This reading is also repeated in the "Course Resources" tab to make it easier to find throughout the course.

This course is an introduction to the basic concepts of programming languages, with a strong emphasis on functional programming. The course uses the languages ML (in Part A), Racket (in Part B), and Ruby (in Part C) as vehicles for teaching the concepts, but the real intent is to teach enough about how any language “fits together” to make you more effective programming in any language -- and in learning new ones.

This course is neither particularly theoretical nor just about programming specifics -- it will give you a framework for understanding how to use language constructs effectively and how to design correct and elegant programs. By using different languages, you will learn to think more deeply than in terms of the particular syntax of one language. The emphasis on functional programming is essential for learning how to write robust, reusable, composable, and elegant programs. Indeed, many of the most important ideas in modern languages have their roots in functional programming. Get ready to learn a fresh and beautiful way to look at software and how to have fun building it.

See below for a specific list of topics the course will cover.

Recommended Background

The course assumes students are familiar with programming covered by most introductory courses, but it is explicitly designed not to be a particularly advanced course. Students should be comfortable with variables, conditionals, arrays, linked lists, stacks, and recursion (though recursion will be reviewed and expanded upon), and the difference between an interface and an implementation. Students should be eager to write programs in languages new to them. Part C of the course analyzes basic object-oriented concepts and contrasts them with those of other languages, so familiarity with Java or a closely related language (e.g., C#) may be helpful, but it is not necessary for the Part C assignments.

This course is based on a course designed for second- and third-year undergraduates: not a first computer science course, but not an advanced course either. So it certainly will not cover everything in the beautiful world of programming languages, but it is a solid introduction.

An introductory video and another reading discuss assumed background in some more detail.

Course Goals

Successful course participants will:

- Internalize an accurate understanding of what functional and object-oriented programs mean
- Develop the skills necessary to learn new programming languages quickly
- Master specific language concepts such that they can recognize them in strange guises
- Learn to evaluate the power and elegance of programming languages and their constructs
- Attain reasonable proficiency in the ML, Racket, and Ruby languages --- and, as a by-product, become more proficient in languages they already know

Advice

We hope you enjoy the course and learn much from it. The course is not designed to be easy because this is an opportunity to make available to you a thorough experience at the core of computer science. A few pieces of advice apply specifically to this course:

- Give yourself time to get used to new languages and programming environments. It can be disorienting to learn a new tool, encounter unfamiliar error messages, etc. It can get frustrating if your only goal is to finish the homework as quickly as possible. Indeed, this strategy will probably be counterproductive: it will take you longer to finish the homework. Instead, enjoy "playing around" with examples and small programs as you start working on an assignment.
- Be particularly patient and determined during the first 2 weeks of the course. During this time, you will be doing programming that may feel totally unlike anything you have done before, which may be more frustrating the more prior experience you have. Stick with it. In our experience, once you get beyond one or two homeworks, the course approach can feel much more comfortable and rewarding, though we hope you find all of it enjoyable.
- If you approach the course by saying, "I will have fun learning to think in new ways," then you will do well. If you instead say, "I will try to fit everything I see into the way I already look at programming," then you may get frustrated. By the end, the course material will relate back to what you know, but be patient.
- The course material builds in sometimes-subtle ways. So do not be in a rush to get through early material even if it seems easy or you can guess what is going on without all the details. We are building an important foundation.
- Approach the homework by thinking about how it relates to the lecture material. The programs you will write are often a bit more sophisticated than those in the lectures, but you should be using the same concepts. Especially in the early assignments, you will not need to come up with a large number of tricks or learn much beyond what is in the lectures.
- For the exams, focus more on the concepts than just how to write programs. The exam questions will have some of each kind of question, but the more conceptual questions are covered less by the homeworks so you need to do more to study for them.
- Treat the peer assessments as a great opportunity to learn by reading other people's code: both to find good examples and to learn what might make your own code difficult for other people to understand. We truly believe peer assessment is more than just a scalable way to grade program style: it will help you learn.

Approximate List of Specific Course Topics

Part A:

- Syntax vs. semantics vs. idioms vs. libraries vs. tools
- ML basics (bindings, conditionals, records, functions)
- Recursive functions and recursive types
- Benefits of no mutation
- Algebraic datatypes, pattern matching

- Tail recursion
- Higher-order functions; closures
- Lexical scope
- Currying
- Syntactic sugar
- Equivalence and effects
- Parametric polymorphism and container types
- Type inference
- Abstract types and modules

Part B:

- Racket basics
- Dynamic vs. static typing
- Laziness, streams, and memoization
- Implementing languages, especially higher-order functions
- Macros
- Eval

Part C:

- Ruby basics
- Object-oriented programming is dynamic dispatch
- Pure object-orientation
- Implementing dynamic dispatch
- Multiple inheritance, interfaces, and mixins
- OOP vs. functional decomposition and extensibility
- Subtyping for records, functions, and objects
- Class-based subtyping
- Subtyping
- Subtyping vs. parametric polymorphism; bounded polymorphism