CMSC 201
Section 40
Spring 2020
Homework # 6: Recursion

Value: 40 points

Collaboration:

For Homework 6, collaboration is allowed. Make sure to consult the syllabus about the details of what is and is not allowed when collaborating. You may not work with any students who are not in Section 60.

If you collaborate with another student, you must both email your instructor and make a note in your code. When emailing your instructor, include the other student's name and UMBC email address, the date that the collaboration took place, and a brief description of what the collaboration was about. Place this same information in the header comment of your code (see below). Even if someone helped you, but you didn't get a chance to help them (or vice versa), you both still need to follow this procedure. You must email your instructor within 24 hours of the collaboration occurring (we recommend emailing immediately, so you don't forget any details).

Make sure that you have a **complete file header comment at the top of each file**, and that all the information is correctly filled in.

```
"""

File:    FILENAME.py
Author:  YOUR NAME
Date:    THE DATE
Section: YOUR DISCUSSION SECTION NUMBER
E-mail:  YOUR_EMAIL@umbc.edu
Description:
   DESCRIPTION OF WHAT THE PROGRAM DOES
Collaboration:
   DESCRIPTION OF HOW YOU COLLABORATED
   (Include this section only if you collaborated.)
"""
```

Each part of this homework assignment is worth the indicated number of points. Following the coding standards is worth 4 points. Failing to have complete file headers or failing to have correctly named files will lead to a deduction in points.

hw6 directory

Just as you did for previous homeworks, you should create a directory to store your Homework 6 files. We recommend calling it **`hw6`** and creating it inside the **`Homeworks`** directory inside your **`201`** directory. You don't need to make a separate folder for each file. You should store all Homework 6 files in the same **`hw6`** folder.

Part 1: Right Triangles (8 points)

Create a program that draws an isosceles right triangle on the screen, of a height to be entered by the user, and using a symbol to be entered by the user. Remember from your geometry class that an isosceles right triangle is a triangle with a right angle and two 45 degree angles. Two legs of the triangle – the base and the height – are equal in length. Only the hypotenuse is longer. So an isosceles right triangle is uniquely determined by the height.

You can draw a triangle on the screen by printing a particular character on the screen, in the shape of a triangle.

The core of your program will be a recursive function,

def draw_triangle(height, symbol):

that takes as parameters a positive integer representing the height of the triangle, and the symbol to be printed to represent the triangle. This function need not return a value, so you need not include a return statement in your function.

Then write a main program that exercises that function by prompting the user for the height of the triangle, and then for the symbol to use to draw the triangle. Use good prompts - your prompts should contain enough information for the user to understand what values should be typed in. But you need not validate the input. The user will input a positive integer; and a single character.

Part 2: Find the factors of a positive integer (8 points)

Create a program that determines all the factors of a positive integer. The program must contain a recursive function, the name of which is up to you.

The main program must prompt the user to enter a positive integer, which will be factored. You need not validate input; the number entered by the user is guaranteed to be an integer greater than zero.

The factors must be printed out from smallest to largest, with one factor per line.

Part 3: Reverse the characters in a String (8 points)

Suppose that we want to reverse the characters in an arbitrary string. We'll prompt the user to type in a string. If the user types

“antidisestablishmentarianism”

We’ll print out that “antidisestablishmentarianism” spelled backward is “msinairatnemhsilbatsesiditna”

Write a recursive function that takes one parameter, a non-empty string, and returns that string backwards.
Then write a main program that exercises that function by prompting the user to enter a string, any string; then calling the reversing function; and printing out a message that says the initial string backwards is the backwards string.

Now, there’s one catch to this that you need to be aware of. In Python, you CANNOT directly change the characters of a string because string variables are immutable! They cannot be changed. In other words, if

s = ‘tiger’

you cannot assign

s[0] = ‘r’

to start the reversing process. That’s not legal in Python, because immutable.

So you need to think about splitting your string, so it becomes a variable of a mutable type. Then you can change the individual elements of that variable. Then when you’re done with the reversing, you can join the elements of that variable back into a single string.

Part 4: “Zip” together the elements of two lists (8 points)

Suppose that we have two lists, a and b, such that

a = [1,2,3,4,5]

and

b = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’]

“Zipping” the lists together means taking one element from each list, alternately, and putting them in order in a single list. So

myzip(a, b) = [1, ‘a’, 2,’b’, 3, ‘c’, 4, ‘d’, 5, ‘e’]

If the lists a and b are the NOT the same length, you zip the two of them together as shown above until the shorter list is empty. Then you copy the rest of the elements of the longer list into the end of the zipped list. E.g., if

c = [1, 2, 3, 4, 5]

and

d = [‘a’, ‘b’, ‘c’}

then myzip(c,d) =]1, ‘a’, 2, ‘b’, 3, ‘c’, 4, 5]

Your task is to write a recursive function, myzip, that takes as parameters two lists, and returns a single, zipped list as above. Note that you *must* implement this as a recursive function. You certainly could do it iteratively, but that’s not the point of this homework.

Think about the base cases. You should probably have more than one!!

Write a main program that exercises your function. You should prompt the user to enter the elements for the first list; entering “q” to indicate the end of the first list. You should then prompt the user to enter the elements of the second list, entering “q” to indicate the end of the second list.

You can safely presume that each parameter is a one-dimensional list. You can treat each list element as a string.

Part 5: Multiplication without multiplying (8 points)

If you did not have a multiplication operator built in to the Python interpreter, you would have to write your own in order to allow users to multiply numbers together. So, you’re going to do that recursively. You’re going to write a recursive function that multiplies two integers together. Note that the numbers will be integers, but they will NOT necessarily be positive integers. You MUST be able to deal with -3 * -4!

If you had to do this iteratively, you would multiply a times b by writing a loop that adds a together b times (presuming b was positive). Doing it recursively is not much harder.

Think about the base case(s). There might be either one or two, depending on how you write your code. Either is acceptable for this exercise.

Write the recursive function, def times (a, b): that takes two integers as parameters and returns their product. Then write a main program that prompts the user to enter the two integers that are to be multiplied. You may assume that the user enters valid integers. If the user enters one or two negative integers, you should deal with that in the main program.