Last Name: UTION First Name: SOL Cornell NetID, all caps: SU1110

CS 1110 Final Exam May 9th, 2013
SOLUTIONS

This 150-minute exam has **??** questions worth a total of **??** points. When permitted to begin, scan the whole test before starting. Budget your time wisely. Use the back of the pages if you need more space. You may tear the pages apart; we have a stapler at the front of the room.

If a question does not explicitly ask for you to write an invariant, you don't have to for that problem. However, we strongly recommend that you provide comments explaining the meaning of your variables if you think they might be unclear to the graders.

The second page of this exam gives you the specifications for some useful functions.

It is a violation of the Academic Integrity Code to look at any exam other than your own, to look at any other reference material, or to otherwise give or receive unauthorized help.
We also ask that you not discuss this exam with students who are scheduled to take a later makeup.
Academic Integrity is expected of all students of Cornell University at all times, whether in the presence or absence of members of the faculty. Understanding this, I declare I shall not give, use or receive unauthorized aid in this examination.

Signature: ________________________________ Date __________

The Important First Question:

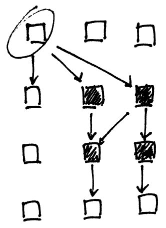
1. [2 points] When allowed to begin, write your last name, first name, and Cornell NetID at the top of each page.

Last Name: ______________________ First Name: ________________ Cornell NetID: ______

2. [6 points] **Recursion.** Consider a class `Snode` representing simplified versions of the nodes from A4: with snodes, we don't worry about "conversion". Snodes have just two instance variables:

- `gen`, an int indicating this snode's generation
- `contacted`, a (possibly empty) list of snodes that this one has contacted. If `s` is an snode, for each item `c` in `s.contacted`, `c.gen` has the value `s.gen + 1`.

We say that an snode `d` is a *descendant* of snode `s` if `d` has been contacted by `s`, or if `d` is a descendant of an snode that `s` has contacted. We also say that an snode is *fruitful* if it has descendants of its own. In the diagram below, the circled snode has 7 descendants but only 4 fruitful descendants (the filled-in squares).



Write a recursive implementation of the following function. To keep your answer short, *don't* employ memoization. *No credit for non-recursive solutions.*

```
def fruitful_descendants(s):
    """Returns: set of fruitful descendants of s. Pre: s is an snode."""
```

Solution:

```
    if s.contacted == []:
        return set()

    out = set()  # fruitful descendants of s known so far
    for c in s.contacted:  # c an snode contacted by s   ##[+1 iterate over children]
        if c.contacted != []:
            out.add(c)                                   ##[+1 do right thing if fruitful]
            out = out.union(fruitful_descendants(c))
                                                         ## [+1 recursive call, +1 on right thing,
                                                         ## +1 union with right thing (union can be
                                                         ## outdented in this case)

    return out                                           ## [+1 return right thing, inc. base case]
```

Alternate solution:

```
    out = set()  # fruitful descendants of s known so far
    for c in s.contacted:  # c is an snode contacted by s
        out = out.union(fruitful_descendants(c))
        if c.contacted != []:
            out.add(c)
    return out
```

3. [12 points] **Loops, objects, try-clauses.** In this question, we model people who have voted on a number of issues. Implement the two methods below according to the given specifications and class invariants.

```
NUM_ISSUES = 10  # the issues that were voted on are numbered 0 .. NUM_ISSUES-1

class Voter(object):
    """An instance is someone who voted on all the issues.

    Instance variables:
    party [string]: voter's political party.  Different strings are considered different
                    parties (e.g., 'green' is different from 'Green')
    record [list of ints]: voter's voting record.  Item i is 1 if this
                    voter voted "Yes" on issue i, 0 otherwise.  Length is NUM_ISSUES.
    """


    def __init__(self, party, record):
        """Initializer: a voter of party <party> and record <record>.

        Pre: <party> a string. <record> is this voter's voting record on the issues
        (not a copy)."""
```

