# JavaScript Variables

Variable means anything that can vary. In JavaScript, a variable stores data that can be changed later on.

## **Declare a Variable**

In JavaScript, a variable can be declared using var, let, const keywords.

* **var** keyword is used to declare variables since JavaScript was created. It is confusing and error-prone when using variables declared using var.
* **let** keyword removes the confusion and error of var. It is the new and recommended way of declaring variables in JavaScript.
* **const** keyword is used to declare a constant variable that cannot be changed once assigned a value.

Here, we will use the let keyword to declare variables. To declare a variable, write the keyword let followed by the name of the variable you want to give, as shown below.

Example: Variable Declaration

let msg; // declaring a variable without assigning a value

In the above example, var msg; is a variable declaration. It does not have any value yet. The default value of variables that do not have any value is [undefined](https://www.tutorialsteacher.com/javascript/javascript-null-and-undefined).

You can assign a value to a variable using the = operator when you declare it or after the declaration and before accessing it.

Example: Variable Initialization

let msg;

msg = "Hello JavaScript!"; // assigning a string value

In the above example, the msg variable is declared first and then assigned a [string](https://www.tutorialsteacher.com/javascript/javascript-string) value in the next line.

You can declare a variable and assign a value to it in the same line. Values can be of any [datatype](https://www.tutorialsteacher.com/javascript/javascript-data-types) such as [string](https://www.tutorialsteacher.com/javascript/javascript-string), [numeric](https://www.tutorialsteacher.com/javascript/javascript-number), [boolean](https://www.tutorialsteacher.com/javascript/javascript-boolean), etc.

Example: Variable Declaration and Initialization

let name = "Steve"; //assigned string value

let num = 100; //assigned numeric value

let isActive = true; //assigned boolean value

Multiple variables can be declared in a single line, as shown below.

Example: Multiple Variables

let name = "Steve", num = 100, isActive = true;

You can copy the value of one variable to another variable, as shown below.

Example: Copy Variable

let num1 = 100;

let num2 = num1;

JavaScript allows multiple white spaces and line breaks when you declare a variables.

Example: Whitespace and Line Breaks

let name = "Steve",

num = 100,

isActive = true;

Variable names are case-sensitive in JavaScript. You cannot declare a duplicate variable using the let keyword with the same name and case. JavaScript will throw a syntax error. Although, variables can have the same name if declared with the var keyword (this is why it is recommended to use let).

Example: Syntax Error

let num = 100;

let num = 200; //syntax error

var num = 100;

var num = 200; //Ok

## **JavaScript Variable Nameing Conventions**

* Variable names are case-sensitive in JavaScript. So, the variable names msg, MSG, Msg, mSg are considered separate variables.
* Variable names can contain letters, digits, or the symbols $ and \_.
* A variable name cannot start with a digit 0-9.
* A variable name cannot be a reserved keyword in JavaScript, e.g. var, function, return cannot be variable names.

## **Dynamic Typing**

JavaScript is a loosely typed language. It means that you don't need to specify what data type a variable will contain. You can update the value of any type after initialization. It is also called dynamic typing.

Example: Loosely Typed Variable

let myvariable = 1; // numeric value

myvariable = 'one'; // string value

myvariable = 1.1; // decimal value

myvariable = true; // Boolean value

myvariable = null; // null value

## **Constant Variables in JavaScript**

Use const keyword to declare a constant variable in JavaScript.

* Constant variables must be declared and initialized at the same time.
* The value of the constant variables can't be changed after initialized them.

Example: Constant Variables

const num = 100;

num = 200; //error

const name; //error

name = "Steve";

The value of a constant variable cannot be changed but the content of the value can be changed. For example, if an object is assigned to a const variable then the underlying value of an object can be changed.

Example: Constant Variables

const person = { name: 'Steve'};

person.name = "Bill";

alert(person.name); //Bill

It is best practice to give constant variable names in capital letters to separate them from other non-constant variables.

## **Variable Scope**

In JavaScript, a variable can be declared either in the global scope or the local scope.

### **Global Variables**

Variables declared out of any function are called global variables. They can be accessed anywhere in the JavaScript code, even inside any function.

### **Local Variables**

Variables declared inside the function are called local variables of that function. They can only be accessed in the function where they are declared but not outside.

The following example includes global and local variables.

Example: Global and Local Variable

let greet = "Hello " // global variable

function myfunction(){

let msg = "JavaScript!";

alert(greet + msg); //can access global and local variable

}

myfunction();

alert(greet);//can access global variable

alert(msg); //error: can't access local variable

Learn [global and local scope in JavaScript](https://www.tutorialsteacher.com/javascript/scope-in-javascript) for more information.

## **Declare Variables without var and let Keywords**

Variables can be declared and initialized without the var or let keywords. However, a value must be assigned to a variable declared without the var keyword.

The variables declared without the var keyword become global variables, irrespective of where they are declared. Visit [Variable Scope in JavaScript](https://www.tutorialsteacher.com/javascript/scope-in-javascript) to learn about it.

It is Recommended to declare variable using the let keyword.

Example: Variable Declaration Without var or let Keyword

function myfunction(){

msg = "Hello JavaScript!";

}

myfunction();

alert(msg); // msg becomes global variable so can be accessed here

# Scope in JavaScript

Scope in JavaScript defines accessibility of variables, objects and functions.

There are two types of scope in JavaScript.

1. Global scope
2. Local scope

## **Global Scope**

Variables declared outside of any function become global variables. Global variables can be accessed and modified from any function.

Example: Global Variable

<script>

var userName = "Bill";

function modifyUserName() {

userName = "Steve";

};

function showUserName() {

alert(userName);

};

alert(userName); // display Bill

modifyUserName();

showUserName();// display Steve

</script>

In the above example, the variable userName becomes a global variable because it is declared outside of any function. A modifyUserName() function modifies userName as userName is a global variable and can be accessed inside any function. The same way, showUserName() function displays current value of userName variable. Changing value of global variable in any function will reflect throughout the program.

Please note that variables declared inside a function without **var** keyword also become global variables.

Example: Global Variable

<script>

function createUserName() {

userName = "Bill";

}

function modifyUserName() {

if(userName)

userName = "Steve";

};

function showUserName() {

alert(userName);

}

createUserName();

showUserName(); // Bill

modifyUserName();

showUserName(); // Steve

</script>

In the above example, variable userName is declared without **var** keyword inside createUserName(), so it becomes global variable automatically after calling createUserName() for the first time.

 Note:

A userName variable will become global variable only after createUserName() is called at least once. Calling showUserName() before createUserName() will throw an exception "userName is not defined".

## **Local Scope**

Variables declared inside any function with **var** keyword are called local variables. Local variables cannot be accessed or modified outside the function declaration.

Example: Local Scope

<script>

function createUserName() {

var userName = "Bill";

}

function showUserName() {

alert(userName);

}

createUserName();

showUserName(); // throws error: userName is not defined

</script>

Function parameters are considered as local variables.

In the above example, userName is local to createUserName() function. It cannot be accessed in showUserName() function or any other functions. It will throw an error if you try to access a variable which is not in the local or global scope. Use try catch block for exception handling.

**Some tips..**

If local variable and global variable have same name then changing value of one variable does not affect on the value of another variable.

Example: Scope

var userName = "Bill";

function ShowUserName()

{

var userName = "Steve";

alert(userName); // "Steve"

}

ShowUserName();

alert(userName); // Bill

JavaScript does not allow block level scope inside { }. For example, variables defined in if block can be accessed outside if block, inside a function.

Example: No Block Level Scope

Function NoBlockLevelScope(){

if (1 > 0)

{

var myVar = 22;

}

alert(myVar);

}

NoBlockLevelScope();

Javascript Operators

JavaScript includes operators same as other languages. An operator performs some operation on single or multiple operands (data value) and produces a result. For example, in 1 + 2, the + sign is an operator and 1 is left side operand and 2 is right side operand. The + operator performs the addition of two numeric values and returns a result.

JavaScript includes following categories of operators.

## **Arithmetic Operators**

Arithmetic operators are used to perform mathematical operations between numeric operands.

|  |  |
| --- | --- |
| operator | Description |
| + | Adds two numeric operands. |
| - | Subtract right operand from left operand |
| \* | Multiply two numeric operands. |
| / | Divide left operand by right operand. |
| % | Modulus operator. Returns remainder of two operands. |
| ++ | Increment operator. Increase operand value by one. |
| -- | Decrement operator. Decrease value by one. |

Example of arthimetic operator program:

let x = 5, y = 10;

let z = x + y; //performs addition and returns 15

z = y - x; //performs subtraction and returns 5

z = x \* y; //performs multiplication and returns 50

z = y / x; //performs division and returns 2

z = x % 2; //returns division remainder 1

Increment and decrement operator:

* The ++ and -- operators are unary operators.
* It works with either left or right operand only. When used with the left operand, e.g., x++, it will increase the value of x when the program control goes to the next statement.
* In the same way, when it is used with the right operand, e.g., ++x, it will increase the value of x there only.
* Therefore, x++ is called post-increment, and ++x is called pre-increment.

