06 May 2019

JavaScript specials

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

Code structure

Statements are delimited with a semicolon:

```
alert('Hello'); alert('World');
```

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

```
alert('Hello')
alert('World')
```

That's called "automatic semicolon insertion". Sometimes it doesn't work, for instance:

```
alert("There will be an error after this message")

[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:

```
function f() {
  // no semicolon needed after function declaration
}

for(;;) {
  // no semicolon needed after the loop
}
```

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

More in: Code structure.

Strict mode

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

```
'use strict';

...
```

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

Without `"use strict"`, everything still works, but some features behave in the old-fashion, “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".

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:

```
let x = 5;
x = "John";
```

There are 7 data types:

- `number` for both floating-point and integer numbers,
- `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:

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

More in: Variables and Data types.

Interaction

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

`prompt(question, [default])`

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

`confirm(question)`

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

`alert(message)`

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:

```
let userName = prompt("Your name?", "Alice");
let isTeaWanted = confirm("Do you want some tea?");

alert( "Visitor: " + userName ); // Alice
alert( "Tea wanted: " + isTeaWanted ); // true
```

More in: Interaction: alert, prompt, confirm.

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:

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

Assignments

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

Bitwise

Bitwise operators work with integers on bit-level: see the docs when they are needed.

Ternary

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. Logical NOT `!` converts the operand to boolean type and returns the inverse value.

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:

```
alert( 0 == false ); // true
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: Operators, Comparisons, Logical operators.

Loops

- We covered 3 types of loops:

```
// 1
while (condition) {
  ...
}

// 2
do {
  ...
} while (condition);

// 3
for(let i = 0; i < 10; i++) {
  ...
}
```

- 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.

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

The "switch" construct

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

For instance:

```
let age = prompt('Your age?', 18);

switch (age) {
  case 18:
    alert("Won't work"); // the result of prompt is a string, not a number

  case "18":
    alert("This works!");
    break;

  default:
    alert("Any value not equal to one above");
}
```

Details in: The "switch" statement.

Functions

We covered three ways to create a function in JavaScript:

1. Function Declaration: the function in the main code flow

```
function sum(a, b) {
  let result = a + b;

  return result;
}
```

2. Function Expression: the function in the context of an expression

```
let sum = function(a, b) {
  let result = a + b;

  return result;
}
```

Function expressions can have a name, like `sum = function name(a, b)`, but that `name` is only visible inside that function.

3. Arrow functions:

```
// expression at the right side
let sum = (a, b) => a + b;

// or multi-line syntax with { ... }, need return here:
let sum = (a, b) => {
  // ...
  return a + b;
}

// without arguments
let sayHi = () => alert("Hello");

// with a single argument
let double = n => n * 2;
```

- Functions may have local variables: those declared inside its body. 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`.

Function Declaration	Function Expression
visible in the whole code block	created when the execution reaches it
-	can have a name, visible only inside the function

More: see Functions, Function expressions and arrows.

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.

Previous lesson | Next lesson

Share

Tutorial map

Comments

- You're welcome to post additions, questions to the articles and answers to them.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>`, for more than 10 lines – use a sandbox (plnkr, JSBin, codepen…)

- If you can't understand something in the article – please elaborate.