CS 1110, Spring 2017: About the final exam

Parting thoughts: back to our first lecture!

> *Like philosophy, computing qua computing is worth teaching less for the subject matter itself and more for the habits of mind that studying it encourages.*
>
> ...
>
> For some, at least, it could be the start of a life-long love affair.
>
> …
>
> That, for me, sums up the seductive intellectual core of computers and computer programming: here is a magic black box. You can tell it to do whatever you want, within a certain set of rules, and it will do it; within the confines of the box you are more or less God, your powers limited only by your imagination. But the price of that power is strict discipline: you have to *really know* what you want, and you have to be able to express it clearly in a formal, structured way that leaves no room for the fuzzy thinking and ambiguity found everywhere else in life...
> **The sense of freedom on offer - the ability to make the machine dance to any tune you care to play - is thrilling.**
>
> Excerpts from *The Economist* blog, August 2010, emphasis added

Time: 9am-11:30am, Thursday May 18th
Location: Barton Hall: the central and east portions of the track/gym floor.

About Barton: where to sit, what to watch for, etc.

- East = towards Teagle.
- West ("the wrong end", some other class's exam *might* be there, so walk past them to get to CS1110) = towards the Statler. If you enter from the West (Statler) end, go up the stairs to the second floor to get to the gym floor.

The setup will look like this picture[1]. Sit two to a table, one person at each end.[2]

Fill in the tables toward the East door first: we'd like students to be as geographically close as possible. The acoustics are terrible and there's no way to post announcements, so we have to yell. It makes it easier for everyone in CS1110 to hear, easier for CS1110ers to *not* hear announcements for the other exam, and faster for us to get to people who have questions, if we can have people close together.

[1] Taken by Jason Koski, Cornell University Photography
[2] Some of the students in the photo are sitting in the middle of a table. Please don't do this unless we actually run out of room.

There's no easily readable clock in Barton. We will attempt (as we have successfully pulled off in previous years) to commandeer one of the Track-and-Field indicator boards, and use it to manually indicate the time. Look for something that resembles this.[3]

Bring writing/erasing utensils and your Cornell ID. The exam is closed book, "closed notes", no electronic or external aids, etc**. Place your ID face up on the table next to you, and also bring it with you when you turn your exam in.**

Topic coverage: *All* material from all course assignments, labs, and lectures *except* for `try-except` statements, since we exempted you from that concept on prelim 1.

You should not be surprised to see questions involving any of the following: string, list, and dictionary processing; testing and debugging; variables, objects, classes (including subclasses and inheritance), name resolution (including the effect of various types of `import` statements); frames and the call stack; recursion; for- and while-loops; sequence and sorting algorithms; loop invariants (you should be able to write code that is effectively based on an invariant, but you will *not* be responsible for coming up with your own invariants during this timed exam; you do not need to provide loop invariants unless asked to do so.)

But, since "all" means "all", the above paragraph is not necessarily an exhaustive topic list.

We will provide function/method references as in prior exams, but will not be able to specify ahead of time what will be on it.

For practice.
In general, Fall class and sub-class questions have included sub-problems involving implementing getters and setters and asserting preconditions. We will not have such sub-problems, but other parts of the class and sub-class questions are fair game.

In general, Spring 2015 and Spring 2016 use different variable naming conventions from what we use: we would reserve capital letters for class names, and use more evocative variable names.

"`if __name__ == '__main__'`" means that the following indented code is executed only when the file is run as a script, *not* when imported;

Fall questions for which one-frame-drawn-per-line notation is used would need to be converted to our one-frame-per-function notation.

1. This year's assignments, labs, lecture problems, lecture demo code (try reproducing what happened in class)
2. Review session materials from over the years
    1. 2017 spring exams page has the review-session materials from this semester. (The final-exam review-session materials are forthcoming.)
    2. 2016 fall review session slides
        i. Not fair game: exceptions and try-except. Questions involving graphics (but the "carpet" recursion question is mostly OK).
    3. 2016 spring review sessions, listed on that semester's lectures page: look for the words "review".

