15-112 Midterm #1a – Spring 2017
80 minutes

Name: _____

Andrew ID: _____@andrew.cmu.edu

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 (not scratch paper) to receive credit.**
- **If you use scratch paper, you must submit it with your andrew id on it, and we will ignore it.**
- **All code samples run without crashing. Assume any imports are already included as required.**
- **Do not use these post-midterm1 topics/constructs: sets, maps, recursion, or classes/OOP.**

DO NOT WRITE IN THIS AREA		
Part 1 (CT)	15 points	
Part 2 (RC)	10 points	
Part 3 (SA)	10 points	
Part 4 (FR / squarish)	20 points	
Part 5 (FR / getCowMove)	20 points	
Part 6 (FR / squareClick)	25 points	
Part 7/bonus	5 points bonus	
Total	100 points	

1. **[15 pts] Code Tracing**

Indicate what each will print. Place your answer in the boxes below each block of code.

```
def ct1(a):
    L, M, N = a, copy.copy(a), copy.deepcopy(a)
    L[0][0] += 2
    M[1] += [3]
    M[1] = M[1] + [4]
    N[1] = L[0]
    N[1][0] = N[1][0] + 5
    return (M, N)
L = [[1], [ ]]
M, N = ct1(L)
for A in [L,M,N]: print(A) # prints 7 int values
                          # (be careful with your brackets and newlines)
```

```
def ct2(s, t):
    r = ''
    for c in s.upper():
        if (c in t):
            r += c
            i = t.find(c)
            t = t[:i] + t[i+1:]
        else:
            t = t.replace(t[0], chr(ord(c)+3))
    return t+'x'+r
print(ct2('aDge', 'bCDEfE')) # prints a 7-letter string
```

```
# Code Tracing, continued

def ct3(a):
    L, M, n = [ ], [ ], len(a)
    for val in a[n-1 : 0 : -2]:
        L.append(val % 10)
        M += [ int(str(val)[0]) ]
        if (sum(M) > 3):
          L.append(M.pop(0))
    return L + M
print(ct3(list(range(17, 23)))) # prints a list with 6 int values
```

2. **[10 pts] Reasoning about code**
   For each function, find values of the parameters so that the function will return True. Place your answers in the box on the right of each function.

```
def rc1(n):
    assert(isinstance(n, int))
    L, M = [ ], [ ]
    while (n > 0):
        L.append(n%10)
        M.insert(0, n//10%10)
        n //= 100
    n = int(''.join([str(v) for v in M]))
    return ((sorted(L+M) == list(range(8))) and
            (n == 256))
```

n = _____

`def rc2(s):`
`  assert(isinstance(s, str))`
`  c, L = 'B', [ ]`
`  for d in s:`
`  L.append(ord(d) - ord(c))`
`  c = d`
`  L = [L[i]*10**i for i in range(len(L))]`
`  return (123 == sum(L))`
	s = _____

3. [10 pts] Short Answer

Answer each of the following very briefly.

A) Since int(True) is 1 and int(False) is 0, and bool(1) is True and bool(0) is False, we can define the following function:

```
def And(b1, b2): return bool(int(b1)*int(b2))
```

This almost works, but not quite. In some cases, given booleans b1 and b2, And(b1, b2) will not work exactly like (b1 and b2). Why not?

B) Give an example of a common MVC violation -- that is, something that is explicitly disallowed by the rules for Model-View-Controller applications.

C) Draw a picture that clearly shows where the anchor is located when drawing the text "ABC" in Tkinter with an anchor of S.

Short Answers (continued)

D) Fill in the blanks (according to our case study):

```
def sieve(n):
    isPrime = [ True ] * (n+1)
    isPrime[0] = isPrime[1] = False
    primes = [ ]
    for prime in range(n+1):

        if (_____________________________________________________________):
            primes.append(prime)
            for multiple in range(2*prime, n+1, prime):

                _____________________________________________________
    return primes
```

E) Fill in the blanks (according to our case study):

```
def selectionSort(a):
    n = len(a)
    for startIndex in range(n):

        minIndex = _____________________________________________________
        for i in range(startIndex+1, n):

            if (_____________________________________________________):
                minIndex = i
        swap(a, startIndex, minIndex)
```

4. **[20 pts] Free Response: isSquarish(n) and nthSquarish(n)**
   Note: for full credit, you may not use strings or lists for this problem. For a 5-pt deduction, you may use strings and/or lists.

   An integer m is squarish (a coined term) if and only if:
   1) m is positive, and
   2) m has at least three digits, and
   3) m contains no 0's, and
   4) each pair of consecutive digits in m are within 2, inclusive, of a perfect square.

   For example, if m == 6251.
   * 62 is within 2 of 64
   * 25 is within 2 of 25
   * 51 is within 2 of 49

   So 6251 is squarish. The first several squarish numbers are:
   111, 114, 115, 116, 117, 118, 147, 148, 149, 151, 162, 163, 164, 165, 166, 179,
   181, 182, 183, 234, 235, ...

   With this in mind, write the functions isSquarish(n) that takes an integer value n and returns True if it squarish, and False otherwise, and also write the function nthSquarish(n), that takes a non-negative int n and returns the nth squarish number.

[this page is blank (for isSquarish and nthSquarish]

5. **[20 pts] Free Response: getCowMove**

Background: this problem involves a simple game where a cow moves on a 2d board trying to get food. We represent the board as a rectangular 2d list of integers, where 0 is empty, 1 is the cow, and 2 is food. There is always exactly one cow on the board, and always several cells containing food. On each step, the cow can only move 'up', 'down', 'left', or 'right', though the cow must stay on the board at all times. Distances from the cow to its food are measured in the number of steps required to reach that food (which is different than using the distance formula). When the cow moves, it always takes one step towards the nearest food (with ties resolved any way you wish), so after that step it is one step closer to that food.

With this in mind, write the function getCowMove(board) that takes a board as described above and returns the direction the cow should move in next, according to the rules described above.

For example, say:

```
board = [ [ 0, 0, 0, 0, 0, 0, 0, 0 ],
          [ 0, 0, 1, 0, 0, 2, 0, 0 ],
          [ 2, 0, 0, 0, 0, 0, 0, 0 ],
          [ 0, 0, 0, 0, 0, 0, 0, 2 ]
        ]
```

We see that the cow is at (1,2) and food is at each of (1,5), (2,0), and (3,7). We check the distances to each food:

(1,5) is 3 from (1,2)
(2,0) is 3 from (1,2)
(3,7) is 7 from (1,2)

So the cow can move towards either (1,5) or (2,0). We can choose either one. If we choose (1,5), the cow must move 'right'. However, if we choose (2,0), then the cow can move either 'left' or 'down'. Thus, for this board, getCowMove(board) can return any one of 'right', 'left', or 'down', as noted in this test case:

```
board = [ [ 0, 0, 0, 0, 0, 0, 0, 0 ],
          [ 0, 0, 1, 0, 0, 2, 0, 0 ],
          [ 2, 0, 0, 0, 0, 0, 0, 0 ],
          [ 0, 0, 0, 0, 0, 0, 0, 2 ]
        ]
assert(getCowMove(board) in ['right', 'left', 'down'])
```

Write your answer on the following page.

[this page is blank (for getCowMove)]

6. **[25 pts] Free Response: squareClick Animation**

Assuming the run() function is already written for you, write init, keyPressed, mousePressed, redrawAll, and timerFired so that when the animation is first run:

A. A small square is centered in the canvas.

B. A score of 0 is displayed near the left top corner of the canvas.

Game play proceeds as such:

C. The square stays centered, but grows larger (at some reasonable speed) until it fills the window, then it shrinks again until it is tiny, and then it grows again, and shrinks again, and so on.

D. Each time the user presses the mouse, if it is within a distance of 20 to the nearest corner of the square, the score increases by 1 point. Otherwise, the score decreases by 1 point, unless it would be negative, and then it just stays at 0.

E. Each time the user presses the Up arrow, the square should grow and shrink faster (twice as fast).

F. Each time the user presses the Down arrow, the square should grow and shrink slower (half as fast).

G. Each time the user presses 'p', if the game is unpaused (as it is at the start), the square pauses, but mouse presses are still processed normally. If paused, pressing 'p' unpauses and the square resumes its growing and shrinking.

Make reasonable assumptions for anything not specified here, and in any case avoid hardcoding values (such as data.width, data.height, or data.timerDelay). Also, while the square sometimes moves at different speeds, you must only use one timer and only one timerFired function.

[this page is blank (for squareClick))

[this page is blank (for squareClick))

[the top of this page is blank

Bonus/Optional: **[2.5 pts] What will this print?**

```
def bonusCt1(s,b):
    def f(s):
        try: return int(str(s)+'0', b)
        except: return s
    return s if (b < 2) else f(bonusCt1(s,b-1))
print(bonusCt1(3,8))
```

Bonus/Optional: **[2.5 pts] What will this print?**

```
def bonusCt2(n, expr='1', ops='+-*/'):
    for i in range(2,n): expr += '%s%d' % (ops[(i-2)%4], i)
    return eval(expr.replace('/','//'))
print(bonusCt2(101))
```