第5章 图形可视化

CHAPTER 5

数据可视化、数据分析是 Python 的主要应用场景之一，Python 提供了丰富的数据分析、数据展示库来支持数据的可视化。数据可视化分析对于挖掘数据的潜在价值、制定企业决策都有非常大的帮助。

5.1 Matplotlib 可视化

Matplotlib 是 Python 中常用的可视化工具之一，使用它可以非常方便地创建海量类型的二维(2D)图表和一些基本的三维(3D)图表。当前基于 Python 的 Matplotlib 在科学计算领域都得到了广泛应用。

5.1.1 安装 Matplotlib

使用 pip 工具来安装 Matplotlib 库，首先需升级 pip：

```
python3 -m pip install -U pip
```

安装 Matplotlib 库的代码如下：

```
python3 -m pip install -U matplotlib
```

安装完成后，可以通过 import 来导入 Matplotlib：

```
import matplotlib
```

可通过以下代码查看 Matplotlib 库的版本号：

```
import matplotlib
print(matplotlib.__version__)
```

5.1.2 Matplotlib Pyplot

Pyplot 是 Matplotlib 的子库，提供了和 MATLAB 类似的绘图 API。Pyplot 是常用的绘图模块，能方便地绘制 2D 图表。它包含一系列绘图的相关函数，每个函数会对当前的图像进行一些修改，例如给图像加上标记、生成新的图像、在图像中产生新的绘图区域等。使用时，可以使用 import 导入 Pyplot 库，并设置一个别名 plt，如：

```
import matplotlib.pyplot as plt
```

这样就可以使用 plt 来引用 Pyplot 包的方法。以下是一些常用的 pyplot 函数。

- plot()：用于绘制线图和散点图。

- scatter()：用于绘制散点图。
- bar()：用于绘制垂直条形图和水平条形图。
- hist()：用于绘制直方图。
- pie()：用于绘制饼图。
- imshow()：用于绘制图像。
- subplots()：用于创建子图。

1. 绘制点和线

除了这些基本的函数，Pyplot 还提供了很多其他函数，例如用于设置图表属性的函数、用于添加文本和注释的函数、用于保存图表到文件的函数等。

利用 plot()函数可以绘制点和线，语法格式如下：

```
plot([x], y, [fmt], *, data = None, **kwargs)          #画单条线
plot([x], y, [fmt], [x2], y2, [fmt2], …, **kwargs)     #画多条线
```

其中，各参数含义如下。

- x，y：点或线的节点，x 为 x 轴数据，y 为 y 轴数据，数据可以是列表或数组。
- fmt：可选，用于定义基本格式(如颜色、标记和线条样式)。
- ** kwargs：可选，用在二维平面图上，设置指定属性，如标签、线的宽度等。

此外，颜色、标记和线条样式如下。

- 颜色字符：'b'表示蓝色，'m'表示洋红色，'g'表示绿色，'y'表示黄色，'r'表示红色，'k'表示黑色，'w'表示白色，'c'表示青绿色，'#008000'表示 RGB 颜色符串。多条曲线不指定颜色时，会自动选择不同的颜色。
- 线条样式参数：'-'表示实线，'--'表示破折线，'-.'表示点画线，':'表示虚线。
- 标记字符：'.'表示点标记，','表示像素标记(极小点)，'o'表示实心圈标记，'v'表示倒三角标记，'^'表示上三角标记，'>'表示右三角标记，'<'表示左三角标记等。

【例 5-1】 绘制正弦和余弦曲线。

解析：在 plt.plot()参数中包含两对 x，y 值，第一对是 x，y，对应正弦函数，第二对是 x，z，对应余弦函数。

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0,4 * np.pi,0.1) #开始，停止，步长
y = np.sin(x)
z = np.cos(x)
plt.plot(x,y,'r + ',x,z)
plt.show()
```

运行程序，正弦和余弦曲线如图 5-1 所示。

2. fmt 参数

fmt 参数定义了基本格式，如标记、线条样式和颜色。

```
fmt = '[marker][line][color]'
```

例如“o:r”中，o 表示实心圆标记，: 表示虚线，r 表示颜色为红色。

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([6, 2, 13, 10])
plt.plot(ypoints, 'o:r')      #定义基本格式效果如图 5-2 所示
plt.show()
```

