Location of JS in HTML Doc

Old JavaScript examples may use a type attribute: <script type="text/javascript">.  
The type attribute is not required. JavaScript is the default scripting language in HTML.

Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display.

***External Scripts***

To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:

  <script src="myScript.js"></script>

You can place an external script reference in <head> or <body> as you like.

The script will behave as if it was located exactly where the <script> tag is located.

External scripts cannot contain <script> tags.

Output & Changing HTML Content

No Direct Output

JavaScript does not have any print object or print methods.

You cannot access output devices from JavaScript.

window.print() simply opens the print page browser option

Output to HTML

*HTML Element*

innerHTML

The innerHTML property defines the HTML content:

onclick='document.getElementById("demo").innerHTML = "Hello JavaScript!"'

*HTML Output*

document.write()

Never *call* document.write after the document has finished loading. It will overwrite the whole document (delete all existing HTML).

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

Output to CSS

*Changing <src> Value*

onclick="document.getElementById('myImage').src='pic\_bulbon.gif'"

*Changing CSS Styles*

document.getElementById("demo").style.fontSize = "35px";

Output to Alert Box

window.alert()

  window.alert(5 + 6);

In JavaScript, the window object is the global scope object. Therefore:

* variables, properties, and methods by default belong to the window object.
* specifying the window keyword is optional:

Output to Console

console.log()

For debugging

Accessing HTML Elements

document.getElementById(id)

Syntax

**Statements (consist of):**

* values
* operators
* expressions
* keywords
* comments

**Semicolons**

* separate statements
* add at end of each executable statement
* statements executed procedurally
* multiple lines on one line are allowed when separated by semicolons

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

**Whitespace**

* is ignored
* you can add or remove to make more readable
* good practice is to put spaces around operators ( = + - \* / )
* For best readability, programmers often like to avoid code lines longer than 80 characters.
  1. If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

**Curly Brackets**

* Groups statements that should be executed together

**Keywords**

* JavaScript statements often start with a keyword to identify the JavaScript action to be performed
* Keywords are reserved words. Reserved words cannot be used as names for variables.

Variables

Literals

Fixed values are called **Literals**.

* **Numbers** are written with or without decimals
* **Strings** are text, written within double or single quotes

Variables

Variable values are called **Variables**. JavaScript variables are containers for storing data values.

*Declaring*

const keyword to define a variable that cannot be reassigned,

let keyword to define a variable with restricted scope

You declare a JavaScript variable with the var keyword

**Declaring without Assigning**

* After the declaration, it has the value of undefined).
* You can also assign a value to the variable when you declare it
* It's a good programming practice to declare all variables at the beginning of a script.
* If you re-declare a JavaScript variable after it has already been declared and assigned, it will not lose its value.

**Declaring Multiple at Once**

* You can declare many variables in one statement.
* Start the statement with var and separate the variables by **comma**:

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

or

var person = "John Doe",

carName = "Volvo",

price = 200;

*Assigning*

**equal sign** is used to **assign values** to variables.

= is for assignment. == is for operations

**// Define Variable x**

var x;

**// Assign value to variable X**

x = 6;

*Unassigning*

**Variables**

* Any **variable** can be emptied, by setting the value to undefined. The type will also be undefined.
* In JavaScript, a variable without a value, has the value undefined. The type is also undefined.

**Values**

* An empty **value** has nothing to do with undefined.
* An empty string has both a legal value and a type.

**Objects**

* You can empty an object by setting it to null or undefined:
* undefined and null are equal in value but different in type:
* In JavaScript null is "nothing". It is supposed to be something that doesn't exist.
* Unfortunately, in JavaScript, the data type of null is an object.
* You can consider it a bug in JavaScript that typeof null is an object. It should be null.

*Naming*

**Identifiers**

* In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels).
* All JavaScript identifiers are **case sensitive**.
* Hyphens are not allowed in JavaScript. They are reserved for subtractions.
* JavaScript programmers tend to use camel case that starts with a lowercase letter:
* JavaScript uses the **Unicode** character set.
  + [Unicode Refrence](https://www.w3schools.com/charsets/ref_html_utf8.asp)

**Dollar Signs**

* Using the dollar sign is not very common in JavaScript, but professional programmers often use it as an alias for the main function in a JavaScript library.
* In the JavaScript library jQuery, for instance, the main function $ is used to select HTML elements. In jQuery $("p"); means "select all p elements".

**Underscore**

* Using the underscore is not very common in JavaScript, but a convention among professional programmers is to use it as an alias for "private (hidden)" variables.

Operators

Arithmetic Operators

|  |  |
| --- | --- |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| \*\* | Exponentiation ([ES2016](https://www.w3schools.com/js/js_es6.asp)) |
| / | Division |
| % | Modulus (Division Remainder) |
| ++ | Increment |
| -- | Decrement |

**Adding Strings and Numbers**

Adding two numbers, will return the sum, but adding a number and a string will return a string:

*Order of Operations – Priority List*

|  |  |  |  |
| --- | --- | --- | --- |
| **Value** | **Operator** | **Description** | **Example** |
| 20 | ( ) | Expression grouping | (3 + 4) |
|  |  |  |  |
| 19 | . | Member | person.name |
| 19 | [] | Member | person["name"] |
| 19 | () | Function call | myFunction() |
| 19 | new | Create | new Date() |
|  |  |  |  |
| 17 | ++ | Postfix Increment | i++ |
| 17 | -- | Postfix Decrement | i-- |
|  |  |  |  |
| 16 | ++ | Prefix Increment | ++i |
| 16 | -- | Prefix Decrement | --i |
| 16 | ! | Logical not | !(x==y) |
| 16 | typeof | Type | typeof x |
|  |  |  |  |
| 15 | \*\* | Exponentiation (ES2016) | 10 \*\* 2 |
|  |  |  |  |
| 14 | \* | Multiplication | 10 \* 5 |
| 14 | / | Division | 10 / 5 |
| 14 | % | Division Remainder | 10 % 5 |
|  |  |  |  |
| 13 | + | Addition | 10 + 5 |
| 13 | - | Subtraction | 10 - 5 |
|  |  |  |  |
| 12 | << | Shift left | x << 2 |
| 12 | >> | Shift right | x >> 2 |
| 12 | >>> | Shift right (unsigned) | x >>> 2 |
|  |  |  |  |
| 11 | < | Less than | x < y |
| 11 | <= | Less than or equal | x <= y |
| 11 | > | Greater than | x > y |
| 11 | >= | Greater than or equal | x >= y |
| 11 | in | Property in Object | "PI" in Math |
| 11 | instanceof | Instance of Object | instanceof Array |
|  |  |  |  |
| 10 | == | Equal | x == y |
| 10 | === | Strict equal | x === y |
| 10 | != | Unequal | x != y |
| 10 | !== | Strict unequal | x !== y |
|  |  |  |  |
| 9 | & | Bitwise AND | x & y |
| 8 | ^ | Bitwise XOR | x ^ y |
| 7 | | | Bitwise OR | x | y |
| 6 | && | Logical AND | x && y |
| 5 | || | Logical OR | x || y |
| 4 | ? : | Condition | ? "Yes" : "No" |
|  |  |  |  |
| 3 | += | Assignment | x += y |
| 3 | /= | Assignment | x /= y |
| 3 | -= | Assignment | x -= y |
| 3 | \*= | Assignment | x \*= y |
| 3 | %= | Assignment | x %= y |
| 3 | <<= | Assignment | x <<= y |
| 3 | >>= | Assignment | x >>= y |
| 3 | >>>= | Assignment | x >>>= y |
| 3 | &= | Assignment | x &= y |
| 3 | ^= | Assignment | x ^= y |
| 3 | |= | Assignment | x |= y |
|  |  |  |  |
| 2 | yield | Pause Function | yield x |
| 1 | , | Comma | 5 , 6 |

Assignment Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| \*= | x \*= y | x = x \* y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |
| \*\*= | x \*\*= y | x = x \*\* y |

Comparison Operators

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

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == | equal to |
| === | equal value and equal type |
| != | not equal |
| !== | not equal value or not equal type |
| > | greater than |
| < | less than |
| >= | greater than or equal to |
| <= | less than or equal to |
| ? | ternary operator |

Logical Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| && | logical and |
| || | logical or |
| ! | logical not |

Type Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| typeof | Returns the type of a variable |
| instanceof | Returns true if an object is an instance of an object type |

Bitwise Operators

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

Data Types

JavaScript has dynamic types. This means that the same variable can be used to hold different data types

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

Primitive Data

A primitive data value is a single simple data value with no additional properties and methods.

The typeof operator can return one of these primitive types:

* string
* number
* boolean
* undefined

*Strings*

A JavaScript string is zero or more characters written inside quotes.

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

Length is a property of the instance of the string object

**String as Objects**

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

Don't create strings as objects. It slows down execution speed.  
The new keyword complicates the code. This can produce some unexpected results:

*Numbers*

**Only One Type**

* JavaScript Numbers are Always 64-bit Floating Point
* Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.
* The maximum number of decimals is 17, but floating point arithmetic is not always 100% accurate:

**Numeric Strings**

JavaScript will try to convert strings to numbers in all numeric operations:

This will work:

var x = "100";

var y = "10";

var z = x / y;       // z will be 10

And this:

var x = "100";

var y = "10";

var z = x \* y;       // z will be 1000

**Illegal Numbres**

* NaN is a JavaScript reserved word indicating that a number is not a legal number.
* Trying to do arithmetic with a non-numeric string will result in NaN (Not a Number):
* You can use the global JavaScript function isNaN() to find out if a value is a number:
* Infinity (or -Infinity) is the value JavaScript will return if you calculate a number outside the largest possible number.

**Hexadecimals**

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

***Scientific Notation***

Extra large or extra small numbers can be written with scientific (exponential) notation:

var y = 123e5;      // 12300000

var z = 123e-5;     // 0.00123

*Boleans*

Booleans can only have two values: true or false.

Booleans are often used in conditional testing.

*Arrays*

JavaScript arrays are written with square brackets.

Array items are separated by commas.

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

Complex Data

*Objects*

JavaScript objects are written with curly braces {}.

Object properties are written as name:value pairs, separated by commas.

The typeof operator returns "object" for objects, arrays, and null.

The typeof operator returns "object" for arrays because in JavaScript arrays are objects.

*Functions*

The typeof operator does not return "object" for functions.

