CS 1110, Spring 2022: Prelim 1 Study Guide

Prepared Wednesday Mar 2, 2022

Updates (all marked in orange in the text):

- Sat Mar 5: (1) pg 1: "lectures 1-12" should be "1-10"; (2) explanation of usage of negative indices in Fall solutions: typo "sp" -> "s"
- Fri Mar 4: spring 2018 prelim had a for-loop, so not fair game for 2022 Spring Prelim 1
- Fri Mar 3: typo in "students should be able to" list: "3.7" should be "3.6"

Table of Contents

Topic coverage

The prelim covers material from lectures 1-10 inclusive (start of course until Thursday, Feb 24th inclusive), assignments A1-A2, and labs 01-10. (The material on the Feb 24th slides was covered in Lab 10).

Exception: Prelim 1 will not cover the split() or join() methods or tuples (i.e., slides 39 and 40 of Lecture 12).

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

Be sure to consult Lecture 12 slide 3 (the one with the annotated calendar) and the description of it given in lecture (consult the lecture recording.)

We will provide a reference sheet of some methods and functions; exact contents to be announced later.

Recommendations for preparing

1. **Be able to do this semester's assignments and labs so far fluently and "from scratch"**.[1]
2. Go through the lecture slides, making sure you understand each example, including the code examples that are posted along with the lecture slides.
3. Do relevant problems from previous exams.
    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.[2]
    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.

    For Spring 2021 prelim 1, there is testing code available! **Use it:** Write your own functions in, and then call the appropriate testing function under the line "if __name__ == '__main__':

4. 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.

Strategies for answering coding questions

1. 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? Do you understand the given examples/test cases? I*f you aren't sure you understand a specification, ask.*

2. 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.
3. 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.[3]
4. Comment your code if you're doing anything unexpected. But don't overly comment - you don't have that much time.
5. Use variable names that make sense, so we have some idea of your intent.

[1] Be reassured that we *didn't* expect you to find them straightforward when first released.

[2] In non-pandemic times, many coding interviews at companies are conducted at a whiteboard.

[3] 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.

6. If there's a portion of the problem that you can't do and a part that 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 know how to do.

Notes on questions/solutions from prior exams

In general

The style of Prelim 1 Spring 2022 is likely to be closer in spirit to the Spring 2021, 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, but is certainly fair game, too.

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

Spring relies less on negative indexing because it's not universal across programming languages. But so you can interpret solutions using them: s[-4] is the same as s[len(s)-4], and s[-3] is the same as s[len(s)-3].

In general, Spring 2015 and Spring 2016 are pretty different than what one can expect for this semester's exams. (If you do look at them, note that they 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-diagram-drawn-per-line notation is used (i.e., the very, very long folder/call-frame solutions) would need to be converted to our one-frame-per-function notation. **Do not use the fall notation on spring exams --- so skip the Fall diagram questions.**

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.

Each individual exam

2021 Fall:
We had access to the original "source" documents, so most comments are in orange in the actual pdfs posted on our exam archive.

- Q4, first bug: we could imagine students seeing from the debugging print output that the else clause in dayize() was being triggered, but not being able to "see" why line 53 wasn't being

triggered. One could have attempted partial credit by stating that lines 60-61 were being executed instead of lines 54-55.
If you find yourself stuck unable to find a bug, we advise moving on to other parts of the exam and then returning with "fresh" eyes to the problem.
See also the notes for Fall 2020 on how to approach such questions.

- Q6(b): although spring students should skip this question, note that all the divisions by 255.0 could also have been divisions by the int 255, since / is float division and its result is always a float.

2021 Spring:
Again, see remarks about using the testing code provided in the exam archive!

- Skip Q3 (we haven't done loops yet)
- Q5: not clear why the solutions have repVal as the last item, instead of the 3rd, but it doesn't really matter.
- Q6: assume that the list of "tag numbers" in the accepted, rejected, and waitlisted lists of a given Academy are all distinct (no duplicates within a list, no lists overlap)

2020 Fall:

- Q2(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.

