**JavaScript specials**

This chapter briefly recaps the features of JavaScript that we’ve learned by now, paying special attention to subtle moments.

**[Code structure](https://javascript.info/javascript-specials" \l "code-structure)**

Statements are delimited with a semicolon:

1. alert('Hello'); alert('World');

Usually, a line-break is also treated as a delimiter, so that would also work:

1. alert('Hello')
2. alert('World')

That’s called “automatic semicolon insertion”. Sometimes it doesn’t work, for instance:

1. alert("There will be an error after this message")
2. [1, 2].forEach(alert)

Most codestyle guides agree that we should put a semicolon after each statement.

Semicolons are not required after code blocks {...} and syntax constructs with them like loops:

1. function f() {
2. // no semicolon needed after function declaration
3. }
4. for(;;) {
5. // no semicolon needed after the loop
6. }

…But even if we can put an “extra” semicolon somewhere, that’s not an error. It will be ignored.

More in: [Code structure](https://javascript.info/structure).

**[Strict mode](https://javascript.info/javascript-specials" \l "strict-mode)**

To fully enable all features of modern JavaScript, we should start scripts with "use strict".

1. 'use strict';
2. ...

The directive must be at the top of a script or at the beginning of a function body.

Without "use strict", everything still works, but some features behave in the old-fashioned, “compatible” way. We’d generally prefer the modern behavior.

Some modern features of the language (like classes that we’ll study in the future) enable strict mode implicitly.

More in: [The modern mode, "use strict"](https://javascript.info/strict-mode).

**[Variables](https://javascript.info/javascript-specials" \l "variables)**

Can be declared using:

* let
* const (constant, can’t be changed)
* var (old-style, will see later)

A variable name can include:

* Letters and digits, but the first character may not be a digit.
* Characters $ and \_ are normal, on par with letters.
* Non-Latin alphabets and hieroglyphs are also allowed, but commonly not used.

Variables are dynamically typed. They can store any value:

1. let x = 5;
2. x = "John";

There are 8 data types:

* number for both floating-point and integer numbers,
* bigint for integer numbers of arbitrary length,
* string for strings,
* boolean for logical values: true/false,
* null – a type with a single value null, meaning “empty” or “does not exist”,
* undefined – a type with a single value undefined, meaning “not assigned”,
* object and symbol – for complex data structures and unique identifiers, we haven’t learnt them yet.

The typeof operator returns the type for a value, with two exceptions:

1. typeof null == "object" // error in the language
2. typeof function(){} == "function" // functions are treated specially

More in: [Variables](https://javascript.info/variables) and [Data types](https://javascript.info/types).

**[Interaction](https://javascript.info/javascript-specials" \l "interaction)**

We’re using a browser as a working environment, so basic UI functions will be:

[**prompt(question, [default])**](https://developer.mozilla.org/en-US/docs/Web/API/Window/prompt)

Ask a question, and return either what the visitor entered or null if they clicked “cancel”.

[**confirm(question)**](https://developer.mozilla.org/en-US/docs/Web/API/Window/confirm)

Ask a question and suggest to choose between Ok and Cancel. The choice is returned as true/false.

[**alert(message)**](https://developer.mozilla.org/en-US/docs/Web/API/Window/alert)

Output a message.

All these functions are *modal*, they pause the code execution and prevent the visitor from interacting with the page until they answer.

For instance:

1. let userName = prompt("Your name?", "Alice");
2. let isTeaWanted = confirm("Do you want some tea?");
3. alert( "Visitor: " + userName ); // Alice
4. alert( "Tea wanted: " + isTeaWanted ); // true

More in: [Interaction: alert, prompt, confirm](https://javascript.info/alert-prompt-confirm).

**[Operators](https://javascript.info/javascript-specials" \l "operators)**

JavaScript supports the following operators:

**Arithmetical**

Regular: \* + - /, also % for the remainder and \*\* for power of a number.

The binary plus + concatenates strings. And if any of the operands is a string, the other one is converted to string too:

1. alert( '1' + 2 ); // '12', string
2. alert( 1 + '2' ); // '12', string

**Assignments**

There is a simple assignment: a = b and combined ones like a \*= 2.