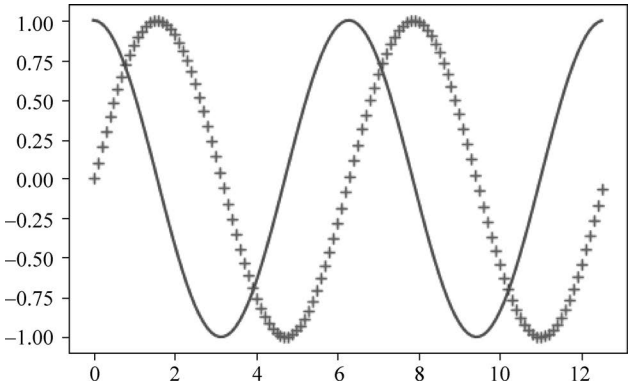

图 5-1　正弦和余弦曲线

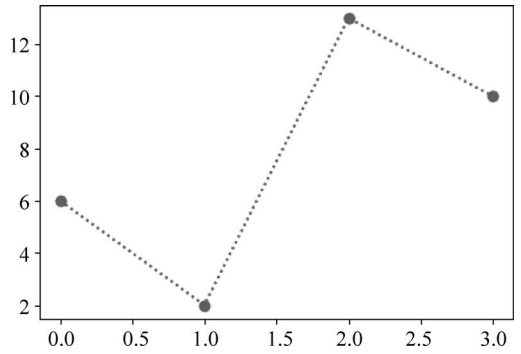

图 5-2　定义基本格式效果

3. 线的宽度

线的宽度可以使用 linewidth 参数来定义，简写为 lw，值可以是浮点数，如 1、2.0、6.5 等。

【例 5-2】 设置指定线的宽度。

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([6, 2, 13, 10])
plt.plot(ypoints, linewidth = '10.5')
plt.show()
```

运行程序，指定线的宽度效果如图 5-3 所示。

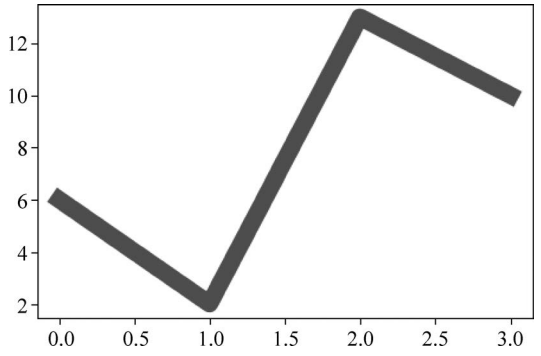

图 5-3　指定线的宽度效果

4. 轴标签和标题

在 Matplotlib 中可以使用 xlabel()和 ylabel()函数分别设置 x 轴和 y 轴的标签。title()

函数用来设置标题。

此外，Matplotlib 作图时默认设置为英文，无法显示中文，若要显示中文，需要添加下面一行代码：

```
plt.rcParams['font.sans-serif'] = ['SimHei']
```

【例 5-3】 在图像中显示中文标注。

```
import numpy as np
from matplotlib import pyplot as plt
import matplotlib
#显示中文
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
x = np.arange(1,11)
y = 2 * x + 5
plt.title("显示中文标注")
plt.xlabel("x 轴")
plt.ylabel("y 轴")
plt.plot(x,y)
plt.show()
```

运行程序，显示中文标注效果如图 5-4 所示。

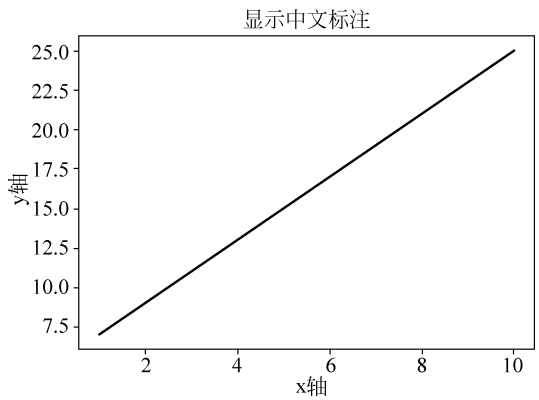

图 5-4　显示中文标注效果

5. 显示负号

一般在数据绘图时，数据的坐标轴部分容易出现负数，但在图中并不显示负号，如图 5-5 所示。

【例 5-4】 图片显示负号演示。

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei']     #设置字体
x = np.linspace(-5, 5, 100)
y = np.cos(x)
plt.plot(x, y)
plt.show()
```

运行程序，不显示负号效果如图 5-5 所示。

由图 5-5 可观察到，图中的负号并没有显示，要解决该问题，在 Maplotlib 中，可通过添加以下语句实现负号的显示，添加代码后运行程序，显示负号效果如图 5-6 所示。