Example of increment and decrement operator program:

let x = 5;

x++; //post-increment, x will be 5 here and 6 in the next line

++x; //pre-increment, x will be 7 here

x--; //post-decrement, x will be 7 here and 6 in the next line

--x; //pre-decrement, x will be 5 here

### **String Concatenation**

* The + operator performs concatenation operation when one of the operands is of string type.
* The following example demonstrates string concatenation even if one of the operands is a string.

**Example of operator with string:**

let a = 5, b = "Hello ", c = "World!", d = 10;

a + b; //returns "5Hello "

b + c; //returns "Hello World!"

a + d; //returns 15

b + true; //returns "Hello true"

c - b; //returns NaN; - operator can only used with numbers

Comparison Operator:

* JavaScript provides comparison operators that compare two operands and return a boolean value true or false.

|  |  |
| --- | --- |
| operator | Description |
| == | Compares the equality of two operands without considering type. |
| === | Compares equality of two operands with type. |
| != | Compares inequality of two operands. |
| > | Returns a boolean value true if the left-side value is greater than the right-side value; otherwise, returns false. |
| < | Returns a boolean value true if the left-side value is less than the right-side value; otherwise, returns false. |
| >= | Returns a boolean value true if the left-side value is greater than or equal to the right-side value; otherwise, returns false. |
| <= | Returns a boolean value true if the left-side value is less than or equal to the right-side value; otherwise, returns false. |

Example of comparison operator program:

let a = 5, b = 10, c = "5";

let x = a;

a == c; // returns true

a === c; // returns false

a == x; // returns true

a != b; // returns true

a > b; // returns false

a < b; // returns true

a >= b; // returns false

a <= b; // returns true

## **Logical Operators**

* In JavaScript, the logical operators are used to combine two or more conditions. JavaScript provides the following logical operators.

|  |  |
| --- | --- |
| operator | Description |
| && | && is known as AND operator. It checks whether two operands are non-zero or not (0, false, undefined, null or "" are considered as zero). It returns 1 if they are non-zero; otherwise, returns 0. |
| || | || is known as OR operator. It checks whether any one of the two operands is non-zero or not (0, false, undefined, null or "" is considered as zero). It returns 1 if any one of of them is non-zero; otherwise, returns 0. |
| ! | ! is known as NOT operator. It reverses the boolean result of the operand (or condition). !false returns true, and !true returns false. |

Example of logical operator program:

let a = 5, b = 10;

(a != b) && (a < b); // returns true

(a > b) || (a == b); // returns false

(a < b) || (a == b); // returns true

!(a < b); // returns false

!(a > b); // returns true

## **‘**

## **Assignment Operators**

* JavaScript provides the assignment operators to assign values to variables with less key strokes.

|  |  |  |  |
| --- | --- | --- | --- |
| operator | Description |  | Description |
| = | Assigns right operand value to the left operand. |  |  |
| += | Sums up left and right operand values and assigns the result to the left  operand. |  |  |
| -= | Subtract right operand value from the left operand value and  assigns the result to the leftoperand. |  |  |
| \*= | Multiply left and right operand values and assigns the result to  the left operand |  |  |
| /= | Divide left operand value by right operand value and assign  the result to the left operand. |  |  |
| %= | Get the modulus of left operand divide by right operand and assign  resulted modulus to the left operand. |  |  |

Example of Assingnment operator program:

let x = 5, y = 10, z = 15;

x = y; //x would be 10

x += 1; //x would be 6

x -= 1; //x would be 4

x \*= 5; //x would be

x /= 5; //x would be 1

x %= 2; //x would be 1

**Ternary operator:**

* JavaScript provides a special operator called ternary operator :? that assigns a value to a variable based on some condition.
* This is the short form of the [if else condition](https://www.tutorialsteacher.com/javascript/javascript-if-else-condition).

**Syntax:**

<condition> ? <value1> : <value2>;

* The ternary operator starts with conditional expression followed by the ? operator.
* The second part (after ? and before :) will be executed if the condition turns out to be true. Suppose, the condition returns false, then the third part (after :) will be executed.

**Example of ternary operator program:**

let a = 10, b = 5;

let c = a > b? a : b; // value of c would be 10

let d = a > b? b : a; // value of d would be 5

Javascript datatypes:

let myvariable = 1; // numeric value

myvariable = 'one'; // string value

myvariable = true; // Boolean value

* different types of values are assigned to the same variable to demonstrate the loosely typed characteristics of JavaScript.
* Here, 1 is the number type, 'one' is the string type, and true is the boolean type.

## **Primitive Data Types**

The primitive data types are the lowest level of the data value in JavaScript. The followings are primitive data types in JavaScript:

| Data Type |  | Description |
| --- | --- | --- |
| [String](https://www.tutorialsteacher.com/javascript/javascript-string) |  | String is a textual content wrapped inside ' ' or " " or ` ` (tick sign).  **Example:** 'Hello World!', "This is a string", etc. |
| [Number](https://www.tutorialsteacher.com/javascript/javascript-number) |  | Number is a numeric value.  **Example:** 100, 4521983, etc. |
| BigInt |  | BigInt is a numeric value in the arbitrary precision format.  **Example:** 453889879865131n, 200n, etc. |
| [Boolean](https://www.tutorialsteacher.com/javascript/javascript-boolean) |  | Boolean is a logical data type that has only two values, true or false. |
| [Null](https://www.tutorialsteacher.com/javascript/javascript-null-and-undefined) |  | A null value denotes an absense of value.  **Example:** let str = null; |
| [Undefined](https://www.tutorialsteacher.com/javascript/javascript-null-and-undefined) |  | undefined is the default value of a variable that has not been assigned any value.  **Example:** In the variable declaration, var str;, there is no value assigned to str.  So, the type of str can be check using typeof(str) which will return undefined. |

## **Structural Data Types**

The structural data types contain some kind of structure with primitive data.

| Data Type | Description |
| --- | --- |
| [Object](https://www.tutorialsteacher.com/javascript/javascript-object) | An object holds multiple values in terms of properties and methods.  **Example:**  let person = {  firstName: "James",  lastName: "Bond",  age: 15  }; |
| [Date](https://www.tutorialsteacher.com/javascript/javascript-date) | The Date object represents date & time including days, months, years, hours, minutes, seconds, and milliseconds.  **Example:** let today = new Date("25 July 2021"); |
| [Array](https://www.tutorialsteacher.com/javascript/javascript-array) | An array stores multiple values using special syntax.  **Example:** let nums = [1, 2, 3, 4]; |

JavaScript Strings

* In JavaScript, a string is a primitive data type that is used for textual data.
* JavaScript string must be enclosed in single quotes, double quotes, or backticks. The followings are string literals in JavaScript.

Example: String literals

"Hello World"

'Hello World'

`Hello World`

* The string literal can be assigned to a [variable](https://www.tutorialsteacher.com/javascript/javascript-variable) using the equal to = operator.

Example: String Variables

let str1 = "This is a double quoted string.";

let str2 = 'This is a single quoted string.';

let str3 = `This is a template string.`;

* The template string (using backticks) is used when you want to include the value of a variable or expressions into a string.
* Use ${variable or expression} inside backticks as shown below.

Example: Template String

let amount = 1000, rate = 0.05, duration = 3;

let result = `Total Amount Payble: ${amount\*(1 + rate\*duration)}`;

* The template string can be spanned in multiple lines which is not allowed with a single or double quoted string, as shown below.

Example: Template String

let str1 = `This

is

multi-line

string`;

/\*let str2 = "This

will

give

error"; \*/

* JavaScript string can be treated like a character array.
* You can access a character in a string using square brackets [index] or using the str.at(pos) method.

Example: String as array

let str = 'Hello World';

let ch1 = str[0] // H

let ch2 = str[1] // e

let ch3 = str.at(2) // l

let ch4 = str.at(3) // l

str[4] = "P"; //error

JavaScript strings can be accessed using a [for loop](https://www.tutorialsteacher.com/javascript/javascript-for-loop), as shown below.

Example: Use for Loops

let str = 'Hello World';

for(let i =0; i< str.length; i++)

console.log(str[i]);

for(let ch of str)

console.log(ch);

### **Quotes Inside String**

* You can include single quotes in double-quoted string or include double quotes in a single quoted string.
* However, you cannot include a single quotes in single quoted string and double quotes in double-quoted string.

Example: Quotes in string

let str1 = "This is 'simple' string";

let str2 = 'This is "simple" string';

let str3 = `This is 'simple' and "easy" string`;

* If you want to include the same quotes in a string value as surrounding quotes then use a backward slash (\) before the quotation mark inside the string value.

Example: Quotes in string

let str1 = "This is \"simple\" string";

let str2 = 'This is \'simple\' string';

## **String Concatenation**

* JavaScript string can be concatenated using the + operator or string.concat() method.

Example: String concatenation

let str1 = 'Hello ';

let str2 = "World ";

let str3 = str1 + str2; //Hello World

let str4 = str1.concat(str2);//Hello World

## **String Objects**

* JavaScript allows you to create a string object using the new keyword, as shown below.

Example: String object

let str1 = new String(); //create string object

str1 = 'Hello World'; //assign value

// or

let str2 = new String('Hello World'); //create and assign value

* String objects and string literals are different. The typeof() method will return the type of a variable. The following distinguished string and string objects.

