## [Pygame tutorial #7: more on lines](http://lorenzod8n.wordpress.com/2008/02/19/pygame-tutorial-7-more-on-lines/%22%20%5Co%20%22Permalink%20to%20Pygame%20tutorial%20#7: more on lines)

This tutorial is a bit of a special. I’ve heard from several people who have had difficulty with *Drawing lines* – Exercise 6 in tutorial #2. So I am going to go through how to solve it here.

**Understand the problem**

To understand the problem we are trying to solve we need to draw a bit so get out your pen and paper. If you have square ruled paper, it might make things a bit easier for you. Draw a horizontal and a vertical line as in the picture below.



You can of course use more squares than I have, but this should be enough to demonstrate the principle. Next, draw a line from (0, -5) to (1, 0) as I have.



In each image, I have used blue to indicate the newest line. It is not something you need to do in your drawing. Next line goes from (0, -4) to (2, 0).



Next line: (0, -3) to (3, 0):



Then (0, -2) to (4, 0):



Draw a final line from (0, -1) to (5, 0):



Now you have the principle behind the pattern in the exercise. To get the horizontal and vertical lines you could do (0, -6) to (0, 0) and (0, 0) to (6, 0). To sum it all up:

(0, -6) to (0, 0)
(0, -5) to (1, 0)
(0, -4) to (2, 0)
(0, -3) to (3, 0)
(0, -2) to (4, 0)
(0, -1) to (5, 0)
(0, 0) to (6, 0)

**Translating to python**

We obviously have two variables here. One is the y-coordinate of the first point in each line that varies from -6 to 0. The other one is the x-coordinate of the second point in each pair that goes from 0 to 6. The following Python snippet can achieve what we want.

1 **for** x, y **in** zip(range(-6, 0+1), range(0, 6+1)):
2     **print** ("(0, %d) to (%d, 0)" % (x, y))
3

(Note that in python, range(a, b) will include all numbers from a to b-1. That is why I wrote 0+1 and 6+1. You could have written 1 and 7 respectively, but I just wanted to add clarity)

If we think about it, the two variables vary together. So we could select one of them, say x, and express y in terms of x. Since x goes from 0 to 6 at the same time as y goes from -6 to 0, we could say that for each pair of points:

y = x - 6

So we could, if we so choose, write the python code as follows instead:

1 **for** x **in** range(0, 6+1):
2     **print** ("(0, %d) to (%d, 0)" % (x-6, x))
3

Got it so far? Good. 

**Preparing to put this on the screen**

First of all, we need to make a slight modification to what we have said so far. On the screen we need some space between each point. So instead of the x-coordinates being 0, 1, 2, 3, 4, 5 and 6, we would want them to be something like 0, 10, 20, 30, 40, 50 and 60. We will of course draw more lines as well.

But there is a more serious issue. I have been using Cartesian coordinates up to now. But screen coordinates are different. The top left hand corner of your screen is the point (0,0). As you move down the screen, y *increases*. So the lines we need to draw thus look as follows:

(0, 6) to (0, 0)
(0, 5) to (1, 0)
(0, 4) to (2, 0)
(0, 3) to (3, 0)
(0, 2) to (4, 0)
(0, 1) to (5, 0)
(0, 0) to (6, 0)

This means that the relationship between x and y is:

y = 6 - x

Make sure you understand this before you go on. Take as much time as you need to (hurry is a bad word in my vocabulary).

**Now on to pygame**

Time to use what we have just learned to put together a little program. The following program will solve the first part of the exercise.

 1 import pygame
 2
 3 screen = pygame.display.set\_mode((500, 500))
 4 clock = pygame.time.Clock()
 5 running = True
 6 size = 250
 7 step = 10
 8
 9 **for** x **in** range(0, size+1, step):
10     pygame.draw.line(screen, (255, 255, 255), (0, 250-x), (x, 0))
11
12 pygame.display.flip()
13
14 **while** running:
15     **for** event **in** pygame.event.get():
16         **if** event.type == pygame.QUIT:
17             running = False
18
19     clock.tick()
20

**Drawing in all corners**

The second part of the exercise was to draw this pattern in all four corners of the screen. I suggest you go back to use the pen and paper and draw each figure so you get a good idea of how the coordinates vary in each case.

Next you need to remember to convert all the lines into screen coordinates. You will need to remember that “nets” on the right side need their x-coordinates to be in relation to the right-most screen coordinate, which is the width of the screen – 1 (if the width of the screen is 501 pixels, the coordinates go from 0 to 500).

The same applies to the “nets” at the bottom of the screen. The y-coordinates need to be expressed in relation to the height of the screen instead.

I am going to let you work out the math for yourself. It is not too difficult if you take your time and solve it one step at a time. Just follow the principle I have outlined already. I will give you the solution in code. If you are having difficulties, you can look at the code and “go backwards” to get to the math.

