

Concepts of Modern Programming Languages

Peter H. Fröhlich
phf@cs.jhu.edu

Munich University of Applied Sciences
Department of Computer Science and Mathematics

1 Course Goals

We will study a variety of modern (but not necessarily recent!) programming languages to broaden our horizons beyond Java and friends. Our focus will be on concepts first, particular languages second: Instead of covering a language “in depth” we will cover “just enough of it” to develop reading comprehension in service of conceptual understanding. However, we will also write small programs in each language to round out our experience. We will pay particular attention to issues such as scoping and parameter passing, modularity and abstract data types, type systems and type checking, parametric and subtype polymorphism, as well as abstractions for concurrent programming.

Prerequisites: Intermediate programming skills, familiarity with an object-oriented programming language.

Requirements: A laptop running a recent Debian or Ubuntu install, possibly as a virtual machine.

2 Text Books

There are no required text books to purchase but of course you're welcome to buy a text book anyway. Here are a few suggestions (any edition will do):

1. Robert W. Sebesta: *Concepts of Programming Languages*. Pearson, 11th edition, 2015.
2. Michael L. Scott: *Programming Language Pragmatics*. Morgan Kaufmann, 4th edition, 2015.
3. Daniel P. Friedman, Mitchell Wand: *Essentials of Programming Languages*. The MIT Press, 3rd edition, 2008.

Instead of a printed text book we're going to use freely available online resources. Links will be provided in class.

3 Tentative Outline

The plan (!) is to cover the following programming languages in the order given, focussing on the highlighted concepts in particular.

1. Lambda Calculus: Programming as Pure Mathematics.
2. Lisp and Scheme: Implementing the Lambda Calculus.
3. Standard ML and OCaml: Pattern Matching, Type Inference, Parametric Polymorphism, Modules.
4. Io: Prototypical Objects, Actors and Futures, literally *everything* is an object.

5. Google's Go: Systems Programming, Subtype Polymorphism, Concurrency.
6. Python, Jython, Cython: Pragmatic Scripting across Multiple Paradigms.

For each language we'll cover the most important features in lecture and then apply what we learned in lab, both by modifying example code and writing small programs independently. Lab exercises (which may have to be finished as homework after lab) will be graded as your *Studienarbeit* and determine 60% of your final grade.

4 More Languages

Here are a number of languages (and an interesting focus area for each) that can serve as the basis for your presentation on the last day of classes:

1. Clojure (software transactional memory)
2. Erlang (fault tolerance)
3. Forth (embedded systems, firmware)
4. Haskell (lazy evaluation)
5. JavaScript (everything web)
6. Julia (multiple dispatch)
7. Lua (embedded scripting)
8. Nim (hygienic macros)
9. Plankalkül (the first “real” programming language?)
10. Prolog (logic programming)
11. SQL (relational database queries)

Presentations are 15 minutes, 15 slides each and can be done in pairs (provided each student talks for about half the time). Please practice and time (!) your presentations in advance to ensure that things go smoothly. (This is especially important if you plan on live coding.) Presentations will be graded as your *Referat* and determine 40% of your final grade.

5 Other Notes

I live and teach in Baltimore, MD for most of the year, so I am not offering this course in English “as an option” or some such thing: I'll *actually* speak English 99.5% of the time. Please try to do the same, not just as a courtesy to me but to actually practice your English. Hey, it's like getting an English course for free with your Computer Science course.