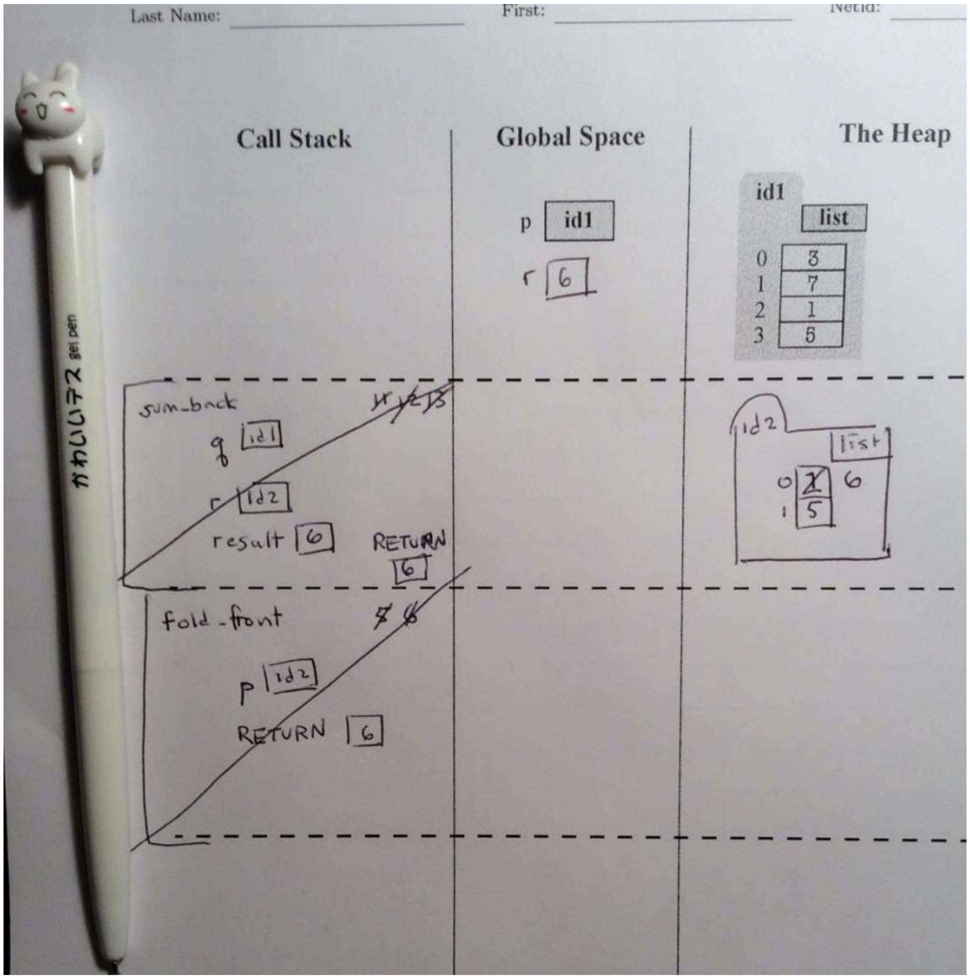
  Having a problem that deals with accessing attributes and creating new objects is certainly fair game.

  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.
- Q2(b): Spring would advocate presenting C' by saying explicitly that if C_1 is not 0 but C_2 is 0, then C' should be 0. (We would not state that a/0 is infinity.)
- Q3(a): this is a nice question. To approach this kind of question, don't try to read all the code first![4] Read the **specifications** of each 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.
- Q3(b): it would be justifiable to have asked a proctor how "overlapping" pairs (like 'yyy' for b being 'y') should count.
- Q3(c): you aren't responsible for the assert syntax, but you should know how to write the Boolean expressions that check the assertions.

[4] Code almost always looks right.

- Q4: here is a solution in Spring notation:



- Q5 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

# if here, 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
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
```

2020 Spring:

- Q1(a) has a for-loop; skip for Spring 2022 prelim 1.
- Q1(b) involves short-circuit evaluation
- Q2: 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.
- Q3(b) forbids for-loops. Once you know about 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.
- Q4(a) and Q4(c) involve for-loops; skip for Spring 2022 prelim 1.

2019 Fall:

- Q2(d): skip; we haven’t covered try/except
- Q3: We suggest skipping Fall diagramming questions, but in case you are looking at the solutions: “The Heap” is “Heap Space.”
- Q4 solution: Spring relies less on negative indexing because it’s not universal across programming languages. s[-4] is the same as s[len(sp)-4], and s[-3] is the same as s[len(s)-3]. The solution has a typo: “sp[-4]” should be “s[-4]”
- Q5(c): skip; we haven’t covered assert statements.
- Q6 involves a bit of geometric reasoning as well as coding ability. We might not stress the mathematical/geometric reasoning as much.

2019 Spring:

- Q6: There are multiple alternative solutions. 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(), but checks the count() of guess in hidden first so that it is safe to apply the index() method.