Example: String object

let str1 = new String('Hello World');

let str2 = "Hello World";

typeof(str1); //"object"

typeof(str2); //"string"

## **Strings Comparison**

* Two strings can be compared using <, >, ==, === operator, and string.localeCompare(string) method.
* The mathematical operators < and > compare two strings and return a boolean (true or false) based on the order of the characters in the string.
* The == operator compares the content of strings and === compares the reference equality of strings. The localeCompare() method compares two strings in the current locale. It returns 0 if strings are equal, else returns 1.

Example: String Comparison

console.log("a" < "b"); //true

console.log("b" < "a"); //false

console.log("Apple" == "Apple"); //true

console.log("Apple" == "apple"); //false

console.log("Apple" === "Apple"); //true

console.log("Apple" === "apple"); //false

console.log("Apple".localeCompare("Apple")); //0

console.log("Apple".localeCompare("apple")); //1

* Note that the === operator compares the reference of strings objects and not the values.

Example: String Object Comparison

let str1 = "Hello";

let str2 = 'Hello';

let str3 = new String('Hello');

console.log(str1 == str2); //true

console.log(str1 === str2);//true

console.log(str1 == str3); //true

console.log(str1 === str3);//false

## **JavaScript String Methods & Properties**

* JavaScript string (primitive or String object) includes default properties and methods which you can use for different purposes.

## **String Properties**

| Property | Description |
| --- | --- |
| length | Returns the length of the string. |

## **String Methods**

| Method | Description |
| --- | --- |
| charAt(position) | Returns the character at the specified position (in Number). |
| charCodeAt(position) | Returns a number indicating the Unicode value of the character at the given position (in Number). |
| concat([string,,]) | Joins specified string literal values (specify multiple strings separated by comma) and returns a new string. |
| indexOf(SearchString, Position) | Returns the index of first occurrence of specified String starting from specified number index. Returns -1 if not found. |
| lastIndexOf(SearchString, Position) | Returns the last occurrence index of specified SearchString, starting from specified position. Returns -1 if not found. |
| localeCompare(string,position) | Compares two strings in the current locale. |
| match(RegExp) | Search a string for a match using specified regular expression. Returns a matching array. |
| replace(searchValue, replaceValue) | Search specified string value and replace with specified replace Value string and return new string. Regular expression can also be used as searchValue. |
| search(RegExp) | Search for a match based on specified regular expression. |
| slice(startNumber, endNumber) | Extracts a section of a string based on specified starting and ending index and returns a new string. |
| split(separatorString, limitNumber) | Splits a String into an array of strings by separating the string into substrings based on specified separator. Regular expression can also be used as separator. |
| substr(start, length) | Returns the characters in a string from specified starting position through the specified number of characters (length). |
| substring(start, end) | Returns the characters in a string between start and end indexes. |
| toLocaleLowerCase() | Converts a string to lower case according to current locale. |
| toLocaleUpperCase() | Converts a sting to upper case according to current locale. |
| toLowerCase() | Returns lower case string value. |
| toString() | Returns the value of String object. |
| toUpperCase() | Returns upper case string value. |
| valueOf() | Returns the primitive value of the specified string object. |

## **String Methods for Html**

The following string methods convert the string as a HTML wrapper element.

| Method | Description |
| --- | --- |
| anchor() | Creates an HTML anchor <a>element around string value. |
| big() | Wraps string in <big> element. |
| blink() | Wraps a string in <blink> tag. |
| bold() | Wraps string in <b> tag to make it bold in HTML. |
| fixed() | Wraps a string in <tt> tag. |
| fontcolor() | Wraps a string in a <font color="color"> tag. |
| fontsize() | Wraps a string in a <font size="size"> tag. |
| italics() | Wraps a string in <i> tag. |
| link() | Wraps a string in <a>tag where href attribute value is set to specified string. |
| small() | Wraps a string in a <small>tag. |
| strike() | Wraps a string in a <strike> tag. |
| sub() | Wraps a string in a <sub>tag |
| sup() | Wraps a string in a <sup>tag |

JavaScript Numbers: Integer, Float, Binary, Exponential, Hexadecimal, Octal

* The Number is a primitive data type used for positive or negative integer, float, binary, octal, hexadecimal, and exponential values in JavaScript.
* The first character in a number type must be an integer value, and it must not be enclosed in quotation marks. The following example shows the [variables](https://www.tutorialsteacher.com/javascript/javascript-variable) having different types of numbers in JavaScript.

Example: Numbers in JavaScript

var num1 = 100; // integer

var num2 = -100; //negative integer

var num3 = 10.52; // float

var num4 = -10.52; //negative float

var num5 = 0xfff; // hexadecimal

var num6 = 256e-5; // exponential

var num7 = 030; // octal

var num8 = 0b0010001; // binary

## **Integers**

* Numbers can be positive or negative integers. However, integers are floating-point values in JavaScript.
* Integers value will be accurate up to 15 digits in JavaScript. Integers with 16 digits onwards will be changed and rounded up or down; therefore, use BigInt for integers larger than 15 digits.

Example: Integers in JavaScript

//16 digit integer

var int1 = 1234567890123456; //accurate

//17 digit integer

var int2 = 12345678901234569; //will be 12345678901234568

//16 digit integer

var int3 = 9999999999999998; //will be 9999999999999998

//16 digit integer, last digit 9

var int4 = 9999999999999999; //will be 10000000000000000

## **BigInt**

* The BigInt type is a numeric primitive type that can store integers with arbitrary precision.
* Use the BigInt for the large integers having more than 15 digits. Append n to the end of an integer to make it BigInt.

Example: Integers in JavaScript

//16 digit integer

var int1 = 1234567890123459n; //will be 1234567890123459

//17 digit integer

var int2 = 12345678901234569n; //will be 12345678901234569

//20 digit integer

var int3 = 9999999999999999999n; //will be 9999999999999999999

## **Floating-point Numbers**

The floating-point numbers in JavaScript can only keep 17 decimal places of precision; beyond that, the value will be changed.

Example: Floating-point Numbers in JavaScript

//17 decimal places

var f1 = 123456789012345.9; //accurate

//18 decimal places

var f2 = 1234567890123456.9; //will be 1234567890123457

//19 decimal places

var f3 = 1234567890123456.79; //will be 1234567890123456.8

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=js-45)

Arithmetic operations on floating-point numbers in JavaScript are not always accurate. For example:

Example: Arithmetic Operations on Floating-point Numbers

var f1 = 5.1 + 5.2; //will be 10.3

var f2 = 10.1 + 10.2; //will be 20.299999999999997

var f3 = (10.1\*100 + 10.2\*100)/100; //instead of 10.1 + 10.2

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=js-46)

Arithmetic operation (except addition) of the numeric string will result in a number, as shown below.

Example: Arithmetic Operation of Numeric Strings

var numStr1 = "5", numStr2 = "4";

var multiplication = numStr1 \* numStr2; //returns20

var division = numStr1 / numStr2; //returns 1.25

var modulus = numStr1 % numStr2; //returns 1

Even if one of the values is a number, the result would be the same.

Example: Arithmetic Operation on Number and String

var num = 5, str = "4";

var multiplication = num \* str; //returns 20

var division = num / str; //returns 1.25

var modulus = num % str; //returns 1

The + operator concatenates if any one value is a literal string.

Example: Arithmetic Operation on Number and String

var num = 5, str = "4";

var result = num + str; //returns "54"

## **Binary, Octal, Hexadecimal, Exponential**

The binary numbers must start with 0b or 0B followed by 0 or 1.

The octal numbers must start with zero and the lower or upper letter 'O', 0o or 0O.

The Hexadecimal numbers must start with zero and the lower or upper letter 'X', 0x or 0X.

The exponential numbers should follow the beN format where b is a base integer or float number followed by e char, and N is an exponential power number.

Example: Binary, Ocal, Hexadecimal, Exponential Numbers

var b = 0b100; // binary

var oct = 0o544; // octal

var hex = 0x123456789ABCDEF; // hexadecimal

var exp = 256e-5; // exponential

## **Number() Function in JavaScript**

The Number() is a constructor function in JavaScript that converts values of other types to numbers.

Example: Number() Function

var i = Number('100');

var f = Number('10.5');

var b = Number('0b100');

typeof(i); // returns number

typeof(f); // returns number

typeof(b); // returns number

By using the [new operator](https://www.tutorialsteacher.com/javascript/new-keyword-in-javascript) with the Number() function will return an object which contains constants and methods for working with numbers.

Example: Number Object

var i = new Number('100');

var f = new Number('10.5');

var b = new Number('0b100');

typeof(i); // returns object

typeof(f); // returns object

typeof(b); // returns object

## **Compare Numbers**

Be careful while comparing numbers using == or === operators. The == operator compares object references and not the values whereas the === operator compare values. The following example compares numbers created by different ways.

Example: Numbers Comparison

var num1 = new Number(100);

var num2 = Number('100');

var num3 = 100;

num1 == num2; // true

num1 === num2; // false

num2 == num3;//true

num2 === num3; // true

num1 == num3;//true

num1 === num3;//false

## **Number Properties**

The Number type includes some default properties. JavaScript treats primitive values as objects, so all the properties and methods are applicable to both literal numbers and number objects.

