CSE P505: Programming Languages
Course Information and Syllabus
Autuman 2016

Logistics and Contact Information: The instructor is Dan Grossman. See the course home page (`http://courses.cs.washington.edu/csep505/16au/`) for all pertinent information. *Ensure your email settings work for promptly receiving course email-list messages.*

Goals: Successful course participants will:

- Master universal programming-language concepts such as control flow, scope, datatypes, functions, continuations, types, and threads such that they can recognize them in strange guises.
- Learn to precisely define and implement core programming-language concepts.
- Attain reasonable proficiency programming in a functional style.
- Find relevant literature somewhat more approachable.

Our primary intellectual tools will be *operational semantics* and *functional programs* (mostly in OCaml but also probably some Haskell and/or Racket). Prior knowledge is not expected.

Audience: This course is for software professionals and 5th-year students with substantial programming experience. No background in functional programming or programming-language theory is needed, though we will "move fast." Experience with imperative and object-oriented programming may be assumed occasionally.

Format: One weekly lecture will develop the course content. Homeworks will extend the material discussed in lecture; expect to learn as you do them. Programming exercises must be done in a particular programming language, mostly OCaml, which will be discussed in class. We will also have a take-home exam at the end of the course.

There is no textbook, but substantial and free materials on OCaml are linked from the course homepage and the course materials will be self-contained. No textbook aligns well with the course materials, but for particular topics, Dan may be able to suggest some relevant textbook chapters or online materials. We will provide pointers to (optional) relevant research papers throughout the quarter as the opportunity arises.

Homeworks, Exams, Grading: We will have 4–6 homeworks (probably 5). Unless announced otherwise, each homework will contribute equally toward the course grade and the final, take-home exam will contribute twice as much as a homework.

Academic Integrity: Any attempt to misrepresent the work *you* have done will result in course failure. If there is any doubt, indicate on an assignment who assisted you and how. In general, you may discuss general approaches to solutions, but you should write your solutions on your own. You should not show your written solution to someone else or view someone else's solution. Particular assignments may include more specific instructions. If not, ask.

Advice:

- In every course, there is a danger that you will not learn much and therefore lose the most important reason to take the course. In P505, avoid "thinking too generally" (not learning enough precise details to appreciate the rigor of programming-language definitions) or "thinking too specifically" (focusing so much on the details that the purpose of the definitions is lost). This balance takes getting used to, but in a ten-week course, time is of the essence.
- If you adopt the attitude, "I will have fun learning to think in new ways" then you will do well. If you instead say, "I will fit everything into the way I already view programming" then frustration is likely.
- While the homework mostly involves writing code, approach this code differently than industrial-strength programs. Your solutions should be short, simple, self-contained, and elegant. Be a minimalist. See the separate document, "Advice on approaching programming in P505."

- The course must necessarily "work" for people with no prior experience studying programming languages and those with quite a bit of experience. Therefore, it is likely that you will find some parts of the course too elementary/slow or too advanced/fast — or both. Do your best to get the most out of each part of the course regardless.
- Come to class. Ask relevant questions.

Approximate Topic Schedule (Very Much Subject to Change):

This list is necessarily woefully incomplete compared to the full breadth and beauty of languages.

- Purpose of studying programming languages
- Functional programming (in OCaml)
  - Lack of mutation
  - Higher-order functions
  - Datatypes
  - Tail recursion
- Defining languages
  - Concrete vs. abstract syntax
  - Introduction/elimination forms
  - Semantics via interpretation
  - Semantics via translation
- Formally defining languages via inference rules
- Lambda-Calculus
- Abstract Machines
- Exceptions, continuations, continuation-passing style
- Lazy Evaluation and Infinite Data Structures
- Type Classes
- Monads
- Types
  - Soundness vs. completeness
  - Type safety
  - Type inference
  - Static vs. dynamic typing
  - Curry-Howard isomorphism
- Polymorphism
  - Subtyping
  - Generics
  - Parametricity
  - Polymorphic references
- Concurrency
  - Threads
  - Locks, Atomicity
  - Relaxed Memory Models
  - Message passing (Concurrent ML)
- Object-oriented programming
  - Extensibility: contrast with functional programming
  - Dynamic dispatch
  - Classes vs. types
  - Multimethods