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

Type Conversion Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Original Value** | **Converted to Number** | **Converted to String** | **Converted to Boolean** |
| false | 0 | "false" | false |
| true | 1 | "true" | true |
| 0 | 0 | "0" | false |
| 1 | 1 | "1" | true |
| "0" | 0 | "0" | **true** |
| "000" | 0 | "000" | **true** |
| "1" | 1 | "1" | true |
| NaN | NaN | "NaN" | false |
| Infinity | Infinity | "Infinity" | true |
| -Infinity | -Infinity | "-Infinity" | true |
| "" | **0** | "" | **false** |
| "20" | 20 | "20" | true |
| "twenty" | NaN | "twenty" | true |
| [ ] | **0** | "" | true |
| [20] | **20** | "20" | true |
| [10,20] | NaN | "10,20" | true |
| ["twenty"] | NaN | "twenty" | true |
| ["ten","twenty"] | NaN | "ten,twenty" | true |
| function(){} | NaN | "function(){}" | true |
| { } | NaN | "[object Object]" | true |
| null | **0** | "null" | false |
| undefined | NaN | "undefined" | false |

Functions

* Function **arguments** are the **values** received by the function when it is invoked.
* Inside the function, the arguments (the parameters) behave as local variables.
* The () Operator Invokes the Function
* Accessing a function without () will return the function object instead of the function result.
* Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.
* Variables declared within a JavaScript function, become **LOCAL** to the function.
  + Local variables can only be accessed from within the function. Created and deleted within the procedure of the function
  + Since local variables are only recognized inside their functions, variables with the same name can be used in different functions

Objects

Declaring Objects

You define (and create) a JavaScript object with an object literal:

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

* Spaces and line breaks are not important. An object definition can span multiple lines:
* When a JavaScript variable is declared with the keyword "new", the variable is created as an object:
* Avoid String, Number, and Boolean objects. They complicate your code and slow down execution speed.

Properties

JavaScript objects are containers for **named values**called properties or methods.

The values are written as **name:value** pairs (name and value separated by a colon).

Methods

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

Accessing Object Properties

You can access object properties in two ways:

1. objectName.propertyName
2. objectName["propertyName"]

Accessing Object Properties

objectName.methodName()

If you access a method **without** the () parentheses, it will return the **function definition**:

“This” Keyword

<https://www.w3schools.com/js/js_this.asp>

In a function definition, this refers to the "owner" of the function.

It has different values depending on where it is used:

* In a method, this refers to the **owner object**.
* Alone, this refers to the **global object**.
* In a function, this refers to the **global object**.
* In a function, in strict mode, this is undefined.
* In an event, this refers to the **element** that received the event.
* Methods like call(), and apply() can refer this to **any object**.

Events

HTML Events

An HTML event can be something the browser does, or something a user does.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

  <element event='some JavaScript'>

  <button onclick="this.innerHTML=Date()">The time is?</button>

JavaScript code is often several lines long. It is more common to see event attributes calling functions:

[Reference of HTML DOM Events](https://www.w3schools.com/jsref/dom_obj_event.asp)v

Escape Sequences

|  |  |
| --- | --- |
| **Code** | **Result** |
| \b | Backspace |
| \f | Form Feed |
| \n | New Line |
| \r | Carriage Return |
| \t | Horizontal Tabulator |
| \v | Vertical Tabulator |

You can also break up a code line **within a text string** with a single backslash:

The \ method is not the preferred method. It might not have universal support.  
Some browsers do not allow spaces behind the \ character.

Strings

Methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

A string can be converted to an array with the split() method:

String Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| [constructor](https://www.w3schools.com/jsref/jsref_constructor_string.asp) | Returns the string's constructor function |
| [length](https://www.w3schools.com/jsref/jsref_length_string.asp) | Returns the length of a string |
| [prototype](https://www.w3schools.com/jsref/jsref_prototype_string.asp) | Allows you to add properties and methods to an object |

String Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| [charAt()](https://www.w3schools.com/jsref/jsref_charat.asp) | Returns the character at the specified index (position) |
| [charCodeAt()](https://www.w3schools.com/jsref/jsref_charcodeat.asp) | Returns the Unicode of the character at the specified index |
| [concat()](https://www.w3schools.com/jsref/jsref_concat_string.asp) | Joins two or more strings, and returns a new joined strings |
| [endsWith()](https://www.w3schools.com/jsref/jsref_endswith.asp) | Checks whether a string ends with specified string/characters |
| [fromCharCode()](https://www.w3schools.com/jsref/jsref_fromcharcode.asp) | Converts Unicode values to characters |
| [includes()](https://www.w3schools.com/jsref/jsref_includes.asp) | Checks whether a string contains the specified string/characters |
| [indexOf()](https://www.w3schools.com/jsref/jsref_indexof.asp) | Returns the position of the first found occurrence of a specified value in a string |
| [lastIndexOf()](https://www.w3schools.com/jsref/jsref_lastindexof.asp) | Returns the position of the last found occurrence of a specified value in 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 match against a regular expression, and returns the matches |
| [repeat()](https://www.w3schools.com/jsref/jsref_repeat.asp) | Returns a new string with a specified number of copies of an existing string |
| [replace()](https://www.w3schools.com/jsref/jsref_replace.asp) | Searches a string for a specified value, or a regular expression, and returns a new string where the specified values are replaced |
| [search()](https://www.w3schools.com/jsref/jsref_search.asp) | Searches a string for a specified value, or regular expression, and returns the 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 the characters from a string, beginning at a specified start position, and through the specified number of character |
| [substring()](https://www.w3schools.com/jsref/jsref_substring.asp) | Extracts the characters from a string, between two specified indices |
| [toLocaleLowerCase()](https://www.w3schools.com/jsref/jsref_tolocalelowercase.asp) | Converts a string to lowercase letters, according to the host's locale |
| [toLocaleUpperCase()](https://www.w3schools.com/jsref/jsref_tolocaleuppercase.asp) | Converts a string to uppercase letters, according to the host's locale |
| [toLowerCase()](https://www.w3schools.com/jsref/jsref_tolowercase.asp) | Converts a string to lowercase letters |
| [toString()](https://www.w3schools.com/jsref/jsref_tostring_string.asp) | Returns the value of a String object |
| [toUpperCase()](https://www.w3schools.com/jsref/jsref_touppercase.asp) | Converts a string to uppercase letters |
| [trim()](https://www.w3schools.com/jsref/jsref_trim_string.asp) | Removes whitespace from both ends of a string |
| [valueOf()](https://www.w3schools.com/jsref/jsref_valueof_string.asp) | Returns the primitive value of a String object |

String HTML Wrapper Methods

The HTML wrapper methods return the string wrapped inside the appropriate HTML tag.

