to convert dates to numbers.

d = new Date();

Number(d)          // returns 1404568027739

The date method getTime() does the same.

d = new Date();

d.getTime()        // returns 1404568027739

## **Converting Dates to Strings**

The global method String() can convert dates to strings.

String(Date())  // returns "Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)"

The Date method toString() does the same.

### **Example**

Date().toString()  // returns "Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)"

In the chapter [Date Methods](https://www.w3schools.com/js/js_date_methods.asp), you will find more methods that can be used to convert dates to strings:

|  |  |
| --- | --- |
| **Method** | **Description** |
| getDate() | Get the day as a number (1-31) |
| getDay() | Get the weekday a number (0-6) |
| getFullYear() | Get the four digit year (yyyy) |
| getHours() | Get the hour (0-23) |
| getMilliseconds() | Get the milliseconds (0-999) |
| getMinutes() | Get the minutes (0-59) |
| getMonth() | Get the month (0-11) |
| getSeconds() | Get the seconds (0-59) |
| getTime() | Get the time (milliseconds since January 1, 1970) |

## **Converting Booleans to Numbers**

The global method Number() can also convert booleans to numbers.

Number(false)     // returns 0

Number(true)      // returns

## **Converting Booleans to Strings**

The global method String() can convert booleans to strings.

String(false)      // returns "false"

String(true)       // returns "true"

The Boolean method toString() does the same.

false.toString()   // returns "false"

true.toString()    // returns "true"

## **Automatic Type Conversion**

When JavaScript tries to operate on a "wrong" data type, it will try to convert the value to a "right" type.

The result is not always what you expect:

5 + null    // returns 5         because null is converted to 0

"5" + null  // returns "5null"   because null is converted to "null"

"5" + 2     // returns "52"      because 2 is converted to "2"

"5" - 2     // returns 3         because "5" is converted to 5

"5" \* "2"   // returns 10        because "5" and "2" are converted to 5 and 2

## **Automatic String Conversion**

JavaScript automatically calls the variable's toString() function when you try to "output" an object or a variable:

document.getElementById("demo").innerHTML = myVar;

// if myVar = {name:"Fjohn"}  // toString converts to "[object Object]"

// if myVar = [1,2,3,4]       // toString converts to "1,2,3,4"

// if myVar = new Date()      // toString converts to "Fri Jul 18 2014 09:08:55 GMT+0200"

Numbers and booleans are also converted, but this is not very visible:

// if myVar = 123             // toString converts to "123"

// if myVar = true            // toString converts to "true"

// if myVar = false           // toString converts to "false"

## **JavaScript Type Conversion Table**

This table shows the result of converting different JavaScript values to Number, String, and Boolean:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Original Value** | **Converted to Number** | **Converted to String** | **Converted to Boolean** | **Try it** |
| False | 0 | "false" | false | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_false) |
| True | 1 | "true" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_true) |
| 0 | 0 | "0" | false | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_number_0) |
| 1 | 1 | "1" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_number_1) |
| "0" | 0 | "0" | **true** | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_string_0) |
| "000" | 0 | "000" | **true** | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_string_000) |
| "1" | 1 | "1" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_string_1) |
| NaN | NaN | "NaN" | false | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_nan) |
| Infinity | Infinity | "Infinity" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_infinity) |
| -Infinity | -Infinity | "-Infinity" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_infinity_minus) |
| "" | **0** | "" | **false** | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_string_empty) |
| "20" | 20 | "20" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_string_number) |
| "twenty" | NaN | "twenty" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_string_text) |
| [ ] | **0** | "" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_array_empty) |
| [20] | **20** | "20" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_array_one_number) |
| [10,20] | NaN | "10,20" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_array_two_numbers) |
| ["twenty"] | NaN | "twenty" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_array_one_string) |
| ["ten","twenty"] | NaN | "ten,twenty" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_array_two_strings) |
| function(){} | NaN | "function(){}" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_function) |
| { } | NaN | "[object Object]" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_object) |
| Null | **0** | "null" | false | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_null) |
| Undefined | NaN | "undefined" | false | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_undefined) |

Values in quotes indicate string values.

**Red values** indicate values (some) programmers might not expect.

**Objects and Arrays**

# Both objects and arrays are considered “special” in JavaScript. Objects represent a special data type that is [mutable](https://gomakethings.com/immutability-in-javascript/) and can be used to store a collection of data (rather than just a single value). Arrays are a special type of variable that is also mutable and can also be used to store a list of values.

# 8.Objects

**When to Use Objects**

Objects are used to represent a “thing” in your code. That could be a person, a car, a building, a book, a character in a game — basically anything that is made up or can be defined by a set of characteristics. In objects, these characteristics are called **properties** that consist of a key and a value.

// Basic object syntax

var object = {

key: 'value'

};//

Example 'person' object

var person = {

name: 'Zac',

age: 33,

likesCoding: true

};

**Access, Add, and Remove Items from Objects**

Properties in objects can be accessed, added, changed, and removed by using either **dot** or **bracket** notation. To get the value of the age key in our personobject with both dot and bracket notation, we’d write:

// Dot notation

person.age // returns 33// Bracket notation

person['age'] // returns 33

Say we wanted to change the value of likesCoding to **false**. We can do that with dot notation like this:

person.likesCoding = false;

And if we wanted to add a new property to our person object, we could accomplish that with dot notation as well:

person.hobbies = ['hiking', 'travel', 'reading'];

Finally, to remove a property from an object, we use the delete keyword like so:

delete person.age;

[Check out this post](https://codeburst.io/javascript-quickie-dot-notation-vs-bracket-notation-333641c0f781) on the difference between dot and bracket notation, and when to use each.

**Iterating Through Objects**

The most common way to loop through properties in an object is with a **for…in**loop:

for (var key in myObject) {

console.log(key); // logs keys in myObject

console.log(myObject[key]); // logs values in myObject

}

Another way to iterate through object properties is by appending the **forEach()**method to **Object.keys()**:

Object.keys(myObject).forEach(function(key) {

console.log(key); // logs keys in myObject

console.log(myObject[key]); // logs values in myObject

});

# 8.Arrays

**When to Use Arrays**

We use arrays whenever we want to create and store a list of multiple items in a single variable. Arrays are especially useful when creating **ordered collections** where items in the collection can be accessed by their numerical position in the list. Just as object properties can store values of any [primitive data type](https://javascript.info/types) (as well as an array or another object), so too can arrays consist of strings, numbers, booleans, objects, or even other arrays.

**Access, Add, and Remove Items from Arrays**

Arrays use [zero-based indexing](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#Accessing_array_elements), so the first item in an array has an index of 0, the second item an index of 1, and so on. For instance, let’s say we wanted to access the third item (‘Mexico City’) in the following array:

var vacationSpots = ['Tokyo', 'Bali', 'Mexico City', 'Vancouver'];

To do so, we’d write:

vacationSpots[2]; // returns 'Mexico City'

Items can be added and removed from the beginning or end of an array using the push(), pop(), unshift(), and shift() methods:

// push() - Adds item(s) to the end of an array  
vacationSpots.push('Miami');// pop() - Removes the last item from an array  
vacationSpots.pop();// unshift() - Adds item(s) to the beginning of an array  
vacationSpots.unshift('Cape Town', 'Moscow');// shift() - Removes the first item from an array  
vacationSpots.shift();

**Iterating Through Arrays**

We can loop through items in an array in a few ways. First there’s the standard **for** loop:

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

console.log(myArray[i]); // logs items in myArray

}

There’s also the **for…of** loop:

for (var item of myArray) {

console.log(item); // logs items in myArray

}

Or we can utilize the **forEach()** method:

myArray.forEach(item) {

console.log(item); // logs items in myArray

});

# Date and time

Let’s meet a new built-in object: [Date](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date). It stores the date, time and provides methods for date/time management.

For instance, we can use it to store creation/modification times, to measure time, or just to print out the current date.

## [Creation](https://javascript.info/date" \l "creation)

To create a new Date object call new Date() with one of the following arguments:

**new Date()**

Without arguments – create a Date object for the current date and time:

let now = new Date();

alert( now ); // shows current date/time

**new Date(milliseconds)**

Create a Date object with the time equal to number of milliseconds (1/1000 of a second) passed after the Jan 1st of 1970 UTC+0.

// 0 means 01.01.1970 UTC+0

let Jan01\_1970 = new Date(0);

alert( Jan01\_1970 );

// now add 24 hours, get 02.01.1970 UTC+0

let Jan02\_1970 = new Date(24 \* 3600 \* 1000);

alert( Jan02\_1970 );

An integer number representing the number of milliseconds that has passed since the beginning of 1970 is called a timestamp.

