M30299 – Programming
Lecture 04 – Graphics and Objects

Matthew Poole & Nadim Bakhshov
moodle.port.ac.uk

School of Computing
University of Portsmouth

2020/21

Introduction to lecture

- In this lecture we'll take a look at how to incorporate some **graphics** into our programs.
- We won't write programs with complex graphical user interfaces.
- Instead, we'll write programs that will use familiar concepts; e.g
  - points, lines & shapes – circles, rectangles, polygons . . . , and
  - basic interaction using mouse clicks and text

  to learn more of the basics of programming.
- We'll introduce some **object-oriented programming** concepts (class, object, object construction, method, reference) as we go.

Using the graphics module

- The graphics system is not built in to the Python language.
- Instead, it is a Python module (i.e. file) written by John Zelle.
- This module **defines** a number of new data types or **classes**.
- What do you think some of these types/classes may be?
- We then need to **import** it; e.g. using the shell:

```
>>> from graphics import *
```

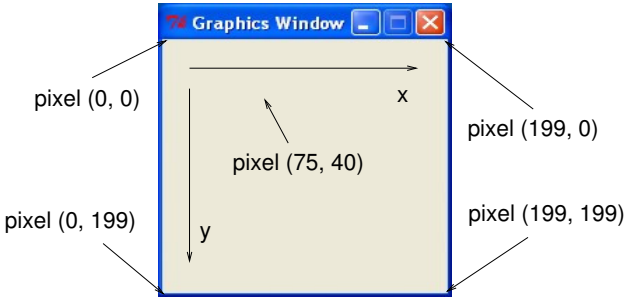
- This means "import all the definitions from the graphics module so we can use them without using the `graphics.` prefix."

Creating a graphics window

- We can now create a graphics window and assign it to a variable:

```
>>> win = GraphWin()
```

- This will give a graphics window of dimensions 200 × 200 **pixels** (picture elements):



Graphical data

- We now wish to draw points, lines, rectangles, circles, polygons, text labels, text entry boxes, etc. on the graphics window.
- All of these are **classes** defined in the graphics module; e.g.:
  - `Point`
  - `Line`
  - `Rectangle`
  - `Circle`
  - `Polygon`
  - `Text`
  - `Entry`
  - `GraphWin`

Creating graphical objects

- It is easy to use/create data values of the built-in types:

```
>>> x = 1.23
```

- How can we create a point **object** (data value of type `Point`)?
- We use the name of the data type with some defining values.
- For example, a point is "defined" by its x and y **coordinates**; so to create a point at $x = 10$ and $y = 20$, we do:

```
>>> p = Point(10, 20)
```

- This is known as **constructing** a `Point` **object**.

Manipulating graphical objects

- The variable p now represents a "point" with coordinates (10, 20).
- We now wish to manipulate the point in various ways; e.g.
  - display it in the graphics window;
  - change its colour, or move it.
- These operations are examples of **methods**.
- We use methods using the **"dot" notation**; for example:

```
>>> p.draw(win)
Point(10.0, 20.0)
>>> p.setOutline("red")
>>> p.move(50,10)
```

- We see that coordinates are stored in the `Point` object as floats.

Methods

- In the above, `draw`, `setOutline` and `move` are methods of the `Point` class.
- The dot notation for calling methods takes the general form:

```
object.method(argument1, argument2, ...)
```

- Different methods will need different arguments of different types:
  - `draw` needs a graphics window on which to draw the point;
  - `setOutline` needs a string representing a colour; and
  - `move` needs a horizontal and a vertical distance (`ints`).

Creating and manipulating objects

- Let's also create a circle with centre coordinates $(20, 30)$ and radius 10, fill it in blue and display it:

```
>>> c = Circle(Point(20, 30), 10)
>>> c.setFill("blue")
>>> c.draw(win)
Circle(Point(20.0, 30.0), 10)
```

- (The top part of) our graphics window will now look like:

Accessor methods

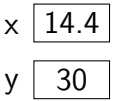
- The above methods for `Point` and `Circle` manipulate (or **mutate**) the objects.
- We can also **access** information using other methods:

```
>>> p.getX()
60.0
>>> p.getY()
30.0
>>> c.getRadius()
10
>>> centre = c.getCenter()
```

- What is the type of the new variable `centre`?

Object diagrams

- We have seen that we can illustrate the values of variables in a program using diagrams such as:



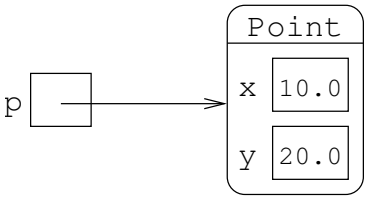
- These diagrams are OK for variables of basic types (e.g. `int`).
- For variables of more complex types (e.g. `Point` & `Circle`), we need to add a little complexity to our diagrams.

Object diagrams

- For example, the value of variable p after the statement:

```
>>> p = Point(10, 20)
```

is best illustrated as:



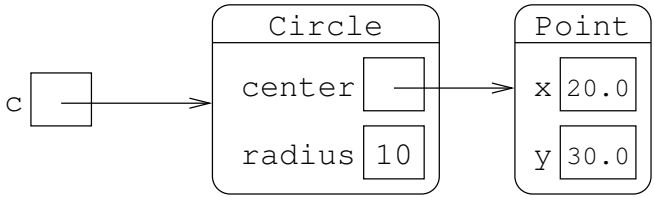
- The box on the right is an **object** of class `Point`.
- The object has internal values (representing x and y coordinates).
- The value of variable p is an arrow (or **reference**) to this object.

Object diagrams

- The following statement:

```
>>> c = Circle(Point(20, 30), 10)
```

results in a more complex diagram:



- Here, the value of the variable `c` is a reference to a `Circle` object.
- The `Circle` object contains a `radius` (an `int`), and a `center` whose value is a reference to a `Point` object.

Copying object references

- The fact that variables' values are references has consequences...
- When we assume that variables contain data values, if we take a copy of a variable's value:

```
>>> x = 42
>>> y = x
```

then changing one:

```
>>> y = y + 1
```

doesn't affect the other:

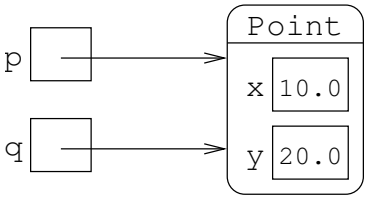
x 42 y 43

Copying object references

- However, if we were to copy the value of p from slide 2:

```
>>> q = p
```

we would obtain:



- If we now make a change using the variable q:

```
>>> q.move(30, 50)
```

this will affect the object referred to by p (why & how?).