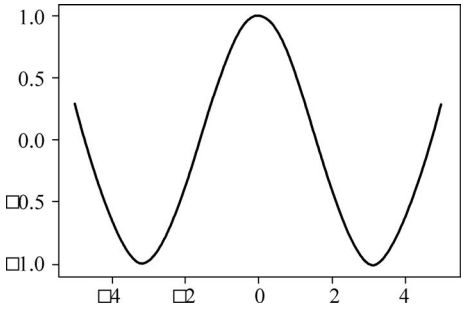

图 5-5　不显示负号

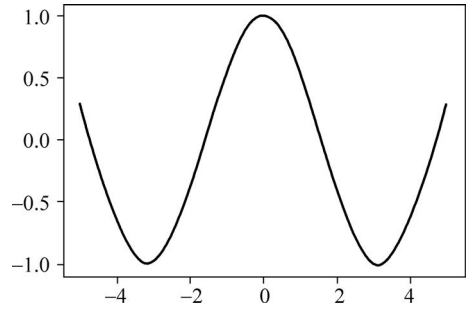

图 5-6　显示负号效果

```
plt.rcParams['axes.unicode_minus'] = False
```

6. 添加网格线

可以使用 Pyplot 中的 grid()函数设置图表中的网格线。语法格式如下：

```
matplotlib.pyplot.grid(b = None, which = 'major', axis = 'both', ** kwargs)
```

其中，各参数含义如下。

- b：可选，默认为 None，可以设置布尔值，true 为显示网格线，false 为不显示，如果设置 ** kwargs 参数，则值为 true。
- which：可选，可选值有'major'、'minor' 和 'both'，默认为'major'，表示应用更改的网格线。
- axis：可选，设置显示哪个方向的网格线，可以取 'both'(默认)、'x'或'y'，分别表示两个方向、x 轴方向或 y 轴方向。
- ** kwargs：可选，设置网格样式，可以是 color＝'r'、linestyle＝'-'和 linewidth＝2，分别表示网格线的颜色、样式和宽度。

【例 5-5】 添加一个简单的网格线，axis 参数使用 x，设置 x 轴方向显示网格线。

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1, 2, 3, 4])
y = np.array([1, 4, 9, 16])
#显示中文
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.title("添加网络实例")
plt.xlabel("x 标签")
plt.ylabel("y 标签")
plt.plot(x, y)
plt.grid(axis = 'x')      #设置 x 就是在 x 轴方向显示网格线
plt.show()
```

运行程序，显示网格线效果如图 5-7 所示。

5.1.3　绘制多子图

在 Matplotlib 中，可以使用 Pyplot 中的 subplot()和 subplots()函数来绘制多个子图。subplot()函数在绘图时需要指定位置，subplots()函数可以一次生成多个子图，在调用时只需要调用生成对象的 ax 即可。

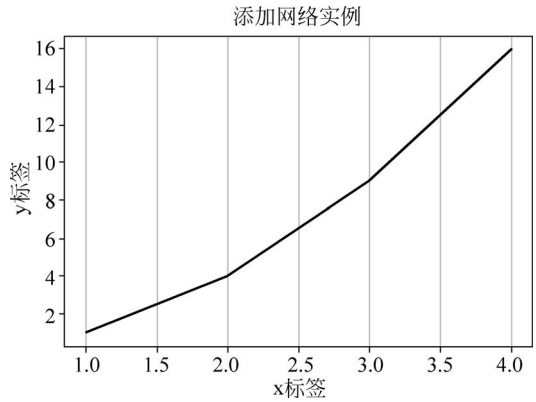

图 5-7　显示网格线效果

1. subplot()函数

subplot()函数的语法格式如下：

```
subplot(nrows, ncols, index, **kwargs)
subplot(pos, **kwargs)
subplot(**kwargs)
subplot(ax)
```

函数将整个绘图区域分成 nrows 行和 ncols 列，然后按照从左到右、从上到下的顺序将子区域编号为 1～N，左上的子区域的编号为 1、右下的子区域编号为 N，编号可以通过参数 index 来设置。

- 如果设置 nrows＝1，ncols＝2，就是将图表绘制成 1×2 的图片区域，对应的坐标为

(1,1)，(1,2)

plotNum＝1，表示的坐标为(1,1)，即第一行第一列的子图。

plotNum＝2，表示的坐标为(1,2)，即第一行第二列的子图。

- 如果设置 nrows＝2，ncols＝2，就是将图表绘制成 2×2 的图片区域，对应的坐标为

(1,1)，(1,2)

(2,1)，(2,2)

plotNum＝1，表示的坐标为(1,1)，即第一行第一列的子图。

plotNum＝2，表示的坐标为(1,2)，即第一行第二列的子图。

plotNum＝3，表示的坐标为(2,1)，即第二行第一列的子图。

plotNum＝4，表示的坐标为(2,2)，即第二行第二列的子图。

【例 5-6】　利用 subplot()绘制多子图。

