CS 1110, Spring 2018: 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

Exam logistics

Time: 9am-11:30am, Thursday May 17th
Location: Barton Hall: the central and east portions of the track/gym floor.

For makeup exam requests and SDS accommodations, CS1110 administrative assistant Ms. Jenna Edwards (JLS478) will be contacting students directly. If you haven't heard anything by Mon May 14th noon, please email Ms. Edwards to check in.

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.

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.

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.

On the exam, you may not use Python we have not introduced in class (lectures, assignments, labs, readings).

The exam is to test your understanding of what we have covered in class.

On the exam, if you are asked to solve a problem a certain way, answers that use a different approach may well receive no credit.

In particular, if we say you must make effective use of recursion, the question is to test your understanding of recursion, so a solution that is *essentially* just a for-loop or while-loop may well receive a score of 0. However, it's OK to use a for-loop within a recursive solution, as in A4 and in prelim 2.

---

[1] Taken by Jason Koski, Cornell University Photography.
http://news.cornell.edu/sites/chronicle.cornell/files/0468_13_045.jpg

[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.

[3] Wouldn't it have been nice if that image had said "1110" instead of "1111"? http://www.ucsspirit.com/track-field/product-detail.cfm/category/Measuring-Devices-Indicators/subcategory/Indicator-Boards/product/4-Digit-Board

Similarly, if we say you must use a loop, then map() or recursion is not allowed as the *core* of your solution, and so on.

Our mechanisms to help you prepare

We have posted many prior CS1110 exams and their solutions to the Exams webpage, http://www.cs.cornell.edu/courses/cs1110/2018sp/exams/index.php; more about these below. Code examples are posted for most lectures to exemplify the corresponding topics; see the course lectures page, http://www.cs.cornell.edu/courses/cs1110/2018sp/lectures/index.php

The staff will be holding topical review and problem-solving sessions on Saturday May 12th (lists & sequences; loop invariants and sequence algorithms) and Sunday May 13th (call frames, classes), both in Hollister B14; watch the course main webpage for precise times. We will try to arrange for Videnote-ing of those sessions.

The full menu of drop-in office/consulting hours starting after the end of classes will be kept updated here: http://www.cs.cornell.edu/courses/cs1110/2018sp/about/staff.php. Individual appointments may also be available; see that webpage for scheduling instructions.

Solutions to A5 will be posted on Thursday May 10th. (Unfortunately, because of the schedules of the staff, we cannot guarantee that the grades for A5 will be available before the final exam.)

Recommendations for preparing, in no particular order

1. Go through the lecture slides, making sure you understand each example.
2. Be able to do the assignments and labs cold.[4]
3. Do relevant problems from previous exams, as noted below.
    a. While you may or may not want to start studying by answering questions directly on a computer, by the time the exam draws nigh, you want to be comfortable answering coding questions on paper, since doing so is a way to demonstrate true fluency.[5]
    b. **Warning:** it is difficult for students to recognize whether their answers are actually similar to or are actually distant from solutions we would accept as correct. So, rather than saying "oh, my solution looks about the same", we suggest you try out your answers by coding them up in Python where possible, and seeing what happens on test instances that the exam problems typically provide.
    c. **Strategies for answering coding questions:**
        i. When asked to write a function body, always first read the specifications carefully: what are you supposed to return? Are you supposed to alter any lists or objects? What are the preconditions? *if you aren't sure you understand a specification, ask.*

[4] But note we *didn't* necessarily expect you to find them straightforward at the time they were assigned.
[5] Many coding interviews at companies are conducted at a whiteboard.

ii. After you write your answer, double-check that it gives the right answers on the test cases --- any we give you, plus any you think of. Also double check that what your code returns on those test cases satisfies the specification.[6]

iii. Comment your code if you're doing anything unexpected. But don't overly comment - you don't have that much time.

iv. Use variable names that make sense, so we have some idea of your intent.

v. If there's a portion of the problem you can't do and a part you can, you can try for partial credit by having a comment like

```
# I don't know how to do <x>, but assume that variable start
# contains ... <whatever it is you needed>"
```

That way you can use variable start in the part of the code you can do.

