Because of my freedom to use as many for-loops as I wanted to (for better or worse) in my Go program, I had less functions to do tasks that I could do with for loops. This made the two functions I wrote for the program very chonky, dense, and—in my opinion—harder to read than my Kotlin program. The difficulty in readability in my Kotlin program comes from the order of my functions in the code, but I think if you follow the flow/stack call, it is easy to follow. Meanwhile, since my Go program has many (many) nested for-loops and if-statements, it can be challenging to read and hard to follow the flow compared to the Kotlin program (since specific tasks in the Kotlin program are named because the functions are named whereas in the Go program…you just got for-loops).

I had a considerably easier time debugging with my Kotlin program because I knew where the bug was, and my issues in writing it was in the design step. Meanwhile, my Go program gave me a lot of grief while debugging, superficially, because I didn't realize I was looking for the bug in the wrong place first, implementation-wise because I didn't break up the program into smaller parts to test the individual parts because I didn't know how to and overconfidence. My style of programming factored into this ("get it all down before I lose the idea"), but the "brick-by-brick" style of functional programming made me to slow down and test functions first, whereas with the imperative style I just had no idea how to do the same. I did also get annoyed by the fact that I had to manually change the slice size when I wanted to test with a smaller sample size in Go whereas I didn't have to with Kotlin, but, again, not sure if that's a "my design decision" thing or a "imperative vs. functional" thing. Given that my Kotlin program is easier to read and debug, and broken up in neater chunks, I think it would be easier to modify the Kotlin program. Also, the fact that it has less hard-coded variables so when I want to use a different sized data set, I don't get errors related to that, like with my Go program.

Speaking of "things that annoyed me when writing Baby Names in Go," Go's being pass by reference. The pass by value of Java and Kotlin is the intuitive system to me, probably because I first learned programming with Java. I found it frustrating to have to work around how to permanently modify a value, which did contribute to my Go program's wordiness. It didn't significantly affect how the two programs ran (in the sense that they both ran and were accurate; I don't know how they compare timewise), but I didn't account for the case in my file reader function where there's an error and it returns nil, which has a chance to be an issue, mostly because I didn't handle it. In the end, I prefer working with the value model since it's how I tend to design the program in, though I do see the value in pass by reference, specifically making you return the variable you're making the changes to so that you are certain that you want to make those changes and have them be permanent.

Personally, it wasn't a huge annoyance to me that I had to compile and run my Kotlin program to get syntax errors, since VSC wouldn't be yelling at me that I have things wrong or unfinished, like with Go, but the convenience of having VSC say what a built-in function does and give those in-editor errors is very nice. Would have been nice if the source-code editor did that for Kotlin as well.