Lecture 3.1 – Graphics and Objects

M30299 Programming

School of Computing
University of Portsmouth

Introduction to lecture

- In this lecture we’ll take a look at how to incorporate some **graphics** into our programs.
- We won’t be writing programs with complex graphical user interfaces.
- Instead, we’ll write programs that will use some 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 `graphix` module

- The graphics system we'll use is not built in to the Python language.
- Instead, it is a Python module (file) `graphix.py`, which should be downloaded from Moodle and placed in the same folder as your Python programs.
- This module **defines** a number of new data types or **classes**.
- We need to import any types we wish to use using an import statement:

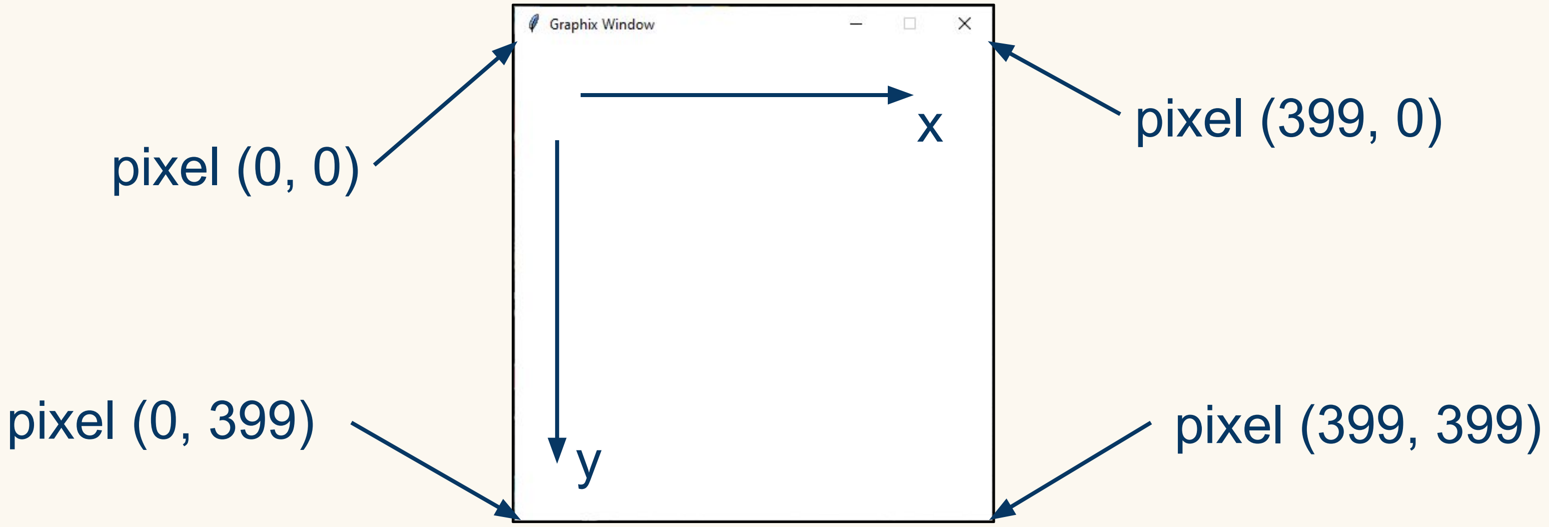
```
>>> from graphix import Window, Point, Circle
```

Creating a `graphix` window

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

```
>>> win = Window()
```

- This will give a window of dimensions 400 × 400 **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.
- Like `Window`, all of these are **classes** defined in the `graphix` module; they are called:
    - `Point`
    - `Line`
    - `Rectangle`
    - `Circle`
    - `Polygon`
    - `Text`
    - `Entry`

Creating graphical objects

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

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

- How can we create a point **object** (a 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 write:

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

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

Accessing an object’s attributes

- Objects each have attributes that we can access, and (in some cases), change.
- We use the **dot notation** to refer to an object’s attributes.
- For example, we can access p’s coordinates (its x and y attributes) as follows:

```
>>> p.x
10
>>> p.y
20
```

- We cannot change p’s coordinates directly, so the following code results in an error:

```
>>> p.x = 50
Traceback ...
```

Using an object's methods

- We can also call an object's **methods** to carry out actions.
- Methods are like functions which we call using the dot notation.
- For example,

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

  draws the `Point` object `p` onto our window, `win`, and:

  ```
  >>> p.move(50, 10)
  ```

  moves the `Point` object `p` 50 pixels to the right and 10 pixels down.
- All graphical objects have the methods `draw` and `move` which are used as above (e.g. we need to pass a `Window` to the `move` method).

