15-112
Summer 2019 Midterm
June 19th, 2019

Name:

Andrew ID:

Recitation Section:

- You may not use any books, notes, or electronic devices during this exam.
- You may not ask questions about the exam except for language clarifications.
- Show your work on the exam to receive credit.
- You may complete the problems in any order you'd like; you may wish to start with the free response problems, which are worth most of the credit.
- All code samples run without crashing unless we state otherwise. Assume any imports are already included as required.
- Do not use these concepts: sets, dictionaries, recursion, classes/OOP

Don't write anything in the table below.

Question	Points	Score
1	15	
2	10	
3	10	
4	20	
5	20	
6	25	
Total:	100	

1. **Code Tracing**

   Indicate what each piece of code will print. Place your answer (and nothing else) in the box below each piece of code.

   (a) (5 points) CT1

```
def ct1(n):
    a, b = 0, 0
    while n > 0:
        d = n % 100

        if d % 3 == 0:
            a = a*100 + d
            print("funky")
        else:
            b = b*10 + d%10
            for i in range(d, 5, -4):
                print(i, end = "-")
            print("rad")

        print(a, b, n)
        n //= 10**(d%2 + 1)


n = 62511827
ct1(n)
```

(b) (10 points) CT2

```
import copy
def ct2(a):
    b = a
    c = copy.copy(a)
    d = copy.deepcopy(a)

    b += [112]
    a = a + [42]
    d[1].extend(c[0])
    c[0] = [4, 2]
    b[0][1] = "husky"
    c[1][0] = "pug"
    c[0][0] = 10
    c[1].insert(1,3)
    b.append("corgi")

    print("a =", a)
    print("b =", b)
    print("c =", c)
    print("d =", d)
a = [ [1, 3], [8, 9] ]
print(ct2(a))
print("z =", a)
```

2. **Reasoning Over Code**

For each function, find parameter values that will make the function return True. Place your answer (and nothing else) in the box below each block of code.

(a) (3 points) ROC1

```
def roc1(x):
    assert(10**2 > x > 10**0)
    y = 0
    for z in range(x):
        y += z
    return (y == 15)
```

(b) (7 points) ROC2

```
def roc2(s, t):
    assert(type(s) == type(t) == str)
    z = 0

    for i in range(len(t)):
        if s.count(t[i]) != t.index(t[i]) + 1:
            return False
        z += t[i:].count(t[i])

    return len(s) == 2*len(t) and z == 3
```

3. **Short Answer**

Answer each of the following *very briefly*.

(a) (1 point) Write one python expression that uses short circuit evaluation.

(b) (2 points) The following function has many style violations. List **two different style rules** from the 15-112 style guide that this code violates, and **circle where each rule is violated in the code.** If you write more than two style rules, only the first two will be graded. You don't need to worry about what the function does!

```
def count_solo_chars(s):
    omg = 0
    for c in s:
        if s.count(c) != 1:
            pass
        else:
            omg += 112
    return omg
```

(c) (3 points) Recall the sideScroller demo from the notes: We have a player that can move horizontally within a larger world. Whenever the player gets close to the left or right sides of the canvas, the world scrolls along with the player. There are also 5 walls in the world, each with their own score, drawn as rectangles on the screen. Whenever the player overlaps with one of the rectangles, the score for that rectangle increases by one, and that wall is drawn in a different color. For the following components of this animation, mark whether these components should be considered as part of the Model, View, or Controller.

1. Pressing the arrow keys moves the player left and right
   ◯ Model ◯ View ◯ Controller
2. The scores of each wall are stored as a list of integers
   ◯ Model ◯ View ◯ Controller
3. The player is drawn as a blue oval
   ◯ Model ◯ View ◯ Controller
4. The amount we have scrolled so far (`scrollX`) is stored as an integer
   ◯ Model ◯ View ◯ Controller
5. When we move the player, we identify which wall the player is intersecting with
   ◯ Model ◯ View ◯ Controller
6. The wall the player is overlapping with is drawn in orange
   ◯ Model ◯ View ◯ Controller

(d) (2 points) The following is a buggy implementation of isPrime. Write one test case that, if run on this function, would crash, thus correctly identifying that this is a buggy function! Then, in one sentence, write how you would fix the above code.

```
import math
def isPrime(n):
    if (n < 2): return False
    for factor in range(2, int(math.sqrt(n))):
        if (n % factor == 0):
            return False
    return True
```

```
assert(isPrime(          ) ==                )
```

How would you fix this code?

