CS111 Fall 2015 Exam 1 Overview

Note: this document is still under development, and may change.

Exam 1 is a 70-minute in-class exam that will be held on Friday, October 9, 2015. At the end of this document is a list of topics for the exam.

In the exam you could be asked to:

1. Read Python programs and explain what they do. What values do they print or return?
2. Modify existing Python programs.
3. Write Python programs that satisfy a specification.

The exam is open notes in the sense that you can bring with you any **printed and handwritten materials**, such as your written notes, printouts of slides and web pages you think are important, and books. We strongly recommend against printing large numbers of pages, since most students don't have time during the exam to consult them. To prepare for the exam, it's better for you to write a few pages of your own notes of what you think is important and might forget.

You are not allowed to use computers or calculators during the exam. If you have a smartphone, you may only use it to tell time and nothing else. In particular, you are not allowed to browse the web during the exam or use a Python interpreter during the exam.

Here are some things we encourage you to do to prepare for the exam:

- Attend Brenna and Cece's SI exam review sessions.
- Review the Exam 1s from Fall, 2014 and Spring, 2015. Note that because of differences in coverage of material between semester, certain topics you are expected to know for the Fall, 2015 exam may not be covered in these older exams (particularly memory diagrams, iteration tables, nested loops, and list comprehensions).
- Review the posted solutions for all psets. Often, the posted solutions may show you how to solve a problem in a better way than you did on your pset.
- Review all course lecture slides and lab materials. Write down anything you're confused about and ask an instructor/SI/tutor.

Concepts for Exam 1

- Python syntax: expressions vs. statements vs. declarations.
  - Expressions are program fragments that denote values. They may be arbitrarily complicated, and are evaluated left-to-right, from the inside out.
  - Statements are program fragments that perform actions. They are composed in sequences that are executed from top down.
  - Declarations introduce variables and function definitions.
- Variables and assignment.
  - To evaluate the assignment expression <var> = <exp>, first evaluate <exp> to a value V.

            - If <var> does not yet exist, create a box labeled <var> in the current scope, and fill it with the value V.
            - If <var> already exists, change the contents of the box labeled <var> to the value V.
            - In the context of assignments involving list slots, <var> can be replaced by an expression denoting a list slot. For example `myList[3] = 17` or `myList[i+1][j-1] = myList[i][j] + myList[i+1][j]`
        - To evaluate the variable reference expression <var>, return the contents of the variable box labeled <var> in the current scope.
- Functions:
    - understanding the difference between function definition and function invocation.
    - function parameters
        - The name of parameters does not matter as long as they are used consistently.
        - In a function invocation frame, each parameter denotes a local variable initialized to the argument value.
    - understanding the difference between **return** and **print**.
- Scope:
    - the locality of parameters and other local variables assigned within a function body.
    - the **global** declaration for declaring a variable global within a function.
- Conditional statements: if statements with optional elif and else clauses.
- Sequences:
    - lists are mutable sequences of values. You can both change indexed slots and add and remove indexed slots from a list.
        - List comprehensions are a compact notation for mapping, filter, and appending without writing explicit loops.
    - tuples are immutable sequences of values.
    - strings are immutable sequences of characters.
- Iteration:
    - Iterations are repeated updates to state variables, as expressed in iteration tables via iteration rules
    - Iterations are expressed in Python using loops:
        - **`while`** loops
        - **`for`** loops range over lists and are just **`while`** loops in disguise
        - loop gotchas:
            - premature **`return`** from list
            - updates to state variables in wrong order
        - Sometimes you **want** to return early from a list via **`return`** or **`break`**
        - It is common to nest one loop within another
- Understanding how to use functions and objects from their contracts.
    - cs1graphics has lots of objects with contracts that you've used.
    - you've also seen contracts for operations on sequences (lists, tuples, strings).
- Memory diagrams:

    - these show the state of a program involving variables and values, including mutable values like lists and objects.
    - they are especially important for understanding:
        - assignment: changing the value stored in a variable, list slot, or object instance variable.
        - adding or removing slots in a list.
        - aliasing: the same mutable value can be accessed by multiple paths.
- Abstraction:
    - cs1graphics Layers group `DrawableObjects`
    - Python functions abstract over behavior.
- Problem solving strategies:
    - Divide/conquer/glue
    - Designing iterations (loops) with iteration tables and iteration rules