CMSC 201
Section 40
Spring 2020
Lab #8 – Recursion

Problem

We have discovered a new planet with an interesting gravitational system. All things in the air eventually fall to the ground due to gravity, but not always in a uniform way. Constant acceleration due to gravity is not a thing. Instead, gravity follows the following rules:

- All objects are always either on the ground, under the ground, or an integer number of feet above the ground at all measurable time units
- Anything one foot above the ground or less immediately falls to the ground in negligible time.
- Anything an even umber of feet above the ground falls half the distance to the ground per time unit
- Anything an odd number of feet above the ground rises during the next time unit. Its new height is three times its previous height, plus one foot.

Your job is to write a program that determines how many units of time – how many steps – it will take an object starting at any height to hit the ground.

You must do this using a recursive function. The skeleton for your program, including both the main program and the function, will be provided.

Recursion

Background

Both recursion and iteration break a large problem down into smaller pieces. The main difference between recursion and iteration can be found if we look at their underlying purpose.

With iteration, the purpose is to repeat an action until a task is done. This is true for **`while`** loops (stop when the conditional evaluates to **`False`**) and **`for`** loops (stop when it reaches the end of the generated range of numbers).

With recursion the purpose is to break a problem down into smaller and smaller pieces of ***itself***. When you combine all of those solved smaller pieces of the problem, the problem as a whole is solved.

The Parts of a Recursive Function

A successful recursive function must have two parts: at least one ***base case*** and at least one ***recursive case***. The base case is similar to the conditional in a **`while`** loop, in that it tells the program when to stop. In a recursive function, it stops calling itself, and typically returns something (a value, a message, etc.). A recursive function may have more than one base case, just like a **`while`** loop may have more than one comparison in its conditional.

The recursive case is the more interesting part, since this is where the function makes the ***recursive calls***. A recursive call is the most important part of a recursive function, and has a few key features:

- It must call the function again with new inputs.
- These new inputs must cause the function to approach at least one of the base cases.
- If needed, the call must also include the **`return`** keyword, in order to be able to return the final result from the original function call.

Example of a Recursive Program

You've seen a number of recursive examples in class already, but let's look at a few more. A very simple one is a "countdown" function – as a reminder, this is a ***toy example***. We could easily do this with a loop, but we want to instead examine how recursion works.

Here is the code for a recursive **`countdown`** function:

```
def count_down(curr_num):

    # BASE CASE
    if curr_num == 0:
        print("The end!")
    # RECURSIVE CASE
    else:
        print("Counting down from", currNum, "...")
        count_down(curr_num - 1)    # <----RECURSIVE CALL
```

Here is a sample run, using the full code (including a simple **`main()`** to get the number and make the initial call to the recursive function):

```
Please enter a number to count down from: 7
Counting down from 7 ...
Counting down from 6 ...
Counting down from 5 ...
Counting down from 4 ...
Counting down from 3 ...
Counting down from 2 ...
Counting down from 1 ...
The end!
```

The base case, when the function ends, is when the number reaches zero. When it stops running, the **`count_down()`** function doesn't need to return anything, it simply doesn't call itself (the recursive function) again.

Tasks for this lab:

1. Copy the file **hailstone.py** from Professor Arsenault's public directory at:

cp /afs/umbc.edu/users/a/r/arsenaul/home/pub/hailstone.py hailstone.py

2. Look at the code for the function **flight()** and for the main program. Make sure that you understand what is going on. The main program asks the user to input the starting height of a hailstone. It converts the entry into an integer, and then calls the flight() function with the starting height as its argument.
3. The flight() function calculates how long it will take for the hailstone to hit the ground when starting at the given height, in our strange system of gravity. The rules of flight in our system are:
    a. If the height of the hailstone is 1 foot or less, it will hit the ground immediately. That is, it will take 0 steps to hit the ground.
    b. If the height of the hailstone is an even number of feet, it will fall half the distance to the ground in 1 step.
    c. If the height of the hailstone is an odd number of feet, it will actually RISE in the next step. Its new height will be 3 times the current height, plus one foot.
4. Write the code for the flight() function as indicated. You MUST write flight() as a recursive function. You could solve this problem with a loop, but you may not do so for this lab!!!
    a. Write the base case code. What happens if the height of the hailstone is 1 foot or less?
    b. There are *two* recursive cases that must be written. One recursive case occurs when the height is an even number of feet. The other occurs when the height is an odd number of feet. Both of these cases must include a recursive call to the flight() function.
    c. Count the number of steps it takes for the hailstone to hit the ground.
5. Test your code with each of the following initial inputs. Make sure you get the right result:

Input	Output
-4	0 steps
1	0 steps

16	4 steps
9	19 steps
30	18 steps
13	9 steps
64	6 steps