Computer Science I — CSCI-141
Arrows — Lab 3

09/09/2020

1 Problem

Using Python's turtle graphics module, design a program that draws a sequence of two figure trails in a randomly colorful fashion; the number of figures drawn is determined by a number entered by the user Each figure element in the lab assignment is an *equilateral triangle* When drawing, the figure elements must stay within the boundaries of a square outlining a bounding box.

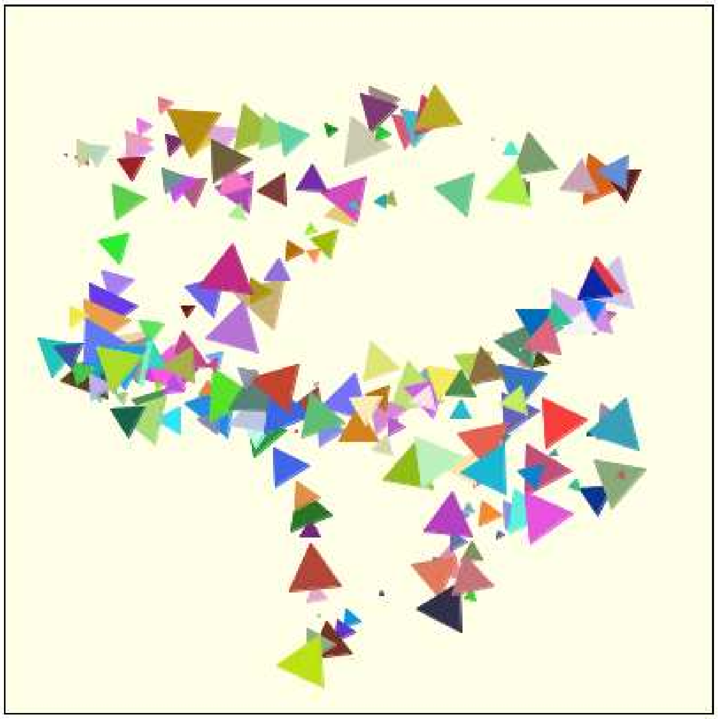

Figure 1: An example of a drawing with 250 arrows.

This image is the kind that the implementation will have to draw both recursively and iteratively. Problem-solving works on a simpler problem whose solution will lead to a solution to this larger problem.

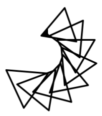

Figure 2: Output of a problem-solving function.

2 Problem-solving Session (15%)

Work in a team of students as determined by the instructor. Each team will work together to complete the following activities.

During problem-solving, do not consider color or any randomness. Do not worry about the bounding box until the last question.

1. Write a recursive function `draw_figures_rec` that takes at least two arguments: a positive integer that represents the number of figure elements to draw, and another positive integer that represents the side of the largest triangle to draw. (Use of additional arguments is acceptable, as are auxiliary, helper/utility functions.)
   Each successive figure should have a side length one unit smaller than the previously drawn figure. Specifically, the recursive function should draw a figure element with the specified side length, then lift the pen, move forward 10 units, turn left 30 degrees, put the pen back down, and make a recursive call to continue drawing the remaining figures after decrementing the side length by one.

   The recursive function should return the sum of the side lengths of each triangle drawn. For example, `draw_figures_rec(3,5)` should draw a total of three triangles with side lengths 5, 4, and 3, ultimately returning 36 as the total of the individual side lengths ($3 * 5 + 3 * 4 + 3 * 3 = 36$).

   The code must use recursion. However, the main function `draw_figures_rec` may call a helper function to do the actual recursion, or the main function may employ additional parameters if they are given default values.

2. Provide a substitution trace for `draw_figures_rec(4, 10)`.

3. Write code for an iterative function, `draw_figures_iter`, that takes two arguments: a positive integer that represents the number of figure elements to draw, and another positive integer that represents the side length of the largest triangle to draw. The function must use a `while` loop instead of recursion, and it should produce the same results as the recursive version (both drawing and return value).