These are not standard methods, and may not work as expected in all browsers.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [anchor()](https://www.w3schools.com/jsref/jsref_anchor.asp) | Creates an anchor |
| [big()](https://www.w3schools.com/jsref/jsref_big.asp) | Displays a string using a big font |
| [blink()](https://www.w3schools.com/jsref/jsref_blink.asp) | Displays a blinking string |
| [bold()](https://www.w3schools.com/jsref/jsref_bold.asp) | Displays a string in bold |
| [fixed()](https://www.w3schools.com/jsref/jsref_fixed.asp) | Displays a string using a fixed-pitch font |
| [fontcolor()](https://www.w3schools.com/jsref/jsref_fontcolor.asp) | Displays a string using a specified color |
| [fontsize()](https://www.w3schools.com/jsref/jsref_fontsize.asp) | Displays a string using a specified size |
| [italics()](https://www.w3schools.com/jsref/jsref_italics.asp) | Displays a string in italic |
| [link()](https://www.w3schools.com/jsref/jsref_link.asp) | Displays a string as a hyperlink |
| [small()](https://www.w3schools.com/jsref/jsref_small.asp) | Displays a string using a small font |
| [strike()](https://www.w3schools.com/jsref/jsref_strike.asp) | Displays a string with a strikethrough |
| [sub()](https://www.w3schools.com/jsref/jsref_sub.asp) | Displays a string as subscript text |
| [sup()](https://www.w3schools.com/jsref/jsref_sup.asp) | Displays a string as superscript text |

Numbers

methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

The toString() method returns a number as a string.

Converting to Number Type

There are 3 JavaScript methods that can be used to convert variables to numbers:

* The Number() method
* The parseInt() method
* The parseFloat() method

Number Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| MAX\_VALUE | Returns the largest number possible in JavaScript |
| MIN\_VALUE | Returns the smallest number possible in JavaScript |
| POSITIVE\_INFINITY | Represents infinity (returned on overflow) |
| NEGATIVE\_INFINITY | Represents negative infinity (returned on overflow) |
| NaN | Represents a "Not-a-Number" value |

Number Properties Cannot be Used on Variables

Number Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| [isFinite()](https://www.w3schools.com/jsref/jsref_isfinite_number.asp) | Checks whether a value is a finite number |
| [isInteger()](https://www.w3schools.com/jsref/jsref_isinteger.asp) | Checks whether a value is an integer |
| [isNaN()](https://www.w3schools.com/jsref/jsref_isnan_number.asp) | Checks whether a value is Number.NaN |
| [isSafeInteger()](https://www.w3schools.com/jsref/jsref_issafeinteger.asp) | Checks whether a value is a safe integer |
| [toExponential(x)](https://www.w3schools.com/jsref/jsref_toexponential.asp) | Converts a number into an exponential notation |
| [toFixed(x)](https://www.w3schools.com/jsref/jsref_tofixed.asp) | Formats a number with x numbers of digits after the decimal point |
| [toLocaleString()](https://www.w3schools.com/jsref/jsref_tolocalestring_number.asp) | Converts a number into a string, based on the locale settings |
| [toPrecision(x)](https://www.w3schools.com/jsref/jsref_toprecision.asp) | Formats a number to x length |
| [toString()](https://www.w3schools.com/jsref/jsref_tostring_number.asp) | Converts a number to a string |
| [valueOf()](https://www.w3schools.com/jsref/jsref_valueof_number.asp) | Returns the primitive value of a number |

Number properties belongs to the JavaScript's number object wrapper called **Number**.

These properties can only be accessed as Number.MAX\_VALUE.

Using myNumber.MAX\_VALUE, where myNumber is a variable, expression, or value, will return undefined:

Arrays

You should use objects when you want the element names to be strings (text).

You should use arrays when you want the element names to be numbers

*Arrays are Objects*

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

But, JavaScript arrays are best described as arrays.

Arrays use **numbers** to access its "elements". In this example, person[0] returns John:

*Array Elements Can Be Objects*

JavaScript variables can be objects. Arrays are special kinds of objects.

Because of this, you can have variables of different types in the same Array.

You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

*Array Properties and Methods*

*Adding at an index*

fruits[6] = "Lemon";

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

JavaScript automatically converts an array to a comma separated string when a primitive value is expected.

*Popping and Pushing*

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

This is what popping and pushing is:

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

*Sort*

When the sort() function compares two values, it sends the values to the compare function, and sorts the values according to the returned (negative, zero, positive) value.

If the result is negative a is sorted before b.

If the result is positive b is sorted before a.

If the result is 0 no changes are done with the sort order of the two values.

*Other Methods*

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

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

The splice() method can be used to add new items to an array:

With clever parameter setting, you can use splice() to remove elements without leaving "holes" in the array:

The concat() method creates a new array by merging (concatenating) existing arrays:

The concat() method does not change the existing arrays. It always returns a new array.

The concat() method can take any number of array arguments:

The slice() method slices out a piece of an array into a new array.

The slice() method can take two arguments like slice(1, 3).

If the end argument is omitted, like in the first examples, the slice() method slices out the rest of the array.

