**DOCUMENTATION AND COMMENTING STANDARDS CHECK**

**VERIFACTS SERVICES PRIVATE LIMITED**

# Introduction

According to a survey by Stack Overflow, 59.7% of developers believe that code comments are "somewhat important" or "very important" for software development.

Meaningful comments can not only benefit engineers but also help fight technical debt, improve code quality, and create a more scalable codebase. By providing better insight into the code, comments can also facilitate code review easier and make collaboration among team members smoother and more efficient

The most prominent and immediate benefit of written code comments is that they make code more accessible for others to understand. But if we use the best practices of the code comments, we unlock even more impactful, long-lasting benefits like

* Smooth code maintenance process
* Improved bug detection for faster issue resolution
* Accelerating development cycles with better engineering velocity
* Managing and reducing technical depts

On the other hand, not following the best practices can make the code harder to read, understand, and maintain, leading to decreased productivity, increased errors, and longer development cycles. In this article, we will discuss about the:

1. Different Types of comments
2. When and where to use comments
3. Best practices for comments
4. Common mistakes to avoid

# Different types of code comments

1. **Line comments**: These are comments that are added to a single line of code and typically begin with two forward slashes (//). Single-line comments help add short explanations or clarifications about specific lines of code.
2. **Block comments**: Also non as Muti-line comments. These comments span multiple lines and are enclosed within a set of opening and closing comment tags. In some languages, multi-line comments begin with /\* and end with \*/. Multi-line comments help provide more detailed explanations of complex or lengthy code sections.
3. **Docstrings**: These are comments that are written in a specific format and can be used to generate documentation for the codebase. Docstrings are typically used in Python and other languages that support the Sphinx documentation tool.
4. **Function/method comments**: These are comments that are added directly above a function or method declaration and provide information about the function's purpose, parameters, and return values.
5. **Class comments**: These are comments that are added directly above a class declaration and provide information about the class's purpose, properties, and methods.
6. **TODO comments**: These are comments that are used to mark parts of the codebase that require additional work or attention in the future. TODO comments typically begin with the word TODO and are followed by a brief description of what needs to be done.

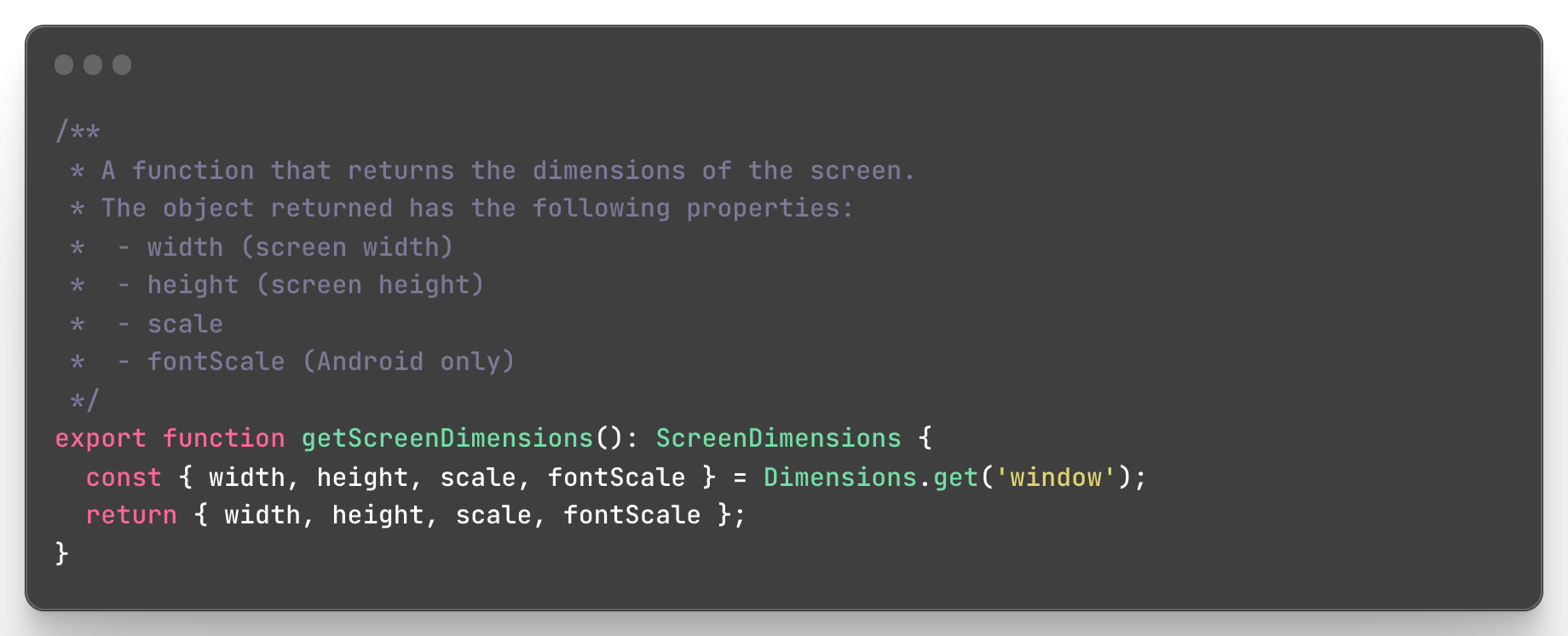
# When and where to use comments

Code comments are an important tool for improving the quality and readability of a codebase. However, it's important to use them effectively and judiciously.

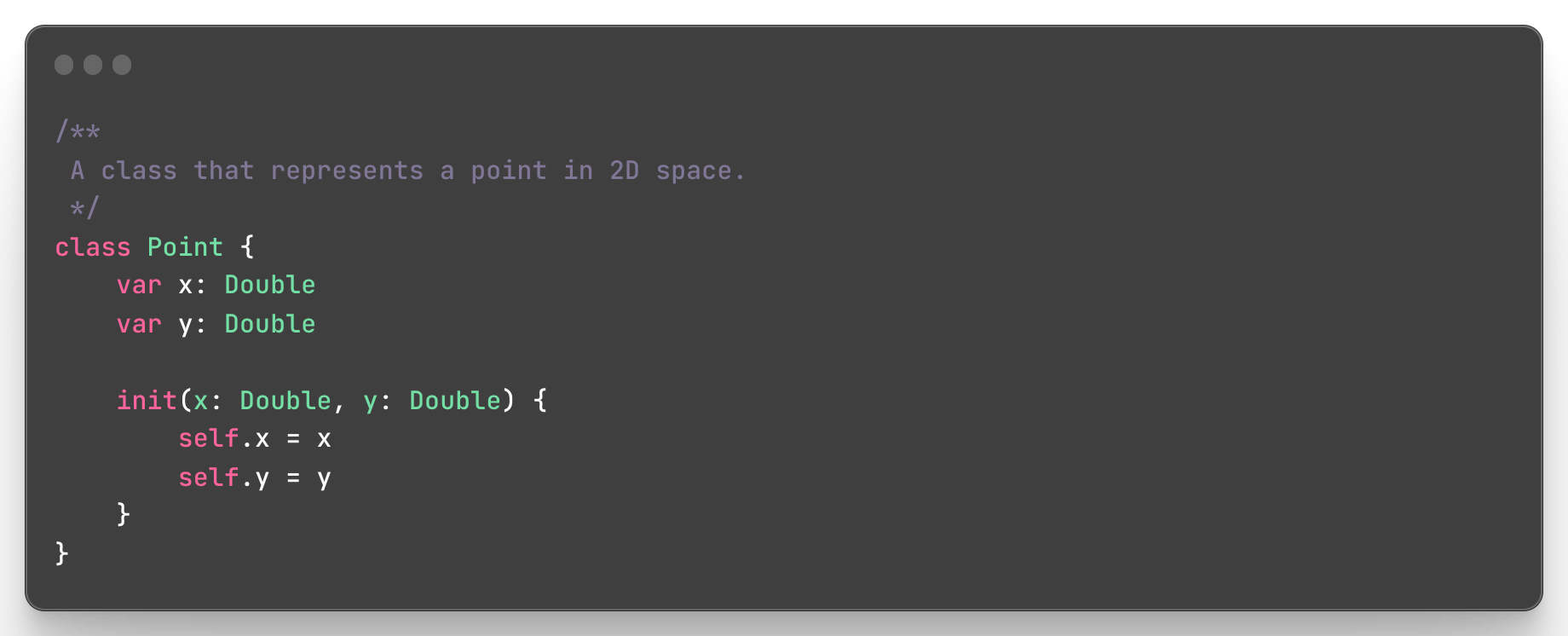
1. **Explaining complex or confusing code**: If a particular section of code is difficult to understand, consider adding comments to provide additional context and explanation. This will make it easier for others to understand the code and make changes if necessary.
2. **Provide code overview**: If you're working on a large project or a complex algorithm, consider adding comments that provide an overview of the code's purpose and how it fits into the project as a whole. This will help other developers understand the big picture and make it easier to navigate the codebase.
3. **Document changes**: If you make changes to the codebase, consider adding comments to document those changes. This will make it easier for other developers to understand how the code has evolved over time and why those changes were made.
4. **Explain edge cases**: If the code has edge cases or unusual behavior, consider adding comments to explain those cases. This will help other developers understand why the code behaves the way it does and how to work with those edge cases.
5. **Sparingly**: While comments are an important tool, it's important not to overuse them. Over-commenting can make the code harder to read and maintain, as developers have to sift through a lot of unnecessary information. Use comments judiciously and focus on providing the most important information.