4. Create a timeline of changes to the parameters and local variables used in `draw_figures_iter(3, 5)`.

5. The finished program must keep the figures within the bounding box as shown on the drawing canvas. Given a bounding box drawn on the canvas, develop and describe a strategy (not code) that will keep the figures being drawn within that box. That is, when the turtle is about to draw, it will detect when all or part of the next figure would go outside of the boundary and stop to avoid drawing outside. You do not have to be precise. It's OK to be conservative and use a simple formula.

Have an SLI or instructor check things over as progress is made.

3 post-Problem-Solving-Session Work (10%)

Following the problem-solving, write the following in a program file named **arrows.py**:

- Fill in the two functions you wrote in problem-solving, `draw_figures_rec` and `draw_figures_iter`.
- Add a main function that invokes those functions as in your traces:
  - `draw_figures_rec(4, 10)`
  - `draw_figures_iter(3, 5)`
- Now write function `eq_tri_area(side)` to compute the area of an equilateral triangle with a given side length. You may search online for the formula. In the main lab assignment you will use this calculation instead of adding lengths of sides.

Work on this for about an hour to get the implementation going. Then *upload the work-in-progress* **.py** *file to the MyCourses post-PSS dropbox by the due date.*

4 Implementation (75%)

The full implementation elaborates the work-in-progress to complete a solution.

Each student will *individually implement* and submit their own solution to the problem in their Python file named `arrows.py`.

This program file should contain the following:

- Python implementations of the functions outlined in the problem-solving session, removing the side length parameter since it will now be randomly generated (Details on other changes to these functions are explained in Section **??**);
- Changes so that the function's return value is a `float`ing-point number that equals *the sum of the areas*.
- A `main` function that does the following (see Requirements Section for more detail):

```
Prompt the user for the number of triangles
If number of triangles is out of range, produce an error and exit
Otherwise:
    Set up the drawing window for the recursive function
    Call the recursive drawing function and print the sum of areas
    Pause the drawing and wait for the user to hit the enter key
    Reset the window for the iterative function
    Call the iterative drawing function and print the sum of areas
    Pause and wait for the user close the window.
```

- Any other, additional support/utility functions that promote function re-use (e.g. setting up the drawing window).

4.1 Requirements, Constraints and Tips

- Use the default turtle window, and do not label the drawings as recursive or iterative.
- The program should run correctly using Python version 3. Each time that the program executes, it should produce one picture using recursion and one using iteration. Below is a transcript of output running the program to produce two different pictures like the example in this document.

```
Arrows (0-500): 250
The total area painted is 34343.53642517762 units.
Hit enter to continue...
The total area painted is 33717.40005824142 units.
Close the canvas window to quit.
```

- The program must prompt the user for the number of triangles/arrows:

```
Arrows (0-500):
```

- If the number of triangles is out of range, print the following message and exit:

```
Arrows must be between 0 and 500 inclusive.
```

- After each drawing is completed, print a message to standard output that displays the total area of all painted triangles; for example:

```
The total area painted is #### units.
```

- Pause the program between displaying the recursive and iterative pictures.

- Use the same number of figure elements for both the recursive and iterative pictures.
- Display the bounding box, and make sure that the figure elements must stay within that box for the entire drawing.

4.2 Constants

Use the constant values below as **bounds on the range of a legal value**. The code needs to check for legal input that is in range. For values generated by the random number generator, the code needs to ensure that the value is also in the correct range.

Do not hardcode these as magic numbers in the code! Instead create a separate global variable for each constant. When the constant is needed in the code, use the constant variable name.

(After defining a global variable's value, do not change that value; global values are supposed to be constant.)

Names for the constant variables must be *fully capitalized* as shown.

- The code prompts the user for the number of triangles. The valid range for the number of triangles is between `0` and `500` inclusive. Therefore `MAX_FIGURES` should be `500`.