Solution:

```
        self.party = party         ## [+1]
        self.record = record       ## [+1]
```

```
def is_maverick(self, i, sample):
    """Returns: True if for issue i, this voter's vote is in the minority for their
    party among the voters in voter list <sample>,  False otherwise.

    In other words, if this voter is in party p, and *including* this voter
    less than 50% of the members of party p in list <sample> voted the
    way this voter did, then this method should return True.

    NOTE: if an IndexError occurs in the code, this function prints
    'bad issue number' and returns False (rather than crashing).
    Any other type of exception should not be handled by this function.

    Pre: i is a valid issue number, and this voter is in <sample>.
    Do not assume all the voters in <sample> are in the same
    party.

    AFTER-THE-FACT ADDITION: we should have removed the valid-issue-number precondition"""
```

Solution: Below is the intended grading guide. Because we accidentally left in the phrase “i is a valid issue number” in the precondition, the try/except part of the question made much less sense; in the end, we decided to be generous (given that the error was our fault) and award the four try/except points for any response.

```
    try:                                              ## [+1 try block encompasses right stuff]
        same_p_and_v = 0  # count of same-party-&-vote ## [+1 initializing these two]
        same_p = 0 # count of people in same party     ##
        for vr in sample:                             ## [+1 loop over sample]
            if vr.party == self.party:
                same_p += 1                           ## [+1 track same-party count]
                if vr.record[i] == self.record[i]:
                    same_p_and_v += 1                 ## [+1 track same-agree-and-party]
        # don't forget to do float arithmetic
        return same_p_and_v < same_p - same_p_and_v   ## [+1 right idea]
                                                      ## [+1 attempt to avoid/deal with floats]
  except IndexError:                                  ## [+1 for except]
                                                      ## [+1 for IndexError]
        print 'bad issue number'                      ##
        return False                                  ## [+1 for both these lines]
```

4. [7 points] **Subclasses.** A Tory is a voter who belongs to the Tory party. Write the class definition, including complete initializer function, for a subclass of the class Voter representing Tory voters.

   In the interests of time, you need not provide any docstrings, but your initializer must satisfy the following spec:

```
"""Initializer: a voter in the Tory party with voting record <record>.
        Pre: <record> a list of where for every issue i,
        <record>[i] is 1 if this voter voted yes for that issue,
        0 otherwise.
"""
```

Solution:

```
class Tory(Voter):                               ## [+1. "def" instead of "class":
                                                 ## let it slide]
    """An instance is a Voter in the Tory party"""

    def __init__(self, record):                  ## [+1 for self;
                                                 ##  +1 for record]
        """Initializer: a voter in the Tory party with voting record <record>.
        Pre: <record> a list of where for every issue i,
        <record>[i] is 1 if this voter voted yes for that issue,
        0 otherwise. """
        Voter.__init__(self, "Tory", record)    ## [+1 for Voter]
                                                 ## [+3: one point for each argument]
```

OK for students to use super(Voter,self).__init__("Tory", record) but we have discouraged it, since "super" seems to serve certain from-a-CS1110-perspective arcane purposes. For solutions using `super`, watch that the arguments are as specified here, not as in the above solution. We took off a point for responses that did not call a superclass method but just set the instance variables manually, since doing so is contrary to the spirit of using subclass structure.

5. [10 points] **Frames and folders.** You're being pressed into service again as an adjunct CS1110 TA, and your job is to mark this student's frames-and-folders drawing by (a) neatly scribbling out each thing (variable, frame, or folder) that appears in a place where it does not belong, (b) circling each variable that is in the right place but has the wrong value, and (c) writing in each variable and value that is missing from the place where it belongs. There may or may not be corrections in all three categories. The original exam question is: *Diagram the execution of the following code*

