31. JS Comparison

JavaScript Comparison and Logical Operators

Comparison and Logical operators are used to test for *true* or *false*.

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that **x = 5**, the table below explains the comparison operators:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Description** | **Comparing** | **Returns** | **Try it** |
| == | equal to | x == 8 | false | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison1) |
| x == 5 | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison2) |
| x == "5" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison12) |
| === | equal value and equal type | x === 5 | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison4) |
| x === "5" | false | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison3) |
| != | not equal | x != 8 | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison5) |
| !== | not equal value or not equal type | x !== 5 | false | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison7) |
| x !== "5" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison6) |
| x !== 8 | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison13) |
| > | greater than | x > 8 | false | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison8) |
| < | less than | x < 8 | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison9) |
| >= | greater than or equal to | x >= 8 | false | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison10) |
| <= | less than or equal to | x <= 8 | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison11) |

How Can it be Used

Comparison operators can be used in conditional statements to compare values and take action depending on the result:

if (age < 18) text = "Too young";

You will learn more about the use of conditional statements in the next chapter of this tutorial.

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that **x = 6** and **y = 3**, the table below explains the logical operators:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| && | and | (x < 10 && y > 1) is true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_and) |
| || | or | (x == 5 || y == 5) is false | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_or) |
| ! | not | !(x == y) is true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_not) |

Conditional (Ternary) Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

*variablename*= (*condition*) ?*value1*:*value2*

Example

var voteable = (age < 18) ? "Too young":"Old enough";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison)

If the variable age is a value below 18, the value of the variable voteable will be "Too young", otherwise the value of voteable will be "Old enough".

Comparing Different Types

Comparing data of different types may give unexpected results.

When comparing a string with a number, JavaScript will convert the string to a number when doing the comparison. An empty string converts to 0. A non-numeric string converts to NaN which is always false.

|  |  |  |
| --- | --- | --- |
| **Case** | **Value** | **Try** |
| 2 < 12 | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_20) |
| 2 < "12" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_21) |
| 2 < "John" | false | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_23) |
| 2 > "John" | false | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_24) |
| 2 == "John" | false | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_25) |
| "2" < "12" | false | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_26) |
| "2" > "12" | true | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_27) |
| "2" == "12" | false | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_28) |

When comparing two strings, "2" will be greater than "12", because (alphabetically) 1 is less than 2.

To secure a proper result, variables should be converted to the proper type before comparison:

age = Number(age);  
if (isNaN(age)) {  
    voteable = "Input is not a number";  
} else {  
    voteable = (age < 18) ? "Too young" : "Old enough";  
}

32.JS CONDITION

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

## Conditional Statements

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

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

In JavaScript we have the following conditional statements:

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

The switch statement is described in the next chapter.

## The if Statement

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

### Syntax

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

Note that **if** is in lowercase letters. Uppercase letters (If or IF) will generate a JavaScript error.

### Example

Make a "Good day" greeting if the hour is less than 18:00:

if (hour < 18) {  
    greeting = "Good day";  
}

The result of greeting will be:

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_ifthen)

## The else Statement

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

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

### Example

If the hour is less than 18, create a "Good day" greeting, otherwise "Good evening":

if (hour < 18) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}

The result of greeting will be:

Good evening

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_ifthenelse)

## The else if Statement

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

### Syntax

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

### Example

If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 20:00, create a "Good day" greeting, otherwise a "Good evening":

if (time < 10) {  
    greeting = "Good morning";  
} else if (time < 20) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}

The result of greeting will be:

Good evening

33.JS Switch

b is done, it's time for a break. There is no need for more testing.

A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

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

## The default Keyword

The **default** keyword specifies the code to run if there is no case match:

### Example

The getDay() method returns the weekday as a number between 0 and 6.

