15-112
Fall 2017 Midterm Exam 2a
November 16, 2017

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 use the backs of pages as scratch paper. Nothing written on the back of any pages will be graded.
- 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.

Don't write anything in the table below.

Question	Points	Score
1	10	
2	10	
3	10	
4	10	
5	20	
6	20	
7	20	
Total:	100	

1. **Code Tracing**

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

   (a) (5 points) CT1

```
class A(object):
    def __init__(self, x):
        self.x = 2 * x

    def __str__(self):
        return "A(" + str(self.x) + ")"

    def add(self, n):
        return self.x + n

class B(A):
    def __init__(self, x, y):
        super().__init__(x)
        self.y = 3 * y

    def sub(self, n):
        return self.x - n

def ct1():
    try:
        a, b = A(1), B(3, 4)
        print(a)
        print(b)
        print(b.add(4))
        print(a.sub(2))
        print("All good!")
    except:
        print("Uh oh!")
ct1()
```

(b) (5 points) CT2

```
def ct2(x):
    def fDec(f):
        def g(x):
            return f(x//2)
        return g

    @fDec
    def myFunc(x):
        print(x)
        if x > 200:
            return 0
        elif x == 1:
            return 1
        else:
            return x + myFunc(x)

    print("ret =", myFunc(x))
ct2(30)
```

2. **Reasoning Over Code**

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

(a) (5 points) ROC1

```
def roc1(L):
    x, y = 0, 0
    S = set(L)
    for n in S:
        if n % 8 in S and n >= 8:
            x += 1
        elif n % 2 == 0:
            y += 2
    return sum(S) != sum(L) and (x,y) == (3,2) \
           and len(L) < 10 and 0 not in S
```

(b) (5 points) ROC2

```
def roc2(d):
    if not isinstance(d, dict):
        return False
    x, y, z = 0, 0, 0
    for k in d:
        x += 1
        if d[k] in d:
            y += 1
        if k == d[k][::-1]:
            z += 1
    return ((x, y, z) == (3, 1, 1))
```

3. **Short Answer**

Answer each of the following *very briefly*.

(a) (2 points) List one benefit and one drawback of using a class attribute instead of a normal instance attribute.

(b) (3 points) Is it a good idea to use a list as the value of a default parameter? Why or why not?

(c) (2 points) When writing a recursive function that deals with file systems, which part of the system represents our base case, and which represents our recursive case?

(d) (3 points) You may recall from the notes that selection sort is $O(n^2)$ and merge sort is $O(nlogn)$. In certain situations, however, a good implementation of selection sort can be faster than a good implementation of merge sort. How can this be?

4. (10 points) **Big-Oh**

What is the big-oh runtime of each of the following, in terms of N?

```
# L is a list of N elements
def bigOh1(L, item):
    L.sort()
    check1 = item in L
    check2 = L.count(item) % 2 == 0
    return check1 and check2
```

```
# S is a set of N strings
def bigOh2(S):
    count = 0
    i = ord('a')
    while i <= ord('z'):
        if chr(i) in S:
            count += 1
        i += 1
    return count
```

```
# L and M are lists of N integers
def bigOh3(L, M):
    R = []
    for item in L:
        if item in M:
            R.insert(0,item)
    return R
```

```
# N is an integer
def bigOh4(N, d=0):
    if N == 1:
        return d
    else:
        return bigOh4(N//10, d+1)
```

5. (20 points) **Free Response: Passphrase Generator**

   In security, a passphrase is like a password, except it is very long and made up of randomly chosen words. For example, "purplemonkeydishwasher" is a passphrase made up of three words: purple, monkey, and dishwasher.

   Write the recursive function `passphraseGenerator(words, k)` that, given a list of unique words `words` and an integer `k`, prints out all possible passphrases with `k` or fewer words from `words` in it. In addition, a generated passphrase should not contain the same word more than once.

   For example, `passphraseGenerator(["apple", "orange", "pear"], 2)` prints out:

```
apple
appleorange
applepear
orange
orangeapple
orangepear
pear
pearapple
pearorange
```

   Notes/Hints:

   1. You must use recursion to solve the problem to receive credit, even if another approach would work.
   2. You may make use of wrapper functions and/or optional arguments.
   3. The order of the printing is not important. Your solution is allowed to print the passphrases in a different order.
   4. You may assume that `words` will not contain any duplicates.
   5. Your function doesn't need to return anything, it should print the passphrases.

Additional Space for Answer to Question 5

Additional Space for Answer to Question 5

6. (20 points) **Free Response: Radio and Channel**

Write the classes Radio and Channel so that the following test code runs without errors. Do not hardcode against the values used in the testcases, though you can assume the testcases cover the needed functionality.

```
# A Channel has a frequency (between 88 and 108), a name, and a playlist
ch1 = Channel("WESA", 90.5, ["The World", "Car Talk", "RadioLab"])
assert(ch1.getFrequency() == 90.5)
assert(str(ch1) == \
    "Channel WESA on 90.5, playlist: ['The World', 'Car Talk', 'RadioLab']")
ch2 = Channel("Bob FM", 96.9, ["Hits"])
assert(str(ch2) == "Channel Bob FM on 96.9, playlist: ['Hits']")
assert(ch2 == Channel("Bob FM", 96.9, ["Hits"]))
assert(ch2 != ch1)
assert(ch2 != "Bob FM") # should not crash!
s = set()
assert(ch2 not in s)
s.add(ch2)
assert(Channel("Bob FM", 96.9, ["Hits"]) in s)
assert(ch1 not in s)

# A Radio can be tuned to receive channels based on different frequencies
channels = { 90.5 : ch1, 96.9 : ch2 }
radio = Radio(channels, 90.55)
assert(radio.getCurrentFrequency() == 90.55)
# A Radio receives a channel if it's tuned within 0.05 Hz of that channel's
# frequency. You're guaranteed that channel frequencies will always be
# at least 0.05 Hz apart.
assert(radio.getCurrentChannel() == ch1)

# You can tune a radio by a given frequency increment.
radio.tune(9.31)
assert(radio.getCurrentFrequency() == 99.86)
assert(radio.getCurrentChannel() == None)

# A radio can't be tuned lower than 88 Hz or higher than 108 Hz.
radio.tune(20)
assert(radio.getCurrentFrequency() == 108)
radio.tune(-30)
assert(radio.getCurrentFrequency() == 88)
```

Additional Space for Answer to Question 6

Additional Space for Answer to Question 6

7. (20 points) **Free Response: Distance List**

   Write a recursive backtracking function, `distList(n)`, that, given a number n, returns a list of size $2n$ that contains all the numbers from 1 to n twice such that if x is in the list then the two xs are separated by exactly x other numbers. If such a list cannot be generated, return None.

   Consider the following examples:

```
distList(3) returns [3, 1, 2, 1, 3, 2]
distList(2) returns None
distList(4) returns [4, 1, 3, 1, 2, 4, 3, 2]
```

   Notes/Hints:

   1. You must use backtracking to solve the problem to receive credit, even if another approach would work.
   2. You may make use of wrapper functions and/or optional arguments.
   3. If you don't understand the problem, look at the examples again. Notice that the 3s always have three other numbers between them, the 2s always have two other numbers between them, etc.
   4. While there are multiple valid approaches to the problem, in ours we start with a list of size $2n$ and fill it in as we go.

Additional Space for Answer to Question 7

Additional Space for Answer to Question 7