Programming good practices

This page is based on a document created by Aurélien Nicosia (ULaval) in 2023 called "Bonnes pratiques de programmation en R." It has been updated and written in a more general context.

Regardless of the programming language, employing good programming practices means adhering to certain standards in order to create "good" code. This raises the question of what constitutes "good" code. For me, "good" code is code that fulfills three objectives:

1. It produces the expected results.
2. It is easy to maintain.
3. Its computation time is reasonable.

Why is it desirable to adopt good practices? It allows the code to be understood and used by anyone (and in particular, by yourself in the future). In the long term, good practices increase our productivity by avoiding unnecessary repetition.

Objective 1: Produce the expected results

The priority when developing any computer code is undoubtedly to write code that does what it is supposed to do. Therefore, "good" code must produce the right results. To achieve this, the code must first be functional, i.e., it must not contain any bugs. To ensure that the code works correctly, it simply needs to be tested. It is better to test frequently, with each small addition, rather than producing a lot of code before testing it. This way, there is much less debugging to do. One school of thought in computer science even advocates writing tests before writing code (*test-driven development*). However, a better practice is to formalize the tests so that they can be easily rerun when future changes are made to the code.

Objective 2: Easy-to-maintain code

Maintaining computer code means ensuring that it continues to function properly in the future, despite any changes made to it. Code that is used frequently is code that will need to be updated, either to add features or to fix bugs that were not detected by testing but discovered by users. Taking over code written by someone else, or written by ourselves a few months ago, is not always an easy task. However, if the code is written correctly, it should not be too difficult to understand and modify.

Code maintenance is based on three principles: versioning, comprehensibility, and reusability.

The principle of versioning is to use software that records the various changes made to the code. The best known is Git. It allows you to navigate between different versions of your code, create multiple versions (called branches), and collaborate with others on the same code. It is truly an essential tool to have in your arsenal. Versioning can then be recorded on Github, which manages the underlying machinery. For example, this site uses Git and Github for versioning, and you can see the different versions here.

Comprehensible code is clear and easy to read (almost like text). It often includes instructions that are self-explanatory. These instructions are typically succinct, as overly long instructions often perform several tasks that are difficult to discern. If reading an instruction does not allow a programmer familiar with the computer language used to understand what it does, it is recommended to insert a comment in the code to explain what the instruction is for. In addition to comments explaining certain instructions, all functions should be documented. The documentation for a function should contain: an explanatory text describing what the function does, a description of the arguments accepted as input, a description of the results produced, and an example of use. When programming, it is also good practice to follow a style guide. A style guide is a set of rules that developers have agreed upon to ensure consistent syntax across different projects. In R, you can use the tidyverse style guide. In Python, you can use the PEP8 style guide written by the creator of Python. And in Julia, you can use the style guide provided with the language manual. You can use a **linter**, a static code analysis tool, to help you comply with these style guides. Note that these style guides are recommendations and there is no obligation to follow them. Some rules contradict each other, so I would even recommend not following some of them.

The most common way to make code easy to reuse is to turn it into functions that can then be shared through a package.

Objective 3: Sufficiently fast code

Once we have ensured that our code works correctly and is easy to maintain, we can focus on its execution time. Although this is not the most important criterion for defining "good" code,

it is still a criterion that should not be overlooked, as code that is too slow may not be used. To produce computationally efficient code, you need to:

- put a few simple tricks into practice, i.e., use the optimized syntaxes of the different languages;
- compare the execution time of different ways of programming a task;
- sometimes perform parallel calculations;
- sometimes program pieces of code in another, lower-level language.

In summary

In summary, to adopt good programming practices, you need to:

- **Test** your code frequently.
- Use **version control** software.
- **Document** your code.
- Follow a **style guide**.
- **Factorize** your code by creating functions and packages.
- **Optimize** execution time.