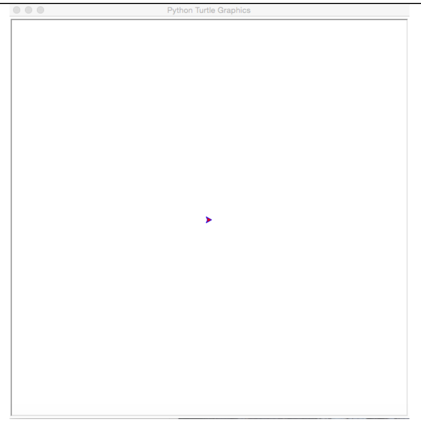
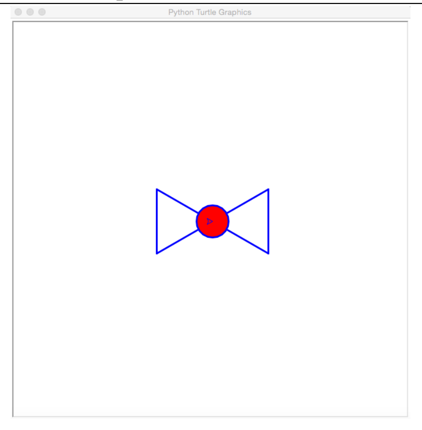
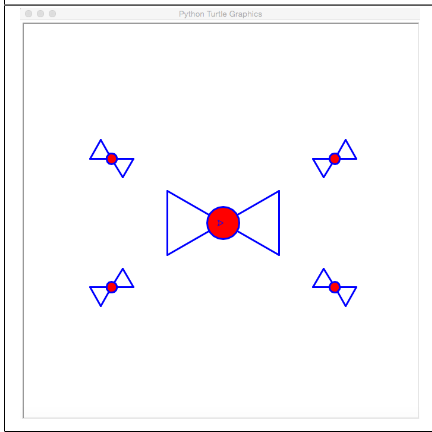
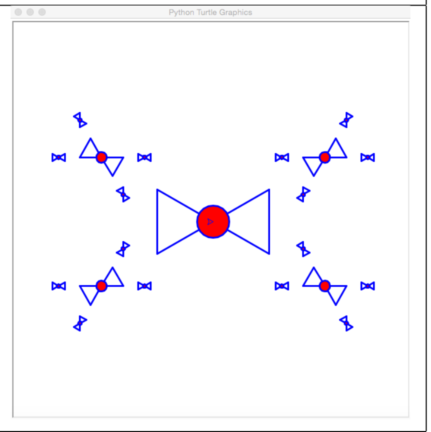
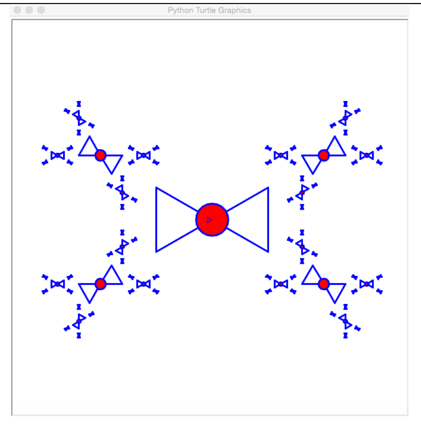
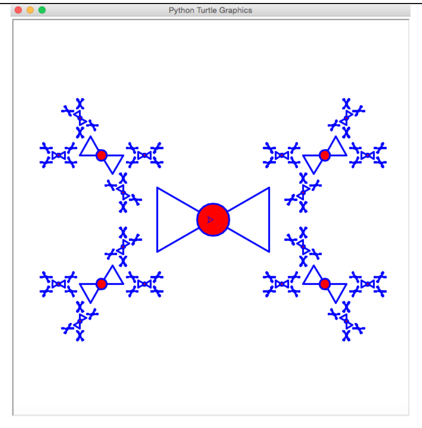
Computer Science I — CSCI-141
Bow Ties — Lab 3

09/09/2023

1 Problem

Write functions and compose a program that draws figures like the pictures shown below. (If it is difficult to see in the images, the outline of the "bowties" uses one color, and the interior of the circle is a second color.) The program will prompt the user for the recursive `depth`, and will use recursion to draw the appropriate picture.

