Professional Lessons and Opinions

Yann Esposito

[2019-07-04]

I could talk to much about that. But a few short written down lessons.

If you want to reach productivity and not necessarily enlightenment. You have to stop learning and use what you know well. And do not try to use things you don't know.

In particular, most programming languages/IDE/workflow/utils/tools have basic and advanced features. If your want to produce and deliver in time. Your best bet is to limit yourselves to a minimal set of features that give you enough power of combination instead of looking for a super generic strong solution.

Clojure for example has:

- functions
- higher-level functions
- destructuring
- defmethods
- protocols
- macros
- atoms, agents, core.async
- Java FFI

In fact, writing a fully working app you only need, basic data structure (edn) and functions.

That's it.

I was part of a team that created an advanced full featured app using only those. No magic, just taking care of the state and not writing spaghetti code.

How I choose

- Functional programming is superior to imperative and object oriented languages

Programming Languages Quality/Fun

- C, too low level, no higher level function, pb with portability
- Ruby, slow, concets not clean enough
- Python, C-like script, pretty efficient, but quite easy to write bad code, often slow
- Java, cumbersome, lot of boilerplate, not that bad, but bad underneath philosophy (try to make programming scalable by multiple user, replace coder by gears, in practice it is often wrong)

Not suitable for production/ toy language

- logo
- basic

Superior languages

Rust supersede: C, pascal, go

Clojure/Haskell supersede: Ada, C++, Eiffel, Ruby, Python

Best choice of language depending of usage

- do something fast and dirty: zsh, perl for program shorter than 200 lines.
- very complex big software: Haskell, Clojure, Purescript, Clojurescript
- low level fast, fine grained memory control: Rust