Chapter 20. Py Art

Well, art is art, isn't it? Still, on the other hand, water is water! And east is east and west is west, and if you take cranberries and stew them like applesauce, they taste much more like prunes than rhubarb does.

—Groucho Marx

This chapter and the next two discuss the application of Python to some common human endeavors: art, business, and science. If you're interested in any of these areas, you may get some helpful ideas or the urge to try something new.

2-D Graphics

All computer languages have been applied to computer graphics to some degree. Many of the heavy-duty platforms in this chapter were written in C or C++ for speed, but added Python libraries for productivity. Let's begin by looking at some 2-D imaging libraries.

Standard Library

Only a few image-related modules are in the standard library:

`imghdr`
: Detects the file type of some image files.

`colorsys`
: Converts colors between various systems: RGB, YIQ, HSV, and HLS.

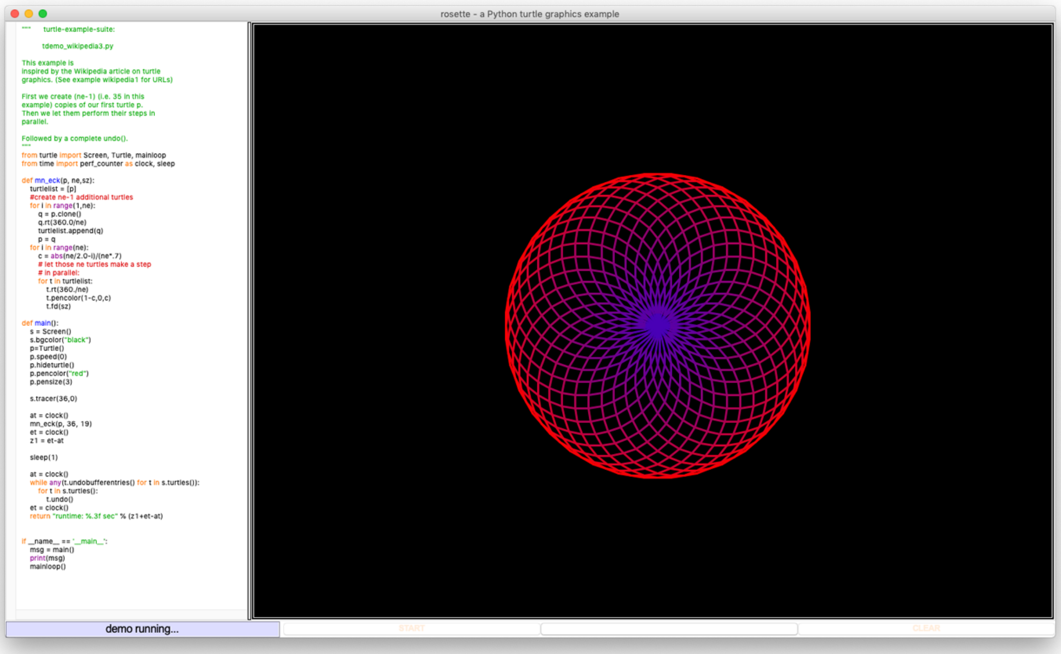
If you downloaded the O'Reilly logo to a local file called *oreilly.png*, you could run this:

```
>>> import imghdr
>>> imghdr.what('oreilly.png')
'png'
```

Another standard library is `turtle`—“Turtle graphics,” which is sometimes used to teach programming to young people. You can run a demo with this command:

```
$ python -m turtledemo
```