It’s a lightweight numeric representation of a date. We can always create a date from a timestamp using new Date(timestamp) and convert the existing Date object to a timestamp using the date.getTime() method (see below).

Dates before 01.01.1970 have negative timestamps, e.g.:

// 31 Dec 1969

let Dec31\_1969 = new Date(-24 \* 3600 \* 1000);

alert( Dec31\_1969 );

**new Date(datestring)**

If there is a single argument, and it’s a string, then it is parsed automatically. The algorithm is the same as Date.parse uses, we’ll cover it later.

let date = new Date("2017-01-26");

alert(date);

// The time is not set, so it's assumed to be midnight GMT and

// is adjusted according to the timezone the code is run in

// So the result could be

// Thu Jan 26 2017 11:00:00 GMT+1100 (Australian Eastern Daylight Time)

// or

// Wed Jan 25 2017 16:00:00 GMT-0800 (Pacific Standard Time)

**new Date(year, month, date, hours, minutes, seconds, ms)**

Create the date with the given components in the local time zone. Only the first two arguments are obligatory.

* The year should have 4 digits. For compatibility, 2 digits are also accepted and considered 19xx, e.g. 98 is the same as 1998 here, but always using 4 digits is strongly encouraged.
* The month count starts with 0 (Jan), up to 11 (Dec).
* The date parameter is actually the day of month, if absent then 1 is assumed.
* If hours/minutes/seconds/ms is absent, they are assumed to be equal 0.

For instance:

new Date(2011, 0, 1, 0, 0, 0, 0); // 1 Jan 2011, 00:00:00

new Date(2011, 0, 1); // the same, hours etc are 0 by default

The maximal precision is 1 ms (1/1000 sec):

let date = new Date(2011, 0, 1, 2, 3, 4, 567);

alert( date ); // 1.01.2011, 02:03:04.567

## [Access date components](https://javascript.info/date" \l "access-date-components)

There are methods to access the year, month and so on from the Date object:

[**getFullYear()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/getFullYear)

Get the year (4 digits)

[**getMonth()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/getMonth)

Get the month, **from 0 to 11**.

[**getDate()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/getDate)

Get the day of month, from 1 to 31, the name of the method does look a little bit strange.

[**getHours()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/getHours)**,**[**getMinutes()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/getMinutes)**,**[**getSeconds()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/getSeconds)**,**[**getMilliseconds()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/getMilliseconds)

Get the corresponding time components.

**Not getYear(), but getFullYear()**

Many JavaScript engines implement a non-standard method getYear(). This method is deprecated. It returns 2-digit year sometimes. Please never use it. There is getFullYear() for the year.

Additionally, we can get a day of week:

[**getDay()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/getDay)

Get the day of week, from 0 (Sunday) to 6 (Saturday). The first day is always Sunday, in some countries that’s not so, but can’t be changed.

**All the methods above return the components relative to the local time zone.**

There are also their UTC-counterparts, that return day, month, year and so on for the time zone UTC+0: [getUTCFullYear()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/getUTCFullYear), [getUTCMonth()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/getUTCMonth), [getUTCDay()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/getUTCDay). Just insert the "UTC" right after "get".

If your local time zone is shifted relative to UTC, then the code below shows different hours:

// current date

let date = new Date();

// the hour in your current time zone

alert( date.getHours() );

// the hour in UTC+0 time zone (London time without daylight savings)

alert( date.getUTCHours() );

Besides the given methods, there are two special ones that do not have a UTC-variant:

[**getTime()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/getTime)

Returns the timestamp for the date – a number of milliseconds passed from the January 1st of 1970 UTC+0.

[**getTimezoneOffset()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/getTimezoneOffset)

Returns the difference between UTC and the local time zone, in minutes:

// if you are in timezone UTC-1, outputs 60

// if you are in timezone UTC+3, outputs -180

alert( new Date().getTimezoneOffset() );

## [Setting date components](https://javascript.info/date" \l "setting-date-components)

The following methods allow to set date/time components:

* [setFullYear(year, [month], [date])](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/setFullYear)
* [setMonth(month, [date])](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/setMonth)
* [setDate(date)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/setDate)
* [setHours(hour, [min], [sec], [ms])](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/setHours)
* [setMinutes(min, [sec], [ms])](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/setMinutes)
* [setSeconds(sec, [ms])](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/setSeconds)
* [setMilliseconds(ms)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/setMilliseconds)
* [setTime(milliseconds)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/setTime) (sets the whole date by milliseconds since 01.01.1970 UTC)

Every one of them except setTime() has a UTC-variant, for instance: setUTCHours().

As we can see, some methods can set multiple components at once, for example setHours. The components that are not mentioned are not modified.

For instance:

let today = new Date();

today.setHours(0);

alert(today); // still today, but the hour is changed to 0

today.setHours(0, 0, 0, 0);

alert(today); // still today, now 00:00:00 sharp.

## [Autocorrection](https://javascript.info/date" \l "autocorrection)

The autocorrection is a very handy feature of Date objects. We can set out-of-range values, and it will auto-adjust itself.

**For instance:**

let date = new Date(2013, 0, 32); // 32 Jan 2013 ?!?

alert(date); // ...is 1st Feb 2013!

Out-of-range date components are distributed automatically.

Let’s say we need to increase the date “28 Feb 2016” by 2 days. It may be “2 Mar” or “1 Mar” in case of a leap-year. We don’t need to think about it. Just add 2 days. The Date object will do the rest:

let date = new Date(2016, 1, 28);

date.setDate(date.getDate() + 2);

alert( date ); // 1 Mar 2016

That feature is often used to get the date after the given period of time. For instance, let’s get the date for “70 seconds after now”:

let date = new Date();

date.setSeconds(date.getSeconds() + 70);

alert( date ); // shows the correct date

We can also set zero or even negative values. For example:

let date = new Date(2016, 0, 2); // 2 Jan 2016

date.setDate(1); // set day 1 of month

alert( date );

date.setDate(0); // min day is 1, so the last day of the previous month is assumed

alert( date ); // 31 Dec 2015

## [Date to number, date diff](https://javascript.info/date" \l "date-to-number-date-diff)

When a Date object is converted to number, it becomes the timestamp same as date.getTime():

let date = new Date();

alert(+date); // the number of milliseconds, same as date.getTime()

The important side effect: dates can be subtracted, the result is their difference in ms.

That can be used for time measurements:

let start = new Date(); // start measuring time

// do the job

for (let i = 0; i < 100000; i++) {

let doSomething = i \* i \* i;

}

let end = new Date(); // end measuring time

alert( `The loop took ${end - start} ms` );

## [Date.now()](https://javascript.info/date" \l "date-now)

If we only want to measure time, we don’t need the Date object.

There’s a special method Date.now() that returns the current timestamp.

It is semantically equivalent to new Date().getTime(), but it doesn’t create an intermediate Date object. So it’s faster and doesn’t put pressure on garbage collection.

It is used mostly for convenience or when performance matters, like in games in JavaScript or other specialized applications.

So this is probably better:

let start = Date.now(); // milliseconds count from 1 Jan 1970

// do the job

for (let i = 0; i < 100000; i++) {

let doSomething = i \* i \* i;

}

let end = Date.now(); // done

alert( `The loop took ${end - start} ms` ); // subtract numbers, not dates

# Conditional Statements

CONDITIONALS

Conditional statements control behavior in JavaScript and determine whether or not pieces of code can run.

There are multiple different types of conditionals in JavaScript including:

“If” statements: where if a condition is true it is used to specify execution for a block of code.

“Else” statements: where if the same condition is false it specifies the execution for a block of code.

“Else if” statements: this specifies a new test if the first condition is false.

Now that you have the basic JavaScript conditional statement definitions, let’s show you examples of each.

If Statement Example

As the most common type of conditional, the if statement only runs if the condition enclosed in parentheses () is [truthy](https://developer.mozilla.org/en-US/docs/Glossary/Truthy).

|  |
| --- |
| EXAMPLE |
| if (10 > 5) {       var outcome = "if block"; } ​outcome; |
| OUTPUT |
| "if block" |

Here’s what’s happening in the example above:

The keyword if tells JavaScript to start the conditional statement.

(10 > 5) is the condition to test, which in this case is true — 10 is greater than 5.

The part contained inside curly braces {} is the block of code to run.

Because the condition passes, the variable outcome is assigned the value "if block".

Else Statement Example

You can extend an if statement with an else statement, which adds another block to run when the if conditional doesn’t pass.

|  |
| --- |
| EXAMPLE |
| if ("cat" === "dog") {       var outcome = "if block"; } else {       var outcome = "else block"; } outcome; |
| OUTPUT |
| "else block" |

In the example above, "cat" and "dog" are not equal, so the else block runs and the variable outcome gets the value "else block".