The following table lists all the properties of Number type.

| **Property** | **Description** |
| --- | --- |
| MAX\_VALUE | Returns the maximum number value supported in JavaScript |
| MIN\_VALUE | Returns the smallest number value supported in JavaScript |
| NEGATIVE\_INFINITY | Returns negative infinity (-Infinity) |
| NaN | Represents a value that is not a number. |
| POSITIVE\_INFINITY | Represents positive infinity (Infinity). |

Example: Number properties

Number.MAX\_VALUE; //1.7976931348623157e+308

Number.MIN\_VALUE; //5e-324

Number.NEGATIVE\_INFINITY; //-Infinity

Number.POSITIVE\_INFINITY; //Infinity

Number.NaN;//NaN

## **Number Methods**

The following table lists all the methods of Number type

| Method | Description |
| --- | --- |
| toExponential(fractionDigits) | Returns exponential value as a string.  **Example:** var num = 100; num.toExponential(2); // returns '1.00e+2' |
| toFixed(fractionDigits) | Returns string of decimal value of a number based on specified fractionDigits.  **Example:** var num = 100; num.toFixed(2); // returns '100.00' |
| toLocaleString() | Returns a number as a string value according to a browser's locale settings.  **Example:** var num = 100; num.toLocaleString(); // returns '100' |
| toPrecision(precisionNumber) | Returns number as a string with specified total digits.  **Example:** var num = 100; num.toPrecision(4); // returns '100.0' |
| toString() | Returns the string representation of the number value.  **Example:** var num = 100; num.toString(); // returns '100' |
| valueOf() | Returns the value of Number object.  **Example:** var num = new Number(100); num.valueOf(); // returns '100' |

JavaScript Booleans

* The boolean (not Boolean) is a primitive data type in JavaScript. It can have only two values: true or false.
* It is useful in controlling program flow using conditional statements like [if else](https://www.tutorialsteacher.com/javascript/javascript-if-else-condition), [switch](https://www.tutorialsteacher.com/javascript/javascript-switch), [while loop](https://www.tutorialsteacher.com/javascript/javascript-while-loop), etc.

The followings are boolean variables.

Example: boolean Variables

var YES = true;

var NO = false;

The following example demonstrates how a boolean value controls the program flow using the [if condition](https://www.tutorialsteacher.com/javascript/javascript-if-else-condition).

Example: Boolean

var YES = true;

var NO = false;

if(YES)

{

alert("This code block will be executed");

}

if(NO)

{

alert("This code block will not be executed");

}

The comparison expressions return boolean values to indicate whether the comparison is true or false. For example, the following expressions return boolean values.

Example: boolean Expressions

var a = 10, b = 20;

var result = 1 > 2; // false

result = a < b; // true

result = a > b; // false

result = a + 20 > b + 5; // true

## **Boolean Function**

JavaScript provides the Boolean() function that converts other types to a boolean type. The value specified as the first parameter will be converted to a boolean value.

The Boolean() will return true for any non-empty, non-zero, object, or array.

Example: Boolean() Function

var a = 10, b = 20;

var b1 = Boolean('Hello'); // true

var b2 = Boolean('h'); // true

var b3 = Boolean(10); // true

var b4 = Boolean([]); // true

var b5 = Boolean(a + b); // true

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=js-56)

If the first parameter is 0, -0, null, false, NaN, undefined, '' (empty string), or no parameter passed then the Boolean() function returns false.

Example: Boolean() Function

var b1 = Boolean(''); // false

var b2 = Boolean(0); // false

var b3 = Boolean(null); // false

var a;

var b4 = Boolean(a); // false

The [new operator](https://www.tutorialsteacher.com/javascript/new-keyword-in-javascript) with the Boolean() function returns a Boolean object.

Example: Boolean Object

var bool = new Boolean(true);

alert(bool); // true

Any boolean object, when passed in a conditional statement, will evaluate to true.

Example: Boolean Object in Condition

var bool = new Boolean(false);

if(bool){

alert('This will be executed.');

}

## **Boolean vs boolean**

* The new Boolean() will return a Boolean object, whereas it returns a boolean without the [new](https://www.tutorialsteacher.com/javascript/new-keyword-in-javascript) keyword.
* The boolean (lower case) is the primitive type, whereas Boolean (upper case) is an object in JavaScript.
* Use the typeof operator to check the types.

Example: Boolean vs boolean

var b1 = new Boolean(true);

var b2 = true;

typeof b1; // object

typeof b2; // boolean

## **Boolean Methods**

Primitive or Boolean object includes following methods.

| Method | Description |
| --- | --- |
| toLocaleString() | Returns string of boolean value in local browser environment.  **Example:** var result = (1 > 2); result.toLocaleString();  // returns "false" |
| toString() | Returns a string of Boolean.  **Example:** var result = (1 > 2); result.toString(); // returns "false" |
| valueOf() | Returns the value of the Boolean object.  **Example:** var result = (1 > 2); result.valueOf(); // returns false |

JavaScript Objects: Create Objects, Access Properties & Methods

Here you will learn objects, object literals, Object() constructor function, and access object in JavaScript.

You learned about [primitive and structured data types in JavaScript](https://www.tutorialsteacher.com/javascript/javascript-data-types). An object is a non-primitive, structured data type in JavaScript. Objects are same as variables in JavaScript, the only difference is that an object holds multiple values in terms of properties and methods.

In JavaScript, an object can be created in two ways: 1) using Object Literal/Initializer Syntax 2) using the Object() Constructor function with the [new keyword](https://www.tutorialsteacher.com/javascript/new-keyword-in-javascript). Objects created using any of these methods are the same.

The following example demonstrates creating objects using both ways.

Example: JavaScript Objects

var p1 = { name:"Steve" }; // object literal syntax

var p2 = new Object(); // Object() constructor function

p2.name = "Steve"; // property

* Above, p1 and p2 are the names of objects. Objects can be declared same as [variables](https://www.tutorialsteacher.com/javascript/javascript-variable) using var or let keywords.
* The p1 object is created using the object literal syntax (a short form of creating objects) with a property named name.
* The p2 object is created by calling the Object() constructor function with the new keyword.
* The p2.name = "Steve"; attach a property name to p2 object with a [string](https://www.tutorialsteacher.com/javascript/javascript-string) value "Steve".

## **Create Object using Object Literal Syntax**

The object literal is a short form of creating an object. Define an object in the { } brackets with key:value pairs separated by a comma. The key would be the name of the property and the value will be a literal value or a function.

Syntax:

var <object-name> = { key1: value1, key2: value2,...};

The following example demonstrates objects created using object literal syntax.

Example: Object Literal Syntax

var emptyObject = {}; // object with no properties or methods

var person = { firstName: "John" }; // object with single property

// object with single method

var message = {

showMessage: function (val) {

alert(val);

}

};

// object with properties & method

var person = {

firstName: "James",

lastName: "Bond",

age: 15,

getFullName: function () {

return this.firstName + ' ' + this.lastName

}

};

Note that the whole key-value pair must be declared. Declaring only a key without a value is invalid, as shown below.

Example: Wrong Syntax

var person = { firstName };

var person = { getFullName: };

## **Create Objects using Objects() Constructor**

* Another way of creating objects is using the Object() constructor function using the [new](https://www.tutorialsteacher.com/javascript/new-keyword-in-javascript) keyword.
* Properties and methods can be declared using the dot notation .property-name or using the square brackets ["property-name"], as shown below.

Example: Create Object using Object() Constructor

var person = new Object();

// Attach properties and methods to person object

person.firstName = "James";

person["lastName"] = "Bond";

person.age = 25;

person.getFullName = function () {

return this.firstName + ' ' + this.lastName;

};

An object can have variables as properties or can have computed properties, as shown below.

Example: Variables as Object Properties

var firstName = "James";

var lastName = "Bond";

var person = { firstName, lastName }

## **Access JavaScript Object Properties & Methods**

An object's properties can be accessed using the dot notation obj.property-name or the square brackets obj["property-name"]. However, method can be invoked only using the dot notation with the parenthesis, obj.method-name(), as shown below.

Example: Access JS Object

var person = {

firstName: "James",

lastName: "Bond",

age: 25,

getFullName: function () {

return this.firstName + ' ' + this.lastName

}

};

person.firstName; // returns James

person.lastName; // returns Bond

person["firstName"];// returns James

person["lastName"];// returns Bond

person.getFullName(); // calling getFullName function

* In the above example, the person.firstName access the firstName property of a person object.
* The person["firstName"] is another way of accessing a property. An object's methods can be called using () operator e.g. person.getFullName().
* JavaScript engine will return the function definition if accessed method without the parenthesis.
* Accessing undeclared properties of an object will return [undefined](https://www.tutorialsteacher.com/javascript/javascript-null-and-undefined). If you are not sure whether an object has a particular property or not, then use the hasOwnProperty() method before accessing them, as shown below.

Example: hasOwnProperty()

var person = new Object();

person.firstName; // returns undefined

if(person.hasOwnProperty("firstName")){

person.firstName;

}

The properties and methods will be available only to an object where they are declared.

Example: Object Constructor

var p1 = new Object();

p1.firstName = "James";

p1.lastName = "Bond";

var p2 = new Object();

p2.firstName; // undefined

p2.lastName; // undefined

p3 = p1; // assigns object

p3.firstName; // James

p3.lastName; // Bond

p3.firstName = "Sachin"; // assigns new value

p3.lastName = "Tendulkar"; // assigns new value

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=js-159)

## **Enumerate Object's Properties**

Use the for in loop to enumerate an object, as shown below.

