Eric Roberts and Mark Hopkins
CSCI 121

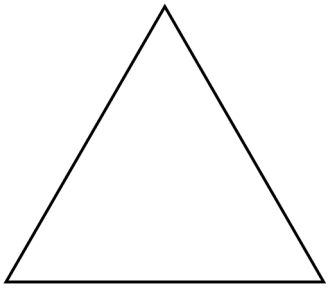
Handout #57
November 26, 2018

Lab Assignment: Recursion

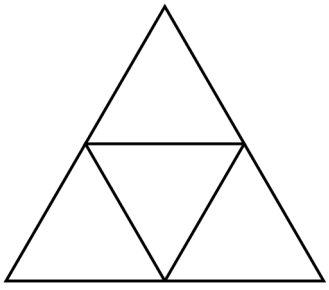
This week's lab consists of two problems intended to help you think about recursive solutions. Learning to solve problems recursively can be challenging, especially at first. Recursive solutions can often be formulated in a few concise, elegant lines but the density and complexity that can be packed into such a small amount of code may surprise you.

Problem 1. Sierpinski Triangle

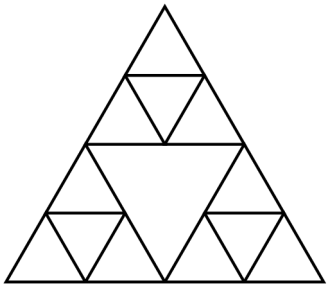
If you search the web for fractal designs, you will find many intricate wonders beyond the Koch snowflake that Eric covered today in lecture. One of these is the ***Sierpinski Triangle,*** named after its inventor, the Polish mathematician Wacław Sierpiński. The order-0 Sierpinski Triangle is an equilateral triangle:



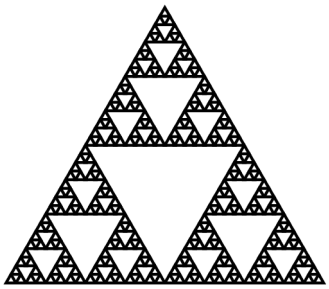
To create an order-N Sierpinski Triangle, you draw three Sierpinski Triangles of order $N-1$, each of which has half the edge length of the original. Those three triangles are placed in the corners of the larger triangle, which means that the order-1 Sierpinski Triangle looks like this:



The downward-pointing triangle in the middle of this figure is not drawn explicitly, but is instead formed by the sides of the other three triangles. That area, moreover, is not recursively subdivided and will remain unchanged at every level of the fractal decomposition. Thus, the order-2 Sierpinski Triangle has the same open area in the middle:



If you continue this process through three more recursive levels, you get the order-5 Sierpinski Triangle, which looks like this:



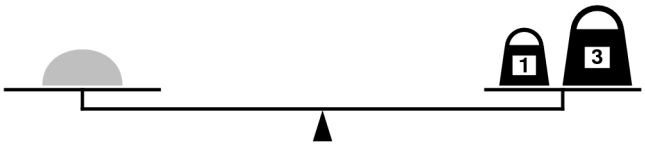
Create a class called **GSierpinskiTriangle** that represents a Sierpinski Triangle as a graphical object. Unlike the snowflake fractal, the Sierpinski Triangle is not a polygon, so it makes more sense to define **GSierpinskiTriangle** as an extension of **GCompound**. The constructor for **GSierpinskiTriangle** should take the edge length and fractal order as parameters. Thus, the following program should display the preceding diagram in the center of the graphics window:

```
def TestSierpinskiTriangle():
    gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT)
    triangle = GSierpinskiTriangle(100, 5)
    gw.add(triangle, gw.getWidth() / 2, gw.getHeight() / 2)
```

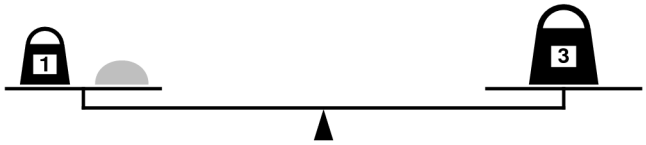
Problem 2. Measurements on a balance scale

> *I am the only child of parents who weighed, measured, and priced everything; for whom what could not be weighed, measured, and priced had no existence.*
>
> —Charles Dickens, *Little Dorrit,* 1857

In Dickens's time, merchants measured many commodities using weights and a two-pan balance—a practice that continues in many parts of the world today. If you are using a limited set of weights, however, you can measure only certain quantities. For example, suppose that you have only two weights: a 1-ounce weight and a 3-ounce weight. With these weights you can easily measure out 4 ounces, as shown:



It is somewhat more interesting to discover that you can also measure out 2 ounces by shifting the 1-ounce weight to the other side, as follows:



Write a recursive function

```
def isMeasurable(target, weights):
```

that determines whether it is possible to measure out the desired target amount with a given set of weights, which is a set of integers. For example, if **`sampleWeights`** has been initialized as

```
sampleWeights = { 1, 3 }
```

the function call

```
isMeasurable(2, sampleWeights)
```

should return **`True`** because it is possible to measure out 2 ounces using the weights as illustrated in the preceding diagram. On the other hand, calling

```
isMeasurable(5, sampleWeights)
```

should return **`False`** because it is impossible to use the 1- and 3-ounce weights to measure out 5 ounces.

The key to solving this problem is formulating the right recursive decomposition. As in the **`subsets`** problem from class, you need to think about what you can do with a single weight from the set and how taking a single weight away from the set generates a smaller problem that you can use in the solution to the more general case.