Table 1: Example Figures for `depth` Values 0-5

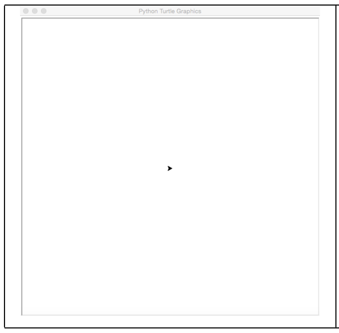
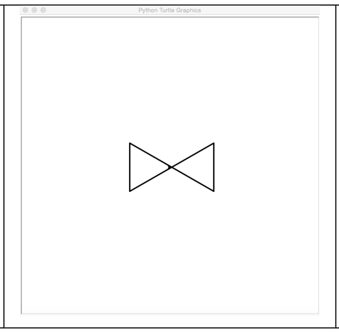
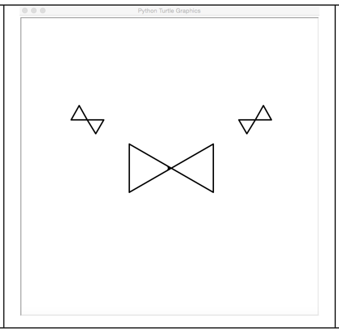
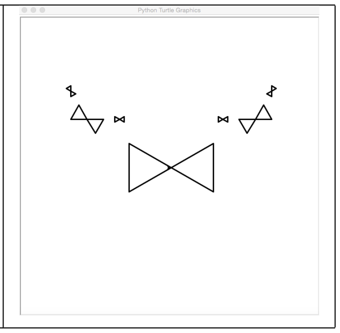
	
	
	

2 Problem-solving Session Deliverables (15%)

Problem-solving simplifies the problem a little by removing the issue of color, eliminating the circles from the drawings, and reducing the number of bowties (the number of recursive calls). For the bowtie images below, the bowties decrease in size by a factor of 3 at each successive level. The center point for the next smaller size bowtie is reached by extending the line (with the turtle 'up') of the current bowtie. If the current bowtie has side length equal to `size`, the center of the next smaller bowtie is reached by extending the line of the current bowtie another `size` units.

(The final lab solution will have to produce the *original* drawings.)

Table 2: Example Problem-solving Figures for `depth` Values 0-3

Recursion happens when a function calls itself. Refer to the lecture code that shows `draw_tree3` calling `draw_tree2` with half the size. If you write a `draw_tree` function that calls itself with half the size, you have beginnings of a recursive function. The problem is: 'how do you stop it?'

Problem-solving involves teamwork with several other students.

1. Write Python code for a function, `draw_one_bowtie`, that draws a bowtie. The function should take one parameter, `size`, representing the length of a side of a triangle in the bowtie. Indicate the initial state of the turtle when the function is called, and make the function return the turtle to the same state when it completes. An additional helper function would be acceptable.

2. Write Python code for a non-recursive function, `draw_bowtie0`, that produces the drawing for `depth == 0`.

3. Write Python code for a non-recursive function, `draw_bowtie1`, that takes one parameter, `size`, and produces the drawing for `depth == 1`. Use the previous function(s).

4. Write Python code for a non-recursive function, `draw_bowtie2`, that takes one parameter, `size`, and produces the drawing for `depth == 2`. Use the previous function(s).

5. Write Python code for a non-recursive function, `draw_bowtie3`, that takes one parameter, `size`, and produces the drawing for `depth == 3`. Use the previous function(s).

3 In-lab Session (10%)

After the end of the problem-solving session, begin work independently to complete the following tasks for upload to the In-lab assignment dropbox for lab 03:

- Implement and demonstrate an updated `draw_one_bowtie` function. Modify it so that it draws a single bowtie according to the final lab specifications in which the bowtie also includes a circle. Additionally, one color should be used for the outline, and a second color for the interior of the circle. Choose any two colors that are distinctly different. The circle should have a radius equal to 1/4 the length of the side of the triangle. The center of the circle should be the point where the triangles meet.

