Software Dev. & Problem Solving I CSEC/SWEN-123

Polygon Flower Practicum 1

Exam Requirements

This practicum is closed notes, closed internet. The only sites you may visit are MyCourses, GitHub Classroom, and Discord. You are also allowed to use the IDE of your choice (Visual Studio Code for most).

- The link for the GitHub Classroom repository is in MyCourses | Content | Instructor Materials | Herring | Section | Practicums.
- You are required to join the course Discord server and join the Practicum voice channel while taking the exam. You will remain on mute, do not ask questions via the voice channel. If you have any questions, please direct message me in Discord.
- You have 75 minutes to complete the practicum and push it to GitHub. The last push before the deadline will be graded
- You are expected to use incremental development, preferably with TDD. You must commit and push after each activity. Failing to do so will result in up to a **50 percent penalty**.
- **Do not** change the names of any of the provided functions.
- You must use the provided functions, but you may generate helper functions if you find them helpful.
- You must include the activity number in each commit message along with a brief description of what you are committing.
  Example: `git commit -m"Completed test for activity 3"`
- You do not need to comment your code except for adding your name to the top of each file you modify.

Activities

Before starting, take a moment to look over the provided code. You will have 3 files, `poly_flower.py`, `poly_flower_tests.py`, `testing.py`. Do not modify `testing.py`. There are function stubs for all of the practicum activities. Do not change their name or parameter lists. You are also provided with constants for colors and `init`, `show`, and `main` functions.

All tests must be placed in the `poly_flower_tests.py` file.

Each test only needs to test the return value of the function under test. This is a simplicity for the practicum to help ensure you have time to complete the exam.

1. (`validate_number (num_str)`) The function should validate that the supplied string (`num_str`) is a number. If the string is a number, return the number as an Integer. If it is not a number, return `None`.
    a. You should use `num_str.isdigit()` to determine if the string is a digit. It will return `True` if the entire string are all integer values and `False` otherwise
    b. Follow the TDD process to implement and test the function.
    c. This function requires two tests, one for a valid input and one for an invalid one. Both are stubbed out in the `poly_flower_tests.py` file.

2. (`get_color (color_code)`) The function maps an integer to a color (string). The input is an integer and the return is the associated color string.
    a. 10 color constants have been provided for you, use them.
    b. Color codes below 3 should always return `WHITE`
    c. Color codes above 10 should always return `BLACK`
    d. Follow the TDD process to implement and test the function.
    e. The associated test is for the color `GREEN`

3. (`draw_polygon (num_sides, length, color)`) The function draws a `num_sides` polygon where each side length is specified by `length` with the given fill color.
    a. The function should return the sum or all drawn sides, aka the perimeter of the polygon
    b. Rotate to the **right** by `360/num_sides` to easily draw any polygon.
    c. I would recommend calling `init` to speed up the turtle draw time.

d. You **must** implement the function using a **while** loop.

e. Follow the TDD process to implement and test the function. Only the return value needs to be tested for the practicum.

4. (`draw_poly_circle (num_sides, length, color)`) The function draws n-sided polygons in an n-sided circle where each polygon has a side length of `length` and a fill color of `color`.

   Example: `draw_poly_circle (7, 50, "blue")` would draw 7 blue septagons which overlap each other in a circular pattern.

   a. It returns the sum of the perimeters of all the polygons drawn while creating the circle.

   b. After drawing a polygon move forward by **half** the polygon's length and turn **left** by `360/num_sides` to draw the circle.

   c. The function **must** be implemented using a **for** loop.

   d. Follow the TDD process to implement and test the function. Only the return value needs to be tested for the practicum.

5. (`draw_poly_flower (side_length, start_sides, end_sides)`) The function draws `start_sides - end_sides` polygon circles, where each side of a polygon is `side_length` long. Each polygon circle will use a different color based on the number of sides in the polygon.

   a. `start_sides` is the number of sides the polygons in the first polygon circle will have.

   b. Each iteration (i.e. polygon circle), will reduce the number of sides of the polygon used to draw the circle by **one**.

   c. The last polygon circle will have `end_sides + 1` sides. I.E. the provided range (`start_sides, end_sides`) is inclusive of the first value but exclusive of the last value.

   d. The function returns the total perimeter of all polygons drawn.

   e. You **must** use a looping construct to implement the function.

   f. Follow the TDD process to implement and test the function. Only the return value needs to be tested for the practicum.

6. (`main ()`) Update `main` to ask the user to input the length of each polygon side, the starting polygon side count, and the ending polygon side count. Use this information to call `draw_poly_flower` and then print the total perimeter of all drawn polygons.

a. After collecting the data from the user, validate each component to make sure it is a number. If any element is not a number, ask the user to re-enter all of the information. Continue to ask until all three values are entered as numbers.

b. You **must** use a **while** loop to implement the repeated request for numbers.

c. There is no test for this activity.

