CS 1110, Spring 2018: Prelim 2 study guide

Prepared Tuesday April 16, 2018

Administrative info

Time and locations of the regular exam listed at http://www.cs.cornell.edu/courses/cs1110/2018sp/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 and SDS accommodations, CS1110 administrative assistant Ms. Jenna Edwards (JLS478) will be contacting students directly. If you haven't heard anything by Tue Apr 17 noon, please email Ms. Edwards 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 1-21 inclusive (start of course until Tue Apr 17 inclusive), assignments A1-A4 and labs 01-12. Emphasis will be on material not tested on prelim 1.

For while-loops, we expect you to be able to analyze the behavior of a given while-loop, but you will not be asked to write one yourself. You should know what assert statements do, although you will not be asked to write them yourself.

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 (although you may be allowed to use a for-loop *in conjunction with* recursion). 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

The lectures of Thu Apr 19th will be (identical) review sessions with a prepared presentation.

The lectures and labs of Tue Apr 24th 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: http://www.cs.cornell.edu/courses/cs1110/2018sp/about/staff.php

Solutions for A2 and A3 have already been posted to the assignments page of the course website. Solutions for A4 will be posted Friday Apr 20th (target: early morning, but no guarantees).[2] We do not currently anticipate being able to have the A4s graded before prelim 2.

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

We have posted many prior CS1110 exams and their solutions to the Exams webpage listed above; 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? *if you aren't sure you understand a specification, ask.*
        ii. For this semester, do NOT spend time writing code that checks or asserts preconditions unless told otherwise. 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]

[1] There will **not** be a new lab exercise released the week of the prelim, and similarly, assignment A5 will **not** be released before the prelim. The lab sessions of Wed Apr 25th are cancelled, so don't show up.

[2] By agreement with other CS1110 instructors, we do not release lab solutions..

[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.

[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.

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 2 Spring 2018 is likely to be closer in spirit to the Spring 2017, 2014, and 2013 exams than the fall exams.

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 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.

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(). Another difference for Python 3 is that one can omit "object" from inside the parentheses in the header of class definitions and the class will still be a subclass of object.

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 (yet), but the former is preferred.

Review session materials from over the years
2017 Fall STARTING WITH SLIDE: 5
The info *before* slide 4 is *not* relevant to our course this semester; nor is any mention of what could be on the exam or guarantees about the exam.
Version with no answers is here:
http://www.cs.cornell.edu/courses/cs1110/2017fa/exams/prelim2/prelim2-review-noanswers.pdf
and version with answers is here:
http://www.cs.cornell.edu/courses/cs1110/2017fa/exams/prelim2/prelim2-review.pdf

- Question starting on slide 8:
    - Since this was not a coding question but a "trace the code execution" question, less documentation was provided than usual.
    - Difference from this semester: we would have crossed-out line numbers in the program counters. Top frame: missing crossed-out 1 and 2. Next frame: missing crossed-out 1, 2, 4. Next frame: missing crossed-out 1, 2. Next frame: missing crossed-out 1, 2, 4, 6.
    - The callout that has both "`s='abc'`" and "s='c'" on slide 10 means that in one call frame, s was 'abc', whereas in another call frame, s was 'c'.
- Question starting on slide 12:
    - In the given solution, xval means the current x-to-the-something-power. xval is initialized to 0 because x*0 is 1 for any x.
    - Alternate solution that loops over (most) indices of p, instead of the elements of p.

```
def evaluate(p, x):
  sum = p[0]  # guaranteed to exist b/c p is guaranteed to be non-empty
  for i in range(1, len(p)):
      sum += (p[i]*(x**i))
  return sum
```

- Question starting on slide 14:
    - Change the first line of the specification to "Returns: list where the item at position i is the max value occurring in column i". (That is, the word "row" in the original spec might be confusing.)
    - Consider the spec to have the extra condition that the input table is not changed.
    - One can also have the first for-loop have the header `for row in table[1:]` .
- Question starting on slide 17:
    - The spec for str should say that because of the possibilities of unknown names or emails, the possible return formats are: '<name> (<email>)', '<name>', '(<email>)', and '' .
    - As noted in the "In General" section above, you can skip parts of the question dealing with getters and setters
    - Slide 21: typo in the assert; should read

      ```
      assert (isinstance(n,str) or n is None) and (y > 1900 or y == -1)
      ```

      Also, we would not have you write setters, so just write the body as

      ```
      self._name = n
      self._email = e
      self._born = y
      ```

    - Slide 22 typo: no equals sign after return, and to handle the space between the name and the email parenthetical, the 2nd-to-last line needs to be changed, one example fix is:
      s = ‘’ if self._name is None else (self._name + ‘ ‘)
      Also, here is an alternate solution if you aren’t comfortable with the conditional expressions:

```
def __str__(self):
    if self._name is None and self._email is None:
        return ‘’
   elif self._name is None:
       return ‘(‘ + self._email + ‘)’
   elif self._email is None:
       return self._email
    else:
       return self._name + ‘ (‘ + self._email + ‘)’
```

    - For the __init__ of PrefCustomer: I advise not using l (the lowercase version of the letter L) as a variable name or as a suffix: in many fonts, it is too easy to confuse the lowercase letter l (ell) with the number 1 (one). This can introduce bugs that can have you tearing your hair out. Also, instead of self.setLevel(l), we would use self.level = l (again, that’s the parameter “ell”)
- Question starting on slide 27:
    - Typo in assert for Senator.__init__(): change isinstance(value, str) to isinstance(s, str)
    - We would 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.

