Chapter 18

Further Graphical Programming

18.1 Python 2 vs Python 3

As of this writing, the most recent version of Python is 3.2, and all the code in this book is designed to run in Python 3.2. The tricky thing is that as of version 3.0, Python broke compatibility with older versions of Python. Code written in those older versions will not always work in Python 3. The problem with this is there were a number of useful libraries written for Python 2 that, as of this writing, have not yet been ported to Python 3. We want to use these libraries, so we will have to learn a little about Python 2. Fortunately, there are only a few big differences that we have to worry about.

Division The division operator, `/`, in Python 2, when used with integers, behaves like `//`. For instance, `5/4` in Python 2 evaluates to 1, whereas `5/4` in Python 3 evaluates to 1.2. This is the way the division operator behaves in a number of other programming languages. In Python 3, the decision was made to make the division operator behave the way we are used from math.

In Python 2, if you want to get 1.25 by dividing 5 and 4, you need to do `5/4.0`. At least one of the arguments has to be a float in order for the result to be a float. If you are dividing two variables, then instead of `x/y`, you may need to do `x/float(y)`.

`print` The `print` function in Python 3 was actually the `print` *statement* in Python 2. So in Python 2, you would write

```
print 'Hello'
```

without any parentheses. This code will no longer work in Python 3 because the `print` statement is now the `print` function, and functions need parentheses. Also, the current `print` function has those useful optional arguments, `sep` and `end`, that are not available in Python 2.

input The Python 2 equivalent of the `input` function is `raw_input`.

range The `range` function can be inefficient with very large ranges in Python 2. The reason is that in Python 2, if you use `range(10000000)`, Python will create a list of 10 million numbers. The `range` statement in Python 3 is more efficient and instead of generating all 10 million things at once, it only generates the numbers as it needs them. The Python 2 function that acts like the Python 3 `range` is `xrange`.

String formatting String formatting in Python 2 is a little different than in Python 3. When using the formatting codes inside curly braces, in Python 2, you need to specify an argument number. Compare the examples below:

```
Python 2: 'x={0:3d},y={1:3d},z={2:3d}'.format(x,y,z)
Python 3: 'x={:3d},y={:3d},z={:3d}'.format(x,y,z)
```

As of Python 3.1, specifying the argument numbers was made optional.

There is also an older style of formatting that you may see from time to time that uses the `%` operator. An example is shown below along with the corresponding new style.

```
Python 2: 'x=%3d, y=%6.2f, z=%3s' % (x,y,z)
Python 3: 'x={:3d},y={:6.2f},z={:3s}'.format(x,y,z)
```

Module names Some modules were renamed and reorganized. Here are a few Tkinter name changes:

Python 2	Python 3
Tkinter	tkinter
ScrolledText	tkinter.scrolledtext
tkMessageBox	tkinter.messagebox
tkFileDialog	tkinter.filedialog

There are a number of other modules we'll see later that were renamed, mostly just changed to lowercase. For instance, `Queue` in Python 2 is now `queue` in Python 3.

Dictionary comprehensions Dictionary comprehensions are not present in Python 2.

Other changes There are quite a few other changes in the language, but most of them are with features more advanced than we consider here.

Importing future behavior The following import allows us to use Python 3's division behavior in Python 2.

```
from __future__ import division
```

There are many other things you can import from the future.

18.2 The Python Imaging Library

The Python Imaging Library (PIL) contains useful tools for working with images. As of this writing, the PIL is only available for Python 2.7 or earlier. The PIL is not part of the standard Python distribution, so you'll have to download and install it separately. It's easy to install, though.

PIL hasn't been maintained since 2009, but there is a project called Pillow that it nearly compatible with PIL and works in Python 3.0 and later.

We will cover just a few features of the PIL here. A good reference is *The Python Imaging Library Handbook*.

Using images other than GIFs with Tkinter Tkinter, as we've seen, can't use JPEGs and PNGs. But it can if we use it in conjunction with the PIL. Here is a simple example:

```
from Tkinter import *
from PIL import Image, ImageTk

root = Tk()
cheetah_image = ImageTk.PhotoImage(Image.open('cheetah.jpg'))

button = Button(image=cheetah_image)
button.grid(row=0, column=0)

mainloop()
```

The first line imports Tkinter. Remember that in Python 2 it's an uppercase `Tkinter`. The next line imports a few things from the PIL. Next, where we would have used Tkinter's `PhotoImage` to load an image, we instead use a combination of two PIL functions. We can then use the image like normal in our widgets.

Images PIL is the Python *Imaging* Library, and so it contains a lot of facilities for working with images. We will just show a simple example here. The program below displays a photo on a canvas and when the user clicks a button, the image is converted to grayscale.

```
from Tkinter import *
from PIL import Image, ImageTk

def change():
    global image, photo
    pix = image.load()
```

```
    for i in range(photo.width()):
        for j in range(photo.height()):
            red,green,blue = pix[i,j]
            avg = (red+green+blue)//3
            pix[i,j] = (avg, avg, avg)
    photo=ImageTk.PhotoImage(image)
    canvas.create_image(0,0,image=photo,anchor=NW)

def load_file(filename):
    global image, photo
    image=Image.open(filename).convert('RGB')
    photo=ImageTk.PhotoImage(image)
    canvas.configure(width=photo.width(), height=photo.height())
    canvas.create_image(0,0,image=photo,anchor=NW)
    root.title(filename)

root = Tk()
button = Button(text='Change', font=('Verdana', 18), command=change)
canvas = Canvas()
canvas.grid(row=0)
button.grid(row=1)
load_file('pic.jpg')

mainloop()
```

Let's first look at the `load_file` function. Many of the image utilities are in the Image module. We give a name, `image`, to the object created by the `Image.open` statement. We also use the `convert` method to convert the image into RGB (Red-Green-Blue) format. We will see why in a minute. The next line creates an `ImageTk` object called `photo` that gets drawn to the Tkinter canvas. The `photo` object has methods that allow us to get its width and height so we can size the canvas appropriately.

