Where have you been?

- Four major topics (and a language)
  1. Learning a language for expressing computations – Python
  2. Learning about the process of writing and debugging a program – Be systematic
  3. Learning to estimate computational complexity
  4. Learning about the process of moving from a problem statement to a computational formulation of a method for solving the problem – Use abstraction
  5. Learning a basic set of recipes – Algorithms

Why Python?

- Relatively easy to learn and use
  - Simple syntax
  - Interpretive, which makes debugging easier
  - Don’t have to worry about managing memory
- Modern
  - Supports currently stylish mode of programming, object-oriented
- Increasingly popular
  - Used in an increasing number of subjects at MIT and elsewhere
  - Increasing use in industry
  - Large and ever growing set of libraries

Writing, testing, and debugging programs

- Take it a step at time
  - Understand problem
  - Think about overall structure and algorithms independently of expression in programming language
  - Break into small parts
  - Identify useful abstractions (data and functional)
  - Code and unit test a part at a time
  - First functionality, then efficiency
  - Start with pseudo code
- Be systematic
  - When debugging, think scientific method
  - Ask yourself why program did what it did, not why it didn't do what you wanted it to do.

Estimating complexity

- Big O notation
  - Orders of growth
  - Exponential, Polynomial, Linear, Log, Constant
- Recognizing common patterns of computation
- Learning to map problems into templates of solutions
  - Bisection search, tree search,
- Some problems inherently expensive to solve

From problem statement to computation

- Break the problem into a series of smaller problems
- Try and relate problem to a problem you or somebody else have already solved
  - E.g., can it be viewed as a knapsack problem? As an optimization problem?
- Think about what kind of output you might like to see
- Think about how to approximate solutions
  - Solve a simpler problem
  - Find a series of solutions that approaches (but may never reach) a perfect answer

Algorithms

- Kinds of Algorithms
  - Exhaustive enumeration, Guess and check, Successive approximation, Greedy algorithms, Divide and conquer, Decision Trees
- Specific algorithms
  - E.g., Binary search, Merge sort, DFS, BFS