Chapter 3
Python Turtle Graphics

3.1 Introduction

Python is very well supported in terms of graphics libraries. One of the most widely used graphics libraries is the Turtle Graphics library introduced in this chapter. This is partly because it is straight forward to use and partly because it is provided by default with the Python environment (and this you do not need to install any additional libraries to use it).

The chapter concludes by briefly considering a number of other graphic libraries including PyOpen GL. The PyOpenGL library can be used to create sophisticated 3D scenes.

3.2 The Turtle Graphics Library

3.2.1 The Turtle Module

This provides a library of features that allow what are known as vector graphics to be created. Vector graphics refers to the lines (or vectors) that can be drawn on the screen. The drawing area is often referred to as a drawing plane or drawing board and has the idea of x, y coordinates.

The Turtle Graphics library is intended just as a basic drawing tool; other libraries can be used for drawing two and three dimensional graphs (such as MatPlotLib) but those tend to focus on specific types of graphical displays.

The idea behind the Turtle module (and its name) derives from the Logo programming language from the 60s and 70s that was designed to introduce programming to children. It had an on screen turtle that could be controlled by commands such as forward (which would move the turtle forward), right (which would turn the turtle by a certain number of degrees), left (which turns the turtle left by a certain number of

J. Hunt, *Advanced Guide to Python 3 Programming*,
Undergraduate Topics in Computer Science,
https://doi.org/10.1007/978-3-030-25943-3_3

degrees) etc. This idea has continued into the current Python Turtle Graphics library where commands such as `turtle.forward(10)` moves the turtle (or cursor as it is now) forward 10 pixels etc. By combining together these apparently simple commands, it is possible to create intricate and quiet complex shapes.

3.2.2 Basic Turtle Graphics

Although the `turtle` module is built into Python 3 it is necessary to *import* the module before you use it:

```
import turtle
```

There are in fact two ways of working with the `turtle` module; one is to use the classes available with the library and the other is to use a simpler set of functions that hide the classes and objects. In this chapter we will focus on the set of functions you can use to create drawings with the Turtle Graphics library.

The first thing we will do is to set up the window we will use for our drawings; the `TurtleScreen` class is the parent of all screen implementations used for whatever operating system you are running on.

If you are using the functions provided by the `turtle` module, then the screen object is initialised as appropriate for your operating system. This means that you can just focus on the following functions to configure the layout/display such as this screen can have a title, a size, a starting location etc.

The key functions are:

- `setup(width, height, startx, starty)` Sets the size and position of the main window/screen. The parameters are:
  - `width`—if an integer, a size in pixels, if a float, a fraction of the screen; default is 50% of screen.
  - `height`—if an integer, the height in pixels, if a float, a fraction of the screen; default is 75% of screen.
  - `startx`—if positive, starting position in pixels from the left edge of the screen, if negative from the right edge, if None, center window horizontally.
  - `starty`—if positive, starting position in pixels from the top edge of the screen, if negative from the bottom edge, if None, center window vertically.
- `title(titlestring)` sets the title of the screen/window.
- `exitonclick()` shuts down the turtle graphics screen/window when the use clicks on the screen.
- `bye()` shuts down the turtle graphics screen/window.
- `done()` starts the main event loop; this must be the last statement in a turtle graphics program.

- `speed(speed)` the drawing speed to use, the default is 3. The higher the value the faster the drawing takes place, values in the range 0–10 are accepted.
- `turtle.tracer(n = None)` This can be used to batch updates to the turtle graphics screen. It is very useful when a drawing become large and complex. By setting the number (n) to a large number (say 600) then 600 elements will be drawn in memory before the actual screen is updated in one go; this can significantly speed up the generation of for example, a fractal picture. When called without arguments, returns the currently stored value of `n`.
- `turtle.update()` Perform an update of the turtle screen; this should be called at the end of a program when `tracer()` has been used as it will ensure that all elements have been drawn even if the tracer threshold has not yet been reached.
- `pencolor(color)` used to set the colour used to draw lines on the screen; the color can be specified in numerous ways including using named colours set as 'red', 'blue', 'green' or using the RGB colour codes or by specifying the color using hexadecimal numbers. For more information on the named colours and RGB colour codes to use see https://www.tcl.tk/man/tcl/TkCmd/colors.htm. Note all colour methods use American spellings for example this method is `pencolor` (not pencolour).
- `fillcolor(color)` used to set the colour to use to fill in closed areas within drawn lines. Again note the spelling of colour!

