Playing with graphics in Python

Before you start:

1. Read through graphics_notes.pdf
2. To find documentation about the graphics library, you can look at the Sections 2,3,6 of the graphics.pdf
3. If you want to change colors of objects throughout this exercise, we provided you rgb codes inside of the rgb.txt

Exercise 1: Drawing a Wheel

Exercise 1.1: Graphics setup

1. You can find graphics.py in the folder graphics_lab
2. Make sure that the file appears in your working directory
3. Run the module as if it were a Python program (python graphics.py).
4. If everything was done correctly you will get a demo window with a triangle and some text.
5. The graphics module does not work well with IDLE. So for all graphics programs your write, you can still use the IDLE editor, but run your program from the terminal window

Exercise 1.2: Basic Graphics Application

Here is a skeleton program of any new graphics program for this class.

```
from graphics import *

#add any functions or classes you might define here

# create a window with width = 700 and height = 500
win = GraphWin('Program Name', 700, 500)
```

```
# add your code below this point

win.mainloop()
```

Exercise 1.3: Animating the wheel

Create a wheel.py file and copy the code there.

```
from graphics import *

class Wheel(object):

    def __init__(self, center, wheel_radius, tire_radius):
        self.tire_circle = Circle(center, tire_radius)
        self.wheel_circle = Circle(center, wheel_radius)

    def draw(self, win):
        self.tire_circle.draw(win)
        self.wheel_circle.draw(win)

    def move(self, dx, dy):
        self.tire_circle.move(dx, dy)
        self.wheel_circle.move(dx, dy)

    def set_color(self, wheel_color, tire_color):
        self.tire_circle.setFill(tire_color)
        self.wheel_circle.setFill(wheel_color)


    def undraw(self):
        self.tire_circle .undraw()
        self.wheel_circle .undraw()

    def get_size(self):
        return self.tire_circle.getRadius()

    def get_center(self):
        return self.tire_circle.getCenter()
```

Now let's try to add an animate method that would move the wheel across the screen. We will make use of the move method in the wheel class that moves the object dx units in the x direction and dy units in the y direction. Here is what the animate method will look like.

```
from graphics import *

class Wheel(object):
    ...
    def animate(self, win, dx, dy, n):
        if n > 0:
            self.move(dx, dy)
            win.after(100, self.animate, win, dx, dy, n-1)
```

The animate method has 4 parameters - a GraphWin object win, the units by which to move the object in the x and y directions, dx and dy, and the number of times to call the animate method, n. The animation will stop when n = 0. The interesting part here is the after method on the GraphWin object. The first parameter is the time in milliseconds after which the GraphWin object will call the animate method again. The second parameter is the function/method object the GraphWin object needs to call, in our case it is the animate method on the Wheel object. As we mentioned in class, in Python everything is an object even functions/methods and they can be passed as parameters to other functions/methods. The rest of the parameters are the new parameters to the animate method. Note that we decrement n by 1 every time we setup a new call to animate.

Now write a program that will use the update Wheel class and create a Wheel object (you can pick the colors of the tire and wheel to be anything you want) and make it move the wheel across the screen by 1 unit in the x direction 100 times. Remember you first need to draw the wheel before you can move it.

Exercise 2: Car

Exercise 2.1: Drawing Rectangles

To display a rectangle, you need to specify two points: the upper left corner and the bottom right corner. Remember our y-axis is flipped.

Try the code below:

```
from graphics import *

win = GraphWin("Rectangle", 300, 300)

rect = Rectangle( Point( 10,10), Point(200, 100 ) )
rect.setFill( "blue" )
rect.draw( win )

win.mainloop()
```

Run your program and make sure that the rectangle appears on the screen.

Try changing the color and width of the outline of the rectangle. Look at the setOutline and setWidth methods.

Exercise 2.2: Drawing the car

In this exercise, we will create a class for a car that will use the Wheel class from exercise 3. The car will contain 3 attributes: two wheel objects and one rectangle object (the body of the car) that is horizontal and whose bottom corners correspond to the centers of the wheels. Below is an example on how to use the Car object. Try to figure out what the class Car should be based on the way it is used.

```
win = GraphWin('Car', 700, 300)

# create a car object
# 1st wheel centered at 50,50 with radius 15
# 2nd wheel centered at 100,50 with radius 15
# rectangle with a height of 40

car1 = Car(Point(50, 50), 15, Point(100,50), 15, 40)
car1.draw( win )

# color the wheels grey with black tires, and the body pink
car1.set_color('black', 'grey', 'pink' )

# make the car move on the screen
car1.animate(win, 1, 0, 400)

win.mainloop()
```

The size of the wheel is given only by the radius of the tire circle. You can compute the radius of the wheel circle as a percentage of the radius of the tire circle, e.g. 60%. Save your code in file car.py.