If today is neither Saturday (6) nor Sunday (0), write a default message:

switch (new Date().getDay()) {  
    case 6:  
        text = "Today is Saturday";  
        break;   
    case 0:  
        text = "Today is Sunday";  
        break;   
    default:   
        text = "Looking forward to the Weekend";  
}

The result of text will be:

Looking forward to the Weekend

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_switch2)

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

### Example

switch (new Date().getDay()) {  
    default:   
        text = "Looking forward to the Weekend";  
        break;  
    case 6:  
        text = "Today is Saturday";  
        break;   
    case 0:  
        text = "Today is Sunday";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_switch4)

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

## Common Code Blocks

Sometimes you will want different switch cases to use the same code.

In this example case 4 and 5 share the same code block, and 0 and 6 share another code block:

### Example

switch (new Date().getDay()) {  
    case 4:  
    case 5:  
        text = "Soon it is Weekend";  
        break;   
    case 0:  
    case 6:  
        text = "It is Weekend";  
        break;  
    default:   
        text = "Looking forward to the Weekend";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_switch3)

## Switching Details

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

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

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

## Strict Comparison

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

The values must be of the same type to match.

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

In this example there will be no match for x:

### Example

var x = "0";  
switch (x) {  
    case 0:  
        text = "Off";  
        break;  
    case 1:  
        text = "On";  
        break;  
    default:  
        text = "No value found";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_switch5)

34.JS Loop For

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

JavaScript Loops

Loops are handy, if you want to run the same code over and over again, each time with a different value.

Often this is the case when working with arrays:

Instead of writing:

text += cars[0] + "<br>";   
text += cars[1] + "<br>";   
text += cars[2] + "<br>";   
text += cars[3] + "<br>";   
text += cars[4] + "<br>";   
text += cars[5] + "<br>";

You can write:

var i;  
for (i = 0; i < cars.length; i++) {   
    text += cars[i] + "<br>";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_loop_for)

Different Kinds of Loops

JavaScript supports different kinds of loops:

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

The For Loop

The for loop has the following syntax:

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

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

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

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

Example

for (i = 0; i < 5; i++) {  
    text += "The number is " + i + "<br>";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_loop_for_ex)

From the example above, you can read:

Statement 1 sets a variable before the loop starts (var i = 0).

Statement 2 defines the condition for the loop to run (i must be less than 5).

Statement 3 increases a value (i++) each time the code block in the loop has been executed.

Statement 1

Normally you will use statement 1 to initialize the variable used in the loop (i = 0).

This is not always the case, JavaScript doesn't care. Statement 1 is optional.

You can initiate many values in statement 1 (separated by comma):

Example

for (i = 0, len = cars.length, text = ""; i < len; i++) {   
    text += cars[i] + "<br>";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_loop_for_om1)

And you can omit statement 1 (like when your values are set before the loop starts):

Example

var i = 2;  
var len = cars.length;  
var text = "";  
for (; i < len; i++) {   
    text += cars[i] + "<br>";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_loop_for_om2)

Statement 2

Often statement 2 is used to evaluate the condition of the initial variable.

This is not always the case, JavaScript doesn't care. Statement 2 is also optional.

If statement 2 returns true, the loop will start over again, if it returns false, the loop will end.

If you omit statement 2, you must provide a **break** inside the loop. Otherwise the loop will never end. This will crash your browser. Read about breaks in a later chapter of this tutorial.

Statement 3

Often statement 3 increments the value of the initial variable.

This is not always the case, JavaScript doesn't care, and statement 3 is optional.

Statement 3 can do anything like negative increment (i--), positive increment (i = i + 15), or anything else.

Statement 3 can also be omitted (like when you increment your values inside the loop):

Example

var i = 0;  
var len = cars.length;  
for (; i < len; ) {   
    text += cars[i] + "<br>";  
    i++;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_loop_for_om3)

The For/In Loop

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

Example

var person = {fname:"John", lname:"Doe", age:25};   
  
var text = "";  
var x;  
for (x in person) {  
    text += person[x];  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_for_in)

35.JS Loop While

Loops can execute a block of code as long as a specified condition is true.

