JavaScript Glossary

Programming reference for JavaScript

Arrays

Accessing array elements

You can get elements out of arrays if you know their index. Array elements' indexes start at 0 and increment by 1, so the first element's index is 0, the second element's index is 1, the third element's is 2, etc.

Syntax

```
array[index]
```

Example

```
var primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37];
primes[0]; // 2
primes[3]; // 7
primes[150]; // undefined
```

Array literals

You can create arrays in two different ways. The most common of which is to list values in a pair of square brackets. JavaScript arrays can contain any types of values and they can be of mixed types.

Syntax

```
var arrayName = [element0, element1, ..., elementN]
```

Example

```
var primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37];
```

Read more

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array
- http://www.javascripter.net/faq/creatingarrays.htm

Multi-dimensional Arrays

A two-dimensional array is an array within an array. If you fill this array with another array you get a three-dimensional array and so on.

Example

```
var multidimensionalArray = [[1,2,3],[4,5,6],[7,8,9]] // two dimensions, 3x3
```

Array constructor

You can also create an array using the Array constructor.

Example

```
var stuff = new Array();

stuff[0] = 34;
stuff[4] = 20;

stuff    // [34, undefined, undefined, undefined, 20]
```

Example

```
var myArray = new Array(45 , "Hello World!" , true , 3.2 , undefined);
console.log(myArray);

// output: [ 45, 'Hello World!', true, 3.2, undefined ]
```

Read more

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#Example.3A_Creating_an_array

Accessing nested array elements

Accessing multi dimensional array elements is quite similar to one-dimension arrays . They are accessed by using [index][index]..... (number of them depends upon the number of arrays deep you want to go inside).

Syntax

```
array[index][index]....
```

Example

```
var myMultiArray = [
            [1,2,3,4,5, [1,2,3,4,5] ],
            [6,7,8,9,10 , [1,2,3,4,6] ],
            [11,12,13,14,15 , [1,2,3,4,5] ],
            [16,17,18,19,20, [1,2,3,4,5] ]
            ];

console.log( myMultiArray[1][5][4] ); // Outputs 6 , the value in the last element of the last element of the second element of myMultiArray.
```

Booleans

Boolean literals

Syntax

```
true
false
```

Boolean logical operators

Syntax

```
expression1 && expression2 //returns true if both the expressions evaluate to true

expression3 || expression4 // return true if either one of the expression evaluates to true

!expression5 // returns the opposite boolean value of the expression
```

Example

```
if ( true && false )alert("Not executed!");
//because the second expression is false

if( false || true )alert("Executed!");
//because any one of the expression is true

if( !false )alert("Executed!");
// because !false evaluates to true

!!true // remains true
```

Example

```
if(!false && ( false || (false && true) ))alert("Guess what...");

/* not executed because
!false && ( false || (false && true) ) - becomes
!false && ( false || false) - becomes
true && false , which is false.*/
```

Example

```
/* An important thing to note here is the Operator Precedence - which determines the order in which operators are evaluated. Operators with higher precede

// Brackets - have the highest precedence
// ! - lower than Brackets
// && - lower than !
// || - the lowest

if(true && !!false || true)alert("Guess again ??");

/* Executed , here is the evaluation process-
true && !!false || true - becomes
true && false || true - (no brackets present , so ! evaluated ) becomes
false || true - (then && evaluated) which becomes true */
```

Example

```
/* Next important thing is the Associativity - which determines the order in which operators of the same precedence are processed. For example, consider a

// Brackets , && , || have left to right associativity
// ! has right to left associativity
// So ,

!false && !!false //false
// evaluated in the manner - !false && false - true && false - false
```

Comparison operators

Syntax

```
x === y // returns true if two things are equal
x !== y // returns true if two things are not equal
x <= y // returns true if x is less than or equal to y
x >= y // returns true if x is greater than or equal to y
x < y // returns true if x is less than y
x > y // returns true if x is greater than y
```

"Truthy" and "Falsey"

Only Boolean literals (true and false) assert truth or false, but there are some other ways too to derive true or false. Have a look at the examples.

