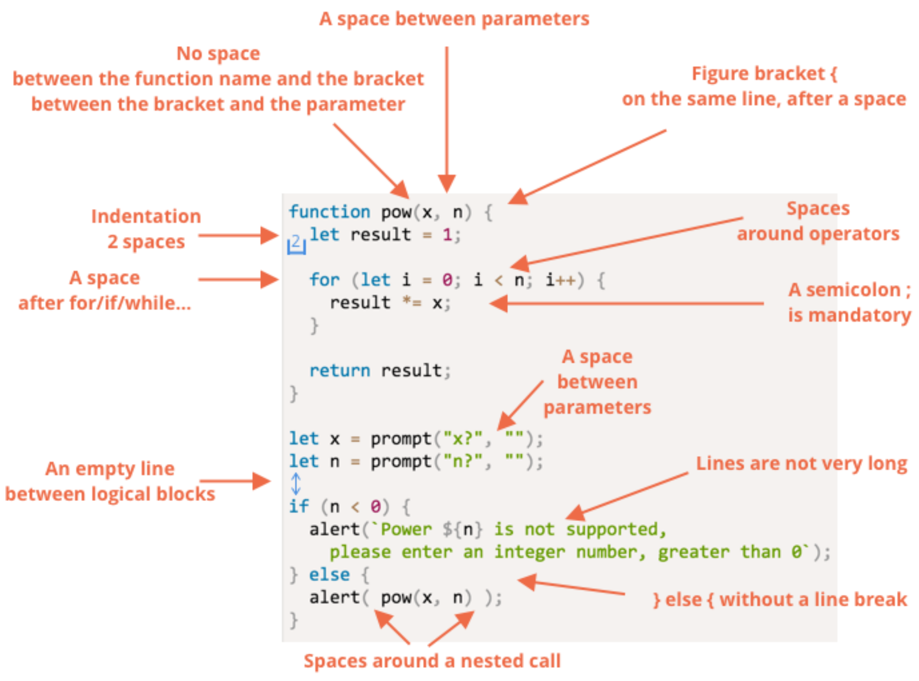
Coding Style

Our code must be as clean and easy to read as possible.

That is actually the art of programming – to take a complex task and code it in a way that is both correct and human-readable.

Syntax

Here is a cheatsheet with some suggested rules (see below for more details):


A space between parameters
No space between the function name and the bracket between the bracket and the parameter
Figure bracket { on the same line, after a space
Indentation 2 spaces
Spaces around operators
A space after for/if/while...
A semicolon ; is mandatory
A space between parameters
An empty line between logical blocks
Lines are not very long
} else { without a line break
Spaces around a nested call

```
function pow(x, n) {
  let result = 1;

  for (let i = 0; i < n; i++) {
    result *= x;
  }

  return result;
}

let x = prompt("x?", "");
let n = prompt("n?", "");

if (n < 0) {
  alert(`Power ${n} is not supported,
    please enter an integer number, greater than 0`);
} else {
  alert( pow(x, n) );
}
```

Now let's discuss the rules and reasons for them in detail.

Irony Detected

Nothing is set in stone here. These are style preferences, not religious dogmas.

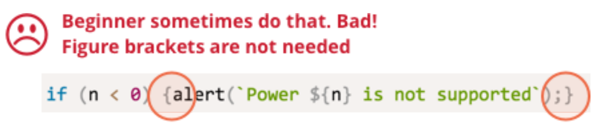
Curly Braces

In most JavaScript projects curly braces are written in "Egyptian" style with the opening brace on the same line as the corresponding keyword – not on a new line. There should also be a space before the opening bracket, like this:

```
if (condition) {
  // do this
  // ...and that
  // ...and that
}
```

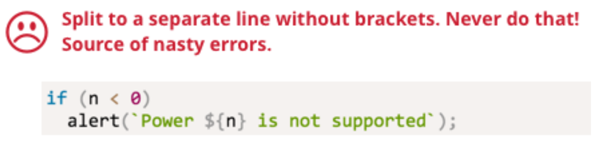
A single-line construct is an important edge case. Should we use brackets at all? If yes, then where?

Here are the annotated variants so you can judge their readability for yourself:



Beginner sometimes do that. Bad!
Figure brackets are not needed

```
if (n < 0) {alert(`Power ${n} is not supported`);}
```



Split to a separate line without brackets. Never do that!
Source of nasty errors.

```
if (n < 0)
  alert(`Power ${n} is not supported`);
```

One line without brackets.
Acceptable if it's short.

```
if (n < 0) alert(`Power ${n} is not supported`);
```

The best variant.

```
if (n < 0) {
  alert(`Power ${n} is not supported`);
}
```

In summary:

- For very short code, one line is acceptable. For example: `if (cond) return null`.
- But a separate line for each statement in brackets is usually easier to read.

