M30299 – Programming

moodle.port.ac.uk

Week 3 – Worksheet P3: Programming with Graphics

Introduction

This worksheet is designed to help you develop your programming skills by writing programs that use graphics. The intention is not to write programs with fully interactive graphical user-interfaces (GUIs), which can be complicated (GUI development is covered in Year 2). Rather, we concentrate here on using a simple graphics system (associated with Zelle's textbook) that will enable you to draw diagrams and allow some basic user interaction with the mouse and keyboard. You will use the skills developed in this practical to write programs that use graphics throughout the Python part of the module. Note that the material discussed in this worksheet is covered in more detail in Chapter 4 of the module textbook.

Work through this worksheet at your own pace. As always, make sure you study each line of code and try to predict its effects before you enter it, and afterwards make sure you understand what it has done and why. Furthermore, feel free to experiment with the shell until you fully understand each concept.

Copy the graphics module

The Python graphics module that we will use is not part of the standard Python software. To use it on your own machine, copy the file ***graphics.py*** from the Software section of the module's Moodle area and save it to somewhere in your file-space where Python will be able to find it (typically the site-packages folder in your Python installation – see the "Python Software and Books" link on Moodle).

Configure Pyzo to use graphics

To ensure that Pyzo can run your graphics programs properly, you first need to make a change to one of its settings. Click on "Python" immediately above the shell, and choose "Edit shell configurations" from the menu. For "gui" choose "Tk - Tk widget toolkit", and then click "Done". Restart the shell and make sure it says "This is the Pyzo interpreter with integrated event loop for TK." (If you don't change this setting, it will be difficult to close graphics windows that your programs opens.)

Import the graphics module

We will experiment with the graphics module using the shell. To check that Python can find the graphics module, import it using:

```
import graphics
```

Now, create a graphics window using the following assignment statement:

```
win = graphics.GraphWin()
```

A small, empty, graphics window should appear which you can access from within the shell using the variable `win`. For now, simply close the window as follows:

```
win.close()
```

Notice, as with the math module studied earlier, we have to type `graphics.GraphWin()`

to use the definition `GraphWin`. Since we'll be using many definitions from the graphics module, it would be quite tedious to use the `graphics.` prefix every time. We can avoid this by using an alternative form of import. Type:

```
from graphics import *
```

This means "import all definitions from the graphics module and make them available without needing to use a prefix". Now, try the following command:

```
win = GraphWin()
```

to create a new graphics window.

Objects and points

Let's begin by plotting some points on the graphics window. The standard dimensions of the window are 200 by 200 ***pixels*** (picture elements). Note that the top-left pixel has coordinates $(0, 0)$ and the bottom-right pixel is at $(199, 199)$. Therefore, the x-coordinates increase from left to right, and y-coordinates increase from ***top to bottom***. (This coordinate system is standard for computer graphics systems.)

Let's create a point with coordinates $(30, 40)$. To do this, enter:

```
p = Point(30, 40)
```

(Note that the point is not yet drawn on the graphics window.) What we have done here is to create an ***object*** of a data type called `Point`, and assigned this object to the new variable `p`. The `Point` data type is defined in the graphics module using ***object-oriented*** programming techniques. Data types that are defined using object-oriented techniques are called ***classes***. You will learn how to define your own classes, albeit in Java rather than Python, in the Java part of the module. All we need to know here is that objects have a ***state*** (i.e. they contain some data) and that we use objects by calling their ***methods*** using the so-called "dot" notation.

There are two types of methods. ***Accessor*** methods access information about the object. For example, the methods `getX` and `getY` give us the x and y coordinates of a point. Try:

```
p.getX()
p.getY()
```

There are also ***mutator methods***, which change the state of the object in some way. For example, try:

```
p.draw(win)
```

to draw the point on the graphics window `win`, and:

```
p.move(20, 10)
```

to move the point 20 pixels right and 10 pixels down. Try the `getX` and `getY` methods again yourself. To introduce another `Point` object, we use a new variable. Type:

```
q = Point(100, 70)
q.draw(win)
```

We now have two `Point` objects, one we use via the variable `p`, and one using variable `q`. To make it clear which one is which, let's change their colours using another mutator method:

```
q.setOutline("red")
p.setOutline("blue")
```

Now, experiment with the `getX`, `getY` and `move` methods until you understand how these

objects work. Try to move the points so that the red one (`q`) is positioned ten pixels directly above the blue one (`p`).

Lines, shapes and text

The graphics module also defines types (classes) for lines, basic shapes, and text. Let's first make a `Circle` object with centre at point $(100, 100)$ and radius 20 pixels. To do this, we can first make a `Point` for the centre:

```
centre = Point(100, 100)
```

Then we make a circle using this centre point and a value for the radius:

```
circle1 = Circle(centre, 20)
```

We then display the circle in the graphics window `win`:

```
circle1.draw(win)
```

Let's make another circle, but this time let's avoid the use of a `Point` variable:

```
circle2 = Circle(Point(30, 30), 10)
circle2.draw(win)
```

We can move and colour the circles in various ways. Try the following:

```
circle1.setWidth(3)
circle1.setOutline("cyan")
circle2.setFill("green")
circle2.move(10, 20)
circle2.getRadius()
```

Other things that can be drawn include lines, rectangles, polygons and text labels. Lines are specified by their two end points. Try the following:

```
line = Line(Point(20, 25), Point(30, 60))
line.draw(win)
line.move(0, 50)
line.setOutline("yellow")
```

Rectangles are specified by their top-left and bottom-right points; try:

```
rectangle = Rectangle(Point(10,30), Point(40, 100))
rectangle.draw(win)
rectangle.setFill("black")
```

Polygons are specified using any number of points for their vertices.

```
triangle = Polygon(Point(100, 20), Point(120, 80), Point(140, 10))
triangle.draw(win)
triangle.setFill("yellow")
```

Text labels are specified using a centre-point and a string; try:

```
text = Text(Point(100, 100), "Hello World")
text.draw(win)
text.setSize(20)
text.setTextColor("magenta")
text.setText("Goodbye")
```

Window size and coordinate transformations

When creating a new graphics window, you can specify both a title (a string) and pixel dimensions. Close the current graphics window and try, for example:

```
win = GraphWin("My Window", 330, 160)
```

It is often difficult to deal directly with pixel-based coordinates because of the complex arithmetic involved. We can, however, define a new coordinate system for a graphics window by specifying alternative ***bottom-left and top-right*** coordinates. For example:

```
win.setCoords(0, 0, 1, 1)
```

sets a new coordinate system where the bottom-left point has coordinates $(0, 0)$ and the top-right point has coordinates $(1, 1)$. We can now use ***float-valued coordinates*** between these bounds, and they will be automatically mapped to the pixel-based coordinates of the graphics window. Try:

```
line = Line(Point(0, 0.25), Point(1, 0.25))
line.draw(win)
```

Notice that the line is a quarter (0.25) the way up from the bottom of the window. Experiment with this new coordinate system until you understand how it works.

Graphical programming exercises

Copy the pract3.py file from Moodle to your file-space, and change the details at the top. Your solutions to this week's exercises should be added to your copy of the pract3.py file. If you get stuck with the final question (4), for now just move onto the next part of the worksheet. Note that the module textbook and the Python Software section of Moodle contain a reference guide to all the classes defined in the graphics module.

1. The `drawStickFigure` function is incomplete. Finish it by giving the figure arms and legs.

2. Write a `drawCircle` function which asks the user for the radius of a circle and then draws the circle in the centre of a graphics window.

3. Write a `drawArcheryTarget` function which draws a coloured target consisting of concentric circles of yellow (innermost), red and blue. The sizes of the circles should be in correct proportion – i.e. the red circle should have a radius twice that of the yellow circle, and the blue circle should have a radius three times that of the yellow circle. (Hint: objects drawn later will appear on top of objects drawn earlier.)