The forEach()method calls a function (a callback function) once for each array element.

The map()method creates a new array by performing a function on each array element.

The filter()method creates a new array with array elements that passes a test.

The every()method check if all array values ​​pass a test.

The reduce()method runs a function on each array element to produce (reduce it to) a single value.

The some()method check if some array values ​​pass a test.

The indexOf()method searches an array for an element value and returns its position.

Array.lastIndexOf()is the same as Array.indexOf(), but returns the position of the last occurrence of the specified element.

The find()method returns the value of the first array element that passes a test function.

The findIndex()method returns the index of the first array element that passes a test function.

JavaScript Math Object

The JavaScript Math object allows you to perform mathematical tasks on numbers.

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

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

Math Object Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| [abs(x)](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 |
| [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) |

[Complete Refernce](https://www.w3schools.com/jsref/jsref_obj_math.asp)

Switch

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

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.

When JavaScript reaches a break keyword, it breaks out of the switch block.

This will stop the execution inside the switch block.

It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway

The default case does not have to be the last case in a switch block:

If default is not the last case in the switch block, remember to end the default case with a break.

switch (new Date().getDay()) {

  case 4:

  case 5:

    text = "Soon it is Weekend";

    break;

  case 0:

  case 6:

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

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

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

Switch cases use **strict** comparison (===).

The values must be of the same type to match.

A strict comparison can only be true if the operands are of the same type.

Loops

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

For

  for (statement 1; statement 2; statement 3)

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

* Normally you will use statement 1 to initialize the variable used in the loop (i = 0).
* This is not always the case, JavaScript doesn't care. Statement 1 is optional.
* You can initiate many values in statement 1 (separated by comma):
* And you can omit statement 1 (like when your values are set before the loop starts):

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

* Often statement 2 is used to evaluate the condition of the initial variable.
* This is not always the case, JavaScript doesn't care. Statement 2 is also optional.
* If statement 2 returns true, the loop will start over again, if it returns false, the loop will end.

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

* Often statement 3 increments the value of the initial variable.
* This is not always the case, JavaScript doesn't care, and statement 3 is optional.
* Statement 3 can do anything like negative increment (i--), positive increment (i = i + 15), or anything else.
* Statement 3 can also be omitted (like when you increment your values inside the loop):

For Of

for/of lets you loop over data structures that are iterable such as Arrays, Strings, Maps, NodeLists, and more.

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.

Do While

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.

If you have read the previous chapter, about the for loop, you will discover that a while loop is much the same as a for loop, with statement 1 and statement 3 omitted.

Break & Continue

The break statement "jumps out" of a loop.

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

Labels

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

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

break labelname;

continue labelname;

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

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

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

Bitwise

JavaScript stores numbers as 64 bits floating point numbers, but all bitwise operations are performed on 32 bits binary numbers.

<https://www.w3schools.com/js/js_bitwise.asp>

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| & | AND | Sets each bit to 1 if both bits are 1 |
| | | OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | NOT | Inverts all the bits |
| << | Zero fill left shift | Shifts left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Signed right shift | Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off |
| >>> | Zero fill right shift | Shifts right by pushing zeros in from the left, and let the rightmost bits fall off |

RegExp

A regular expression is a sequence of characters that forms a **search pattern**.

A regular expression can be a single character, or a more complicated pattern.

Regular expressions can be used to perform all types of **text search** and **text replace** operations.

Using String Methods

In JavaScript, regular expressions are often used with the two **string methods**: search() and replace().

The search() method uses an expression to search for a match, and returns the position of the match.

The replace() method returns a modified string where the pattern is replaced.

**Example:**

**/w3schools/i**  is a regular expression.

**w3schools**  is a pattern (to be used in a search).

**i**  is a modifier (modifies the search to be case-insensitive).

var str = "Visit W3Schools";

var n = str.search(/w3schools/i);

Modifiers

Modifiers are used to perform case-insensitive and global searches:

|  |  |
| --- | --- |
| **Modifier** | **Description** |
| [g](https://www.w3schools.com/jsref/jsref_regexp_g.asp) | Perform a global match (find all matches rather than stopping after the first match) |
| [i](https://www.w3schools.com/jsref/jsref_regexp_i.asp) | Perform case-insensitive matching |
| [m](https://www.w3schools.com/jsref/jsref_regexp_m.asp) | Perform multiline matching |

Brackets

Brackets are used to find a range of characters:

|  |  |
| --- | --- |
| **Expression** | **Description** |
| [[abc]](https://www.w3schools.com/jsref/jsref_regexp_charset.asp) | Find any character between the brackets |
| [[^abc]](https://www.w3schools.com/jsref/jsref_regexp_charset_not.asp) | Find any character NOT between the brackets |
| [[0-9]](https://www.w3schools.com/jsref/jsref_regexp_0-9.asp) | Find any character between the brackets (any digit) |
| [[^0-9]](https://www.w3schools.com/jsref/jsref_regexp_not_0-9.asp) | Find any character NOT between the brackets (any non-digit) |
| [(x|y)](https://www.w3schools.com/jsref/jsref_regexp_xy.asp) | Find any of the alternatives specified |

