CS 1110, Spring 2021: Final Exam Study Guide

Prepared Wednesday May 12, 2021 by Lillian Lee

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

Topic coverage

All material from all course assignments, labs, and lectures. Material after the second prelim is fair game, but will *not* be more heavily weighted.

Topics include but are not limited to: string, list, and dictionary processing; testing and debugging; variables, objects, classes (including subclasses and inheritance), if/elif/else; 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.

We will provide syntax and specification references for uncommon functions/methods, but will not be able to specify ahead of time what will be provided. You will be expected to already know without need of reference (non-exclusive list): operators like %, /, //, list and string slicing, `index()` method for strings and lists, string method `find()`, and that lists don't have a find() method, the `append()` method and + operator for lists, list and dictionary item access and operators like `in`; how to create lists, dictionaries; the `range()` function with one argument.

You should know how to write assert statements, but you do not need to assert preconditions on the exam unless we tell you to.

You will not be expected to draw class folders (since we never set up a formal notation), but you should understand diagrams that provide them. You should be able to draw call frames for methods. (Give the

method name as <class name>.<method name>).

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.

For instance, 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.

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.

Recommendations for preparing

1. Absolutely most important: (re)do (and run the testing code on) assignment/lab problems that you didn't do entirely on your own.
2. Second most important: Do relevant problems from previous exams, as noted below.
    a. Spring 2018 is a good choice of a first exam to try. As of May 14, 2021, we have provided skeletons/testing code for that final and one other; see https://www.cs.cornell.edu/courses/cs1110/2021sp/exams/#exam-archive
    If we are able to supply testing code for more exams, we will announce this on Ed Discussions.
    b. While you may or may not want to start studying by answering questions directly on a computer --- and we recommend that you do --- 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.[1]
    c. **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.
    d. **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 we didn't have to balance the in-person and online conditions, we would say: *If you aren't sure you understand a specification, ask.*)

[1] 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.[2]

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.

3. Be able to do the assignments and labs cold.[3]
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 2021, we did not cover try/except or exceptions or loop invariants.

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.

---

[2] 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.

[3] But note we *didn't* necessarily expect you to find them straightforward at the time they were assigned.

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.
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.

Previous finals

2019 Fall:

- **Skip 2(b)** (Spring 2021 did not cover try/except), **3** (too dependent on Fall 2019's A7)
- Q4:
    - Note that by the specification (the text), it's OK for upper-triangular matrices to have 0s above the diagonal, and for lower-triangular to have 0s below the diagonal. Having a 0 on the diagonal does not disqualify from being upper-triangular or lower-triangular or diagonal. *Exam-taking moral: don't over-generalize from examples. Make sure to read the specifications and text that's given in the question!*
    - Hint: a matrix is diagonal if it is both upper-triangular and lower-triangular
    - Hint: matrix[i][j] is in the "northeast" if j > i, and it is in the "southwest" if j < i
    - Hint: A way of rephrasing the key concepts is:
        - A matrix is NOT upper-triangular if it has a non-zero in the "southwest"
        - A matrix is NOT lower-triangular if it has a non-zero in the "northeast"
    - Alternate solution using while-loops:

```
    n = len(matrix[0])

    # Can we find evidence that matrix is not upper triangular?
    seems_upper = True # currently no evidence to the contrary
    irow = 1
    while irow < n and seems_upper:
        row = matrix[irow]
        icol = 0
        while icol < irow and seems_upper: # icol < irow is checking the southwest
            if row[icol] != 0:
                seems_upper = False
            icol += 1
        irow += 1

    # Can we find evidence that matrix is not lower triangular?
    seems_lower = True # currently no evidence to the contrary
    irow = 0
    while irow < n-1 and seems_lower:
        row = matrix[irow]
        icol = irow+1 : # icol > irow is checking the northeast
        while icol < n  and seems_lower:
            if row[icol] != 0:
                seems_lower = False
            icol += 1
        irow += 1

    if seems_upper:
        return PURE_DIAGONAL if seems_lower else UPPER_TRIANGULAR
    if seems_lower:
        return LOWER_TRIANGULAR
    else:
        return NORMAL_MATRIX
```

