INF-3910-5

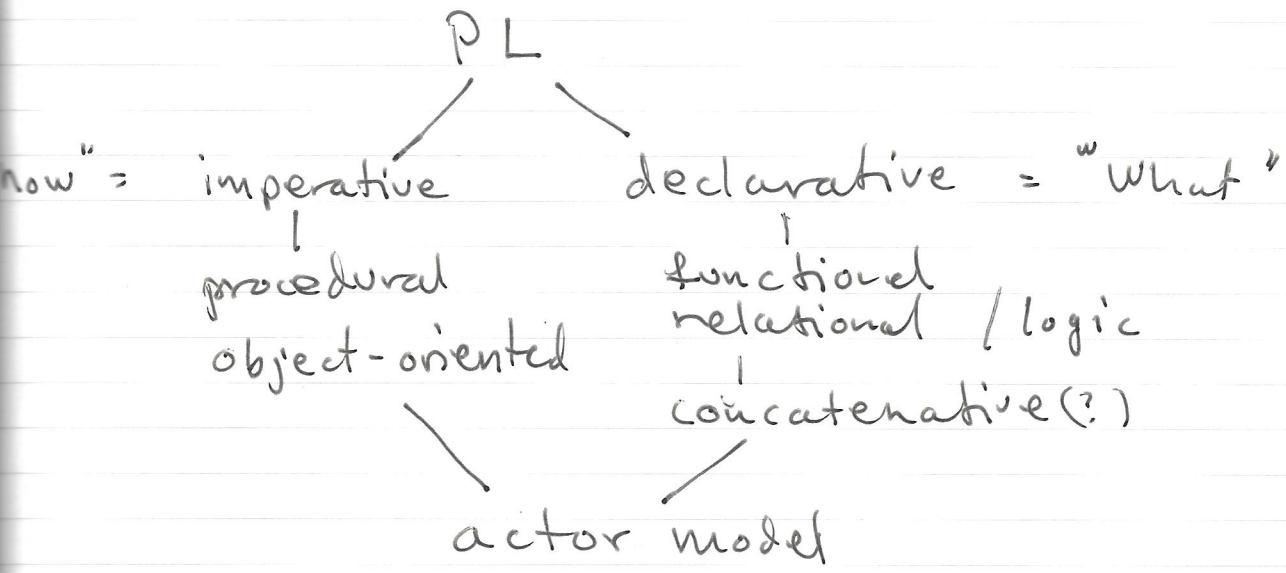
Lecture 1

- Practical information:
  - ~~Assistant?~~
  - 3 obligs
  - Home exam
  - github, Slack/Zulip
  - JJ: background, CV, contact
- Agenda on github
- Useful links and presentations on github
- This is a course in applied functional programming, not a course in programming
- Although the aim is practical, we need a theoretical context to guide our understanding
- Abstraction as a guiding principle

- What is programming really?
- Earliest programmers: Jaquard, Babbage, Ada
- 1936: Turing and Church are working on the Entscheidungsproblem in logic
- Turing invents the Turing machine
- Church invents lambda calculus
- Turing machines are imperative, operating on statements of 0:s and 1:s, like a CPU
- Church Lambda calculus is declarative, operating on expressions

- Turing machines can be implemented in HW, and Turing "wins"

[- x86-64: 981 - 3,683 instructions
⇒ MoVfuscator]



- Each paradigm has its languages and its strengths and weaknesses
- Unless we learn and understand each paradigm, we cannot really judge the merits and drawbacks
- The "Galaxy Brain" list:

1957: Fortran	1974: SQL
1958: LISP	1980: SmallTalk
1968: ALGOL68	1986: Eiffel, Erlang
1972: C	1989: Coq
1972: Prolog	1995: Java
1973: ML	2017: Idris

[Ref. Bruce Tate: Seven Languages in 7 weeks]

[Ref. Schwapfirfun and profit / vides / Four languages from Forty Years Ago]

Functional programming:

- programming (mostly) with pure, mathematical functions
- Expressions over statements
- Equational reasoning
- Solid mathematical foundation
- Type theory and category theory

⇒ guarantees about program behaviour

- Mathematicians have used 2500 years developing and perfecting their language for correctness and precision. Programmers have spent the last 50 years doing the opposite. (culminating in JavaScript)

Why Functional Programming?

- Simplicity!
- Reasoning
- Abstraction
- Refactoring
- Fun
- Code length
- Composition
- Reuse

Complexity I

Inherent: Part of the problem specification, not your fault!
(Quantum mechanics is pretty hard)

Incidental: Part of the solution, your fault!
(Programming languages, editors, Computers)

Complex: Intertwined, coupled, opaque, spaghetti, hard

Simplex: Unbraided, transparent, decoupled, simple

Simple is not the same as easy!