Exercise 3: Sierpinski Triangle

Read about Sierpinski Triangles here:

http://en.wikipedia.org/wiki/Sierpinski_triangle

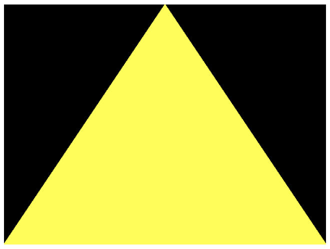
Our goal is to use graphics.py to construct Sierpinski triangles. In order to draw triangles using this module, you need to construct Polygon object by specifying three corner points of your triangle, and then draw it.

The following code draw a yellow triangle, copy it and try it:

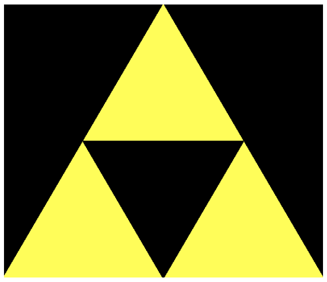
```
from graphics import *

# create a window with width = 700 and height = 500

win = GraphWin('Program Name', 700, 500)
tri = Polygon(Point(0,500),Point(350,0), Point(700,500))
tri.setFill('yellow')
tri.draw(win)

win.mainloop()
```

Construct a recursive function that takes level of recursion, and three starting points of the large, bounding triangle as arguments and draws Sierpinski triangles.

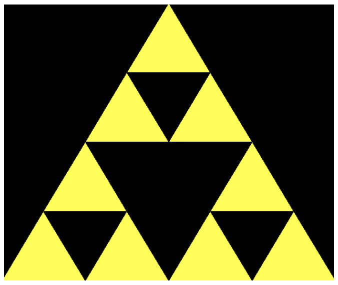
So if we run the function for level = 1, we should get:



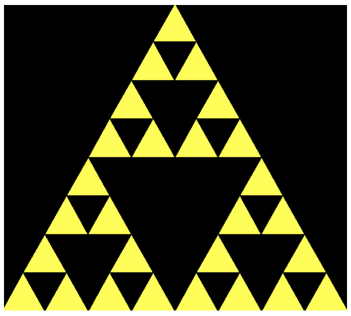
For level = 2:



For level = 3:



For level = 4:



etc.

If you need additional clarification or help, seek us out! Be as creative and as colorful as you want. You can change colors, you can try different shapes, not only triangles. If you obtain some cool results, let us know, we would love to see them.

Exercise 3: Digital Clock

Exercise 3.1: Drawing Text

The code below is an example of how to draw text on the screen.

```
from graphics import *

# create the graphics window
win = GraphWin("Digital Clock", 300, 300)

# create a text objects centered at (100, 100)
msg1 = Text( Point( 100, 100 ), "Hello, AITI Kenya!" )
msg1.draw( win )

# process events
win.mainloop()
```

Run your program and make sure the string prints on the screen.

Try changing the font size and style and the color of the text. Look at the setSize, setStyle, and setTextColor methods in the documentation. All the set methods that change the attributes of the graphics object, automatically update its appearance on the screen. You can use the list of colors available in the rgb.txt linked from the course website.

Exercise 3.2: Drawing a Digital Clock

Create a class called DigitalClock in a file called digitalclock.py that has attributes hour, minute, second and pos, and a draw method. The attributes store the time in military time, i.e. 3:30pm will be hour = 15, minute = 30, second = 23 and the position - the upper left corner of the rectangle face. Here is the code on how to use it:

```
from graphics import *

# DigitalClock class definition goes here

win = GraphWin("Digital Clock", 300, 300)
clock = DigitalClock(15, 30, 23)
clock.draw(win)

win.mainloop()
```

Feel free to choose the appearance of your clock :)

Hint: You should add extra methods to help you draw the clock, e.g. a method for drawing the face, a method for drawing the text, a method returning the time as string. Choose appropriate names for your methods.

Exercise 3.3: Updating the clock

Now you probably created a text object to display the time. Make it an attribute of the clock. Then add an update method that will update the time - both the object attributes and the display on the screen. Think about how you would increment the time. You may want to add other methods to help you. Take a look at the setText function on the Text class. *Note* The setText method will automatically redraw the text for you. You do not (and should not) call the draw method on the Text object again. You can only draw an object to the screen once.

You can create a tick method that would call update every second similar to the animate method in the previous exercise.

Update the program from 4.3 to start running the clock.

Hint: One thing you will have to worry about is handling scenarios, e.g. 05:35:59. The next time the clock updates it should show 05:36:00, not 05:35:60. Similarly for the minutes and hours. The modulus operator is your friend here.

Another hint: Here is an easy way to avoid trying to handle a lot of different cases. When you update, you first convert the time into seconds from the beginning of the day, then do the update, and then convert back to hour, minute, second. For example,

Current time: 01:01:01 ==> 1*3600 + 1*60 + 1 = 3661
Update time: 3661 + 1 = 3662
New time: 3662 ==> 01:01:02

Now you will only need to worry about how to handle updating 23:59:59 to 00:00:00

You may want to add extra methods to help you with this functionality - e.g. a method for converting the time to seconds, a method for splitting it back into hours, minutes, seconds, etc.