Alternate solution due to T. Sarver, with an early-return inside a for-loop:

```
diag = -1

upper = True
lower = True

n = len(matrix) # We have an nxn matrix (credits to Aidan McNay and Professor Lee for this part)

for row_num in range(n):
    diag += 1 # Gives the index of the diagonal for this row.
    for entry_num in range(n): # entry_num is the "index" of that entry in the row.
        entry = matrix[row_num][entry_num]

        if entry != 0:

            if entry_num > diag: # There is a non-zero entry above the diagonal.
                # This means, the matrix cannot be a lower-triangular.
                lower = False
            elif entry_num < diag: # # There is a non-zero entry below the diagonal.
                # This means, the matrix cannot be a upper-triangular.
                upper = False

        if lower == False and upper == False: # Prof. Lee note: early return works here!
            return NORMAL_MATRIX

if lower == True and upper == True:
    return PURE_DIAGONAL
elif lower == True:
    return LOWER_TRIANGULAR
elif upper == True:
    return UPPER_TRIANGULAR
```

- Q5: Alternate solution that treats codes of length 4-7, inclusive, the same way:

```
if len(code) < 4:
    return ""
elif len(code) <= 7:
    return code[:4].upper()
else:
    return code[:4].upper() + '-' + couponify(code[4:])
```

- Q7: Skip Q7(a) (spring 2021 didn't worry about this terminology). Skip 7(d), which is related to material on loop invariants (not covered in spring 2021).
- Q8: skip (based on loop invariants, not covered in spring 2021)

2019 Spring:

- Q1(b): "class method" should be "instance method"
- Q1(c): class invariants are conditions expressed in the docstring for a class, such as "every Student is in the list of Students for every Course they are enrolled in".
- Q4 alternate solution:

```
class Student(Person):
    next_num = 1

   def __init__(self, first, last, parent):
        super().__init__(first, last, parent)
        self.netID = first[0].lower() + last[0].lower() +
str(Student.next_num)
        Student.next_num+=1
```

- Q5: spring 2021 didn't cover invariants, but you might well be able to see that writing code that respects the invariant makes it easy to write this while-loop --- assuming someone had given the invariant to you in the first place.
- Q7: There is a hint given for stock(self, p) that
  "Using "in" to test if p is already on the list WILL NOT WORK. Instead, check each element in goods for equality with p, making use of the equals method you wrote for Product."
  This is actually not strictly true **under the conditions stated by questions 6(d) and 7** that an __eq__ method for class Product has been written that **ignores** whether two Products have different quantities.

  Nonetheless, be sure you know that if a class has an __eq__ method defined, then that __eq__ method is what is used for ==.

  Alternate solution with while-loop:

```
found = False
i = 0
while not found and i < len(self.goods):
    g = self.goods[i]
    if g == p:
```

```
            g.quantity += p.quantity
        i += 1
    if not found:
        self.goods.append(p)
```

-

2018 Fall:

- **Skip 2(b)** (Spring 2021 did not talk about raising Exceptions), **2(c)** (Spring 2021 did not cover try/accept).
- Q3: too based on Fall 2018's A7.
  Q4 alt solution:

```
for row_offset in range(n):
    row1 = table[k+row_offset]
    row2 = table[h+row_offset]
    for col_offest in range(n):
        temp = row1[k+col_offset]
        row1[k+col_offset] = row2[h+col_offset]
        row2[h+coloffset] = temp
```

- Q6:
  - Solutions typo:the value of the global variable p should be id5, not id1.
  - Spring 2021 would include 8 as a line in the first odds() call frame.
  - Assume that `[] + some_list` evaluates to `some_list`, not a copy of `some_list`. Python Tutor code: [link]