**Bitwise**

Bitwise operators work with 32-bit integers at the lowest, bit-level: see the [docs](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators#bitwise_operators) when they are needed.

**Conditional**

The only operator with three parameters: cond ? resultA : resultB. If cond is truthy, returns resultA, otherwise resultB.

**Logical operators**

Logical AND && and OR || perform short-circuit evaluation and then return the value where it stopped (not necessary true/false). Logical NOT ! converts the operand to boolean type and returns the inverse value.

**Nullish coalescing operator**

The ?? operator provides a way to choose a defined value from a list of variables. The result of a ?? b is a unless it’s null/undefined, then b.

**Comparisons**

Equality check == for values of different types converts them to a number (except null and undefined that equal each other and nothing else), so these are equal:

1. alert( 0 == false ); // true
2. alert( 0 == '' ); // true

Other comparisons convert to a number as well.

The strict equality operator === doesn’t do the conversion: different types always mean different values for it.

Values null and undefined are special: they equal == each other and don’t equal anything else.

Greater/less comparisons compare strings character-by-character, other types are converted to a number.

**Other operators**

There are few others, like a comma operator.

More in: [Basic operators, maths](https://javascript.info/operators), [Comparisons](https://javascript.info/comparison), [Logical operators](https://javascript.info/logical-operators), [Nullish coalescing operator '??'](https://javascript.info/nullish-coalescing-operator).

**[Loops](https://javascript.info/javascript-specials" \l "loops)**

* We covered 3 types of loops:

1. // 1
2. while (condition) {
3. ...
4. }
5. // 2
6. do {
7. ...
8. } while (condition);
9. // 3
10. for(let i = 0; i < 10; i++) {
11. ...
12. }

* The variable declared in for(let...) loop is visible only inside the loop. But we can also omit let and reuse an existing variable.
* Directives break/continue allow to exit the whole loop/current iteration. Use labels to break nested loops.

Details in: [Loops: while and for](https://javascript.info/while-for).

Later we’ll study more types of loops to deal with objects.

**[The “switch” construct](https://javascript.info/javascript-specials" \l "the-switch-construct)**

The “switch” construct can replace multiple if checks. It uses === (strict equality) for comparisons.

For instance:

1. let age = prompt('Your age?', 18);
2. switch (age) {
3. case 18:
4. alert("Won't work"); // the result of prompt is a string, not a number
5. break;
6. case "18":
7. alert("This works!");
8. break;
9. default:
10. alert("Any value not equal to one above");
11. }

Details in: [The "switch" statement](https://javascript.info/switch).

**[Functions](https://javascript.info/javascript-specials" \l "functions)**

We covered three ways to create a function in JavaScript:

1. Function Declaration: the function in the main code flow
2. function sum(a, b) {
3. let result = a + b;
4. return result;
5. }
6. Function Expression: the function in the context of an expression
7. let sum = function(a, b) {
8. let result = a + b;
9. return result;
10. };
11. Arrow functions:
12. // expression on the right side
13. let sum = (a, b) => a + b;
14. // or multi-line syntax with { ... }, need return here:
15. let sum = (a, b) => {
16. // ...
17. return a + b;
18. }
19. // without arguments
20. let sayHi = () => alert("Hello");
21. // with a single argument
22. let double = n => n \* 2;

* Functions may have local variables: those declared inside its body or its parameter list. Such variables are only visible inside the function.
* Parameters can have default values: function sum(a = 1, b = 2) {...}.
* Functions always return something. If there’s no return statement, then the result is undefined.

Details: see [Functions](https://javascript.info/function-basics), [Arrow functions, the basics](https://javascript.info/arrow-functions-basics).

**[More to come](https://javascript.info/javascript-specials" \l "more-to-come)**

That was a brief list of JavaScript features. As of now we’ve studied only basics. Further in the tutorial you’ll find more specials and advanced features of JavaScript.

**[Comments](https://javascript.info/javascript-specials" \l "comments)**

read this before commenting…