INSTRUCTIONS

1. For these problems, you are expected to write python files (with a .py extension), rather than doing them as .ipynb files. Run all your programs from within one folder in your computer. You can use an editor of your choice (e.g. Sublime, Visual Studio Code, etc) to create your code files.
2. NB: at the end of Exercise 4 and 5, there is a challenge asking you to add something unique to your program, something that differentiates your work from what other students have done.
3. When submitting your work, submit two files:
    a. a zip file of the folder with the python files you will have created. The zipped folder will also contain the short answers of Exercises 1 and 2.
    b. A text document briefly describing the creative and unique features you have added to Exercise 4 and 5.

Exercise 1: Short answers

1. What is the difference between a local variable and an object's attribute?
2. What method is called when the object is created?
3. If you have an object instance, `obj`, and you want to call its `do_something()` method (assuming it has one), how would you do this?

Exercise 2: Understanding Objects

1. Write a class called Address that has two attributes, **`number`** and **`street_name`**. Make sure you have an `__init__()` method that initializes the object appropriately.

2. Consider the following code:

```
class Clock(object):
    def __init__(self, time):
       self.time = time

    def print_time(self):
       time = '6:30'
       print(self.time)

clock = Clock('5:30')
clock.print_time()
```

    a. What does the code print out? (Do not run the code)
    b. Create a Python file with the code above and run it. Is that what you expected in 2a above? Why?

3. Consider the following code:

```
class Clock(object):
    def __init__(self, time):
       self.time = time

    def print_time(self, time):
       print(time)

clock = Clock('5:30')
clock.print_time('10:30')
```

   a. What does the code print out? If you aren't sure, you can create a Python file and run it.
   b. What does this tell you about giving parameters the same name as object attributes?

4. Consider the following code:

```
class Clock(object):
    def __init__(self, time):
       self.time = time

    def print_time(self):
       print(self.time)

nairobi_clock = Clock('5:30')
cairo_clock = nairobi_clock
cairo_clock.time = '10:30'
nairobi_clock.print_time()
```

   a. What does the code print out?
   b. Why does it print what it does? (Are `nairobi_clock` and `cairo_clock` different objects? Why or why not?)

Exercise 3: Drawing a Wheel

Exercise 3.1: Graphics setup

1. Download **graphics.py** available on the same Drive folder linked to.
2. Copy the file into your working directory.
3. Run the module as if it were a Python program (**python graphics.py** from the command prompt).
4. You should get a demo window with a triangle and some text.
5. To find the documentation of the graphics library, you can look at the Sections 2, 3, 6 of the **graphics.pdf**, also available in the same Drive folder linked to.

6. The graphics module does not work well with IDLE (in case you are using IDLE). So for all graphics programs you write, run your program from the command prompt.

Exercise 3.2

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

```
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 3.2: Animating the wheel

Create a **wheel.py** file and type this code there.

```
from graphics import *

class Wheel(object):
    def __init__(self, centre, wheel_radius, tire_radius):
       self.tire_circle = Circle(centre, tire_radius)
       self.wheel_circle = Circle(centre, 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_centre(self):
      return self.tire_circle.getCentre()
```

Now add an **`animate()`** method that would move the wheel across the screen. 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):
            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.

In Python everything is an object, even functions/methods; they can therefore 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 set up a new call to **`animate`** (remember recursion and the base case we looked at in FEE231!).

Write a program that will use the updated Wheel class and creates 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 4: Car

Exercise 4.1: Drawing Rectangles

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

Run 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()
```

You should see a rectangle appear on the screen.
Try changing the color and width of the outline of the rectangle. Look at the **`setOutline()`** and **`setWidth()`** methods.

Exercise 4.2: 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 centres 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 centred at 50, 50 with radius 15
# 2nd wheel centred 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 as a percentage of the radius of the circle, e.g. 60%.

Challenge:
Animate your car in a way that is creative and unique to you/your group e.g. make it move forward and then in reverse; or make it move up and down the window; or to move diagonally; or to move to the end of the window and then jump onto a second window in which it continues moving; or to have two cars which move in opposite directions and then collide and get deformed somehow. Let your imagination run wild and do something interesting with these simple shapes and animations.

Save your code in a file called **car.py**.

Exercise 5: Digital Clock

Exercise 5.1: 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 object centred at 100, 100
msg1 = Text(Point(100, 100), 'Hello, EIE Dept!')
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` which is provided in the same Drive folder.

Exercise 5.2: 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. Be creative :-)

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 5.3: 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 of the **Text** class. **Note** that 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 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.

Challenge:

- Update your program to enable it to pick the system time and display that instead of an arbitrary hardcoded time.
- Add something creative and unique to your clock, something to differentiate it from other clocks.