Else If Statement Example

You can also extend an if statement with an else if statement, which adds another conditional with its own block.

|  |
| --- |
| EXAMPLE |
| if (false) {       var outcome = "if block"; } else if (true) {       var outcome = "else if block"; } else {       var outcome = "else block"; } outcome; |
| OUTPUT |
| "else if block" |

You can use multiple if else conditionals, but note that only the first else if block runs. JavaScript skips any remaining conditionals after it runs the first one that passes.

|  |
| --- |
| EXAMPLE |
| if (false) {       var outcome = "if block"; } else if (true) {       var outcome = "first else if block"; } else if (true) {       var outcome = "second else if block"; } else {       var outcome = "else block"; } ​outcome; |
| OUTPUT |
| "first else if block" |

An else if statement doesn’t need a following else statement to work. If none of the if or else if conditions pass, then JavaScript moves forward and doesn’t run any of the conditional blocks of code.

|  |
| --- |
| EXAMPLE |
| if (false) {       var outcome = "if block"; } else if (false) {       var outcome = "else if block"; } ​outcome; |
| OUTPUT |
| "first else if block" |

**ARRAYS**

Array are container-like values that can hold other values. The values inside an array are called elements.

|  |
| --- |
| EXAMPLE |
| var breakfast = ["coffee", "croissant"];  breakfast; |
| **OUTPUT** |
| ["coffee", "croissant"] |

Array elements don’t all have to be the same type of value. Elements can be any kind of JavaScript value — even other arrays.

|  |
| --- |
| **EXAMPLE** |
| var hodgepodge = [100, "paint", [200, "brush"], false];  hodgepodge; |
| **OUTPUT** |
| [100, "paint", [200, "brush"], false] |

Accessing Elements

To access one of the elements inside an array, you’ll need to use the brackets and a number like this: myArray[3]. JavaScript arrays begin at 0, so the first element will always be inside [0].

|  |
| --- |
| **EXAMPLE** |
| var sisters = ["Tia", "Tamera"];  sisters[0]; |
| **OUTPUT** |
| "Tia" |

To get the last element, you can use brackets and `1` less than the array’s length property.

|  |
| --- |
| **EXAMPLE** |
| var actors = ["Felicia", "Nathan", "Neil"];  actors[actors.length - 1]; |
| **OUTPUT** |
| "Neil" |

This also works for setting an element’s value.

|  |
| --- |
| **EXAMPLE** |
| var colors = ["red", "yelo", "blue"]; colors[1] = "yellow"; colors; |
| **OUTPUT** |
| ["red", "yellow", "blue"] |

Properties and methods

Arrays have their own built-in variables and functions, also known as properties and methods. Here are some of the most common ones.

length

An array’s length property stores the number of elements inside the array.

|  |
| --- |
| **EXAMPLE** |
| ["a", "b", "c", 1, 2, 3].length; |
| **OUTPUT** |
| 6 |

concat

An array’s concat method returns a new array that combines the values of two arrays.

|  |
| --- |
| **EXAMPLE** |
| ["tortilla chips"].concat(["salsa", "queso", "guacamole"]); |
| **OUTPUT** |
| ["tortilla chips", "salsa", "queso", "guacamole"] |

pop

An array’s pop method removes the last element in the array and returns that element’s value.

|  |
| --- |
| **EXAMPLE** |
| ["Jupiter", "Saturn", "Uranus", "Neptune", "Pluto"].pop(); |
| **OUTPUT** |
| "Pluto" |

push

An array’s push method adds an element to the array and returns the array’s length.

|  |
| --- |
| **EXAMPLE** |
| ["John", "Kate"].push(8); |
| **OUTPUT** |
| 3 |

reverse

An array’s reverse method returns a copy of the array in opposite order.

|  |
| --- |
| **EXAMPLE** |
| ["a", "b", "c"].reverse(); |
| **OUTPUT** |
| ["c", "b", "a"] |

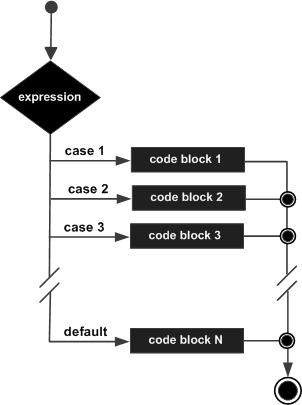
# Switch Case

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.

## **Flow Chart**

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



### **Syntax**

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

switch (expression) {

case condition 1: statement(s)

break;

case condition 2: statement(s)

break;

...

case condition n: statement(s)

break;

default: statement(s)

}

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

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

### **Example**

Try the following example to implement switch-case statement.

[Live Demo](http://tpcg.io/pNWbNd)

<html>

<body>

<script type = "text/javascript">

<!--

var grade = 'A';

document.write("Entering switch block<br />");

switch (grade) {

case 'A': document.write("Good job<br />");

break;

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

break;

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

break;

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

break;

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

break;

default: document.write("Unknown grade<br />")

}

document.write("Exiting switch block");

//-->

</script>

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

</body>

</html>

### **Output**

Entering switch block

Good job

Exiting switch block

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

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

[Live Demo](http://tpcg.io/ouqbd3)

<html>

<body>

<script type = "text/javascript">

<!--

var grade = 'A';

document.write("Entering switch block<br />");

switch (grade) {

case 'A': document.write("Good job<br />");

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

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

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

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

default: document.write("Unknown grade<br />")

}

document.write("Exiting switch block");

//-->

</script>

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

</body>

</html>

### **Output**

Entering switch block

Good job

Pretty good

Passed

Not so good

Failed

Unknown grade

Exiting switch block

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

# Looping in JS

# JavaScript Loops

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

There are four types of loops in JavaScript.

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

## **1) JavaScript For loop**

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

1. for (initialization; condition; increment)
2. {
3. code to be executed
4. }

Let’s see the simple example of for loop in javascript.

Play Video

1. **<script>**
2. for (i=1; i**<**=5; i++)
3. {
4. document.write(i + "**<br/>**")
5. }
6. **</script>**

Output:

1  
2  
3  
4  
5

## **2) JavaScript while loop**

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

1. while (condition)
2. {
3. code to be executed
4. }

Let’s see the simple example of while loop in javascript.

1. **<script>**
2. var i=11;
3. while (i**<**=15)
4. {
5. document.write(i + "**<br/>**");
6. i++;
7. }
8. **</script>**

Output:

11  
12  
13  
14  
15

## **3) JavaScript do while loop**

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

1. do{
2. code to be executed
3. }while (condition);

Let’s see the simple example of do while loop in javascript.

1. **<script>**
2. var i=21;
3. do{
4. document.write(i + "**<br/>**");
5. i++;
6. }while (i**<**=25);
7. **</script>**

Output:

21  
22  
23  
24  
25

## **4) JavaScript for in loop**

The **JavaScript for in loop** is used to iterate the properties of an object. We will discuss about it later.

# Functions

# JavaScript Functions

**JavaScript functions** are used to perform operations. We can call JavaScript function many times to reuse the code.

#### **Advantage of JavaScript function**

There are mainly two advantages of JavaScript functions.

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

## **JavaScript Function Syntax**

The syntax of declaring function is given below.

1. function functionName([arg1, arg2, ...argN]){
2. //code to be executed
3. }

JavaScript Functions can have 0 or more arguments.

Play Video

## **JavaScript Function Example**

Let’s see the simple example of function in JavaScript that does not has arguments.

1. **<script>**
2. function msg(){
3. alert("hello! this is message");
4. }
5. **</script>**
6. **<input** type="button" onclick="msg()" value="call function"**/>**

#### **Output of the above example**

## **JavaScript Function Arguments**

We can call function by passing arguments. Let’s see the example of function that has one argument.

1. **<script>**
2. function getcube(number){
3. alert(number\*number\*number);
4. }
5. **</script>**
6. **<form>**
7. **<input** type="button" value="click" onclick="getcube(4)"**/>**
8. **</form>**

#### **Output of the above example**

Top of Form

Bottom of Form

## **Function with Return Value**

We can call function that returns a value and use it in our program. Let’s see the example of function that returns value.

1. **<script>**
2. function getInfo(){
3. return "hello javatpoint! How r u?";
4. }
5. **</script>**
6. **<script>**
7. document.write(getInfo());
8. **</script>**

#### **Output of the above example**

hello javatpoint! How r u?

## **JavaScript Function Object**

In JavaScript, the purpose of **Function constructor** is to create a new Function object. It executes the code globally. However, if we call the constructor directly, a function is created dynamically but in an unsecured way.

