

Tips to Master the Art of Clean Code



What is Clean Code exactly?

In simple terms Clean Code refers to a codebase that is easy to read and understand.

Clean Code is not about writing code that works, it is about writing code that is easy to read and maintain in the long term.

The cost of maintaining a Dirty Code base drastically **increases** over time, whereas in the case of Clean Code, it remains fairly **constant**.

1. Intuitive Variable Names

Quite evidently nobody does all the computation mentally to check what a variable stores.



```
const x = n.filter(e => e > 0);
```



```
const positiveElements = numbers.filter(num => num > 0);
```

As you can see the second example is far easier to understand than the first one.

2. Self-Explanatory functions

There are two things to keep in mind while writing functions or methods:

- Follow the Single Responsibility Principle (SRP)
- Function names should be actions words (verbs)

```
const validate = (data) => {  
}  
  
const sendData= (data) => {  
}  
  
const submit = (data) => {  
  if (!validate(data)){  
    return;  
  }  
  sendData(data)  
}
```


3. Group Similar Functions Together

Here we find that the IOHelper class only groups together functions for io operations, without having any utility of its own.


```
class IOHelper:
    @staticmethod
    def read_data(file_name: string) -> None:
        pass

    @staticmethod
    def write_data(file_name: string, data: Any) -> None:
        pass
```

4. Minimize The Number Of Comments

The code you write should be self-explanatory, anyone viewing your code should not have to rely on comments to understand what it does.

There are a few rare cases where you might need comments in case there is some unintuitive code without a workaround.



```
// using setTimeout to avoid race-condition  
setTimeout(fn, 0)
```

5. Code Formatting

The codebase should always follow a strict set of formatting rules. It is also a good idea to use a formatter like black or prettier to automate the formatting.

The variables and function naming convention should also be specified beforehand (isValid or hasValidity, etc.) and language-specific cases should be kept in mind (eg: snake case in python and camel case in JS/TS).

```
snake_case_variable = "value"  
camelCaseVariable = "value"
```