Metacharacters

Metacharacters are characters with a special meaning:

|  |  |
| --- | --- |
| **Metacharacter** | **Description** |
| [.](https://www.w3schools.com/jsref/jsref_regexp_dot.asp) | Find a single character, except newline or line terminator |
| [\w](https://www.w3schools.com/jsref/jsref_regexp_wordchar.asp) | Find a word character |
| [\W](https://www.w3schools.com/jsref/jsref_regexp_wordchar_non.asp) | Find a non-word character |
| [\d](https://www.w3schools.com/jsref/jsref_regexp_digit.asp) | Find a digit |
| [\D](https://www.w3schools.com/jsref/jsref_regexp_digit_non.asp) | Find a non-digit character |
| [\s](https://www.w3schools.com/jsref/jsref_regexp_whitespace.asp) | Find a whitespace character |
| [\S](https://www.w3schools.com/jsref/jsref_regexp_whitespace_non.asp) | Find a non-whitespace character |
| [\b](https://www.w3schools.com/jsref/jsref_regexp_begin.asp) | Find a match at the beginning/end of a word, beginning like this: \bHI, end like this: HI\b |
| [\B](https://www.w3schools.com/jsref/jsref_regexp_begin_not.asp) | Find a match, but not at the beginning/end of a word |
| [\0](https://www.w3schools.com/jsref/jsref_regexp_nul.asp) | Find a NULL character |
| [\n](https://www.w3schools.com/jsref/jsref_regexp_newline.asp) | Find a new line character |
| [\f](https://www.w3schools.com/jsref/jsref_regexp_formfeed.asp) | Find a form feed character |
| [\r](https://www.w3schools.com/jsref/jsref_regexp_carriagereturn.asp) | Find a carriage return character |
| [\t](https://www.w3schools.com/jsref/jsref_regexp_tab.asp) | Find a tab character |
| [\v](https://www.w3schools.com/jsref/jsref_regexp_vtab.asp) | Find a vertical tab character |
| [\xxx](https://www.w3schools.com/jsref/jsref_regexp_octal.asp) | Find the character specified by an octal number xxx |
| [\xdd](https://www.w3schools.com/jsref/jsref_regexp_hex.asp) | Find the character specified by a hexadecimal number dd |
| [\udddd](https://www.w3schools.com/jsref/jsref_regexp_unicode_hex.asp) | Find the Unicode character specified by a hexadecimal number dddd |

Quantifiers

|  |  |
| --- | --- |
| **Quantifier** | **Description** |
| [n+](https://www.w3schools.com/jsref/jsref_regexp_onemore.asp) | Matches any string that contains at least one n |
| [n\*](https://www.w3schools.com/jsref/jsref_regexp_zeromore.asp) | Matches any string that contains zero or more occurrences of n |
| [n?](https://www.w3schools.com/jsref/jsref_regexp_zeroone.asp) | Matches any string that contains zero or one occurrences of n |
| [n{X}](https://www.w3schools.com/jsref/jsref_regexp_nx.asp) | Matches any string that contains a sequence of *X* *n*'s |
| [n{X,Y}](https://www.w3schools.com/jsref/jsref_regexp_nxy.asp) | Matches any string that contains a sequence of X to Y *n*'s |
| [n{X,}](https://www.w3schools.com/jsref/jsref_regexp_nxcomma.asp) | Matches any string that contains a sequence of at least X *n*'s |
| [n$](https://www.w3schools.com/jsref/jsref_regexp_ndollar.asp) | Matches any string with n at the end of it |
| [^n](https://www.w3schools.com/jsref/jsref_regexp_ncaret.asp) | Matches any string with n at the beginning of it |
| [?=n](https://www.w3schools.com/jsref/jsref_regexp_nfollow.asp) | Matches any string that is followed by a specific string n |
| [?!n](https://www.w3schools.com/jsref/jsref_regexp_nfollow_not.asp) | Matches any string that is not followed by a specific string n |

RegExp Object Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| [constructor](https://www.w3schools.com/jsref/jsref_regexp_constructor.asp) | Returns the function that created the RegExp object's prototype |
| [global](https://www.w3schools.com/jsref/jsref_regexp_global.asp) | Checks whether the "g" modifier is set |
| [ignoreCase](https://www.w3schools.com/jsref/jsref_regexp_ignorecase.asp) | Checks whether the "i" modifier is set |
| [lastIndex](https://www.w3schools.com/jsref/jsref_regexp_lastindex.asp) | Specifies the index at which to start the next match |
| [multiline](https://www.w3schools.com/jsref/jsref_regexp_multiline.asp) | Checks whether the "m" modifier is set |
| [source](https://www.w3schools.com/jsref/jsref_regexp_source.asp) | Returns the text of the RegExp pattern |

RegExp Object Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| [compile()](https://www.w3schools.com/jsref/jsref_regexp_compile.asp) | Deprecated in version 1.5. Compiles a regular expression |
| [exec()](https://www.w3schools.com/jsref/jsref_regexp_exec.asp) | Tests for a match in a string. Returns the first match |
| [test()](https://www.w3schools.com/jsref/jsref_regexp_test.asp) | Tests for a match in a string. Returns true or false |
| [toString()](https://www.w3schools.com/jsref/jsref_regexp_tostring.asp) | Returns the string value of the regular expression |

Modifiers

Modifiers are used to perform case-insensitive and global searches:

|  |  |
| --- | --- |
| **Modifier** | **Description** |
| [g](https://www.w3schools.com/jsref/jsref_regexp_g.asp) | Perform a global match (find all matches rather than stopping after the first match) |
| [i](https://www.w3schools.com/jsref/jsref_regexp_i.asp) | Perform case-insensitive matching |
| [m](https://www.w3schools.com/jsref/jsref_regexp_m.asp) | Perform multiline matching |