## **Syntax**

1. new Function ([arg1[, arg2[, ....argn]],] functionBody)

## **Parameter**

**arg1, arg2, .... , argn** - It represents the argument used by function.

**functionBody** - It represents the function definition.

## **JavaScript Function Methods**

Let's see function methods with description.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [apply()](https://www.javatpoint.com/javascript-function-apply-method) | It is used to call a function contains this value and a single array of arguments. |
| [bind()](https://www.javatpoint.com/javascript-function-bind-method) | It is used to create a new function. |
| [call()](https://www.javatpoint.com/javascript-function-call-method) | It is used to call a function contains this value and an argument list. |
| [toString()](https://www.javatpoint.com/javascript-function-tostring-method) | It returns the result in a form of a string. |

## **JavaScript Function Object Examples**

### **Example 1**

Let's see an example to display the sum of given numbers.

1. **<script>**
2. var add=new Function("num1","num2","return num1+num2");
3. document.writeln(add(2,5));
4. **</script>**

**Output:**

7

### **Example 2**

Let's see an example to display the power of provided value.

1. **<script>**
2. var pow=new Function("num1","num2","return Math.pow(num1,num2)");
3. document.writeln(pow(2,3));
4. **</script>**

Output:

8

# 7.Math and String Manipulation

## **The Math Object**

Unlike other objects, the Math object has no constructor.

The Math object is static.

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

## **Math Properties (Constants)**

The syntax for any Math property is : Math.*property*.

JavaScript provides 8 mathematical constants that can be accessed as Math properties:

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 ½

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

**Example**

**<!DOCTYPE html>**

**<html>**

**<body>**

**<h2>JavaScript Math Constants</h2>**

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

**<script>**

**document.getElementById("demo").innerHTML =**

**"<p><b>Math.E:</b> " + Math.E + "</p>" +**

**"<p><b>Math.PI:</b> " + Math.PI + "</p>" +**

**"<p><b>Math.SQRT2:</b> " + Math.SQRT2 + "</p>" +**

**"<p><b>Math.SQRT1\_2:</b> " + Math.SQRT1\_2 + "</p>" +**

**"<p><b>Math.LN2:</b> " + Math.LN2 + "</p>" +**

**"<p><b>Math.LN10:</b> " + Math.LN10 + "</p>" +**

**"<p><b>Math.LOG2E:</b> " + Math.LOG2E + "</p>" +**

**"<p><b>Math.Log10E:</b> " + Math.LOG10E + "</p>";**

**</script>**

**</body>**

**</html>**

**Output:**

## JavaScript Math Constants

**Math.E:** 2.718281828459045

**Math.PI:** 3.141592653589793

**Math.SQRT2:** 1.4142135623730951

**Math.SQRT1\_2:** 0.7071067811865476

**Math.LN2:** 0.6931471805599453

**Math.LN10:** 2.302585092994046

**Math.LOG2E:** 1.4426950408889634

**Math.Log10E:** 0.4342944819032518

## **Math Methods**

The syntax for Math any methods is : Math.*method*(*number*)

## **Number to Integer**

There are 4 common methods to round a number to an integer:

|  |  |
| --- | --- |
| Math.round(x) | Returns x rounded to its nearest integer |
| Math.ceil(x) | Returns x rounded up to its nearest integer |
| Math.floor(x) | Returns x rounded down to its nearest integer |
| Math.trunc(x) | Returns the integer part of x ([new in ES6](https://www.w3schools.com/js/js_es6.asp)) |

## **JavaScript Math Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [abs(x)](https://www.w3schools.com/jsref/jsref_abs.asp) | Returns the absolute value of x |
| [acos(x)](https://www.w3schools.com/jsref/jsref_acos.asp) | Returns the arccosine of x, in radians |
| [acosh(x)](https://www.w3schools.com/jsref/jsref_acosh.asp) | Returns the hyperbolic arccosine of x |
| [asin(x)](https://www.w3schools.com/jsref/jsref_asin.asp) | Returns the arcsine of x, in radians |
| [asinh(x)](https://www.w3schools.com/jsref/jsref_asinh.asp) | Returns the hyperbolic arcsine of x |
| [atan(x)](https://www.w3schools.com/jsref/jsref_atan.asp) | Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians |
| [atan2(y, x)](https://www.w3schools.com/jsref/jsref_atan2.asp) | Returns the arctangent of the quotient of its arguments |
| [atanh(x)](https://www.w3schools.com/jsref/jsref_atanh.asp) | Returns the hyperbolic arctangent of x |
| [cbrt(x)](https://www.w3schools.com/jsref/jsref_cbrt.asp) | Returns the cubic root of x |
| [ceil(x)](https://www.w3schools.com/jsref/jsref_ceil.asp) | Returns x, rounded upwards to the nearest integer |
| [cos(x)](https://www.w3schools.com/jsref/jsref_cos.asp) | Returns the cosine of x (x is in radians) |
| [cosh(x)](https://www.w3schools.com/jsref/jsref_cosh.asp) | Returns the hyperbolic cosine of x |
| [exp(x)](https://www.w3schools.com/jsref/jsref_exp.asp) | Returns the value of Ex |
| [floor(x)](https://www.w3schools.com/jsref/jsref_floor.asp) | Returns x, rounded downwards to the nearest integer |
| [log(x)](https://www.w3schools.com/jsref/jsref_log.asp) | Returns the natural logarithm (base E) of x |
| [max(x, y, z, ..., n)](https://www.w3schools.com/jsref/jsref_max.asp) | Returns the number with the highest value |
| [min(x, y, z, ..., n)](https://www.w3schools.com/jsref/jsref_min.asp) | Returns the number with the lowest value |
| [pow(x, y)](https://www.w3schools.com/jsref/jsref_pow.asp) | Returns the value of x to the power of y |
| [random()](https://www.w3schools.com/jsref/jsref_random.asp) | Returns a random number between 0 and 1 |
| [round(x)](https://www.w3schools.com/jsref/jsref_round.asp) | Rounds x to the nearest integer |
| [sign(x)](https://www.w3schools.com/jsref/jsref_sign.asp) | Returns if x is negative, null or positive (-1, 0, 1) |
| [sin(x)](https://www.w3schools.com/jsref/jsref_sin.asp) | Returns the sine of x (x is in radians) |
| [sinh(x)](https://www.w3schools.com/jsref/jsref_sinh.asp) | Returns the hyperbolic sine of x |
| [sqrt(x)](https://www.w3schools.com/jsref/jsref_sqrt.asp) | Returns the square root of x |
| [tan(x)](https://www.w3schools.com/jsref/jsref_tan.asp) | Returns the tangent of an angle |
| [tanh(x)](https://www.w3schools.com/jsref/jsref_tanh.asp) | Returns the hyperbolic tangent of a number |
| [trunc(x)](https://www.w3schools.com/jsref/jsref_trunc.asp) | Returns the integer part of a number (x) |

# JavaScript String Manipulation Techniques

## **JavaScript Strings**

A JavaScript string stores a series of characters like "John Doe".

A string can be any text inside double or single quotes:

# <!DOCTYPE html>

# <html>

# <body>

# <h1>JavaScript Strings</h1>

# <p>Strings are written inside single or double quotes:</p>

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

# <script>

# let carName1 = "Volvo XC60"; // Double quotes

# let carName2 = 'Volvo XC60'; // Single quotes

# document.getElementById("demo").innerHTML =carName1 + " " + carName2;

# </script>

# </body>

# </html>

# Output:

# JavaScript Strings

Strings are written inside single or double quotes:

Volvo XC60 Volvo XC60

String indexes are zero-based:

The first character is in position 0, the second in 1, and so on.

