Name:_____ Section:____ AndrewID:_____

15-112 Fall 2021 Quiz 9b

*** Up to 20 minutes. * No calculators, no notes, no books, no computers. * Show your work!**

Code Tracing 1 [12pts]: Indicate what the following code prints. Place your answers (and nothing else) in the box to the right of the code.

```
def g(L):
    if L == [ ]:
        return ['yes']
    elif sum(L) >= L[0]:
        return ['no']
    else:
        return [L[-1]] + g(L[1:-1]) + [L[0]]

def ct1(L):
    return g(L), g(list(reversed(L)))

print(ct1([8, 3,-4,-1]))
```

Code Tracing 2 [12pts]: Indicate what the following code prints. Place your answers (and nothing else) in the box to the right of the code.

```
def ct2(n):
    if (n < 10):
        return n
    else:
        return 10*ct2(n//100) + n%10

print(ct2(57249))
```

Fill In The Blank: largestDigits [24 pts]

Note: On the following page, you are provided with a solution to this problem, with a few lines removed. While there are other ways to solve this problem, you must complete the given solution. Fill in the blanks with just one line of code each. (Also, you may not use the ";" to evade this restriction.)

With that, the problem is to write largestDigits(L), that takes a 1D list L of possibly-negative integers, and returns a set of only the largest digits from each number. For example:

```
    assert(largestDigits([12, 44, 771, 30])   == {2, 4, 7, 3})
    assert(largestDigits([17, 44, -771, -30]) == {4, 7, 3})
    assert(largestDigits([22, 44, 882, 40, 0]) == {2, 4, 8, 0})
```

You may not use iteration (for/while loops). Here is the solution, with 4 lines left for you to fill in:

```
def largestDigits(L):
    s = set()
    return largestDigitsHelper(L, s)

def largestDigitsHelper(L, s):
    if L == []:
        return s
    else:
        s.add(getLargestDigit(L[0]))

        _____________________________________________(A) # Recursively process the rest
                                                        # of the values in L
def getLargestDigit(n):
    if n < 0:

        _____________________________________________(B) # Recursively process negatives
    elif n == 0:

        _____________________________________________(C) # Base case when n is 0
    else:
        onesDigit = n % 10
        # Recursively process the rest of the digits

        partialResult = ____________________________________________________(D)
        if onesDigit > partialResult:
            return onesDigit
        else:
            return partialResult

def testLargestDigits():
    assert(largestDigits([12, 44, 771, 30])   == {2, 4, 7, 3})
    assert(largestDigits([17, 44, -771, -30]) == {4, 7, 3})
    assert(largestDigits([22, 44, 882, 40, 0]) == {2, 4, 8, 0})
    assert(largestDigits([]) == set())
testLargestDigits()
```

Free Response: Recursive indexMap(L) [52 pts]

Without using iteration or builtins that run in O(N) (like .find, or .count), write the function indexMap(L) that takes a 1D list L and returns a dictionary that maps each value in L to a set of the indexes in L where that value occurs. For example:

```
assert(indexMap([5, 6, 5]) ==  { 5:{0,2}, 6:{1} })
assert(indexMap([9, 6, 3, 6, 9]) ==  { 3:{2}, 6:{1,3}, 9:{0,4} })
assert(indexMap([3, 2, 1, 3, 2, 3]) ==  { 1:{2}, 2:{1, 4}, 3:{0,3,5} })
```

You may use helper and/or wrapper functions if you wish.

Bonus/Optional: Code Tracing [+2pts]

Indicate what this prints. Place your answer (and nothing else) in the box.

```
def bonusCt1(n):
    def f(n): return  0 if  (n  <=  0)  else  2*n-1+f(n-1)
    def g(n,  m): return  0 if  (m  ==  0)  else  f(n)+g(n,m-1)
    return  (g(n, n)  - f(n)) //  f(n)
print(bonusCt1(43))
```