Now look at the `change` function. The `image` object has a method called `load` that gives access to the individual pixels that make up the image. This returns a two-dimensional array of RGB values. For instance, if the pixel in the upper left corner of the image is pure white, then `pix[0,0]` will be `(255,255,255)`. If the next pixel to the right is pure black, `pix[1,0]` will be `(0,0,0)`. To convert the image to grayscale, for each pixel we take the average of its red, green, and blue components, and reset the red, green, and blue components to all equal that average. Remember that if the red, green, and blue are all the same, then the color is a shade of gray. After modifying all the pixels, we create a new `ImageTk` object from the modified pixel data and display it on the canvas.

You can have a lot of fun with this. Try modifying the change function. For instance, if we use the following line in the `change` function, we get an effect that looks like a photo negative:

```
pix[i,j] = (255-red, 255-green, 255-blue)
```

Try seeing what interesting effects you can come up with.

Note, though, that this way of manipulating images is the slow, manual way. PIL has a number of much faster functions for modifying images. You can very easily change the brightness, hue, and contrast of images, resize them, rotate them, and much more. See the PIL reference materials for more on this.

putdata If you are interested drawing mathematical objects like fractals, plotting points pixel-by-pixel can be very slow in Python. One way to speed things up is to use the `putdata` method. The way it works is you supply it with a list of RGB pixel values, and it will copy it into your image. Here is a program that plots a 300 × 300 grid of random colors.

```
from random import randint
from Tkinter import *
from PIL import Image, ImageTk

root = Tk()
canvas = Canvas(width=300, height=300)
canvas.grid()
image=Image.new(mode='RGB',size=(300,300))

L = [(randint(0,255), randint(0,255), randint(0,255))
      for x in range(300) for y in range(300)]

image.putdata(L)

photo=ImageTk.PhotoImage(image)
canvas.create_image(0,0,image=photo,anchor=NW)
mainloop()
```

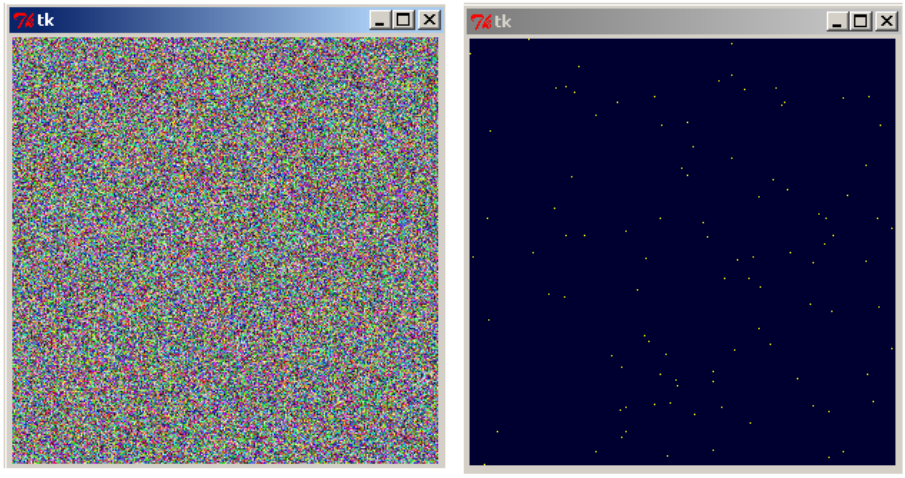

Figure 18.1: (Left) putdata example (Right) ImageDraw example

ImageDraw The `ImageDraw` module gives another way to draw onto images. It can be used to draw rectangles, circles, points, and more, just like Tkinter canvases, but it is faster. Here is a short example that fills the image with a dark blue color and then 100 randomly distributed yellow points.

```
from random import randint
from Tkinter import *
```

```
from PIL import Image, ImageTk, ImageDraw

root = Tk()
canvas = Canvas(width=300, height=300)
canvas.grid()
image=Image.new(mode='RGB',size=(300,300))
draw = ImageDraw.Draw(image)

draw.rectangle([(0,0),(300, 300)],fill='#000030')
L = [(randint(0,299), randint(0, 299)) for i in range(100)]
draw.point(L, fill='yellow')

photo=ImageTk.PhotoImage(image)
canvas.create_image(0,0,image=photo,anchor=NW)
mainloop()
```

To use `ImageDraw`, we have to first create an `ImageDraw` object and tie it to the `Image` object. The `draw.rectangle` method works similarly to the `create_rectangle` method of canvases, except for a few differences with parentheses. The `draw.point` method is used to plot individual pixels. A nice feature of it is we can pass a list of points instead of having to plot each thing in the list separately. Passing a list is also much faster.

18.3 Pygame

Pygame is a library for creating two-dimensional games in Python. It can be used to can make games at the level of old arcade or Nintendo games. It can be downloaded and easily installed from www.pygame.org. There are a number of tutorials there to help you get started. I don't know a whole lot about Pygame, so I won't cover it here, though perhaps in a later edition I will.