## **JavaScript String Methods**

|  |  |
| --- | --- |
| **Name** | **Description** |
| [charAt()](https://www.w3schools.com/jsref/jsref_charat.asp) | Returns the character at a specified index (position) |
| [charCodeAt()](https://www.w3schools.com/jsref/jsref_charcodeat.asp) | Returns the Unicode of the character at a specified index |
| [concat()](https://www.w3schools.com/jsref/jsref_concat_string.asp) | Returns two or more joined strings |
| [constructor](https://www.w3schools.com/jsref/jsref_constructor_string.asp) | Returns the string's constructor function |
| [endsWith()](https://www.w3schools.com/jsref/jsref_endswith.asp) | Returns if a string ends with a specified value |
| [fromCharCode()](https://www.w3schools.com/jsref/jsref_fromcharcode.asp) | Returns Unicode values as characters |
| [includes()](https://www.w3schools.com/jsref/jsref_includes.asp) | Returns if a string contains a specified value |
| [indexOf()](https://www.w3schools.com/jsref/jsref_indexof.asp) | Returns the index (position) of the first occurrence of a value in a string |
| [lastIndexOf()](https://www.w3schools.com/jsref/jsref_lastindexof.asp) | Returns the index (position) of the last occurrence of a value in a string |
| [length](https://www.w3schools.com/jsref/jsref_length_string.asp) | Returns the length of a string |
| [localeCompare()](https://www.w3schools.com/jsref/jsref_localecompare.asp) | Compares two strings in the current locale |
| [match()](https://www.w3schools.com/jsref/jsref_match.asp) | Searches a string for a value, or a regular expression, and returns the matches |
| [prototype](https://www.w3schools.com/jsref/jsref_prototype_string.asp) | Allows you to add properties and methods to an object |
| [repeat()](https://www.w3schools.com/jsref/jsref_repeat.asp) | Returns a new string with a number of copies of a string |
| [replace()](https://www.w3schools.com/jsref/jsref_replace.asp) | Searches a string for a value, or a regular expression, and returns a string where the values are replaced |
| [search()](https://www.w3schools.com/jsref/jsref_search.asp) | Searches a string for a value, or regular expression, and returns the index (position) of the match |
| [slice()](https://www.w3schools.com/jsref/jsref_slice_string.asp) | Extracts a part of a string and returns a new string |
| [split()](https://www.w3schools.com/jsref/jsref_split.asp) | Splits a string into an array of substrings |
| [startsWith()](https://www.w3schools.com/jsref/jsref_startswith.asp) | Checks whether a string begins with specified characters |
| [substr()](https://www.w3schools.com/jsref/jsref_substr.asp) | Extracts a number of characters from a string, from a start index (position) |
| [substring()](https://www.w3schools.com/jsref/jsref_substring.asp) | Extracts characters from a string, between two specified indices (positions) |
| [toLocaleLowerCase()](https://www.w3schools.com/jsref/jsref_tolocalelowercase.asp) | Returns a string converted to lowercase letters, using the host's locale |
| [toLocaleUpperCase()](https://www.w3schools.com/jsref/jsref_tolocaleuppercase.asp) | Returns a string converted to uppercase letters, using the host's locale |
| [toLowerCase()](https://www.w3schools.com/jsref/jsref_tolowercase.asp) | Returns a string converted to lowercase letters |
| [toString()](https://www.w3schools.com/jsref/jsref_tostring_string.asp) | Returns a string or a string object as a string |
| [toUpperCase()](https://www.w3schools.com/jsref/jsref_touppercase.asp) | Returns a string converted to uppercase letters |
| [trim()](https://www.w3schools.com/jsref/jsref_trim_string.asp) | Returns a string with removed whitespaces |
| [trimEnd()](https://www.w3schools.com/jsref/jsref_string_trim_end.asp) | Returns a string with removed whitespaces from the end |
| [trimStart()](https://www.w3schools.com/jsref/jsref_string_trim_start.asp) | Returns a string with removed whitespaces from the start |
| [valueOf()](https://www.w3schools.com/jsref/jsref_valueof_string.asp) | Returns the primitive value of a string or a string object |

# JavaScript String charAt()

# <!DOCTYPE html>

# <html>

# <body>

# <h1>JavaScript Strings</h1>

# <h2>The charAt() Method</h2>

# <p>charAt() returns the character at a specified index (position) in a string.</p>

# <p>Get the first character in a string:</p>

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

# <script>

# let text = "HELLO WORLD";

# let letter = text.charAt(0);

# document.getElementById("demo").innerHTML = letter;

# </script>

# </body>

# </html>

# Output:

# JavaScript Strings

## The charAt() Method

charAt() returns the character at a specified index (position) in a string.

Get the first character in a string:

H

# JavaScript String charCodeAt()

# <!DOCTYPE html>

# <html>

# <body>

# <h1>JavaScript Strings</h1>

# <h2>The charCodeAt() Method</h2>

# <p>charCodeAt() returns the Unicode of the character at a specified position in a string.</p>

# <p>Get the Unicode of the first character:</p>

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

# <script>

# let text = "HELLO WORLD";

# let code = text.charCodeAt(0);

# document.getElementById("demo").innerHTML = code;

# </script>

# </body>

# </html>

# Output

# JavaScript Strings

## The charCodeAt() Method

charCodeAt() returns the Unicode of the character at a specified position in a string.

Get the Unicode of the first character:

72

# JavaScript String codePointAt()

# <!DOCTYPE html>

# <html>

# <body>

# <h1>JavaScript Strings</h1>

# <h2>The codePointAt() Method</h2>

# <p>codePointAt() returns the code point at a specified position in a string.</p>

# <p>Get the code point at the first character:</p>

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

# <script>

# let text = "HELLO WORLD";

# let code = text.codePointAt(0);

# document.getElementById("demo").innerHTML = code;

# </script>

# </body>

# </html>

# Output

# JavaScript Strings

## The codePointAt() Method

codePointAt() returns the code point at a specified position in a string.

Get the code point at the first character:

72

# JavaScript String concat()

# <!DOCTYPE html>

# <html>

# <body>

# <h1>JavaScript Strings</h1>

# <h2>The concat() Method</h2>

# <p>The concat() method joins two or more strings.</p>

# <p>Join "sea" and "food":</p>

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

# <script>

# let text1 = "sea";

# let text2 = "food";

# let result = text1.concat(text2);

# document.getElementById("demo").innerHTML = result;

# </script>

# </body>

# </html>

# Output

# JavaScript Strings

## The concat() Method

The concat() method joins two or more strings.

Join "sea" and "food":

seafood

# JavaScript String constructor

# <!DOCTYPE html>

# <html>

# <body>

# <h1>JavaScript Strings</h1>

# <h2>The constructor Property</h2>

# <p>The constructor property returns the function that created the String prototype:</p>

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

# <script>

# let message = "Hello World!";

# let text = message.constructor;

# document.getElementById("demo").innerHTML = text;

# </script>

# </body>

# </html>

# Output

# JavaScript Strings

## The constructor Property

The constructor property returns the function that created the String prototype:

function String() { [native code]

# JavaScript String endsWith()

# <!DOCTYPE html>

# <html>

# <body>

# <h1>JavaScript Strings</h1>

# <h2>The endsWith() Method</h2>

# <p>endsWith() returns true if a string ends with a specified string, otherwise false.</p>

# <p>Check if "Hello world" ends with "world":</p>

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

# <p>endsWith() is not supported in IE 11 (or earlier versions).</p>

# <script>

# let text = "Hello world";

# let result = text.endsWith("world");

# document.getElementById("demo").innerHTML = result;

# </script>

# </body>

# </html>

# Output

# JavaScript Strings

## The endsWith() Method

endsWith() returns true if a string ends with a specified string, otherwise false.

Check if "Hello world" ends with "world":

true

endsWith() is not supported in IE 11 (or earlier versions).

# JavaScript String replace()

# <!DOCTYPE html>

# <html>

# <body>

# <h1>JavaScript Strings</h1>

# <h2>The replace() Method</h2>

# <p>replace() searches a string for a value,

# and returns a new string with the specified value(s) replaced:</p>

# <p id="demo">Visit Microsoft!</p>

# <script>

# let text = document.getElementById("demo").innerHTML;

# document.getElementById("demo").innerHTML = text.replace("Microsoft", "W3Schools");

# </script>

# </body>

# </html>

# Output

# JavaScript Strings

## The replace() Method

replace() searches a string for a value, and returns a new string with the specified value(s) replaced:

Visit W3Schools!

# JavaScript String match()

# <!DOCTYPE html>

# <html>

# <body>

# <h1>JavaScript Strings</h1>

# <h2>The match() Method</h2>

# <p>match() searches for a match in a string.</p>

# <p>Do a search for "ain":</p>

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

# <script>

# let text = "The rain in SPAIN stays mainly in the plain";

# let result = text.match("ain");

# 

# document.getElementById("demo").innerHTML = result;

# </script>

# </body>

# </html>

# Output

# JavaScript Strings

## The match() Method

match() searches for a match in a string.

