CS 1110, 2022SP: Final Exam Study Guide

Prepared May 10, 2022 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

Table of Contents (entries are clickable!)

Rules

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, you may not use break statements

In Spring 2022, you must use while-loops (without break statements) instead of for-loops with break statements.

On the exam, if you are asked to solve a problem a certain way, answers that use a different approach may 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 receive no cedit.
(It's OK to use a loop within a *truly* recursive solution, unless specified otherwise; you've seen many examples in labs and assignments.)

Similarly, if we say you must use a loop, then `map()` or recursion is not allowed as the *core* of your solution. [*We didn't even cover map() in Spring 2022/]*

Topic coverage

All material from all course assignments, labs, and lectures. Material after the second prelim (subclasses, while-loops, searching and sorting) is fair game, but it would be reasonable to expect that questions on string/list (including for nested lists)/dictionary processing, memory diagrams, testing and debugging, for-loops (including nested for-loops), recursion, and objects and classes would appear, among others.

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

Class folders and method call frames

It is possible you will be asked to draw class folders. For formatting, we won't be too picky, but do have the "tab" on the top right with the class name (not the object ID), for a subclass, include the parent name in parentheses on the tab. and do put the class attributes and the names of methods in the class folders (not the object folders.)

For method call frames, give the method name as <class name>.<method name>, not just <method name>, since we need to know which class's method is being called.

While-loops: practice problems

In Spring 2022, examples of real-life while-loops can be seen in all the "interactive" assignments: A1, A5, A6.

While-loop "lab-length" practice problems were distributed in a May 1, 2022 announcement.

Final-exam questions that are good while-loop practice include:
2021SP Q3
2019FA Q4 alternate solutions
2019SP Q7 alternate solution
2015SP Q4a

Searching and sorting: practice problems

2021 Fall Q3(c), when adapted to the algorithms we covered in Spring 2022, is a valid question about how much "work" is done by the sorting algorithms that were presented.

2015 Fall Q2(c) is a valid question, again when adapted to the algorithms of Spring 2022.

Be able to implement linear search, binary search, insertion sort, and merge sort, and know the amount of work each requires as a function of n, the length of the input sequence.

Reference sheet for functions and methods

We will provide the Prelim 2 reference sheet.

Recommendations for preparing (only change from Prelim 2 study guide in purple)

Our recommendations from the Prelim 1 study guide still hold. We especially emphasize:

- *Work coding problems at your computer (including perhaps Python Tutor)*, not on paper. Fluency at the keyboard will translate to fluency on paper, and most beginning students *need* to start with the feedback that Python gives.
- Test your solutions by writing your own testing code --- at least include the examples given in function specifications --- or using any we provide; see the Exams archive page for previous Prelim 2 problems and solutions.
- You can use past prelim 1s and 2s for study as well as past finals.

Making your own testing code

You can make quick-and-dirty testing code as follows. Say you're writing a function prelim2, and the spec says that prelim2("hi") returns "there" and prelim2(["hi", "there"]) returns the list ["no", "fair"].

Then, add to the bottom of the file, **non-indented**, something like the following:

Option 1 – simplest, but a little tedious

Write each test out as its own little block of code.

```
print("Testing input 'hi'")
expected = 'there'
result = prelim2("hi")
assert result == expected, "Error: Expected "+repr(expected)+" but got "+repr(result)

print('Testing input["hi", "there"]')
expected = ["no", "fair"]
result = prelim2(["hi", "there"])
 assert result == expected, "Error: Expected "+repr(expected)+" but got "+repr(result)
```

Option 2 – better if you have multiple tests

Better is to loop through a list consisting of input/output pairs.

```
tests = [
   [“hi”, “there”],
   [[“hi”, “there”], [“no”, “fair”]]
        ]

for [theinput, expected] in tests:    # convenient shorthand
    print(“Testing input “+repr(theinput)”)
    result = prelim2(theinput)
    assert result == expected, “Error: Expected “+repr(expected)+” but got “+repr(result)
```

Strategies for answering coding questions (same text as in the Prelim 1 study guide)

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.[1]
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.
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 from prior exams and review materials

[1] 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.

In general

Differences from Fall exams

Spring does not cover getters/setters, mutable vs. immutable attributes, generators.

Spring students do not need to assert preconditions unless explicitly asked.

Fall-exam solutions for which a-full-diagram-drawn-per-line convention 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 convention on Spring exams. Also, note that Spring *includes* else lines in the program/instruction counter. Fall semesters and Python Tutor *don't.*

Spring 2015 and 2016

These fiado have some good problems for Spring 2022, as specifically indicated in the notes below, but otherwise have a quite different style than what can be expected for Spring exams.

