4th February 2020

Code structure

The first thing we'll study is the building blocks of code.

Statements

Statements are syntax constructs and commands that perform actions.

We've already seen a statement, `alert('Hello, world!')`, which shows the message "Hello, world!".

We can have as many statements in our code as we want. Statements can be separated with a semicolon.

For example, here we split "Hello World" into two alerts:

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

Usually, statements are written on separate lines to make the code more readable:

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

Semicolons

A semicolon may be omitted in most cases when a line break exists.

This would also work:

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

Here, JavaScript interprets the line break as an "implicit" semicolon. This is called an automatic semicolon insertion.

In most cases, a newline implies a semicolon. But "in most cases" does not mean "always"!

There are cases when a newline does not mean a semicolon. For example:

```
alert(3 +
1
+ 2);
```

The code outputs `6` because JavaScript does not insert semicolons here. It is intuitively obvious that if the line ends with a plus `"+"`, then it is an "incomplete expression", so the semicolon is not required. And in this case that works as intended.

But there are situations where JavaScript "fails" to assume a semicolon where it is really needed.

Errors which occur in such cases are quite hard to find and fix.

An example of an error

If you're curious to see a concrete example of such an error, check this code out:

```
[1, 2].forEach(alert)
```

No need to think about the meaning of the brackets `[]` and `forEach` yet. We'll study them later. For now, just remember the result of the code: it shows `1` then `2`.

Now, let's add an `alert` before the code and *not* finish it with a semicolon:

```
alert("There will be an error")

[1, 2].forEach(alert)
```

Now if we run the code, only the first `alert` is shown and then we have an error!

But everything is fine again if we add a semicolon after `alert`:

```
alert("All fine now");

[1, 2].forEach(alert)
```

Now we have the "All fine now" message followed by `1` and `2`.

The error in the no-semicolon variant occurs because JavaScript does not assume a semicolon before square brackets `[...]`.

So, because the semicolon is not auto-inserted, the code in the first example is treated as a single statement. Here's how the engine sees it:

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

But it should be two separate statements, not one. Such a merging in this case is just wrong, hence the error. This can happen in other situations.

We recommend putting semicolons between statements even if they are separated by newlines. This rule is widely adopted by the community. Let's note once again – *it is possible* to leave out semicolons most of the time. But it's safer – especially for a beginner – to use them.

Comments

As time goes on, programs become more and more complex. It becomes necessary to add *comments* which describe what the code does and why.

Comments can be put into any place of a script. They don't affect its execution because the engine simply ignores them.

One-line comments start with two forward slash characters `//`.

The rest of the line is a comment. It may occupy a full line of its own or follow a statement.

Like here:

```
// This comment occupies a line of its own
alert('Hello');

alert('World'); // This comment follows the statement
```

Multiline comments start with a forward slash and an asterisk `/*` and end with an asterisk and a forward slash `*/`.

Like this:

```
/* An example with two messages.
This is a multiline comment.
*/
alert('Hello');
alert('World');
```

The content of comments is ignored, so if we put code inside `/* … */`, it won't execute.

Sometimes it can be handy to temporarily disable a part of code:

```
/* Commenting out the code
alert('Hello');
*/
alert('World');
```

Use hotkeys!

In most editors, a line of code can be commented out by pressing the `Ctrl+/` hotkey for a single-line comment and something like `Ctrl+Shift+/` – for multiline comments (select a piece of code and press the hotkey). For Mac, try `Cmd` instead of `Ctrl`.

Nested comments are not supported!

There may not be `/*...*/` inside another `/*...*/`.

Such code will die with an error:

```
/*
  /* nested comment ?!? */
*/
alert( 'World' );
```

Please, don't hesitate to comment your code.

Comments increase the overall code footprint, but that's not a problem at all. There are many tools which minify code before publishing to a production server. They remove comments, so they don't appear in the working scripts. Therefore, comments do not have negative effects on production at all.

Later in the tutorial there will be a chapter Code quality that also explains how to write better comments.

Previous lesson Next lesson

Share Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- 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…)