The following code snippet illustrates some of these functions:

```
import turtle

# set a title for your canvas window
turtle.title('My Turtle Animation')

# set up the screen size (in pixels)
# set the starting point of the turtle (0, 0)
turtle.setup(width=200, height=200, startx=0, starty=0)

# sets the pen color to red
turtle.pencolor('red')

# …

# Add this so that the window will close when clicked on
turtle.exitonclick()
```

We can now look at how to actually draw a shape onto the screen.

The cursor on the screen has several properties; these include the current drawing colour of the *pen* that the cursor moves, but also its current position (in the x, y coordinates of the screen) and the direction it is currently facing. We have already seen that you can control one of these properties using the `pencolor()` method, other methods are used to control the cursor (or turtle) and are presented below.

The direction in which the *cursor* is pointing can be altered using several functions including:

- `right(angle)` Turn cursor right by angle units.
- `left(angle)` Turn the cursor left by angle units.
- `setheading(to_angle)` Set the orientation of the cursor to `to_angle`. Where 0 is east, 90 is north, 180 is west and 270 is south.

You can move the cursor (and if the pen is down this will draw a line) using:

- `forward(distance)` move the cursor forward by the specified distance in the direction that the cursor is currently pointing. If the pen is down then draw a line.
- `backward(distance)` move the cursor backward by distance in the opposite direction that in which the cursor is pointing.

And you can also explicitly position the cursor:

- `goto(x, y)` move the cursor to the x, y location on the screen specified; if the pen is down draw a line. You can also use steps and set position to do the same thing.
- `setx(x)` sets the cursor's x coordinate, leaves the y coordinate unchanged.
- `sety(y)` sets the cursor's y coordinate, leaves the x coordinate unchanged.

It is also possible to move the cursor without drawing by modifying whether the pen is up or down:

- `penup()` move the pen up—moving the cursor will no longer draw a line.
- `pendown()` move the pen down—moving the cursor will now draw a line in the current pen colour.

The size of the pen can also be controlled:

- `pensize(width)` set the line thickness to width. The method `width()` is an alias for this method.

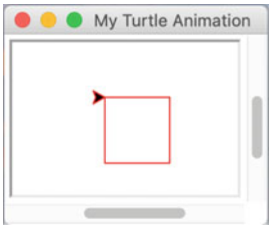
It is also possible to draw a circle or a dot:

- `circle(radius, extent, steps)` draws a circle using the given radius. The extent determines how much of the circle is drawn; if the extent is not given then the whole circle is drawn. Steps indicates the number of steps to be used to drawn the circle (it can be used to draw regular polygons).
- `dot(size, color)` draws a filled circle with the diameter of size using the specified color.

You can now use some of the above methods to draw a shape on the screen. For this first example, we will keep it very simple, we will draw a simple square:

```
# Draw a square
turtle.forward(50)
turtle.right(90)
turtle.forward(50)
turtle.right(90)
turtle.forward(50)
turtle.right(90)
turtle.forward(50)
turtle.right(90)
```

The above moves the cursor forward 50 pixels then turns 90° before repeating these steps three times. The end result is that a square of 50 × 50 pixels is drawn on the screen:



Note that the cursor is displayed during drawing (this can be turned off with `turtle.hideturtle()` as the cursor was originally referred to as the turtle).

3.2.3 Drawing Shapes

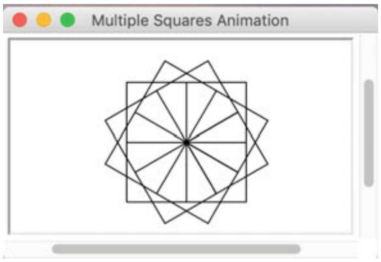
Of course you do not need to just use fixed values for the shapes you draw, you can use variables or calculate positions based on expressions etc.

For example, the following program creates a sequences of squares rotated around a central location to create an engaging image:

```
import turtle

def setup():
    """ Provide the config for the screen """
    turtle.title('Multiple Squares Animation')
    turtle.setup(100, 100, 0, 0)
    turtle.hideturtle()

def draw_square(size):
    """ Draw a square in the current direction """
    turtle.forward(size)
    turtle.right(90)
    turtle.forward(size)
    turtle.right(90)
    turtle.forward(size)
    turtle.right(90)
    turtle.forward(size)

setup()

for _ in range(0, 12):
    draw_square(50)
    # Rotate the starting direction
    turtle.right(120)

# Add this so that the window will close when clicked on
turtle.exitonclick()
```

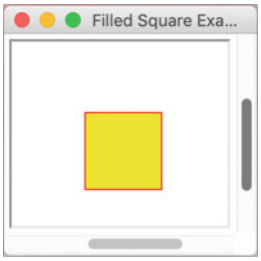
In this program two functions have been defined, one to setup the screen or window with a title and a size and to turn off the cursor display. The second function takes a size parameter and uses that to draw a square. The main part of the program then sets up the window and uses a for loop to draw 12 squares of 50 pixels each by continuously rotating 120° between each square. Note that as we do not need to reference the loop variable we are using the '_' format which is considered an anonymous loop variable in Python.