A `Circle` object

- Let’s make a `Circle` object; to do this we need to supply a value for the centre (a `Point`) and for the radius (an `int`):

```
>>> c = Circle(Point(20, 100), 50)
```

- `Circle` objects have attributes too; for example we can change and access the outline and fill colours as follows:

```
>>> c.outline_colour = "blue"
>>> c.fill_colour = "red"
>>> c.outline_colour
'blue'
>>> c.fill_colour
'red'
```

`Circle` methods

- We can access the `radius` attribute of a `Circle` object as follows:

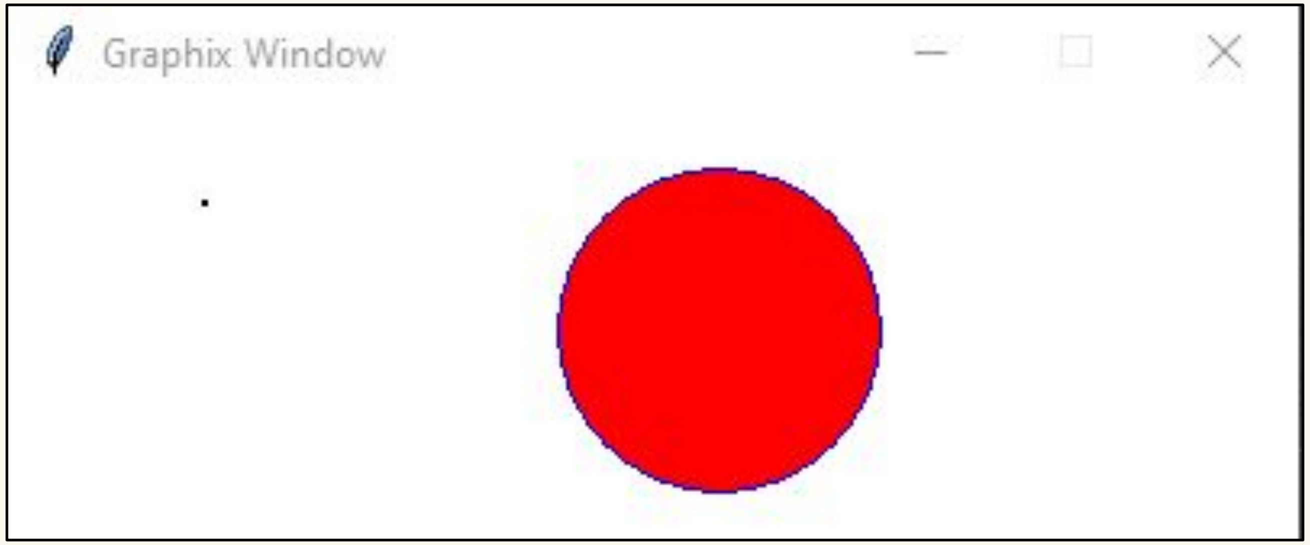
```
>>> c.radius
10
```

- We can draw, move, and obtain a copy of the centre of our `Circle` object by calling methods:

```
>>> c.draw(win)
>>> c.move(200, -30)
>>> c.get_centre()
Point(220, 70)
```

Summary of `graphix`

- After running the code from the previous pages, the top part of our window will look like:



Object diagrams

- In lecture 1 we illustrated the values of variables in a program using diagrams such as:

$$\begin{tikzcd}
\texttt{kilos} \arrow[r] & \texttt{10.0} \\
\texttt{pounds} \arrow[r] & \texttt{22.0}
\end{tikzcd}$$

