Programming in Python: Graphics

Graphics, pygame basics

Up until this point everything we have done has used a fairly crude text interface. There are a number of ways to do graphics in Python. Each different way amounts to importing a module and learning/using the functions, objects and methods of that module. Each of the modules have their strengths and weaknesses. The only graphics module that comes with the standard installation of Python is called Tkinter. It does a decent job of setting up GUIs, but many people are using wxPython to do GUIs these days. We might look at some GUI programming later in the course, but the main focus of our graphics will be using pyGame. pyGame, like most Python add-ons, is a free download, with no strings attached.

Confirm pygame is properly installed on your machine
Whenever I am using add-ons, I like to be sure it is properly installed.
In the shell, type the following:

```
>>> import pygame
>>> help(pygame)
```

The help for pygame will now scroll very quickly. This can always be used as a reference.

As you can see the pygame module is VERY involved. We are not going to try to understand the whole thing before we program with it. What we are going to do is map out a small subset of commands that we understand and use them to get the results we want.

You can also get help at http://www.pygame.org/docs/.

WARNING : Messing up with pyGame can crash your machine. When it says "do you want to exit altogether" always say yes. If you appear to have frozen your machine go to your python shell and type: `pygame.quit()`

Colour

Colours in pyGame are represented by tuples that state the amount of Red, Green, and Blue are in them. Each value is an integer from 0 to 255 indicating how much of each colour goes into the mix.
e.g.
(0, 0, 255) - This means 0% Red 0% Green and 100% Blue. The result is, of course, basic blue.
0, 0, 255 - This is another way of defining a tuple

To explore different colour combinations, do a search for RGB Color Codes Chart.
Note: RGB stands for Red Green Blue

Copy the following program into Wing IDE and examine it carefully.

```
# pygame1.py
# the basic template to draw a picture that is not moving

import pygame
pygame.init()   # ALL pygame programs need to initialize the pygame engine
                # before they can use it

                  # Defining a colour constant(variables we should not be changing)
RED = (255, 0, 0) # we use ALL CAPS for constants so we remember not to change them

SIZE = (800, 600)                 # Open a pygame window
screen = pygame.display.set_mode(SIZE)

pygame.draw.line(screen, RED, (0,0), (100,50))  # draw our "scene"

pygame.display.flip()   # pygame is designed for fast moving graphics if you
                        # don't use page flipping you can get a flickery mess
                        # basically Everything we draw is drawn in memory
                        # when we call flip() that picture gets copied to the
                        # screen

pygame.time.wait(3000)  # pause for three seconds so we can see it
pygame.quit()           # pygame likes to crash computers when you don't
                        # quit()
```

Try this second program:

```
# the following code will always put the screen in the top corner
import os
os.environ['SDL_VIDEO_WINDOW_POS'] = "%d, %d" %(20, 20)

# pygame2a.py
# draw a face using a few of the pygame draw methods

import pygame
pygame.init()
RED = (255, 0, 0)
BLACK = (0,0,0)
SIZE = (800, 600)
screen = pygame.display.set_mode(SIZE)

# Draws a red face
pygame.draw.circle(screen, RED, (200,100), 50)
pygame.draw.circle(screen, BLACK, (185,70), 5)        # Left eye
pygame.draw.circle(screen, BLACK, (215,70), 5)        # Right eye
pygame.draw.line(screen, BLACK, (185,85), (215, 85))  # Mouth

pygame.display.flip()
pygame.time.wait(3000)
pygame.quit()
```

We will be concentrating (at least at the start) on mainly the draw function.

The main functions are:

Rect Draws a rectangular shape on the screen

Form: pygame.draw.rect(screen, colour, vertices*, width*)

Variables: screen - the created screen where we are drawing the rectangle
colour - RGB colour, can be predefined like RED in the example
or defined in the statement as (red, green, blue) e.g. (100, 200, 155)
vertices - (x1, y1, w, h) where x1, y1 are the coordinates of the top left
and w and h are the dimensions of the rectangle
width - optional, does not have to be there, will fill in the rectangle if it is not.
If it is, then width represents the thickness of the outside of the rectangle.

Examples: pygame.draw.rect(screen, RED, (120,330, 200, 500))
pygame.draw.rect(screen, RED, (0,0, 200, 500), 2)

Ellipse Draws an ellipse(oval) on the screen.

Form: pygame.draw.ellipse(screen, colour, vertices*, width*)

Variables: screen - the created screen where we are drawing the ellipse
colour - RGB colour, can be predefined like RED in the example
or defined in the statement as (red, green, blue) e.g. (100, 200, 155)
vertices - (x1, y1, w, h) where x1, y1 are the coordinates of the top left
and w, h are the width and height of the rectangle in which the
ellipse would fit within.
width - optional, does not have to be there, will fill in the ellipse if it is not.
If it is, then width represents the thickness of the outside of the ellipse.