Example

```
if(1)console.log("True!"); // output True! , since any non-zero number is considered to be true

if(0)console.log("I doubt if this gets executed"); // not executed , since 0 is considered to be false

if("Hello")alert("So, any non-empty String is also true."); //Gets executed

if("")alert("Hence , an empty String is false"); // Not executed
Read more
http://www.sitepoint.com/javascript-truthy-falsy/
```

== VS. ===

A simple explanation would be that == does just value checking (no type checking) , whereas , === does both value checking and type checking . Seeing the examples may make it all clear. It is always advisable that you never use == , because == often produces unwanted results

Syntax

```
expression == expression
expression === expression
```

Example

```
'1' == 1 //true (same value)
'1' === 1 // false  (not the same type)

true == 1 // true (because 1 stands for true ,though it's not the same type)
true === 1 // false (not the same type)
```

Code Comments

Code comments are used for increasing the readability of the code.If you write 100 lines of code and then forget what each function did , it's not useful at all. Comments are like notes , suggestions , warnings ,etc. that you can put for yourself. Code comments are not executed

Single Line Comment

Anything on the line following `//` will be a comment while anything before will still be code.

Syntax

```
console.log("This code will be run")
//console.log("Because this line is in a comment, this code will not be run.")
// This is a single line comment.
```

Multi-Line Comment

Anything between `/*` and `*/` will be a comment.

Syntax

```
/* This is
   a multi-line


   comment!
*/
```

Example

```
/*
alert("Hello,I won't be executed.");
console.log("Hello ,I also will not be executed");
*/
```

Console

console.log

Prints text to the console. Useful for debugging.

Example

```
var name = "Codecademy";
console.log(name);
```

console.time

This function starts a timer which is useful for tracking how long an operation takes to happen.You give each timer a unique name, and may have up to 10,000 timers running on a given page.When you call `console.timeEnd()` with the same name, the browser will output the time, in milliseconds, that elapsed since the timer was started.

Syntax

```
console.time(timerName);
```

Example

```
console.time("My Math");
var x = 5 + 5;
console.log(x);
console.timeEnd("My Math");
console.log("Done the math.");

/* Output:
10
My Math: (time taken)
Done the math.
*/
```

Read more

- https://developer.mozilla.org/en-US/docs/Web/API/console.time
- https://developer.mozilla.org/en-US/docs/Web/API/console.timeEnd

console.timeEnd

Stops a timer that was previously started by calling `console.time()`.

Syntax

```
console.timeEnd(timerName);
```

Example

```
console.time("My Math");
var x = 5 + 5;
console.log(x);
console.timeEnd("My Math");

/* Output :
10
My Math: (time taken)
*/
```

Read more

- https://developer.mozilla.org/en-US/docs/Web/API/console.timeEnd

Functions

A function is a JavaScript procedure—a set of statements that performs a task or calculates a value.It is like a reusable piece of code. Imagine , having 20 for loops ,and then having a single function to handle it all . To use a function, you must define it somewhere in the scope from which you wish to call it. A function definition (also called a function declaration) consists of the function keyword, followed by the name of the function, a list of arguments to the function, enclosed in parentheses and separated by commas, the JavaScript statements that define the function, enclosed in curly braces, `{ }`.

Syntax

```
function name(argument1 , argument2 .... argumentN){
    statement1;
    statement2;
    ..
    ..
    statementN;
}
```

Example

```
function greet(name) {
  return "Hello" + name + "!";
}
```

Read more

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions

Function calling

Syntax

```
functionName(argument1, argument2, ..., argumentN);
```

Example

```
greet("Anonymous");
// Hello Anonymous!
```

Function hoisting

The two ways of declaring functions produce different results. Declaring a function one way "hoists" it to the top of the call, and makes it available before it's actually defined.

Example

```
hoistedFunction(); // Hello! I am defined immediately!
notHoistedFunction(); // ReferenceError: notHoistedFunction is not defined

function hoistedFunction () {
  console.log('Hello! I am defined immediately!');
}

var notHoistedFunction = function () {
  console.log('I am not defined immediately.');
}
```

Read more

- http://jamesallardice.com/explaining-function-and-variable-hoisting-in-javascript/

If statement

It simply states that if this condition is true, do this, else do something else (or nothing). It occurs in varied forms.

if

Syntax

```
// Form : Single If
if (condition) {
  // code that runs if the condition is true
}
```

Example

```
if (answer === 42) {
  console.log('Told you so!');
}
```

else

A fallback to an `if` statement. This will only get executed if the previous statement did not.

