

# Presentation

## MPRI 2.4

François Pottier



2024

## Why follow this course?

Computers are wonderful machines...



## Why follow this course?

Computers are wonderful machines...



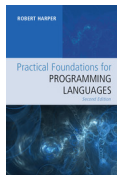
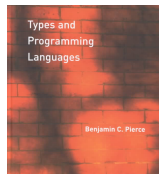
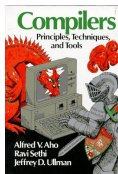
... but they don't always do what was intended.

## Why follow this course?

The **theory of programming languages** aims to describe

how programs are **structured**,  
what they **mean**,  
how they are **interpreted** or **compiled**,

and how one can **prove**  
properties of **programs**  
and properties of **tools**,  
such as type-checkers or compilers.



## What is functional programming?

Programming in Scheme, OCaml, Haskell, Scala, Agda, Coq, F\*, ...

Key features:

- **Immutable variables** and **values**. Mutable state discouraged.
- **Functions** as values. Higher-order functions.
- **Algebraic data structures** (lists, trees, ...) as values.
- **Recursion**. Tail recursion preferred to loops.
- Close to **mathematical language** and to the  $\lambda$ -calculus.
- A taste for expressive, safe, static **type systems**. Polymorphism.
- **Automatic memory management** preferred.
- **Equational reasoning** about pure programs.
  - *A program does not “do” something; it “is” something.*

# What is functional programming?

```
(* Do not think of data as memory blocks and pointers --  
   think in terms of sums, products, and recursion. *)  
type 'a list =  
| []  
| (::) of 'a * 'a list
```

# What is functional programming?

```
(* Parameterize [map] with the transformation [f]
   that should be applied to every list element. *)
let rec map f xs =
  (* Let the structure of the data
     guide the structure of the code. *)
  match xs with
  | [] -> []
  | x :: xs -> f x :: map f xs
      (* Do not modify the input list
         -- allocate a new list. *)

let add x ys =
  map (fun y -> x + y) ys
      (* ~~~~~ This closure refers to [x]. *)
```

## What is functional programming?

```
(* Do not write a loop -- write a tail-recursive function. *)  
(* [rev_append xs ys] is equivalent to [rev xs @ ys]. *)  
let rec rev_append xs ys =  
  match xs with  
  | [] -> ys  
  | x :: xs -> rev_append xs (x :: ys)  
  
(* Do not be afraid to write many small functions. *)  
let rev xs =  
  rev_append xs []
```

Steele, **Lambda: the ultimate GOTO**, 1977.



## Why learn functional programming?

Functional programming is a **culture** — a **school of thought**.

It differs from “mainstream” programming in **pedagogical** ways:

- A belief that mutable data, jumps and loops are not fundamental,
- A belief that functions are more fundamental than objects,
- A taste for **declarative** thinking.

Furthermore, it has a tradition of **solid (meta)theory**:

- **formal definitions** of semantics, type systems, code transformations...
- **proofs** of type soundness, proofs of semantic preservation, ...
- moving towards **machine-checked** definitions and proofs.

## Why follow this course?

An introduction to **programming languages** (PL),  
a subfield of computer science (CS).

Our conferences are **POPL**, **ICFP**, PLDI, OOPSLA, ESOP, LICS, etc.

In this course, we wish to teach at the same time:

- several key **programming techniques**;
- the (meta)**theory of programming languages**.

## What topics are hot these days?

The 270 papers submitted to **POPL 2023** discuss, among other topics:

- quantum programs;
- program synthesis;
- probabilistic programs;
- automatic differentiation;
- proofs of (concurrent / distributed) programs  
(with relaxed / persistent memory);
- information flow (differential privacy, ...);
- static analysis (via CFL reachability, abstract interpretation...);
- type theory (with linear types, modal types, ...);
- industrial languages (WebAssembly, SPIR-V, Hack, ...);
- features of industrial hardware (ARM, ...).

We aim to offer the basic culture needed to approach these topics.

## What is in this course?

Four main segments:

- Semantics, Type Systems, and Program Transformations (**Pottier**)
- Semantic Proofs of Type Soundness and Logical Relations (**Scherer**)
- Typed Programming (**Dagand**)
- Programming with Resources in Rust (**Jourdan**)

A **detailed syllabus** appears online.