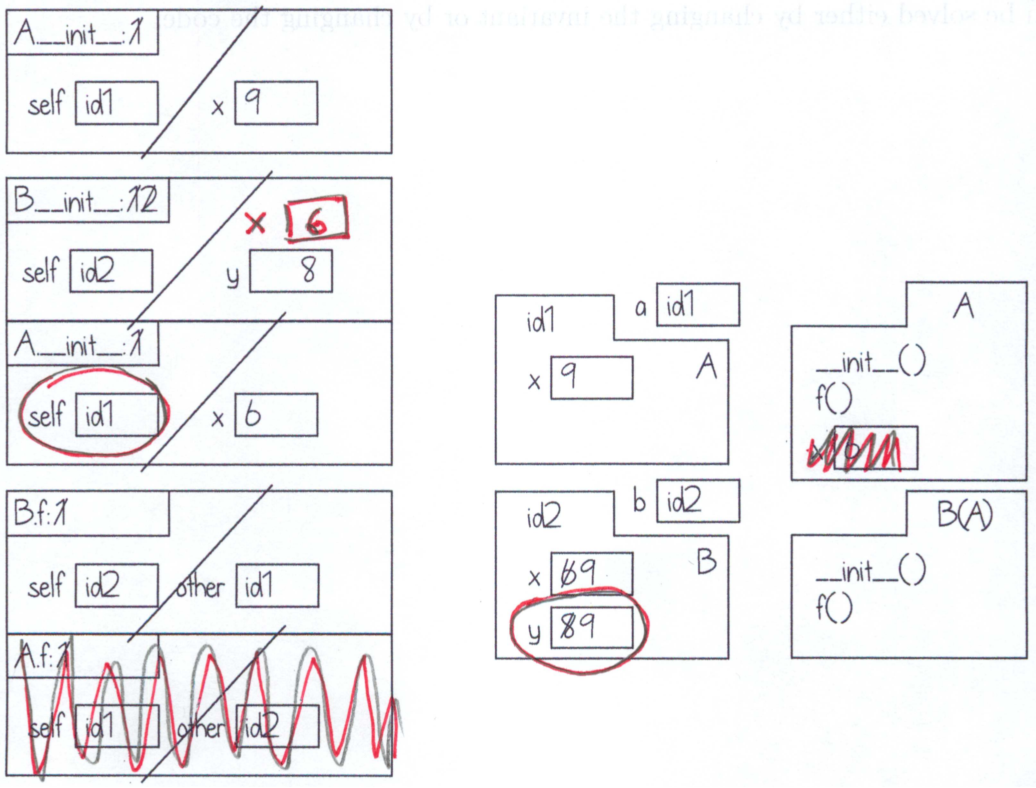
```
a = A(9)
b = B(6,8)
b.f(a)
```

given the following class definitions.

```
class A(object):                  class B(A):

    def __init__(self, x):            def __init__(self, x, y):
        self.x = x                        A.__init__(self, x)
                                          self.y = y
    def f(self, other):
        self.y = other.x              def f(self, other):
                                          self.x = other.x
```

Please include class folders for both classes.



Solution: [+2 for each of the 5 things indicated above; -1 for things marked that aren't actually wrong]

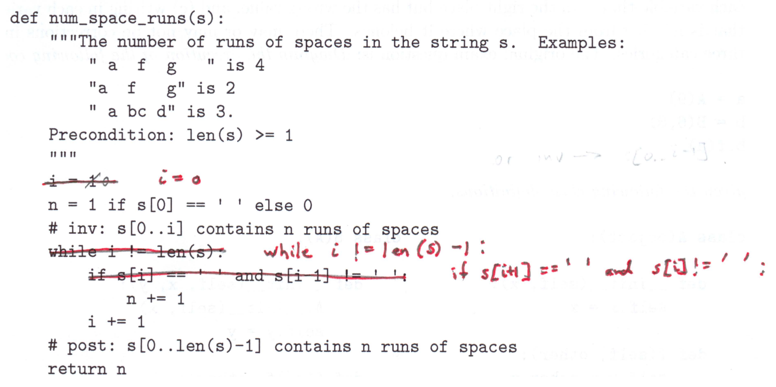
6. [6 points] **Loop invariants.** The following code to count the number of distinct runs of consecutive spaces has an invariant that doesn't correspond to the code.

```
def num_space_runs(s):
    """The number of runs of spaces in the string s.  Examples:
        " a  f  g   " is 4
        "a  f   g" is 2
        " a bc d" is 3.
    Precondition: len(s) >= 1
    """
    i = 1
    n = 1 if s[0] == ' ' else 0
    # inv: s[0..i] contains n runs of spaces
    while i != len(s):
        if s[i] == ' ' and s[i-1] != ' ':
            n += 1
        i += 1
    # post: s[0..len(s)-1] contains n runs of spaces
    return n
```