The image generated by this program is shown below:



3.2.4 Filling Shapes

It is also possible to fill in the area within a drawn shape. For example, you might wish to fill in one of the squares we have drawn as shown below:



To do this we can use the `begin_fill()` and `end_fill()` functions:

- `begin_fill()` indicates that shapes should be filled with the current fill colour, this function should be called just before drawing the shape to be filled.
- `end_fill()` called after the shape to be filled has been finished. This will cause the shape drawn since the last call to `begin_fill()` to be filled using the current fill colour.
- `filling()` Return the current fill state (`True` if filling, `False` if not).

The following program uses this (and the earlier `draw_square()` function) to draw the above filled square:

```
turtle.title('Filled Square Example')
turtle.setup(100, 100, 0, 0)
turtle.hideturtle()

turtle.pencolor('red')
turtle.fillcolor('yellow')
turtle.begin_fill()

draw_square(60)

turtle.end_fill()
turtle.done()
```

3.3 Other Graphics Libraries

Of course Turtle Graphics is not the only graphics option available for Python; however other graphics libraries do not come pre-packed with Python and must be downloaded using a tool such as Anaconda, PIP or PyCharm.

- **PyQtGraph**. The PyQtGraph library is pure Python library oriented towards mathematics, scientific and engineering graphic applications as well as GUI applications. For more information see http://www.pyqtgraph.org.
- **Pillow**. Pillow is a Python imaging library (based on PIL the Python Imaging library) that provides image processing capabilities for use in Python. For more information on Pillow see https://pillow.readthedocs.io/en/stable.
- **Pyglet**. pyglet is another windowing and multimedia library for Python. See https://bitbucket.org/pyglet/pyglet/wiki/Home.

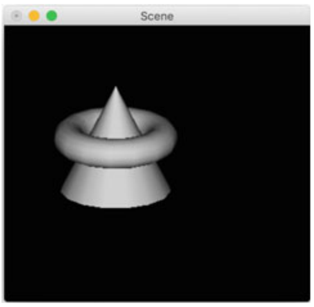
3.4 3D Graphics

Although it is certainly possible for a developer to create convincing 3D images using Turtle Graphics; it is not the primary aim of the library. This means that there is no direct support for creating 3D images other than the basic cursor moving facilities and the programers skill.

However, there are 3D graphics libraries available for Python. One such library is Panda3D (https://www.panda3d.org) while another is VPython (https://vpython.org) while a third is pi3d (https://pypi.org/project/pi3d). However we will briefly look at the PyOpenGL library as this builds on the very widely used OpenGL library.

3.4.1 PyOpenGL

PyOpenGL his an open source project that provides a set of bindings (or wrappings around) the OpenGL library. OpenGL is the Open Graphics Library which is a cross language, cross platform API for rendering 2D and 3D vector graphics. OpenGL is used in a wide range of applications from games, to virtual reality, through data and information visualisation systems to Computer Aided Design (CAD) systems. PyOpenGL provides a set of Python functions that call out from Python to the underlying OpenGL libraries. This makes it very easy to create 3D vector based images in Python using the industry standard OpenGL library. A very simple examples of an image created using PyOpenGL is given below:



3.5 Online Resources

The following provide further reading material:

- https://docs.python.org/3/library/turtle.html Turtle graphics documentation.
- http://pythonturtle.org/ The Python Turtle programming environment—this intended for teaching the basic concepts behind programming using the Turtle graphics library.
- http://pyopengl.sourceforge.net The PyOpenGL home page.
- https://www.opengl.org The OpenGL home page.

3.6 Exercises

The aim of this exercise is to create a graphic display using Python Turtle Graphics.

You should create a simple program to draw an octagon on the Turtle Graphics screen.

Modify your program so that there is an hexagon drawing function. This function should take three parameters, the x, and y coordinates to start drawing the octagon and the size of each side of the octagon.

Modify your program to draw the hexagon in multiple locations to create the following picture: