15-112 Midterm #2a – Spring 2016
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.**
- **When problems specify to use recursion, you must use it meaningfully. In such cases, you may use wrapper functions or add optional parameters if it helps.**

DO NOT WRITE IN THIS AREA		
Part 1 (CT)	10 points	
Part 2 (RC)	5 points	
Part 3 (DB)	5 points	
Part 4 (SA)	16 points	
Part 5 (FR)	16 points	
Part 6 (FR)	16 points	
Part 7 (FR)	16 points	
Part 8 (FR)	16 points	
Part 9/bonus	5 points bonus	
Total	100 points	

1. **[10 pts] Code Tracing**
   Indicate what these will print:

Statement(s):	**Prints:**

```
def ct1(L):
    if (len(L) % 3 < 2):
        return L
    else:
        k = len(L)//2 + 1
        return [sum(L), ct1(L[:k])]
L = [-1, 0, 1, 2] * 2
print(ct1(L)) # Hint: prints 5 integers overall
              # Be sure to show the nested
              # lists properly
```

Statement(s):	**Prints:**

```
def ct2(n):
    def f(n):
        if (n < 10):
            return (n, n)
        else:
            (a, b) = f(n//10)
            d = n%10
            if (d > a): a = d
            elif (d < b): b = d
            return (a, b)
    return f(n)
print(ct2(3276))  # Hint: prints a tuple of
                  #       2 integers
```

2. **[5 pts] Reasoning about code**
For this function, find values of the parameters so that the function will return True. Place your answers in the box on the right of the function.

`def rc1(s):`
`    def f(s):`
`        if (len(s) < 2): return s`
`        else: return s[-1] + f(s[:-1])`
`    t = ""`
`    while (s != ""):`
`        t += s[0]`
`        s = f(s[1:])`
`    return (t == "Yes!")`
	s = ______________________

3. **[5 pts] Debugging**
You are provided with an example from the course notes. This function has one bug added or one line deleted, however. Find and circle the bug or missing line in the code, being sure to circle just the code that has the bug and no additional code. Then, fill in the box below the function with code to replace the code you circled so this function works correctly.

```
def printFiles(path):
    if (os.path.isdir(path) == False):
        # base case:  not a folder, but a file, so print its path
        print(path)
    else:
        # recursive case: it's a folder
        for filename in os.listdir(path):
            printFiles(filename)
```

4. **[16 pts] Short Answer**
   Answer each of the following very briefly.

   A. When you write a new class in Python, how do you define how the == operator works over instances of that class?

   B. In the course notes, we provide a default hash method with this line:

   ```
   return hash(self.getHashables())
   ```

   This may appear to be recursive, because our hash method is calling hash again. However, it is not recursive. Very briefly, why not?

   C. In the following, assume L is a list, and functions g and h are both defined already:

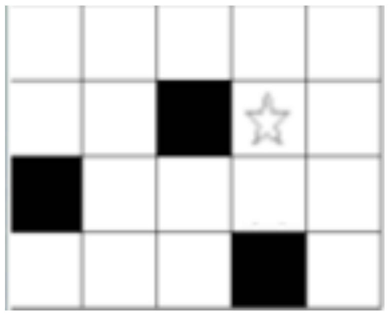
   ```
   def f(L):
       if (len(L) == 0): return g(L)
       else: return h(L[0]) + f(L[1:])
   ```

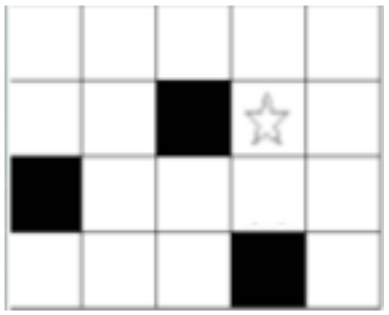
   Rewrite the function f(L) so that it works the same, only without using any recursion. You may use iteration here.

D. Why does memoization speed up fib(n) so much more than fact(n)?

E. If a level-0 Sierpinski Triangle is a solid black triangle, draw a level-2 Sierpinski Triangle.

F. Say we start with a 4x5 board and fill the black squares as shown below (in the code from our notes, the black squares would be green, and the white squares would be cyan). Then we right-click where the star is to start a floodFill from there. On the left, write the numeric labels for the depths that result in each cell after the floodFill completes. On the right, write the numeric labels for the ordinals.


depths


ordinals

Hint #1: the numbers on the left are depths, so some labels may occur more than once.
Hint #2: the numbers on the right are ordinals, so labels must occur only once each.
Hint #3: the first label is 0, not 1 (so a 0 should be where the star is).
Hint #4: Our floodFill code recursively tries to go up, then down, then left, then right.

5. **[16 pts] Free Response: makeTriangle(n)**
   Using recursion meaningfully, and **without using iteration or string multiplication**, write the function makeTriangle(n) that takes a positive integer n and returns a multi-line string that is a triangle of asterisks, where the first line contains n asterisks. Here is a sample test case for you:
   ```
   assert(makeTriangle(3) == "***\n**\n*")
   ```
   Reminder: while you cannot use iteration, you may use recursive helper functions here if you wish.

6. **[16 pts] Free Response: containsSelf(L)**

Background: in Python, it is possible for a list to contain itself. Not a copy of itself, but an actual alias to itself. For example:

```
L = [ ]
L.append(L)      # we just added an alias of L to itself!
print(L[0] is L) # True!
```

With this in mind, write the function containsSelf(L) that takes a list L and returns True if L contains an alias to itself at any level (including sublists of L, sublists of sublists of L, etc), and False otherwise.

7. **[16 pts] Free Response: Playlist and Song classes**

Write the Playlist and Song classes so the following test code works. You may not use the Struct class from the course notes, and you may not hardcode the specific test cases.

```
song1 = Song("Happy", 10)
assert((song1.title == "Happy") and (song1.rating == 10))
song2 = Song("Joyful")
assert((song2.title == "Joyful") and (song2.rating == 6))
playlist1 = Playlist([song1, song2])
assert(playlist1.songs == [song1, song2])
assert(playlist1.averageRating() == 8) # average of 10 and 6
playlist2 = Playlist([song1])
assert(playlist2.songs == [song1])
assert(playlist2.averageRating() == 10)
assert(not (playlist1 == playlist2))
playlist2.addSong(song2)
assert(playlist2.songs == [song1, song2])
assert(playlist2.averageRating() == 8)
assert(playlist1 == playlist2)
```

[this page is blank (for Playlist and Song classes)

8. **[16 pts] Free Response: increasingPathCount(board)**

Background: we will say that a board is a square 2d list of integers. As with mazes, a solution is a path from the left-top to the right-bottom, only here we will only allow moves that are up, down, left and right (no diagonals). A solution is an "increasing path" (a coined term) if each value on the solution path is strictly larger than the one before it on that path. Consider this board:

```
board = [[ 1, 3, 2, 4 ],
         [ 0, 4, 0, 3 ],
         [ 5, 6, 8, 9 ],
         [ 0, 7, 8, 9 ]]
```

This board has exactly one increasing path: right to 3, down to 4, down to 6, down to 7, right to 8, right to 9.

With this in mind, write the function increasingPathCount(board) that takes such a board and returns the number of increasing paths running from the left-top to right-bottom in that board. For the example board above, your function would return 1. Similarly, consider this board:

```
board = [ [3, 5],
          [4, 7] ]
```

For this board, your function would return 2 -- those paths being right,down and also down,right.

For full credit, you need to use backtracking, so that you do not explore every possible path, but instead you backtrack once you know a subpath cannot lead to a solution.

Important note: if you cannot get your function to properly return the counts, then for most-but-not-all of the credit you can instead return True if there are any solutions (that is, the count is positive) and False otherwise (that is, the count is 0).

[this page is blank (for increasingPathCount))

9. **Bonus/Optional:**

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

```
def bonusCT1(g=lambda a:a+2, a=3):
    def f(s,n=10): return "f("*n + str(s) + n*")"
    (s, f) = (f(f(f("a", 5))), g)
    return(eval(s))
print(bonusCT1())
```

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

```
def bonusCT2(x, y, L=None):
    class A(object):
        def __init__(s, n): s.n = n
        def __eq__(s, t):
            try: t = t.n; L.append(t); return (s.n==t or s==A(t//10))
            except: return False
    L=L or []; return(A(x) == A(y) and L)
print(bonusCT2(5, 567), bonusCT2(567, 5))
```