Last Name: UTION First Name: SOL Cornell NetID: CS1110

Solution: CS 1110 Prelim 1 March 15th, 2016

1. [10 points] Write the body of the function below so that it satisfies the given specification by making effective use of a loop. Omit its doc string.

   Specification for `stopsby1(inst, targetfloor)`: Imagine an elevator that starts at floor 0 of a building that goes up to floor $+\infty$ and down to floor $-\infty$. `inst` is a non-empty string made up of 'U' and 'D' characters, where 'U' means the elevator goes up one floor, 'D' means it goes down one floor.

   The function returns `True` if the elevator is ever at floor `targetfloor`, where `targetfloor` is an int. The function returns `False` otherwise.

   Example input and output pairs:

```
'UUU', 0 --> True           'UDUDUD', 2 --> False
'UUU', 1 --> True           'UUUDDD', 2 --> True
'UUU', 3 --> True           'UDDUDDUDDU', -3 --> True
'UUU', 4 --> False
```

```
def stopsby1(inst, targetfloor):
```

Solution:

```
def convert(c):
    """return 1 if c is 'U', -1 if c is 'D'.  PreC: c is one of 'U', 'D'"""
    return 1 if c=='U' else -1


def stopsby1(inst, targetfloor):

                                    # RUBRIC FOR NEARLY-RIGHT SOLUTIONS
    if targetfloor == 0:            # +1 for correctly checking if targetfloor = 0
        return True                 #   (while loops may do this implicitly)

    floor = 0                       # +1 for tracking the current floor
    for i in inst:                  # +3 for going through items in inst
                                    #   (-1 for while loops without increment)
                                    #   (-1 for while loop with wrong stop)
                                    #   (-1 for while loop not converting index to char)

        floor += convert(i)         # +2 for if-then reasoning on U and D
        if floor == targetfloor:    # +1 for check of targetfloor for True
            return True
    return False
                                    # +1 for returning False on obvious cases
                                    # +1 for not returning wrong value too early
```

```
SPECIAL CASES
-4: if/then/return logic faulty
-4: just return last floor (is similar to above, in a way)
-3: trying to record the floor in a string (think about double-digit floors)
-1: major syntax error (causing incorrect results). max of -2 for multiple
    such errors
-1: missed one of the returns
-2: rewriting inputs somehow - shows misunderstanding of spec or parameters


def stopsby2(inst, targetfloor):
    floor = 0
    ind = 0 # next index to check in inst
    while (floor != targetfloor) and (ind < len(inst)):
        floor += convert(inst[ind])  # NOTE: inst[ind] is "converting index to char"
        ind+=1
    return floor == targetfloor


# an alternative for-loop
def stopsby3(inst, targetfloor):
    floor = 0
    for i in inst:  # i here means: next instruction to check IF not at target
        if floor == targetfloor:
            return True
        floor += convert(i)
    return floor == targetfloor
```

2. (Booleans)

   (a) [3 points] Assign Boolean values to `B1`, `B2`, and `B3` so that the values assigned to `C1` and `C2` are not the same.

   Hint: you can just mechanically try all possible values for the B variables. Or, you could take this approach: look carefully at the expression for C1 and see if you can force it to have a certain value by setting the values of just one of the B variables; do the same for C2.

```
B1 = ______________________________________

B2 = ______________________________________

B3 = ______________________________________

C1 = (B1 or B2) and B3

C2 = B1 or (B2 and B3)
```

Given your choices for `B1`, `B2`, and `B3`, what values are assigned to `C1` and `C2`?

```
C1 is _____________________________________

C2 is _____________________________________
```

Solution: -1: B1,B2 and B3 aren't of Boolean type. (ints are not Boolean values)
no credit if use a Boolean "expression" where all variables aren't defined
no credit if C1, C2 don't demonstrate you know what Boolean operations do
Looking at C1, we notice that if B3 is False, then C1 is False no matter what.
Similarly, looking at C2, we notice that if B1 is True, then C2 is True no matter what.
In fact, the only possible correct answers are when B1 is True and B3 is False.
The full "truth table" is as follows (and yes, we used loops and Python to create it):

```
B1 B2 B3    C1 C2    C1 != C2?
 T  T  T     T  T          F
 T  T  F     F  T          T  <-----
 T  F  T     T  T          F
 T  F  F     F  T          T  <-----
```

F	T	T	T	T	F
F	T	F	F	F	F
F	F	T	F	F	F
F	F	F	F	F	F

(b) [2 points] Consider the following code.

```
x = input('Enter x')
y = input('Enter y')
if 1<=x<=3 and 1<=y<=3:
    print 'A'
elif x>3:
    print 'B'
elif y<1:
    print 'C'
```

Complete the following table with x-values and y-values so that the specified output is achieved. We've completed the first row for you.

x	y	Output
2	2	A
		B
		C

Solution:
+1: first row has any $x > 3$
+1: second row has any $x \leq 3$ and $y < 1$

3. [5 points] While-loops.

Consider the following function:

```
def F(s):
    """
    PreC: s is a string that contains at least one 'N'
    """
    x = ''
    i = s.find(x+'N')

    while i>=0 :
        print i
        x = x + 'N'
        print x
        i = s.find(x+'N')
    print i
    return len(x)
```

What is printed out if we call `F('ENSNNNW')`? (No explanations necessary.)

Solution: Note that `x` represents the longest sequence of 'N's found so far, and `i` represents the position of next longest sequence (-1 if not there).

```
1
N
3
NN
3
NNN
-1
```

Update, April 3: It was pointed out that the question did not specify where the call happened, and that in interactive Python mode, returned values get printed. After some discussion after the grading session, we decided that students could not assume that the call happened in interactive mode, and would need to give the fully generic solution given above — that is, without the printout of the returned value.

