Plan on:

• Writing some code uScheme and ML code
• Reasoning about code (uScheme or ML)
• Working with semantics
Recursion and Induction

• Write recursive functions
  – Explain why function terminates

• Prove properties using induction
  – Explain why induction is well-founded

• Algebraic laws can lead naturally to recursive functions and inductive proofs
Understanding a language: Key Questions

1. What is the abstract syntax?
2. What are the values?
3. What are the environments?
4. How does evaluation happen?
5. What is the initial basis?
6. What are the types?
First-class functions

- What they are
- How to use them effectively
- Lambdas create anonymous functions
- Closures are run-time representation of functions; they capture the environment at closure-definition time
- Continuations capture the rest of the computation
Local bindings

- Various forms: `let`, `let*`, and `letrec`
- What is `let` used for?
- How do the various forms differ?
Data structures and associated operations

- **S-expressions** ((), null?, cons, car, cdr, ...)
- Lists
- Tuples
Cost Models and Optimizations

- How many cons cells?
- How many activation records?
- Method of accumulating parameters
- Tail calls
Operational semantics

- Precisely describe meaning of programs
  - What value does a program evaluate to?
  - What side effects are caused in the process?
- Written using inference rules
- Judgement forms capture all relevant information
- Environments track information about variables
- Stores map locations to values
ML

• Datatypes
  – Declarations introduce type constructor & data constructors
  – Datatypes can be recursive
  – Type variables allow polymorphic data structures
ML Pattern Matching

- Deconstruct values: datatypes, lists, tuples, …
- Bind variables
- Appear in function definitions, case expressions, and let bindings
ML Exceptions

• Declarating, raising, handling