Figure 20-1 shows a screenshot of its *rosette* example.



```
"""      turtle-example-suite:

         tdemo_wikipedia3.py

This example is
inspired by the Wikipedia article on turtle
graphics. (See example wikipedia1 for URLs)

First we create (ne-1) (i.e. 35 in this
example) copies of our first turtle p.
Then we let them perform their steps in
parallel.

Followed by a complete undo().
"""
from turtle import Screen, Turtle, mainloop
from time import perf_counter as clock, sleep

def mn_eck(p, ne,sz):
    turtlelist = [p]
    #create ne-1 additional turtles
    for i in range(1,ne):
        q = p.clone()
        q.rt(360.0/ne)
        turtlelist.append(q)
        p = q
    for i in range(ne):
        c = abs(ne/2.0-i)/(ne*.7)
        # let those ne turtles make a step
        # in parallel:
        for t in turtlelist:
            t.rt(360./ne)
            t.pencolor(1-c,0,c)
            t.fd(sz)

def main():
    s = Screen()
    s.bgcolor("black")
    p=Turtle()
    p.speed(0)
    p.hideturtle()
    p.pencolor("red")
    p.pensize(3)

    s.tracer(36,0)

    at = clock()
    mn_eck(p, 36, 19)
    et = clock()
    z1 = et-at

    sleep(1)

    at = clock()
    while any(t.undobufferentries() for t in s.turtles()):
        for t in s.turtles():
            t.undo()
    et = clock()
    return "runtime: %.3f sec" % (z1+et-at)


if __name__ == '__main__':
    msg = main()
    print(msg)
    mainloop()
```

Figure 20-1. Image from turtledemo

To do anything serious with graphics in Python, we need to get some third-party packages. Let’s see what’s out there.

PIL and Pillow

For many years, the Python Image Library (PIL), although not in the standard library, has been Python’s best-known 2-D image processing library. It predated installers such as `pip`, so a “friendly fork” called Pillow was created. Pillow’s imaging code is backward-compatible with PIL, and its documentation is good, so let’s use it here.

Installation is simple; just type the following command:

```
$ pip install Pillow
```

If you've already installed operating system packages such as `libjpeg`, `libfreetype`, and `zlib`, they'll be detected and used by Pillow. See the installation page for details on this.

Open an image file:

```
>>> from PIL import Image
>>> img = Image.open('oreilly.png')
>>> img.format
'PNG'
>>> img.size
(154, 141)
>>> img.mode
'RGB'
```

Although the package is called `Pillow`, you import it as `PIL` to make it compatible with the older `PIL`.

To display the image on your screen using the `Image` object's `show()` method, you'll first need to install the ImageMagick package described in the next section, and then try this:

```
>>> img.show()
```

The image displayed in Figure 20-2 opens in another window. (This screenshot was captured on a Mac, where the `show()` function used the Preview application. Your window's appearance might vary.)

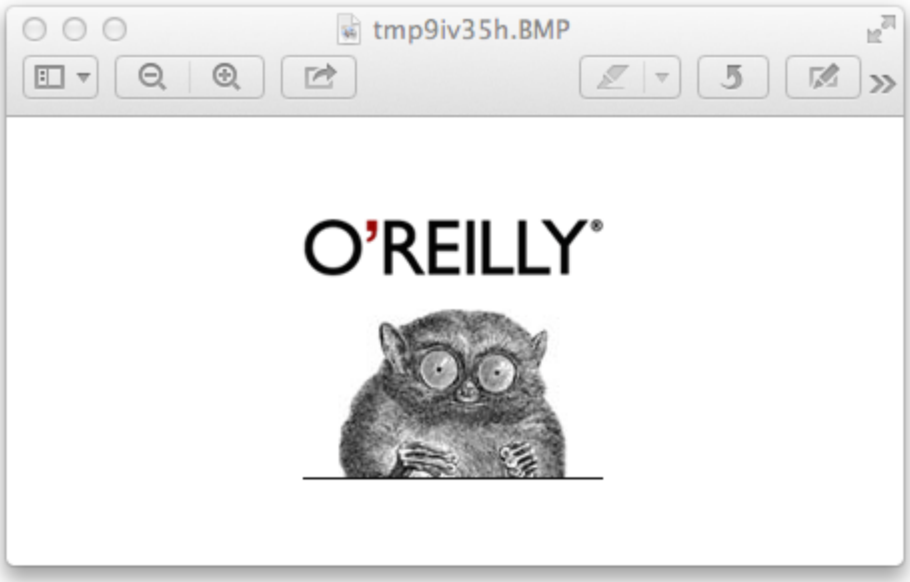


Figure 20-2. Image displayed with the Python Image Library

Let's crop the image in memory, save the result as a new object called `img2`, and display it. Images are always measured by horizontal (x) values and vertical (y) values, with one corner of the image known as the *origin* and arbitrarily assigned an x and y of 0. In this library, the origin (0, 0) is at the upper left of the image, x increases to the right, and y increases as you move down. We want to give the values of left x (55), top y (70), right x (85), and bottom y (100) to the `crop()` method, so pass it a tuple with those values in that order:

```
>>> crop = (55, 70, 85, 100)
>>> img2 = img.crop(crop)
>>> img2.show()
```

