# CSE 505: Concepts of Programming Languages
# Course Information and Syllabus
**Fall 2003**

**Logistics and Contact Information:** The instructor is Dan Grossman. See the course home page (`http://www.cs.washington.edu/education/courses/cse505/03au/`) for all pertinent information.

**Goals:** We will investigate concepts essential to programming languages including control flow, assignment, scope, functions, types, and objects. As time permits, we will explore more advanced concepts such as continuations, exceptions, concurrency, and connections to logic. Our primary intellectual tools will be *operational semantics* and *O'Caml programs*. (Prior knowledge of neither is required nor expected.)

Successful course participants will learn to:

- Give precise definitions to programming-language features
- Prove properties of inductively defined sets (e.g., well-typed programs)
- Appreciate some programming-language theory jargon (e.g., inference rules) and find the literature more approachable
- Write better programs by exploiting modern language features such as higher-order functions and objects

In short, our goal is to use "theory" to make us better programmers and better researchers. Always think, "how is this related to programs I have written?"

**Audience:** This course is an introductory course for CSE Ph.D. students. Others should contact the instructor to discuss appropriateness. It is intended for students who will never take another languages course and those who will take many more. Experience with functional languages, proofs by induction, and logic may prove useful. (Having such experience might let you "slack" a bit; students without it should not necessarily run away.)

**Format:** Two weekly lectures will develop the course content. The textbook (Types and Programming Languages by Pierce) covers similar material. It will serve as an excellent "second explanation" but we will not follow it closely. Homeworks will extend the material discussed in lecture; expect to learn as you do them. Programming exercises must be done in the O'Caml language, which will be discussed in class. Exams will assess understanding of the lectures and homeworks.

**Homeworks, Exams, and Grading:** We will have four or five homeworks, one midterm, and one final. These six or seven graded items will contribute equally to your course grade unless the schedule demands that one homework ends up significantly "smaller" than the others.

**Academic Integrity:** Any attempt to misrepresent the work *you* have done will result in the maximum penalty the university allows. 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 must write your solutions on your own. You should never show your written solution to someone else or view someone else's solution. Individual assignments may include more specific instructions. If not, *ask*.

**Advice:** It is likely you can pass this course without appreciating what we are doing or retaining much of what you learned. Doing so is a poor use of your time. To "appreciate" and "retain" it is best to:

- Determine how each lecture topic and homework problem fits into the course outline and course goals.
- Master the details of the early topics in the course. The material builds on early material more than it seems.

**Probable Topics:** (subject to change)

- Inductive definitions
- Operational semantics
- Assignment and basic control flow
- Functional programming and introduction to O'Caml
- Scope and variable binding
- Lambda calculus (functions)
- Simple types
- Type safety
- Universal and existential types
- Object-oriented programming
- Inheritance
- Multimethods
- Exceptions and continuations
- Curry-Howard isomorphism

**Probably Nontopics:** There are many topics that are appropriate for this course but we simply will not have time for. We will try to give a hint about what we're missing with respect to the following.

- Denotational semantics
- Axiomatic semantics
- Lambda encodings (e.g., Church numerals)
- Parametricity
- Type inference