4. Check out the code examples that are posted along with the lecture handouts. See that you understand what they are doing, and perhaps even see if you can reproduce them.
5. Buddy up: at office hours, lab, or via Piazza, try to find a study partner who would be well-matched with you, and try solving problems together.

Notes on questions from prior exams and review materials

In general

In Spring 2018, we did not cover try/except or exceptions.

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.

Fall questions for which one-frame-drawn-per-line notation is used would need to be converted to our one-frame-per-function notation. In Spring 2018, we said that "else" statements should affect the program (line) counter.

In general, Fall class and sub-class questions have included sub-problems involving implementing getters and setters, mutable vs. immutable attributes, and asserting preconditions. We will not have such sub-problems, but other parts of the class and sub-class questions are fair game. When you have to draw class folders, we will tell you whether we would want you to include all methods inside the class folders, and if so, whether you need to include all the method parameters in the signatures of the methods, and whether you need to include the superclass in parentheses in the tab of a class folder.

Where you see lines of the form "if __name__ == '__main__':", think of them as indicating that the indented body underneath it should be executed for doing the problem.

Before Fall 2017, the course was taught in Python 2. Here are some differences this makes in terms of the relevance of previous prelims:

1. questions regarding division (/) need to be rephrased.

---

[6] It seems to be human nature that when writing code, we focus on what the code *does* rather than what the code was *supposed* to do. This is one reason we so strongly recommend writing test cases before writing the body of a function.

2. Python 2's print didn't require parentheses and allowed you to give multiple items of various types separated by commas (which would print as spaces).
3. In some cases, instances of range() in a Python 2 for-loop header might need to be replaced with list(range()), and similarly for map() and filter().
4. In Python 3, one can omit "object" from inside the parentheses in the header of class definitions and the class will still be a subclass of object.
5. The usage of super() in Python 2 and Python 3 differ slightly.

You may notice that many solutions check whether something is None by " if x is None:" rather than "x==None:". We haven't discussed this in Spring 2018, but the former is preferred.

Previous finals

2017 Fall:

- Q1(a) explanation typo: the second line makes b be [[4, 5, 6], [7, 8. 9]]. Here is a Python Tutor link: https://goo.gl/k4L7Ct
- Q1(d): one could alternately phrase this question as, "describe the four steps that happen when you make a call of the form <classname>(<arguments>).
- Skip Q3 (too dependent on that semester's graphical A7)
- Q4: to do this problem, you need only assume that each Alien object has a y attribute, and that to "be in the bottom row" means "to have a value for the y attribute that is the minimum among all the aliens in the list aliens".
- Q5: although in Spring 2018 we haven't had an assignment using Turtles (we *did* have a lecture demo using them), you should be able to do this problem, or at least understand the solution.
- Q6
  - (a): assume that for a non-letter character, "swapping the case" leaves the character alone.
  - (b) skip (try/except)
- Q7:
  - if mylist is a length-1 list, then mylist[1:0] returns the empty list.
  - Using the Spring 2018 conventions, the line number 4 should occur in the frame for take.
- Q8(c):
  - It would be reasonable to impose the precondition that k >= h -1 (we allow empty lists) and that h and k are valid indices into b.
  - Here is a Python Tutor link where you can test your own solutions (BUT **Python tutor cannot tell if you are maintaining the invariant!**): https://goo.gl/VGR8tu
  - alternate solution:

```
i = k # for the invariant, OK to say that either b[k+1..k] (i.e., the empty list)
      # is sorted as in the given solution, or that b[k..k] is sorted (i.e., a 1-
element
      # list is already sorted

# inv: b[h..i-1] ???, b[i..k] sorted
while i > h:
          # push b[j] rightwards; inv: b[i-1..j] are sorted
          j = i – 1
          while j < k and b[j] > b[j+1]: # must check for j < k first.
                    b[j], b[j+1] = b[j+1], b[j] # perform the swap
                    j += 1
          i-= 1
```