The results are shown in Figure 20-3.

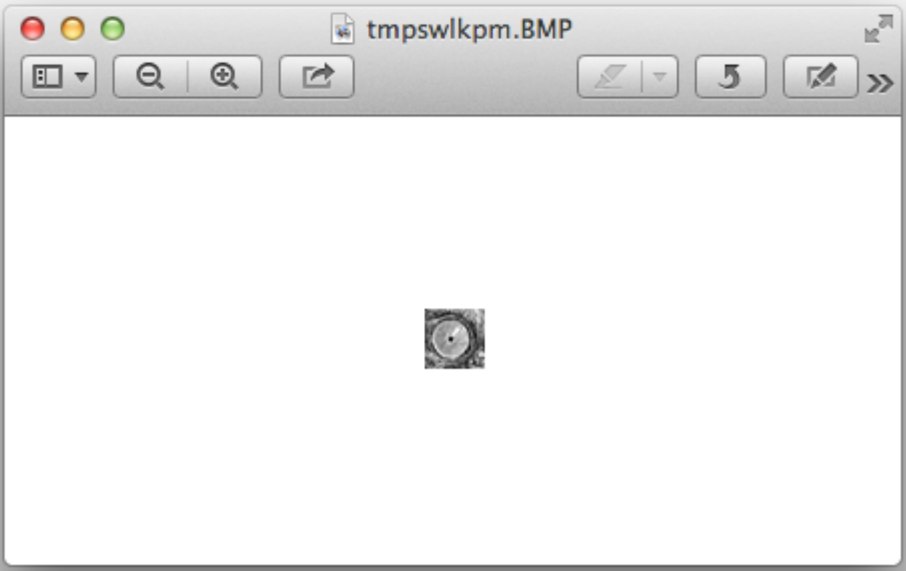


Figure 20-3. The cropped image

Save an image file with the `save` method. It takes a filename and an optional type. If the filename has a suffix, the library uses that to determine the type. But you can also specify the type explicitly. To save our cropped image as a GIF file, do the following:

```
>>> img2.save('cropped.gif', 'GIF')
>>> img3 = Image.open('cropped.gif')
>>> img3.format
'GIF'
>>> img3.size
(30, 30)
```

For our last example, let's "improve" our little mascot. First download copies of our original critter, shown in Figure 20-4.

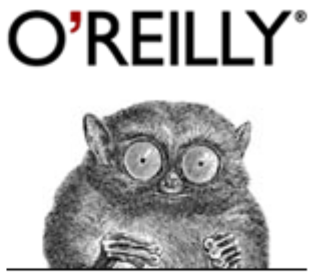


Figure 20-4. Beloved ur-critter

He has a sort of scruffy five o'clock shadow, so let's get an image to improve his, um, image; see Figure 20-5.

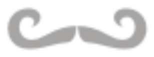


Figure 20-5. Alien technology

Let's put them together, with some *alpha* channel magic to make the overlap semi-transparent, demonstrated in Example 20-1.

Example 20-1. ch20_critter.py

```
from PIL import Image

critter = Image.open('ch20_critter.png')
stache = Image.open('ch20_stache.png')
stache.putalpha(100)
img = Image.new('RGBA', critter.size, (255, 255, 255, 0))
img.paste(critter, (0, 0))
img.paste(stache, (45, 90), mask=stache)
img.show()
```

Figure 20-6 presents his makeover.