Example: Access Object Keys

var person = new Object();

person.firstName = "James";

person.lastName = "Bond";

for(var prop in person){

alert(prop); // access property name

alert(person[prop]); // access property value

};

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=js-154)

## **Pass by Reference**

Object in JavaScript passes by reference from one function to another.

Example: JS Object Passes by Reference

function changeFirstName(per)

{

per.firstName = "Steve";

}

var person = { firstName : "Bill" };

changeFirstName(person)

person.firstName; // returns Steve

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=js-155)

## **Nested Objects**

An object can be a property of another object. It is called a nested object.

Example: Nested JS Objects

var person = {

firstName: "James",

lastName: "Bond",

age: 25,

address: {

id: 1,

country:"UK"

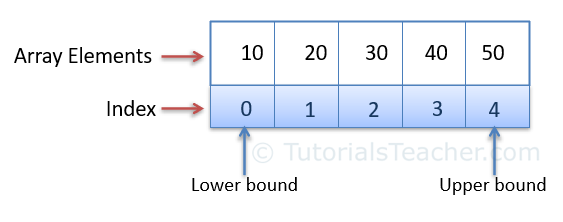
}

};

person.address.country; // returns "UK"

# JavaScript Arrays: Create, Access, Add & Remove Elements

* We have learned that a variable can hold only one value. We cannot assign multiple values to a single variable.
* JavaScript array is a special type of variable, which can store multiple values using a special syntax.
* The following declares an array with five numeric values.
* let numArr = [10, 20, 30, 40, 50];
* In the above array, numArr is the name of an array variable. Multiple values are assigned to it by separating them using a comma inside square brackets as [10, 20, 30, 40, 50].
* Thus, the numArr variable stores five numeric values. The numArr array is created using the literal syntax and it is the preferred way of creating arrays.
* Another way of creating arrays is using the Array() constructor, as shown below.
* let numArr = new Array(10, 20, 30, 40, 50);
* Every value is associated with a numeric index starting with 0. The following figure illustrates how an array stores values.

[](https://www.tutorialsteacher.com/Content/images/js/array1.png)**JavaScript Array Representation**

The following are some more examples of arrays that store different types of data.

Example: Array Literal Syntax

let stringArray = ["one", "two", "three"];

let numericArray = [1, 2, 3, 4];

let decimalArray = [1.1, 1.2, 1.3];

let booleanArray = [true, false, false, true];

* It is not required to store the same type of values in an array. It can store values of different types as well.

let data = [1, "Steve", "DC", true, 255000, 5.5];

## **Get Size of an Array**

Use the length property to get the total number of elements in an array. It changes as and when you add or remove elements from the array.

Example: Get Array Size

let cities = ["Mumbai", "New York", "Paris", "Sydney"];

console.log(cities.length); //4

cities[4] = "Delhi";

console.log(cities.length); //5

## **Accessing Array Elements**

* Array elements (values) can be accessed using an index. Specify an index in square brackets with the array name to access the element at a particular index like arrayName[index].
* Note that the index of an array starts from zero.

Example: Accessing Array Elements

let numArr = [10, 20, 30, 40, 50];

console.log(numArr[0]); // 10

console.log(numArr[1]); // 20

console.log(numArr[2]); // 30

console.log(numArr[3]); // 40

console.log(numArr[4]); // 50

let cities = ["Mumbai", "New York", "Paris", "Sydney"];

console.log(cities[0]); // "Mumbai"

console.log(cities[1]); // "New York"

console.log(cities[2]); // "Paris"

console.log(cities[3]); // "Sydney"

//accessing element from nonexistance index

console.log(cities[4]); // undefined

For the new browsers, you can use the arr.at(pos) method to get the element from the specified index. This is the same as arr[index] except that the at() returns an element from the last element if the specified index is negative.

Example: Accessing Array using at()

let numArr = [10, 20, 30, 40, 50];

console.log(numArr.at(0)); // 10

console.log(numArr.at(1)); // 20

console.log(numArr.at(2)); // 30

console.log(numArr.at(3)); // 40

console.log(numArr.at(4)); // 50

console.log(numArr.at(5)); // undefined

//passing negative index

console.log(numArr.at(-1)); // 50

console.log(numArr.at(-2)); // 40

console.log(numArr.at(-3)); // 30

console.log(numArr.at(-4)); // 20

console.log(numArr.at(-5)); // 10

console.log(numArr.at(-6)); // undefined

You can iterate an array using Array.forEach(), for, for-of, and for-in loop, as shown below.

Example: Accessing Array Elements

let numArr = [10, 20, 30, 40, 50];

numArr.forEach(i => console.log(i)); //prints all elements

for(let i=0; i<numArr.length; i++)

console.log(numArr[i]);

for(let i of numArr)

console.log(i);

for(let i in numArr)

console.log(numArr[i]);

## **Update Array Elements**

You can update the elements of an array at a particular index using arrayName[index] = new\_value syntax.

Example: Update Array Elements

let cities = ["Mumbai", "New York", "Paris", "Sydney"];

cities[0] = "Delhi";

cities[1] = "Los angeles";

console.log(cities); //["Delhi", "Los angeles", "Paris", "Sydney"]

## **Adding New Elements**

* You can add new elements using arrayName[index] = new\_value syntax. Just make sure that the index is greater than the last index.
* If you specify an existing index then it will update the value.

Example: Add Array Elements

let cities = ["Mumbai", "New York", "Paris", "Sydney"];

cities[4] = "Delhi"; //add new element at last

console.log(cities); //["Mumbai", "New York", "Paris", "Sydney", "Delhi"]

cities[cities.length] = "London";//use length property to specify last index

console.log(cities); //["Mumbai", "New York", "Paris", "Sydney", "Delhi", "London"]

cities[9] = "Pune";

console.log(cities); //["Mumbai", "New York", "Paris", "Sydney", "Delhi", "Londen", undefined, undefined, undefined, "Pune"]

* In the above example, cities[9] = "Pune" adds "Pune" at 9th index and all other non-declared indexes as undefined.
* The recommended way of adding elements at the end is using the push() method. It adds an element at the end of an array.

Example: Add Element At Last using push()

let cities = ["Mumbai", "New York", "Paris", "Sydney"];

cities.push("Delhi"); //add new element at last

console.log(cities); //["Mumbai", "New York", "Paris", "Sydney", "Delhi"]

* Use the unshift() method to add an element to the beginning of an array.

Example: Add Element using unshift()

let cities = ["Mumbai", "New York", "Paris", "Sydney"];

cities.unshift("Delhi"); //adds new element at the beginning

console.log(cities); //["Delhi", "Mumbai", "New York", "Paris", "Sydney"]

cities.unshift("London", "Pune"); //adds new element at the beginning

console.log(cities); //["London", "Pune", "Delhi", "Mumbai", "New York", "Paris", "Sydney"]

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=js-89)

## **Remove Array Elements**

The pop() method returns the last element and removes it from the array.

Example: Remove Last Element

let cities = ["Mumbai", "New York", "Paris", "Sydney"];

let removedCity = cities.pop(); //returns and removes the last element

console.log(cities); //["Mumbai", "New York", "Paris"]

The shift() method returns the first element and removes it from the array.

Example: Remove First Element

let cities = ["Mumbai", "New York", "Paris", "Sydney"];

let removedCity = cities.shift(); //returns first element and removes it from array

console.log(cities); //["New York", "Paris", "Sydney"]

You cannot remove middle elements from an array. You will have to create a new array from an existing array without the element you do not want, as shown below.

Example: Remove Middle Elements

let cities = ["Mumbai", "New York", "Paris", "Sydney"];

let cityToBeRemoved = "Paris";

let mycities = cities.filter(function(item) {

return item !== cityToBeRemoved

})

console.log(mycities); //["Mumbai", "New York", "Sydney"]

console.log(cities); //["Mumbai", "New York", "Paris", "Sydney"]

# Array Methods Reference

The following table lists all the Array methods.

| Method | Description |
| --- | --- |
| concat() | Returns new array by combining values of an array that is specified as parameter with existing array values. |
| every() | Returns true or false if every element in the specified array satisfies a condition specified in the callback function. Returns false even if single element does not satisfy the condition. |
| filter() | Returns a new array with all the elements that satisfy a condition specified in the callback function. |
| forEach() | Executes a callback function for each elements of an array. |
| indexOf() | Returns the index of the first occurrence of the specified element in the array, or -1 if it is not found. |
| join() | Returns string of all the elements separated by the specified separator |
| lastIndexOf() | Returns the index of the last occurrence of the specified element in the array, or -1 if it is not found. |
| map() | Creates a new array with the results of calling a provided function on every element in this array. |
| pop() | Removes the last element from an array and returns that element. |
| push() | Adds one or more elements at the end of an array and returns the new length of the array. |
| reduce() | Pass two elements simultaneously in the callback function (till it reaches the last element) and returns a single value. |
| reduceRight() | Pass two elements simultaneously in the callback function from right-to-left (till it reaches the last element) and returns a single value. |
| reverse() | Reverses the elements of an array. Element at last index will be first and element at 0 index will be last. |
| shift() | Removes the first element from an array and returns that element. |
| slice() | Returns a new array with specified start to end elements. |
| some() | Returns true if at least one element in this array satisfies the condition in the callback function. |
| sort() | Sorts the elements of an array. |
| splice() | Adds and/or removes elements from an array. |
| toString() | Returns a string representing the array and its elements. |
| unshift() | Adds one or more elements to the front of an array and returns the new length of the array. |

# Difference between null and undefined in JavaScript

Here you will learn what is null and undefined in JavaScript and what is the difference between them.

