Question 1 is all about basic PGL objects and placement. Question 2 is about working with coordinate positions in PGL, while Question 3 introduces you to mouse-driven events in PGL. Note that I've given you an extra day to get this turned in owing to the late posting of this assignment.

Get Assignment link: https://classroom.github.com/a/PnjxcCY-

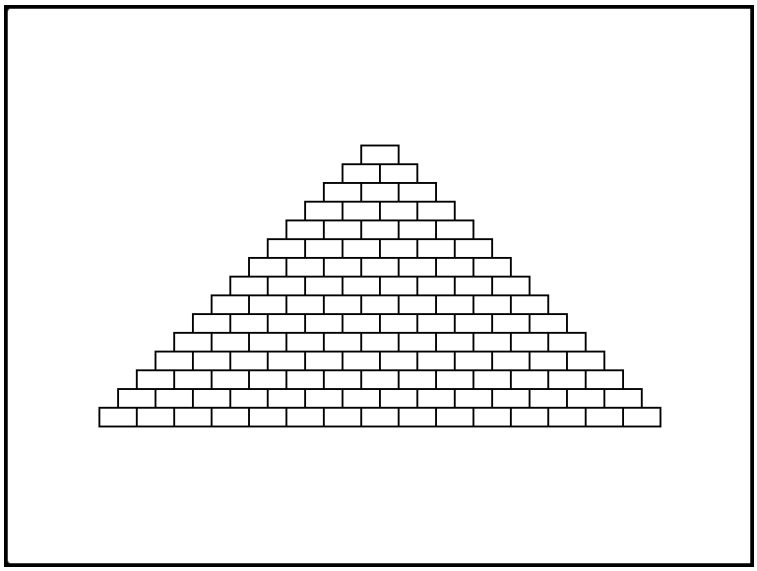
1. (7 points) While printing content or inputting content from the terminal is nice, often times you want to have more control over graphic elements of your program. To that end, we are using the `pgl` library in class this semester. To start things off in a very simple manner, in this problem you will just need to draw a pretty picture of whatever you might like. A few qualifications though to get full points:

   - It must be a coherent picture. No purely abstract art comprised of random shapes.
   - You must use multiple colors
   - You must use multiple types of `GObject`s (ovals, rectangles, lines, etc).
   - You must define at least one function which groups together some code relating to a particular object (or objects) in your image (for instance, a function to draw a tree at some location, or a cloud, etc). The function must take some form of input in the form of arguments. It can not, however, just be a helper functions we've provided, such as `draw_filled_rect`, though you are free to use those as well.
   - You must use comments or docstrings to label what different functions or parts of your code are responsible for drawing.
   - You must use a loop to draw some repeating portion of your image.
   - You must title your masterpiece at the top or bottom using a `GLabel` centered horizontally within the window.

   If you need a list of known colors in `pgl` I've taken the time to give you a visual chart on the last page of this document! Or you can use something like a color picker to get the hex value for a color (starts with a # symbol) and provide that string (including the #) directly to the `set_color` method.

   As a bit of an example, below is my creation, which uses purely lines, circles, rectangles, and a label:

2. (10 points) Complete the template given in `Prob2.py` to display a pyramid of rectangles on the graphics window. The pyramid consists of bricks arranged in horizontal rows, arranged so that the number of bricks decreases by one as you move upward, as shown in the following image:



The pyramid should be centered in the window both vertically and horizontally. You should be able to change the following constants in your program and have it react appropriately:

`BRICK_WIDTH`	The width of each brick
`BRICK_HEIGHT`	The height of each brick
`BRICKS_IN_BASE`	The number of bricks in the base row

The tricky part of this problem is always in getting the position coordinates correct. As a series of stepping stones, I might suggest:

- First writing code to generate a solid grid of rectangles
- Adjust one of your loops to get a triangle of rectangles, but where they are still stacked on one side
- Adjust to get the rectangles displaced appropriately to center them on the previous row
- Adjust initial coordinates to ensure the entire thing is always centered

3. (8 points) The goal for this problem is to create a simple clicking type game that might amuse a 4-year-old, or maybe a cat (or maybe a college aged individual, I'm not judging!). The program will run by displaying a square on the screen. When the square is clicked, it will move to a different random part of the screen, and the process can be repeated. You are welcome to use code from other libraries you might have written, but make sure to upload those libraries along with this code back to GitHub. You have been provided a starting template named `Prob3.py`, and I'll provide for you the following steps.

   (a) Add a colored and filled square to the center of the window. You can choose the color of the square, but it should have a width and height as provided by the constant `SQUARE_SIZE`. You will only be altering the properties of this object, *not* reassigning it, so you don't need to add it as an attribute to the `GWindow`. Make sure your square is displaying centered in the screen when you run your program before continuing.

   (b) Add a listener to your window which will listen for when the user presses down the mouse button, and calls the `on_mouse_down` function when that occurs. Run your program and ensure that now, when you click the mouse in the window, a message appears saying as much on your terminal!

   (c) Now erase the print function inside `on_mouse_down` and add code so that if (and only if) you click *inside* the square, the square moves to a new random position that is *entirely* within the window bounds (no squares should ever be sticking outside the window!). It can be fun to make the color of the square change to a new random color as well, but that is optional. Make sure that **nothing** happens if you click outside the square, it should only move if you clicked within the square boundary. There are several ways you can check to see if the mouse was clicked within the confines of the square, some easier and some harder. You may want to look at our Python Summary to refresh your memory about some functions that may be useful.

   (d) Lastly, every time that you click inside the square, you want to add a point to the score, and every time you click outside the square, the score should reset to 0. There are a few ways you could handle this. One would be to keep track of the score purely inside a `GLabel`, getting and setting the text of the label as necessary. Another would be to create a variable which you would increment or reset as needed, and then update the `GLabel` from that variable. Just be aware that if you go the variable route, you will need to add that variable as an attribute to the `GWindow`, else you won't be able to globally set its value within the callback function. Your label depicting the score should be placed in the bottom left corner of the window: `SCORE_DX` from the left wall and `SCORE_DY` up from the bottom.

   I'm including an animation of the game being played in the template repository if you want a visual to compare against!

Figure 1: All available named colors in `pgl.py`