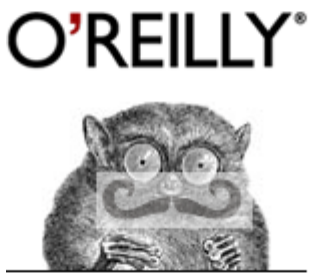


Figure 20-6. Our new, dapper mascot

ImageMagick

ImageMagick is a suite of programs to convert, modify, and display 2-D bitmap images. It's been around for more than 20 years. Various Python libraries have connected to the ImageMagick C library. A recent one that supports Python 3 is wand. To install it, type the following command:

```
$ pip install Wand
```

You can do many of the same things with `wand` as you can with Pillow:

```
>>> from wand.image import Image
>>> from wand.display import display
>>>
>>> img = Image(filename='oreilly.png')
>>> img.size
(154, 141)
>>> img.format
'PNG'
```

As with Pillow, this displays the image on the screen:

```
>>> display(img)
```

`wand` includes rotation, resizing, text and line drawing, format conversion, and other features that you can also find in Pillow. Both have good APIs and documentation.

3-D Graphics

Some basic Python packages include the following:

- VPython has examples that can run in your browser.
- pi3d runs on the Raspberry Pi, Windows, Linux, and Android.
- Open3D is a full-featured 3-D library.

3-D Animation

Watch the long end-credits for almost any contemporary movie, and you'll see mass quantities of people doing special effects and animation. Most of the big studios—Walt Disney Animation, ILM, Weta, Dreamworks, Pixar—hire people with Python experience. Do a web search for "python animation jobs" to see what's available now.

Some Python 3-D packages are:

Panda3D
: It's open source and free to use, even for commercial applications. You can download a version from the Panda3D website.

VPython
: Comes with many examples.

Blender
: Blender is a free 3-D animation and game creator. When you download and install it, it comes bundled with its own copy of Python 3.

Maya
: This is a commercial 3-D animation and graphic system. It also comes bundled with a version of Python, currently 2.7. Chad Vernon has written a free downloadable book, *Python Scripting for Maya Artists*. If you search for Python and Maya on the web, you'll find many other resources, both free and commercial, including videos.

Houdini
: Houdini is commercial, although you can download a free version called Apprentice. Like the other animation packages, it comes with a Python binding.

Graphical User Interfaces

The name includes the word graphic, but graphical user interfaces (GUIs) concentrate more on the user interface: widgets to present data, input methods, menus, buttons, and windows to frame everything.

The GUI programming wiki page and FAQ list many Python-powered GUIs. Let's begin with the only one that's built in to the standard library: Tkinter. It's plain, but it works on all platforms to produce native-looking windows and widgets.

Here's a teeny, tiny Tkinter program to display our favorite googly-eyed mascot in a window:

```
>>> import tkinter
>>> from PIL import Image, ImageTk
>>>
>>> main = tkinter.Tk()
>>> img = Image.open('oreilly.png')
>>> tkimg = ImageTk.PhotoImage(img)
>>> tkinter.Label(main, image=tkimg).pack()
>>> main.mainloop()
```

Notice that it used some modules from PIL/Pillow. You should see the O'Reilly logo again, as shown in Figure 20-7.

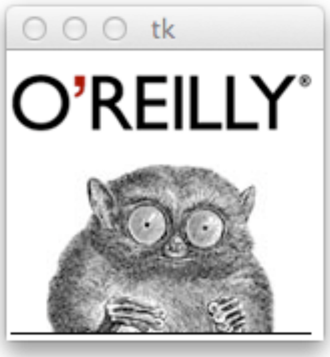


Figure 20-7. Image displayed with Tkinter

To make the window go away, click its close button, or leave your Python interpreter.

You can read more about Tkinter at the tkinter wiki. Now for the GUIs that are not in the standard library:

Qt
: This is a professional GUI and application toolkit, originated about 20 years ago by Trolltech in Norway. It's been used to help build applications such as Google Earth, Maya, and Skype. It was also used as the base for KDE, a Linux desktop. There are two main Python libraries for Qt: PySide is free (LGPL license), and PyQt is licensed ei-

ther with the GPL or commercially. The Qt folks see these differences. Download PySide from PyPI or Qt and read the tutorial. You can download Qt for free online.

