15-112 Midterm #2a – Fall 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)	15 points	
Part 2 (RC)	5 points	
Part 3 (SA)	5 points	
Part 4 (Big-Oh)	10 points	
Part 5 (FR/firstLast)	10 points	
Part 6 (FR/A-and-B)	20 points	
Part 7 (FR/files)	15 points	
Part 8 (FR/tripleSum)	20 points	
Part 9/bonus	5 points bonus	
Total	100 points	

1. **[15 pts] Code Tracing**
   Indicate what these will print:

Statement(s):	**Prints:**

`def ct1(L):`
`  t = sum(L)`
`  if (t < 2):`
`  return L`
`  elif (len(L) % 2 == 0):`
`  return ct1(L[1:]) + [t]`
`  else:`
`  return [t] + ct1(L[:-1])`
`L = [4,3,2,1]`
`print(ct1(L)) # prints 4 integers`
	_____

Statement(s):	**Prints:**

`def f(m=0, d=2):`
`  def g(n):`
`  nonlocal m`
`  m += 1`
`  if (n <= 1): return [ n ]`
`  else: return [n*m] + g(n//d)`
`  return g`
`ct2 = f(d=3)`
`print(ct2(8))  # prints 3 integers`
`ct2 = f(m=1)`
`print(ct2(5))  # prints 3 more integers`
	_____
	
	_____

Statement(s):	**Prints:**

`def ct3(L):`
`  return (list(map(lambda x:max(L) - x,`
`  filter(lambda x: x%2==0, L))) +`
`  [reduce(lambda x,y: 2*x + y, L, 0),`
`  reduce(lambda x,y: x + 2*y, L, 0)])`
`print(ct3([2,3,4,5])) # prints 4 integers`
	_____

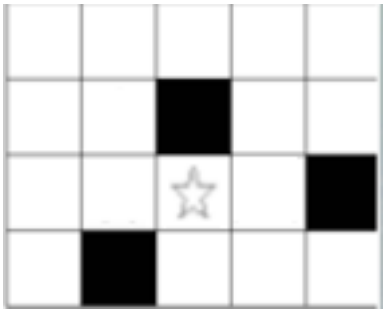
2. **[5 pts] Reasoning about code**
For this function, find values of the parameters so that the function will return True. Place your answers in the box on the right of the function.

```
def rc1(s):
    t = ord(s[0])
    def f(s, offset=0):
        try:
            offset += ord(s[0]) - t
            return f(s[1:], offset) + [offset]
        except:
            return [ ]
    return (f(s[1:]) == [10, 5, 4])
```

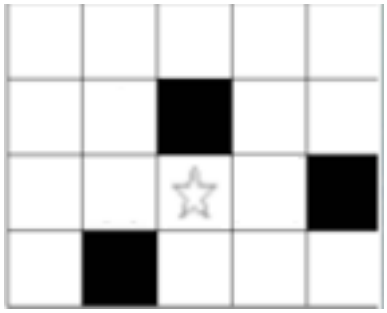
s = _____

3. **[5 pts] Short Answer (just FloodFill)**
Answer the following very briefly.

A. 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.

4. **[10 pts] Big-Oh:**
   State the Big-Oh for each of the following functions:

Function:	**Big-Oh:**

`def bigOh1(L):`
`  # assume L is a 1d list`
`  N = len(L)`
`  for val in copy.copy(L):`
`  L += [val**2]`
`  i = N`
`  while (i > 0):`
`  L[i] += i`
`  i //= 4`
`  return (sum(L) / len(L))`
	_____

`def bigOh2(L):`
`  # assume L is an NxN (square) 2d list`
`  # assume selectionSort is written as usual`
`  N = len(L)`
`  A, B = [ row*N for row in L ], [ ]`
`  for i in range(len(A)):`
`  for j in range(len(A[0])):`
`  B.append(A[i][j] + i + j)`
`  selectionSort(B)`
`  return list(filter(lambda val: val%2==0, B))`
	_____

`def bigOh3(L):`
`  # assume L is a pre-sorted 1d list (don't count`
`  # the cost of sorting L in your answer)`
`  # assume binarySearch is written as usual`
`  def f(L):`
`  N = len(L)`
`  M = [ ]`
`  for val in L: M.append(binarySearch(L, val))`
`  return M`
`  return f(f(f(L))) # note the nested calls`
	_____

`def bigOh4(n):`
`  # assume n is an integer`
`  N = math.log(n, 2)`
`  d = dict()`
`  for c in str(n): d[c] = d.get(c, 0) + 1`
`  return d`
	_____

5. **[10 pts] Free Response: firstLast decorator**

Write the function firstLast so that any function f defined with the @firstLast decorator works the same as it would have worked without the decorator, except if it would have returned a list of length 2 or greater, instead it just returns a list of the first and last values. For example:

```
@firstLast
def f(): return 42
assert(f() == 42)
@firstLast
def g(n): return list(range(n))
assert(g(1) == [0])
assert(g(10) == [0, 9]) # here it shortened the result
@firstLast
def h(*args): return [val*2 for val in args]
assert(h(1,2,3,4,5,'a','b','c') == [2,'cc']) # and here, too
```