Line Length

No one likes to read a long horizontal line of code. It's best practice to split them up and limit the length of your lines.

The maximum line length should be agreed upon at the team-level. It's usually 80 or 120 characters.

Indents

There are two types of indents:

- **Horizontal indents: 2 or 4 spaces.**

  A horizontal indentation is made using either 2 or 4 spaces or the "Tab" symbol. Which one to choose is an old holy war. Spaces are more common nowadays.

  One advantage of spaces over tabs is that spaces allow more flexible configurations of indents than the "Tab" symbol.

  For instance, we can align the arguments with the opening bracket, like this:

```
show(parameters,
     aligned, // 5 spaces padding at the left
     one,
     after,
     another
  ) {
  // ...
}
```

- **Vertical indents: empty lines for splitting code into logical blocks.**

  Even a single function can often be divided into logical blocks. In the example below, the initialization of variables, the main loop and returning the result are split vertically:

```
function pow(x, n) {
  let result = 1;
  //              <--
  for (let i = 0; i < n; i++) {
    result *= x;
  }
  //              <--
  return result;
}
```

  Insert an extra newline where it helps to make the code more readable. There should not be more than nine lines of code without a vertical indentation.

Semicolons

A semicolon should be present after each statement, even if it could possibly be skipped.

There are languages where a semicolon is truly optional and it is rarely used. In JavaScript, though, there are cases where a line break is not interpreted as a semicolon, leaving the code vulnerable to errors.

As you become more mature as a programmer, you may choose a no-semicolon style like StandardJS. Until then, it's best to use semicolons to avoid possible pitfalls.

Nesting Levels

Try to avoid nesting code too many levels deep.

Sometimes it's a good idea to use the "continue" directive in a loop to avoid extra nesting.

For example, instead of adding a nested `if` conditional like this:

```
for (let i = 0; i < 10; i++) {
  if (cond) {
    ... // <- one more nesting level
  }
}
```

We can write:

```
for (let i = 0; i < 10; i++) {
  if (!cond) continue;
  ...  // <- no extra nesting level
}
```

A similar thing can be done with `if/else` and `return`.

For example, two constructs below are identical.

Option 1:

```
function pow(x, n) {
  if (n < 0) {
    alert("Negative 'n' not supported");
  } else {
    let result = 1;

    for (let i = 0; i < n; i++) {
      result *= x;
    }

    return result;
  }
}
```

Option 2:

```
function pow(x, n) {
  if (n < 0) {
    alert("Negative 'n' not supported");
    return;
  }

  let result = 1;

  for (let i = 0; i < n; i++) {
    result *= x;
  }

  return result;
}
```

The second one is more readable because the “edge case” of `n < 0` is handled early on. Once the check is done we can move on to the “main” code flow without the need for additional nesting.

Function Placement

If you are writing several “helper” functions and the code that uses them, there are three ways to organize the functions.

1. Functions declared above the code that uses them:

```
// function declarations
function createElement() {
  ...
}

function setHandler(elem) {
  ...
}

function walkAround() {
  ...
}

// the code which uses them
let elem = createElement();
setHandler(elem);
walkAround();
```

2. Code first, then functions

```
// the code which uses the functions
let elem = createElement();
setHandler(elem);
walkAround();

// --- helper functions ---
function createElement() {
  ...
}

function setHandler(elem) {
  ...
}

function walkAround() {
  ...
}
```

3. Mixed: a function is declared where it's first used.

Most of time, the second variant is preferred.

That's because when reading code, we first want to know *what it does*. If the code goes first, then it provides that information. Then, maybe we won't need to read the functions at all, especially if their names are descriptive of what they actually do.

Style Guides

A style guide contains general rules about "how to write" code, e.g. which quotes to use, how many spaces to indent, where to put line breaks, etc. A lot of minor things.

When all members of a team use the same style guide, the code looks uniform, regardless of which team member wrote it.

Of course, a team can always write their own style guide. Most of the time though, there's no need to. There are many existing tried and true options to choose from, so adopting one of these is usually your best bet.

Some popular choices:

- Google JavaScript Style Guide
- Airbnb JavaScript Style Guide
- Idiomatic.JS
- StandardJS
- (plus many more)

If you're a novice developer, start with the cheatsheet at the beginning of this chapter. Once you've mastered that you can browse other style guides to pick up common principles and decide which one you like best.