Grading:

+2 for the right numbers. (-1 for each conceptual error, don't go negative)

+2 for an increasing sequences of Ns (-1 if don't stop at right place)

+1 for not printing out anything after the -1 (because of the return statement)

SPECIAL CASES:

If you only said "3" BUT had reasoning/comments/sidework that showed that you understood completely the while loop, you get 4 out of 5. (You **did** still have a conceptual error: when you call a function, all of its print functions print things out.)

4. [8 points] Function calls.

Consider the following python file.

```
def  F(x,y):
    z = x + y
    return z

if __name__ == '__main__':
    x = 2
    y = 3
    z = 'x+y'
    x = F(z,'x')
```

Fill in the boxes below with the values that would result by executing the Python file above. To get you started, we've done the first box for you.

For the Application Script, before the call to F, we have:

x: [2] y: [3] z: ['x + y']

+1 for y-box

+1 for z-box (OK if no quotes)

Then comes the call to F.

In F's call frame, x: ['x+y'] y: ['x'] z: ['x+yx']

+2 if x-box match's global z-box; all or none

+1 for y-box (no quotes OK)

+1 for z-box consistent with x, y, and what + does

Back to the Application Script after the call to F, we have

x: ['x+yx'] y: [3] z: ['x + y']

+1 x-box same as F's z-box

+1 if the y-box and z-box are same as in first row; all or none

SPECIAL CASES -3: string/variable-evaluation misunderstanding; Additional -1 if also have variable names (misunderstanding expression evaluation)

5. [5 points] String processing.

Complete the implementation of the following function:

```
def F(s):
    """ Returns an int that is the value of the integer specified by the characters
    that are between the two slashes in s.

    Example: if s is '234/05/65', F should return the int 5.

    PreC: s is a string that is made up of characters from '0123456789/'.
    It contains exactly two slashes and they are not next to each other."""
```

Solution:

```
    n1 = s.find('/')        # +1: locate first slash (perhaps implicitly)
    n2 = s.rfind('/')       # +1: locate second slash
    t = s[n1+1:n2]          # +1: locate stuff between the two slashes.
                            #    (no point if off by one)
    u = int(t)              # +1: convert to int
    return u                # +1: return (-1 if print)
```

Alternate one-liner: return int(s[s.find('/')+1:s.rfind('/')])

SPECIAL CASES: -2 if made a call to raw_input (i.e., ignored given argument value)

6. [7 points] The following script displays 1000 random rectangles in the figure window:

```
from SimpleGraphics import *
from random import randint as randi
from random import uniform as randu

N = 1000
m = 10
MakeWindow(m+1)
# Comment 1
for k in range(N):
      x = randu(-m,m); y = randu(-m,m); L = randu(0,1); W = randu(0,1)
      i = randi(1,2)
      if i%2==0:
            fillcolor=CYAN
      else:
            fillcolor=MAGENTA
      DrawRect(x, y, L, W, FillColor=fillcolor)
# Comment 2
ShowWindow()
```

If run as is, approximately half of the displayed rectangles will be magenta.

On the next page, write a different version of the code between the two comments so that if the new version of the script is run,

1. about two-thirds of the displayed rectangles will be magenta, and
2. a new variable `AreaAve` is assigned the average area of *all and only* the magenta rectangles.

We have provided a template for your answer on the next page —just insert the required code in the blank areas. For your information, `randi(m,n)` returns a random integer with the property that `m<=randi(m,n)<=n` is `True`.

Write your answer in the template on the next page.

```
# Comment 1
```

```
for k in range(N):
      x = randu(-m,m); y = randu(-m,m); L = randu(0,1); W = randu(0,1)




      if                                    :

            fillcolor=CYAN




      else:
           fillcolor=MAGENTA




      DrawRect(x,y,L,W,FillColor=fillcolor)




# Comment 2
```

Solution:

```
# number of magenta rectangles
count_m = 0

# sums of areas of magenta rectangles.
# Note: forced to be float.
sum_m_areas = 0.0

for k in range(N):
    x = randu(-m,m); y = randu(-m,m); L = randu(0,1); W = randu(0,1)
```

```
    i = randi(1,3)
  if i==3:
        fillcolor=CYAN
    else:
        fillcolor=MAGENTA
        count_m += 1
        sum_m_areas += L*W
    DrawRect(x, y, L, W, FillColor=fillcolor)
AreaAve = sum_m_areas/count_m

### grading

+4: correctly updating number of magenta triangles and sum of the areas
   Potentially common deductions:
   -2 if reinitialize variables sum or count inside the loop
   -1 if they don't increment one or more of these quantities (e.g.,
        sum_m_areas = L*W instead of += L*W)
   -1 if one or more of these are done outside the for-loop
   -1 if their update doesn't depend on whether the color is magenta or not
      (i.e, update done without if-stmt or in wrong place w.r.t if-stmt,
       or, if they end up dividing by k to get the average)
   (don't assign negative points overall!)

+2: correctly altering random choice (randi and if-stmt based on it.)
   Don't deduct points if something small is syntactically wrong with call to randi


+1: compute the average correctly (in the right place)

Note: because randu happens to return a float, it turns out not to be necessary
to force float conversion on the sum of the areas.
```

7. [1 point] Write your last name, first name, and Cornell NetID at the top of *each* page, and circle your lab time on the first page.

   Solution: Every time a student doesn't do this, somewhere, a kitten weeps.

   More seriously, we sometimes have exams come apart during grading, so it is actually important to write your name on each page. Also, remember that if we need to figure out your lab section at the end of the grading session, our chances of putting it the wrong pile, and thus you not being able to find it when you get to lab, grow high.