CS 157 - Introduction to Programming and Modeling

Graphics and Event-driven Programs

Dr. Stephen P. Carl

Quote of the Day

[Computer programming] resembles the magic of legend in this respect, too. If one character, one pause, of the incantation is not strictly in proper form, the magic doesn't work. Human beings are not accustomed to being perfect, and few areas of human activity demand it. Adjusting to the requirement for perfection is, I think, the most difficult part of learning to program.

- Fred Brooks

The Mythical Man-Month, 1975

Graphics: The Basics

In our *bit-mapped graphical displays,* every screen is divided into a grid of thousands of picture elements, or pixels. Each pixel can be a different color.

- In one sense, a *window* is just a rectangular region of pixels. A window of width 200 and height 300 is made up of 200 * 300 or 60,000 pixels. But in reality a window is much more than that.
- In graphics systems, a window has its own identity, so it is an *object* that can be drawn on, moved, and modified in various ways. Typically, a window is made up of a *frame,* a *title bar,* and a *canvas.*

Graphics: More Basics

With the graphics module, we can write code that draws points, lines, text, or shapes of various kinds. The simple graphics *primitives* can then be combined into more complex objects onscreen.

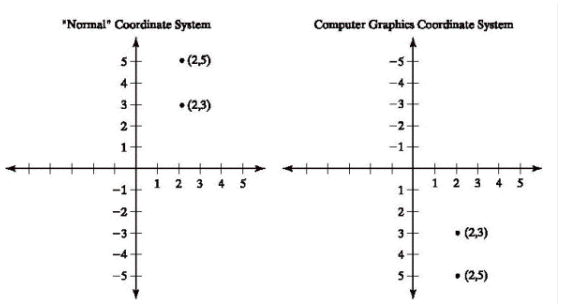
When we want to draw into a window, we actually draw on the window's *canvas.* To draw an object, we first have to know where on the canvas we want it to go, also known as its *coordinates.* A location on the screen is modeled by the `Point` type.

Graphics: More Basics

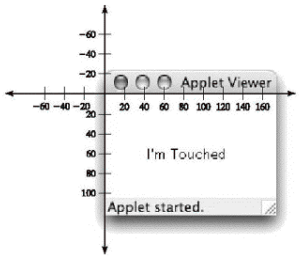
A **`Point`** is simply an *(x, y)* coordinate pair, *x* and *y* both numbers, representing the *row* and *column* of a location on the canvas. This will be a pixel.

For example, the coordinate *(40, 50)* means "40 pixels to the right and 50 pixels down" from the upper left-hand corner of the canvas, which is (0, 0).

Coordinate Systems



Relating a Window to Coordinate



Simple Graphics Example

```
import graphics

window = graphics.GraphWin('Click Me!')


pt = window.getMouse()

txt = graphics.Text(pt, "Click!")

txt.draw(window)


pt = window.getMouse()

txt = graphics.Text(pt, "Click!")

txt.draw(window)


window.getMouse()

window.close()
```

Programming with Graphics

The first statement in the example is:

```
import graphics
```

This tells the system to load the contents of the module **graphics** which contains a set of functions and *class definitions* designed for drawing graphics and interacting with windows.

The names **GraphWin**, **Point**, and **Text** are *classes* defined in this module.

The graphics module was developed by John Zelle, the author of our textbook.

SEWANEE
The University of the South

Understanding the graphics module - classes

To really understand how to use the graphics module, we need to learn about the Python **class** construct.

A *class* defines the *attributes* of a specific set of objects. You've seen this before, when we queried the *type* of the values we've used. For example, if you call the function type(int) the result is <class 'int'>

Python programmers use **class** to create new data types. The class defines the attributes and operations of the new type.

SEWANEE
The University of the South

Understanding the graphics module - methods

A *method* is a function associated with a class, or specific type of object.

Methods are called using the same *dot operator* used to select a specific function from a module. This should reinforce the idea that a class is essentially a *sub-module* - a smaller, more specialized type contained within a module.