  The turtle commands below will be useful; to look up the turtle documentation online, a good search string would be 'python 3 turtle documentation' because it specifies the major version of the language.
  - `turtle.color()`, `turtle.pencolor()`, `turtle.fillcolor()`
  - `turtle.begin_fill()`, `turtle.end_fill()`.

Do not be afraid to ask for help from the SLI, TA, instructor, or mentoring center if you are having trouble!

Zip your `bowties.py` file into `lab03.zip` and upload this work-in-progress zip to the In-lab 03 box before the deadline in MyCourses.

The implementation writeup will be announced after problem-solving.

4 Implementation (75%)

The final, *individual implementation* must complete the following tasks and include these items:

1. Your solution must be a recursive implementation, such that the same function `draw_bowties` is reused to draw the bowties at every depth.
2. The program file must be named `bowties.py` .
3. The program must prompt for recursive depth, and then draw the correct picture. Make sure the program draws the original pictures, not those from problem-solving!

4.1 Additional Problem Constraints

1. The turtle may *not teleport (e.g., use* `setpos()`*)* from one location to another. The turtle may lift its pen and move to a new location by a relative amount.
2. The program must use recursion to draw the figure. (Non-recursive iteration constructs are prohibited; the course has not introduced these yet.)
3. For each function, include in its *docstring* comment a description of the **pre-conditions** and **post-conditions** that apply to that function. For example, if a function expects to leave the turtle pen-up and facing North, there should be a line of text in the function's docstring that looks something like the one in this definition fragment:
```
def some_function():
    """
    ...
    post-conditions: turtle is pen-up, facing North, ...
    """
```
4. The program may not use global variables. This prohibition is in place because many students think that a global will help them; in fact, a global variable might cause problems and adversely affect the recursive execution.
5. Use two separate colors: one for the outline, and a second color for the interior of the circle. Choose any two colors that are distinctly different.
6. Use `turtle.setup()` to create a window that is square.
7. The largest bowtie should have a `size` value (the length of one side of a triangle) that is roughly 1/6 the total width (or height) of the window. That will make the whole drawing fit reasonably well in the overall window size.
8. The location within the window in which the figure is drawn does not need to exactly match the given examples, but the whole figure must fit inside the window.
9. The program must prompt the user for the `depth` value. Assume the input value for `depth` is ≥ 0. Error checking of the input is unnecessary.
10. Follow the specifications outlined in the problem-solving and lab sessions describing the relative size and placement of the bowties. In particular, the bowties decrease in size by a factor of 3 at each successive level. The center point for the next smaller size bowtie is reached by extending the line (with the turtle 'up') of the current bowtie. If the current bowtie has side length equal to `size`, the center of the next smaller bowtie is reached by extending the line of the current bowtie another `size` units. The circle

should have a radius equal to 1/4 the length of the side of the triangle. The center of the circle should be the point where the triangles meet.

4.2 Notes

For recursive drawings such as this, one way to think of the recursive function is accomplishing two things:

1. drawing some portion of a picture
2. making recursive calls that draw the remaining parts of the picture.

Different people have different valid conceptions of the overall drawing sequence within the recursive function. Here are two such approaches (there may be others):

- A two-stage organization. In stage 1, draw everything needed from this specific instance of the function call. In stage 2, move the turtle to the necessary locations and make all necessary recursive calls. This is the approach suggested in problem-solving. Within a recursive call, completely draw the current bowtie first, and then secondarily move to all necessary locations to make recursive calls that ultimately draw the remaining bowties.
- A 'call as you go' approach. For some drawings, it may be more efficient or intuitive to intersperse the recursive calls throughout the function. Draw a little bit, make a recursive call, draw some more, make another recursive call. By the time it is done, all the recursive calls have completed, as well as the drawing necessary from this specific function call.

4.3 Grading

The final implementation will be graded according to the following:

- 30% for recursive function design, including base case (no infinite loops);
- 30% for generating correct image;
- 10% for style and documentation, including 5% for *pre-conditions* and *post-conditions* for each function;
- 5% for correct `input()` usage and conversion to integer.

4.4 Submission

Zip the `bowties.py` file into `lab03.zip` and submit it to the MyCourses dropbox for this lab.