- Skip the Exceptions questions on slides 34-37; we haven’t covered try/except (yet).

Previous prelim 2s

2017 Fall:

- Q2 pairswap:
    - Alternate solution:

```
for ind in range(len(nlist) – 1):  # will not include the last index
    if ind % 0 == 0:
        temp = nlist[ind]
        nlist[ind] = nlist[ind+1]
        nlist[ind+1] = temp
```

    - An alternative while-loop solution (for 2018 spring, you wouldn’t have to write it, but you should be able to analyze it) is:

```
even_pos = 0  # Next even position to handle
while even_pos + 1 < len(nlist):
   # We know here exists a later item to swap with
   temp = nlist[even_pos]
   nlist[even_pos] = nlist[even_pos+1]
   nlist[even_pos+1] = temp

   even_pos += 2
```

- Q2 colavg: might be wise to add to specification that the table should not be altered.
  - Alternative solution:

```
sum_list = table[0][:]
for row in table[1:]:  # Add in all the other rows
   for i in range(len(row)):
      sum_list[i] += row[i]
n = len(table)
for i in range(len(sum_list)):
   sum_list[i] /= n
return sum_list
```

- Q3 segregate:
  - alternate solution that does not use conditional expressions, but does use assignment to a tuple:

```
if len(nlist) == 0:
   return (-1, [])

head = nlist[0]
(tail_pos, outlist) = segregate(nlist[1:])
# tail_pos is start of nonnegs in initial outlist. We must now add the head to outlist.
if head < 0:
   outlist.insert(0, head)
   if tail_pos != -1:
      # There were non-negative numbers in outlist already
      return (tail_pos+1, outlist)
   else:
      return (-1, outlist)
else:
   # head is non-negative
   if tail_pos != -1:
      outlist.insert(tail_pos, head)
      return (tail_pos, outlist)
   else:
      # There were no non-negative numbers before
      return (len(outlist), outlist + [head])
```

- If you want to test your own implementation, here's some code you can use:

```
import cornellasserts as ca

# keys are tuple versions of input lists.
test_cases = {(1, -1, 2, -5, -3, 0): (3, [-1, -5, -3, 1, 2, 0]),
    (-1, -3, -3): (-1, [-1, -5, -3]),
   (1, 5, 4): (0, [1, 5, 4]),
   (1, 2, 3, 4, -1, -5, -2): (2, [-1, -2, 1, 2, 3, 4, 5]),
   (): (-1, [])
}

for tc in test_cases:
   print('Testing ' + str(list(tc)))
   ca.assert_equals(test_cases[tc], segregate(list(tc)))
```

- Q5: we would say that you **do** need to provide specifications for any helpers.

2017 Spring:

- Q1: some online versions have some weird stray (and incorrect) code included after the line
  ```
  ### Your implementation must make effective use of a for-loop.
  ```
  If you see that stray code, cross it out; and such versions also don't make it clear that: repeats *should* be included.
- Q2: assumes one has done A4 of 2017 Spring, so skipping this question certainly makes sense. But do observe that you should not be surprised for the exam to have questions that assume significant experience with this semester's assignments. Also, if you are looking for more practice with recursion on a realistic(-ish) setting, you are encouraged to check out that the 2017SP A4 (http://www.cs.cornell.edu/courses/cs1110/2017sp/assignments/assignment4/A4.pdf) and its solution (http://www.cs.cornell.edu/courses/cs1110/2018sp/assignments/assignment4/a4_2017sp_soln.py)
- Q3:
  - Assume that the direction of a train, and thus the direction that is "outward" or "facing out", is predetermined.
  - __str__: Some posted versions of the 2017 spring solutions have the line "text = "Domino" + …. .
    Replace "text = " with "return"

  - Alternate solution to addDomino:

```
def addDomino(self, d):
  dsides = [d.side1, d.side2]
  if self.outwardFacingSide not in dsides:
    return False
  elif not self.canExtend or d.prior is not None:  # Yes, in Python you can say "is not”!
    return False

  # We now know we can add d to self
  d.prior = self
  self.next = d
  dsides.remove(self.outwardFacingSide)   # this leaves the value *not* matching self’s end
  d.outwardFacingSide = dsides[0]
  return True
```

- Q5: skip: we haven’t covered invariants (yet)

2016 Fall:

- Q3: typo in solutions, animation cells 4-6: self should be id3, not id2
- Q4a: typo in solutions: if statement should have “!=”, not “==”

2016 Spring:

- Q3(c): solution typo: “lineseg.P2.pos()” should be “lineseg.P2.Pos()”
- Q4: We would explain that estimating the probability would just mean counting the number of times the dice came up with exactly two having the same value, divided by the number of rolls.

2015 Fall: skip 3(a) (is vs ==), 3(d) (exceptions)

2015 Spring: skip 4(d) (graphics), 6(b) (try/except) , 6(c) (timing)

- Q2: ignore remark about being familiar with the Traveling Fanatic problem
- Q6(d): answer is “no”: if L is a list, it doesn’t have an attribute nWords.

2014 Spring: skip Q3 and Q4 (writing while-loops); Q6 (loop invariants)

- Q5: the “separate handout” being referred to is http://www.cs.cornell.edu/courses/cs1110/2017sp/exams/prelim2/2014-spring-prelim2-enroll.pdf

2013 Fall: skip Q6(b) (exception types)

2013 Spring: skip Q3 (loop invariants), Q4 (writing while-loops)