Automated Linters

Linters are tools that can automatically check the style of your code and make suggestions for refactoring.

The great thing about them is that style-checking can also find some bugs, like typos in variable or function names. Because of this feature, installing a linter is recommended even if you don't want to stick to one particular "code style".

Here are the most well-known linting tools:

- JSLint – one of the first linters.
- JSHint – more settings than JSLint.
- ESLint – probably the newest one.

All of them can do the job. The author uses ESLint.

Most linters are integrated with many popular editors: just enable the plugin in the editor and configure the style.

For instance, for ESLint you should do the following:

1. Install Node.js.
2. Install ESLint with the command `npm install -g eslint` (npm is a JavaScript package installer).
3. Create a config file named `.eslintrc` in the root of your JavaScript project (in the folder that contains all your files).
4. Install/enable the plugin for your editor that integrates with ESLint. The majority of editors have one.

Here's an example of an `.eslintrc` file:

```
{
  "extends": "eslint:recommended",
  "env": {
    "browser": true,
    "node": true,
    "es6": true
  },
  "rules": {
    "no-console": 0,
  },
  "indent": 2
}
```

Here the directive `"extends"` denotes that the configuration is based on the “eslint:recommended” set of settings. After that, we specify our own.

It is also possible to download style rule sets from the web and extend them instead. See http://eslint.org/docs/user-guide/getting-started for more details about installation.

Also certain IDEs have built-in linting, which is convenient but not as customizable as ESLint.

Summary

All syntax rules described in this chapter (and in the style guides referenced) aim to increase the readability of your code, but all of them are debatable.

When we think about writing “better” code, the questions we should ask are, “What makes the code more readable and easier to understand?” and “What can help us avoid errors?” These are the main things to keep in mind when choosing and debating code styles.

Reading popular style guides will allow you to keep up to date with the latest ideas about code style trends and best practices.

Tasks

Bad style

importance: 4

What’s wrong with the code style below?

```
function pow(x,n)
{
  let result=1;
  for(let i=0;i<n;i++) {result*=x;}
  return result;
}

let x=prompt("x?",''), n=prompt("n?",'')
if (n<=0)
{
  alert(`Power ${n} is not supported, please enter an integer number greater than zero`);
}
else
{
  alert(pow(x,n))
}
```

Fix it.

solution

You could note the following:

```
function pow(x,n)  // <- no space between arguments
{  // <- figure bracket on a separate line
  let result=1;   // <- no spaces before or after =
  for(let i=0;i<n;i++) {result*=x;}   // <- no spaces
  // the contents of { ... } should be on a new line
  return result;
}

let x=prompt("x?",''), n=prompt("n?",'') // <-- technically possible,
// but better make it 2 lines, also there's no spaces and missing ;
if (n<0)  // <- no spaces inside (n < 0), and should be extra line above it
{   // <- figure bracket on a separate line
  // below - long lines can be split into multiple lines for improved readability
  alert(`Power ${n} is not supported, please enter an integer number greater than
}
else // <- could write it on a single line like "} else {"
{
  alert(pow(x,n))  // no spaces and missing ;
}
```

The fixed variant:

```
function pow(x, n) {
  let result = 1;

  for (let i = 0; i < n; i++) {
    result *= x;
  }

  return result;
}

let x = prompt("x?", "");
let n = prompt("n?", "");

if (n < 0) {
  alert(`Power ${n} is not supported,
    please enter an integer number greater than zero`);
} else {
  alert( pow(x, n) );
}
```

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.

25 Comments Javascript.info 1 Login

Recommend 8 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Ninja JS • 9 months ago

Mission accomplished, sergeant.

3 ^ | ⌄ • Reply • Share ›

Kees de Vreugd ➦ Ninja JS • 7 months ago

Yeah!
This NEVER gets old!

^ | ⌄ • Reply • Share ›

Pretheesh G Presannan • 7 months ago

lol :) "not religious dogmas, just coding rules"

3 ^ | ⌄ • Reply • Share ›

BociTarka • 6 months ago

I never understood why some people like to use spaces as opposed to tabs for indentation. Developers (especially for anything web) seem to be obsessed with small file sizes. Compress everything, fast transmission, page load, etc. It's more of a "mentality" approach. If you are obsessed with small file sizes and want to compress and merge files for faster transfer and less network requests, then why are you using spaces!? I don't see the common sense or reasoning behind it. And don't get me wrong, it's not about the fact that at the end of the day, minifiers and mergers will create a small and "space efficient" file. As I was saying, this has nothing to do with that, and everything with the "mentality" behind it.

