Turiba University

Bachelor's Degree in computer system

Bismaya Aneesh

Group : CSA1D1

History of Programming

Institution: Turiba University

Introduction

In the world of programming, writing code that works is only the beginning. Equally important is writing code that is **friendly** — meaning it is easy to read, understand, maintain, and reuse. "Friendly coding" refers to adopting practices that help not just the original coder, but others who may read or work with the code later (including your future self!).

Why is friendly coding important?

- It improves collaboration: when you work with others, friendly code makes teamwork smoother.
- It saves time: if code is clear, maintenance and updates are faster.
- It increases reliability: readable and well-structured code tends to have fewer bugs.

  In this report, we will explore what friendly coding means, the key principles, tools and methods for achieving it, examples, benefits and challenges, and conclude with best practices you can apply now.

What Does Friendly Coding Mean?

Friendly coding involves making code understandable, modular, and maintainable. Some of the key features include:

- **Readability**: Code should be written so that someone else can follow the logic without struggle. This includes meaningful names for variables and functions, consistent formatting, comments where needed. DataCamp+2FreeCodeCamp+2
- **Simplicity**: Avoid unnecessary complexity. Use clear logic instead of overly clever tricks. DEV Community+1
- **Reusability & modularity**: Write code in small pieces (functions/modules) so parts can be reused and updated easily. FreeCodeCamp+1
- **Maintainability**: The code should be easy to maintain, update and fix. Others (or you in the future) should not be stuck trying to figure out what you meant. mitcommlab.mit.edu

- **Consistent style & documentation**: Using coding standards, consistent naming conventions, comments and documentation help make code "friendly". BrowserStack+1

When code is friendly, it becomes easier to share, to teach, and to build upon. For a student like you, adopting friendly coding early means less frustration and more enjoyment in learning programming.

Key Principles of Friendly Coding

Let's look more deeply at some of the core principles you should follow.

1. Meaningful Names

Give variables, functions, classes names that describe their purpose. For example, instead of naming a variable n or x, use studentCount or totalScore. This makes the code self-explanatory. DataCamp+1

2. Clear Formatting & Structure

Indentation, spacing, grouping of code blocks, using blank lines to separate logical sections—all these help readability. Consistent style is key. FreeCodeCamp+1

3. Keep It Simple — The KISS Principle

KISS stands for Keep It Simple, Stupid (a humorous phrase) — meaning don't make logic more complex than needed. Simple logic is more reliable and easier to maintain. DEV Community

4. Avoid Repetition — DRY Principle

DRY stands for Don't Repeat Yourself. If you find yourself copying and pasting the same code, it's a sign you should create a function or module instead. This helps later when you need to update logic in just one place. DEV Community+1

5. Modularity & Single Responsibility

Break down your code into modules/functions where each part does one job. This way, you can test and maintain parts separately. FreeCodeCamp

6. Documentation & Comments

A few well-placed comments can help explain "why" something is done (not just "what"). But avoid over-commenting obvious code — the code itself should express what it does where possible. cipherschools.com+1

7. Coding Standards & Convention

Following a style guide (indentation, naming, file structure) makes it easier for you and others to understand the code. Even across different programming languages, many of the ideas are similar. BrowserStack+1

Tools & Methods to Achieve Friendly Coding

Here are some practical methods and tools you can use as you code.

- **Code Editors / IDEs**: Many editors support auto-formatting, indentation helpers, linters (which check for style issues). Using them helps you stay consistent.
- **Linting Tools & Static Analysis**: These tools automatically check your source code for style violations, possible bugs, etc. For example, one tool for C++ is cpplint. Wikipedia
- **Version Control & Collaboration Tools**: Using systems like Git (not required for beginners but good to know) helps track changes, collaborate with others and maintain code history.
- **Code Reviews / Pair-Programming**: Working with someone else or reviewing code helps spot unclear logic and improve readability.
- **Refactoring Regularly**: Improving the structure of existing code (without changing its behavior) helps keep the code friendly over time.

