Goals

CS51 is a course that will teach you more rigorous methods of developing and analyzing software. You will learn to write code that is reliable, efficient, readable, testable, provable, maintainable... beautiful!

Outcomes

At the end of taking CS51, you will be able to define the following terms:

- Recursion, tail-calls
- Higher order functions: encapsulation, closures, environments
- Static typing, dynamic typing, type abstraction, polymorphism, sub-typing
- Asymptotic time and space complexity
- Modules, signatures, functors
- Objects, classes, interfaces, inheritance
- Unit testing

you will have mastered:

- Using and implementing Abstract Data Types
- Writing higher order functions
• Writing functional programs in ML
• Reasoning about the execution models of ML and Java programs
• Algorithmic complexity and design

you will have some experience with:

• Designing your own abstractions
  ◦ Gathering requirements
  ◦ Decomposing systems into recognizable patterns
  ◦ Mapping abstractions onto available libraries
• Applying a more systematic approach to solving CS problems:
  ◦ Using types, modules, and classes for modeling
  ◦ Functional and logical specification
  ◦ Designing interfaces
  ◦ Algorithm and data structure selection and design
  ◦ Contracts
  ◦ Testing strategies
• Object oriented design:
  ◦ Classes
  ◦ Subtyping
  ◦ Inheritance
  ◦ Design patterns: visitor, model-view-controller, factory.
  ◦ Testing strategies: unit, end-to-end, corner cases.
• Writing Java programs with multiple interacting classes
• Reviewing other people's code, and having your code reviewed by others.

and you will have been exposed to:

• Concurrency principles
  ◦ How to make an Abstract Data Type work in a concurrent environment
  ◦ General concurrency primitives for creating new threads of execution, exclusion, signaling, and message passing
• Using stream abstractions in programs
• Using induction and algebraic reasoning to formalize program behavior
• How to combine state and functional programming
• Why minimizing mutable state is important for object-oriented programming
• Declarative programming (query processing)

Who should take this course
This course is aimed at a broad audience:
• Students of computer science.
• Students of other quantitative subjects (physics, economics, biology,...).
• Students seeking an intellectually interesting elective.

Tentative Outline of Lectures

ML
1/26 Course overview
1/28  Ocaml and program evaluation
2/2   Recursive datatypes: integers and lists
2/4   Higher order functions: map, fold and the map-reduce paradigm
2/9   Polymorphism and parameterized types
2/11  Scope, evaluation and the substitution model
2/16  The substitution model of evaluation--mini ML interpreter
2/18  Modules and data abstraction (structures and signatures)
2/23  Data Abstraction, Modularity and Invariants--basic data structs--stacks, queues, sets,
2/25  More data abstraction -- finite maps, graph, bignums, fractions
3/2   Big-O and analysis: Example--functional queues
3/4   RB trees, state
3/9   More state, streams
3/11  **Midterm**
     SPRING BREAK!  Have fun!

**Java**
3/23  Introduction to Java
3/25  Subtyping, inheritance
3/30  Design patterns
4/1   Design patterns
4/6   GUIs as an example of OO design patterns
4/8   Software engineering 101: contracts, unit testing.
4/13  Contrasting functional and OO design
4/15  Concurrency
4/20  Concurrency
4/22  Concurrency
4/27  Wrapup