GTK+

GTK+ is a competitor of Qt, and it, too, has been used to create many applications, including GIMP and the Gnome desktop for Linux. The Python binding is PyGTK. To download the code, go to the PyGTK site, where you can also read the documents.

WxPython

This is the Python binding for WxWidgets. It's another hefty package, free to download online.

Kivy

Kivy is a free modern library for building multimedia user interfaces portably across platforms—desktop (Windows, macOS, Linux), and mobile (Android, iOS). It includes multitouch support. You can download for all the platforms on the Kivy website. Kivy includes application development tutorials.

PySimpleGUI

Write native or web-based GUIs with one library. PySimpleGUI is a wrapper for some of the other GUIs mentioned in this section, including Tk, Kivy, and Qt.

The web

Frameworks such as Qt use native components, but some others use the web. After all, the web is a universal GUI, and it has graphics (SVG), text (HTML), and even multimedia now (in HTML5). You can build web applications with any combination of frontend (browser-based) and backend (web server) tools. A *thin client* lets the backend do most of the work. If the frontend dominates, it's a *thick*, or *fat*, or *rich* client; the last adjective sounds more flattering. It's common for the sides to communicate with RESTful APIs, Ajax, and JSON.

Plots, Graphs, and Visualization

Python has become a leading solution for plots, graphs, and data visualization. It’s especially popular in science, which is covered in Chapter 22. Useful overviews, with examples, include the official Python wiki and the Python Graph Gallery.

Let’s look at the most popular ones. In the next chapter, you’ll see some of these again, but being used to create maps.

Matplotlib

The Matplotlib 2-D plotting library can be installed by using the following command:

```
$ pip install matplotlib
```

The examples in the gallery show the breadth of Matplotlib.

Let’s first try the same image display application (with results shown in Figure 20-8), just to see how the code and presentation look:

```
import matplotlib.pyplot as plot
import matplotlib.image as image

img = image.imread('oreilly.png')
plot.imshow(img)
plot.show()
```

