Reflection on the assignment regarding “Meaningful names” and “Functions”.
Author: Jimmy Karlsson, 2023-09-23

Robert describes his initial code-writing as long and complicated. I resonate with this, calling this phase ‘Brutecoding’ essentially a blunt, inelegant draft focused on solely functionality. This assignment was similarly approached in a method of increment refinement.

Robert and I agree on a lot of things in terms of function writing, though I believe he over-emphasise small. My aim is for functions to transparently convey their purpose, with a mindful approach to minimizing nested conditions. Adopting an 'abort early' mindset has proven effective.

Robert’s ideal is that code should behave exactly as expected, “you know you’re working on clean code when a function does exactly what you expect of it”, but also acknowledge that this rarely occurs. I agree with the ideal, but want to highlight the potential pitfall: with increasing levels of abstraction, there's a risk of misinterpreting functionality, and consequently having to dig deeper before you find out.

In pursuit of that ideal I have during the assignment identified that I can be better at writing small check functions, isString as one example, to make, validators smaller, and that I should also lowering my threshold for when a painted type should become a class.

When it comes to naming, I personally consider my names to be fairly on par.
I like my code to make sense even a week from now. My major bane in Martins naming rule, and that is the “Don’t be cute” (as I renamed it “Don’t be a smartass”) principle. In that name I consider too obvious tend to fall for this,
prime example the nonMagicZero previously mentioned in my identifier table, it to be polite, too witty.

Like Robert, I value automated test suites. During this coding assignment, when my grasp on the necessary class characteristics was sufficient, I found myself oriented towards test-driven development.

Retrospectively and in conclusion, further time spent on my initial analysis would have allowed me to reach this, more productive point of test-driven development, sooner.