**Translate the following words and phrases:**

|  |  |
| --- | --- |
| Однострочный комментарий |  |
| Многострочный комментарий |  |
|  | Starting with |
| Очевидный | Evident, obvious |
| Понять неправильно |  |
| Пояснительный комментарий |  |
| Переписать |  |
| Реорганизовать в функцию |  |
|  | Functions themselves |
| Нечего комментировать |  |
| избегать |  |
| самоописывающая |  |
| Взаимодействовать, воздействовать друг на друга |  |
| Решать задачу |  |
| Более очевидный и правильный вариант |  |
| возвращаться | Revert to |
| Поддерживать код | maintain |
| Общая архитектура | overall |
| Высокоуровневый вид |  |
| Использование функции | usage |

**Comments**

Comments can be single-line: starting with // and multiline: /\* ... \*/.

We normally use them to describe how and why the code works.

From the first sight, commenting might be obvious, but novices in programming usually get it wrong.

**Bad comments**

Novices tend to use comments to explain “what is going on in the code”. Like this:

// This code will do this thing (...) and that thing (...)

// ...and who knows what else...

very;

complex;

code;

But in good code the amount of such “explanatory” comments should be minimal. Seriously, code should be easy to understand without them.

There’s a great rule about that: “if the code is so unclear that it requires a comment, then maybe it should be rewritten instead”.

And if we have a long “code sheet” like this:

// here we add whiskey

for(let i = 0; i < 10; i++) {

let drop = getWhiskey();

smell(drop);

add(drop, glass);

}

// here we add juice

for(let t = 0; t < 3; t++) {

let tomato = getTomato();

examine(tomato);

let juice = press(tomato);

add(juice, glass);

}

// ...

Then it might be a better variant to refactor it into functions like:

addWhiskey(glass);

addJuice(glass);

function addWhiskey(container) {

for(let i = 0; i < 10; i++) {

let drop = getWhiskey();

//...

}

}

function addJuice(container) {

for(let t = 0; t < 3; t++) {

let tomato = getTomato();

//...

}

}

Once again, functions themselves tell what’s going on. There’s nothing to comment. And also the code structure is better when split. It’s clear what every function does, what it takes and what it returns.

In reality, we can’t totally avoid “explanatory” comments. There are complex algorithms. And there are smart “tweaks” for purposes of optimization. But generally we should try to keep the code simple and self-descriptive.

**Good comments**

So, explanatory comments are usually bad. Which comments are good?

**Describe the architecture**

Provide a high-level overview of components, how they interact, what’s the control flow in various situations… In short – the bird’s eye view of the code.

**Why is the task solved this way?**

What’s written is important. But what’s *not* written may be even more important to understand what’s going on. Why is the task solved exactly this way? The code gives no answer.

If there are many ways to solve the task, why this one? Especially when it’s not the most obvious one.

Without such comments the following situation is possible:

1. You (or your colleague) open the code written some time ago, and see that it’s “suboptimal”.
2. You think: “How stupid I was then, and how much smarter I’m now”, and rewrite using the “more obvious and correct” variant.
3. …The urge to rewrite was good. But in the process you see that the “more obvious” solution is actually lacking. You even dimly remember why, because you already tried it long ago. You revert to the correct variant, but the time was wasted.

Comments that explain the solution are very important. They help to continue development the right way.

[**Summary**](https://javascript.info/comments#summary)

An important sign of a good developer is comments: their presence and even their absence.

Good comments allow us to maintain the code well, come back to it after a delay and use it more effectively.

**Comment this:**

* Overall architecture, high-level view.
* Function usage.
* Important solutions, especially when not immediately obvious.

**Avoid comments:**

* That tell “how code works” and “what it does”.
* Put them only if it’s impossible to make the code so simple and self-descriptive that it doesn’t require those.

**I. Translate into English:**

1. Твой комментарий не должен описывать, как работает код и что он делает
2. Попытайся сделать свой код простым и не требующим дополнительного описания self-descriptive, таким образом тебе не потребуются комментарии.
3. Отсутствие комментариев the absence of comments – это тоже хорошая практика. Особенно если эти комментарии очевидны или носят пояснительный характер.
4. Если вы корректно называете свои функции и правильно составляете тело функций, то в вашем коде нечего комментировать.
5. Если у тебя получился слишком длинный код, то лучше зарефакторить его в функции.
6. Сами по себе функции сообщают, что происходит в твоем коде.
7. Хороший комментарий объясняет почему задача has been solved /is solved решена именно этим способом
8. Хорошие комментарии помогают нам поддерживать код.
9. Тебе следует комментировать общую архитектуру, высокоуровневый вид и использование функций.

**II. Enumerate things that should/should not be commented.**

**III. Active vocabulary:**

|  |  |
| --- | --- |
| Использовать однострочный комментарий |  |
| Комментировать структуру кода |  |
| Функция принимает (возвращает) данные |  |
| Сложные алгоритмы |  |
| Решить задачу |  |
| Требовать комментарий (требование) |  |
| Наличие / отсутствие |  |
| Решения | solutions |
| Переписать |  |
| Описать как работает код |  |