Pseudocode 101: An Introduction to Writing Good Pseudocode

towardsdatascience.com/pseudocode-101-an-introduction-to-writing-good-pseudocode-1331cb855be7

23 June 2021

Make it clear, easy to follow, and understand

As developers or data scientists, we often go through many stages, from getting an idea to reaching a valid, working implementation of it. We need to design/ validate an algorithm, apply it to the problem at hand, and then test it for various input datasets.

In the initial state of solving a problem, it helps a lot if we could eliminate the hassle of having to be bound by the syntax rules of a specific programming language when we are designing or validating an algorithm. By doing this, we can focus our attention on the thought process behind the algorithm, how it will/ won't work instead of paying much attention to how correct our syntax is.

Here where *pseudocode* comes to the rescue. *Pseudocode* is often used in all various fields of programming, whether it be app development, data science, or web development. *Pseudocode* is a technique used to describe the distinct steps of an algorithm in a manner that is easy to understand for anyone with basic programming knowledge.

How to Learn Programming The Right Way

The syntax shouldn't be the first step!

towardsdatascience.com

Although *pseudocode* is a *syntax-free* description of an algorithm, it must provide a full description of the algorithm's logic so that moving from it to implementation should be merely a task of translating each line into code using the syntax of any programming language.

Why use pseudocode at all?

1. **Better readability**. Often, programmers work alongside people from other domains, such as mathematicians, business partners, managers, and so on. Using pseudocode to explain the mechanics of the code will make the communication between the different backgrounds easier and more efficient.
2. **Ease up code construction.** When the programmer goes through the process of developing and generating pseudocode, the process of converting that into real code written in any programming language will become much easier and faster as well.
3. **A good middle point between flowchart and code.** Moving directly from the idea to the flowchart to the code is not always a smooth ride. That's where pseudocode presents a way to make the transition between the different stages somewhat smoother.
4. **Act as a start point for documentation.** Documentation is an essential aspect of building a good project. Often, starting documentation is the most difficult part. However, pseudocode can represent a good starting point for what the documentation should include. Sometimes, programmers include the pseudocode as a docstring at the beginning of the code file.
5. **Easier bug detection and fixing.** Since pseudocode is written in a human-readable format, it is easier to edit and discover bugs before actually writing a single line of code. Editing pseudocode can be done more efficiently than testing, debugging, and fixing actual code.

The main constructs of pseudocode

The core of pseudocode is the ability to represent 6 programming constructs (always written in uppercase): *SEQUENCE, CASE, WHILE, REPEAT-UNTIL, FOR, and IF-THEN-ELSE*. These constructs — also called keywords —are used to describe the control flow of the algorithm.

1. **SEQUENCE** represents linear tasks sequentially performed one after the other.
2. **WHILE** a loop with a condition at its beginning.
3. **REPEAT-UNTIL** a loop with a condition at the bottom.
4. **FOR** another way of looping.
5. **IF-THEN-ELSE** a conditional statement changing the flow of the algorithm.
6. **CASE** the generalization form of IF-THEN-ELSE.

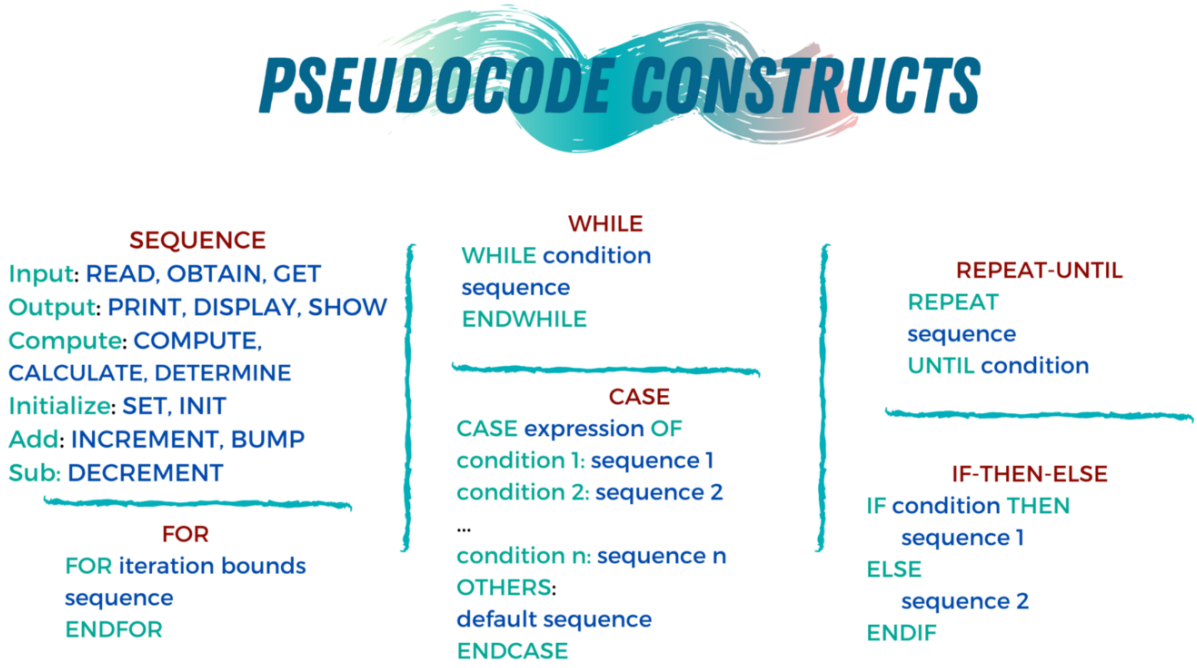

Image by the author (made using Canva)

Although these 6 constructs are the most often used ones, you can theoretically use them to implement any algorithm. You might find yourself needing some more based on your specific application. Perhaps the two most needed commands are:

1. Invoking classes or calling functions (using the *CALL* keyword).
2. Handling exceptions (using *EXCEPTION, WHEN* keywords).

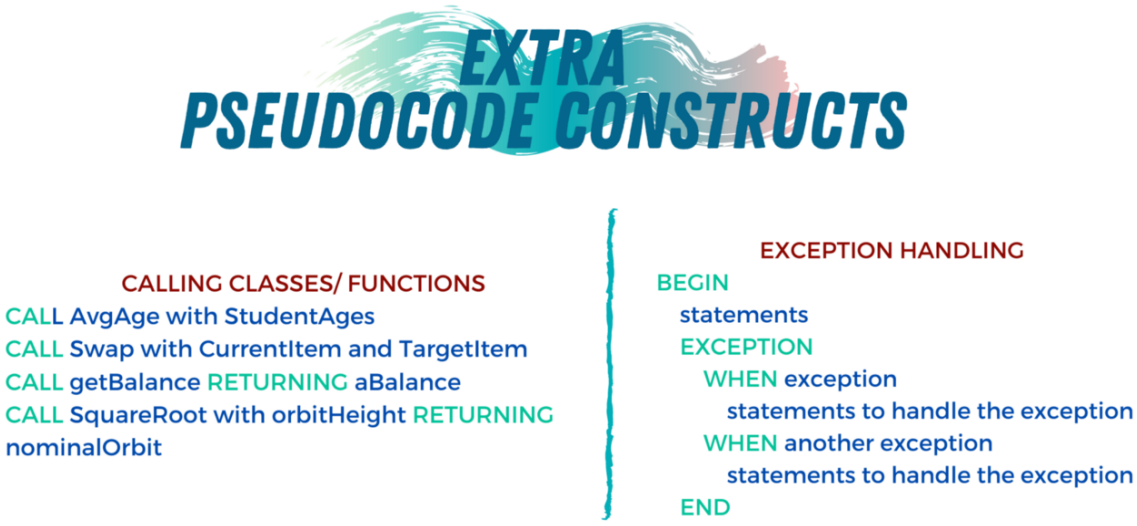

Image by the author (made using Canva)

Of course, based on the field you're working in, you might add more constructs (keywords) to your pseudocode glossary as long as you never use these keywords as variable names and that they are well known within your field or company.