Syntax

```
// If the condition is true, statement1 will be executed.
// Otherwise, statement2 will be executed.

if (condition) {
  // statement1: code that runs if condition is true
} else {
  // statement2: code that runs if condition is false
}
```

Example

```
if (gender == "male") {
  console.log("Hello, sir!");
} else {
  console.log("Hello, ma'am!");
}
```

else if

This is like an `else` statement, but with its own condition. It will only run if its condition is true, and the previous statement's condition was false.

Syntax

```
// Form : else if . If the condition is true, statement1 will be executed. Otherwise, condition2 is checked . if it is true , then statement2 is executed.
if (condition1) {
  statement1;
} else if (condition2) {
  statement2;
} else {
  statement3;
}
```

Example

```
if (someNumber > 10) {
  console.log("Numbers larger than 10 are not allowed.");
} else if (someNumber < 0) {
  console.log("Negative numbers are not allowed.");
} else {
  console.log("Nice number!");
}
```

Loops

For Loops

You use `for` loops, if you know how often you'll loop. The most often used varName in loops is `i`.

Syntax

```
for ([var i = startValue];[i < endValue]; [i+=stepValue]) {
    // Your code here
}
```

Example

```
for (var i = 0; i < 5; i++) {
  console.log(i); // Prints the numbers from 0 to 4
}
```

Example

```
var i;  // "outsourcing" the definition
for (i = 10; i >= 1; i--) {
    console.log(i); // Prints the numbers from 10 to 1
}
```

Example

```
/* Note that all of the three statements are optional, i.e. , */
var i = 9;
for(;;){
    if(i === 0)break;
    console.log(i);
    i--;
}

//This loop is perfectly valid.
```

While Loops

You use `while` loops, if you don't know how often you'll loop.

Syntax

```
while (condition) {
  // Your code here
}
```

Example

```
var x = 0;
while (x < 5) {
  console.log(x); // Prints numbers from 0 to 4
  x++;
}
```

Example

```
var x = 10;
while (x <= 5) {
    console.log(x); // Won't be executed
    x++;
}
```

Read more

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/while

Do While Loops

You use do while loops, if you have to loop at least once, but if you don't know how often.

Syntax

```
do {
  // Your code here
} while (condition);
```

Example

```
var x = 0;
do {
    console.log(x);  // Prints numbers from 0 to 4
    x++;
} while (x < 5);
```

Example

```
var x = 10;
do {
    console.log(x); // Prints 10
    x++;
} while (x <= 5);
```

Read more

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/do...while

Math

random

Returns a random number between 0 and 1.

Syntax

```
Math.random()
```

Example

```
Math.random(); // A random number between 0 and 1.
```

floor

Returns the largest integer less than or equal to a number.

Syntax

```
Math.floor(expression)
```

Example

```
Math.floor(9.99); // 9
Math.floor(1 + 0.5); // 1
Math.floor(Math.random() * X + 1); // Returns a random number between 1 and X
```

pow

Returns base raised to exponent.

Syntax

```
Math.pow(base,exponent)
```

Example

```
Math.pow(2,4); // gives 16
```

Read more

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/pows

ceil

Returns the smallest integer greater than or equal to a number.

Syntax

```
Math.ceil(expression)
```

Example

```
Math.ceil(45.4); //  46
Math.ceil(4 - 1.9); //  3
```

Read more

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/ceil

PI

Returns the ratio of the circumference of a circle to its diameter, approximately 3.14159 or in better terms, the value of PI (π). Note in syntax , we do not put `()` at the end of `Math.PI` because `Math.PI` is not a function.

Syntax

```
Math.PI
```

Example

```
Math.round(Math.PI); // rounds the value of PI ,gives 3
Math.ceil(Math.PI); // 4
```

Read more

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/PI

sqrt

Returns the square root of a number.

Syntax

```
Math.sqrt(expression)
```

Example

```
Math.sqrt(5+4); // 3
Math.sqrt(Math.sqrt(122+22) + Math.sqrt(16)); //4
```

Read more

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/sqrt

Numbers

% (Modulus)

It returns the remainder left after dividing the left hand side with the right hand side.

