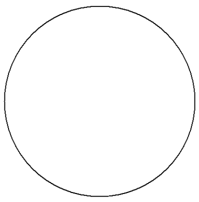
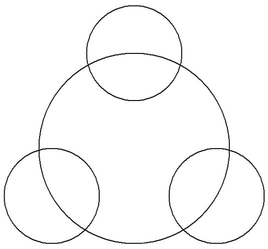
Name: ______________________ Andrew Id: ______________

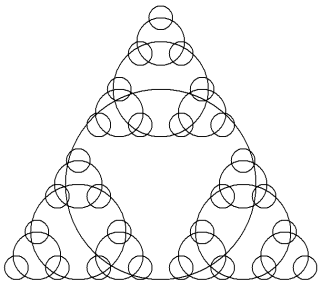
15-112 Spring 2023 Quiz 9

Up to 20 + 10 minutes (finish within 20 minutes for 20% proficiency bonus). No calculators, no notes, no books, no computers. Show your work!

1. (3 points) **Short Answer**: Consider the following fractals drawn with a maximum recursion depth of 1, 2, and 4. Assume that the function `drawCierclinskyFractal` is called to draw the fractal. In the code, how many calls to `drawCierclinskyFractal` and `canvas.create_oval` occur in the base case? What about the recursive case?


Maximum depth = 1


Maximum depth = 2


Maximum depth = 4

Base Case		Recursive Case	
Number of calls to `drawCierclinskyFractal`		Number of calls to `drawCierclinskyFractal`	
Number of calls to `canvas.create_oval`		Number of calls to `canvas.create_oval`	

2. (5 points) **Code Tracing**: Indicate what the following program prints. Place your answer (and nothing else) in the box next to the code. Ensure strings are enclosed with quotes and the uppercase and lowercase are distinguishable.

```python
def recFunc(L, m):
    print(f"{L} {m}")
    L.append(m)
    if m == 0:
        return 42
    else:
        return m + recFunc(L, m//2)

def ct(n):
    L = [ ]
    result = recFunc(L, n)
    return L + [result]
print(ct(3))
```

3. (4 points) **Reasoning Over Code**: Find an argument, `s`, for the function `roc` to cause it to return `True`. Place your answer (and nothing else) in the box below the code. Make sure strings are enclosed with quotes.

   Note: `rocHelper(s, d)` is a helper function, and your answer should refer to `roc(s)`.

```
def rocHelper(s, d):
    if d == 1:
        return (s[0] == 'l')
    if d == 2:
        return (s[:2] == 's4')
    h = d//2
    return rocHelper(s[:h], h) and rocHelper(s[h:], d-h)

def roc(s):
    return len(s) == 5 and rocHelper(s, 5)
```

4. (8 points) **Free Response**: Write the **recursive** function `recGetDigitFreqs(n)` that, given an integer `n`, returns a dictionary that maps each digit of `n` to the number of times the digit occurs in `n`. For instance,

```
assert(recGetDigitFreqs(15112) == {2: 1, 1: 3, 5: 1})
assert(recGetDigitFreqs(2023) == {3: 1, 2: 2, 0: 1})
assert(recGetDigitFreqs(-2023) == {3: 1, 2: 2, 0: 1})
assert(recGetDigitFreqs(11223344) == {4: 2, 3: 2, 2: 2, 1: 2})
assert(recGetDigitFreqs(0) == {0: 1})
assert(recGetDigitFreqs(1000) == {0: 3, 1: 1})
assert(recGetDigitFreqs(-111) == {1: 3})
```

Your solution must use recursion. If you use any loops or iterative functions, you will receive no points on this problem. You should not convert `n` to a string.