Think about it. One character in a file is one byte (assuming you don't use 2-byte characters). So if you use 4 spaces for indent, that's 4 bytes. Count the number of indents you have in the file, and your file is now bloated by that number of bytes.

A tab character is a single byte. So for every 4 bytes (for 4 spaces) you only use 1 byte (a tab character). How is this not more efficient?

I'm sure some will disagree because of how "cheap" hard drive space is. But this has nothing to do with hard drive space prices. It has to do with the "mentality" and common sense. If you advocate for faster file transfer, and zipping and merging your script and css files, but on the flip side you are using 4 bytes for indent...well, I just don't understand the "mentality" and "reasoning" of the "logic".

It's like being an advocate for health and exercise, but instead of walking or biking you sit in your car; instead of raking the leaves you use a blower, etc.

1 ^ | ⌄ • Reply • Share ›

Yilmaz Durmaz ➦ BociTarka • 6 months ago

easy to answer : you can't do what is done in "Indents -> Horizontal indents: 2 or 4 spaces." by using tabs. you need extra spaces there. and then if you have to use spaces, why not to use them all the time?

and with minifiers, why bother to try to use tab for the size of the file? we are not in ages anymore where total capacity of a computer was to be few kilobytes.

and also tab's spacing does not behave well with character encodings at all. it is also not visible so while fixing a tabbed code, it really becomes annoying sometime after.

oh, and the author has stated at the beginning styling is "not religious dogmas", and yet here you are being dogmatic with a page load of things just to make yourself seem clever explaining about the 1byte versus 4byte difference. I suggest you to take a break on your job or see a doctor for your own mental health.

^ | ⌄ • Reply • Share ›

Timothy Lee Banks • 2 years ago

eslint + w0rp/ale + vim + airbnb style guide

1 ^ | ⌄ • Reply • Share ›

webconnector • 2 months ago

nice one here

^ | ⌄ • Reply • Share ›

Divi • 4 months ago

This is the great tutorial I ever seen.
This is exactly what I want to know as a beginner.
Thank you so much.

• Reply • Share ›

Mohammed Boudad • 6 months ago

To gain one more line, you can replace this:
function pow(x, n) {
if (n < 0) {
alert("Negative 'n' not supported");
return;
}

let result = 1;

for (let i = 0; i < n; i++) {
result *= x;
}

return result;
}

with this:

function pow(x, n) {

see more

• Reply • Share ›

Victor Onlite • 10 months ago

Thanks a lot! I've learnt so much new!

• Reply • Share ›

Igettäjä • 10 months ago

The `.eslintrc` example configuration is wrong, the indent rule should be within the rules block.

• Reply • Share ›

Mohamed Cherif • a year ago

I think we can avoid writing the else condition here:
} else {
alert(pow(x, n));
}
it's better to keep it like this:
alert(pow(x, n));

• Reply • Share ›

jewel rana • a year ago

cool :)

• Reply • Share ›

Prokhor Vasilyev • a year ago

cool

• Reply • Share ›

X Axis • a year ago

The forth rule in JSstandard says not to use semicolons

• Reply • Share ›

Paranolverizer Pzr ➦ X Axis • a year ago

I noticed that one too. Seems fishy to me considering semicolon is pretty damn important.

• Reply • Share ›

Pavel • a year ago

I think mentioning prettier would be a good idea.

• Reply • Share ›

Sagnik Chakraborti ➦ Pavel • a year ago

Sometimes, prettier does something that we don't need. For instance, we want to write something on a separate line, which prettier pushes to a single line. Though it's more compact, but less readable. Just my opinion.

Reply • Share ›

Juraj Pecháč • 2 years ago

Correct version:

```
{
"extends": "eslint:recommended",
"env": {
"browser": true,
"node": true,
"es6": true
},
"rules": {
"no-console":0,
"indent": 2
}
}
```

Reply • Share ›

bjatta • 2 years ago

"JSHint – more settings than JSHint."
Is it right?

Reply • Share ›

Voros Rukmini Xanathor ➦ bjatta • 2 years ago

yes, me too

Reply • Share ›

Yofriadi Yahya ➦ bjatta • 2 years ago

i made a pull request

Reply • Share ›

Voros Rukmini Xanathor ➦ Yofriadi Yahya • 2 years ago

i think you're banned

Reply • Share ›

Show more replies

Gs Wang • 2 years ago

According to Functions below the code, the last solution could be improved.

Reply • Share ›