Scene

The goal of this assignment is to practice using the graphics module, *for* loops, and simple control structures (`if/else` statements). You should complete this assignment using good pair-programming techniques with your assigned partner from partners1 on Moodle.

Deliverables: your solution file named `username1_username2.py` Please be sure to use this naming convention so that the grader can easily see from the file the 2 partners that worked on the project. Also don't forget to include a comment at the top of your .py file with both authors names!

Your task:

Use Zelle's graphics module to write a program that draws a scene. Remember you must have the file `graphics.py` in the same directory as your program and to import graphics at the top of your file. Your scene can be as simple or as complex as you like, as long as it meets the basic requirements. I do not expect any art skills whatsoever (I myself have basically none), but it's worth spending a little time playing around with the size, color, and position of even basic shapes in order to create some recognizable picture (e.g. a rectangle and 2 circles to form a car)…

The minimum requirements for your scene:

- uses at least 3 different shape objects from the graphics package (the 5 possible options are *line*, *circle*, *oval*, *rectangle*, and *polygon*).
- includes a stationary background of which some portion is sky (see the next part of the assignment for how the sky's color will be determined by the user running your program), the sky portion could be most of the scene, or could just be a small bit seen through a window of some sort.
- includes something animated - this should be something simple made of only a few shapes at most so that it doesn't slow your program down too much. Examples from the past include a puff of smoke moving up from a chimney, a simple V shape to represent a bird that flies across the sky, or a basic fish made up of just an oval and a triangle that swims.

In applications like CG animation, a single scene may be used as the general "set" for much of a story, though the time of day or current weather constantly changes. It is useful, therefore, to be able to change only the sky portion of a given scene based on whatever current conditions are desired. For this project we will only worry about changing the background sky color based on the time of day that is input by the user. For simplicity we will assume that 12pm (noon) is the brightest time of any day, so the sky at 12pm should be pure white (rgb values 255, 255, 255). The darkest sky should occur at 12am (midnight) and be pure black (rgb values 0,0,0). The times in between should be varying shades of gray (any values where r, g, and b are all equal will yield a shade of gray).

You will ask the user to enter an integer 1-12 indicating the time of the day (only the hour, we are ignoring minutes), and then to enter 'am' or 'pm'. Your scene will be drawn with the appropriate sky color based on the user's input. Note that from 12am to 12pm the sky should get lighter each hour, while from 12pm to 12am the sky should get darker each hour. You should scale the lightness/darkness of the sky color more or less linearly, so at 6pm and 6am the sky should be approximately the rgb color (255/2, 255/2, 255/2), which should look like a gray halfway in between black and white.

You could write this program using something like 12 "if" statements, however there are much more efficient methods (remember your modulo division with the % operator), and for full credit you must do

something more efficient. However, a few “if/else” statements are fine (even necessary) to deal with the edge cases of 12pm and 12am.

Check out the 2 videos I have included as examples of what your program should do. Note in the terminal I entered a time and am/pm, then the program opened a graphics window with the scene using an appropriately shaded sky and immediately running the animated bit (super simplistic birds in my example).

Bonus (2pts):

For a 2 point bonus, automate your scene to cycle through the 24 hours of a day while running your animation. If you choose to do this add an option for the user to enter the value -1 for the time, in which case the automated 24-hour cycle will run, otherwise your program should still run correctly as described above! See the video I provided for an example of this working, though note I didn’t hit record until a couple of seconds into the program so the video starts at 3am or so, and I quit before it reaches the end of the 24 hour cycle.

If the user does enter -1, you should start the scene at midnight (12am), and tick through the next 24 hours pausing for 1 second on each hour, so it will take ~24 seconds total for your program to finish the entire cycle. You can make your program “sleep” for some amount of time by importing the *time* module and then using the *sleep(num_seconds)* function. So if you want to sleep for 1 second you could say *time.sleep(1)*. You may also sleep for a fraction of a second, e.g. *time.sleep(.01)*. The tricky part is coordinating the color change with your animation loop, as you will want your animation loop to run several times in the second between each color change. You will have to think about loops within loops, where your animation loop runs many times in each tick of your time loop.

You will receive 1 bonus point if you get just the automated 24-hour cycle working without any animation.