Easy is just something we master, near at hand. Simple is hard!

"Simplicity is prerequisite of reliability"
- Dijkstra

"Simplicity is the ultimate sophistication"
DaVinci

[Ref: Hickey: Simple made easy]

Complexity II

- The n°1 root cause of complexity is mutable state (variables).
- Variables destroy referential transparency
- Variables introduce artificial time dependence into programs
- Variables destroy mathematical reasoning
- Variables couple code at long distances

⇒ Variables are like ink in water, they taint everything which comes in contact

$f: (x+1)(x+2)$

```
def x:
    x = x + 1
    y = x + 1
    return x * y
```

⇒ ↕

```
def x:
    y = x + 1
    x = x + 1
    return x * y
```

} $(x+1)^2$

```
def x:
    a = x + 1
    b = a + 1
    return a * b
```

or

```
def x:
    a = x + 1
    b = x + 2
    return a * b
```

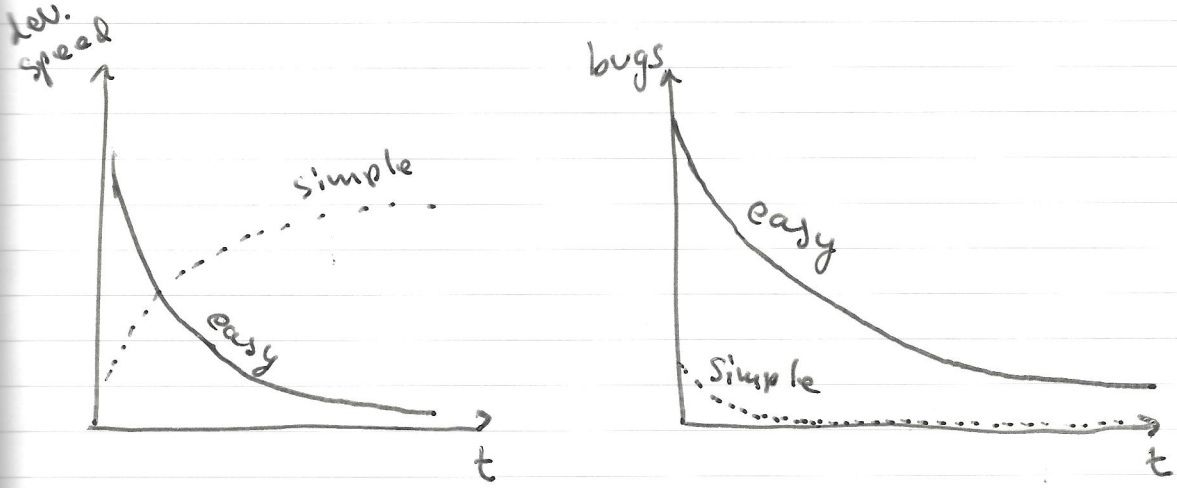
[Ref: Moseley & Marks: Out of the tar pit]

Complexity III

Discarding mutable state leads to functional or declarative programming in some form or another since the straight imperative form becomes impossible.

⇒ Functional programming reduces complexity compared to imperative approaches.

When combined with type theory and category theory, invalid state can be made unrepresentable. This can give us great confidence in the correctness of our code!



[Red: Hickey: Simple made easy]

Abstraction

- FP is quite normal programming, the major difference is in the abstractions used.
- Abstraction is about generalizing form and patterns, extracting the bare essence of ideas and things
- Abstraction is hard! Our brains have trouble accepting abstractions at first glance
- Examples:
  - a map abstracts landscape and location
  - geometry:


icecream


road sign

cone

$$V = \frac{\pi r^2 h}{3}$$

  - written language: fun, 乙, 开, 天
  - Code:

F#:
```
let fibSeq = Seq.unfold (fun (a,b) ->
                Some (a+b, (b, a+b))) (0,1)
```

JS:
```
function fib(n) {
  const result = [0,1];
  for (var i=2; i <= n; i++) { const a = result[i-1];
                               const b = result[i-2];
                               result.push(a+b); }
  return [n]; }
```

Isomorphism is a very important concept in mathematics and programming.

Two objects are isomorphic if we can transform between the two forms without loss or addition of information.

Ex. [1, 3, 5, 3] $\Leftrightarrow$ {(2,3); (4,3); (1,1); (3,5)}

Composition allows us to build complex components from simpler, reusable building blocks.

- It is essential that the building blocks are independent and locally defined for the product to be well defined.

- Examples:

  shell: cat hello.txt | tr A h | sort | uniq

  function composition: $f \circ g = f(g(x))$

  print(sin(exp(-1)))

Category theory is the mathematical theory of composition