

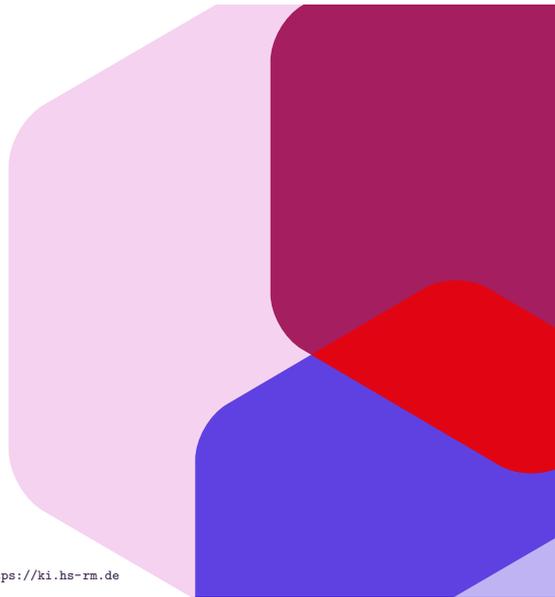
Programming Language Foundations

01 Introduction

Prof. Dr. David Sabel
Winter term 2025/26

Last update: December 17, 2025

Image credits: All images are self-drawn or generated with AI from OpenAI at <https://k1.hs-rm.de>



Prof. Dr. David Sabel

- Professor for Theoretical Computer Science
- since July 2024 at Hochschule RheinMain
- www.davidsabel.de

Consultation hours

- by appointment
- office building C (north), room C 031 or online

Contact: David.Sabel@hs-rm.de

Lecture and Exercises

Exam

- Lecture: Wednesday, 10:00 - 11:30, C 407
- Practical Course: Wednesday, 11:45 - 13:15, C 413

- written exam
- register via COMPASS
- exam registration: 29.12.2025. – 12.01.2026
- bonus for successfully accomplishing the exercises

Exercises



Tasks:

- fill the definitions / results with examples
- calculate examples with pen and paper
- “implement” the definitions as a program (in Haskell and/or Java)
Note: the programs itself are not the learning objective

Rules and bonus for the exam:

- 10 weekly sheets in Subato
- each sheet: 6 points
- total points: 60
- bonus applies if the grade without bonus is 4,0 or better
(bonus will not help to pass the exam)

received points	bonus (better grade)
≥ 18	0.3
≥ 36	0.7
≥ 48	1.0

Ressources



- lecture notes
- slides
- exercises
- references to books etc.

will be made available in StudIP → ILIAS

Books (Selection)



- Glynn Winskel: The Formal Semantics of Programming Languages: An Introduction, MIT Press, 1993
- John C. Mitchell: Foundations for Programming Languages, MIT Press, 1996
- Benjamin C. Pierce: Types and programming languages, MIT Press, 2002
- Aaron Stump: Programming Language Foundations, Wiley 2013
- Chris Hankin: An Introduction to Lambda Calculi for Computer Scientists, King's College Publications, 2004
- Henk Barendregt: The Lambda Calculus. Its Syntax and Semantics, Studies in logic and the foundations of mathematics 103, North-Holland, 1985
- Tobias Nipkow and Gerwin Klein: Concrete Semantics With Isabelle/HOL, Springer, 2014

Websites (Selection)



- Programming Language Foundations in Agda:
<https://plfa.github.io/>
- Software Foundations:
<https://softwarefoundations.cis.upenn.edu/>
- Concrete Semantics:
<https://www21.in.tum.de/~nipkow/Concrete-Semantics/>

Objectives of the Course



Objectives;

- know some formal foundations of programming languages
- know the techniques and methods
- be able to apply most of the techniques

Formal foundations of programming languages

- include problems to get the source code into the computer (lexing and parsing)
we mainly do not care about these problems!
- of course, to represent programs we have to define their syntax:
we use grammars and side-conditions
- our main question is:

How to define and reason about the meaning of programs?

Which Language Should We Investigate?



Characteristics of Programming Languages

- **Programming Paradigm**
 - Imperative programming languages:
 - focus on **how** to execute tasks
 - subclass: object-oriented languages
 - examples: C, C++, Python, Java
 - Declarative programming languages:
 - focus on **what** the program computes
 - subclasses:
 - logical programming languages (e.g. Prolog)
 - functional programming languages (e.g. Haskell, ML).

Which Language Should We Investigate? (Cont'd)



- **Level of Abstraction**
 - Machine languages
 - High-level languages
 - Mid-level Languages
- **Scope of Languages**
 - General-purpose languages
 - Domain-specific languages
- **Computational Power**
 - Turing completeness
 - Non-Turing complete languages

Which Language Should We Investigate? (Cont'd)



Modern programming languages have:

- rich syntax
- difficult constructs
- often: no formal semantics, non-unique semantics
- thus: too complex to investigate in a lecture

→ We look for more basic models:

Turing Machine: Alan Turing's model of computation

WHILE Language: A very simple imperative language

Lambda Calculus: A very simple functional language

Note that the lambda calculus is often used to describe the semantics of imperative languages, logics, ...

- ① Computability: Intuitive computability, Turing machines, Turing computability, Church-Turing thesis
- ② Lambda Calculus: syntax, α -renaming and β -reduction, Church-Rosser-Theorem, call-by-name-, call-by-value, and call-by-need semantics, contextual equivalence, context lemma, encodings of data and recursion
- ③ Functional Core Languages: extended lambda calculi as core language of functional programming, data constructors, case-expressions, recursive super combinators, types, seq-operator
- ④ Polymorphic Type Inference: polymorphic types, type inference for expressions, type inference for recursive functions, iterative type inference, Hindley-Damas-Milner type inference
- ⑤ Semantics: overview of formal semantics, variants of operational semantics for an imperative core-language, denotational semantics for an imperative core language