2017 Spring

- Q1 solutions: some lines have been cut off. Here is a suite of alternative solutions:

```
def putSideBySide(two_line_strings):
    line1 = ""
    line2 = ""
    first = True
    for a_string in two_line_strings:
        parts = a_string.split("\n")
        if first:
            first = False
        else:
            line1 += " "
            line2 += " "
        line1 += parts[0]
        line2 += parts[1]
    return line1 + "\n" + line2

def solAP1(two_line_strings):
    top_str = ''
    bottom_str = ''
    for x in two_line_strings:
        top_str += x[0:2] + ' '
        bottom_str += x[3:5] + ' '
    return top_str[:-1] + '\n' + bottom_str[:-1]

def solAP2(two_line_strings):
    top_str = two_line_strings[0][0:2]
    bottom_str = two_line_strings[0][3:5]
    for x in two_line_strings[1:]:
        top_str += ' ' + x[0:2]
        bottom_str += ' ' + x[3:5]
    return top_str + '\n' + bottom_str
```

```
def solLL2(two_line_strings):
    converted = list(map(splitem, two_line_strings))

    # top is output[0], bottom is output[1]
    output = [converted[0][0], converted[0][1]]

    for i in range(1, len(converted)):
        for j in range(2):
            output[j] += (" " + converted[i][j])
    return output[0] + "\n" + output[1]


# helper for solLL2; to allow use of map
def splitem(tls):
    """Version of split() that takes the string to split as an argument"""
    return tls.split()


def solLL1(two_line_strings):
    top = ""
    bottom =""
    for tls in two_line_strings:
        nindex = tls.find("\n")
        front = tls[:nindex]
        back = tls[nindex+1:]
        top += (front + " ")
        bottom += ( back + " ")
    return top.strip() + "\n" + bottom.strip()
```

- Q2:
  - Here is a Python Tutor link (to a Python 3 version): https://goo.gl/vgnA4v
  - The solution "Delete print at 27; add print at line 16" seems dangerous, in that if move() is called but self.topDisk() somehow happened to be None, there would be a printout, but nothing would be appended.
- Q4(a):
  - Although without having down Spring 2017's A4, this question might not make intuitive sense, it is still doable just from reading the specifications carefully.
  - Alternative, and, in Python 3, preferred first line: `super().__init__(in1, in2, one_won)`
- Q4(b): missing line from solution: outside of the for-loop, there should be "`return output`"

2016 Fall:

- skip 2(b) (we didn't cover this terminology), 3 (graphics content we didn't cover), 4b (we haven't talked about types of exceptions)

- Q8 solution typo: in the 5th "animation cell" (at the top of page 11), the list with identifier id1 should only contain a single element, and that element, at index 0, should be 0. (Thanks to a student for catching this!)

2016 Spring:

- skip 1(b) (dependent on Python 2 definition of /).
- Solution to 5: for Python 3, first line should be "y = x//100"
- Solution to 6b: change if-statement to "if P.Dist(Q) <=1:"
- 7(a) "Lady Macbeth" is a full, independent name[7] ; pretend the example says "Gruoch" instead of "Lady Macbeth"

2015 Fall: Skip 1(a) (we have not emphasized "is" vs. "=="); 1(c) (we wouldn't ask for memorization of sorting algorithm names or runtimes), 3 (uses classes we haven't used)

2015 Spring:

- Skip 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.

2014 Fall

- Skip Q2b (we have not emphasized "is" vs. "=="), 2c (too dependent on Python 2), 2d (we did not discuss the order of optional parameters in function headers), 3(b) (2018 Spring did not cover try/except)
- Q4: the specifications could be more clear that one is altering *this* GPanel's _contents attribute. Change first assignment statement in __init__ to "super().__init__(x, y, w, h, color)" (for 2018 spring, we'll assume non-keyword parameters). Skip the draw() method.
- Q6: you should get the gist of what is happening, but we would not be testing you on getting the positions of the rectangles exactly correct (line b = GRectangle(…)

2014 Spring: Skip Q5 (we cannot post the code on the accompanying handout due to standing agreements with other instructors.)

2013 Fall:

- Skip 3(b) (try-except); 7(a) (we would not ask you to memorize a loop invariant) 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:

[7] 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):

    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.