  The corresponding definition for this constant at the top of the source file would be:

  ```
  MAX_FIGURES = 500
  ```

- The bounding box should be between the points `(-200,-200)` and `(200,200)`. This is the drawing area; the triangles should stay within that. That means `BOUNDING_BOX` should be `200`, the absolute value of the box's x or y value.
- `MAX_DISTANCE` should be `30`. The maximum distance between the starting points of two figure elements should be 30. Individual distances should be integers between `1` and `MAX_DISTANCE` inclusive.
- `MAX_SIZE` of a single figure element should be 30. Individual sizes should be integers between `1` and `MAX_SIZE` inclusive.
- `MAX_ANGLE` should be `30`. The maximum angle corresponds to the absolute value of the angle the turtle turns before moving to the starting position for the next figure element. The range of the angle turn should be an integer between `-MAX_ANGLE` and `MAX_ANGLE` inclusive.

4.3 Random

There are two functions in the `random` module that will be useful:

- `randint(a,b)` returns a random integer in the range `[a,b]`, including the endpoints This function is useful for generating the random distance between figure elements, the triangle side length and the angle to turn.

```
>>> random.randint(1,20)
7
```

- `random()` returns a random floating point value in the range `[0,1]`, including the end points This is useful for randomizing the color of each segment (see the Turtle section ?? below for more detail).

```
>>> random.random()
0.2128164618278381
```

4.4 Turtle

A variety of capabilities from the `turtle` library will help produce the drawing.

- `color(r,g,b)` Colors are represented as intensities of red, green, and blue: *RGB*. The range of floating-point values is 0.0 to 1.0. These numbers represent emitted light, so 0.0 is black and 1.0 is maximum light. For example, the call `turtle.color(0,0,1)` would set the color to blue, and `turtle.color(0,0.5,0)` would set the color to a dim green.
- `done()` Wait for the user to close the turtle window. This ends the program.
- `reset()` Reset the window by clearing everything and setting the turtle to its default state. Use this in between the two drawings.
- `setheading(angle)`: This function call sets the heading of the turtle so that it faces a certain way. The default unit of measurement is degrees, and they follow the usual mathematical convention: starting pointing east, increasing counter-clockwise. For example, `turtle.setheading(90)` would make the turtle face north.
- `xcor()` or `ycor()`: can be used to return the current x or y coordinate of the turtle on the screen. This can be useful when trying to determine whether or not the turtle is too close to the bounding box.

4.5 Algorithm

To draw the next 'arrow' in a sequence, the program needs to generate several random values. Below are descriptions of what the algorithm should do when it is deciding how and where to draw the next figure.

- Randomly generate the distance to move in the range $[1...MAX\_DISTANCE]$.
- Randomly generate the side length in the range $[1...MAX\_SIZE]$.
- Randomly generate a new color to fill the next figure. Each of the three color channel values should in the range [0.0...1.0] so that it will appear as a shadow at a minimum.
- Randomly generate the relative turning angle the $[\pm MAX\_ANGLE]$ range.

While doing these calculations, the algorithm needs to check whether the next figure will likely break out of the bounding box. The algorithm must adjust things if that is the case.

Note: the above random operations may not necessarily be in the order given.

4.6 Grading

The implementation grade is based on these factors:

- 20%: The program draws the figure elements recursively (15%) and computes its painted area (5%).
- 20%: The program draws the figure elements iteratively (15%) and computes its painted area (5%).
- 5%: The program prompts for and uses the number of figure elements.
- 5%: The program displays an error message and exits if the number of figure elements is out of range.
- 5%: The figure elements stay within the bounding box when drawn.
- 5%: The program pauses after drawing the first picture.
- 5%: The program resets and clears the turtle window before drawing the second picture.
- 5%: The program uses constants with no *magic numbers* when drawing.
- 5%: The code follows the style guidelines on the course web site, including docstrings for each function.

4.7 Submission

ZIP the **arrows.py** implementation into a file named **lab03.zip** and submit to the **My-Courses dropbox** by the due date for this lab.