# Foundations of Software

Sébastien Doeraene, EPFL

Course material by Martin Odersky, EPFL

Slides in part adapted from:
University of Pennsylvania CIS 500: Software Foundations - Fall 2006
by Benjamin Pierce



# Course Overview



## Staff

| | |
|---|---|
| Instructor: | Sébastien Doeraene<br>sebastien.doeraene@epfl.ch |
| Pronunciation | Dutch/French/Japanese/English<br>Doeraene/Douranne/ドゥラン/Doorann |
| Teaching Assistants: | Aleksander Boruch-Gruszecki<br>aleksander.boruch-gruszecki@epfl.ch |
| | Cao Nguyen Pham<br>nguyen.pham@epfl.ch |



## What is "software foundations"?

Software foundations (or "theory of programming languages") is the mathematical study of the meaning of programs.

The goal is finding ways to describe program behaviors that are both precise and abstract.

- precise so that we can use mathematical tools to formalize and check interesting properties
- abstract so that properties of interest can be discussed clearly, without getting bogged down in low-level details

## Why study software foundations?

- To prove specific properties of particular programs (i.e., program verification)
  - Important in some domains (safety-critical systems, hardware design, security protocols, inner loops of key algorithms, ...), but still quite difficult and expensive
- To develop intuitions for informal reasoning about programs
- To prove general facts about all the programs in a given programming language (e.g., safety or isolation properties)
- To understand language features (and their interactions) deeply and develop principles for better language design
  (PL is the "materials science" of computer science...)



## What you can expect to get out of the course

- A more sophisticated perspective on programs, programming languages, and the activity of programming
  - See programs and whole languages as formal, mathematical objects
  - Make and prove rigorous claims about them
  - Detailed knowledge of a variety of core language features
- Deep intuitions about key language properties such as type safety
- Powerful tools for language design, description, and analysis

Most software designers are language designers!



## Greenspun's Tenth Rule Of Programming

> Any sufficiently complicated C or Fortran program contains an ad-hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp.
> – Philip Greenspun



## What this course is not

- An introduction to programming
- A course on functional programming (though we'll be doing some functional programming along the way)
- A course on compilers (you should already have basic concepts such as lexical analysis, parsing, abstract syntax, and scope under your belt)
- A comparative survey of many different programming languages and styles

## Approaches to Program Meaning

Quescussion: How can we precisely model and study the semantics of computer programs?



## Approaches to Program Meaning

Question: What systems have you heard of or studied that study the semantics of programs?



## Approaches to Program Meaning

- Denotational semantics and domain theory view programs as simple mathematical objects, abstracting away their flow of control and concentrating on their input-output behavior.
- Program logics such as Hoare logic and dependent type theories focus on logical rules for reasoning about programs.
- Operational semantics describes program behaviors by means of abstract machines. This approach is somewhat lower-level than the others, but is extremely flexible.
- Process calculi focus on the communication and synchronization behaviors of complex concurrent systems.
- Type systems describe approximations of program behaviors, concentrating on the shapes of the values passed between different parts of the program.



## Overview

This course will concentrate on operational techniques and type systems.

- Part I: Modeling programming languages
  - Abstract syntax
  - Operational semantics
  - Inductive proof techniques
  - The lambda-calculus
  - Syntactic sugar; fully abstract translations
- Part II: Type systems
  - Simple types
  - Type safety
  - References
  - Subtyping

## Overview

- Part III: Object-oriented features (case study)
  - A simple imperative object model
  - An analysis of core Java
  - An analysis of core Scala.js IR



# Organization of the Course



## Information

Textbook: Types and Programming Languages,
Benjamin C. Pierce, MIT Press, 2002

Webpage: `https://fos-2023.github.io`
Q&A Ed Discussions via Moodle

The electronic version of the book is available for free in the EPFL digital library.



## Elements of the Course

- The Foundations of Software course consists of
  - book reading
  - lectures (Tue 14:15-16:00, INF 1)
  - exercises and project work (Wed 11:15-13:00, INJ 218)
- The lectures will build on required reading from the textbook.
- The course website lists what sections of the book must be read before each lecture.

## Homework and Projects

You will be asked to

- solve and hand in some written exercise sheets,
- do a number of programming assignments, including
  - interpreters and reduction engines,
  - type checkers, and
  - small illustration programs

  for a variety of small languages.
- The implementation language for these assignments is Scala.



## Scala

- Scala is a functional and object-oriented language that is closely interoperable with Java.
- It is very well suited as an implementation language for type-checkers, in particular because it supports:
  - pattern matching,
  - higher-order functions.



## Learning Scala

If you don't know Scala yet, there's help:

- The Scala web site:

  www.scala-lang.org
- On this site, the documents:
  - A Brief Scala Tutorial - an introduction to Scala for Java programmers. (short and basic).
  - An Introduction to Scala (longer and more comprehensive).
  - An Overview of the Scala Programming Language (high-level).
  - Scala By Example (long, comprehensive, tutorial style).
- The assistants.



## Grading and Exams

Final course grades will be computed as follows:

- Homework: 20%
- Projects: 20%
- Final exam: 60%

## Collaboration

- Collaboration on homework and projects is strongly encouraged.
- Studying with other people is the best way to internalize the material
- Groups of 2 or 3 students.

"You never really misunderstand something
until you try to teach it...
" – Anon.



## Plagiarism

- A single group will of course share code.
- But plagiarizing code by other groups as part of a project is unethical and will not be tolerated, whatever the source.