My Comments for Better Python Programming

1. Take advantage of Python itself. (a.k.a. *Pythonic*)

- e.g. Swap using unpacking

```
temp = a
a = b
b = temp
```

vs.

```
(a, b) = (b, a)
```

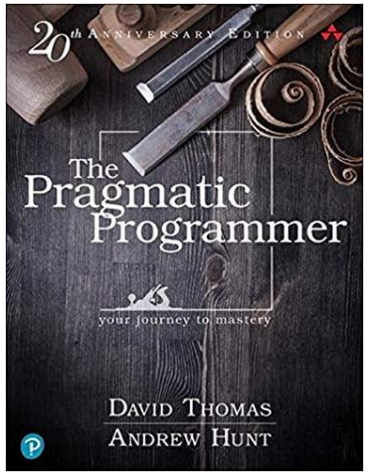
- References
  - Code Style, The Hitchhiker's Guide to Python
  - Write More Pythonic Code, Real Python
  - PEP 8 – Style Guide for Python Code, Python

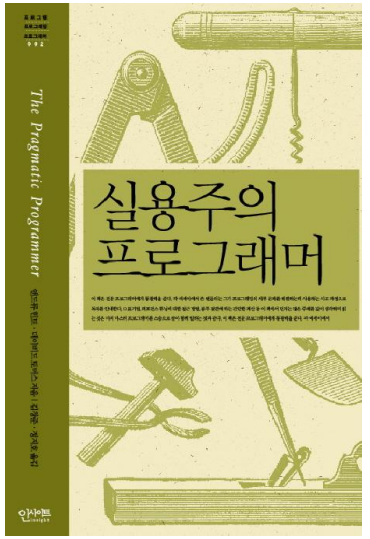
2. Utilize the existing libraries (a.k.a. *Don't reinvent the wheel*) and master them if they are useful.

- Problem #1) Too many libraries
  - ~~Search your keywords in **Google**, Github (with *python*), ...~~
  - ~~Select *related* and *popular* libraries (if possible) and read their tutorials and documents.~~ → Ask **ChatGPT** for help.
- Problem #2) A few documents and examples
  - Search your problem in **Google** (or read and analyze their source codes).

My Comments for Better Programming

[The Pragmatic Programmer]

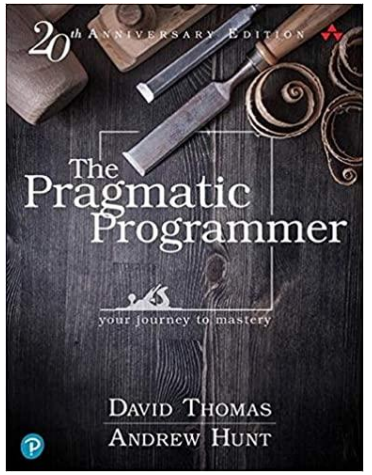


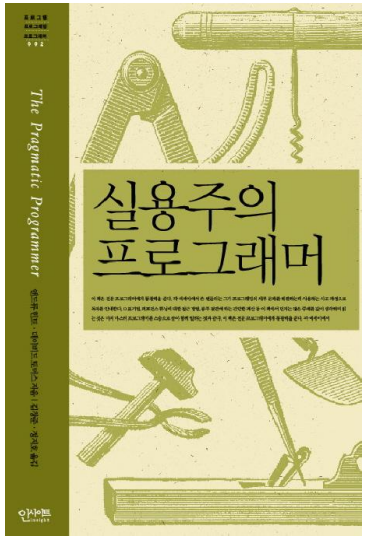


- **Please love your work.**
  - Your work is *your child* and *your part of your soul*. → Please give a name to *your work*.
  - Your work is *your career portfolio*. → Please manage *your Github profile*.
- **More practice makes better programming.**
  - Programming is similar to *problem solving* rather than theoretical understanding.
  - Programming is similar to *swimming* and *riding a bicycle*.
- **Debugging is more important than programming.**
  - You may spend much more time on debugging, than on programming.
  - For better debugging
    - **Data visualization**
      - Please use (conditional) breakpoints rather than printing out data.
    - **Data recording** (a.k.a. dataset)
      - You should be able to *reproduce* your problem.
    - **Test codes** (a.k.a. test-driven development and unit-tests)
    - **Convenient development environments!**
      - It includes not only *software* environments but also *physical* environments.

My Comments for Better Programming

The Pragmatic Programmer





- **Please take advantage of advanced tools.**
  - Advances in IDEs: Syntax highlighting → Auto-completion → **Code suggestion**
  - e.g. Git, CI/CD (automation), **ChatGPT**, **Github Copilot** (free for students)
- ~~**Design Pattern?**~~
  - Please read (and analyze) good implementations (available on Github).
  - My humble design principles
    - **Decoupling** (or divide-and-conquer)
      - e.g. Engine/GUI, data/algorithm, .../test (~ MVC (model-view-controller) pattern)
    - **Avoid repetition.**
    - **Readability** (conciseness and consistency) **matters.**
      - e.g. Naming conventions, comments, coding convention, avoiding a long function, ...
    - **Faster is better.**
- Miscellaneous
  - **Pilot projects** (as *tracer bullets*)
  - **Refactoring** (to avoid a *broken window*)