w3schools.com

THE WORLD'S LARGEST WEB DEVELOPER SITE

Learn How To Create CSS Alert Buttons

Success Info Warning Danger Default

JavaScript Best Practices

❮ Previous Next ❯

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.

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
// 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).

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 = {};
```

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 »

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 »

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
```

Try it Yourself »

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 »

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 »

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 »

Use === Comparison

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

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

```
0 == "";        // true
1 == "1";       // true
1 == true;      // true

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

Try it Yourself »

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 »

Read more about function parameters and arguments at Function Parameters

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";
        day = "Wednesday";
        break;
    case 4:
        day = "Thursday";
        break;
    case 5:
        day = "Friday";
        break;
    case 6:
        day = "Saturday";
        break;
    default:
        day = "Unknown";
}
```

Try it Yourself »

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.

❮ Previous Next ❯