Clean Code

Mohamed Emary

August 31, 2023

1 Clean Code From Odin Lesson Page

Learning to write clean code is a process of constant improvement. This lesson is meant to serve as a primer, a starting point for that journey.

Don't try to write perfectly clean code, this will only lead to frustration. Instead focus on gradual improvement, not perfection. Writing "spaghetti" is inevitable, everyone does it sometimes. Just keep these ideas in mind and with time and patience, your code will start to get cleaner.

Single characters can be used as variable names in the context of a loop or a callback function, but avoid them elsewhere.

What makes a good variable or function name.

- A good name is descriptive
- Use a consistent vocabulary. (example below)

```
 // Good
function getPlayerScore();
function getPlayerName();
function getPlayerTag();
```

```
// Bad
function getUserScore();
function fetchPlayerName();
function retrievePlayer1Tag();
```

In the bad example, three different names are used to refer to the player and the actions taken. Additionally, three different verbs are used to describe these actions. The good example maintains **consistency in both variable naming and the verbs used**.

- **Variables** should always begin with a **noun** or an **adjective** (that is, a noun phrase) and ***functions*** with a ***verb***. (Two examples below)

Example 1:

```
// Good
const numberOfThings = 10;
const myName = "Thor";
const selected = true;
```

```
// Bad (these start with verbs, could be
↪  confused for functions)
const getCount = 10;
const isSelected = true;
```

Example 2:

```
// Good
function getCount() {
  return numberOfThings;
}
```

```
// Bad (it's a noun)
function myName() {
  return "Thor";
}
```

- Use searchable and immediately understandable names

Sometimes, it can be tempting to use an undeclared variable. Let's take another look at an example:

```
setTimeout(stopTimer, 3600000);
```

The problem is obvious. What does the undeclared variable 3600000 mean and how long is this timeout going to count down before executing stopTimer? Even if you know that **JavaScript understands time in milliseconds**, a calculation is needed.

Now, let's make this code more meaningful by introducing a descriptive variable:

```javascript
const MILLISECONDS_PER_HOUR = 60 * 60 * 1000; // 3,600,000;

setTimeout(stopTimer, MILLISECONDS_PER_HOUR);
```

Letters in `MILLISECONDS_PER_HOUR` are all uppercase because this is a convention to be used when the programmer is absolutely sure that the variable is truly a constant that will never change.

- Choose a way to indent (Either tabs or spaces) and stick to it.
- Keep a line length that you generally don't exceed

Generally your code will be easier to read if you manually break lines that are longer than about 80 characters. Many code editors have a line in the display to show when you have crossed this threshold (Like that one vertical line I used to disable in jetbrains IDEs).

Code formatters like prettier can take care of the line length issue.

- Semicolons are mostly optional in JavaScript because the JS compiler **will automatically insert them** if they are omitted. This functionality CAN break in certain situations leading to bugs in your code so it is better to get used to adding semi-colons. *Again: consistency is the main thing.*
- Comments Part has some great advices that I think it's better to **READ FROM ODIN WEBSITE**.