Wrapping it all up

- Where have you been?
  - What are the key topics from this course?
  - What are the key lessons to take from this course?
- Where are you headed?
  - How might you use the knowledge you have gained?

What do computer scientists do?

- They think computationally
  - Abstractions, algorithms, automated execution
- Computational thinking will be a fundamental skill used by everyone in the world by the middle of the 21st Century
- Just like the three r's: reading, 'riting, and 'rithmetic
  - Ubiquitous computing and computers will enable the spread of computational thinking as a fundamental skill for every well educated person

I  6.00.1.x

Computational Thinking: the Process

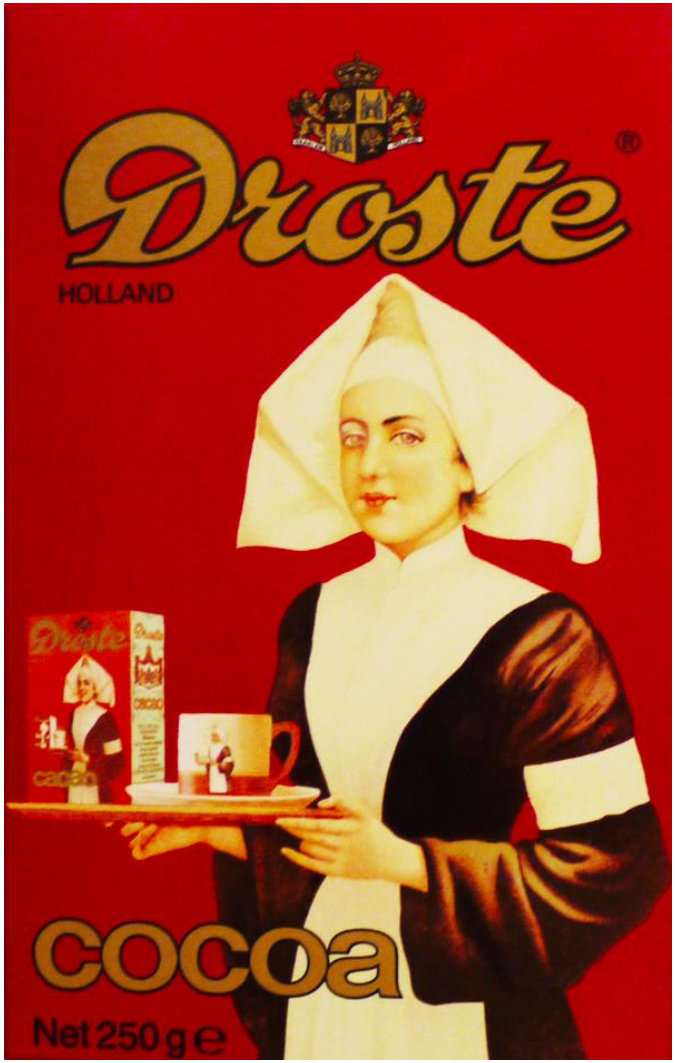
- Identify or invent useful abstractions
  - Suppressing details, formulating interfaces
- Formulate solution to a problem as a computational experiment using abstractions
- Design and construct a sufficiently efficient implementation of experiment
- Validate experimental setup (i.e., debug it)
- Run experiment
- Evaluate results of experiment
- Repeat as needed

The three A's of computational thinking

- Abstraction
  - Choosing the right abstractions
  - Operating in terms of multiple layers of abstraction simultaneously
  - Defining the relationships the between layers
- Automation
  - Think in terms of mechanizing our abstractions
  - Mechanization is possible
    - Because we have precise and exacting notations and models
    - There is some "machine" that can interpret our notations
- Algorithms
  - Language for describing automated processes
  - Also allows abstraction of details

Examples of computational thinking

- How difficult is this problem and how best can I solve it?
  - Theoretical computer science gives precise meaning to these and related questions and their answers
- Thinking recursively
  - Reformulating a seemingly difficult problem into one which we know how to solve
  - Reduction, embedding, transformation, simulation


Droste
HOLLAND
Droste
cacao
cocoa
Net 250 g e

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

Where are you headed?

- Many of you have worked very hard
  - The staff greatly appreciate it
- Only you know your return on investment
  - Take a look at early problem sets
  - Think about what you would be willing tackle now
- Remember that you can write programs to get answers to problems
  - Computational thinking is now a new arrow in your quiver

Where are you headed?

- There are other CS courses for which you are now prepared
  - Second part of 6.00 – optimization, modeling, simulation
  - Introductory algorithms and data structures
  - Introduction to artificial intelligence
  - Software engineering
  - Computer architecture