PYTHON NOTES
(drawing.py and drawstuff.py)

INTRODUCTION TO PROGRAMMING USING PYGAME

STEP 1: Importing Modules and Initialization

All the Pygame functions that are required to implement features like graphics and sound are stored in the Pygame module and sys module. These modules must be imported into our namespace for us to access the functions inside these modules.

To do this, we use the statement:

```
import pygame, sys
```

Note: When we import a module, we gain access to the sub-modules too.

Note: There are two formats to import modules:

1. import modulename
2. from modulename import *

Normally, when we use import modulename, we can refer to the functions inside by calling modulename.functionname.

However, using from modulename import *, we can refer to the function simply by its name without the modulename prefix.

The module pygame.locals contains many constant variables. So, to reduce the typing workload, we import pygame.locals in the second format.

```
from pygame.locals import *
```

```
pygame.init()
```

It is a function call, that is performed before any other Pygame function call. It needs to be called first in order for many other Pygame functions to work.

STEP 2: Setting the Window Size

```
DISPLAYSURF = pygame.display.set_mode((700, 500))
```

The set_mode() function accepts a tuple of two integers as its argument and returns a pygame.Surface object for the display window. We pass a tuple value of two integers to the function: (700, 500). This tuple tells the set_mode() function how wide and how high to make the window in pixels. (700, 500) will make a window with a width of 700 pixels and height of 500 pixels.

Note: Make sure to pass a sequence, i.e., a tuple of two integer values to the set_mode() function, not two integers.

STEP 3: Adding a Window Caption

pygame.display.set_caption('Summer Camp on fleek')

This sets the caption text that will appear at the top of the window by calling the pygame.display.set_caption() function. The string value 'Summer Camp on fleek' is passed in this function call to make that text appear as the caption.

STEP 4: Game Loops and Events

while True: # main game loop

for event in pygame.event.get():

This is a while loop that has a condition of simply the value True. This means that it never exits and the only way the program execution will ever exit the loop is if a break statement is executed or sys.exit() (which terminates the program). If a loop like this was inside a function, a return statement will also move execution out of the loop (as well as the function too).

A **game loop** (also called a **main loop**) is a loop where the code does three things:

1. Handles events.
2. Updates the game state.
3. Draws the game state to the screen.

A game state is just a way of referring to a set of values for all the variables in a game program. Since a game is affected by passage of time or outside events like mouse click or a key press, the program needs to be constantly checked for events and these evets are checked for using the pygame.event.get() function.

STEP 5: Program Termination

The QUIT Event and pygame.quit() Function

```
if event.type == QUIT:
    pygame.quit()
sys.exit()
```

This statement checks if the event type is QUIT. If yes, then it calls the pygame.quit() function and sys.exit() to terminate the program.

Some Useful Pygame Functions

1. pygame.draw

Pygame module to draw shapes: (pygame.draw)

Function name	Functionality
pygame.draw.line	Draws a straight line segment
pygame.draw.circle	Draws a circle around a point
pygame.draw.rect	Draws a rectangle shape
pygame.draw.ellipse	Draws a round shape inside a rectangle
pygame.draw.polygon	Draws a shape with any number of sides
pygame.draw.arc	Draws a partial section of an ellipse
pygame.draw.lines	Draws multiple line segments

2. pygame.Color

Define the colors used in RGB format as follows:

COLOR	R	G	B
Red	255	0	0
Green	0	255	0
Blue	0	0	255
Black	0	0	0
White	255	255	255

3. pygame.time

Function name	Functionality
clock	To track the time used by a frame
delay	Pauses for a given time (returns the number of milliseconds)
get_ticks	Time since pygame.time was imported (in ms)
set_timer	Sets the timer for an event (the event then gets placed on an event queue)
wait	Pauses for a given time (like delay but less accurate)

Example Program: drawstuff.py

In this example program, we have the following set up in the file first:

```
# Arup Guha
# 7/12/2015
# Drawing using pygame

import pygame, sys
from pygame.locals import *

pygame.init()
DISPLAYSURF = pygame.display.set_mode((1000, 600))
pygame.display.set_caption("Drawing!")

black = pygame.Color(0,0,0)
red = pygame.Color(255,0,0)
green = pygame.Color(0,255,0)

pts = [(100,200), (200,200), (300,150), (150,100), (25, 175)]
```

Many pyGame programs utilize constant values. Though there are no constants in Python, to be consistent with other programming languages, many people define variables to store constant values. Our intention is to never change the values stored in the variables black, red, green and pts, even though, the compiler would not get mad at us if we did. When we define these constants at the top of the program, we are free to use these names later to refer to the particular objects. There are many reasons to use constants in programs, but one big reason is so the code is easier to read. We haven't seen lists yet, but a list is just a sequence of items. The list pts is a sequence of points, each of which is an ordered pair (technically a tuple).) We will use these points in the program to draw a polygon defined by the points.

This is followed by the basic game loop, which you can just copy for any program where you are drawing a static canvas:

```
while True:

    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

    # Draw on canvas here
```

As previously described, the game loop runs the whole time. The only way to get out of it is if a quit event is generated (clicking away the screen in the top right corner). In short, the while loop runs many, many times, each time, the for loop inside of it goes through each even that recently occurred. If any of these are quitting events, then pygame.quit() and sys.exit() are executed, which stop the program from continuing to run. Most times, there won't be a quit event. In this case, we follow the for loop with our drawing. Thus, what's really happening in a program of this kind is that we're drawing the same static picture over and over again, very fast, so it just looks like a painting.

Now, to analyze each of the items drawn.

Our first line of code (part of #Draw on canvas here) is:

```
    DISPLAYSURF.fill(black)
```

This just initially fills the color of the whole display surface to be black.

This is followed by:

```
    pygame.draw.ellipse(DISPLAYSURF, green, (500, 300, 50, 150), 10)
```

The ellipse function's first parameter is the displaysurface which we have called DISPLAYSURF. The second parameter of this method is the color of the ellipse. This is followed by a 4-tuple that describes the ellipse. The order of the items in the 4-tuple are the x-coordinate and y-coodinate of the center of the ellipse (in pixels) followed by the length of the horizontal axis of the ellipse and the last parameter is the length of the vertical axis of the ellipse. So, this ellipse will be tall and narrow. Finally, the last parameter of the function (for this function call, 10), represents the thickness of the ellipse.

The next two lines of code are:

```
    pygame.draw.line(DISPLAYSURF, red, (500, 0), (500, 550))
    pygame.draw.line(DISPLAYSURF,   pygame.Color(255,255,255),   (0,   250),
(950, 250))
```

The line function takes in the display surface to write on as its first parameter, followed by the color to draw the line, followed by two 2-tuples, representing the end points of the corresponding segment. This example simply shows two ways of specifying a color. One uses a constant previously set up, the other hard codes in the color directly using the pyGame.Color function.

This is followed by the following function call that draws a rectangle. As we have in all of the other similar functions, the first two parameters are the display surface and color. The third parameter is a 4-tuple, which indicates the top left corner and bottom right corner, respectively, of the rectangle. This is followed by the width of the rectangle.

```
    pygame.draw.rect(DISPLAYSURF, pygame.Color(95, 12, 183), (50, 200, 700,
50), 10)
```

To draw an arbitrary polygon, the third parameter is simply a list of 2-tuples, where each 2-tuple represents a point on the polygon in the order of tracing the polygon. The last point on this list is connected to the first with a line segment:

```
    pygame.draw.polygon(DISPLAYSURF, red, pts)
```

Finally, for any of this to show up, we must update the display as follows:

```
    pygame.display.update()
```