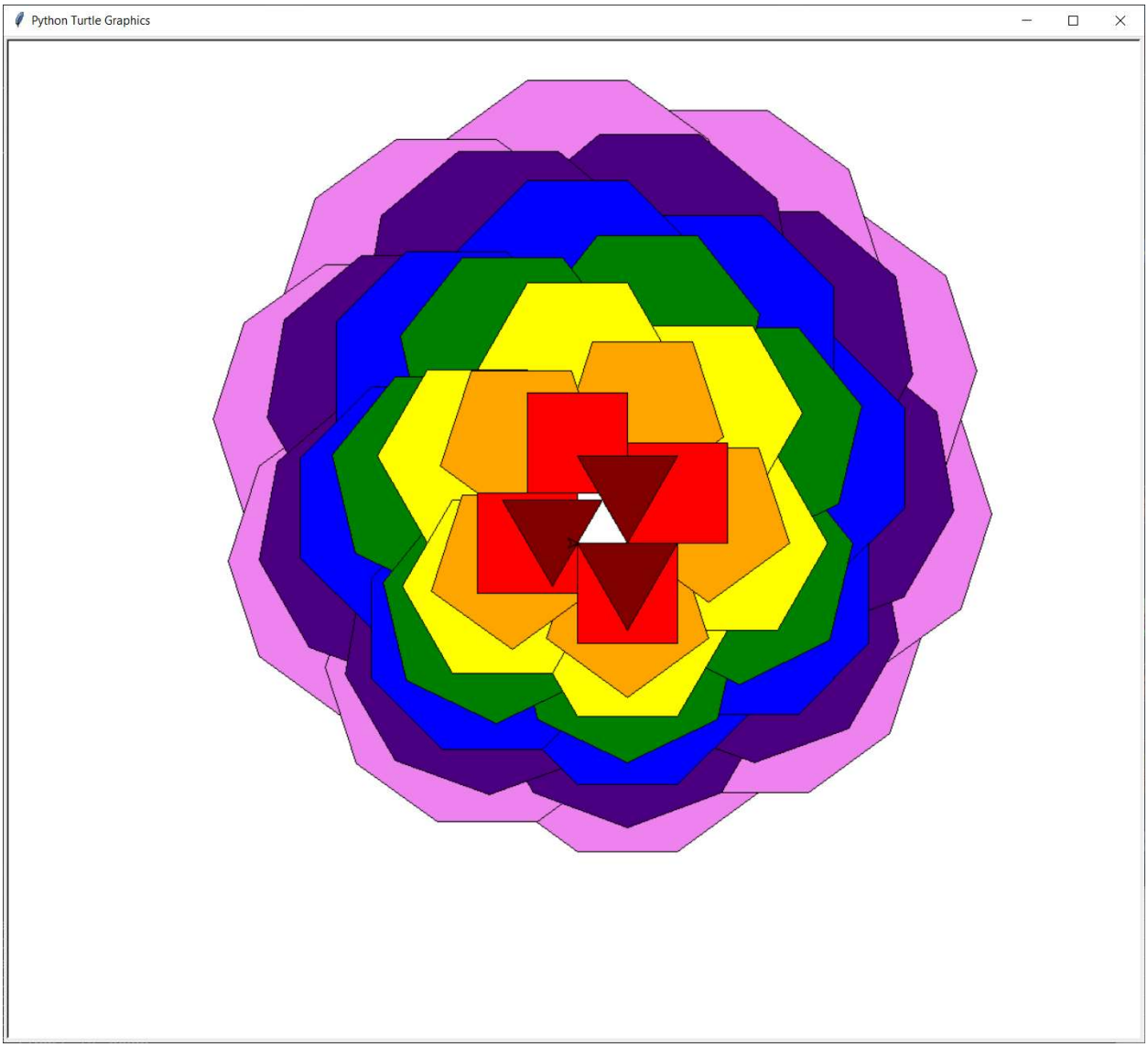

Figure 1: `draw_poly_flower (100, 10, 2)`

Submission Instructions & Grading

Be sure that you have committed all of your solution files to your repository ***within*** 75 minutes of the start time. Commits after the cutoff will not be pulled.

Grading Breakdown:

10% validate_number
10% get_color
10% draw_polygon
10% draw_poly_circle
10% draw_poly_flower
10% main loop

5% test_validate_number_valid
5% test_validate_number_invalid
5% test_get_color_green
5% test_draw_hexagon
5% test_draw_octogon_circle
5% test_draw_poly_flower
5% run_all_tests

5% Clear commit messages

Up to 50% penalty for not performing incremental development. You must have at least 1 commit per activity to receive full credit. So, at an absolute minimum that is 6 commits.