6. **[20 pts] Free Response: A and B classes**

Write the classes A and B so that the following test code runs without errors. Do not hardcode against the values used in the example test cases, though you may assume the types of those values match the examples. For full credit, you must use proper OOP, placing methods at the right level and using inheritance appropriately.

```
assert(str(A(2,3)) == "<2 and 3>")
assert(str(A(3,2)) == "<3 and 2>")
assert(str(A(3,3)) == "<Just 3>")

assert(isinstance(B(42), A))
assert(B(42) == A(42, 42))
assert(B(42) != A(2,3))
assert(B(42) != "don't crash here!")
assert(str(B(42)) == "<Just 42>")

assert(A(1,2).plus(A(3,4)) == A(1+3,2+4))
assert(B(42).plus(A(3,4)) == A(42+3,42+4))

a1 = A(4,5)
a1.reverse()
assert(str(a1) == "<5 and 4>")
a2 = a1.reversed() # note: a1.reversed() is not a1.reverse()
assert(str(a1) == "<5 and 4>")
assert(str(a2) == "<4 and 5>")

s = set()
assert(B(42) not in s)
s.add(B(42))
assert(B(42) in s)

# Note: only B's and no other A's can call doubled:
assert(B(2).doubled() == B(4))
assert(type(B(2).doubled()) == B)
crashed = False
try: a = A(3,3).doubled() # this should crash
except: crashed = True
assert(crashed == True)
print("Passed!")
```

[this page is blank (for A and B classes)

7. **[15 pts] Free Response: selfReferencingFileCount(path)**

   Background: we will say that a text file is self-referencing if it contains its own filename (ignoring the path to that file, just the filename) anywhere within its content. So if a file is at /folderA/folderB/someFile.txt, then that file is self-referencing if its content contains "someFile.txt" anywhere in it.

   With this in mind, write the function selfReferencingFileCount(path), that you can abbreviate as sr(path), that takes a path to a folder and returns the number of self-referencing files in that folder or recursively in any sub-folder of that folder.

   You may assume that readFile(path) is already defined for you, and returns a string containing the given file's contents.

   You may not use os.walk here and you must use recursion meaningfully, but you may also use iteration if it helps.

   Hint: You may wish to use os.path.isdir(path) and os.listdir(path).

[this page is blank (for selfReferencingFileCount)

8. **[20 pts] Free Response: findTripleSum(L)**

Background: say we are given a list L of possibly-negative integers, where L is of length N and N is a multiple of 3. For example:

L = [1, 2, 3, 4, 5, 9]

We will say that M is a "tripleSum" (a coined term) of L if M contains all the same values of L, ordered such that the each of the N//3 sublists of length 3 have the same sum. For example, given L as above:

M = [1, 2, 9, 3, 4, 5]

We see:

1 + 2 + 9 == 12

3 + 4 + 5 == 12

So M is a tripleSum of L.

Here is another example:

L = [1, 1, 1, 1, 2, 2, 3, 3, 4]

M = [1, 1, 4, 1, 2, 3, 1, 2, 3]

We see:

1 + 1 + 4 == 6

1 + 2 + 3 == 6

1 + 2 + 3 == 6

So M is a tripleSum of L.

With this in mind, write the function findTripleSum(L) that takes a list L as described above, and returns a tripleSum of L, or None if none exists. If a list has more than one tripleSum, you may return any one of them. For example:

```
assert(findTripleSum([1, 2, 3, 4, 5, 9]) == [ 1, 2, 9, 3, 4, 5])
assert(findTripleSum([1, 2, 3, 4, 5, 6]) == None)
```

Hint #1: we suggest you create a second list, M, that is the same length as L but is initially full of None's. Then use backtracking to try to fill M with values from L, checking legality as you go.

Hint #2: first write the function isLegal(M) that takes a list M of integers and None values, and returns True if it is legal so far, properly taking the None values in M into account.

Note: to receive any credit on this problem, you must properly use backtracking, even if there might be another way to solve this problem. Also, in particular, you may not create every possible permutation of L, as that would take far too long.

[this page is blank (for findTripleSum))

9. **Bonus/Optional:**

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

```
def bonusCt1(n, L=[ ], f=lambda x:x%3):
    if (n <= 1): return sum(L)
    L.extend(range(n))
    L[:], M = filter(f, L), list(map(f, L))
    return bonusCt1(n//3) + bonusCt1(n//3,M)
print(bonusCt1(8))
```

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

```
def bonusCt2(n=5,z=0):
    def g(n): return 0 if (n == 0) else n%2 + g(n//2)
    def f(r=0, i=0): return r if (i == 2**n) else f(r+g(i), i+1)
    while (z < 999): z = f(z)
    return z
print(bonusCt2())
```