# Best practices for writing meaningful code comments

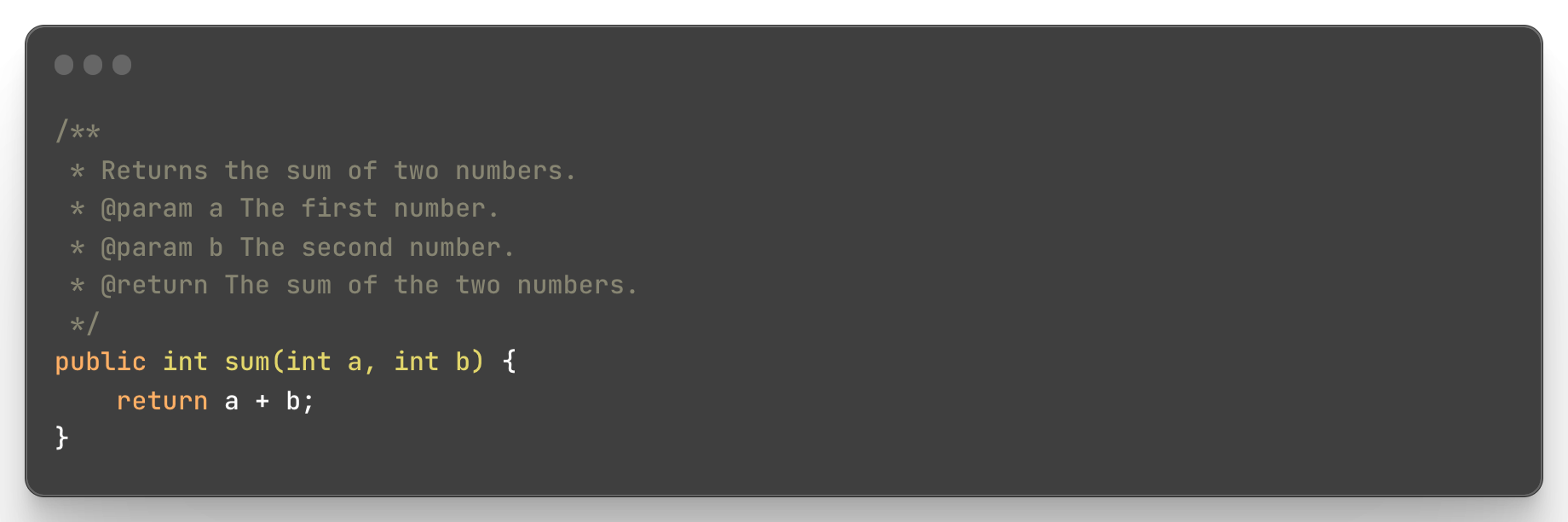
* **Keep comments concise and focused**: Comments should be brief and to the point. They should explain the purpose of the code without getting bogged down in unnecessary details. For example, here's a comment from React Native code that explains the purpose of a function:

Keep comments concise and focused

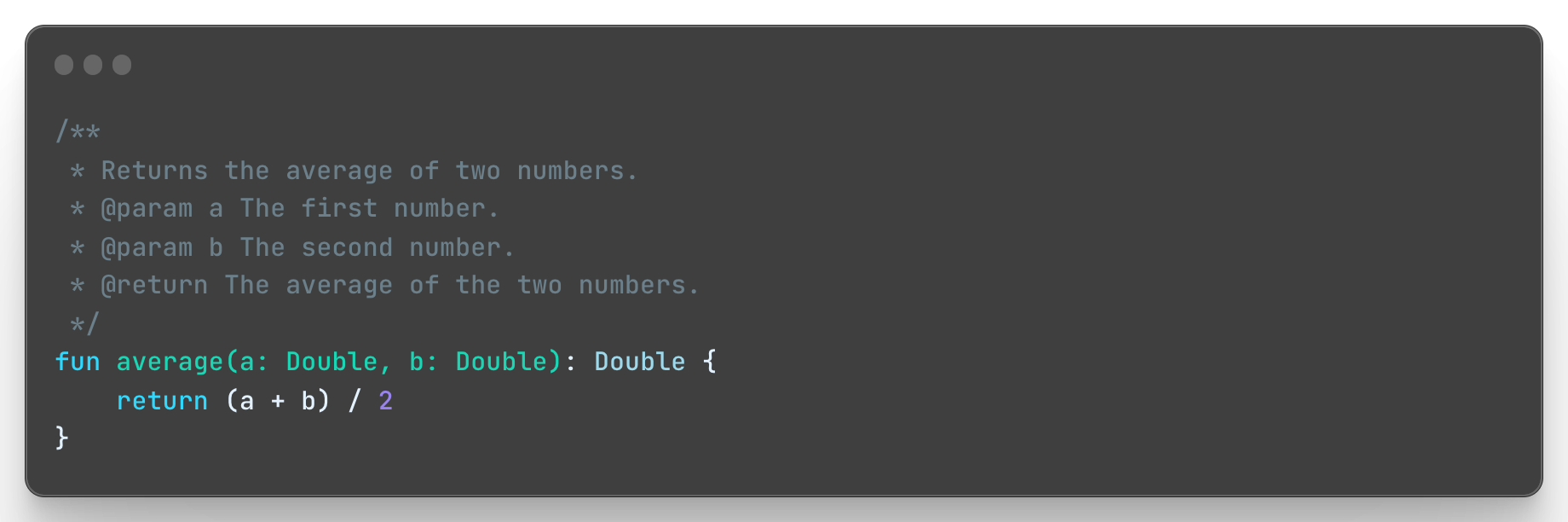
* **Clear and descriptive language**: Comments should use plain language that is easy to understand. They should describe what the code does, not how it does it. For example, here's a comment from Swift code that explains the purpose of a class:

Clear and descriptive language

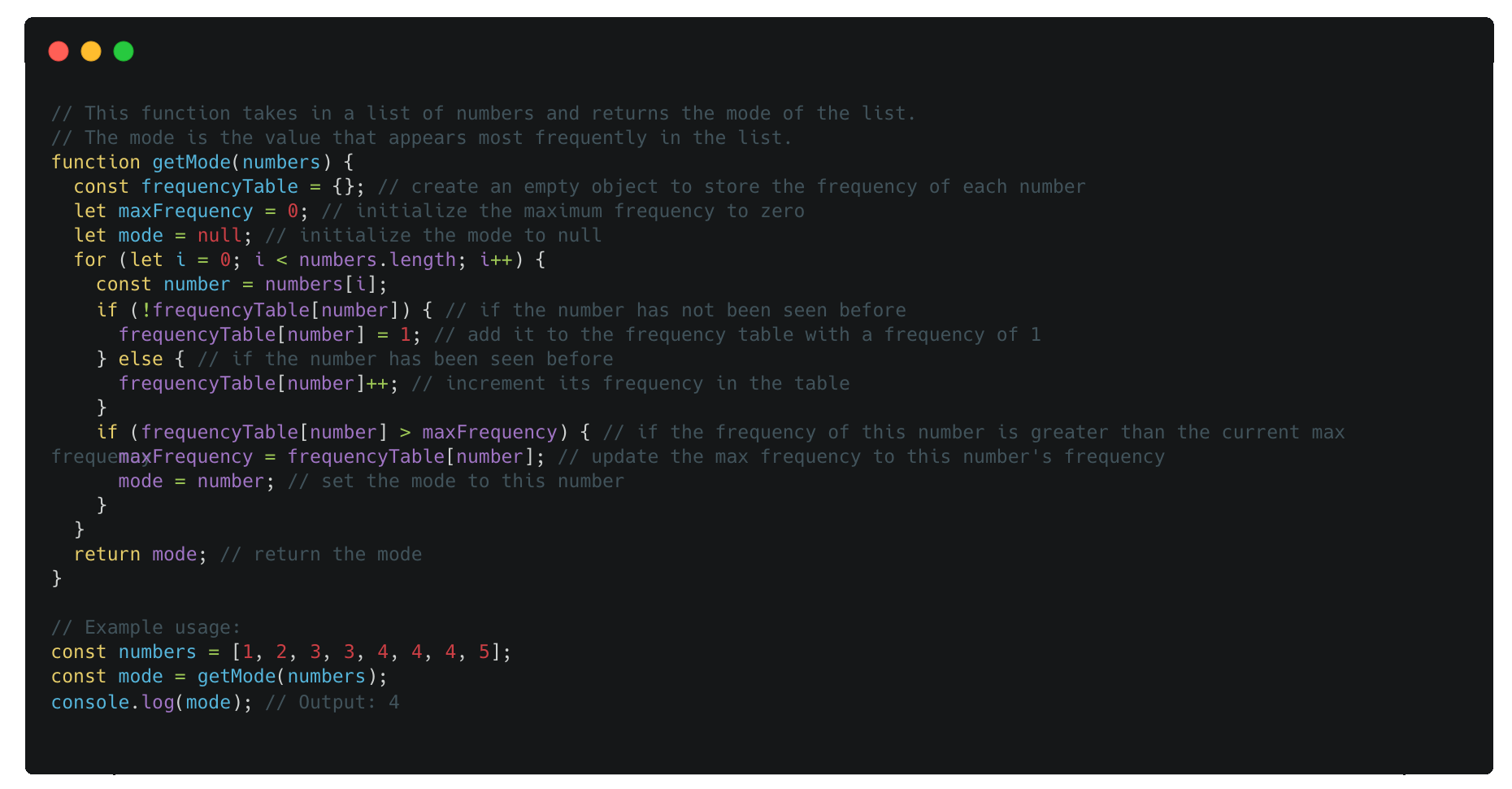
* **Follow a consistent commenting style**: Comments should follow a consistent style throughout the codebase. This makes it easier for developers to read and understand the code. For example, here's a comment from Java code that explains the purpose of a method:

Follow a consistent commenting style

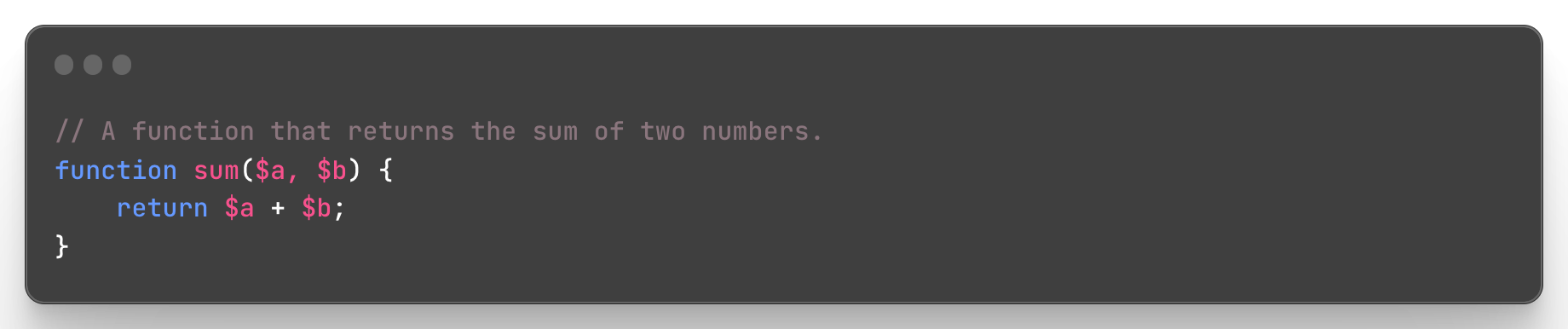
* Document your code as you write **it**: Comments should be written as you write the code, not as an afterthought. This makes it easier to remember why the code was written in a certain way. For example, here's a comment from Kotlin code that explains the purpose of a function:

Document your code as you write it

* **Use comments to explain complex or unusual code**: Comments should be used to explain code that is difficult to understand or that deviates from standard practices. For example, here's a comment from Javascript code that explains the purpose of a function:

Comments to explain complex code

* **Avoid redundant or unnecessary comments**: Comments should not repeat what the code is already saying. For example, here's a comment from PHP code that explains the purpose of a function:

Avoid redundant or unnecessary comments

# Common mistakes to avoid

* Writing comments that are too long or overly complex, as this can make them harder to read and understand.
* Failing to update comments when code changes, which can lead to outdated or inaccurate information.
* Writing comments that are redundant or duplicative, which can make the code harder to read and understand.
* Failing to provide enough context or explanation in comments, which can make the code harder to understand for other developers.
* Writing comments that are vague or unclear, which can lead to confusion and errors in the code.
* Failing to follow established commenting conventions or standards, which can make the code harder to read and maintain.
* Writing comments that are overly negative or critical, which can create a negative team environment and hurt morale.
* Including personal or irrelevant information in comments, which can make the code harder to read and understand.
* Writing comments that are too specific or technical, which can make them harder to understand for non-technical stakeholders.
* Failing to prioritize code readability and maintainability in comments, which can lead to longer development cycles and higher costs in the long run.

# Conclusion

We have discussed about the risks of poor or ineffective code comments, such as confusion, errors, and decreased productivity. On the other hand, well-written code comments can improve the readability and maintainability of the codebase, leading to faster development cycles, reduce technical dept and higher quality code.