This code works correctly but does not agree with the invariant. Change at most three lines in this function so that it is still correct but the code agrees with the invariant. Indicate your changes by crossing out the line to be changed and rewriting it neatly to the right. (Hence, any reordering of lines should be done by rewriting the lines, not by drawing arrows.) This problem can be solved either by changing the invariant or by changing the code.

Solution: If one changes the invariant, just change "`s[0..i] contains n runs`" to "`s[0..i-1] contains n runs`" [+6]

Otherwise, [+2] points for each of the following three lines being changed (in terms of partial credit, subtract one point of two for lines where a one-character fix makes it correct):



```
def num_space_runs(s):
    """The number of runs of spaces in the string s.  Examples:
        " a  f  g   " is 4
        "a  f   g" is 2
        " a bc d" is 3.
    Precondition: len(s) >= 1
    """
    ~~i = 1 0~~      i = 0
    n = 1 if s[0] == ' ' else 0
    # inv: s[0..i] contains n runs of spaces
    ~~while i != len(s):~~     while i != len (s) -1 :
        ~~if s[i] == ' ' and s[i-1] != ' ':~~     if s[i+1] == ' ' and s[i] != ' ' :
            n += 1
        i += 1
    # post: s[0..len(s)-1] contains n runs of spaces
    return n
```

Or, change while-loop body to

```
        i += 1
        if s[i] == ' ' and s[i-1] != ' ':
            n += 1
```

(same grading scheme, since three lines changed)

7. [10 points] **Iteration and recursion.** In this question we approach a simple problem, that of making a reversed copy of a list (without modifying the original list), in two ways.

   (a) Implement the following function according to spec, using recursion. Your function must call itself.

```
def reverse(x):
    """Return a copy of the list x, in reverse order."""
```

Solution:

```
    if len(x) == 0: ## [+1 base case, OK if length can be 1]
        return []   ## [+1; if length can be 1, must return x[:] or other copy
    else:
        return reverse(x[1:]) + [x[0]]
                    ## [+1 recursion, +1 on right thing, +1 combined with right thing]
```

Alternate recursive call:

```
return [x[-1]] + reverse(x[:-1])
```

   (b) Implement the following function according to spec, using iteration. Your function must be based on a loop. *Hint: There are solutions based on counting forwards, counting backwards, and looping over the sequence directly; use whichever you find easiest.*

```
def reverse(x):
    """Return a copy of the list x, in reverse order."""
```

Solution: In all solutions: +1 initialize loop; +1 stopping condition for loop; +1 append/insert correct; +1 increment (for loops get this for free); +1 return right thing

```
    result = []   # portion of reversal built so far
    for k in range(len(x)):
        result.append(x[len(x) - 1 - k])
    return result
```

or

```
    result = []  # portion of reversal built so far
    k = len(x) - 1
    while k >= 0:
        # inv: k is index of next thing to append
        result.append(x[k])
        k -= 1
    return result
```

or

```
    result = []  # portion of reversal built so far
    for litem in x
       result.insert(0, litem)
   return result
```

or

```
    result = []  # portion of reversal built so far
    j = len(x)
    while j > 0:
        # inv: [j..len(x)-1] have been reversed into result
        result.append(x[j-1])
        j -= 1
    return result
```

Other solutions are possible, such as swapping within a copy of `x`.

Did you write your name and netID on each page, and re-read all specs?
Then, have a great summer break!