Do a search for "ain":

ain

# JavaScript String slice()

# <!DOCTYPE html>

# <html>

# <body>

# <h1>JavaScript Strings</h1>

# <h2>The slice() Method</h2>

# <p>slice() extracts a part of a string and returns the extracted part:</p>

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

# <script>

# let text = "Hello world!";

# let result = text.slice(0, 5);

# document.getElementById("demo").innerHTML = result;

# </script>

# </body>

# </html>

# Output

# JavaScript Strings

## The slice() Method

slice() extracts a part of a string and returns the extracted part:

Hello

# JavaScript String split()

# <!DOCTYPE html>

# <html>

# <body>

# <h1>JavaScript Strings</h1>

# <h2>The split() Method</h2>

# <p>split() splits a string into an array of substrings, and returns the array:</p>

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

# <script>

# let text = "How are you doing today?";

# const myArray = text.split(" ");

# document.getElementById("demo").innerHTML = myArray;

# </script>

# </body>

# </html>

# Output

# JavaScript Strings

## The split() Method

split() splits a string into an array of substrings, and returns the array:

How,are,you,doing,today?

# JavaScript String toUpperCase()

# <!DOCTYPE html>

# <html>

# <body>

# <h1>JavaScript Strings</h1>

# <h2>To toUpperCase() Method</h2>

# <p>toUpperCase() converts a string to uppercase letters:</p>

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

# <script>

# let text = "Hello World!";

# let result = text.toUpperCase();

# document.getElementById("demo").innerHTML = result;

# </script>

# </body>

# </html>

# Output

# JavaScript Strings

## To toUpperCase() Method

toUpperCase() converts a string to uppercase letters:

HELLO WORLD!

# 8.Objects and Arrays

## **The JavaScript Array Object**

The Array object is used to store multiple values in a single variable.

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript Arrays</h2>

# <p>The Array object is used to store multiple values in a single variable:</p>

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

# <script>

# const cars = ["Saab", "Volvo", "BMW"];

# document.getElementById("demo").innerHTML = cars;

# </script>

# </body>

# </html> output

## JavaScript Arrays

The Array object is used to store multiple values in a single variable:

Saab,Volvo,BMW

## **JavaScript Array Methods and Properties**

|  |  |
| --- | --- |
| **Name** | **Description** |
| [concat()](https://www.w3schools.com/jsref/jsref_concat_array.asp) | Joins arrays and returns an array with the joined arrays |
| [constructor](https://www.w3schools.com/jsref/jsref_constructor_array.asp) | Returns the function that created the Array object's prototype |
| [copyWithin()](https://www.w3schools.com/jsref/jsref_copywithin.asp) | Copies array elements within the array, to and from specified positions |
| [entries()](https://www.w3schools.com/jsref/jsref_entries.asp) | Returns a key/value pair Array Iteration Object |
| [every()](https://www.w3schools.com/jsref/jsref_every.asp) | Checks if every element in an array pass a test |
| [fill()](https://www.w3schools.com/jsref/jsref_fill.asp) | Fill the elements in an array with a static value |
| [filter()](https://www.w3schools.com/jsref/jsref_filter.asp) | Creates a new array with every element in an array that pass a test |
| [find()](https://www.w3schools.com/jsref/jsref_find.asp) | Returns the value of the first element in an array that pass a test |
| [findIndex()](https://www.w3schools.com/jsref/jsref_findindex.asp) | Returns the index of the first element in an array that pass a test |
| [forEach()](https://www.w3schools.com/jsref/jsref_foreach.asp) | Calls a function for each array element |
| [from()](https://www.w3schools.com/jsref/jsref_from.asp) | Creates an array from an object |
| [includes()](https://www.w3schools.com/jsref/jsref_includes_array.asp) | Check if an array contains the specified element |
| [indexOf()](https://www.w3schools.com/jsref/jsref_indexof_array.asp) | Search the array for an element and returns its position |
| [isArray()](https://www.w3schools.com/jsref/jsref_isarray.asp) | Checks whether an object is an array |
| [join()](https://www.w3schools.com/jsref/jsref_join.asp) | Joins all elements of an array into a string |
| [keys()](https://www.w3schools.com/jsref/jsref_keys.asp) | Returns a Array Iteration Object, containing the keys of the original array |
| [lastIndexOf()](https://www.w3schools.com/jsref/jsref_lastindexof_array.asp) | Search the array for an element, starting at the end, and returns its position |
| [length](https://www.w3schools.com/jsref/jsref_length_array.asp) | Sets or returns the number of elements in an array |
| [map()](https://www.w3schools.com/jsref/jsref_map.asp) | Creates a new array with the result of calling a function for each array element |
| [pop()](https://www.w3schools.com/jsref/jsref_pop.asp) | Removes the last element of an array, and returns that element |
| [prototype](https://www.w3schools.com/jsref/jsref_prototype_array.asp) | Allows you to add properties and methods to an Array object |
| [push()](https://www.w3schools.com/jsref/jsref_push.asp) | Adds new elements to the end of an array, and returns the new length |
| [reduce()](https://www.w3schools.com/jsref/jsref_reduce.asp) | Reduce the values of an array to a single value (going left-to-right) |
| [reduceRight()](https://www.w3schools.com/jsref/jsref_reduceright.asp) | Reduce the values of an array to a single value (going right-to-left) |
| [reverse()](https://www.w3schools.com/jsref/jsref_reverse.asp) | Reverses the order of the elements in an array |
| [shift()](https://www.w3schools.com/jsref/jsref_shift.asp) | Removes the first element of an array, and returns that element |
| [slice()](https://www.w3schools.com/jsref/jsref_slice_array.asp) | Selects a part of an array, and returns the new array |
| [some()](https://www.w3schools.com/jsref/jsref_some.asp) | Checks if any of the elements in an array pass a test |
| [sort()](https://www.w3schools.com/jsref/jsref_sort.asp) | Sorts the elements of an array |
| [splice()](https://www.w3schools.com/jsref/jsref_splice.asp) | Adds/Removes elements from an array |
| [toString()](https://www.w3schools.com/jsref/jsref_tostring_array.asp) | Converts an array to a string, and returns the result |
| [unshift()](https://www.w3schools.com/jsref/jsref_unshift.asp) | Adds new elements to the beginning of an array, and returns the new length |
| [valueOf()](https://www.w3schools.com/jsref/jsref_valueof_array.asp) | Returns the primitive value of an array |

# JavaScript Array sort()

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript Arrays</h2>

# <p>The Array.sort() method sorts the elements of an array.</p>

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

# <script>

# const fruits = ["Banana", "Orange", "Apple", "Mango"];

# document.getElementById("demo").innerHTML = fruits.sort();

# </script>

# </body>

# </html>

# Output

## JavaScript Arrays

The Array.sort() method sorts the elements of an array.

Apple,Banana,Mango,Orange

# Date and Time

## **Date Object**

The Date object works with dates and times.

Date objects are created with new Date().

There are four ways of instantiating (creating) a date:

Example:1

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript Dates</h2>

# <p>new Date() creates a new date object with the current date and time:</p>

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

# <script>

# const d = new Date();

# document.getElementById("demo").innerHTML = d;

# </script>

# </body>

# </html>

# Output

## JavaScript Dates

new Date() creates a new date object with the current date and time:

Mon Mar 13 2023 20:15:11 GMT+0530 (India Standard Time)

# Example:2

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript new Date()</h2>

# <p>JavaScript stores dates as milliseconds from Jan 1, 1970.</p>

# <p>100000000000 milliseconds from Jan 1, 1970, is approximately Mar 3, 1973:</p>

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

# <script>

# const d = new Date(100000000000);

# document.getElementById("demo").innerHTML = d;

# </script>

# </body>

# </html>

# Output

## JavaScript new Date()

JavaScript stores dates as milliseconds from Jan 1, 1970.

100000000000 milliseconds from Jan 1, 1970, is approximately Mar 3, 1973:

Sat Mar 03 1973 15:16:40 GMT+0530 (India Standard Time)

# Example:3

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript new Date()</h2>

# <p>Create a date using ISO notation:</p>

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

# <script>

# const d = new Date("2015-03-25");

# document.getElementById("demo").innerHTML = d;</script>

# </body>

# </html>

# Output

## JavaScript new Date()

Create a date using ISO notation:

Wed Mar 25 2015 05:30:00 GMT+0530 (India Standard Time)

# Example:4

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript new Date()</h2>

# <p>new Date(7 numbers) creates a date object with the specified date and time:</p>

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

# <script>

# const d = new Date(2018, 11, 24, 10, 33, 30, 0);

# document.getElementById("demo").innerHTML = d;

# </script>

# </body>

# </html>

# Output

## JavaScript new Date()

new Date(7 numbers) creates a date object with the specified date and time:

Mon Dec 24 2018 10:33:30 GMT+0530 (India Standard Time)