Brackets

Brackets are used to find a range of characters:

|  |  |
| --- | --- |
| **Expression** | **Description** |
| [[abc]](https://www.w3schools.com/jsref/jsref_regexp_charset.asp) | Find any character between the brackets |
| [[^abc]](https://www.w3schools.com/jsref/jsref_regexp_charset_not.asp) | Find any character NOT between the brackets |
| [[0-9]](https://www.w3schools.com/jsref/jsref_regexp_0-9.asp) | Find any character between the brackets (any digit) |
| [[^0-9]](https://www.w3schools.com/jsref/jsref_regexp_not_0-9.asp) | Find any character NOT between the brackets (any non-digit) |
| [(x|y)](https://www.w3schools.com/jsref/jsref_regexp_xy.asp) | Find any of the alternatives specified |

Metacharacters

Metacharacters are characters with a special meaning:

|  |  |
| --- | --- |
| **Metacharacter** | **Description** |
| [.](https://www.w3schools.com/jsref/jsref_regexp_dot.asp) | Find a single character, except newline or line terminator |
| [\w](https://www.w3schools.com/jsref/jsref_regexp_wordchar.asp) | Find a word character |
| [\W](https://www.w3schools.com/jsref/jsref_regexp_wordchar_non.asp) | Find a non-word character |
| [\d](https://www.w3schools.com/jsref/jsref_regexp_digit.asp) | Find a digit |
| [\D](https://www.w3schools.com/jsref/jsref_regexp_digit_non.asp) | Find a non-digit character |
| [\s](https://www.w3schools.com/jsref/jsref_regexp_whitespace.asp) | Find a whitespace character |
| [\S](https://www.w3schools.com/jsref/jsref_regexp_whitespace_non.asp) | Find a non-whitespace character |
| [\b](https://www.w3schools.com/jsref/jsref_regexp_begin.asp) | Find a match at the beginning/end of a word, beginning like this: \bHI, end like this: HI\b |
| [\B](https://www.w3schools.com/jsref/jsref_regexp_begin_not.asp) | Find a match, but not at the beginning/end of a word |
| [\0](https://www.w3schools.com/jsref/jsref_regexp_nul.asp) | Find a NULL character |
| [\n](https://www.w3schools.com/jsref/jsref_regexp_newline.asp) | Find a new line character |
| [\f](https://www.w3schools.com/jsref/jsref_regexp_formfeed.asp) | Find a form feed character |
| [\r](https://www.w3schools.com/jsref/jsref_regexp_carriagereturn.asp) | Find a carriage return character |
| [\t](https://www.w3schools.com/jsref/jsref_regexp_tab.asp) | Find a tab character |
| [\v](https://www.w3schools.com/jsref/jsref_regexp_vtab.asp) | Find a vertical tab character |
| [\xxx](https://www.w3schools.com/jsref/jsref_regexp_octal.asp) | Find the character specified by an octal number xxx |
| [\xdd](https://www.w3schools.com/jsref/jsref_regexp_hex.asp) | Find the character specified by a hexadecimal number dd |
| [\udddd](https://www.w3schools.com/jsref/jsref_regexp_unicode_hex.asp) | Find the Unicode character specified by a hexadecimal number dddd |

Quantifiers

|  |  |
| --- | --- |
| **Quantifier** | **Description** |
| [n+](https://www.w3schools.com/jsref/jsref_regexp_onemore.asp) | Matches any string that contains at least one n |
| [n\*](https://www.w3schools.com/jsref/jsref_regexp_zeromore.asp) | Matches any string that contains zero or more occurrences of n |
| [n?](https://www.w3schools.com/jsref/jsref_regexp_zeroone.asp) | Matches any string that contains zero or one occurrences of n |
| [n{X}](https://www.w3schools.com/jsref/jsref_regexp_nx.asp) | Matches any string that contains a sequence of *X* *n*'s |
| [n{X,Y}](https://www.w3schools.com/jsref/jsref_regexp_nxy.asp) | Matches any string that contains a sequence of X to Y *n*'s |
| [n{X,}](https://www.w3schools.com/jsref/jsref_regexp_nxcomma.asp) | Matches any string that contains a sequence of at least X *n*'s |
| [n$](https://www.w3schools.com/jsref/jsref_regexp_ndollar.asp) | Matches any string with n at the end of it |
| [^n](https://www.w3schools.com/jsref/jsref_regexp_ncaret.asp) | Matches any string with n at the beginning of it |
| [?=n](https://www.w3schools.com/jsref/jsref_regexp_nfollow.asp) | Matches any string that is followed by a specific string n |
| [?!n](https://www.w3schools.com/jsref/jsref_regexp_nfollow_not.asp) | Matches any string that is not followed by a specific string n |

RegExp Object Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| [constructor](https://www.w3schools.com/jsref/jsref_regexp_constructor.asp) | Returns the function that created the RegExp object's prototype |
| [global](https://www.w3schools.com/jsref/jsref_regexp_global.asp) | Checks whether the "g" modifier is set |
| [ignoreCase](https://www.w3schools.com/jsref/jsref_regexp_ignorecase.asp) | Checks whether the "i" modifier is set |
| [lastIndex](https://www.w3schools.com/jsref/jsref_regexp_lastindex.asp) | Specifies the index at which to start the next match |
| [multiline](https://www.w3schools.com/jsref/jsref_regexp_multiline.asp) | Checks whether the "m" modifier is set |
| [source](https://www.w3schools.com/jsref/jsref_regexp_source.asp) | Returns the text of the RegExp pattern |

RegExp Object Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| [compile()](https://www.w3schools.com/jsref/jsref_regexp_compile.asp) | Deprecated in version 1.5. Compiles a regular expression |
| [exec()](https://www.w3schools.com/jsref/jsref_regexp_exec.asp) | Tests for a match in a string. Returns the first match |
| [test()](https://www.w3schools.com/jsref/jsref_regexp_test.asp) | Tests for a match in a string. Returns true or false |
| [toString()](https://www.w3schools.com/jsref/jsref_regexp_tostring.asp) | Returns the string value of the regular expression |

Using the RegExp Object—RegExp Methods

In JavaScript, the RegExp object is a regular expression object with predefined properties and methods.

