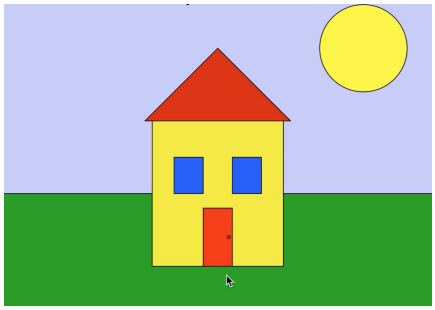
COM 110 Fall 2019 - Lab 4
Graphics and Animation

1. Write a module that creates a graphical window object (i.e., a GraphWin object) at least 600 x 600 pixels large. We will be drawing things in it for the remainder of the lab. Add code to make the window close upon a user click. (Hint: Remember the `getMouse()` method for GraphWin objects? Don't forget you'll need to import the graphics.py module too.)

☺Get check 1☺

2. In the coming checkpoints, we will add to your module so that when the graphics window opens, it draws a scene with a house.

   Here is some inspiration, but your design is up to you (NB: this is cropped just to show the sun and the house):



   a. Start by adding a green lawn and a sun in the sky. (You could use a big Rectangle object for the lawn and a Circle object for the sun. See section 4.8 for a summary list of the various Zelle graphics objects and their methods.) You will color in each piece of the scene using the `setFill(<color>)` method on each object you draw. (For a list of possible colors, see this page.)

☺Get check 2☺

   b. For the house itself you can use a Polygon (or Triangle) object for the roof, a Rectangle object for the base, and more Rectangles for the door and windows. See the text for examples of how to use these Zelle graphics objects. (Note: this task is much easier if you first take some pen and paper, draw the graphics window dimensions on it, then plan out the position of each of the objects you'll be drawing.) You can also add a chimney or anything else you'd like. Feel free to take creative license with this drawing. (But don't spend *too* much time on it for now; you can always come back to perfect things when you've completed all the checkpoints!)

☺Get check 3☺

   a. Animate the sunset upon a user-click by simply moving the sun off the screen. Use the `move(<dx>,<dy>)` function to achieve this. E.g., `obj.move(10,20)` moves the object 10 pixels to the right and 20 pixels down. First try just a single call to `move()`. The units for the parameters of the `move()` function is in pixels, so, considering the resolution of your screen, a value of 10 is a very tiny amount of movement. And it may happen so quickly that you don't even notice it. Try a larger value so that you can actually see that the sun is in a different location than where you initially drew it.

b. Then, to make the animation smooth, use a `move()` call inside a `for` loop, as we did in class with the square. Since you will be doing many repeated moves now, to achieve the effect of one smooth motion, you want each move to be very small. And then make sure you make enough of them (but not too many, since the object continues moving even when it's off the screen) by setting the range of your for loop just high enough.
c. The `sleep(s)` function freezes the program for `s` seconds, so it can be helpful to insert a `sleep()` call after each `move()` call if your object is moving too quickly to see it clearly. Remember that to use `sleep` you will need to import it from the time module, like this:

```
from time import sleep
```

The units for the parameter of the `sleep()` call is in seconds. So if you sleep for 0.1 seconds between each move, that is actually a very long time and your animation will look choppy. Hundredths of a second usually work better.
d. Adjust the smoothness, speed and distance of your sunset by tuning the range of your for loop, the parameters of your `move()` call and the parameter of your sleep call.

☺Get check 4☺

e. After the sun sets, change the sky color from "cyan" (or whatever sky color you have chosen) to "black," indicating night has fallen. This amounts to adding one more line of code to your program. (Hint: see 4.8.2 for the method that will help.) A bonus check will be to have the sky slowly fade to black, but that is not part of this check point.

☺Get check 5☺

3. For the next two checkpoints, in a new module, we will draw a solid square one Point at a time by using nested for loops. This will be very similar to, but a slight departure from, the way we did it in class. (In class we did it row by row, here we will do it column by column.)
   a. First open the program we wrote in class that did virtually the same thing (but row by row). It was called graphicswarmup.py. Make sure this code works. Review it, and let us know if you have any questions about it.
   b. Now, in a 600x600 graphical window, just as we did in class, use a loop to draw the top horizontal line of the square. Make it 100 pixels wide, starting at the 250th pixel over and the 200th pixel down.

☺Get check 6☺

   c. Then, modify your code so that in each iteration of the loop, for each of these 100 different x coordinates, rather than draw a single Point, as you have done for check 6, draw an entire vertical column of them. This will require you to nest a for loop inside your for loop. Each vertical column should be 100 pixels tall, with the top pixel at y=200 and the rest going downward from there, ending at y=300.

☺Get check 7☺

4. Read section 4.8.5 on page 121 of Zelle. Color the square your favorite color using the `color_rgb()` function. Note that lower integer arguments for the red, green, and blue

parameters mean darker colors, and higher integers mean lighter colors. E.g., `color_rgb(0,0,0)` is pure black, while `color_rgb(255,255,255)` is pure white. If you need help with figuring out how rgb parameters work, you can try this page: http://www.rapidtables.com/web/color/RGB_Color.htm

☺Get check 8☺

5. Do a "Save Copy As..." to save out a separate copy of your work before going on. (It's always good to do this between all checkpoints anyway.) Now modify your code so that it is colored with a randomly generated color. (To do this you will need to generate **three** random integer values between 0 and 255, one for each of the **three** arguments of your `color_rgb()` call. To generate a random integer value, see the `randrange()` function on page 287 of your text. E.g., `randrange(1,100)` returns a random integer between 1 and 99. You will need to import the `random` package to use `randrange()`.) The color of your square will now be different randomly-generated color *each time you run the program.*

☺Get check 9☺

Extra time? (You may get these bonus checks one at a time, in any order, as usual):

A. First save out a copy of your program. Now modify the code so that each *column* of the square is a different randomly generated color.

B. First save out a copy of your program. Now modify the code again so that each *pixel* of the square is a randomly generated color.

C. If you started your loop for checks 6&7 like this:

```
for x in range(250,351,1):
```

figure out how to modify your code so that it so it starts like this:

```
for x in range(100):
```

without changing the behavior of the program.
If your loop was already written the second way, modify it so it starts the first way.

D. In your house animation you can make the color of the sky change gradually from a daytime color to night fall as the sun is setting. Probably the easiest way to do this is using `color_rgb(<red>,<green>,<blue>)` which allows for more refined color control. See section 4.8.5 on page 121 of Zelle for help with it.

E. The .move(dx,dy) method we have been using for animating Zelle graphics objects, basically has the effect of undrawing an object from its current position (x,y), then redrawing it in the new position (x+dx, y+dy). In fact, there *is* an .undraw() method we can call to "undraw" any Zelle graphics object we have already drawn.

Take something you have previously animated using the .move(dx,dy) method and instead animate it by repeatedly using .undraw() and .draw(). Note that the parameters for .move(dx,dy) indicate a *relative* change in x and y value, while the fresh creation and placement of an object uses *absolute* (x,y) coordinate values.