2018 Spring:

- Some posted versions (ones that don't have "slight edits in Spring 2021" in the title) have a few places that need correction:
  - Q1 6th question: "class method" should say "instance method" or "object method".
  - Q2(b) "executes line 31" should continue with "(assuming lines 29-30 were just executed)"
  - Q2(c) "executes line 32" should continue with "(assuming lines 29-31 were just executed)"
  - Q3 the first line of the docstring should say "Returns a list of every non-empty sequence of non-space, non-@ characters that directly follows an @ in s.
- Q2(c): the word "invariant" here just means, "condition that shouldn't be violated"; the question is fair game for Spring 2021.
- Q2(b): The first line in the solutions says "for i in list(range(len(the_menu)))". The preferred usage nowadays would be "for i in range(len(the_menu))"; that is, there's no reason (in our context) to use list().
- Q6: skip (no loop invariants this semester)

2017 Fall:

- Q2(a) explanation typo: it should read "the second line (the slice) makes b be [[4, 5, 6], [7, 8. 9]]". Here is a Python Tutor link: https://goo.gl/k4L7Ct
- Q2(3) skip: assertions (properties/statements) not covered in spring 2021.
- Q2(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 2021 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:] evaluates to the empty list.
    - Using the Spring call-frame conventions, the line number 4 should occur in the frame for take.
- Q8 skip (loop-invariants, not covered in spring 2021)

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()
```

- Q3:
  - Here is a Python Tutor link (to a Python 3 version): https://goo.gl/vgnA4v
  - For Q3(c), 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"
- Skip 6 (loop-invariants, not covered in spring 2021)
- Q7: The question asks for a while-loop based on a given loop invariant. You can either skip that question in Spring 2021, or attempt it ignoring the loop invariant.

2016 Fall:

- skip 2(b) (we didn't cover this terminology), 2c (related to loop invariants, which we didn't cover in 2021 spring) 3 (graphics content we didn't cover), 4b (we haven't talked about types of exceptions), 5 (no loop invariants in 2021 spring)
- 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[4] ; pretend the example says "Gruoch" instead of "Lady Macbeth"

2015 Fall:

- 1(c): we did insertion sort and merge sort in Spring 2021.
- Skip 3 (or assume you would be given the specification for the superclass's __init__() method.)
- Skip 8 (loop invariants not in 2021 Spring)

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 and uses numpy)
- 4(b): the notation `sum(abs(A[0:k,i]))` means: the sum over rows 0 up to but not including `k` of the absolute value of the entry at index `i`.
- 10(b): the conversions to float are not necessary in Python 3. 10(c) is fine but would need to be converted to our notation.

2014 Fall

[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.

- **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)
- Skip 3(b) (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)". We would give In the draw() method, we would prefer `super().draw(view)` instead of `GRectangle.draw(self, view)`.
- Q5: take care to note that `selected()` is a 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(…)`)
- Q8: skip (loop invariants)

2014 Spring:

- Skip Q5 (we cannot post the code on the accompanying handout due to standing agreements with other instructors).
- Skip Q6 (loop invariants)

2013 Fall:

- Q2(a): skip the getters.
- 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!)
- Q4(a), pair_average(): In Spring 2021, we might have supplied a hint about how to handle pairs of numbers, that is, looking at `2*k` and `2*k+1` for `k` in 0, 1, 2, …
- Q6: skip in Spring 2021
- Q7: skip in Spring 2021

2013 Spring

- Q2: in 2021 Spring, we would tell you about sets and the union method for sets.
- Q3, ignore the "try/except" part of the is_maverick, i.e., assume the precondition would have been given this semester. The "don't forget to do float arithmetic" comment is in regards to the fact that in Python 2, "x/y" would have been floor division on ints and so wouldn't give the desired result. The given answer avoids doing any division at all.
- Q4: change of super in Python 3: the solution would instead be `super().__init__("Tory", record)`
- Q6: skip in Spring 2021.