In the code example, several methods are called:

```
window = graphics.GraphWin('Click Me!')

window.getMouse()

txt.draw(window)
```

SEWANEE
The University of the South

Understanding the graphics module - methods

Specialized methods called *object constructors,* create new *instances* (a.k.a. objects) of a class type.

The method names in the example that start with a capital letter such as **GraphWin** and **Text** are object constructors.

By convention, Python programmers capitalize these special method names and leave all others lowercase.

SEWANEE
The University of the South

All types have methods

Just so you know, most every type in Python we've looked at already is defined as a class, and has one or more methods defined specifically on that type.

For example, type **<class 'str '>** that we informally call "strings" have several methods that work with them:

```
>>> str = 'ladder'
>>> str.upper()
'LADDER'
>>> str.find('d')
2
```

SEWANEE
The University of the South

The GraphWin class

The next statement in the example:

```
window = graphics.GraphWin()
```

creates an object of type **GraphWin** and assigns it to the variable **window**. **GraphWin** objects provide functionality to manipulate windows, for example plotting points, changing the background, and handling *mouse and keyboard events.*

Object constructors consist of the name of the class followed by a parenthesized list of values.

SEWANEE
The University of the South

The GraphWin class

The parenthesized list is called an *argument list.* In an object construction, the argument list specifies the object's *initializers.* We are using the defaults here, and haven't provided any arguments. **GraphWin** can accept a *title string,* a value for the *width* of the window (in pixels), and a value for the *height* of the window. For example:

```
win = graphics.GraphWin('New Window', 200, 300)
```

SEWANEE
The University of the South

The Text class

The next two statements:

```
txt = graphics.Text(pt, "Click!")
txt.draw(window)
```

construct a **Text** object and *call* its **draw** method.

To construct a **Text** object we pass it the coordinates of the point where the Text will be centered and then the text to display as a string.

The **draw** method operates on the **Text** object. It tells it which window to draw the text into.

SEWANEE
The University of the South

The Point Class

Coordinates are represented by objects belonging to the **Point** class. The method **getMouse** returns a **Point** object giving the coordinates inside the window where the mouse was clicked. We can also create a **Point** using object construction like this:

```
pt = graphics.Point(100, 150)
```

The arguments are the x- and y-coordinate of the Point. We can draw a point as well, like we did the **Text**

SEWANEE
The University of the South

The Line Class

Another graphics class, for drawing lines, is the Line class. An example object construction for Line:

```
ln = graphics.Line(graphics.Point(100, 150), graphics.Point(200, 0))

ln.draw(window)
```

This draws a line from coordinates (100, 150) to (200, 0) on the canvas. Let's see if we can figure out what this would look like in a 200 by 200 window.

Object constructions work the same way for all classes. The only difference is that different classes have different argument lists, because each class has its own set of initializers.

SEWANEE
The University of the South

More Graphics Classes

To draw shapes, we have the following classes:

- Rectangle creates a transparent rectangle with a border. For example:

```
r = graphics.Rectangle(graphics.Point(20, 50), graphics.Point(50, 80))
```

- Circle creates a transparent circle with a border. For example:

```
c = Circle(graphics.Point(30, 40), 10.5)
```

- Oval creates an oval with a border. For example:

```
o = graphics.Oval(graphics.Point(10,20), graphics.Point(30,40))
```

There are also classes for creating arbitrary polygons, entry boxes, and displaying images. See the docs for details.

SEWANEE
The University of the South

Reactive Programs

Our textbook introduces programming in the context of *reactive programs,* which are programs that perform operations only in response to input from the mouse or the keyboard.

but, not all our programs will be strictly reactive

Most modern computer systems have a bit-mapped graphical display with a mouse for interacting with the screen. The first full-scale implementation of this type of system was done by Xerox PARC in the 1970's, and brought to "the rest of us" by Apple's Macintosh in 1984. This style of *graphical user interface* encourages writing reactive programs.

SEWANEE
The University of the South