CS 1110, Spring 2020: Prelim 1 Study Guide

Prepared Monday March 2, 2020 by L. Lee and A. Kellison

Updated Tuesday, March 3, 2020.

Updated Wednesday, March 4, 2020

Administrative info

Time and locations of the regular exam listed at http://www.cs.cornell.edu/courses/cs1110/2020sp/exams What room to go to is determined by your NetID: **check the website beforehand for where you, personally, should go.**

For makeup exam requests, CS1110 administrative assistant Ms. Lacy Lucas (LSL92) will be contacting students directly. If you haven't heard anything by noon on Fri. Mar. 6, please email Ms. Lucas to check in.

Bring your Cornell ID and writing/erasing utensils**.** The exam is closed book, "closed notes," no electronic or external aids, etc**.** We will be checking IDs, possibly at the beginning of the exam, possibly when you turn in your exam.

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

Topic coverage

The prelim covers material from lectures 1-12 inclusive (start of course until Tuesday, March 3rd inclusive), assignments A1-A3, and labs 01-06.

For objects, we will explain to you any necessary information about the objects' class, so you do *not* need to understand the mechanics of class definitions. For for-loops, we expect you to be able to analyze the behavior of a given for-loop and you may be asked to write one yourself.

String methods `split` and `join` are on the last slide for lecture 10 (Thursday, February 20th). Since they are so useful, you ought to get acquainted with them; note in various prior prelims how they lead to short solutions for realistic problems.

Our mechanisms to help you prepare

The lecture of Thu. Mar 5th will be a review session with a prepared presentation.

The lectures and labs of Tue. Mar 10th will be open office hours held at the usual locations of those labs and lectures.[1] The full menu of office/consulting hours can be viewed here: https://www.cs.cornell.edu/courses/cs1110/2020sp/staff

The staff are offering a limited number of extra 1-on-1s; keep an eye on the corresponding CMS "assignment." These 1-on-1 appointments should be reserved for questions that would not be suitable for a group setting.

Solutions and grades for A2 will be posted by Tues. March 3rd (target: early morning, but no guarantees).[2]

Code examples are posted for most lectures to exemplify the corresponding topics; see the course lectures page, https://www.cs.cornell.edu/courses/cs1110/2019sp/schedule/

We have posted many prior CS1110 exams and their solutions to the web page, https://www.cs.cornell.edu/courses/cs1110/2020sp/exams/; more about these below.

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.[3]
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.[4]
    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? I*f you aren't sure you understand a specification, ask.*
        ii. For this semester, do NOT spend time writing code that checks or asserts preconditions, in the interest of time. That is, don't worry about input that doesn't satisfy the preconditions.

---

[1] There will **not** be a new lab exercise released the week of the prelim. The lab sessions of Wed. Mar 11th are canceled, so don't show up.

[2] By agreement with other CS1110 instructors, we do not release lab solutions. The A1 solutions will likely be released by Wednesday, March 4th.

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

[4] Many coding interviews at companies are conducted at a whiteboard.

iii. 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.[5]

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

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

vi. 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

The style of Prelim 1 Spring 2020 is likely to be closer in spirit to the Spring 2018, 2017, 2014, and 2013 exams than the fall exams and other spring exams.

Some prelim 1s have used `assert`; we have not covered it and you are not expected to know it for the Spring 2020 prelim.

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.

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; perhaps the biggest difference this makes in terms of the relevance of previous prelims is that questions regarding division (`/`) need to be rephrased. Also, python2's print didn't require parentheses and allowed you to give multiple items of various types

[5] 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.

separated by commas (which would print as spaces). 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()`.

Review Session Materials From Previous Semesters

Update Wednesday March 4th, 2020 (previous version had URL to 2017 exams, see below)
2019 Fall:
The version with no answers is here:
https://www.cs.cornell.edu/courses/cs1110/2019fa/exams/prelim1/prelim1-review-noanswers.pdf
(2017: https://www.cs.cornell.edu/courses/cs1110/2017fa/exams/prelim1/prelim1-review.pdf)
The version with answers is here:
https://www.cs.cornell.edu/courses/cs1110/2019fa/exams/prelim1/prelim1-review.pdf
(2017: https://www.cs.cornell.edu/courses/cs1110/2017fa/exams/prelim1/prelim1-review.pdf)

STARTING WITH SLIDE 8, the info *before* slide 8 is *not* relevant to our course this semester; **nor is any mention of what could be on the exam or guarantees about the exam.**

- Question starting on slide 8: you definitely need to be able to use find/index and string slicing, 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]
    lletters = components[len(components) - 1][0] # last letter; might
add mid initial

