Rob Pike's 5 Rules of Programming

- from: https://users.ece.utexas.edu/~adnan/pike.html

Rob Pike's 5 Rules of Programming

Rule 1. You can't tell where a program is going to spend its time. Bottlenecks occur in surprising places, so don't try to second guess and put in a speed hack until you've proven that's where the bottleneck is.

Rule 2. Measure. Don't tune for speed until you've measured, and even then don't unless one part of the code overwhelms the rest.

Rule 3. Fancy algorithms are slow when n is small, and n is usually small. Fancy algorithms have big constants. Until you know that n is frequently going to be big, don't get fancy. (Even if n does get big, use Rule 2 first.)

Rule 4. Fancy algorithms are buggier than simple ones, and they're much harder to implement. Use simple algorithms as well as simple data structures.

Rule 5. Data dominates. If you've chosen the right data structures and organized things well, the algorithms will almost always be self-evident. Data structures, not algorithms, are central to programming.

Pike's rules 1 and 2 restate Tony Hoare's famous maxim “Premature optimization is the root of all evil.” Ken Thompson rephrased Pike's rules 3 and 4 as “When in doubt, use brute force.”. Rules 3 and 4 are instances of the design philosophy KISS. Rule 5 was previously stated by Fred Brooks in The Mythical Man-Month. Rule 5 is often shortened to “write stupid code that uses smart objects”.

John Carmack's Advice

On the Lex Friedman podcast, John Carmack who should need no introduction was asked what the best programming language was, given that the answer would also reflect what his favourite programming language was. I will add this to the list as the sixth rule; the Carmack rule.

Rule 6. “The value of not having a random mutable state that you lose track of”. It's not about writing of the program initially, it's the whole lifespan of the program, and that's when it's not necessarily how fast you wrote it, or how fast it operates– but it's how can it bend and adapt as situations change.

Rule 7, he continues, is how well it hands off between the continuous revolving door of programmers taking over maintenance and different things and how you get people up to speed in different areas and there's all these other different aspects of it.

Rule 8. The best programming language is the one you're using. In almost every case I have seen where people mixed languages on a project, it has been a mistake.