## The While Loop

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

### Syntax

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

### Example

In the following example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:

### Example

while (i < 10) {  
    text += "The number is " + i;  
    i++;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_while)

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

## The Do/While Loop

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

### Syntax

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

### Example

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

### Example

do {  
    text += "The number is " + i;  
    i++;  
}  
while (i < 10);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_dowhile)

Do not forget to increase the variable used in the condition, otherwise the loop will never end!

## Comparing For and While

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

The loop in this example uses a **for loop** to collect the car names from the cars array:

### Example

var cars = ["BMW", "Volvo", "Saab", "Ford"];  
var i = 0;  
var text = "";  
  
for (;cars[i];) {  
    text += cars[i] + "<br>";  
    i++;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_loop_for_cars)

The loop in this example uses a **while loop** to collect the car names from the cars array:

### Example

var cars = ["BMW", "Volvo", "Saab", "Ford"];  
var i = 0;  
var text = "";  
  
while (cars[i]) {  
    text += cars[i] + "<br>";  
    i++;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_loop_while_cars)

35.JS Break

The break statement "jumps out" of a loop.

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

## The Break Statement

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch() statement.

The break statement can also be used to jump out of a loop.

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

### Example

for (i = 0; i < 10; i++) {  
    if (i === 3) { break; }  
    text += "The number is " + i + "<br>";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_break)

## The Continue Statement

The **continue statement** breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 3:

### Example

for (i = 0; i < 10; i++) {  
    if (i === 3) { continue; }  
    text += "The number is " + i + "<br>";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_continue)

## JavaScript Labels

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

label:  
statements

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

Syntax:

break labelname;   
  
continue labelname;

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

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

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

### Example

var cars = ["BMW", "Volvo", "Saab", "Ford"];  
list: {  
    text += cars[0] + "<br>";   
    text += cars[1] + "<br>";   
    break list;  
    text += cars[2] + "<br>";   
    text += cars[3] + "<br>";   
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_break_list)

35.JS Type Conversion

d.getTime()        // returns 1404568027739

## Automatic Type Conversion

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

The result is not always what you expect:

5 + null    // returns 5         because null is converted to 0  
"5" + null  // returns "5null"   because null is converted to "null"  
"5" + 2     // returns "52"      because 2 is converted to "2"  
"5" - 2     // returns 3         because "5" is converted to 5  
"5" \* "2"   // returns 10        because "5" and "2" are converted to 5 and 2

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_auto)

## Automatic String Conversion

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

document.getElementById("demo").innerHTML = myVar;  
  
// if myVar = {name:"Fjohn"}  // toString converts to "[object Object]"  
// if myVar = [1,2,3,4]       // toString converts to "1,2,3,4"  
// if myVar = new Date()      // toString converts to "Fri Jul 18 2014 09:08:55 GMT+0200"

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

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

## JavaScript Type Conversion Table

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

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

Values in quotes indicate string values.

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

36.JS Let

## ECMAScript 2015

ES2015 introduced two important new JavaScript keywords: **let** and **const**.

These two keywords provide **Block Scope** variables (and constants) in JavaScript.

Before ES2015, JavaScript had only two types of scope: **Global Scope** and **Function Scope**.

## Global Scope

Variables declared **Globally** (outside any function) have **Global Scope**.

### Example

var carName = "Volvo";  
  
// code here can use carName  
  
function myFunction() {  
    // code here can also use carName   
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_scope_global)

**Global**variables can be accessed from anywhere in a JavaScript program.

## Function Scope

Variables declared **Locally** (inside a function) have **Function Scope**.

### Example

// code here can NOT use carName  
  
function myFunction() {  
    var carName = "Volvo";  
    // code here CAN use carName  
}  
  
// code here can NOT use carName

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_scope_local)

**Local** variables can only be accessed from inside the function where they are declared.

## JavaScript Block Scope

Variables declared with the **var keyword** can not have **Block Scope**.

Variables declared inside a block **{}** can be accessed from outside the block.

### Example

{   
    var x = 2;   
}  
// x CAN be used here

Before ES2015 JavaScript did not have **Block Scope**.