[3] Wouldn't it have been nice if that image had said "1110" instead of "1111"?

i. Not fair game: practice prelim 1, Q3 (we haven't covered random walks), Q5 (graphics)

4. 2014 spring prelim 2 review (link is to a version updated in spring 2017)
5. 2014 spring prelim 1 review , which is much like the Fall 2016 prelim 1 review
   i. Notes on the questions on slide 5 about creating netids: In the solution, the "backquotes", as you'll recall from lecture, cause a call to the `__repr__` method; it's fine to also use `str()`. You definitely need to be able to use find/index and string slicing for the final, but for the record, here's an alternate solution that uses `split`:

```
def make_netid(name, n):
    components = name.lower().split()
    fletter = components[0][0]  # first letters
    lletters = components[len(components) - 1][0]

    # glue middle initial in front of lletters if there is one
    if len(components) == 3:
        lletters = components[1][0] + lletters

    return fletter + lletters + str(n)
```

   ii. Note on the testing question on slide 22: "`cunittest2`" is our "`cornelltest`"

3. Previous exams.
   Not fair game (note that some problems that weren't fair game earlier have *become* fair game at this point in the course):
   1. Prelim 1:
      i. 2016 Fall: ignore 3b (too lecture-dependent)
      6(a), assume that `math` has been imported
      ii. 2016 Spring: 6 (we didn't do as much with the random module)
      iii. 2015 Fall: 4(a) – solutions have typos. 4(c) (asserts)
      iv. 2015 Spring: 4 (too assignment dependent)

      For 1(b), the question is better stated as, "under what conditions on s will s and u print out as the same string s, where contains some arbitrary, unknown string?" (we didn't formally cover `raw_input`)

      3's solution should be:

      1 2

      1 1 3

      3 2

      B

      v. 2014 Fall: 2(a) (memorizing which types are "basic" is something we would not ask for) 4(a) (asserts)

6 involves quite a bit of geometric reasoning as well as coding ability

vi. 2013 Spring:

6: change cunittest2 to cornelltest

2. Prelim 2:
   i. 2016 Fall:

      4 has an error in the solutions: if statement should have "!=", not "=="

   ii. 2016 Spring:

      We would explain that estimating the probability would just mean counting the number of times the dice came up with exactly two having the same value, divided by the number of rolls.

   iii. 2015 Fall: 3(a) (is vs ==), 3(d) (exceptions)
   iv. 2015 Spring: 4(d) (graphics), 6(b) (try/except) , 6(c) (timing)
   v. 2013 Fall: 6(b) (exception types)
3. Final:
   i. 2016 Fall: 3 (graphics content we didn't cover), 4b (we haven't talked about types of exceptions)
   ii. 2015 Spring: 7(a) "Lady Macbeth" is a full, independent name[4] ; pretend the example says "Gruoch" instead of "Lady Macbeth"
   iii. 2015 Fall: 1(a) (we have not emphasized "is" vs. "=="); 1(c) (we wouldn't ask for memorization of sorting algorithm names or runtimes)

      For 3 we would give you the information on what parameters GImage __init__ has.

   iv. 2015 Spring: 1(c) (we would not ask about expected numbers or percentages); 7 (c) (we didn't cover numpy), 9 (too assignment-dependent)

      10(c) is fine but would need to be converted to our notation.

   v. 2014 Fall: 1(b) (we have not emphasized "is" vs. "==");
   vi. 2014 Spring: 5 (we cannot post the code on the accompanying handout due to standing agreements with other instructors. But the *style* of this question is fair game)
   vii. 2013 Fall: 3(b) (try-except); 8(d) (don't have to memorize names of sorts of variables, but *should* know what local, global and class variables, parameters, and (instance attributes are!)

6: alternate solution for same invariant:

[4] An unfortunate ambiguity in that question that year: people thought the method was supposed to add the word "Lady" to the person's name.

```
def f2013q6v3(b, k):
    """version by Lillian Lee using the 'typical' initialization that a
    student asked about on Piazza."""
    p = k
    h = k
    # inv:
    # b[h+1..k] is original b[p+1..k] with no duplicates
    # b[p+1..h] is unchanged from the origin llist w/ values in b[h+1..k]
    # b[0..p] is unchanged from the original list

    while p+1 != 0:
        # Is b[p] in b[h+1..k]?  If it were in, would it be at h+1?
        # Note that if h==k, then there is nothing in b[h+1..k]
            # Also note that you have to be careful about whether b[p+1] exists.
        if h==k or b[p] != b[p+1]:
            b[h] = b[p]
            h -= 1
        p -= 1
```

7(b): solution typo: occurrences of variable i should be replaced by n, and j by m.

viii. 2013 Spring:

For 2 we would tell you about sets and the union method for sets. For 3, ignore the “try/except” part of the solution, i.e., assume the precondition would have been given this semester.