w3schools.com

THE WORLD'S LARGEST WEB DEVELOPER SITE

JavaScript Common Mistakes

❮ Previous Next ❯

This chapter points out some common JavaScript mistakes.

Accidentally Using the Assignment Operator

JavaScript programs may generate unexpected results if a programmer accidentally uses an assignment operator (=), instead of a comparison operator (==) in an if statement.

This **if** statement returns **false** (as expected) because x is not equal to 10:

```
var x = 0;
if (x == 10)
```

Try it Yourself »

This **if** statement returns **true** (maybe not as expected), because 10 is true:

```
var x = 0;
if (x = 10)
```

Try it Yourself »

This **if** statement returns **false** (maybe not as expected), because 0 is false:

```
var x = 0;
if (x = 0)
```

Try it Yourself »

Expecting Loose Comparison

In regular comparison, data type does not matter. This if statement returns true:

```
var x = 10;
var y = "10";
if (x == y)
```

Try it Yourself »

In strict comparison, data type does matter. This if statement returns false:

```
var x = 10;
var y = "10";
if (x === y)
```

Try it Yourself »

It is a common mistake to forget that switch statements use strict comparison:

This case switch will display an alert:

```
var x = 10;
switch(x) {
    case 10: alert("Hello");
}
```

Try it Yourself »

This case switch will not display an alert:

```
var x = 10;
switch(x) {
    case "10": alert("Hello");
}
```

Try it Yourself »

HTML CSS JAVASCRIPT MORE REFERENCES

Confusing Addition & Concatenation

Addition is about adding **numbers**.

Concatenation is about adding **strings**.

In JavaScript both operations use the same + operator.

Because of this, adding a number as a number will produce a different result from adding a number as a string:

```
var x = 10 + 5;          // the result in x is 15
var x = 10 + "5";        // the result in x is "105"
```

Try it Yourself »

When adding two variables, it can be difficult to anticipate the result:

```
var x = 10;
var y = 5;
var z = x + y;           // the result in z is 15

var x = 10;
var y = "5";
var z = x + y;           // the result in z is "105"
```

Try it Yourself »

Misunderstanding Floats

All numbers in JavaScript are stored as 64-bits **Floating point numbers** (Floats).

All programming languages, including JavaScript, have difficulties with precise floating point values:

```
var x = 0.1;
var y = 0.2;
var z = x + y            // the result in z will not be 0.3
```

Try it Yourself »

To solve the problem above, it helps to multiply and divide:

```
var z = (x * 10 + y * 10) / 10;       // z will be 0.3
```

Try it Yourself »

Breaking a JavaScript String

JavaScript will allow you to break a statement into two lines:

Example 1

```
var x =
"Hello World!";
```

Try it Yourself »

But, breaking a statement in the middle of a string will not work:

Example 2

```
var x = "Hello
World!";
```

Try it Yourself »

You must use a "backslash" if you must break a statement in a string:

Example 3

```
var x = "Hello \
World!";
```

Try it Yourself »

Misplacing Semicolon

Because of a misplaced semicolon, this code block will execute regardless of the value of x:

```
if (x == 19);
{
```

Try it Yourself »

Breaking a Return Statement

It is a default JavaScript behavior to close a statement automatically at the end of a line.

Because of this, these two examples will return the same result:

Example 1

```
function myFunction(a) {
    var power = 10
    return a * power
}
```

Try it Yourself »

Example 2

```
function myFunction(a) {
    var power = 10;
    return a * power;
}
```

Try it Yourself »

JavaScript will also allow you to break a statement into two lines.

Because of this, example 3 will also return the same result:

Example 3

```
function myFunction(a) {
    var
    power = 10;
    return a * power;
}
```

Try it Yourself »

But, what will happen if you break the return statement in two lines like this:

```
function myFunction(a) {
    var
    power = 10;
    return
    a * power;
}
```

Try it Yourself »

The function will return undefined!

Why? Because JavaScript thinks you meant:

Example 5

```
function myFunction(a) {
    var
    power = 10;
    return;
    a * power;
}
```

Try it Yourself »

Explanation

If a statement is incomplete like:

```
var
```

JavaScript will try to complete the statement by reading the next line:

```
power = 10;
```

But since this statement is complete:

```
return
```

JavaScript will automatically close it like this:

This happens because closing (ending) statements with semicolon is optional in JavaScript.

JavaScript will close the return statement at the end of the line, because it is a complete statement.

Never break a return statement.

Accessing Arrays with Named Indexes

Many programming languages support arrays with named indexes.

Arrays with named indexes are called associative arrays (or hashes).

JavaScript does **not** support arrays with named indexes.

In JavaScript, **arrays** use **numbered indexes**:

Example

```
var person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
var x = person.length;         // person.length will return 3
var y = person[0];             // person[0] will return "John"
```

Try it Yourself »

In JavaScript, **objects** use **named indexes**.

If you use a named index, when accessing an array, JavaScript will redefine the array to a standard object.

After the automatic redefinition, array methods and properties will produce undefined or incorrect results:

Example:

```
var person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
var x = person.length;         // person.length will return 0
var y = person[0];             // person[0] will return undefined
```

Try it Yourself »

Trailing commas in object and array definition are legal in ECMAScript 5.

Object Example:

```
person = {firstName:"John", lastName:"Doe", age:46,}
```

Array Example:

```
points = [40, 100, 1, 5, 25, 10,];
```

WARNING !!

Internet Explorer 8 will crash.

JSON does not allow trailing commas.

JSON:

```
person = {firstName:"John", lastName:"Doe", age:46}
```

JSON:

```
points = [40, 100, 1, 5, 25, 10];
```

Undefined is Not Null

With JavaScript, **null** is for objects, **undefined** is for variables, properties, and methods.

To be null, an object has to be defined, otherwise it will be undefined.

If you want to test if an object exists, this will throw an error if the object is undefined:

Incorrect:

```
if (myObj !== null && typeof myObj !== "undefined")
```

Because of this, you must test typeof() first:

Correct:

HTML CSS JAVASCRIPT MORE REFERENCES

Expecting Block Level Scope

JavaScript **does not** create a new scope for each code block.

It is true in many programming languages, but **not true** in JavaScript.

This code will display the value of i (10), even OUTSIDE the for loop block:

Example

```
for (var i = 0; i < 10; i++) {
    // some code
}
return i;
```

Try it Yourself »

❮ Previous Next ❯

Templates

by

W3.CSS