Variables declared with the **let keyword** can have Block Scope.

Variables declared inside a block **{}** can not be accessed from outside the block:

### Example

{   
    let x = 2;  
}  
// x can NOT be used here

## Redeclaring Variables

Redeclaring a variable using the **var keyword** can impose problems.

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

### Example

var x = 10;  
// Here x is 10  
{   
    var x = 2;  
    // Here x is 2  
}  
// Here x is 2

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_es6_var)

Redeclaring a variable using the **let keyword** can solve this problem.

Redeclaring a variable inside a block will not redeclare the variable outside the block:

### Example

var x = 10;  
// Here x is 10  
{   
    let x = 2;  
    // Here x is 2  
}  
// Here x is 10

37.JS Const

## ECMAScript 2015

ES2015 intoduced two important new JavaScript keywords: **let** and **const**.

Variables defined with **const** behave like **let** variables, except they cannot be reassigned:

### Example

const PI = 3.141592653589793;  
PI = 3.14;      // This will give an error  
PI = PI + 10;   // This will also give an error

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_const_value)

## Block Scope

Declaring a variable with **const** is similar to **let** when it comes to **Block Scope**.

The x declared in the block, in this example, is not the same as the x declared outside the block:

### Example

var x = 10;  
// Here x is 10  
{   
    const x = 2;  
    // Here x is 2  
}  
// Here x is 10

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_const)

You can learn more about [Block Scope](https://www.w3schools.com/js/js_let.asp) in the previous chapter: [JavaScript Let](https://www.w3schools.com/js/js_let.asp).

## Assigned when Declared

JavaScript const variables must be assigned a value when they are declared:

### Incorrect

const PI;  
PI = 3.14159265359;

### Correct

const PI = 3.14159265359;

## Not Real Constants

The keyword const is a little misleading.

It does NOT define a constant value. It defines a constant reference to a value.

Because of this, we cannot change constant primitive values, but we can change the properties of constant objects.

## Primitive Values

If we assign a primitive value to a constant, we cannot change the primitive value:

### Example

const PI = 3.141592653589793;  
PI = 3.14;      // This will give an error  
PI = PI + 10;   // This will also give an error

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_const_value)

## Constant Objects can Change

You can change the properties of a constant object:

### Example

// You can create a const object:  
const car = {type:"Fiat", model:"500", color:"white"};  
  
// You can change a property:  
car.color = "red";  
  
// You can add a property:  
car.owner = "Johnson";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_const_object)

But you can NOT reassign a constant object:

### Example

const car = {type:"Fiat", model:"500", color:"white"};  
car = {type:"Volvo", model:"EX60", color:"red"};    // ERROR

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_const_object_assign)

## Constant Arrays can Change

You can change the elements of a constant array:

### Example

// You can create a constant array:  
const cars = ["Saab", "Volvo", "BMW"];  
  
// You can change an element:  
cars[0] = "Toyota";  
  
// You can add an element:  
cars.push("Audi");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_const_array)

But you can NOT reassign a constant array:

### Example

const cars = ["Saab", "Volvo", "BMW"];  
cars = ["Toyota", "Volvo", "Audi"];    // ERROR

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_const_array_assign)

38.JS Style Guide

Always use the same coding conventions for all your JavaScript projects.

JavaScript Coding Conventions

Coding conventions are **style guidelines for programming**. They typically cover:

* Naming and declaration rules for variables and functions.
* Rules for the use of white space, indentation, and comments.
* Programming practices and principles

Coding conventions **secure quality**:

* Improves code readability
* Make code maintenance easier

Coding conventions can be documented rules for teams to follow, or just be your individual coding practice.

This page describes the general JavaScript code conventions used by W3Schools.  
You should also read the next chapter "Best Practices", and learn how to avoid coding pitfalls.

Variable Names

At W3schools we use **camelCase** for identifier names (variables and functions).

All names start with a **letter**.