## **What is a null?**

* A null means the absence of a value.
* You assign a null to a variable with the intention that currently this variable does not have any value but it will have later on.
* It is like a placeholder for a value. The type of null is the object.

Example: null

let num = null;

console.log(num); // null

console.log(typeof num); // "object"

Sometimes, null variables are the result of erroneous code. For example, if you try to find an HTML element using document.getElelementByID() with the wrong id, then it will return null. So it is recommended to check for null before doing something with that element.

Example: null

var saveButton = document.getElementById("save");

if (saveButton !== null)

saveButton.submit();

## **What is undefined?**

A variable is undefined when you haven't assigned any value yet, not even a null.

Example: undefined Variable

let num;

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

Generally, variables are undefined when you forgot to assign values or change existing code. For example, consider the following Greet() function that returns a string.

function Greet(){

return "Hi";

}

let str = Greet();//"Hi"

Now, suppose somebody changes the function as below. So now, str will be undefined.

function Greet(){

alert("Hi");

}

let str = Greet();//undefined

Thus, undefined variables are the result of some code problems.

## **Difference between null and undefined**

You must explicitly assign a null to a variable. A variable has undefined when no value assigned to it.

Example: null and undefined Variables

let num1 = null;

let num2;

console.log(num1);//null

console.log(num2); //undefined

The '' is not the same as null or undefined.

let str = '';

console.log(typeof str);//string

console.log(str === null); //false

console.log(str === undefined); //false

The type of null variable is object whereas the type of undefined variable is "undefined".

Example: Types

let num1 = null;

let num2;

console.log(typeof num1);//"object"

console.log(typeof num2); //"undefined"

Use the === operator to check whether a variable is null or undefined. The == operator gives the wrong result.

Example: Comparison using === and ==

let num1 = null;

let num2;

console.log(num1 == null); //true

console.log(num2 == undefined);//true

console.log(num1 == undefined);//true (incorrect)

console.log(num2 == null);//true (incorrect)

console.log(num1 === null); //true

console.log(num2 === undefined);//true

console.log(num1 === undefined);//false

console.log(num2 === null);//false

console.log(num1 == num2);//true (incorrect)

console.log(num1 === num2);//false

The null and undefined variables are falsy to if-statements and ternary operators.

Example: Null and undefined with if-statements

let num1 = null;

let num2;

if(num1)

{

console.log(num1);

}

else

{

console.log("num1 is null");

}

if(num2)

{

console.log(num2);

}

else

{

console.log("num2 is undefined");

}

A null variable treated as 0 in an numeric expression whereas undefined variable will be NaN.

Example: With Numeric Expresion

let num1 = null;

let num2;

console.log(num1 + 10);//10

console.log(num2 + 10); //NaN

It will give wrong result when concatenated with string.

Example: With String Values

let num1 = null;

let num2;

console.log(num1 + " Hello");//"null Hello"

console.log(num2 + " Hello"); //"undefined Hello"

Note: The null and undefined variables are one of the main reasons for runtime errors in JavaScript. The best practice is to check variables for null or undefined before using them.

# Functions in JavaScript

Functions are the basic building block of JavaScript. Functions allow us to encapsulate a block of code and reuse it multiple times.

Functions make JavaScript code more readable, organized, reusable, and maintainable.

Syntax:

function <function-name>(arg1, arg2, arg3,...)

{

//write function code here

};

In JavaScript, a function can be defined using the function keyword, followed by the name of a function and parentheses. Optionally, a list of input parameters can be included within the parentheses. The code block that needs to be executed when the function is called is written within curly braces.

## **Defining a Function in JavaScript**

The following defines a function named greet that will display an alert box.

Example: Define a Function

function greet() {

alert("Hello World!");

}

The above greet() function does not include any input parameters. It contains a single statement that displays an alert message.

Now, you can call or invoke the greet function by using the function name followed by the () operator, as shown below. When you call a function, JavaScript will execute the codes written inside the calling function.

Example: Calling a Function

greet();

## **Function Parameters**

You can pass values to a function using parameters. A function can have one or more parameters, and the values will be passed by the calling code.

Example: Function Parameters

function greet(firstName, lastName) {

alert("Hello " + firstName + " " + lastName);

}

greet("Steve", "Jobs");

JavaScript is a dynamic type scripting language, so a function parameter can have a value of any data type.

Example: Function Parameters

function greet(firstName, lastName) {

alert("Hello " + firstName + " " + lastName);

}

greet("Bill", "Gates");

greet(100, 200);

You can pass fewer or more arguments while calling a function. If you pass fewer arguments then the rest of the parameters will become [undefined](https://www.tutorialsteacher.com/javascript/javascript-null-and-undefined). If you pass more arguments then additional arguments will be ignored.

Example: Function Parameters

function greet(firstName, lastName) {

alert("Hello " + firstName + " " + lastName);

}

greet("Steve", "Jobs", "Mr."); // display Hello Steve Jobs

greet("Bill"); // display Hello Bill undefined

greet(); // display Hello undefined undefined

You can also use the built-in [arguments object](https://www.tutorialsteacher.com/articles/javascript-arguments-object) to access parameters inside a function.

## **Return a Value From a Function**

A function can return a value to the calling code using the return keyword followed by a variable or a value.

The following returns a number 10.

Example: Return a value of a Function

function getNumber() {

return 10;

};

let result = getNumber();

console.log(result); //output: 10

Typically, a function returns some calculated value using paramters or an expression from a function. For example, the following sum function adds two parameters values using the + operator and returns the result of an expression.

Example: Return value from a Function

function Sum(num1, num2) {

return num1 + num2;

};

var result = Sum(10,20); // returns 30

A function can return another function in JavaScript.

Example: Function Returning a Function

function multiple(x) {

function fn(y)

{

return x \* y;

}

return fn;

}

var triple = multiple(3);

triple(2); // returns 6

triple(3); // returns 9

## **Function Expression**

A function expression in JavaScript is a function that is stored as a value, and can be assigned to a variable or passed as an argument to another function.

Example: Function Expression

var add = function (num1, num2) {

return num1 + num2;

};

var result = add(10, 20);//returns 30

## **Anonymous Function**

In JavaScript, you can also create anonymous functions, which are functions without a name. Anonymous functions are often used as arguments to other functions, and are

Anonymous functions are typically used in functional programming e.g. callback function, creating closure or [immediately invoked function expression](https://www.tutorialsteacher.com/javascript/immediately-invoked-function-expression-iife).

Example: Anonymous Function

let numbers = [10, 20, 30, 40, 50];

let squareNumbers = numbers.map(function(number) {

return number \* number;

});

## **Arrow Functions**

Arrow functions are a shorthand syntax for defining anonymous functions in JavaScript. They have compact syntax compared to anonymous functions. However, they do not have their own this value.

Example: Arrow Function

let square = num => num \* num;

let result = square(5);

console.log(result); //25

## **Nested Functions**

In JavaScript, a function can have one or more inner functions. These nested functions are in the scope of outer function. Inner function can access variables and parameters of outer function. However, outer function cannot access variables defined inside inner functions.

Example: Nested Functions

function greet(firstName)

{

function SayHello() {

alert("Hello " + firstName);

}

return SayHello();

}

greet("Steve");

# JavaScript if...else Statements

JavaScript includes if/else conditional statements to control the program flow, similar to other programming languages.

JavaScript includes following forms of if-else statements:

1. if Statement
2. if else Statement
3. else if Statement

## **if Statement**

Use if conditional statement if you want to execute something based on some condition.

Syntax:

if(boolean expression)

{

// code to be executed if condition is true

}

Example: if condition

if( 1 > 0)

{

alert("1 is greater than 0");

}

if( 1 < 0)

{

alert("1 is less than 0");

}

* In the above example, the first if statement contains 1 > 0 as conditional expression. The conditional expression 1 > 0 will be evaluated to true, so an alert message "1 is greater than 0" will be displayed, whereas conditional expression in second if statement will be evaluated to false, so "1 is less than 0" alert message will not be displayed.
* In the same way, you can use variables in a conditional expression.

Example: if condition

var mySal = 1000;

var yourSal = 500;

if( mySal > yourSal)

{

alert("My Salary is greater than your salary");

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=js-112)

 Note:

curly braces { } is not required when if block contains only a single line to execute.

Use comparison operators carefully when writing conditional expression. For example, == and === is different.

Example: if condition

if(1=="1")

{

alert("== operator does not consider types of operands");

}

if(1==="1")

{

alert("=== operator considers types of operands");

}

## **else condition**

Use else statement when you want to execute the code every time when if condition evaluates to false.

The else statement must follow **if** or **else if** statement. Multiple else block is NOT allowed.

