InTrOdUcTiOn To GRAPHICS

Graphics make programming more fun for many people. To fully introduce graphics would involve many ideas that would be a distraction now. This section introduces a simplified graphics module developed by John Zelle for use with his Python Programming book. My slight elaboration of his package is `graphics.py` in the example programs.

Graphics in python by pygame

Note You will just be a user of the graphics.py code, so you do not need to understand the inner workings! It uses all sorts of features of Python that are way beyond these tutorials. There is no particular need to open graphics.py in the Idle editor.

Load into Idle and start running example graphIntroSteps.py, or start running from the operating system folder. Each time you press return, look at the screen and read the explanation for the next line(s).

Press return:

```
from graphics import *
win = GraphWin()
```

Zelle's graphics are not a part of the standard Python distribution. For the Python interpreter to find Zelle's module, it must be imported. The first line above makes all the types of object of Zelle's module accessible, as if they were already defined like built-in types str or list.

Look around on your screen, and possibly underneath other windows: There should be a new window labeled "Graphics Window", created by the second line. Bring it to the top, and preferably drag it around to make it visible beside your Shell window. A GraphWin is a type of object from Zelle's graphics package that automatically displays a window when it is created. The assignment statement remembers the window object as win for future reference. (This will be our standard name for our graphics window object.) A small window, 200 by 200 pixels is created. A pixel is the smallest little square that can by displayed on your screen. Modern screen usually have more than 1000 pixels across the whole screen.

Press return:

```
pt = Point(100, 50)
```

This creates a Point object and assigns it the name pt. Unlike when a GraphWin is created, nothing is immediately displayed: In theory you could have more than one GraphWin. Zelle designed the graphics module so you must tell Python into which GraphWin to draw the Point. A Point object, like each of the graphical objects that can be drawn on a GraphWin, has a method [1] draw.

Press return:

pt.draw(win)

Now you should see the Point if you look hard in the Graphics Window - it shows as a single, small, black pixel. Graphics windows have a Cartesian (x,y) coordinate system. The dimensions are initially measured in pixels. The first coordinate is the horizontal coordinate, measured from left to right, so 100 is about half way across the 200 pixel wide window. The second coordinate, for the vertical direction, increases going down from the top of the window by default, not up as you are likely to expect from geometry or algebra class. The coordinate 50 out of the total 200 vertically should be about 1/4 of the way down from the top. We will see later that we can reorient the coordinate system to fit our taste.

Henceforth you will see a draw method call after each object is created, so there is something to see.

Press return:

cir = Circle(pt, 25)

cir.draw(win)

The first line creates a Circle object with center at the previously defined pt and with radius 25. This object is remembered with the name cir. As with all graphics objects that may be drawn within a GraphWin, it is only made visible by explicitly using its draw method.

So far, everything has been drawn in the default color black. Graphics objects like a Circle have methods to change their colors. Basic color name strings are recognized. You can choose the color for the circle outline as well as filling in the inside.

Press return:

cir.setOutline('red')

cir.setFill('blue')

Note the method names. They can be used with other kinds of Graphics objects, too. (We delay a discussion of fancier colors until Color Names and Custom Colors.)

Press return:

line = Line(pt, Point(150, 100))

line.draw(win)

A Line object is constructed with two Points as parameters. In this case we use the previously named Point, pt, and specify another Point directly. Technically the Line object is a segment between the the two points.

Warning In Python (150, 100) is a tuple, not a Point. To make a Point, you must use the full constructor: Point(150, 100). Points, not tuples, must be used in the constructors for all graphics objects.

A rectangle is also specified by two points. The points must be diagonally opposite corners.

Press return:

rect = Rectangle(Point(20, 10), pt)

rect.draw(win)

In this simple system, a Rectangle is restricted to have horizontal and vertical sides. A Polygon, introduced in the next section, is used for all more general straight-sided shapes.

You can move objects around in a GraphWin. Shortly this will be handy for animation. The parameters to the move method are the amount to shift the x and y coordinates. See if you can guess the result before you press return:

line.move(10, 40)

Did you remember that the y coordinate increases down the screen?

Take your last look at the Graphics Window, and make sure that all the steps make sense. Then destroy the window win with the GraphWin method close.

Press return:

win.close()

The example program graphIntro.py starts with the same graphics code as graphIntoSteps.py, but without the need for pressing returns.

An addition I have made to Zelle's package is the ability to print a string value of graphics objects for debugging purposes. If some graphics object isn't visible because it is underneath something else of off the screen, temporarily adding this sort of output might be a good reality check.

At the end of graphIntro.py, I added print lines to illustrate the debugging possibilites:

print('cir:', cir)

print('line:', line)

print('rect:', rect)

You can load graphIntro.py into Idle, run it, and add further lines to experiment if you like. Of course you will not see their effect until you run the whole program!

PROGRAMME

```
'''A simple graphics example constructs a face from basic shapes.
'''

from graphics import *


def main():
    win = GraphWin('Face', 200, 150) # give title and dimensions
    win.yUp() # make right side up coordinates!

    head = Circle(Point(40,100), 25) # set center and radius
    head.setFill("yellow")
    head.draw(win)

    eye1 = Circle(Point(30, 105), 5)
    eye1.setFill('blue')
    eye1.draw(win)
```

```
    eye2 = Line(Point(45, 105), Point(55, 105)) # set endpoints
    eye2.setWidth(3)
    eye2.draw(win)

    mouth = Oval(Point(30, 90), Point(50, 85)) # set corners of bounding box
    mouth.setFill("red")
    mouth.draw(win)

    label = Text(Point(100, 120), 'A face')
    label.draw(win)

    message = Text(Point(win.getWidth()/2, 20), 'Click anywhere to quit.')
    message.draw(win)
    win.getMouse()
    win.close()

main()
```

GraphWin method yUp (y increases upward)

When you first create a GraphWin, the y coordinates increase down the screen. To reverse to the normal orientation use my GraphWin yUp method.

```
win = Graphwin('Right side up', 300, 400)
win.yUp()
```

GraphWin method promptClose (Prompt and Close Graphics Window)

You generally want to continue displaying your graphics window until the user chooses to have it closed. The GraphWin promptClose method posts a prompt, waits for a mouse click, and closes the GraphWin. There are two ways to call it, depending on whether you want to use an existing Text object, or just specify a location for the center of the prompt.

```
win.promptClose(win.getWidth()/2, 30) # specify x, y coordinates of prompt
```

or

```
msg = Text(Point(100, 50), 'Original message...')
msg.draw(win)
# ...
# ... just important that there is a drawn Text object
win.promptClose(msg)  # use existing Text object
```

String Representations of all Graphics Object Types

Each graphical type can be converted to a string or printed, and a descriptive string is produced (for debugging purposes). It only shows position, not other parts of the state of the object.

```
>>> pt = Point(30, 50)
>>> print(pt)
Point(30, 50)
>>> ln = Line(pt, Point(100, 150))
>>> print(ln)
Line(Point(30, 50), Point(100, 150))
```