## [Pygame tutorial #1: getting started](http://lorenzod8n.wordpress.com/2007/05/25/pygame-tutorial-1-getting-started/%22%20%5Co%20%22Permalink%20to%20Pygame%20tutorial%20#1: getting started)

This is the first part of my [pygame](http://pygame.org/) tutorial, aimed at beginners. I will keep it small and simple, meaning I will proceed *very* slowly. I am not going to go into too much detail. I will explain just about as much as is necessary to get you started. You will need to have the pygame documentation available. Always keep it open while you are going through this tutorial.

One little thing before we begin. I am chronically lazy. It is amazing that I even manage to get out of bed every day. So please don’t expect this to be a regular thing. I will write tutorials as and when my eyes are open.

**What you will need**

* Python: I am using Python 2.4 which is the default Python version on Debian Etch. Other versions will work as well.
* Pygame: There really isn’t much point in programming in pygame unless you have it installed.
* A text editor: I recommend [Vim](http://www.vim.org/) a powerful editor with an unfortunately bad reputation for being complex. Don’t believe the rumors. They are not true.

If you are on Debian or one of its derivates, try this (as root):

# aptitude install python-pygame

That will install pygame and all dependencies for you. Other distros and operating systems: both Python and pygame are most likely available in your repositories. Figure out how to install them and you are ready to go.

**What you should know already**

It is a good idea if you at least have some basic knowledge of [Python](http://python.org/). I guess you could be learning it as you go along, but I will assume that you know how to program in Python already.

**Getting started**

First of all, let us look at creating a pygame application that does absolutely nothing. Well, nearly absolutely nothing. It actually does display a window. It also does some very rudimentary event handling. You can see this as a template for the next few programs that we will write.

Here is the code:

1 import pygame

2

3 screen = pygame.display.set\_mode ((640, 400))

4

5 **while** 1:

6 **pass**

First of all, to use pygame we have to import it. Then we create a Surface that is 640 pixels wide and 400 pixels high. There are a lot more things you can do with set\_mode, but we will keep it simple at this point.

Next we enter into an infinite loop. We need this because otherwise the window will
If you run this program it should display a 640×400 window. That wasn’t too difficult was it. If you try to close this application by clicking on the window’s close button, you will notice that it does nothing. You will have to go activate the terminal that you started the program from and hit CTRL+C to stop the program. We will fix that soon.

**Exercises**

1. Create a window that is 320 pixels wide and 200 pixels high.
2. Create a program where the user can specify the width and the height as command line arguments.
3. Create a program that asks the users for the width and the height and then displays the window.
4. Write a program that calls pygame.display.set\_mode twice with different sizes. What do you expect should happen? Run the program. What actually happens? Can you explain why?

**Adding an event loop**

Our first example was maybe a little too simple. Staring at a completely blank window soon gets boring. Also, having to go to the terminal and hit CTRL+C to close the window seems a little awkward. Let’s add a bit of code!

Here is an updated version of the first program:

 1 #! /usr/bin/env python

 2

 3 import pygame

 4

 5 screen = pygame.display.set\_mode((640, 400))

 6 running = 1

 7

 8 **while** running:

 9 event = pygame.event.poll()

10 **if** event.type == pygame.QUIT:

11 running = 0

12

What is new is that I have added a simple event loop. The loop is controlled by a flag called running. As long as the flag is set the loop keeps running. Inside the loop we use the pollmethod to grab the next event from the event queue. There are other ways of doing this, but polling works just fine for now.

There are several different event types that pygame knows about and can respond to. One of these is QUIT, which gets triggered when the user clicks on the window’s close button. All we do if we get an event of this type is clear the running flag, which will exit the loop and cause the program to terminate. Still simple, isn’t it.

**Exercises**

1. Adapt each of the programs you wrote for the exercises in the previous section to use an event loop.
2. Rewrite the program to do away with the running flag. Make sure that the program still jumps out of the event loop on the QUIT event.

**Finishing touches to the template**

As a final step before we start doing real things, let’s add just a little bit more so that we have a complete template for what follows. We will paint the our Surface and we will learn how to do a bit of screen flipping. First the code:

 1 #! /usr/bin/env python

 2

 3 import pygame

 4

 5 screen = pygame.display.set\_mode((640, 400))

 6 running = 1

 7

 8 **while** running:

 9 event = pygame.event.poll()

10 **if** event.type == pygame.QUIT:

11 running = 0

12 screen.fill((0, 0, 0))

13 pygame.display.flip()

We have added just two lines of code. The first one sets a background color for the Surface. We have passed in a sequence of three values: red, green and blue. Each one can be a value between 0 and 255. Since we set all to zero, we get a black screen. You should experiment with different values for red, green and blue.

The next thing that is new is that we call pygame.display.flip. Drawing directly to the screen is usually a very bad idea. Instead, we have a invisible buffer that we draw onto. When we have finished drawing we make the buffer visible. That way we get flicker-free animations (when we get to that).

***Exercises***

1. Create a window with a white background color.
2. Create a window with a red background color.
3. Experiment with setting different background colors. If you are not familiar with RGB values then spend a little extra time to figure out how to get colors like yellow, brown, cyan etc.
4. Create a program that asks the user to specify the values for red, green and blue. Check that the values are in the valid range (0-255) and then use these for the background color.
5. Create a program that upon start-up checks the time of the day and sets the brightness of the background color accordingly. Use a blue scale. If it is midnight, the screen should be black. If it is midday, the screen should be bright blue. Any other time the background should be something in between corresponding to the brightness outside.

**Conclusion**

So what have you learned so far? Not an awful lot, by the looks of it. But in a way you have. You have acquired some basics that you will need to understand the next tutorial, which will come soon. In the meantime, try your hands on a few of the exercises. If you complete all of them, you can make up a few of your own.