Clean Code and Steps to Generate it Using GitHub Copilot

What is Clean Code?

Clean Code is:
- Easy to read and understand
- Simple and elegant
- Focused, maintainable, and testable
- Written with clear naming, small functions, and minimal side effects

Robert C. Martin (Uncle Bob) defines clean code as:
"Looks like it was written by someone who cares."

Key Principles of Clean Code

• Meaningful Names: Use descriptive, unambiguous names for variables, functions, classes

• Small Functions: Functions should do one thing and be short

• Single Responsibility: Each function or class should have one purpose

• Avoid Magic Numbers: Use named constants instead of hard-coded values

• Use Comments Sparingly: Code should be self-explanatory; comment only when necessary

• Consistent Formatting: Use proper indentation and spacing

• Readable Error Handling: Don’t ignore exceptions; handle them clearly

• Write Tests: Code should be testable and accompanied by meaningful tests

Steps to Generate Clean Code Using GitHub Copilot

Think of Copilot as a junior pair programmer — you provide guidance, and it assists.

• Write Clear, Descriptive Prompts: Use clear comments or starting lines of functions to guide Copilot.

• Use Meaningful Function and Variable Names: Copilot mimics your naming; choose good ones upfront.

• Enforce Clean Formatting: Use formatters like Prettier or ESLint to guide Copilot's output.

• Refactor Copilot Output: Don't accept suggestions blindly; improve them.

• Add Defensive Programming (Validation & Error Handling): Prompt Copilot to include validation and handle errors explicitly.

• Write Unit Tests for Each Function: Copilot can generate unit tests if you start writing them yourself.

• Use Code Review and Linters: Use automated and manual review to catch violations of clean code.

Pro Tips to Use Copilot for Clean Code

• Use inline comments as prompts: Guides Copilot to generate clean, purposeful code

• Keep your codebase clean: Copilot learns from your code style and conventions

• Review every suggestion: Don't trust AI without validation

• Combine with TDD: Copilot can generate implementations from your tests

Example: Copilot + Clean Code Practice

Prompt:

```
// Convert Celsius to Fahrenheit
function convertCelsiusToFahrenheit(celsius: number): number {
```

Copilot Output:

```
  return (celsius * 9/5) + 32;
}
```

Clean: Clear name, no magic numbers, one responsibility

Summary

1. Write clear prompts and comments

2. Choose meaningful names

3. Use formatting tools

4. Refactor suggestions

5. Add validations

6. Generate and maintain tests

7. Run reviews and linting tools