```
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:

- Q5, part (c) : Skip; assert statements will not be on Prelim 1 for spring 2020.
- Q6: 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:

- Q2, 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.
- Q3(a): involves for-loops, so you would not be responsible for this question in Spring 2022 Prelim 1.
- Q5: Although the question refers to a homework assignment A2, it's not necessary to have done that assignment to be able to do this exam question.
- Q6: involves for-loops, so you would not be responsible for this question in Spring 2022 Prelim 1.

2017 Fall:

- Q1(b) Skip "How do they differ?"
- Q1(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.
- Q1(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.
- Q4: replace "arbitrary number" with "arbitrary positive number". It seems OK to leave unspecified whether 'LL0' (ell-ell-zero) is a valid netid.

  Typos in given solution:
  "if netid1[pos2].isalpha()"

should be
“if netid2[pos2].isalpha()”, and
“pref2 = netid1[:pos2].lower()”
should be
“pref2 = netid2[:pos2].lower()”.

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)
```

- Q5: alternate fix to 4th bug: change the if-statement to begin
  ```
  if pos > 0 and pos < minpos:
  ```
- Q6: 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:

- Q2(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 against not being easy to read and comprehend.
- Q4: 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)
```

- Q6: a very nice series of questions, but involves for-loops, so skip for Spring 2022 Prelim 1.
- Q8: you can ignore the solution that uses try/except: we haven't covered it yet.

2016 Fall:

- Q2(a): also acceptable for the definition of parameter is, "the variables in which the arguments (input values) to a function are initially stored.
- Skip Q2(b) (we did not introduce the terms being asked about)
- Q2(d) solution: ignore phrase "or 3/2.0" (based on Python2's / being int division for integers)
- Skip most of Q3(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
- Q4: specification is unclear as to whether year could be a single digit. Be able to handle either case.
- Q6(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:

- Q1: uses a for-loop, so skip for Spring 2022 Prelim 1.
  But incidentally, the solution using a construction we haven't seen in Spring 2022.
  Consider the body of the convert() function to be:

```
if c== 'U':
  return 1
else:
  return -1
```

It also uses the idiom “floor += convert(i)”. Consider this to be “floor = floor + convert(i)”

- Skip Q3 (we haven’t done while-loops yet)
- Skip Q6 (we didn’t do as much with the random module)

2015 Fall:

- Q4(a) – solutions have typos.
- Skip Q4(b) (we have not covered asserts)
- Q6: see notes on Fall 2021 prelim 1.

2015 Spring:

- Q1(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)
- Q3(c): replace / with // because of switch to Python 3
- Q3(e): solution should be:
  1 2
  1 1 3
  3 2
  B
- Skip Q4 (too assignment dependent)
- Q5: for-loop and while-loop, so skip for Spring 2022 Prelim 1. You don’t have to know what raw_input ()(or, in Python 3, input()) does to answer the question.
- Q6: loop, so skip for Spring 2022 Prelim 1.

2014 Fall:

- Q2(b): solution is based on / being int division in python 2
- Skip Q2(d): we did not formally define watches and traces
- Skip Q4(a): (we have not covered “bare” asserts)
- Q6 involves quite a bit of geometric reasoning as well as coding ability. We might not stress mathematical/geometric reasoning to quite the same degree.

2014 Spring

- Q4: has for-loop, skip for Spring 2022 Prelim 1
- Q5: the last line in the code, which is a print statement, must, in Python 3, be written as print(nextlist[0].name)
- Q6: there is no need to explicitly cast to floats in Python 3, because / in Python 3 is float division.
- Q7: 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()))).

Fall 2013:

- Skip Q2(d) - we have not covered “bare” asserts
- Q4: instead of relying on find() giving a -1 when the target isn’t found, one can use index() in conjunction with checking whether the target exists using count():

```
# set up some useful boolean values
has_spaces = s.count(‘ ‘) > 0
is_cap = s[0].isupper()

if  has_spaces:
    pos = s.index(‘ ‘)
    word = s[:pos]
    tail = s[pos:]
else:
    word = s
    tail =’’   # empty string

if is_cap:
      word = word.upper()
 else:
      word = word.lower()

return word+tail
```

An alternate, tricky solution we don’t expect beginning students to see: you can actually assign *functions and methods* to variables!

```
# set up some useful boolean values
has_spaces = s.count(‘ ‘) > 0

if  has_spaces:
    pos = s.index(‘ ‘)
    word = s[:pos]
    tail = s[pos:]
else:
    word = s
    tail =’’   # empty string

if  s[0].isupper():
    case_method =word.upper    # do NOT include parens
else:
    case_method = word.lower


return case_method()+tail     #DO include parens to call case_function
```

Spring 2013

- Q5(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)
```

- Q6: change cunittest2 to cornellasserts.

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 3.6
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.~~