Syntax:

if(condition expression)

{

//Execute this code..

}

else{

//Execute this code..

}

Example: else condition

var mySal = 500;

var yourSal = 1000;

if( mySal > yourSal)

{

alert("My Salary is greater than your salary");

}

else

{

alert("My Salary is less than or equal to your salary");

}

## **else if condition**

Use "else if" condition when you want to apply second level condition after if statement.

Syntax:

if(condition expression)

{

//Execute this code block

}

else if(condition expression){

//Execute this code block

}

Example: else if condition

var mySal = 500;

var yourSal = 1000;

if( mySal > yourSal)

{

alert("My Salary is greater than your salary");

}

else if(mySal < yourSal)

{

alert("My Salary is less than your salary");

}

JavaScript allows multiple **else if** statements also.

Example: Multiple if else conditions

var mySal = 500;

var yourSal = 1000;

if( mySal > yourSal)

{

alert("My Salary is greater than your salary");

}

else if(mySal < yourSal)

{

alert("My Salary is less than your salary");

}

else if(mySal == yourSal)

{

alert("My Salary is equal to your salary");

}

# JavaScript switch

The **switch** is a conditional statement like if statement. Switch is useful when you want to execute one of the multiple code blocks based on the return value of a specified expression.

Syntax:

switch(expression or literal value){

case 1:

//code to be executed

break;

case 2:

//code to be executed

break;

case n:

//code to be executed

break;

default:

//default code to be executed

//if none of the above case executed

}

Use break keyword to stop the execution and exit from the switch. Also, you can write multiple statements in a case without using curly braces { }.

As per the above syntax, switch statement contains an expression or literal value. An expression will return a value when evaluated. The switch can includes multiple cases where each case represents a particular value. Code under particular case will be executed when case value is equal to the return value of switch expression. If none of the cases match with switch expression value then the default case will be executed.

Example: switch Statement

var a = 3;

switch (a) {

case 1:

alert('case 1 executed');

break;

case 2:

alert("case 2 executed");

break;

case 3:

alert("case 3 executed");

break;

case 4:

alert("case 4 executed");

break;

default:

alert("default case executed");

}

* In the above example, switch statement contains a literal value as expression.
* So, the case that matches a literal value will be executed, case 3 in the above example.
* The switch statement can also include an expression. A case that matches the result of an expression will be executed.

Example: switch Statement

var a = 3;

switch (a/3) {

case 1:

alert("case 1 executed");

break;

case 2:

alert("case 2 executed");

break;

case 3:

alert("case 3 executed");

break;

case 4:

alert("case 4 executed");

break;

default:

alert("default case executed");

}

In the above example, switch statement includes an expression a/3, which will return 1 (because a = 3). So, case 1 will be executed in the above example.

The switch can also contain string type expression.

Example: switch with String Type Case

var str = "bill";

switch (str)

{

case "steve":

alert("This is Steve");

case "bill":

alert("This is Bill");

break;

case "john":

alert("This is John");

break;

default:

alert("Unknown Person");

break;

}

Multiple cases can be combined in a switch statement.

Example: Combined switch Cases

var a = 2;

switch (a) {

case 1:

case 2:

case 3:

alert("case 1, 2, 3 executed");

break;

case 4:

# alert("case 4 executed");

# JavaScript for Loop

JavaScript includes for loop like Java or C#. Use for loop to execute code repeatedly.

Syntax:

for(initializer; condition; iteration)

{

// Code to be executed

}

The for loop requires following three parts.

* Initializer: Initialize a counter variable to start with
* Condition: specify a condition that must evaluate to true for next iteration
* Iteration: increase or decrease counter

Example: for loop

for (var i = 0; i < 5; i++)

{

console.log(i);

}

Output:

0 1 2 3 4

In the above example, var i = 0 is an initializer statement where we declare a variable **i** with value 0. The second part, **i < 5** is a condition where it checks whether i is less than 5 or not. The third part, **i++** is iteration statement where we use ++ operator to increase the value of i to 1. All these three parts are separated by semicolon **;**.

The for loop can also be used to get the values for an array.

Example: for loop

var arr = [10, 11, 12, 13, 14];

for (var i = 0; i < 5; i++)

{

console.log(arr[i]);

}

Output:

10 11 12 13 14

Please note that it is not mandatory to specify an initializer, condition and increment expression into bracket. You can specify initializer before starting for loop. The condition and increment statements can be included inside the block.

Example: for loop

var arr = [10, 11, 12, 13, 14];

var i = 0;

for (; ;) {

if (i >= 5)

break;

console.log(arr[i]);

# JavaScript - While Loop

JavaScript includes while loop to execute code repeatedly till it satisfies a specified condition. Unlike for loop, while loop only requires condition expression.

Syntax:

while(condition expression)

{

/\* code to be executed

till the specified condition is true \*/

}

Example: while loop

var i =0;

while(i < 5)

{

console.log(i);

i++;

}

Output:

0 1 2 3 4

Make sure condition expression is appropriate and include increment or decrement counter variables inside the while block to avoid infinite loop.

As you can see in the above example, while loop will execute the code block till i < 5 condition turns out to be false. Initialization statement for a counter variable must be specified before starting while loop and increment of counter must be inside while block.

## **do while**

JavaScript includes another flavour of while loop, that is do-while loop. The do-while loop is similar to while loop the only difference is it evaluates condition expression after the execution of code block. So do-while loop will execute the code block at least once.

Syntax:

do{

//code to be executed

}while(condition expression)

Example: do-while loop

var i = 0;

do{

alert(i);

i++;

} while(i < 5)

Output:

0 1 2 3 4

The following example shows that do-while loop will execute a code block even if the condition turns out to be false in the first iteration.

Example: do-while loop

var i =0;

do{

alert(i);

i++;

} while(i > 1)

Output:

0

i++;

}

Output:

10 11 12 13 14

break;

default:

alert("default case executed");

}

# JavaScript eval

eval() is a global function in JavaScript that evaluates a specified string as JavaScript code and executes it.

Example: eval

eval("alert('this is executed by eval()')");

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=js-181)

The eval() function can also call the function and get the result as shown below.

Example: eval

var result;

function Sum(val1, val2)

{

return val1 + val2;

}

eval("result = Sum(5, 5);");

alert(result);

eval can convert string to JSON object.

Example: eval with JSON Object

var str = '({"firstName":"Bill","lastName":"Gates"})';

var obj = eval(str);

obj.firstName; // Bill

# JavaScript Hoisting

Hoisting is a concept in JavaScript, not a feature. In other scripting or server side languages, variables or functions must be declared before using it.

In JavaScript, variable and function names can be used before declaring it. The JavaScript compiler moves all the declarations of variables and functions at the top so that there will not be any error. This is called hoisting.

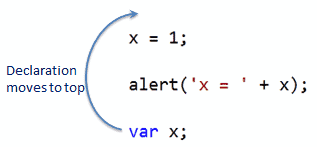
Example: Hoisting

x = 1;

alert('x = ' + x); // display x = 1

var x;

The following figure illustrates hoisting.

[](https://www.tutorialsteacher.com/Content/images/js/hoisting.png)JavaScript Hoisting

Also, a variable can be assigned to another variable as shown below.

Example: Hoisting

x = 1;

y = x;

alert('x = ' + x);

alert('y = ' + y);

var x;

var y;

Hoisting is only possible with declaration but not the initialization. JavaScript will not move variables that are declared and initialized in a single line.

Example: Hoisting not applicable for initialized variables

alert('x = ' + x); // display x = undefined

var x = 1;

As you can see in the above example, value of x will be undefined because var x = 1 is not hoisted.

## **Hoisting of Function**

JavaScript compiler moves the function definition at the top in the same way as variable declaration.

Example: Function Hoisting

alert(Sum(5, 5)); // 10

function Sum(val1, val2)

{

return val1 + val2;

}

Please note that JavaScript compiler does not move function expression.

Example: Hoisting on function expression

Add(5, 5); // error

var Add = function Sum(val1, val2)

{

return val1 + val2;

}

## **Hoisting Functions Before Variables**

JavaScript compiler moves a function's definition before variable declaration. The following example proves it.

Example: Function Hoisting Before Variables

alert(UseMe);

var UseMe;

function UseMe()

{

alert("UseMe function called");

}

# Define Class in JavaScript

JavaScript ECMAScript 5, does not have class type. So it does not support full object oriented programming concept as other languages like Java or C#. However, you can create a function in such a way so that it will act as a class.

The following example demonstrates how a function can be used like a class in JavaScript.

Example: Class in JavaScript

function Person() {

this.firstName = "unknown";

this.lastName = "unknown";

}

var person1 = new Person();

person1.firstName = "Steve";

person1.lastName = "Jobs";

alert(person1.firstName + " " + person1.lastName);

var person2 = new Person();

person2.firstName = "Bill";

person2.lastName = "Gates";

alert(person2.firstName + " " + person2.lastName );

In the above example, a Person() function includes firstName, lastName & age variables using [this](https://www.tutorialsteacher.com/javascript/this-keyword-in-javascript) keyword. These variables will act like properties. As you know, we can create an object of any function using [new](https://www.tutorialsteacher.com/javascript/new-keyword-in-javascript) keyword, so person1 object is created with new keyword. So now, Person will act as a class and person1 & person2 will be its objects (instances). Each object will hold their values separately because all the variables are defined with [this](https://www.tutorialsteacher.com/javascript/this-keyword-in-javascript) keyword which binds them to particular object when we create an object using new keyword.