## **JavaScript Date Methods and Properties**

|  |  |
| --- | --- |
| **Name** | **Description** |
| [constructor](https://www.w3schools.com/jsref/jsref_constructor_date.asp) | Returns the function that created the Date object's prototype |
| [getDate()](https://www.w3schools.com/jsref/jsref_getdate.asp) | Returns the day of the month (from 1-31) |
| [getDay()](https://www.w3schools.com/jsref/jsref_getday.asp) | Returns the day of the week (from 0-6) |
| [getFullYear()](https://www.w3schools.com/jsref/jsref_getfullyear.asp) | Returns the year |
| [getHours()](https://www.w3schools.com/jsref/jsref_gethours.asp) | Returns the hour (from 0-23) |
| [getMilliseconds()](https://www.w3schools.com/jsref/jsref_getmilliseconds.asp) | Returns the milliseconds (from 0-999) |
| [getMinutes()](https://www.w3schools.com/jsref/jsref_getminutes.asp) | Returns the minutes (from 0-59) |
| [getMonth()](https://www.w3schools.com/jsref/jsref_getmonth.asp) | Returns the month (from 0-11) |
| [getSeconds()](https://www.w3schools.com/jsref/jsref_getseconds.asp) | Returns the seconds (from 0-59) |
| [getTime()](https://www.w3schools.com/jsref/jsref_gettime.asp) | Returns the number of milliseconds since midnight Jan 1 1970, and a specified date |
| [getTimezoneOffset()](https://www.w3schools.com/jsref/jsref_gettimezoneoffset.asp) | Returns the time difference between UTC time and local time, in minutes |
| [getUTCDate()](https://www.w3schools.com/jsref/jsref_getutcdate.asp) | Returns the day of the month, according to universal time (from 1-31) |
| [getUTCDay()](https://www.w3schools.com/jsref/jsref_getutcday.asp) | Returns the day of the week, according to universal time (from 0-6) |
| [getUTCFullYear()](https://www.w3schools.com/jsref/jsref_getutcfullyear.asp) | Returns the year, according to universal time |
| [getUTCHours()](https://www.w3schools.com/jsref/jsref_getutchours.asp) | Returns the hour, according to universal time (from 0-23) |
| [getUTCMilliseconds()](https://www.w3schools.com/jsref/jsref_getutcmilliseconds.asp) | Returns the milliseconds, according to universal time (from 0-999) |
| [getUTCMinutes()](https://www.w3schools.com/jsref/jsref_getutcminutes.asp) | Returns the minutes, according to universal time (from 0-59) |
| [getUTCMonth()](https://www.w3schools.com/jsref/jsref_getutcmonth.asp) | Returns the month, according to universal time (from 0-11) |
| [getUTCSeconds()](https://www.w3schools.com/jsref/jsref_getutcseconds.asp) | Returns the seconds, according to universal time (from 0-59) |
| getYear() | Deprecated. Use the [getFullYear()](https://www.w3schools.com/jsref/jsref_getfullyear.asp) method instead |
| [now()](https://www.w3schools.com/jsref/jsref_now.asp) | Returns the number of milliseconds since midnight Jan 1, 1970 |
| [parse()](https://www.w3schools.com/jsref/jsref_parse.asp) | Parses a date string and returns the number of milliseconds since January 1, 1970 |
| [prototype](https://www.w3schools.com/jsref/jsref_prototype_date.asp) | Allows you to add properties and methods to an object |
| [setDate()](https://www.w3schools.com/jsref/jsref_setdate.asp) | Sets the day of the month of a date object |
| [setFullYear()](https://www.w3schools.com/jsref/jsref_setfullyear.asp) | Sets the year of a date object |
| [setHours()](https://www.w3schools.com/jsref/jsref_sethours.asp) | Sets the hour of a date object |
| [setMilliseconds()](https://www.w3schools.com/jsref/jsref_setmilliseconds.asp) | Sets the milliseconds of a date object |
| [setMinutes()](https://www.w3schools.com/jsref/jsref_setminutes.asp) | Set the minutes of a date object |
| [setMonth()](https://www.w3schools.com/jsref/jsref_setmonth.asp) | Sets the month of a date object |
| [setSeconds()](https://www.w3schools.com/jsref/jsref_setseconds.asp) | Sets the seconds of a date object |
| [setTime()](https://www.w3schools.com/jsref/jsref_settime.asp) | Sets a date to a specified number of milliseconds after/before January 1, 1970 |
| [setUTCDate()](https://www.w3schools.com/jsref/jsref_setutcdate.asp) | Sets the day of the month of a date object, according to universal time |
| [setUTCFullYear()](https://www.w3schools.com/jsref/jsref_setutcfullyear.asp) | Sets the year of a date object, according to universal time |
| [setUTCHours()](https://www.w3schools.com/jsref/jsref_setutchours.asp) | Sets the hour of a date object, according to universal time |
| [setUTCMilliseconds()](https://www.w3schools.com/jsref/jsref_setutcmilliseconds.asp) | Sets the milliseconds of a date object, according to universal time |
| [setUTCMinutes()](https://www.w3schools.com/jsref/jsref_setutcminutes.asp) | Set the minutes of a date object, according to universal time |
| [setUTCMonth()](https://www.w3schools.com/jsref/jsref_setutcmonth.asp) | Sets the month of a date object, according to universal time |
| [setUTCSeconds()](https://www.w3schools.com/jsref/jsref_setutcseconds.asp) | Set the seconds of a date object, according to universal time |
| setYear() | Deprecated. Use the [setFullYear()](https://www.w3schools.com/jsref/jsref_setfullyear.asp) method instead |
| [toDateString()](https://www.w3schools.com/jsref/jsref_todatestring.asp) | Converts the date portion of a Date object into a readable string |
| toGMTString() | Deprecated. Use the [toUTCString()](https://www.w3schools.com/jsref/jsref_toutcstring.asp) method instead |
| [toISOString()](https://www.w3schools.com/jsref/jsref_toisostring.asp) | Returns the date as a string, using the ISO standard |
| [toJSON()](https://www.w3schools.com/jsref/jsref_tojson.asp) | Returns the date as a string, formatted as a JSON date |
| [toLocaleDateString()](https://www.w3schools.com/jsref/jsref_tolocaledatestring.asp) | Returns the date portion of a Date object as a string, using locale conventions |
| [toLocaleTimeString()](https://www.w3schools.com/jsref/jsref_tolocaletimestring.asp) | Returns the time portion of a Date object as a string, using locale conventions |
| [toLocaleString()](https://www.w3schools.com/jsref/jsref_tolocalestring.asp) | Converts a Date object to a string, using locale conventions |
| [toString()](https://www.w3schools.com/jsref/jsref_tostring_date.asp) | Converts a Date object to a string |
| [toTimeString()](https://www.w3schools.com/jsref/jsref_totimestring.asp) | Converts the time portion of a Date object to a string |
| [toUTCString()](https://www.w3schools.com/jsref/jsref_toutcstring.asp) | Converts a Date object to a string, according to universal time |
| [UTC()](https://www.w3schools.com/jsref/jsref_utc.asp) | Returns the number of milliseconds in a date since midnight of January 1, 1970, according to UTC time |
| [valueOf()](https://www.w3schools.com/jsref/jsref_valueof_date.asp) | Returns the primitive value of a Date object |

# Time:

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

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

## **Syntax**

Its syntax is as follows −

Date.getTime()

<html>

<head>

<title>JavaScript getTime Method</title>

</head>

<body>

<script type = "text/javascript">

var dt = new Date( "December 25, 1995 23:15:20" );

document.write("getTime() : " + dt.getTime() );

</script>

</body>

</html>

# Output

# getTime() : 819913520000

# Example: Date and Time

# <!DOCTYPE html>

# <html>

# <body>

# <h1>JavaScript Dates</h1>

# <h2>Using new Date()</h2>

# <p>100000000000 milliseconds from January 01 1970 UTC is:</p>

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

# <script>

# const d = new Date(100000000000);

# document.getElementById("demo").innerHTML = d;

# </script>

# </body>

# </html>

# Output

# JavaScript Dates

## Using new Date()

100000000000 milliseconds from January 01 1970 UTC is:

Sat Mar 03 1973 15:16:40 GMT+0530 (India Standard Ti

# 10.Conditional Statements

# JavaScript if, else, and else if

## Conditional statements are used to perform different actions based on different conditions. **Conditional Statements**

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

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 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

## **The if Statement**

Use the if statement to specify a block of JavaScript code to be executed if a condition is true.

### **Syntax**

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

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript if</h2>

# <p>Display "Good day!" if the hour is less than 18:00:</p>

# <p id="demo">Good Evening!</p>

# <script>

# if (new Date().getHours() < 18) {

# document.getElementById("demo").innerHTML = "Good day!";

# }

# </script>

# </body>

# </html>

# Output

## JavaScript if

Display "Good day!" if the hour is less than 18:00:

Good Evening!

