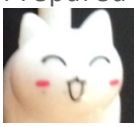
CS 1110, Spring 2021: Prelim 1 Study Guide

Prepared Tuesday March 23, 2021



Update Fri Mar 26: noted an error in some versions of the 2020sp solutions.

Some administrative matters

We will announce later whether or not a function-specification list will be given and/or its contents announced. (We need to figure out how to treat online and in-person exams equally.)

Specific personal arrangements (where's my seat? Am I approved to take an online exam? Where should I go if I have an SDS accommodation, I haven't heard yet? When is my time-zone adjusted prelim happening?) may have to wait until Thursday March 25 due to some complexity behind the scenes. *Watch your email and the Canvas announcement page/course announcement mirror.*

If you have an update to your situation, please do **both** of the following: (1) update your answer on CMS at the "alternate format/time request assignment" (2) send an email to our course administrative assistant Ms. Amy Elser, AHF42, *with a cc: to cs1110-prof@cornell.edu,* to tell Ms. Elser of your update, including "CS1110 prelim 1" in your email's subject line.

We reserve the right to do "spot check" interviews to be scheduled after the exam, where selected students (online or in-person exam takers) would be asked to explain (*not* reproduce from scratch) their answer or approach to/reasoning on one or a few exam questions to either Prof. Fan, Prof. Lee, or a trusted graduate-student TA. At the interview, a selected student would be provided the answers they supplied on the exam.

- It is our hope that we would conduct only a few or even no such spot checks. Our decision would be influenced by, for instance, discovering that there were exam questions or answers being distributed (as has happened in some of the "cheating scandals" in other courses recently).
- The idea is to ameliorate concerns that students have about other students cheating, by instituting the possibility of an extra check that a student generated the answer they supplied.
- But, if a student's exam answer indicates limited understanding, we emphasize that that absolutely would limit the scope of our interview about that question.

  We are thus trying to reduce the effect that nervousness could have on the result of the interview.

  Our goal, if a spot check is run, is: if an exam response indicates a certain level of understanding, we just want to verify that the student who produced the answer can exhibit something like that level of understanding again.

Table of Contents

SOME ADMINISTRATIVE MATTERS 1

TOPIC COVERAGE 2

OUR MECHANISMS TO HELP YOU PREPARE 2

RECOMMENDATIONS FOR PREPARING, IN NO PARTICULAR ORDER 3

NOTES ON QUESTIONS FROM PRIOR EXAMS AND REVIEW MATERIALS 4

IN GENERAL 4

REVIEW SESSION MATERIALS FROM PREVIOUS SEMESTERS 4

NOTES ON PRELIM 1 FROM PREVIOUS SEMESTERS 5

APPENDIX: "STUDENTS SHOULD BE ABLE TO…" LIST 13

Topic coverage

The prelim covers material from lectures 1-12 inclusive (start of course until Tuesday, March 23rd inclusive), assignments A1-A3, and labs 01-12.

Exception: dictionaries will not be on the exam.

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.

Our mechanisms to help you prepare

The lecture of Thu. Mar 25th will be a review session with a prepared presentation, which, as usual, will be recorded.

The lectures and labs of Tue. Mar 30th will be open office hours.[1] But note that we have to figure out whether the in-person labs would be in-person office hours; watch for announcements. The full menu of office/consulting hours can be viewed here: https://www.cs.cornell.edu/courses/cs1110/2021sp/staff/

A2 solutions have been posted to the course schedule page

[1] There will **not** be a new lab exercise released the week of the prelim. The lab sessions of Wed. Mar 31st are canceled, so don't show up.

(https://www.cs.cornell.edu/courses/cs1110/2021sp/schedule/); grades and letter-grade correspondences targeted to be posted by some time Thu Mar 25rd (target: early morning, but no guarantees).[2]

A3 solutions will be posted on Monday Mar. 29th (the day after A3 is due). Grades will have to wait until after we grade prelim 1.

We have posted many prior CS1110 assignments and their solutions to https://www.cs.cornell.edu/courses/cs1110/2021sp/resources/doing-assignments.html and exams and their solutions to https://www.cs.cornell.edu/courses/cs1110/2021sp/exams/
More about the latter 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 often 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 see 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.
    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.

---

[2] By agreement with other CS1110 instructors, we do not release lab solutions. Given the feedback/revision cycle of A1, solutions to A1 will not be released.
[3] Note that we *didn't* expect you to find them straightforward when first released.
[4] In non-pandemic times, many coding interviews at companies are conducted at a whiteboard.
[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.

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 Ed Discussions, 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 2021 is likely to be closer in spirit to the Spring 2018, 2017, 2014, and 2013 exams than the fall exams and other spring exams. Spring 2020 was kind of a mix of Spring and Fall exams, and is certainly fair game, too.

In some semesters, for-loops were not on prelim 1. Be careful not to assume that our semester's prelim 1 won't have for-loops; it will!

Some prelim 1s have used assert; we have not covered it and you are not expected to know it for the this semester's 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 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() if we covered them this semester.

Review Session Materials from Previous Semesters

2020 Fall: We did not have time to go through these; sorry.

2019 Fall:

The version with no answers is here:
http://www.cs.cornell.edu/courses/cs1110/2019fa/exams/prelim1/prelim1-review-noanswers.pdf
The version with answers is here:
http://www.cs.cornell.edu/courses/cs1110/2019fa/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

2020 Spring:

- Question 2: some versions of posted solutions have a mistake in the instruction counter for the first moveCircle() frame. There should not be an "11" in it.
- Question 3(b) forbids for-loops. For practice, you could try writing it with a for-loop, but for this question, not using a for-loop actually seems better.

2020l:

- Question 2(a):

  Where the question says, "NOTE: You do not need to worry about importing introcs", assume this means "you can create an RGB object with a call of the form RGB(r, g, b), where r, g, and b will be the values of the new RGB object's red, green, and blue attributes."

  you can kind of do this one just from the problem description (except that we would give you the specification for function round(), but Fall 2020 students would have been more used to dealing with the particular example class RGB from an assignment.

  Having a problem that deals with accessing attributes and creating new objects is certainly fair

game.

An alternate solution would be to write a helper function to do the repetitive divides and multiplies.

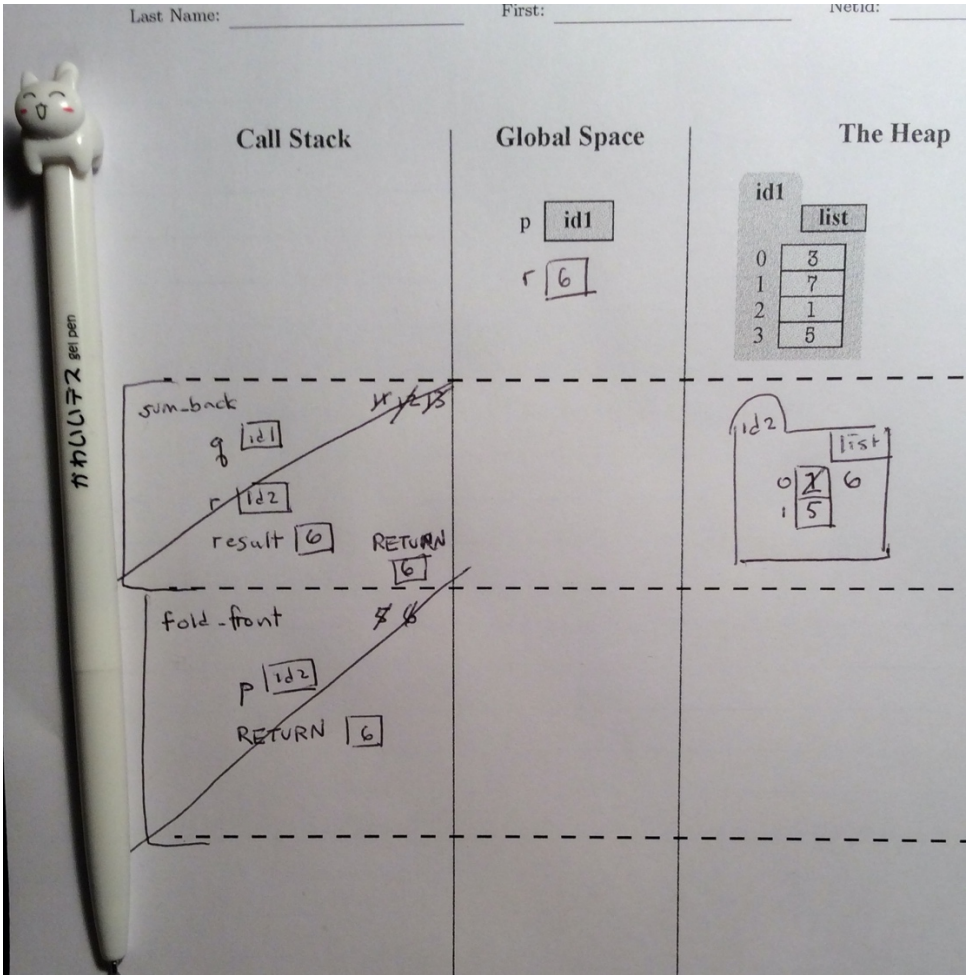
Using 255 instead of 255.0 would be fine, since "/" always treats its arguments as floats.

- Question 2(b):

  A different way of presenting C' would be to explicitly state that if C_1 is not 0 but C_2 is 0, then C' should be 0.
- Question 3(a): this is a nice question. To approach this kind of question, don't try to read the code first![6] Read the specifications of the function, then look at the test case printouts to determine what looks wrong. Then look to the code to figure out what's causing the wrong output.
- Question 3(b): it would be justifiable to have asked a proctor how "overlapping" pairs (like 'yyy' for `b` being 'y' should count.
- Question 3(c): you aren't responsible for the `assert` syntax, but you should know how to write the Boolean expressions that check the assertions.

[6] Code almost always looks right.

- Question 4: here is a solution in Spring notation:



- Question 5 alternate solution. The idea is that the problem would be much simpler if we knew ahead of time which string came first, a or b. So let's just make variables that indicate whichever is first!

```
if a not in word or b not in word:
   return word

# now, a and b are in word. Figure out first one, and use variables to
# refer to the first and second; this simplifies our reasoning
if word.index(a) < word.index(b):
   swap1 = a; swap2 = b # strings themselves
else:
   swap1 = b; swap2 = a

start1 = word.index(swap1); # indices of strings
start2 = word.index(swap2)

if start2 < start1 + len(swap1):
   # overlap
   return word

beg = word[:start1]
mid = word[start1+len(swap1):start2]
end= word[start2+len(swap2):]

return beg + swap2 + mid + swap1 + end
```

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:

- Question 6: There are multiple alternative solutions; note that the solution provided relies on hidden.find() returning -1 if there are 0 occurrences of "guess" in "hidden;" Here is an alternative 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 and so should not be in the solution: you should not in general test for cases that do not meet the preconditions of the script.
- Question 5: The question refers to assignment 2 from 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 (modulo typo for the word "parameter", 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) assumes import math was executed. For this question, 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: Skip this question if map hasn't been covered this semester. 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.

Appendix: "Students should be able to…" list.

(written mostly by Ariel Kellison)

1. Types and Expressions.

1. Students should be familiar with the basic types.
2. Students should know that strings are immutable and that lists are mutable.
3. Students should know how to find the type of a value in python.
4. Students should be able to write the results of operations on the basic types and evaluate expressions according to the precedence of python operators by hand.
5. Students should be able to convert between types and identify/fix type errors in written code.

2. Variables and Assignments.

1. Students should be able to describe how to execute an assignment statement.
2. See 3.7
3. See 6.2
4. Students should be able to show diagrammatically that multiple variables can reference the same object.

3. Functions and Modules.

1. Students should be able to access module variables appropriately based on import.
2. Students should be able to identify and fix errors caused by calling functions from modules that have not been properly imported.
3. Students should be able to describe the danger of importing everything from a module; students should be able to identify and fix errors caused by importing everything from a module.
4. Students should be able to describe the difference between the statements `print` and `return` and be able to give examples of when it is appropriate to use each.
5. Students should be able to write a basic function using proper syntax based on a given specification.
6. Students should be able to define and give examples of local variables.

4. Strings and String Slicing.

1. Students should be able to access substrings of a string using the correct syntax.
2. Students should have basic string methods memorized and should be able to use these methods to carry out specified operations on strings.
3. Students should be able to write basic functions for string manipulation.

5. Specifications, Testing, and Debugging.

1. Students should be able to identify and fix basic errors caused by improper commenting, importing, and indentation.
2. Students should be able to write print statements correctly in appropriate places for debugging.
3. See 1.5
4. Students should be able to write multiple distinct test cases given the specification of a function.

6. Objects.

1. Students should be able to write a call to a specified constructor using the correct syntax.
2. Students should be able to access and assign the attributes of an object variable using the correct syntax.
3. Students should be able to diagrammatically express that object variables hold ids.

7. Conditionals and Control Flow.

1. Students should be able to correctly evaluate basic boolean expressions
2. Students should be able to identify and produce correct if and if-else statements on paper.
3. Students should be able to show that they follow control flow for conditionals by providing the correct values for variables and print statements, as well as constructing diagrams correctly.

8. Memory in Python.

1. See 6.3
2. See 7.3
3. See 2.4
4. See 9.3

9. Lists and Sequences.

1. See 1.2
2. Students should have basic list methods memorized and should be able to use these methods to carry out specified operations on lists.
3. Students should be able to diagrammatically represent list indexing correctly.
4. Students should be able to diagrammatically represent that lists can contain objects.
5. Students should be able to access instance attributes, when appropriate, of list elements using the correct syntax.
6. Students should be able to use sequences appropriately in for-loops.