So this is how a function can be used like a class in the JavaScript.

## **Add Methods in a Class**

We can add a function expression as a member variable in a function in JavaScript. This function expression will act like a method of class.

Example: Method in Class

function Person() {

this.firstName = "unknown";

this.lastName = "unknown";

this.getFullName = function(){

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

}

};

var person1 = new Person();

person1.firstName = "Steve";

person1.lastName = "Jobs";

alert(person1.getFullName());

var person2 = new Person();

person2.firstName = "Bill";

person2.lastName = "Gates";

alert(person2.getFullName());

In the above example, the Person function includes function expression that is assigned to a member variable getFullName. So now, getFullName() will act like a method of the Person class. It can be called using dot notation e.g. person1.getFullName().

## **Constructor**

In the other programming languages like Java or C#, a class can have one or more constructors. In JavaScript, a function can have one or more parameters. So, a function with one or more parameters can be used like a constructor where you can pass parameter values at the time or creating an object with new keyword.

Example: Constructor

function Person(FirstName, LastName, Age) {

this.firstName = FirstName || "unknown";

this.lastName = LastName || "unknown";

this.age = Age || 25;

this.getFullName = function () {

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

}

};

var person1 = new Person("James","Bond",50);

alert(person1.getFullName());

var person2 = new Person("Tom","Paul");

alert(person2.getFullName());

In the above example, the Person function includes three parameters FirstName, LastName and Age. These parameters are used to set the values of a respective property.

 Note:

Please notice that parameter assigned to a property, if parameter value is not passed while creating an object using new then they will be undefined.

## **Properties with Getters and Setters**

As you learned in the previous section, Object.defineProperty() method can be used to define a property with getter & setter.

The following example shows how to create a property with getter & setter.

Example: Property

function Person() {

var \_firstName = "unknown";

Object.defineProperties(this, {

"FirstName": {

get: function () {

return \_firstName;

},

set: function (value) {

\_firstName = value;

}

}

});

};

var person1 = new Person();

person1.FirstName = "Steve";

alert(person1.FirstName );

var person2 = new Person();

person2.FirstName = "Bill";

alert(person2.FirstName );

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=oojs-14)

In the above example, the Person() function creates a FirstName property by using Object.defineProperties() method. The first argument is this, which binds FirstName property to calling object. Second argument is an object that includes list of properties to be created. We have specified FirstName property with get & set function. You can then use this property using dot notation as shown above.

## **Read-only Property**

Do not specify set function in order to create read-only property as shown below.

Example: Read-only Property

function Person(firstName) {

var \_firstName = firstName || "unknown";

Object.defineProperties(this, {

"FirstName": {

get: function () {

return \_firstName;

}

}

});

};

var person1 = new Person("Steve");

//person1.FirstName = "Steve"; -- will not work

alert(person1.FirstName );

var person2 = new Person("Bill");

//person2.FirstName = "Bill"; -- will not work

alert(person2.FirstName );

## **Multiple Properties**

Specify more than one property in defineProperties() method as shown below.

Example: Multiple Properties

function Person(firstName, lastName, age) {

var \_firstName = firstName || "unknown";

var \_lastName = lastName || "unknown";

var \_age = age || 25;

Object.defineProperties(this, {

"FirstName": {

get: function () { return \_firstName },

set: function (value) { \_firstName = value }

},

"LastName": {

get: function () { return \_lastName },

set: function (value) { \_lastName = value }

},

"Age": {

get: function () { return \_age },

set: function (value) { \_age = value }

}

});

this.getFullName = function () {

return this.FirstName + " " + this.LastName;

}

};

var person1 = new Person();

person1.FirstName = "John";

person1.LastName = "Bond";

alert(person1.getFullName());

# JavaScript Objects in Depth

You have already learned about [JavaScript object](https://www.tutorialsteacher.com/javascript/javascript-object) in the JavaScript Basics section. Here, you will learn about object in detail.

As you know, object in JavaScript can be created using object literal, object constructor or constructor function. Object includes properties. Each property can either be assigned a literal value or a function.

Consider the following example of objects created using object literal and constructor function.

Example: JavaScript Object

// object literal

var person = {

firstName:'Steve',

lastName:'Jobs'

};

// Constructor function

function Student(){

this.name = "John";

this.gender = "Male";

this.sayHi = function(){

alert('Hi');

}

}

var student1 = new Student();

console.log(student1.name);

console.log(student1.gender);

student1.sayHi();

In the above example, person object is created using object literal that includes firstName and lastName properties and student1 object is created using constructor function Student that includes name, gender and sayHi properties where function is assigned to sayHi property.

 Note:

Any javascript function using which object is created is called constructor function.

Use Object.keys() method to retrieve all the properties name for the specified object as a string array.

Example: Edit Property Descriptor

function Student(){

this.title = "Mr.";

this.name = "Steve";

this.gender = "Male";

this.sayHi = function(){

alert('Hi');

}

}

var student1 = new Student();

Object.keys(student1);

Output:

["title", "name", "gender", "sayHi"]

Use for-in loop to retrieve all the properties of an object as shown below.

Example: Enumerable Properties

function Student(){

this.title = "Mr.";

this.name = "Steve";

this.gender = "Male";

this.sayHi = function(){

alert('Hi');

}

}

var student1 = new Student();

//enumerate properties of student1

for(var prop in student1){

console.log(prop);

}

Output:

title name gender sayHi

# Immediately Invoked Function Expression - IIFE

Immediately Invoked Function Expression (IIFE) is one of the most popular design patterns in JavaScript. It pronounces like iify. IIFE has been used since long by JavaScript community but it had misleading term "self-executing anonymous function". [Ben Alman](http://benalman.com/) gave it appropriate name "Immediately Invoked Function Expression"

As you know that a function in JavaScript creates the local scope. So, you can define variables and function inside a function which cannot be access outside of that function. However, sometime you accidently pollute the global variables or functions by unknowingly giving same name to variables & functions as global variable & function names. For example, there are multiple .js files in your application written by multiple developers over a period of time. Single JavaScript file includes many functions and so these multiple .js files will result in large number of functions. There is a good chance of having same name of function exists in different .js files written by multiple developer and if these files included in a single web page then it will pollute the global scope by having two or more function or variables with the same name. Consider following example of two different JavaScript file included in single page.

Consider the following example of MyScript1.js and MyScript2.js with same variable & function name.

Example: MyScript1.js

var userName = "Bill";

function display(name)

{

alert("MyScript1.js: " + name);

}

display(userName);

Example: MyScript2.js

var userName = "Steve";

function display(name)

{

alert("MyScript2.js: " + name);

}

display(userName);

Now, if you include these JS files in your web page then guess what will happen?

Example: Script tag in <head>

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>JavaScript Demo</title>

<script src="/MyScript1.js"></<script>

<script src="/MyScript2.js"></<script>

</head>

<body>

<h1> IIFE Demo</h1>

</body>

</html>

If you run above example, you will find that every time it call display() function in MyScript2.js because MyScript2.js included after MyScript1.js in a web page. So JavaScript considers last definition of a function if two functions have the same name.

IEFE solves this problem by having its own scope and restricting functions and variables to become global. The functions and variables declare inside IIFE will not pollute global scope even they have same name as global variables & functions. So let's see what is an IIFE is.

## **What is an IIFE?**

As name suggest, IIFE is a function expression that automatically invokes after completion of the definition. The parenthesis () plays important role in IIFE pattern. In JavaScript, parenthesis cannot contain statements; it can only contain an expression.

Example: Parenthesis ()

(var foo = 10 > 9); // syntax error

(var foo = "foo", bar = "bar"); // syntax error

(10 > 9); // valid

(alert("Hi")); // valid

First of all, define a function expression.

Example: IIFE

var myIIFE = function () {

//write your js code here

};

Now, wrap it with parenthesis. However, parenthesis does not allow declaration. So just remove declaration part and just write anonymous function as below.

Example: IIFE

(function () {

//write your js code here

});

Now, use () operator to call this anonymous function immediately after completion of its definition.

Example: IIFE

(function () {

//write your js code here

})();

So, the above is called IIFE. You can write all the functions and variables inside IIFE without worrying about polluting the global scope or conflict with other's JavaScript code which have functions or variables with same name.

To solve the our above problem, wrap all the code in MyScript1.js & MyScript2.js file in IIFE as shown below.

Example: IIFE

(function () {

var userName = "Steve";

function display(name)

{

alert("MyScript2.js: " + name);

}

display(userName);

})();

So, even if MyScript1.js & MyScript2.js file includes functions and variables with the same name, they won't conflict with each other and pollute the global scope.

Also, you can pass arguments in IIFE as shown below.

Example: IIFE

var userName = "Bill";

(function (name) {

function display(name)

{

alert("MyScript2.js: " + name);

}

display(name);

})(userName);

Advantages of IIFE:

1. Do not create unnecessary global variables and functions
2. Functions and variables defined in IIFE do not conflict with other functions & variables even if they have same name.
3. Organize JavaScript code.
4. Make JavaScript code maintainable.