CSC 148 Homework 5 Recursion and Recursive Graphics

Due date: December 14th, 2021

**Total: 20+6Pts = 26Pts**

1. Recursion (5pts)

We can determine how many digits a positive integer has by repeatedly dividing by 10 (without keeping the remainder) until the number is less than 10, consisting of only 1 digit. We add 1 to this value for each time we divide by 10.

Here is the recursive algorithm:

1. If *number < 10* return 1.

2. Otherwise, return *1 + the number of digits in (number//10)* (ignoring the fractional part).

Implement a recursive algorithm in Python to solve this problem and test it using a

*main* function that calls this with the values 15, 105, and 15105.

(HINT: Remember that if the variable ‘number’ is an integer, (number//10) will be an integer without the fractional part.)

2. (5pts) Recall the function is\_palindrome() that we learned in the recursion lecture, to check if a string is a palindrome (reverse is the same as the itself, e.g. “doggod” is a plaindrome), scan a file line by line find the palindrome in it and return a list with palindrome found. You can use Alice.txt uploaded on canvas.

And please put Alice.txt in the same folder as your code.

Make sure your code also works for other files.

You can define a function is\_pal(word), which takes a string and return True or False.

And then call this function in your main function:

def is\_Pal(theWord):

“””” given a string, return True if it is a palidrom”””

def find\_palidrome(filename):

“”” given a file name, return a list of words that are palidrom”””

Please exclude words that has length of 1 (e.g. “a”).

3 (10pts) A Sierpinski triangle of order 0 is an equilateral triangle. An order 1 triangle can be drawn by drawing 3 smaller triangles (shown slightly disconnected here, just to help our understanding). Higher order 2 and 3 triangles are also shown. Draw Sierpinski triangles of any order input by the user using a recursion function.

Shape

Description automatically generated

To do this, you need to use the *turtle* module for drawing.

Here is the starter code to draw triangles with *turtle*.

Here are some tutorials:

<https://openbookproject.net/thinkcs/python/english3e/recursion.html>

**import** turtle

board **=** turtle.Turtle()

board.forward(100) # draw base

board.left(120)

board.forward(100)

board.left(120)

board.forward(100)

turtle.done()

1. Make the above code into a function (see example in textbook).
2. Try to set up a recursive function call to make a recursive graph. But make sure the triangles are drawn with shorter edges every time the function is called.

The textbook has a good tutorials on turtle()

<https://openbookproject.net/thinkcs/python/english3e/recursion.html>

You can watch this demo video to learn how to draw recursive graphics using turtle graphics:

<https://www.youtube.com/watch?v=31pKs9UuUvA>

Note: If, for some reason, turtle graphics doesn’t work, you can use PyGame

import pygame, math

pygame.init()

1. (Bonus 3pts) Adapt the above program to change the color of its three sub-triangles at some depth of recursion. The illustration below shows two cases: on the left, the color is changed at depth 0 (the outmost level of recursion), on the right, at depth 2. If the user supplies a negative depth, the color never changes. (Hint: add a new optional parameter colorChangeDepth (which defaults to -1), and make this one smaller on each recursive subcall. Then, in the section of code before you recurse, test whether the parameter is zero, and change color.)

Diagram, shape

Description automatically generated

You might want to use PyGame. You can read how to do it here:

<https://openbookproject.net/thinkcs/python/english3e/recursion.html>

1. **(Bonus 3pts**) Create another recursion graphics, preferably with color!! Be creative!

**Here are some examples of recursive graphics:**

<https://www.cs.princeton.edu/courses/archive/spr15/cos126/art/index.php>

But please do not copy code from the Internet. You can always start with something simple.

A picture containing vector graphics

Description automatically generated