Rules of writing pseudocode

When writing pseudocode, everyone often has their own style of presenting things out since it's read by humans and not by a computer; its rules are less rigorous than that of a programming language. However, there are some simple rules that help make pseudocode more universally understood.

1. Always **capitalize** the initial word (often one of the main 6 constructs).
2. Have only **one** statement per line.
3. **Indent** to show hierarchy, improve readability, and show nested constructs.
4. Always **end** multiline sections using any of the END keywords (*ENDIF*, *ENDWHILE*, etc.).
5. Keep your statements programming language **independent**.
6. Use the naming domain of the **problem**, not that of the **implementation**. E.g., "*Append the last name to the first name*" instead of "*name = first+ last.*"
7. Keep it **simple**, **concise**, and **readable**.

Following these rules help you generate readable pseudocode and be able to recognize a not well-written one.

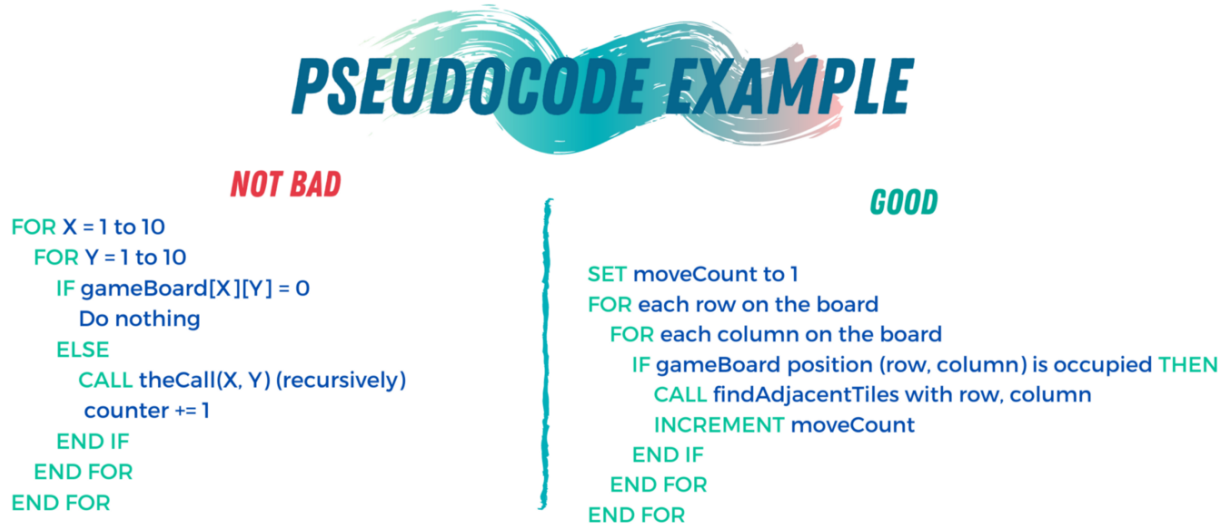

Image by the author (made using Canva)

Final Thoughts

If you're a computer science major, went to Bootcamp, or took any programming class, you've probably heard of pseudocode before. When I teach my students' pseudocode, at first, they don't see the use of it; they think it's a waste of time; as they put it, "*why write "code" twice?*".

That might be correct in the case of simple, straightforward problems. However, as the complexity and the size of the problem increase, they start to realize how generating pseudocode makes writing the actual code much easier. It helps you realize possible problems or design flaws in the algorithm earlier in the development stage.

Hence, saving more time and effort on fixing bugs and avoiding errors. Moreover, pseudocode allowed programmers to communicate more efficiently with others from different backgrounds, as it delivers the algorithm's idea without the complexity of syntax restrictions.

A clear, concise, straightforward pseudocode can make a big difference in the road from idea to implementation, a smooth ride for the programmer. It’s one of the overall tools underestimated by the programming community but defiantly, needs to be utilized more.