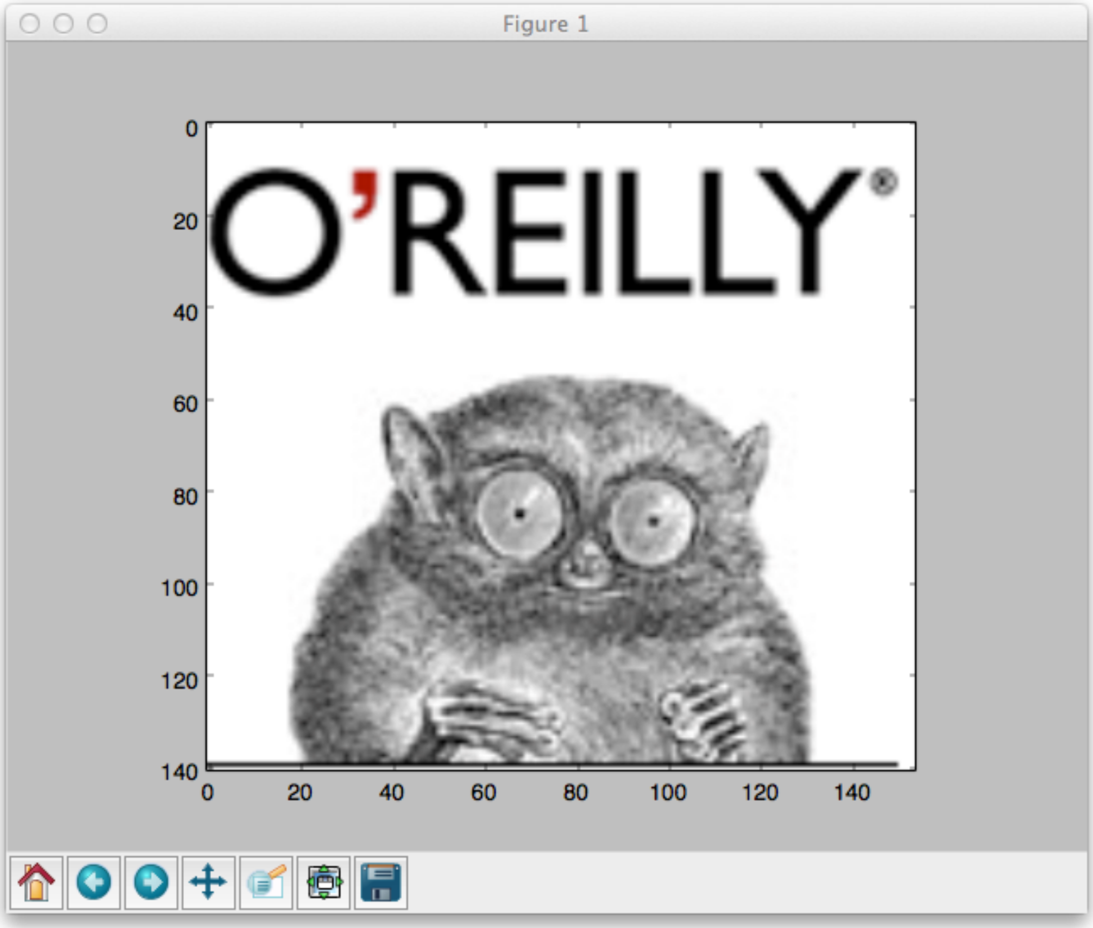


Figure 20-8. Image displayed with Matplotlib

The real strength of Matplotlib is in *plot*ting, which is, after all, its middle name. Let's generate two lists of 20 integers, one smoothly increasing from 1 to 20, and another like the first, but with slight wobbles now and then (Example 20-2).

Example 20-2. ch20_matplotlib.py

```
import matplotlib.pyplot as plt
from random import randint

linear = list(range(1, 21))
wiggly = list(num + randint(-1, 1) for num in linear)

fig, plots = plt.subplots(nrows=1, ncols=3)

ticks = list(range(0, 21, 5))
for plot in plots:
    plot.set_xticks(ticks)
    plot.set_yticks(ticks)

plots[0].scatter(linear, wiggly)
plots[1].plot(linear, wiggly)
plots[2].plot(linear, wiggly, 'o-')

plt.show()
```

If you run this program, you'll see something like what's shown in Figure 20-9 (not exactly, because the `randint()` calls make random wiggles).

