Computer graphics - lab exercise 2

1. Using vectors for motion and direction

Make a class that holds the necessary variables to represent a vector. To begin with just x and y should be enough.
Make another class for a point.
Now initialize a point that represents the position of a particular object and draw that object in the correct position.
Next initialize a vector that represents the motion of that object.
In each frame add the vector to the position.
Add keyboard and/or mouse input that affects the motion vector.
Now you control the motion of this object using a vector.

2. Time management in games

In order to get an even speed on different devices, but still get an experience that is as smooth as possible on each of them we can not keep moving objects an even amount in each frame. Instead we need a way to find a way to move each object at an even speed (even amount per second - or other time unit) regardless of frame-rate. In order to do this we check at the beginning of each frame's rendering how long it took to draw the previous frame, then we use that value as a multiplication factor on all our motion in this frame.
In PyGame we have the following operations:

```
self.clock = pygame.time.Clock()
```

The operation Clock returns an instance of a class that we can use to access time operations. In this case we are using a class variable in a game class to store this instance.
Put this somewhere in you initialization. This only happens once per game run.

```
def update(self):
    delta_time = self.clock.tick() / 1000
```

delta_time will now get a value equal to the time elapsed since tick() was last called on this clock instance. In this case it will be in seconds because of the division by 1000, but the tick function itself returns milliseconds.
Put this at the very beginning of your update routine. Then send this value into any other objects that need to update themselves based on time.

Now, instead of adding your vector directly to your point in the update() operation, multiply it with the delta_time first.

```
    position.x += motion.x * delta_time
    position.y += motion.y * delta_time
```

3. Building a vector from an angle and size (speed)

In many cases it can be important to keep track of the rotation of an object independently of its speed. We will now look at a case where we use a vector to affect the position of an object, but we use the angle to draw it rotated correctly.

Make a variable for an angle and make keyboard input that changes this angle, using correct time management. Something like 180 degrees per second might be a nice place to start.

Now draw an object (preferably not too symmetrical so you can see its rotation) at a location represented by a Point object. Also set a specific speed in a single number variable.

Every frame you now need to build the vector using the angle and the speed.

```
    motion.x = speed * cos(angle * 3.1415/180.0)
    motion.y = speed * sin(angle * 3.1415/180.0)
```

Python's math.sin() uses radians, so we convert using PI/180

Now, when drawing we can use the angle directly in a particular OpenGL operation:

glRotate(angle, 0, 0, 1)

Actually 0, 0, 1 could be any vector that you wish to rotate around, but in 2D this is the only one that makes sense.

Now, this rotation may yield weird results, so let's look at other transformation operations in OpenGL:

glTranslate(x, y, z)

glScale(x, y, z)

Any calls to glVertex that are called after these operations will be affected by these transformations. Experiment.

In order for these operations to affect only the vertices that you want it is good to use

glPushMatrix()

glPopMatrix()

glPushMatrix stores the current transformation and glPopMatrix restores the last "pushed" transformation. Make sure that when you want these to affect the transformations that you have previously called:

glMatrixMode(GL_MODELVIEW)

Keep experimenting with more instances of the shapes and colors and transformations. See if you can have shapes orbit around other shapes or move in relation to another shape, while the original shape (and the whole system) move in a different way.