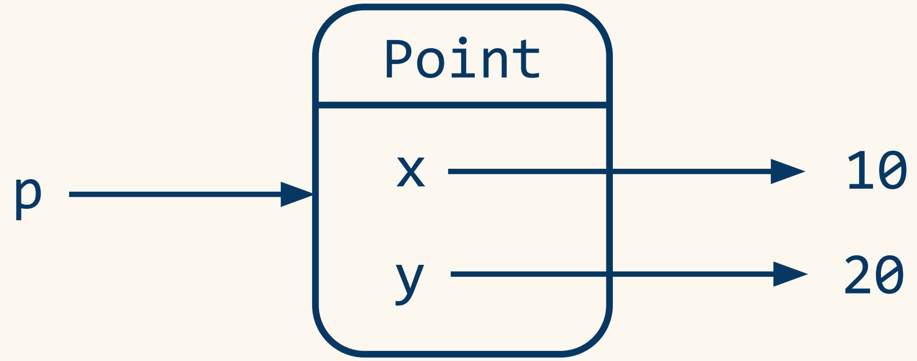
- We can draw equivalent diagrams for variables of our graphical types (e.g. `Point` & `Circle`).

Object diagrams

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

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

  is best illustrated as



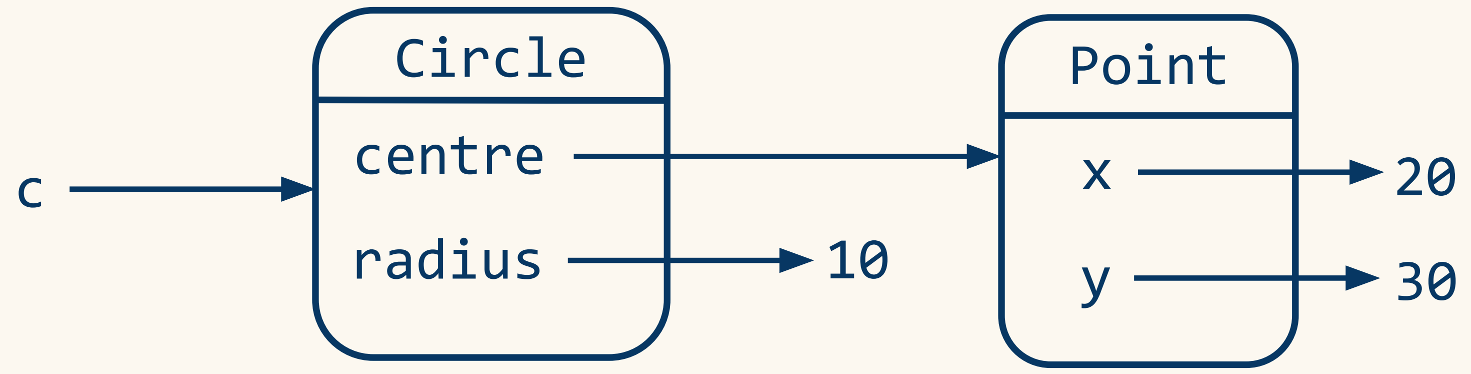
- The box represents an object of type `Point`.
- The value of the variable p is a reference (arrow) to this `Point` object.
- The object includes attributes x and y with values 10 and 20, respectively.

Object diagrams

- The following statement:

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

  results in a slightly more complex diagram:



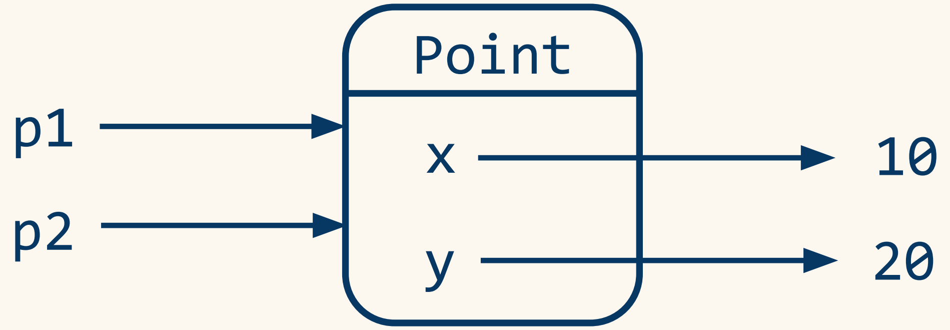
- The value of the variable `c` is a reference to a `Circle` object which contains a `radius`, and a `centre` that refers to a `Point` object.

Object aliasing

- The code:

```
>>> p1 = Point(10, 20)
>>> p2 = p1
```

results in the following, where two variables refer to the same object:



- The object's attributes can now be accessed or modified using either variable:

```
>>> p2.move(200, 0)
>>> p2.x
210
```