What programming paradigms?

Petr Svarny

Semestr

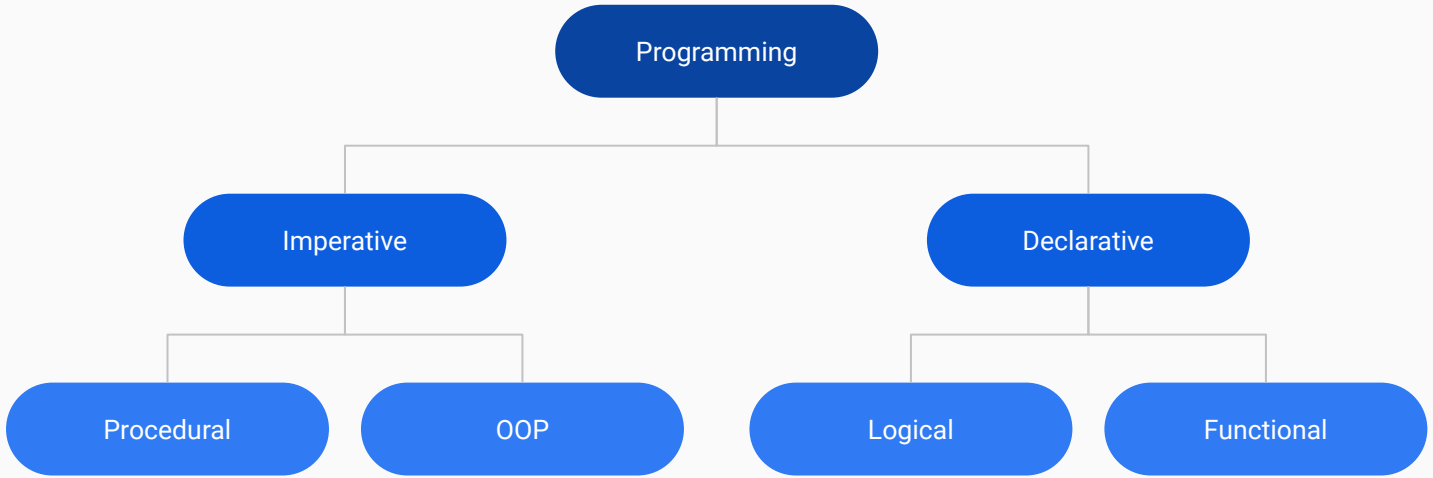
- Seminář, tj. očekává se aktivní účast
- Zápočet na základě prezentace či práce
- Teoretický přehled referáty
  - Procedurální
  - Objektově orientované
  - Logické
  - Funkcionální
- Praktické učení se pomocí Pythonu a dalších jazyků

Programming paradigm

“A paradigm is a way of **doing** something (like programming), not a concrete thing (like a language). Now, it’s true that if a programming language L happens to make a particular programming paradigm P easy to express, then we often say “L is a P language” (e.g. “Haskell is a functional programming language”) but that does not mean there is any such thing as a “functional language paradigm”.”

Ray Toal

Paradigm list



Separate question - “ the level”

High level programming

- more abstraction from the HW
- majority of modern languages are “high level” but can differ in degree

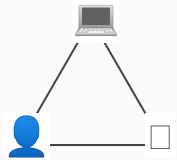
Low level programming

- close to the actual HW it is running on

The three tribes

- **ART** You are a poet and a mathematician. Programming is your poetry
- **HACK** You are a hacker. You make hardware dance to your tune
- **TOOL** You are a maker. You build things for people to use

Seen by their source code, code execution, notion of correctness and user-interface (UI) preference.



“Artist”

Source code: dense, hard to read, easy to execute.

Execution: exact details of execution are not important, but the code should be elegant.

Correctness: A program is correct if it implements the specification exactly.

UI: Beautiful code is more important than beautiful UI.

Haskell, Lisp, ML (Ocaml, etc), Clojure, ADA

“Hacker”

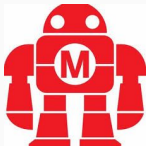
Source code: clean, but clean code is less important than a clean execution.

Execution: How the computer executes your code is paramount. Always think about the program execution.

Correctness: A program is correct if it runs the way you expect it to run, given normal parameters. Execution elegance is more important than correctness.

UI: interaction with humans? Optional.

A short video about history and also making fun of this: http://worrydream.com/dbx/



“Doer”

Source code: The code should be clean, but only because cleaner code is easier to iterate on.

Execution: The program only has to be fast enough for the users. Don’t optimize, add features.

Correctness: Bugs are bad only in proportion to their impact. The program should act the way the users expect it to act.

UI: Users are important and thus also the UI!

JS, Ruby, Python, Swift, C#.

Example

```
float Q_rsqrt( float number )

{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                       // evil floating point bit level hacking
    i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration

//    y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed
    return y;
}
```

https://en.wikipedia.org/wiki/Fast_inverse_square_root Quake III code

References

LMU notes: https://cs.lmu.edu/~ray/notes/paradigms/

Three tribes article: https://josephg.com/blog/3-tribes/