Examples: pygame.draw.ellipse(screen, (100, 200, 155), (120,330, 200, 500))
pygame.draw.ellipse(screen, RED, (0,0, 200, 500), 2)

Arc Draws an arc on the screen

Form: pygame.draw.arc(screen, colour, vertices, startAngle, endAngle*, width*)

Variables: screen - the created screen where we are drawing the arc
colour - RGB colour, can be predefined like RED in the example
or defined in the statement as (red, green, blue) e.g. (100, 200, 155)
vertices - (x1, y1, w, h) where x1, y1 are the coordinates of the top left
and w,h are the height and width of the rectangle in which the
arc would fit within.
startAngle - the angle (in radians) at which the arc would start (on the right is 0)
endAngle - the angle (in radians) at which the arc ends
width - optional, thickness to draw the outer edge

Examples:
pygame.draw.arc(screen,(100, 200, 155), (50, 50, 200, 200), 0, math.pi)
pygame.draw.arc(screen,RED, (0, 0, 300, 250), math.pi / 2 , 3*math.pi/2)
pygame.draw.arc(screen, BLUE, (100, 100, 200, 200), math.radians(30), math.radians(150))

Note: Radian measure means angles written in terms of pi.
0 pi = 0 degrees
1 pi = 180 degrees
pi/2 = 90 degrees
3*pi/2 = 270 degrees
2*pi = 360 degrees
NoteNote: To use math.pi, add import math at the top of your program
Use math.radians to convert from degrees to radians.

Circle

Draws a circle on the screen

Form: pygame.draw.circle(screen, colour, pos, radius*, width*)

Variables: screen - the created screen where we are drawing the circle
colour - RGB colour, can be predefined like RED in the example
or defined in the statement as (red, green, blue) e.g. (100, 200, 155)
pos- (x1, y1) where x1, y1 are the coordinates of the center of the circle
radius - the radius of the circle
width - optional, does not have to be there, will fill in the circle if it is not.
If it is, then width represents the thickness of the outside of the circle.
Examples: pygame.draw.circle(screen, RED, (0, 0), 100)
pygame.draw.circle(screen, RED, (200,100), 50, 10)

Line

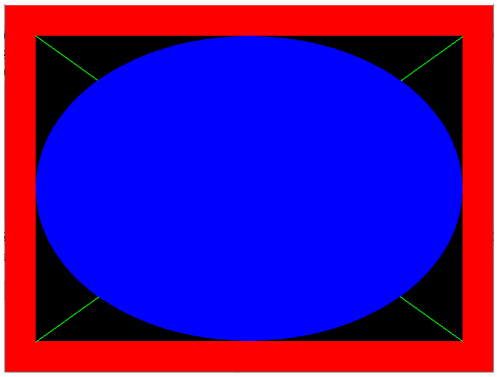
Draws a line on the screen

Form: pygame.draw.line(screen, colour, start, end*, width*)

Variables: screen - the created screen where we are drawing the line
colour - RGB colour, can be predefined like RED in the example
or defined in the statement as (red, green, blue) e.g. (100, 200, 155)
start - (x1, y1), where x1, y1 are the coordinates of the beginning of the line
end - (x2, y2), where x2, y2 are the coordinates of the end of the line
width - optional, does not have to be there
If it is, then width represents the thickness of the line.
Examples: pygame.draw.line(screen, (100, 200, 155), (0, 0), (100, 200))
pygame.draw.line(screen, RED, (100, 200), (50, 350), 10)

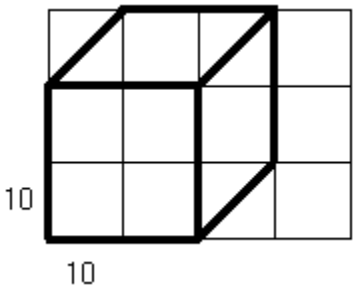
Graphics Exercises

1. Create the following picture with the following:



2. Write a program that simulates a game of tic-tac-toe that the user can watch the computer play.

3. Write a program that draws a 3D box after the user inputs the location and the dimensions are 100 by 100 by 100.

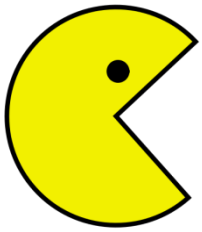


4. Draw a character (a person, a superhero, a car) on the screen and be colourful and creative with your drawing. Feel free to experiment and to have fun with this! Can you get the character to move?

5. Set the graphics screen to be 500 by 500. Make the background blue, create a filled in rectangle (choose your colour) with dimensions 100 by 100 that is in the exact center of the graphics screen.

6. Repeat Question 5 but make the graphics screen 300 by 300. Did you have to modify your code to draw the rectangle again? What about your background? What if we change the size again of the screen? Modify your code to work for any graphics screen. Can you make the rectangle always have dimensions $1/5^{th}$ the height of the screen size?

7. Draw a picture of pacman centered on your screen.



8. Use delay to make pacman start on the left side of the screen and move across to the right side at a reasonable pace.

9. Some other things to try:

- Pacman opens and closes its mouth as it moves
- use keyboard commands to move it.