4. Write a `drawRectangle` function which asks the user for the height and width of a rectangle and draws it in the centre of a graphics window of size 200×200 (i.e. with an equal space to the left and right of the rectangle, and also above and below the rectangle). Assume that the user enters values less than 200. (Hint: you need to work out the coordinates of the top-left and bottom-right points from the rectangle dimensions and the dimensions of the window — think about how much space there should be to the left of, and above, the rectangle.)

Interactive graphics

We will now investigate how to make graphics programs interactive. In general, writing programs where the user has full control (e.g. where he/she can select between several

buttons at any time) is complicated and beyond the capabilities of the graphics module. We will write some code that achieves just a basic level of interactivity.

Handling mouse clicks

We can obtain the position of where the user clicks her/his mouse on a graphics window using its `getMouse` method. When `getMouse` is called, it waits for the user to click on the window, and then gives a `Point` representing the coordinates of the clicked pixel. Try the following code (you'll have to click somewhere in the graphics window after entering the second line!)

```
win = GraphWin("Click Me!")
p = win.getMouse()
print(p.getX(), p.getY())
p.draw(win)
```

Try re-writing (or copying and pasting) the final three lines within a loop as follows:

```
for i in range(5):
    p = win.getMouse()
    print(p.getX(), p.getY())
    p.draw(win)
```

What happens?

Handling textual input

You can use text `Entry` boxes to allow the user to enter strings into a graphics window, and the method `getText` to obtain the strings entered by the user. For example, try the following:

```
win = GraphWin("Greeter", 400, 300)
message = Text(Point(200, 100), "Enter your name & click mouse")
message.draw(win)
inputBox = Entry(Point(200, 200), 10)
inputBox.draw(win)
win.getMouse()
message.setText("Hello, " + inputBox.getText())
```

Make sure you understand each line of code entered above before moving on to attempt the following exercises. (Note that the + in the final line joins two strings together — see lecture 05.)

Interactive graphical programming exercises

Your solutions to the following exercises should be added to your copy of practP.py. Begin by studying the `drawLine` function, and execute it to see what it does. Notice the use of `win.getMouse()` followed by `win.close()` to close the window.

5. Write a `blueCircle` function that allows the user to draw a blue circle of radius 50 by clicking the location of its centre on the window.

6. The function `drawLine` allows the user to draw a line by choosing two points of his/her choice. Write a function `tenLines` that allows the user to draw 10 such lines. (Hint: combine the code from `drawLine` with a loop that uses range(10).)

7. Write a `tenStrings` function which allows the user to plot 10 strings of their choice at locations of a graphics window chosen by clicking on the mouse (the strings should be entered one-by-one by the user within a text entry box at the top of the graphics window, clicking the mouse after entering each one).

8. Write a `tenColouredRectangles` function to allow the user to draw 10 coloured rectangles on the screen. The user should choose the coordinates of the top-left and bottom-right corners by clicking on the window. The colour of each rectangle should be chosen by the user by entering a colour in a text entry box at the top of the window. (The colour of each rectangle is given by the string that is in this box when the user clicks its bottom-right point.) The entry box should initially contain the string "blue". (Assume that the user never enters an invalid colour into the entry box.)

9. [harder] Write a `fiveClickStickFigure` function that allows the user to draw a (symmetric) stick figure in a graphics window using five clicks of the mouse to determine the positions of its features, as illustrated in Figure 1. Each feature should be drawn as the user clicks the points. (Hint: the radius of the head is the distance between points 1 and 2 — see the previous worksheet.) Note that only the y-coordinate of point (3) should be used—its x coordinate should be copied from point (1).

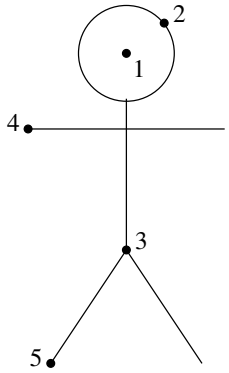

Figure 1: A five-click stick figure.

10. [harder] Write a `plotRainfall` function, which plots a histogram for daily rainfall figures over a 7 day period. The rainfall figures should be entered one-by-one into a text entry box within the window. The bars of the histogram should be numbered along the bottom.