Where you see lines of the form "if __name__ == '__main__':", this means that the indented body underneath it is executed when the code is run (the program is executed from the command line).

Before Fall 2017

In the early years, the course was taught in Python 2. Here are some differences this makes in terms of the relevance of previous prelims:

a. questions regarding division (/) need to be rephrased.
b. 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).
c. 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().
d. 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.
e. The usage of super() in Python 2 and Python 3 differ slightly.

Previous finals

2021 Fall

- Q2(a) solution: The "rule of numbers" should perhaps be generalized as: "none", "exactly one", and "some" rather than specifically "2" in "0, 1, 2".
- Q2(b): Spring 2022 should skip (try/accept)
- Q3
    - (a): in Spring, while we don't ask you to memorize names of sorts of variables, you should know what local, global and class variables, parameters, and instance attributes are.

    - (b): Just to be clear: the solutions are (presumably) giving three different answers that were deemed acceptable for receiving full credit.
    - (c) In Spring 2022, the sorting algorithms we saw were *insertion sort*, which scales worst case like n^2 work for input size n, and *mergesort*, which scales worst case like $n\log_2(n)$ work for input size n.
- Q5: In "real life", using binary search and then the list insert() method seems like the most reasonable and efficient approach to this question, given that the input list is sorted.
- Q6: Spring students should skip (too much context from Fall graphical assignments is needed)
- Q7: Spring students should skip (we didn't cover generators)
- Q8: Spring students should skip (we didn't cover coroutines)

2021 Spring

- Q1: the question is fine for an exam, but it was inspired by the Spring 2021 assignments, where there were strings with the given format for a reason. So context is lacking.
- Q3: good question for incorporating while-loops, lists, and objects in a single problem.
    - Self-test: why would it be incorrect to reverse the order of the two operands for "add". That is, what's wrong with saying
      `while wlist[k].code != prod and k < len(wlist):`
      ?
- Q5: If asked to draw class folders, there would be one for class Interval, containing the methods __init__() and mystery().

2019 Fall:

- Skip 2(b) (Spring doesn't cover try/except)
- Skip 3 (too dependent on Fall 2019's A7)
- Q4: It might be clearer to rewrite the definition of "upper triangular" as: "**every** element below the diagonal is 0", and similarly for "lower triangular".
    - Note that according to 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 a matrix 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: matrix[i][j] is in the "northeast" if j > i, and it is in the "southwest" if j < i
    - Alternate solutions:
    -
    - 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 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:"DLC"= "[key for] downloadable content". Although loop-based solutions are also possible in real life (banned for purposes of the Fall 2019 exam), this recursive solution is actually pretty natural (if one doesn't worry about extra call frames being created). 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:])
```

- Q6: Spring call-frame notation would have the "else" line (line 6) occur in the program counter.
- Q7: Skip Q7(a) (Spring doesn't worry about this terminology, 7(d), which is related to material on loop invariants (no longer covered in CS1110).
- Q8: skip (based on loop invariants, no longer covered in CS1110)

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: CS1110 no longer covers 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):
    "*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 ==.
  - In 2022 Spring, the given solution is **illegal** because it uses a break statement. 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:

- Q2:Skip 2(b) and 2(c) (Spring 2022 did not cover Exceptions or try/except).
- 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_offset 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 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]
- Q7: Skip (CS1110 no longer covers invariants)

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 students.
- Q2(d): 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 any more)

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 2022..
- 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 we don't have Turtle-based assignments (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 any more)

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 done 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 anymore)
- 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 are no longer covered) 3 (graphics content we didn't cover), 4b (we haven't talked about types of exceptions), 5 (no loop invariants anymore)
- Q8
    - This is an example of drawing call frames for a recursive function. Students should make sure they understand the (spring version of) solutions, and in general, what happens with call frames for recursive functions.
    - 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[2] ; pretend the example says "Gruoch" instead of "Lady Macbeth"

2015 Fall:

- 1(c): we did insertion sort and merge sort in Spring 2021. We would prefer mergesor'ts nlog(n) time to insertion sort's n^2 steps, for a list of length n.
- 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 Q1(c) (we would not ask about expected numbers or percentages)
- Q2: Probably the question should have included a requirement that a for-loop be used to check for duplicates. Otherwise, a one-line alternative solution for the question as written would be: return sorted(list(set(L)))
- Q4
    - (a): this is a **good** while-loop question.
    - (b)
      Here's a rephrasing of the specification that doesn't involve numpy.

[2] An unfortunate ambiguity in that question that year: people thought the method was supposed to add the word "Lady" to the person's name.

