CSCI 141 - Fall 2019
Lab 4: Drawing with Loops and Turtles

Due Date: Friday, October 25th at 9:59pm

Introduction and Setup

This lab gives you practice with Python loops and Turtle graphics. You'll complete three programs that draw pictures: the first two are so-called "ASCII art", which refers to pictures made from text characters; the third one is made using a turtle. If you have questions, be sure to ask the TA: your TA is there to help you! Log into your operating system of choice and make a lab4 directory in your N drive or home directory. You will write three programs: `triangle.py`, `flag.py`, and `turtledraw.py`.

1 `triangle.py`

Your first task is to write a program that asks the user for a width, then prints a sideways isosceles triangle made of asterisks ("`*`") with the given maximum width. For example, a width of 3 would print:

```
*
**
***
**
*
```

A width of 8 would print:

```
*
**
***
****
*****
******
*******
********
*******
******
*****
****
***
**
*
```

A triangle of width 1 would print a single asterisk, and width 0 should print no asterisks.

Your solution should use at least one `for` loop. Here's a suggested way to approach this problem:

1. Create an empty file `triangle.py` in your lab4 directory. Write a comment at the top listing the code's author, date and a short description of the program.
2. Write a code snippet that prints a given number of asterisks in a row, followed by a newline. *Hint:* there are ways to do this with or without using a loop.
3. Draw the top half of the triangle by putting the code you wrote in Step 2 in the body of a loop that changes the number of asterisks drawn.
4. Draw the bottom half of the triangle using a similar approach.
5. Test your code on widths 2 and 3 first, then check 0 and 1, then try a larger number such as 8.

2 `flag.py`

The purpose of this program is to print an ASCII art approximation of the American flag.

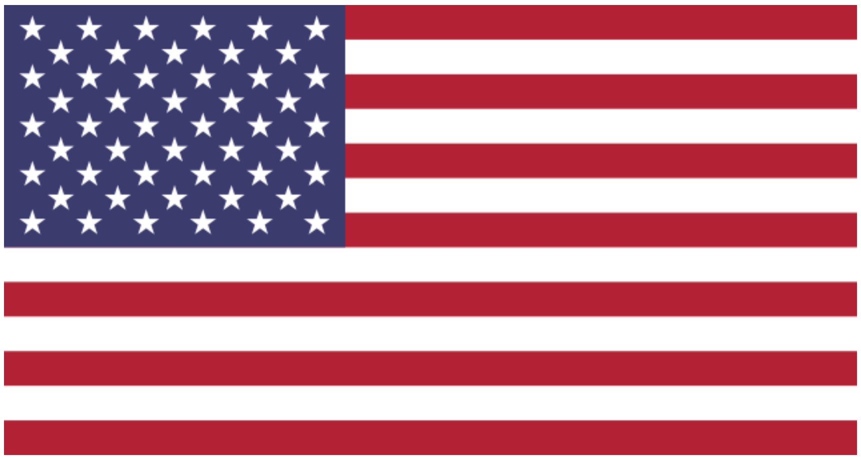

Figure 1: The true American flag

Some things to notice:

- We have to get a bit approximate with ASCII art, because the rows of stars in the true flag don't line up with the stripes. Our version lines them up, and thus has only 4 stripes below the stars, instead of 6 as in the true flag.
- The whole flag has 13 rows.
- The whole flag is 56 characters wide.
- The 9 rows of stars alternate between having 6 and 5 per row, with two spaces in between. The 6-star rows have no spaces at the beginning and two before the stripes begin; the 5-star rows have two spaces at the beginning and thee spaces before the stripes begin.

Here are some guidelines and suggestions:

- You must control structures (loops, if statements, etc.) to print the rows of the flag.
- You may use no more than 6 `print` function calls

```
*  *  *  *  *  *  ======================================
  *  *  *  *  *   ======================================
*  *  *  *  *  *  ======================================
  *  *  *  *  *   ======================================
*  *  *  *  *  *  ======================================
  *  *  *  *  *   ======================================
*  *  *  *  *  *  ======================================
  *  *  *  *  *   ======================================
*  *  *  *  *  *  ======================================
========================================================
========================================================
========================================================
========================================================
```

Figure 2: An ASCII art approximation of the American flag.

- All `print` function calls must be inside the body of a loop.
- The characters must be printed exactly as displayed in the Figure above.
- Try to write this program as concisely as possible. I (Scott Wehrwein) wrote a solution that has 7 lines of code, not counting comments. If you're looking for a challenge, see if you can fit your program in a space smaller than the flag it prints (13 lines long, 56 characters wide).

3 `turtledraw.py`

In this section, you'll write a program `turtledraw.py` that creates a picture like the one shown in Figure 3. This may seem intimidating at first! But the code you'll need to write isn't actually that complicated. There's a lot of repetition in the picture, and you'll use loops to perform this repetition effortlessly.