(e) (2 points) The following code snippet is an almost-complete solution to `wordSearchFromCellInDirection`, from the wordSearch problem in the course notes. Fill in the blanks with the missing code so that this function works correctly.

```
def wordSearchFromCellInDirection(board, word, startRow, startCol, dir):
    (rows, cols) = (len(board), len(board[0]))
    dirs = [ (-1, -1), (-1, 0), (-1, +1),
             ( 0, -1),          ( 0, +1),
             (+1, -1), (+1, 0), (+1, +1) ]
    dirNames = [ "up-left"  ,   "up", "up-right",
                 "left"     ,         "right",
                 "down-left", "down", "down-right" ]
    (drow,dcol) = dirs[dir]
    for i in range(len(word)):
        row = _________________________________
        col = _________________________________
        if ((row < 0) or (row >= rows) or
            (col < 0) or (col >= cols) or
            ______________________________________):
            return None
    return (word, (startRow, startCol), dirNames[dir])
```

4. (20 points) **Free Response: nthPyramidalPrime**

   A *pyramidal* number (a coined term) is a number with an odd number of digits, that has strictly increasing digits for the first half of the number, and strictly decreasing digits for the second half of the number. For example, 12321 is pyramidal, as is 251. 1221 is **not** pyramidal because it has an even number of digits. 12331 is **also not** pyramidal since the digits are not **strictly** decreasing in the second half of the number.

   Write the function `isPyramidalPrime(n)` that takes in an integer n, and returns True if that number is both prime and pyramidal, and False otherwise. Then, write `nthPyramidalPrime(n)`, that takes an integer n and returns the nth pyramidal prime. `nthPyramidalPrime(0)` should return 2.

   You may not use strings, or any string-type method in this problem. Specifically, you may not call len(n) to access the number of digits in n. However, you may assume isPrime(n) is already written for you.

Additional Space for Answer to Question 4

5. (20 points) **Free Response: findThePyramids**

   Given a 2D list (grid) of 0's and 1's, write the function `findThePyramids(grid)` that finds all the pyramids made out of 1's, and returns a list of their tips (the top center of the pyramid), as a list of tuples of their row and col. A pyramid will always be contained in a 3x2 box, and has 1's in the top center, lower left, lower center, and lower right. The other values in the 3x2 box don't matter - they can be 0's **or** 1's!

   For example, consider the following grid:

```
[ [ 0, 0, 0, 0, 0 ],
  [ 0, 0, 1, 0, 0 ],
  [ 0, 1, 1, 1, 0 ],
  [ 0, 0, 1, 1, 1 ] ]
```

   This grid has two pyramids - one with its tip at row 1 and column 2, and the other with its tip at row 2 and column 3, so `findThePyramids` should return `[(1, 2), (2, 3)]`. The order of the returned list doesn't matter - the tuples can appear in any order. If there are no pyramids in the grid, the function should return the empty list.

   You are guaranteed that the grid will be rectangular, and that it has at least 2 rows and 3 columns.

Additional Space for Answer to Question 5

6. (25 points) **Free Response: catchThePyramids**

Assuming the `run()` function is already written for you, write `init, keyPressed, mousePressed, timerFired, and redrawAll` so that an animation is created with the following properties:

- A grid with 10 rows and 5 columns is drawn that fills the entire screen. **Note: you cannot hardcode the width and height of the canvas.**
- Every 2 seconds, a *pyramid* is spawned in a random cell. It should be spawned in a random row **in the top half of the grid**, and in a random col.
- Pyramids should be drawn as pink triangles, with one point in the top center of the cell, one point in the bottom left corner of the cell, and one point in the bottom right corner of the cell.
- A score should be displayed in the top left corner of the canvas, starting at 0. (Don't worry about the score overlapping with the grid - that is OK!)
- Every 0.5 seconds, all the pyramids should move one row down.
- If the user clicks inside a cell that contains a pyramid, that pyramid should be removed from the screen.
- If any of the pyramids reach the bottom row of the grid, the game is over! Instead of drawing the grid, the text "Game Over!" should appear in the center of the screen.
- When the "r" key is pressed, the game should start over with a score of 0, and no pyramids currently present on the grid.

To save time, you may to abbreviate data as d, event as e, and canvas as c.

Note: you cannot assume any code from the course notes has been written for you. In particular, if you want to use any of the logic from functions like `getCell` and `getCellBounds`, you must write that code yourself.

Additional Space for Answer to Question 6

Additional Space for Answer to Question 6