219116/117 Programming II

Extended Lab 2

1. num_ways.py

Here are three ways to get to 3 using only addition and the two numbers 1 and 2.

```
1 + 1 + 1
1 + 2
2 + 1
```

Here are the 5 ways to get to 4 using only addition and the two numbers 1 and 2.

```
1 + 1 + 1 + 1
1 + 1 + 2
1 + 2 + 1
2 + 1 + 1
2 + 2
```

The following function calculates the number of ways to get to the value n using only addition and the digit 1 or 2:

```
def num_ways_with_1_2_to_get(n):
    if n < 2:
        return 1
    return num_ways_with_1_2_to_get(n - 1) + num_ways_with_1_2_to_get(n - 2)
```

You can assume that the n value is greater than or equal to 1.

To understand the code above, you need to think recursively as follow:

"The last addition to get to n must be +1 or +2.

If it is +1, then all the previous additions sum up to n - 1. The possible combination of +1 and +2 leading up to n - 1 is num_ways_with_1_2_to_get(n-1)

If it is +2 then, all the previous additions sum up to n - 2. The possible combination of +1 and +2 leading up to n - 2 is num_ways_with_1_2_to_get(n-2)"

Your job is to modify the above code so that it also prints out the details of all the possible ways as well as returns the total number of possible ways. For example, a call

```
num_ways_with_1_2_to_get(3)
```

will output:

```
1 + 1 + 1
1 + 2
2 + 1
3
```

A call

```
num_ways_with_1_2_to_get(4)
```

will output:

```
1 + 1 + 1 + 1
1 + 1 + 2
1 + 2 + 1
2 + 1 + 1
2 + 2
5
```

Hint: passing one argument to num_ways_with_1_2_to_get may not be enough to accomplish this task. You may want to create a helper function where num_ways_with_1_2_to_get(n) makes a call to another function num_ways_with_1_2_to_get_helper(n, s) where s 'remembers' what needs to be printed out.

After you are done with the above task, write another function

```
def num_ways_with_1_2_3_to_get(n):
```

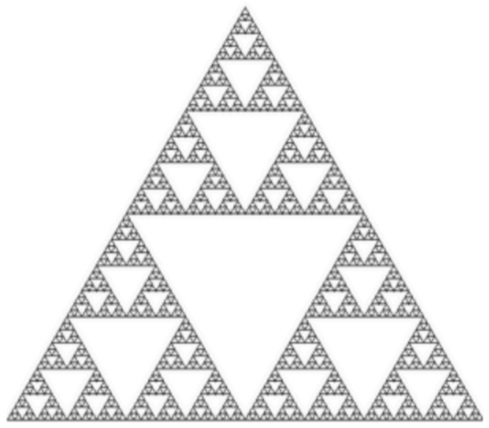
that calculates the number of ways to get the value n using only addition and the digit 1 or 2 or 3. Also, make it prints out the details of all the possible ways. For example, a call `num_ways_with_1_2_3_to_get(3)` will output:

```
1 + 1 + 1
1 + 2
2 + 1
3
4
```

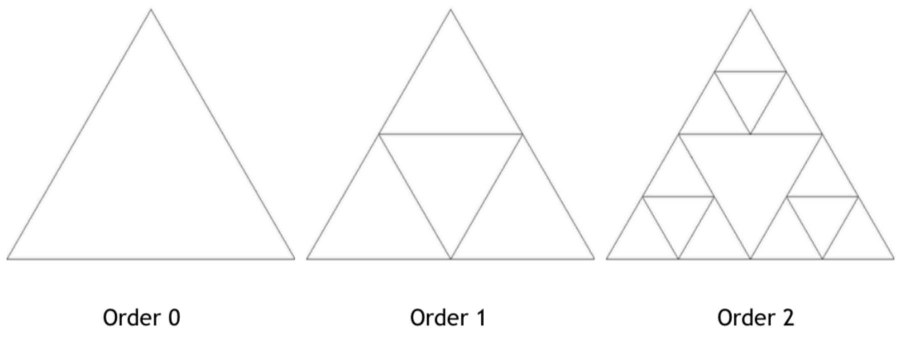
A call `num_ways_with_1_2_3_to_get(4)` will output:

```
1+ 1 + 1 + 1
1+ 1 + 2
1+ 2 + 1
1+ 3
2 + 1 + 1
2 + 2
3 + 1
7
```

2. Sierpinski.py



The above is a picture of a Sierpinski triangle, which can be constructed recursively. Look at how you create a Seirpinski triangle of order 0, 1, and 2 as follows:



Write recursive Python code in Sierpinski.py using Turtle graphics to produce a Seirpinski triangle of order n, where n is a parameter that a user can specify. Run your code with n equals to 5 and capture the screenshot showing the output. Save the screenshot in a file named Seirpinski_5.png and include it in the submission folder.

3. recursive_art.py

Write recursive Python code in recursive_art.py using Turtle graphics to produce any recursive art piece that is limited only by your imagination :) Run your code and capture the screenshot of your art in a file named recursive_art.png and include it in the submission folder. See the following link for sample art works created by students at UC Berkeley in the USA.

https://inst.eecs.berkeley.edu//~cs61a/fa15/proj/scheme_gallery/

Submission:

- ***Create StudentID_Firstname_lab2_ext folder, where StudentID is your KU ID and Firstname is your given name***
- ***Put the files to submit, num_ways.py, Sierpinski.py, Seirpinski_5.png, recursive_art.py, and recursive_art.png, into this folder***
- ***Zip the folder and submit the zip file to the course's Google Classroom before the due date***