You'll also get some practice looking at the documentation for a module; the `turtle` module has tons of functionality, most of which you won't use or understand—that's fine! The ability to sift through unfamiliar documentation and find out how to use the pieces relevant to solving your problem is a valuable skill.

Each pattern is identical, and is simply composed of many squares with side length 100, each drawn with one corner in the center of the pattern, but at different angles.

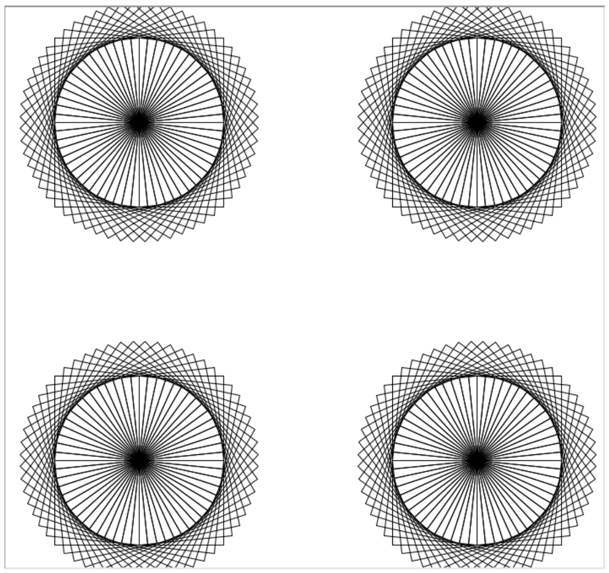

Figure 3: The result of running `turtledraw.py`

3.1 Drawing a Square

Start by writing code to draw a square with side length 100. Recall that you saw an example like this in lecture - you may want to download `square_for.py` from last Friday's lecture on the course webpage as a starting point. After this step, your program should generate the picture shown in Figure 4.

3.2 Drawing one pattern with many squares

Next, you'll create a single one of the patterns. To draw 60 squares that go all the way around a circle (360 degrees), you'll need each one to be different by 6 degrees from the previous one. Put the code for drawing a square inside a `for` loop that draws all 60 squares. When it works correctly, you should get a picture like the one in Figure 5.

3.3 Speeding up the drawing

At this point, you're probably sitting around waiting for the turtle to finish drawing, and thinking "wow, CS is boring". Most programs we write run very quickly, but you can see that

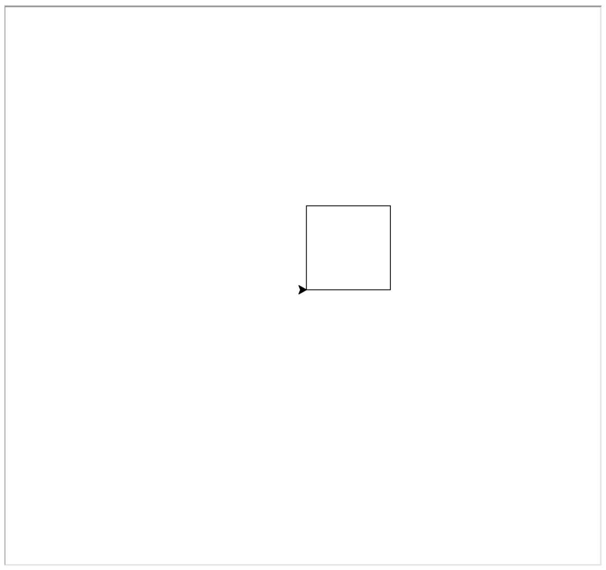

Figure 4: Drawing a square with side length 100

loops make it easy to write short programs that still take a long time to finish! Fortunately, the `turtle` module has some features to speed up the drawing. Start by looking in the `turtle` module's documentation (`https://docs.python.org/3.3/library/turtle.html`). There's a lot there, but don't worry about all the stuff you don't understand! Find the section on the `speed` method, and read the description to see if you can figure out how it's used. In your code above where your turtle starts drawing anything, call the `speed` method on your turtle object (e.g., `scott.speed(`*`arg`*`)`) with the correct argument to make the turtle move as quickly as possible. Try running your program again - do the squares draw faster now?

That helps some, but it still takes a few seconds to draw 60 squares! The reason for this is that Python is re-drawing the entire picture every time the turtle moves. You can think of this as Python having to loop over all 120,000 pixels and re-color each one. We can speed things up even more by telling the turtle not to re-draw between moves, and to draw once at the end.