2013 Spring

- Q2: in 2018 Spring, we would tell you about sets and the union method for sets.
- Q3, ignore the "try/except" part of the solution, i.e., assume the precondition would have been given this semester.
- Q4: change of super in Python 3: super().__init__("Tory", record)

Previous prelims: questions that weren't fair game earlier in the semester but **are now**.

See also the Spring 2018 Prelim 1 study guide,
http://www.cs.cornell.edu/courses/cs1110/2018sp/exams/prelim1/2018-spring-prelim1-studyguide.pdf
And the Spring 2018 Prelim 2 study guide,
http://www.cs.cornell.edu/courses/cs1110/2018sp/exams/prelim2/2018-spring-prelim2-studyguide.pdf

Prelim 1s:

- 2016 spring Q3 – while-loops
- 2015 Spring Q2(b) – for-loops
- 2014 Fall 2(e) – dictionaries

Prelim 2s:

- 2017 spring Q5 – invariants
- 2014 spring Q3 – while-loops; Q4 – while-loops; Q6
- 2013 spring Q3 – loop invariants; Q4 – while-loops (although also recursion and for-loops!)

For 2014 Q6 partition question, here is a longer explanation of the given solution, taken from a Piazza post of Prof. Lee in Spring 2017.

Have you had a chance to read the reading "Loop invariants" posted with the April 18th [2017] lecture on loop invariants?

First, I'm going to run through an example from there, of how you would write code given an invariant; then go to what you would do to do a question like #6 Spring 2014 Prelim 2.

From Section 2 of the invariants reading: suppose you're given b, h, and k, where b is a list and 0 <= h <= k < len(b), and you must write a loop to find the location of the minimum item in the list between h and k inclusive.

Invariant in section 2.1 : b[i] is the minimum of b[h..t] So, t marks the last thing you've checked, h marks the first thing you've checked, and i is the location of the minimum in b[h..t].

So, how to start? Might as well check the item at b[h], since maybe that's the minimum location. That suggests the initialization:

```
t = h
i = h # since that's the location of the minimum of b[h..h]
```

When do we still have work to do? While we haven't checked everything in b[h..k]. So if t, the last thing we've checked, is less than k, we still have work to do. This suggests the following condition for the while-loop:

while t < k: # note that we are guaranteed in this problem that k < len(b), so we don't have to check for falling off.

How do we make progress? We want to know whether the next item in the list, the item b[t+1] which we haven't checked yet, is smaller than b[i] or not. If it is, then we change i (because we found a new minimum); if it isn't, we don't change i.

```
if b[t+1] < b[i]:
    i = t+1
# don't update if b[i] is smaller than b[t+1]
```

Now we've checked the position t+1, that is, the last thing we;ve checked is position t+1, so we update our "last-checked" variable inside the loop (but outside the if)

```
t = t+1
```

OK. Now for the problem on the previous exam (and see solutions on the previous exam):

The "partition" function[']s goal is as specified in the loop invariants reading, section 4 (Spring 2014 students would have covered this in class already). In terms of the exam question, the goal is given as the post-condition: given a list b and a value z, make it so that b[0..j-1] <= z and b[j ..] > z

The invariant given is

```
b[0..i-1] <= z and b[j..] > z
```

The meaning of this is,

(i-1) is the position of the "last"/"rightmost" thing we know to be <=z; and everything before it is also <=z

j is the position of the "first"/"leftmost" thing we know to be > z, and everything to the right is also > z

So, first thing is we check: *does the code initialize the invariant correctly, i.e., is the start of the code correct?*

Well, at the beginning we don't know if anything is <=z or anything is greater than z.

So, we can do

```
i = 0       # b[0..i-1] in this case is b[0..-1], which by our notational convention means the empty list. So this makes sense.
j = len(b)  # b[j ..] in this case is b[len(b)..] which is b[len(b)..len(b)-1] in our notational convention, and so it's the empty list.
```

In both cases, we're saying that we don't know any values being <= z or > z.

But the code given says

j = len(b) -1

This doesn't match what we thought!

And it isn't true: we don't know whether b[j..], which in this case is b[len(b)-1 .. len(b)-1] is < z, because we haven't checked that item yet!

Hence, the initialization of j given in the code is incorrect.