Lab: Graphics, Functions, and Local Variables

1. Create a program to draw a bullseye (looks like the Target logo) on the screen.

   - First, define a `main()` function that opens a canvas, draws *one* circle using `draw_filled_circle` of radius 50 and closes the canvas on a click. (Do this just to make sure your graphics are working correctly.)

   - Then, modify `main()` so it draws *five* concentric circles using the `draw_filled_circle` function. The circles should have radii of 10, 20, 30, 40, and 50, and should alternate colors (use `set_color`) so you get a bullseye effect (see the picture to the right). Hint: the circles must be drawn in a certain order.

2. Define a function called "`draw_bullseye`" that takes two parameters that will represent the x- and y-coordinates of the center of the bullseye. Add code to the body of `draw_bullseye` so that when called, it draws a bullseye at the (x, y) location specified by the parameters. Then, edit your `main()` function to call `draw_bullseye` twice with two different sets of (x, y) locations (pick any coordinates you'd like). The effect should be that your program should draw the bullseye twice at different locations on the canvas.

   Hints:
   - Your function definition line will look like this:
     ```
     def draw_bullseye(x, y):
     ```

   - You should keep the `open_canvas` and `close_canvas` function calls in `main`, because you only want those to happen once in the whole program, not once for every bullseye.

   - When you're done, there should not be any circle-drawing function calls inside `main` anymore; they should only be inside of `draw_bullseye`. All `main` should do is open a canvas, call `draw_bullseye` twice, and close the canvas on a click.

3. Modify your `draw_bullseye` function so it takes a third and fourth parameter. Call these parameters `color1` and `color2`. These parameters will be strings, and will allow the caller of the bullseye function choose two alternating colors for the bullseye. Modify the code in the bullseye function to use the `color1` and `color2` variables rather than the fixed colors you picked earlier.

   For instance, if the user wanted a bullseye centered at (100, 100) colored red and black, they should be able to call the function like this: `draw_bullseye(100, 100, "red", "black")`

   Once you make this change to the definition of `draw_bullseye`, make sure you also change the calls to `draw_bullseye` in main to reflect the additional arguments.

(turn over)

4. Modify your `main` function so the user can type in the (x, y) coordinates of the center of a bullseye they want to draw, and the colors they want to paint it.
    - This will require four separate input statements: one for x, one for y, and two for the colors.
    - ***The input statements should not go inside the bullseye function; they should be inside the `main` function, and the information typed in should be passed as arguments to the bullseye function***.

5. Modify your bullseye function definition to take a fifth parameter: the radius of the bullseye. You'll need to do some math to figure out what the radii of the nested circles should be changed to. Don't forget to modify the function call(s) in main to add in the fifth parameter (try different radii so you know the bullseye scales up and down in size).

6. Modify your `main` function so *three* bullseyes are drawn after the user inputs the four variables. The first bullseye should be drawn as in problem 4 (based off the inputs the user supplies), but the second and third bullseyes should be offset slightly so as to create the pattern to the right (which sort of looks like Mickey Mouse ears). The point of this problem is that if the user types in different (x, y) coordinates, all three bullseyes should shift locations on the canvas so the final drawing always looks like the one to the right.

7. **Challenge**: Write a function called `draw_square(x, y, side)` that takes as arguments the (x, y) coordinates of the center of a square and the length of a side. You should use `draw_line`, `draw_polyline`, or `draw_rect`. Remember: x and y should be the center of the square, not a corner.

8. **Challenge**: Write a function called `draw_nested_squares` that acts just like `draw_bullseye`, but the effect is a set of nested squares, rather than nested circles. This is more mathematically challenging. You can use `draw_filled_rect` for this.

9. **Challenge**: Modify your program so the user can choose the center of the bullseye using a mouse click rather than typing in the coordinates. Refer to the graphics library handout.

10. **Challenge**: Modify your program so the user can choose the center and radius of the bullseye with two mouse clicks (first click chooses the center, second click chooses a point on the border of the bullseye from which you can compute the radius). You'll need to use the distance formula for this one:

    The distance from point (x_1, y_1) to (x_2, y_2) is $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. To use the square root function in Python, put `from math import *` at the top of your program, then you can use the function `sqrt()`.

(turn over)