Syntax

```
number1 % number2
```

Example

```
14 % 9 // returns 5
```

isNaN

Returns true if the given number is not a number, else returns false.

Syntax

```
isNaN([value])
```

Example

```
var user_input = prompt("Enter a number"); // Enter "a number"

if(isNaN(user_input))
    alert("I told you to enter a number.");

//alert executed , since "a number" is not a number

//Another important thing:

if( isNaN("3") )
    alert("bad");

//Not executed , because the string "3" gets converted into 3 ,and 3 is a number
```

Basic Arithmetic

Doing basic arithmetic is simple.

Syntax

```
4 + 5;  // 9
4 * 5;  // 20
5 - 4;  // 1
20 / 5; // 4
```

Prefix and Postfix increment/decrement operators

Prefix increment / decrement operators are operators that first increase the value of the variable by 1 (increment) or decrease the value of an expression / variable by 1 (decrement) and then return this incremented / decremented value. They are used like ++(variable) [increment] or --(varaible) [decrement] On the other hand , Postfix increment / decrement operators are operators that first return the value of the variable and then increase the value of that variable by 1 (increment) or decrease the value

of the variable by 1 (decrement) . They are used like (variable)++ [increment] or (varaible)-- [decrement]

Syntax

```
--variable   //Prefix Decrement
++variable   //Prefix Increment
variable--   //Postfix Decrement
variable++  //Postfix Increment
```

Example

```
//The examples will make it clear

var x = 15; // x has a value of 15
var y = x++;
// since it is postfix , the value of x (15) is first assigned to y and then the value of x is incremented by 1
console.log(y); //15
console.log(x); //16

var a = 15; // a has a value of 15
var b = ++a;
// since it is prefix , the value of a (15) is first incremented by 1 and then the value of x is assigned to b
console.log(b); //16
console.log(a); //16
```

Objects

Object Literals

Syntax

```
{
  "property 1": value1,
  property2: value2,
  number: value3
}
```

Example

```
var obj = {
  name: "Bob",
  married: true,
  "mother's name": "Alice",
  "year of birth": 1987,
  getAge: function () {
    return 2012 - obj["year of birth"];
  },
  1: 'one'
};
```

Property Access

Syntax

```
name1[string]
name2.identifier
```

Example

```
obj['name'];  // 'Bob'
obj.name;     // 'Bob'
obj.getAge(); // 24
```

OOP

Classes

A class can be thought of as a template to create many objects with similar qualities. Classes are a fundamental component of object-oriented programming (OOP).

Syntax

```
SubClass.prototype = new SuperClass();
```

Example

```
var Lieutenant = function (age) {
  this.rank = "Lieutenant";
  this.age = age;
};

Lieutenant.prototype = new PoliceOfficer();

Lieutenant.prototype.getRank = function () {
  return this.rank;
};

var John = new Lieutenant(67);

John.getJob(); // 'Police Officer'
```

```
John.getRank(); // 'Lieutenant'
John.retire(); // true
```

Popup boxes

alert

Display an alert dialog with the specified message and an OK button. The alert dialog should be used for messages which do not require any response on the part of the user, other than the acknowledgement of the message.

Syntax

```
alert(message);
```

Example

```
alert("Hello World");
```

confirm

Displays a dialog with the specified message and two buttons, OK and Cancel.

Syntax

```
confirm("message") //returns true if confirmed, false otherwise
```

Example

```
if ( confirm("Are you sure you want to delete this post?") ) {
  deletePost();
}
```

Read more

- https://developer.mozilla.org/en-US/docs/Web/API/window.confirm

prompt

The `prompt()` displays a dialog with an optional message prompting the user to input some text. If the user clicks the "Cancel" button, null is returned.

Syntax

```
prompt(message);
```

Example

```
var name = prompt("Enter your name:");
console.log("Hello " + name + "!");
```

Read more

- https://developer.mozilla.org/en-US/docs/Web/API/window.prompt

Strings

Strings are text. They are denoted by surrounding text with either single or double quotes.

Syntax

```
"string of text"
'string of text'
```

Concatenation

Syntax

```
string1 + string2
```

Example

```
"some" + "text"; // returns "sometext"
var first = "my";
var second = "string";
var union = first + second; // union variable has the string "mystring"
```

length

Returns the length of the string.