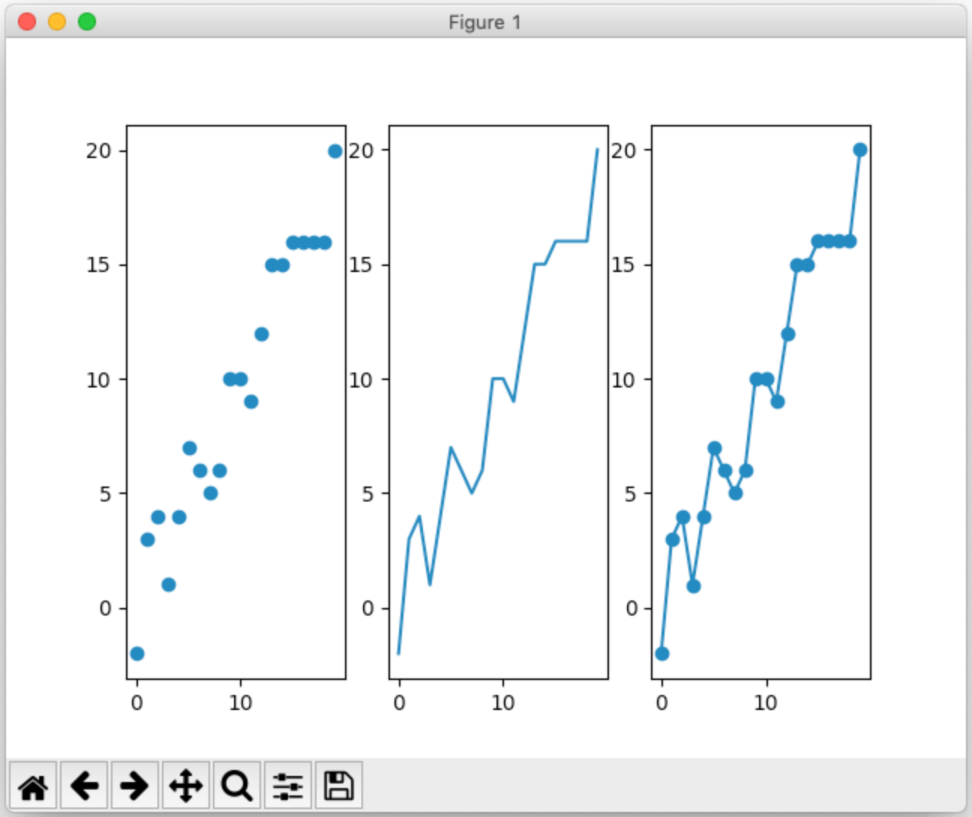


Figure 20-9. Basic Matplotlib scatter and line plots

This example showed a scatterplot, a line plot, and a line plot with data markers. All of the styles and colors used Matplotlib defaults, but they can be customized very extensively. For details, see the Matplotlib site or an overview like Python Plotting With Matplotlib (Guide).

You can see more of Matplotlib in Chapter 22; it has strong ties to NumPy and other scientific applications.

Seaborn

Seaborn is a data visualization library (Figure 20-10), built on Matplotlib and with connections to Pandas. The usual installation mantra (`pip install seaborn`) works.