At the bottom of this page, you will find a wider discussion about naming rules.

firstName = "John";  
lastName = "Doe";  
  
price = 19.90;  
tax = 0.20;  
  
fullPrice = price + (price \* tax);

Spaces Around Operators

Always put spaces around operators ( = + - \* / ), and after commas:

Examples:

var x = y + z;  
var values = ["Volvo", "Saab", "Fiat"];

Code Indentation

Always use 4 spaces for indentation of code blocks:

Functions:

function toCelsius(fahrenheit) {  
    return (5 / 9) \* (fahrenheit - 32);  
}

Do not use tabs (tabulators) for indentation. Different editors interpret tabs differently.

Statement Rules

General rules for simple statements:

* Always end a simple statement with a semicolon.

Examples:

var values = ["Volvo", "Saab", "Fiat"];  
  
var person = {  
    firstName: "John",  
    lastName: "Doe",  
    age: 50,  
    eyeColor: "blue"  
};

General rules for complex (compound) statements:

* Put the opening bracket at the end of the first line.
* Use one space before the opening bracket.
* Put the closing bracket on a new line, without leading spaces.
* Do not end a complex statement with a semicolon.

Functions:

function toCelsius(fahrenheit) {  
    return (5 / 9) \* (fahrenheit - 32);  
}

Loops:

for (i = 0; i < 5; i++) {  
    x += i;  
}

Conditionals:

if (time < 20) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}

Object Rules

General rules for object definitions:

* Place the opening bracket on the same line as the object name.
* Use colon plus one space between each property and its value.
* Use quotes around string values, not around numeric values.
* Do not add a comma after the last property-value pair.
* Place the closing bracket on a new line, without leading spaces.
* Always end an object definition with a semicolon.

Example

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

Short objects can be written compressed, on one line, using spaces only between properties, like this:

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

Line Length < 80

For readability, avoid lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it, is after an operator or a comma.

Example

document.getElementById("demo").innerHTML =  
    "Hello Dolly.";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_line_break)

Naming Conventions

Always use the same naming convention for all your code. For example:

* Variable and function names written as **camelCase**
* Global variables written in **UPPERCASE**(We don't, but it's quite common)
* Constants (like PI) written in **UPPERCASE**

Should you use **hyp-hens**, **camelCase**, or **under\_scores** in variable names?

This is a question programmers often discuss. The answer depends on who you ask:

**Hyphens in HTML and CSS:**

HTML5 attributes can start with data- (data-quantity, data-price).

CSS uses hyphens in property-names (font-size).

Hyphens can be mistaken as subtraction attempts. Hyphens are not allowed in JavaScript names.

**Underscores:**

Many programmers prefer to use underscores (date\_of\_birth), especially in SQL databases.

Underscores are often used in PHP documentation.

**PascalCase:**

PascalCase is often preferred by C programmers.

**camelCase:**

camelCase is used by JavaScript itself, by jQuery, and other JavaScript libraries.

Do not start names with a $ sign. It will put you in conflict with many JavaScript library names.

Loading JavaScript in HTML

Use simple syntax for loading external scripts (the type attribute is not necessary):

<script src="myscript.js"></script>

Accessing HTML Elements

A consequence of using "untidy" HTML styles, might result in JavaScript errors.

These two JavaScript statements will produce different results:

var obj = getElementById("Demo")  
  
var obj = getElementById("demo")

If possible, use the same naming convention (as JavaScript) in HTML.

[Visit the HTML Style Guide](https://www.w3schools.com/html/html5_syntax.asp).

File Extensions

HTML files should have a **.html** extension (not **.htm**).

CSS files should have a **.css** extension.

JavaScript files should have a **.js** extension.

Use Lower Case File Names

Most web servers (Apache, Unix) are case sensitive about file names:

london.jpg cannot be accessed as London.jpg.

Other web servers (Microsoft, IIS) are not case sensitive:

london.jpg can be accessed as London.jpg or london.jpg.

If you use a mix of upper and lower case, you have to be extremely consistent.

If you move from a case insensitive, to a case sensitive server, even small errors can break your web site.

To avoid these problems, always use lower case file names (if possible).