The test() method searches a string for a pattern, and returns true or false, depending on the result.

The exec() method  searches a string for a specified pattern, and returns the found text as an object.

Error Handling

The try statement lets you test a block of code for errors.

The catch statement lets you handle the error.

The throw statement lets you create custom errors.

The finally statement lets you execute code, after try and catch, regardless of the result.

The Error Object

JavaScript has a built in error object that provides error information when an error occurs.

The error object provides two useful properties: name and message.

Error Object Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| name | Sets or returns an error name |
| message | Sets or returns an error message (a string) |

Error Name Values

Six different values can be returned by the error name property:

|  |  |
| --- | --- |
| **Error Name** | **Description** |
| EvalError | An error has occurred in the eval() function |
| RangeError | A number "out of range" has occurred |
| ReferenceError | An illegal reference has occurred |
| SyntaxError | A syntax error has occurred |
| TypeError | A type error has occurred |
| URIError | An error in encodeURI() has occurred |

Scope

Scope determines the accessibility of variables, objects, and functions from different parts of the code.

If you assign a value to a variable that has not been declared, it will automatically become a **GLOBAL** variable.

In a web browser, global variables are deleted when you close the browser window (or tab).

Undeclared Variables

In "Strict Mode", undeclared variables are not automatically global.

Variables in Functions

Variables declared within a JavaScript function, become **LOCAL** to the function.

Function arguments (parameters) work as local variables inside functions.

Danger of Global Variables

Do NOT create global variables unless you intend to.

Your global variables (or functions) can overwrite window variables (or functions).  
Any function, including the window object, can overwrite your global variables and functions.

Global Variables in HTML

With JavaScript, the global scope is the complete JavaScript environment.

In HTML, the global scope is the window object. All global variables belong to the window object.

Hoisting

Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).

Variables defined with let and const are hoisted to the top of the block, but not initialized. Meaning: The block of code is aware of the variable, but it cannot be used until it has been declared.

JavaScript only hoists declarations, not initializations.

Variables declared with the let keyword can have Block Scope.

Redeclaring a variable inside a block will also redeclare the variable outside the block:

Variables declared with var and let are quite similar when declared inside a function. They will both have **Function Scope**.

Variables declared with var and let are quite similar when declared outside a block. They will both have **Global Scope**.

Declare Your Variables At the Top !

Hoisting is (to many developers) an unknown or overlooked behavior of JavaScript.

If a developer doesn't understand hoisting, programs may contain bugs (errors).

To avoid bugs, always declare all variables at the beginning of every scope.

Since this is how JavaScript interprets the code, it is always a good rule

Classes

Use the keyword class to create a class.

Always add a method named constructor():

* A JavaScript class is **not** an object.
* It is a **template** for JavaScript objects.
* The constructor method is a special method:
  + It has to have the exact name "constructor"
  + It is executed automatically when a new object is created
  + It is used to initialize object properties

JSON

[JSON Tutorial](https://www.w3schools.com/js/js_json_intro.asp)

* JSON stands for **J**ava**S**cript **O**bject **N**otation
* JSON is a lightweight data interchange format
* JSON is language independent **\***
* JSON is "self-describing" and easy to understand
* Curly braces hold objects
* Square brackets hold arrays

The JSON format is syntactically identical to the code for creating JavaScript objects.

Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects.

// First, create a JavaScript string containing JSON syntax:

var text = '{ "employees" : [' +

'{ "firstName":"John" , "lastName":"Doe" },' +

'{ "firstName":"Anna" , "lastName":"Smith" },' +

'{ "firstName":"Peter" , "lastName":"Jones" } ]}';

// Then, use the JavaScript built-in function JSON.parse() to convert the string into a JavaScript object:

var obj = JSON.parse(text);

// Finally, use the new JavaScript object in your page:

<p id="demo"></p>

<script>

document.getElementById("demo").innerHTML =

obj.employees[1].firstName + " " + obj.employees[1].lastName;

</script>

Improving Performance

Reduce Activity in Loops

Loops are often used in programming.

Each statement in a loop, including the for statement, is executed for each iteration of the loop.

Statements or assignments that can be placed outside the loop will make the loop run faster.

Reduce DOM Access

Accessing the HTML DOM is very slow, compared to other JavaScript statements.

If you expect to access a DOM element several times, access it once, and use it as a local variable:

Reduce DOM Size

Keep the number of elements in the HTML DOM small.

This will always improve page loading, and speed up rendering (page display), especially on smaller devices.

Every attempt to search the DOM (like getElementsByTagName) will benefit from a smaller DOM.

Avoid Unnecessary Variables

Don't create new variables if you don't plan to save values.

Often you can replace code like this:

Delay JavaScript Loading

Putting your scripts at the bottom of the page body lets the browser load the page first.

While a script is downloading, the browser will not start any other downloads. In addition all parsing and rendering activity might be blocked.

The HTTP specification defines that browsers should not download more than two components in parallel.

An alternative is to use defer="true" in the script tag. The defer attribute specifies that the script should be executed after the page has finished parsing, but it only works for external scripts.

If possible, you can add your script to the page by code, after the page has loaded:

Avoid Using with

Avoid using the with keyword. It has a negative effect on speed. It also clutters up JavaScript scopes.

The with keyword is **not allowed** in strict mode.