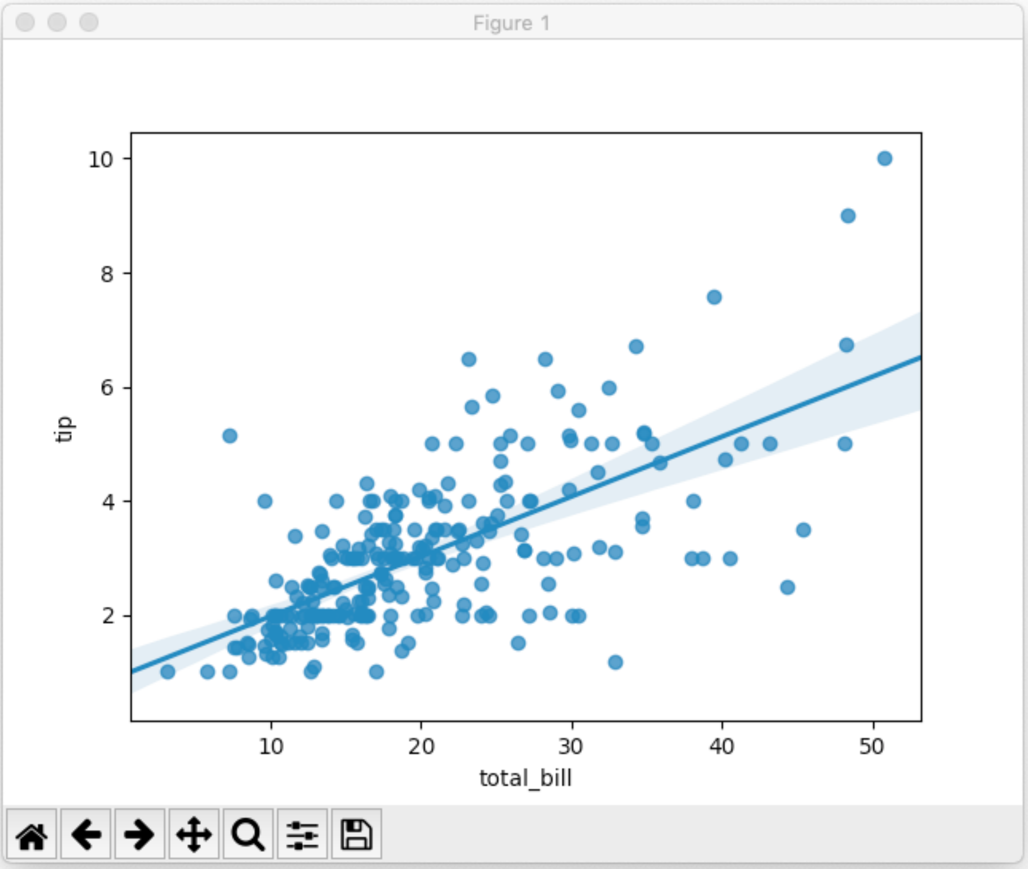


Figure 20-10. Basic Seaborn scatter plot and linear regression

The code in Example 20-3 is based on a Seaborn example; it accesses test data on restaurant tipping and plots tips versus total bill amounts with a fitted linear regression line.

Example 20-3. ch20_seaborn.py

```
import seaborn as sns
import matplotlib.pyplot as plt

tips = sns.load_dataset("tips")
sns.regplot(x="total_bill", y="tip", data=tips);

plt.show()
```

NOTE

If you run the preceding code with the standard Python interpreter, you need that initial import line (`import matplotlib.pyplot as plt`) and final line (`plt.show()`), as shown in Example 20-3, or else the plot just won't display. If you're using Jupyter, Matplotlib is built in and you don't need to type them. Remember this when you read code examples of Python mapping tools.

Like Matplotlib, Seaborn has a vast number of options for data handling and display.

Bokeh

In the old web days, developers would generate graphics on the server and give the web browser some URL to access them. More recently, JavaScript has gained performance and client-side graphics generation tools like D3. A page or two ago, I mentioned the possibility of using Python as part of a frontend-backend architecture for graphics and GUIs. A new tool called Bokeh combines the strengths of Python (large data sets, ease of use) and JavaScript (interactivity, less graphics latency). Its emphasis is quick visualization of large data sets.

If you've already installed its prerequisites (NumPy, Pandas, and Redis), you can install Bokeh by typing this command:

```
$ pip install bokeh
```

(You can see NumPy and Pandas in action in Chapter 22.)

Or, install everything at once from the Bokeh website. Although Matplotlib runs on the server, Bokeh runs mainly in the browser and can take advantage of recent advances on the client side. Click any image in the gallery for an interactive view of the display and its Python code.

Games

Python is such a good game de-vel-op-ment plat-form that peo-ple have writ-ten books about it:

- *Invent Your Own Computer Games with Python* by Al Sweigart
- *The Python Game Book* by

Horst Jens (a docuwiki book)

There’s a general discussion at the Python wiki with even more links.

The best known Python game platform is probably pygame. You can download an ex- e- cutable in- staller for your plat- form from the Pygame web- site, and read a line- by- line ex- am- ple of a “pum- mel the chimp” game.

A u

d
i
o
a
n
d
M
u
s
i
c

I
sought
the
serif
But
that
did
not
suit
Claude
Debussy.

—Deservedly
Anonymous

What
about
sound,

and mu- sic, and cats singing “Jingle Bells”? Well, as Meatloaf says, two out of three ain’t bad.

It’s hard to rep- re- sent sound in a printed book, so here are some up- to- date links to Python packages for sound and music, but Google has many more:

- Standard library audio modules
- Third-party audio tools
- Dozens of third-party music applica-

tions: graphic and CLI play- ers, con- vert- ers, no- ta- tion, anal- y- sis, playlists, MIDI, and more

Finally, how about some on- line sources of mu- sic? You've seen code ex- am- ples throughout this book that access the Internet Archive. Here are links to some of its audio archives:

- Audio recordings (>5 million)
- Live music (>200,000)
- Live Grateful Dead shows (>13,000)

C
o
m
i
n
g
U
p

Act
busy!
It’s
Python
in
busi-
ness.

T
h
i
n
g
s
t
o
D

o

20.1 Install `matplotlib.` Draw a scatter diagram of these (x, y) pairs: `((0, 0), (3,` `5),(6,2),(9,8),(14,10))`.

20.2 Draw a line graph of the same data.

20.3 Draw a

plot
(a
line
graph
with
mark-
ers)
of
the
same
data.