Performance

Coding conventions are not used by computers. Most rules have little impact on the execution of programs.

Indentation and extra spaces are not significant in small scripts.

For code in development, readability should be preferred. Larger production scripts should be minified.

39.JS Best Practices

Avoid global variables,  avoid new,  avoid  ==,  avoid eval()

Avoid Global Variables

Minimize the use of global variables.

This includes all data types, objects, and functions.

Global variables and functions can be overwritten by other scripts.

Use local variables instead, and learn how to use [closures](https://www.w3schools.com/js/js_function_closures.asp).

Always Declare Local Variables

All variables used in a function should be declared as **local** variables.

Local variables **must** be declared with the **var**keyword, otherwise they will become global variables.

Strict mode does not allow undeclared variables.

Declarations on Top

It is a good coding practice to put all declarations at the top of each script or function.

This will:

* Give cleaner code
* Provide a single place to look for local variables
* Make it easier to avoid unwanted (implied) global variables
* Reduce the possibility of unwanted re-declarations

// Declare at the beginning  
var firstName, lastName, price, discount, fullPrice;  
  
// Use later  
firstName = "John";  
lastName = "Doe";  
  
price = 19.90;  
discount = 0.10;  
  
fullPrice = price \* 100 / discount;

This also goes for loop variables:

// Declare at the beginning  
var i;  
  
// Use later  
for (i = 0; i < 5; i++) {

By default, JavaScript moves all declarations to the top ([JavaScript Hoisting](https://www.w3schools.com/js/js_hoisting.asp)).

Initialize Variables

It is a good coding practice to initialize variables when you declare them.

This will:

* Give cleaner code
* Provide a single place to initialize variables
* Avoid undefined values

// Declare and initiate at the beginning  
var firstName = "",  
    lastName = "",  
    price = 0,  
    discount = 0,  
    fullPrice = 0,  
    myArray = [],  
    myObject = {};

Initializing variables provides an idea of the intended use (and intended data type).

Never Declare Number, String, or Boolean Objects

Always treat numbers, strings, or booleans as primitive values. Not as objects.

Declaring these types as objects, slows down execution speed, and produces nasty side effects:

Example

var x = "John";               
var y = new String("John");  
(x === y) // is false because x is a string and y is an object.

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_best_object_string1)

Or even worse:

Example

var x = new String("John");               
var y = new String("John");  
(x == y) // is false because you cannot compare objects.

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_best_object_string2)

Don't Use new Object()

* Use {} instead of new Object()
* Use "" instead of new String()
* Use 0 instead of new Number()
* Use false instead of new Boolean()
* Use [] instead of new Array()
* Use /()/ instead of new RegExp()
* Use function (){} instead of new Function()

Example

var x1 = {};           // new object  
var x2 = "";           // new primitive string  
var x3 = 0;            // new primitive number  
var x4 = false;        // new primitive boolean  
var x5 = [];           // new array object  
var x6 = /()/;         // new regexp object  
var x7 = function(){}; // new function object

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_best_constructors)

Beware of Automatic Type Conversions

Beware that numbers can accidentally be converted to strings or NaN (Not a Number).

JavaScript is loosely typed. A variable can contain different data types, and a variable can change its data type:

Example

var x = "Hello";     // typeof x is a string  
x = 5;               // changes typeof x to a number

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_best_typeof)

When doing mathematical operations, JavaScript can convert numbers to strings:

Example

var x = 5 + 7;       // x.valueOf() is 12,  typeof x is a number  
var x = 5 + "7";     // x.valueOf() is 57,  typeof x is a string  
var x = "5" + 7;     // x.valueOf() is 57,  typeof x is a string  
var x = 5 - 7;       // x.valueOf() is -2,  typeof x is a number  
var x = 5 - "7";     // x.valueOf() is -2,  typeof x is a number  
var x = "5" - 7;     // x.valueOf() is -2,  typeof x is a number  
var x = 5 - "x";     // x.valueOf() is NaN, typeof x is a number

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_best_valueof)

Subtracting a string from a string, does not generate an error but returns NaN (Not a Number):

Example

"Hello" - "Dolly"    // returns NaN

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_best_string_op1)

Use === Comparison

The == comparison operator always converts (to matching types) before comparison.