    # 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)
```

- Example with a mutable object: our notation differs a little – there should be a `RETURN` value in the frame, even if it is `None`.

Notes on Prelim 1 from Previous Semesters

2019 Fall:

- Question 2(d): skip; we haven't covered try/except
- Question 3: "The Heap" is "Heap Space." Fall questions for which one-frame-drawn-per-line notation is used would need to be converted to our one-frame-per-function notation.

- Question 5(c): skip; we haven't covered `assert` statements.
- Question 6 involves a bit of geometric reasoning as well as coding ability. We might not stress the mathematical/geometric reasoning as much.

2019 Spring:

- Update Tuesday March 3rd Question 3(b): Read the problem carefully! A correct solution is dependent on the condition that participation grades are values > 0.0 but <= 2.0.[6]

  Alternate solution using a for-loop:

```
def needs_alarm(complete_scores, num_lectures):
    count = 0
    for score in complete_scores:
        if score == 0:
            # count = count + 1
            count += 1
    return num_lectures/3 < count < 2*num_lectures/3
```

  Alternate solution (adapted from) suggestion from CS 1110 student on Piazza:

```
def needs_alarm(complete_scores, num_lectures):
    missed_class = complete_scores.count(0)
    return num_lectures/3 < missed_class < 2*num_lectures/3
```

Note that an `if-else` statement is not used in either of the above alternate solutions; it is unnecessary. The original solution provided for the prelim could also easily avoid the use of an `if` statement.

- Update Tuesday March 3rd Question 5(a) should have had a specification for the function `fight(attack)` explaining what it is supposed to do. We will be sure to give the specifications of necessary functions in your Spring 2020 Prelim 1. It is important to note that, although the function `fight` sometimes triggers a Python error, *there isn't an error inside the function* `fight`.

---

[6] Spring 2020 staff would have written the problem with the condition that missing a lecture results in a 0, *not* a 0.0. In particular, we might want to differentiate between `list` and `list2` below, but the method `count()` does not.

```
>>> list = [0.0,0]
>>> list2 = [0,0]
>>> list.count(0)
2
>>> list2.count(0)
2
```

- Question 6: There are multiple alternate solutions; note that the solution provided relies on `hidden.find()` returning `-1` if there are 0 occurrences of "guess" in "hidden;" An alternate solution that uses `hidden.index()`:

```
def process_guess(hidden, shown, guess, guesses_left):
    count = hidden.count(guess)
    new_shown = shown
    if count != 0:
        i = hidden.index(guess)
        new_shown = shown[:i]+guess+shown[i+1:]
    if (new_shown == hidden):
        print("YOU WIN!")
    elif guesses_left == 1:
        print("YOU LOSE!")
    return new_shown
```

2018 Fall:

- Question 5, part (c) : Skip; `assert` statements will not be on Prelim 1 for spring 2020.
- Question 6: You are not expected to know what "invariants" are for the spring 2020 prelim, but you should still be able to implement the function according to the specification.

2018 Spring:

- Question 2, part (a): The informal specification for the script `make_my_grade()` states that the argument is a *list of ints*. The example test cases listed in the solutions includes a test on a list of floats: you should not test for cases that do not meet the preconditions of the script, you want cases that represent valid inputs.
- Question 5: The question refers to assignment 2 form Spring 2018, which can be found here: https://www.cs.cornell.edu/courses/cs1110/2018sp/assignments/index.php

2017 Fall:

- Question 1(b) Skip "How do they differ?"
- Question 1(c) Alternate answers: a parameter of a function is a special kind of local variable that is where the arguments to the function are initially stored; an argument is a value that is passed in as input to a function; argument values are placed in parameter variables at the beginning of the execution of a function call.
- Question 1(d) be sure you understand why the answer is a good one, but we are not asking you to memorize four specific points.
- Question 4: replace "arbitrary number" with "arbitrary positive number". It seems OK to leave unspecified whether 'LL0' (ell-ell-zero) is a valid netid.

Alternate solution:

```
def twinsies(netid1, netid2):
    netid1 = netid1.lower()
    netid2 = netid2.lower()

    if netid[2].isalpha():
        numstart1 = 3  # where the numbers start in netid1
    else:
        numstart1 = 2

    init1 = netid1[:numstart1]
    digits_as_int = int(netid1[numstart1:])

    return netid2 == init1+str(digits_as_int+1) or netid2 == init1+str(digits_as_int=1)
```

- Question 5: Alternate fix to 4th bug: change the if-statement to begin

  `if pos > 0 and pos < minpos:`

- Question 6: Alternate solution:

```
def expand(rect, x,y):
    if x > rect.x + rect.width:
        # x is outside to the right
        rect.width = x - x.rect   # Don't change x.rect
    elif x < rect.x:
        # x is outside to the left
        new_width = rect.width + (rect.x - x)
        rect.x = x
        rect.width = new_width

    if y < rect.y:
        # y is outside above
        new_height = rect.height + (rect.y - y)
        rect.y = y
        rect.height = new_height
    elif y > rect.y + rect.height:
        # y is outside below
        rect.height = rect.height + (y - rect.y)
```

2017 Spring:

- Question 2(a) solution: we were definitely *not* expecting student answers along the lines of the latter two solutions. As for that one-liner solution: it trades off a cleverness with the tools Python supplies with not being very easy to read and comprehend.
- Question 4: In some versions of the solutions pdf, the first couple of lines have been cut off. The first code block should read:

```
# Making some aliases to reduce typing
old_a1 = p1.bank_acct
old_a2 = p2.bank_acct
new_acct = Acct(old_a1.balance + old_a2.balance)
```

- Question 8: you can ignore the solution that uses try/except: we haven't covered it yet.

2016 Fall:

- Question 2(a): also acceptable for the definition of parameter is, "the variables in which the arguments (input values) to a function are initially stored.
- Skip Question 2(b) (we did not introduce the terms being asked about)
- Question 2(d) solution: ignore phrase "`or 3/2.0`" (based on Python2's `/` being `int` division for integers)
- Skip most of Question 3(b) (good question, but too lecture-dependent, and also have to convert to Python3 `int` division) BUT:
  - The following question is fair game: where and what is the cause of the bug that causes the `UnboundLocalError` error message (the second test in the question).
  - `return prefix` in the solution version of anglicize (third bug) should be `return pref`
- Question 4: specification is unclear as to whether year could be a single digit. Be able to handle either case.
- Question 6(a) assume `import math` was executed. Don't worry about the fact that we're comparing equality of floats. An alternate solution is

```
def normalize(v):
    norm = math.sqrt(v.x**2 + v.y**2)
    if norm != 0.0:
       v.x = v.x/norm
       v.y = v.y/norm
```

2016 Spring:

- Skip Question 3 (we haven't done while-loops yet)
- Skip Question 6 (we didn't do as much with the random module)

2015 Fall:

- Question 4(a) – solutions have typos.
- Skip Question 4(b) (we have not covered asserts)

2015 Spring:

- Question 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?” (Also, Python3 replaced `raw_input` with `input`)

- Question 3(c): replace `/` with `//` because of switch to Python 3

- Question 3(e): solution should be:
  1 2
  1 1 3
  3 2
  B

- Skip Question 4 (too assignment dependent)

- Question 5(a): you don’t have to know what `raw_input ()`(or, in Python 3, `input()`) does to answer the question.

2014 Spring

- Question 5: the last line in the code, which is a print statement, must, in Python 3, be written as `print(nextlist[0].name)`

- Question 6: there is no need to explicitly cast to floats in Python 3, because `/` in Python 3 is float division.

- Question 7: for the avg function from Q6 to work, and also to be consistent with what we’ve said about “listifying” the output of the map function in Python 3, the answer for Python3 should be `return avg(list(map(float, num_as_str.split())))`

2014 Fall:

- Question 2(b): solution is based on `/` being int division in python 2

- Skip Question 2(d): we did not formally define watches and traces

- Skip Question 4(a): (we have not covered “bare” asserts)

- Question 6 involves quite a bit of geometric reasoning as well as coding ability. We might not stress mathematical/geometric reasoning to quite the same degree.

2013 Spring:

- Question 5(b): some version of the solutions use a Python-specific trick about what happens when a slice uses invalid indices. Since this trick has surprised and confused generations of CS1110 students, we have tried to replace that solution pdf online wherever possible, but versions keep coming up. Here is a solution that doesn't inflict that Python-specific trick on CS1110 students:

```
# Many solutions were possible.
# A common error was to try something like if inits in all last. The
problem is
# that all last is a list of LastUsed objects, not strings, and inits
is a string.

if mname == '':
    inits = fname[0] + lname[0]
else:
    inits = fname[0] + mname[0] + lname[0]
i = last.ind(all last, inits) # inits is new iff i is -1

if i != -1:
    all last[i].suffix = all last[i].suffix + 1
    suf = all last[i].suffix
else:
    all last.append(last.LastUsed(inits,1))
    suf = 1

return inits + str(suf)
```

- Question 6: change cunittest2 to cornellasserts.

Fall 2013:

- Skip Question 2(d) - we have not covered "bare" asserts.