```
def overbudget(a, m):
   """Returns: smallest k such that:
      M <= the sum of the absolute values of entries at position 0 of the rows
      0, 1, 2, ... up to but not including k is

      and

      M <= the sum of the absolute values of entries at position 1 of the rows
      0, 1, 2, ... up to but not including k is

      and

      M <= the sum of the absolute values of entries at position 2 of the rows
      0, 1, 2, ... up to but not including k is
   If there is no such k, returns 0.

   Preconditions: a is an n-by-3 table (nested list) of ints.  m is an int."""
```

See the Exams webpage for a solution that doesn't use numpy that you can edit/try in Python Tutor.
It's a hard question to understand, and a bit tricky to get right because of the "up to but not including k" part, but a good question for practice with while-loops.

- 5: typo in solutions. If R==L: return x == a[L]. Needs double-equals. In the "Better solution": typo: "return true" should be "return True". (Capital letter).
- Skip 7c (we didn't cover numpy).
- Skip 9 (too assignment-dependent and uses numpy)
- 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

- Q2: skip (a) (not too meaningful a question)
  Python 3 solutions to (c):
  1.5 (float division);
  True (short-circuit evaluation for operator "or" so the divide-by-zero never happens)
  BAD (divide by zero)
- Skip 3(b) (try/except)
- Q4: Spring students should skip (too dependent on Fall assignments). But for the curious:
  - Spring would not require asserting preconditions
  - The specifications could be more clear that one is altering *this* GPanel's _contents attribute.
  - Solutions for _ _ init__(): Change first assignment statement in __init__ to "super().__init__(x=x, y=y, width=w, height=h, fillcolorcolor)".
  - Solutions for draw() method: we would prefer `super().draw(view)` to `GRectangle.draw(self, view).`
- Q5: take care to note that `selected()` is a method. Also, you could imagine solutions that move from right to left through self._contents, since we are told the topmost would be the *last* item in self._contents that contains (x,y)

- Q6: you should get the gist of what is happening in the solution, but overall, the question is too dependent on having done the Fall 2014 A7., 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):
  - Spring students: skip the getters, and Spring semesters do not require asserting preconditions. The solutions are accidentally missing the initializer docstrings.
    - Docstring for init for Course should say something like "A new Course with _name n and _prof set to None".
    - Docstring for Init for Instructor should say something like "A new Instructor with _name n and _list set to the empty list"
  - In the solutions for makeInstructor(), it is not necessary to perform any checks before running self.removeInstructor(), given that removeInstructor() has no preconditions.
  - The solutions for both makeInstructor() and addCourse() have assertions saying "assert type(x) == SomeClass". It is recommended to use "assert isinstance(x, SomeClass)" instead.
- Skip 3(b) (try-except)
- Q4(a), pair_average(): In Spring, 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, … len(data)/2.
- Q5: *OK* for Spring (the solutions use the Spring diagram conventions)
- Q6: skip in Spring
- Q7: skip in Spring
- Q8(d), in Spring, while we don't ask you to memorize names of sorts of variables, you *should* know what local, global and class variables, parameters, and instance attributes are.
-

2013 Spring

- Q2: in 2022 Spring, we would tell you about sets and the union method for sets, and you wouldn't need to know what "memorization" means. But you should be able to do this problem if all mentions of "set" were changed to "lists"; presuming duplicates were to be avoided:

```
if s.contacted == []
  return []

out = []
for child in s.contacted:
    if child.contacted != [] and child not in out: # shouldn't be a repeat, but just in case
      out.append(child)
      childresults = fruitful_descendants(child)
      for fd in childresults:
        if fd not in out:
          out.append(fd)
return out
```

- 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 2022 (loop invariants).

Appendix: "Students should be able to…" list

1. Everything from the first and second prelim study guide's "students should be able to", adding in the difference between == and **is**, and how to define and work with optional parameters with default values.
2. Write and work with subclasses
    1. Understand how to write a subclass.
    2. Understand the bottom-up rule for references to instance attributes, class attributes, and methods
    3. Understand method and attribute inheritance from a superclass.
    4. Write methods that over-ride superclass methods, and understand when it is and is not appropriate to do so.
    5. Know how to use super() within a subclass method to call methods from the superclass, and understand when it is and is not appropriate to do so.
    6. Use ininstance() appropriately
2. Write and analyze while-loops
    1. Ideally, determine which kind of loop (for- or while-) is most appropriate to a given programming task
3. Understand searching and sorting
    1. Be able to implement linear and binary search
    2. Know how much work linear and binary search take in terms of n, the length of the input sequence
    3. Be able to implement insertion sort and merge sort
    4. Know how much work insertion sort and merge sort take in terms of n, the length of the input sequence