Comments

As we know from the chapter Code structure, 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".

Recipe: factor out functions

Sometimes it's beneficial to replace a code piece with a function, like here:

```
function showPrimes(n) {
  nextPrime:
  for (let i = 2; i < n; i++) {

    // check if i is a prime number
    for (let j = 2; j < i; j++) {
      if (i % j == 0) continue nextPrime;
    }

    alert(i);
  }
}
```

The better variant, with a factored out function `isPrime` :

```
function showPrimes(n) {

  for (let i = 2; i < n; i++) {
    if (!isPrime(i)) continue;

    alert(i);
  }
}

function isPrime(n) {
  for (let i = 2; i < n; i++) {
    if (n % i == 0) return false;
  }

  return true;
}
```

Now we can understand the code easily. The function itself becomes the comment. Such code is called *self-descriptive*.

Recipe: create functions

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. There's a special diagram language UML for high-level architecture diagrams. Definitely worth studying.

Document a function usage

There's a special syntax JSDoc to document a function: usage, parameters, returned value.

For instance:

```
/**
 * Returns x raised to the n-th power.
 *
 * @param {number} x The number to raise.
 * @param {number} n The power, must be a natural number.
 * @return {number} x raised to the n-th power.
 */
function pow(x, n) {
  ...
}
```

Such comments allow us to understand the purpose of the function and use it the right way without looking in its code.

By the way, many editors like WebStorm can understand them as well and use them to provide autocomplete and some automatic code-checking.

Also, there are tools like JSDoc 3 that can generate HTML-documentation from the comments. You can read more information about JSDoc at http://usejsdoc.org/.

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.

Any subtle features of the code? Where they are used?

If the code has anything subtle and counter-intuitive, it's definitely worth commenting.

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.

Comments are also used for auto-documenting tools like JSDoc3: they read them and generate HTML-docs (or docs in another format).

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.