## **The else Statement**

Use the else statement to specify a block of code to be executed if the condition is false.

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}

# Example

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript if .. else</h2>

# <p>A time-based greeting:</p>

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

# <script>

# const hour = new Date().getHours();

# let greeting;

# if (hour < 18) {

# greeting = "Good day";

# } else {

# greeting = "Good evening";

# }

# document.getElementById("demo").innerHTML = greeting;

# </script>

# </body>

# </html>

# Output

## JavaScript if .. else

A time-based greeting:

Good evening

## **The else if Statement**

Use the else if statement to specify a new condition if the first condition is false.

### **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}

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript if .. else</h2>

# <p>A time-based greeting:</p>

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

# <script>

# const time = new Date().getHours();

# let greeting;

# if (time < 10) {

# greeting = "Good morning";

# } else if (time < 20) {

# greeting = "Good day";

# } else {

# greeting = "Good evening";

# }

# document.getElementById("demo").innerHTML = greeting;

# </script>

# </body>

# </html>

# Output

## JavaScript if .. else

A time-based greeting:

Good evening

# 11.Switch Case

# JavaScript Switch Statement

The switch statement is used to perform different actions based on different conditions.

## **The JavaScript Switch Statement**

Use the switch statement to select one of many code blocks to be executed.

### **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.

# Example:

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript switch</h2>

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

# <script>

# let day;

# switch (new Date().getDay()) {

# case 0:

# day = "Sunday";

# break;

# case 1:

# day = "Monday";

# break;

# case 2:

# day = "Tuesday";

# break;

# case 3:

# day = "Wednesday";

# break;

# case 4:

# day = "Thursday";

# break;

# case 5:

# day = "Friday";

# break;

# case 6:

# day = "Saturday";

# }

# document.getElementById("demo").innerHTML = "Today is " + day;

# </script>

# </body>

# </html>

# Output:

## JavaScript switch

Today is Monday

# 12. Looping in JS

# JavaScript For Loop

# Loops can execute a block of code a number of times.

# Example

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript For Loop</h2>

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

# <script>

# const cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];

# let text = "";

# for (let i = 0; i < cars.length; i++) {

# text += cars[i] + "<br>";

# }

# document.getElementById("demo").innerHTML = text;

# </script>

# </body>

# </html>

# Output

## JavaScript For Loop

BMW  
Volvo  
Saab  
Ford  
Fiat  
Audi

## **The For In Loop**

The JavaScript for in statement loops through the properties of an Object:

### **Syntax**

for (key in object) {  
  // *code block to be executed*  
}

# Example:

# <!DOCTYPE html>

# <html>

# <body>

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

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

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

# <script>

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

# let txt = "";

# for (let x in person) {

# txt += person[x] + " ";

# }

# document.getElementById("demo").innerHTML = txt;

# </script>

# </body>

# </html>

# Output

## JavaScript For In Loop

The for in statement loops through the properties of an object:

John Doe 25

# Explanation

* The **for in** loop iterates over a **person** object
* Each iteration returns a **key** (x)
* The key is used to access the **value** of the key
* The value of the key is **person[x]**

## **The For Of Loop**

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

It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and more:

### **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.

# Example

# <!DOCTYPE html>

# <html>

# <body>

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

# <p>The for of statement loops through the values of any iterable object:</p>

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

# <script>

# const cars = ["BMW", "Volvo", "Mini"];

# let text = "";

# for (let x of cars) {

# text += x + "<br>";

# }

# document.getElementById("demo").innerHTML = text;

# </script>

# </body>

# </html>

# Output

## JavaScript For Of Loop

The for of statement loops through the values of any iterable object:

BMW  
Volvo  
Min

## **The While Loop**

The while loop loops through a block of code as long as a specified condition is true.

### **Syntax**

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

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript While Loop</h2>

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

# <script>

# let text = "";

# let i = 0;

# while (i < 10) {

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

# i++;

# }

# document.getElementById("demo").innerHTML = text;

# </script>

# </body>

# </html>

# Output

## JavaScript While Loop

The number is 0  
The number is 1  
The number is 2  
The number is 3  
The number is 4  
The number is 5  
The number is 6  
The number is 7  
The number is 8  
The number is 9

## **The Do While Loop**

The do while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

### **Syntax**

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

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript Do While Loop</h2>

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

# <script>

# let text = ""

# let i = 0;

# do {

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

# i++;

# }

# while (i < 10);

# document.getElementById("demo").innerHTML = text;

# </script>

# </body>

# </html>

# Output

## JavaScript Do While Loop

The number is 0  
The number is 1  
The number is 2  
The number is 3  
The number is 4  
The number is 5  
The number is 6  
The number is 7  
The number is 8  
The number is 9

# 12.Functions

JavaScript functions are **defined** with the function keyword.

You can use a function **declaration** or a function **expression**.

## **Function Declarations**

Earlier in this tutorial, you learned that functions are **declared** with the following syntax:

function functionName(parameters) {  
  // code to be executed  
}

Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are invoked (called upon).

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript Functions</h2>

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

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

# <script>

# var x = myFunction(4, 3);

# document.getElementById("demo").innerHTML = 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

## **Function Expressions**

A JavaScript function can also be defined using an **expression**.

A function expression can be stored in a variable:

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript Functions</h2>

# <p>A function can be stored in a variable:</p>

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

# <script>

# const x = function (a, b) {return a \* b};

# document.getElementById("demo").innerHTML = x;

# </script>

# </body>

# </html>

# Output

## JavaScript Functions

A function can be stored in a variable:

function (a, b) {return a \* b

## **Functions are Objects**

The typeof operator in JavaScript returns "function" for functions.

But, JavaScript functions can best be described as objects.

JavaScript functions have both **properties** and **methods**.

The arguments.length property returns the number of arguments received when the function was invoked:

# <!DOCTYPE html>

# <html>

# <body>

# <p>The arguments.length property returns the number of arguments received by the function:</p>

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

# <script>

# function myFunction(a, b) {

# return arguments.length;

# }

# document.getElementById("demo").innerHTML = myFunction(4, 3);

# </script>

# </body>

# </html>

# Output

The arguments.length property returns the number of arguments received by the function:

2

# JavaScript Function call()

## **Method Reuse**

With the call() method, you can write a method that can be used on different objects.

## **All Functions are Methods**

In JavaScript all functions are object methods.

If a function is not a method of a JavaScript object, it is a function of the global object (see previous chapter).

The example below creates an object with 3 properties, firstName, lastName, fullName.

# Example

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript Functions</h2>

# <p>This example creates an object with 3 properties (firstName, lastName, fullName).</p>

# <p>The fullName property is a method:</p>

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

# <script>

# const myObject = {

# firstName:"John",

# lastName: "Doe",

# fullName: function() {

# return this.firstName + " " + this.lastName;

# }

# }

# document.getElementById("demo").innerHTML = myObject.fullName();

# </script>

# </body>

# </html>

# Output

## JavaScript Functions

This example creates an object with 3 properties (firstName, lastName, fullName).

The fullName property is a method:

John Doe

## **The JavaScript call() Method**

The call() method is a predefined JavaScript method.

It can be used to invoke (call) a method with an owner object as an argument (parameter).

With call(), an object can use a method belonging to another object.

This example calls the **fullName** method of person, using it on **person1**:

# Example

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript Functions</h2>

# <p>This example calls the fullName method of person, using it on person1:

# </p>

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

# <script>

# const person = {

# fullName: function() {

# return this.firstName + " " + this.lastName;

# }

# }

# const person1 = {

# firstName:"John",

# lastName: "Doe"

# }

# const person2 = {

# firstName:"Mary",

# lastName: "Doe"

# }

# document.getElementById("demo").innerHTML = person.fullName.call(person1);

# </script>

# </body>

# </html>

# Output

## JavaScript Functions

This example calls the fullName method of person, using it on person1:

John Doe

## **The call() Method with Arguments**

The call() method can accept arguments:

# <!DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript Functions</h2>

# <p>This example calls the fullName method of person, using it on person1:

# </p>

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

# <script>

# const person = {

# fullName: function(city, country) {

# return this.firstName + " " + this.lastName + "," + city + "," + country;

# }

# }

# const person1 = {

# firstName:"John",

# lastName: "Doe"

# }

# const person2 = {

# firstName:"Mary",

# lastName: "Doe"

# }

# document.getElementById("demo").innerHTML = person.fullName.call(person1, "Oslo", "Norway");

# </script>

# </body>

# </html>

# Output

## JavaScript Functions

This example calls the fullName method of person, using it on person1:

John Doe,Oslo,Norway