 1 import pygame
 2
 3 w = h = 500
 4 screen = pygame.display.set\_mode((w+1, h+1))
 5 clock = pygame.time.Clock()
 6 running = True
 7 size = 250
 8 step = 10
 9 color = 255, 255, 255
10
11 **for** x **in** range(0, size+1, step):
12     pygame.draw.line(screen, color, (0, size-x), (x, 0))
13     pygame.draw.line(screen, color, (w - (size-x), 0), (w, x))
14     pygame.draw.line(screen, color, (w, h - (size-x)), (w-x, h))
15     pygame.draw.line(screen, color, (250-x, h), (0, h-x))
16
17 pygame.display.flip()
18
19 **while** running:
20     **for** event **in** pygame.event.get():
21         **if** event.type == pygame.QUIT:
22             running = False
23
24     clock.tick()
25

Note that the screen width is actually w+1 and the height is h+1. This is acceptable in my code because I make the rules.  On a serious note, it makes the draw\_line statements a little simpler.

You can (and should) experiment with size, step and color. Don’t worry if something ends up looking differently than you expected. If that happens, just take the time to understand why the program produces the results it does. This can be done with pen paper and a few debug statements (you could get the program to print out the coordinates of each line).

**Animating it**

Okay, let’s do something that is a little more interesting.

 3 import pygame
 4
 5 w = h = 500
 6 size = 250
 7 step = 10
 8 lines = []
 9 pos = 0
10 maxlines = 40
11
12 **for** x **in** range(0, size+1, step):
13     lines.append((0, size-x, x, 0))
14
15 **for** x **in** range(0, size+1, step):
16     lines.append((w - (size-x), 0, w, x))
17
18 **for** x **in** range(0, size+1, step):
19     lines.append((w, h - (size-x), w-x, h))
20
21 **for** x **in** range(0, size+1, step):
22     lines.append((size-x, h, 0, h-x))
23
24 screen = pygame.display.set\_mode((w+1, h+1))
25 clock = pygame.time.Clock()
26 running = True
27
28 **while** running:
29     **for** event **in** pygame.event.get():
30         **if** event.type == pygame.QUIT:
31             running = False
32
33     screen.fill((0, 0, 0))
34     col = 0
35     cur = pos
36
37     **for** i **in** range(maxlines):
38         x1, y1, x2, y2 = lines[cur]
39         pygame.draw.line(screen, (col, col, col), (x1, y1), (x2, y2))
40
41         cur += 1
42         **if** cur >= len(lines): cur = 0
43         col += 240 / maxlines
44
45     pos += 1
46     **if** pos >= len(lines): pos = 0
47
48     pygame.display.flip()
49     clock.tick(40)

I am not going to explain how it works here. Rather the following exercises are all related to this code. Solve the exercises as a guide to help you build up an understanding of how the program works.

Remember that as a programmer, one of the skills you will need to have is the ability to read code and make sense of it. That is the purpose of this. Just take it all one step at a time and you will be okay (I promise). If you are not sure of what a particular line does, comment it out or modify it in some way and see how it affects the result. That is an excellent and fun way to learn more.

Once you have been able to understand how the program works, see how you can improve upon it. There are several things you could do. You could look at optimizing it, making it more readable, cut out unnecessary variables, etc. Feel free to post your code.

**Exercises**

1. What date does the lines array hold?

2. What is the purpose of the pos variable?

3. What is the purpose of the maxlines variable? What happens when you change maxlines to 10? Or to 60? Try with different values for maxlines and see how it affects the program.

4. Why is it that I use four for loops to populate the lines array? What would happen if you used a single for loop containing all four appends? Is there any way you could re-write this part to use a single for loop without the lines ending up in the wrong order?

5. Why have I created the cur variable. It seems to always be initialized to the value of pos. Would it be possible to do away with cur and only use pos?

6. What on earth goes on in the loop that starts with for i in range(maxlines)?

7. What is it that causes the lines to “fade”?

8. Play around with the expression that changes the color value (col += 240 / maxlines)

9. Modify the program to draw red, green, blue or yellow lines. Better yet, make it use any color you prefer. Make sure that the lines still fade properly.

10. What is the purpose of this code?

pos += 1

if pos >= len(lines): pos = 0

11. Change the “tick” value. How does it affect the speed of the animation when you increase or decrease the value?

12. Can you reverse the direction of the animation?

**Conclusion**

Okay, that’s it for this time. Hope you enjoyed it. Don’t forget that curiosity is a good thing. Try things out. Modify the code. Any time you are uncertain about what something does, play around with it. You won’t damage your computer or your OS by messing around with the code.

Also, as the beginning of this tutorial was meant to show, there are times when pen and paper are a programmer’s best friend (sorry, Matz).

Most importantly, don’t give up! If I’m able to work things like this out, so can you.