- **Templates & Coding Standards Documentation**: Having a simple document listing naming conventions, file organisation, comment style helps your future code be consistent.
- **Testing**: Writing tests helps ensure your code works and helps others understand how to use the modules/functions you created.

Using these methods helps translate the friendly-coding principles into your everyday coding actions.

Examples & Case Studies

Let's look at some simple examples to illustrate friendly vs unfriendly coding.

Unfriendly Code Example:

```
def a(x,y):
  return x+y
```

This function name a and variable names x,y are not descriptive. A reader has to guess what a means.

Friendly Code Example:

```
def compute_total_price(base_price, tax_rate):
    total_price = base_price + (base_price * tax_rate)
    return total_price
```

Here the function name and parameter names tell what the function does. It's easier to understand and maintain.

In research on readability, a study titled To What Extent Cognitive-Driven Development Improves Code Readability? found that code designed with smaller units (i.e., more modular) was perceived by many developers as more readable. arXiv

Also, according to the article on coding best practices, using consistent naming, formatting and avoiding deep nesting helps readability. BrowserStack+1

Case for Beginners:
A beginner programmer might write many lines in one function doing everything at once—input, processing, output. Friendly coding suggests breaking this into smaller functions: one for input, one for processing, one for output/display. This makes each piece easier to understand and test.

Benefits & Challenges of Friendly Coding

Benefits

- **Better collaboration**: When code is clear, other students or developers can pick it up and continue work easily.
- **Easier maintenance & fewer bugs**: Clear structure reduces the chance of errors and makes fixing bugs easier.
- **Improved learning**: As you practice friendly coding, you yourself understand programming concepts better.
- **Reuse in future**: Modules/functions you write cleanly today can be reused in other projects.
- **Professional growth**: Even if you code casually now, friendly coding is a habit used by professional programmers and will help you in future.

Challenges

- It takes **time and discipline**: Writing friendly code may slow initial development because you pay attention to naming, structure, comments.
- Over-engineering risk: Sometimes beginners might over-modularize or over-comment, making code complex.
- Balancing readability vs performance: On rare occasions, extremely optimized code may look less readable; you must balance friendly coding with performance.
- Changing bad habits: If you are used to writing “quick and dirty” code, moving to friendly coding may require unlearning shortcuts.

Despite these challenges, the long-term benefits far outweigh the initial effort.

Conclusion & Best Practices Summary

In conclusion, friendly coding is about writing code that works and is easy to work with. For a student like you, adopting friendly coding habits now will set you up for better projects, clearer understanding and less frustration down the road.

Best Practices Summary:

- Use meaningful names for variables/functions.
- Keep code simple; prefer clarity over cleverness.
- Follow consistent formatting and style.
- Break code into functions/modules; each does one job.
- Avoid repeating yourself (DRY).
- Add comments where logic is complex, but let the code speak for itself.
- Use tools (linters, formatters) and review code.
- Refactor regularly to keep code clean.
- Always assume someone else (or your future self) will read your code.

Start with one friendly coding habit in your next project. After some practice, you'll find your code is more understandable and you feel more confident.

References

1. "Coding Best Practices and Guidelines for Better Code", DataCamp. DataCamp

2. “Best Practices for Coding, Organization, and Documentation”, MIT Communication Lab. mitcommlab.mit.edu
3. “Coding Standards and Best Practices to Follow”, BrowserStack. BrowserStack
4. “How to Write Clean Code – Tips and Best Practices (Full Handbook)”, freeCodeCamp. FreeCodeCamp
5. “Best Coding Practices For Beginners”, CipherSchools Blog. cipherschools.com
6. “Best Practices in Programming: Clean Code for You and Your Team”, Dev.to. DEV Community
7. “Coding best practices”, Wikipedia. Wikipedia
8. Barbosa, L. et al., “To What Extent Cognitive-Driven Development Improves Code Readability?”, arXiv. arXiv
9. Pruim, R. et al., “Fostering better coding practices for data scientists”, arXiv.