Near the top of your code, before your turtle starts drawing, add the following line:

```
turtle.tracer(0, 0)
```

Notice that this a `turtle` module function, not a method of your turtle object. This disables re-drawing after each move. Next, at the end of your program, add the following line:

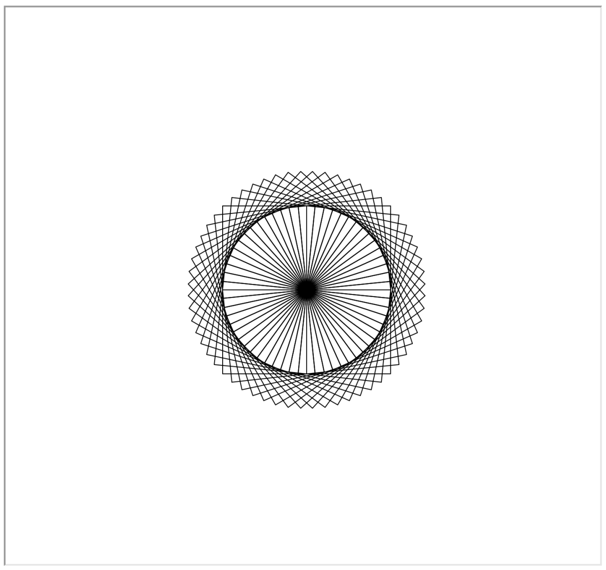
Figure 5: Drawing a single radial pattern.

```
turtle.update()
```

This tells Python to re-draw the picture to the screen. Now we're not wasting effort re-drawing each time the turtle moves! When debugging your code, you may find it helpful to keep the animations to see the sequence of moves the turtle makes. Make sure that the submitted verision of your code has the above lines and draws the whole picture in less than a couple seconds.

3.4 Repeating the pattern four times

The last thing we need to do is draw the pattern four times, near each corner of the window. In particular, draw the pattern centered at all four corners 200 units away from the middle of the screen (at coordinates (0,0), where the turtle starts). Start by looking at the documentation for the `penup`, `pendown`, and `goto` methods. Start by using these methods to move your turtle to position (-200, -200) before drawing the pattern. Make sure the turtle doesn't draw a line when it's moving from the middle of the screen to the corner.

Finally, we need to move the turtle to each of the four corners and repeat the same pattern. You could copy/paste the same code four times and change the coordinates, but that wouldn't be ideal if we wanted to draw 400 of these patterns instead. So let's use loops instead!

The coordinates where we want to draw the pattern are:

```
(-200, -200)
(-200,  200)
( 200, -200)
( 200,  200)
```

Notice that this is just all possible ordered pairs of -200 and 200. You saw an example of how to print all possible pairs of 1 through 6 in lecture; this is similar, but with -200 and 200 instead of 1 through 6, and instead of printing them, you're moving the turtle to those coordinates and drawing the pattern. In class we used `while` loops, but `for` loops could also work (and might be a more natural choice). You may choose whichever loop you like. The pseudocode I'd write for this is something like:

```
for i = each of {-200, 200}:
  for j = each of {-200, 200}:
    move to (i,j) without drawing a line
    draw the pattern
```

3.5 Color the drawing

Look up the turtle's `color` method and figure out how to make your drawing more colorful. You can color the drawing however you want (it could be as simple as making each pattern a different color), but **don't** change the pattern drawn. An example coloring is shown in Figure 6.

Submission

Submit `triangle.py`, `flag.py`, and `turtledraw.py`

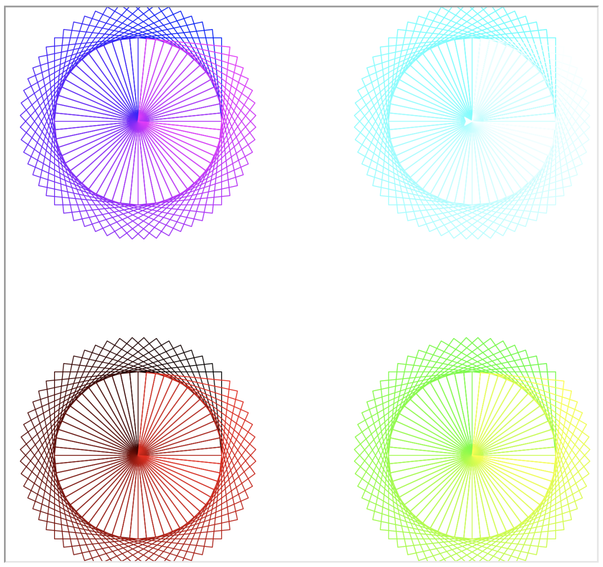

Figure 6: Sample colored pattern.

Rubric

The top of each program has comments including your name, date, and a short description of the program's purpose.	3
Variables are named well and explanatory comments are included as needed	1
`triangle.py` produces the correct output	6
`flag.py` produces the correct output `flag.py` uses 6 or less print statements, and no print statements outside a loop	5 5
`turtledraw.py` draws at least one instance of the pattern. `turtledraw.py` draws four copies of the pattern in the correct corner positions using loops. `turtledraw.py` draws the pattern quickly and without animation.	4 4 2
Total	30 points