The === operator forces comparison of values and type:

Example

0 == "";        // true  
1 == "1";       // true  
1 == true;      // true  
  
0 === "";       // false  
1 === "1";      // false  
1 === true;     // false

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_best_comparison)

Use Parameter Defaults

If a function is called with a missing argument, the value of the missing argument is set to **undefined**.

Undefined values can break your code. It is a good habit to assign default values to arguments.

Example

function myFunction(x, y) {  
    if (y === undefined) {  
        y = 0;  
    }  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_best_parameter_default)

Read more about function parameters and arguments at [Function Parameters](https://www.w3schools.com/js/js_function_parameters.asp)

End Your Switches with Defaults

Always end your switch statements with a default. Even if you think there is no need for it.

Example

switch (new Date().getDay()) {  
    case 0:  
        day = "Sunday";  
        break;  
    case 1:  
        day = "Monday";  
        break;  
    case 2:  
        day = "Tuesday";  
        break;  
    case 3:  
        day = "Wednesday";  
        break;  
    case 4:  
        day = "Thursday";  
        break;  
    case 5:  
        day = "Friday";  
        break;  
    case 6:  
        day = "Saturday";  
        break;  
    default:  
        day = "Unknown";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_break_switch)

Avoid Using eval()

The eval() function is used to run text as code. In almost all cases, it should not be necessary to use it.

Because it allows arbitrary code to be run, it also represents a security problem

40.JS Json

JSON is a format for storing and transporting data.

JSON is often used when data is sent from a server to a web page.

What is JSON?

* JSON stands for **J**ava**S**cript **O**bject **N**otation
* JSON is lightweight data interchange format
* JSON is language independent **\***
* JSON is "self-describing" and easy to understand

\* The JSON syntax is derived from JavaScript object notation syntax, but the JSON format is text only. Code for reading and generating JSON data can be written in any programming language.

JSON Example

This JSON syntax defines an employees object: an array of 3 employee records (objects):

JSON Example

{  
"employees":[  
    {"firstName":"John", "lastName":"Doe"},   
    {"firstName":"Anna", "lastName":"Smith"},  
    {"firstName":"Peter", "lastName":"Jones"}  
]  
}

The JSON Format Evaluates to JavaScript Objects

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

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

JSON Syntax Rules

* Data is in name/value pairs
* Data is separated by commas
* Curly braces hold objects
* Square brackets hold arrays

JSON Data - A Name and a Value

JSON data is written as name/value pairs, just like JavaScript object properties.

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

"firstName":"John"

JSON names require double quotes. JavaScript names do not.

JSON Objects

JSON objects are written inside curly braces.

Just like in JavaScript, objects can contain multiple name/value pairs:

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

JSON Arrays

JSON arrays are written inside square brackets.

Just like in JavaScript, an array can contain objects:

"employees":[  
    {"firstName":"John", "lastName":"Doe"},   
    {"firstName":"Anna", "lastName":"Smith"},   
    {"firstName":"Peter", "lastName":"Jones"}  
]

In the example above, the object "employees" is an array. It contains three objects.

Each object is a record of a person (with a first name and a last name).

Converting a JSON Text to a JavaScript Object

A common use of JSON is to read data from a web server, and display the data in a web page.

For simplicity, this can be demonstrated using a string as input.

First, create a JavaScript string containing JSON syntax:

var text = '{ "employees" : [' +  
'{ "firstName":"John" , "lastName":"Doe" },' +  
'{ "firstName":"Anna" , "lastName":"Smith" },' +  
'{ "firstName":"Peter" , "lastName":"Jones" } ]}';

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

var obj = JSON.parse(text);

Finally, use the new JavaScript object in your page:

Example

<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML =  
obj.employees[1].firstName + " " + obj.employees[1].lastName;  
</script>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_json_parse)