Syntax

```
string.length
```

Example

```
"My name".length // 7 , white space is also counted
"".length // 0
```

toUpperCase(), toLowerCase()

Changes the cases of all the alphabetical letters in the string.

Example

```
"my name".toUpperCase(); // Returns "MY NAME"
"MY NAME".toLowerCase(); // Returns "my name"
```

trim()

Removes whitespace from both ends of the string.

Syntax

```
string.trim()
```

Example

```
"       a        ".trim(); // 'a'
"     a  a       ".trim(); // 'a  a'
```

Read more

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/Trim

replace()

Returns a string with the first match substring replaced with a new substring.

Example

```
"original string".replace("original", "replaced"); // returns "replaced string"
```

charAt()

Returns the specified character from a string. Characters in a string are indexed from left to right. The index of the first character is 0, and the index of the last character in a string called stringName is `stringName.length - 1`. If the index you supply is out of range, JavaScript returns an empty string.

Syntax

```
string.charAt(index) // index is an integer between 0 and 1 less than the length of the string.
```

Example

```
"Hello World!".charAt(0); // 'H'
"Hello World!".charAt(234); // ''
```

Read more

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/charAt

substring()

Returns the sequence of characters between two indices within a string.

Syntax

```
string.substring(indexA[, indexB])
//indexA : An integer between 0 and the length of the string
// indexB : (optional) An integer between 0 and the length of the string.
```

Example

```
"adventures".substring(2,9); // Returns "venture"
// It starts from indexA(2) , and goes up to but not including indexB(9)
"hello".substring(1); // returns "ello"
"Web Fundamentals".substring(111); // returns ''
"In the market".substring(2,999); // returns ' the market'
"Fast and efficient".substring(3,3); // returns ''
"Go away".substring("abcd" , 5); // returns 'Go aw'
// Any non-numeric thing is treated as 0
```

indexOf()

Returns the index within the calling String object of the first occurrence of the specified value, starting the search at `fromIndex`, Returns `-1` if the value is not found. The `indexOf` method is case sensitive.

Syntax

```
string.indexOf(searchValue[, fromIndex]) //fromIndex is optional.It specifies from which index should the search start.Its default value is 0.
```

Example

```
"My name is very long.".indexOf("name"); // returns 3
"My name is very long.".indexOf("Name"); // returns -1 , it's case sensitive
"Where are you going?".indexOf("are",11); //returns -1
"Learn to Code".indexOf(""); //returns 0
"Learn to Code".indexOf("",3); //returns 3
"Learn to Code".indexOf("",229); returns 13 , which is the string.length
```

Read more

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/indexOf

Switch statements

Acts like a big if / else if / else chain. Checks a value against a list of cases, and executes the first case that is true. It goes on executing all other cases it finds after the first true case till it finds a breaking statement,after which it breaks out of the switch If it does not find any matching case , it executes the default case.

Syntax

```
switch (expression) {
  case label1:
    statements1
    [break;]
  case label2:
    statements2
    [break;]
  ...
  case labelN:
    statementsN
    [break;]
  default:
    statements_def
    [break;]
}
```

Example

```
var gender = "female";

switch (gender) {
  case "female":
    console.log("Hello, ma'am!");
  case "male":
    console.log("Hello, sir!");
  default:
    console.log("Hello!");
}
```

Ternary Operator

The ternary operator is usually used as a shortcut for the if statement.

Syntax

```
condition ? expr1 : expr2
```

Example

```
var grade = 85;
console.log("You " + (grade > 50 ? "passed!" : "failed!"));

//Output: You passed!

/* The above statement is same as saying:
if(grade > 50){
    console.log("You " + "passed!");  //or simply "You passed!"
}
else{
    console.log("You " + "failed!");
}
*/
```

Variables

Variable Assignment

Syntax

```
var name = value;
```

Example

```
var x = 1;
var myName = "Bob";
var hisName = myName;
```

```
Variable changing
```

Syntax

```
varname = newValue
```

Example

```
var name = "Michael" //declare variable and give it value of "Michael"
name = "Samuel" //change value of name to "Samuel"
```

Teaching the world how to code.