```
import matplotlib.pyplot as plt
import numpy as np
#显示中文
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
#图 1:
x = np.array([0, 6])
y = np.array([0, 100])
plt.subplot(2, 2, 1)
plt.plot(x,y)
plt.title("plot 1")
#图 2:
```

```
x = np.array([1, 2, 3, 4])
y = np.array([1, 4, 9, 16])
plt.subplot(2, 2, 2)
plt.plot(x,y)
plt.title("plot 2")
#图 3:
x = np.array([1, 2, 3, 4])
y = np.array([3, 5, 7, 9])
plt.subplot(2, 2, 3)
plt.plot(x,y)
plt.title("plot 3")
#图 4:
x = np.array([1, 2, 3, 4])
y = np.array([4, 5, 6, 7])
plt.subplot(2, 2, 4)
plt.plot(x,y)
plt.title("plot 4")
plt.suptitle("subplot 绘制多子图测试")
plt.show()
```

运行程序,subplot 绘制多子图效果如图 5-8 所示。

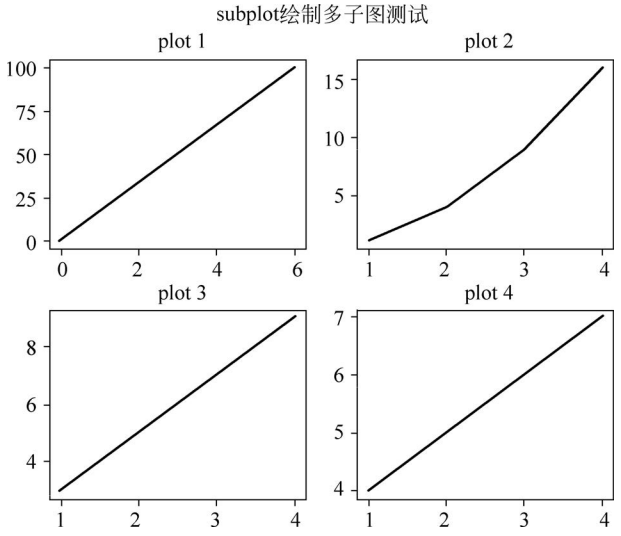

图 5-8 subplot 绘制多子图效果

2. subplots()函数

subplots()函数的语法格式如下:

```
matplotlib.pyplot.subplots(nrows = 1, ncols = 1, *, sharex = False, sharey = False, squeeze = True, subplot_kw = None, gridspec_kw = None, **fig_kw)
```

各参数含义如下。

- nrows:默认为 1,设置图表的行数。
- ncols:默认为 1,设置图表的列数。
- sharex、sharey:设置 x、y 轴是否共享,默认为 false,可设置为'none'、'all'、'row' 或 'col'。为 false 或'none'时,表示每个子图的 x 轴或 y 轴都是独立的;为 true 或'all'时,表示所有子图共享 x 或 y 轴;为'row'时,设置每个子图行共享一个 x 轴或 y 轴;为'col'时,设置每个子图列共享一个 x 轴或 y 轴。
- squeeze:布尔值,默认为 true,表示额外的维度从返回的 Axes(轴)对象中挤出,对于

$N\times1$ 或 $1\times N$ 个子图，返回一个一维数组，对于 $N\times M(N>1$ 和 $M>1)$ 返回一个二维数组。如果设置为 false，则不进行挤压操作，返回一个元素为 Axes 对象的二维数组，即使它最终是 1×1。

- subplot_kw：可选，字典类型。把字典的关键字传递给 add_subplot()来创建每个子图。
- gridspec_kw：可选，字典类型。把字典的关键字传递给 GridSpec 构造函数，创建子图放在网格中(grid)。
- ** fig_kw：把详细的关键字参数传给 figure()函数。

【例 5-7】 利用 subplots()函数创建多子图。

```
import matplotlib.pyplot as plt
import numpy as np

#创建一些测试数据
x = np.linspace(0, 2 * np.pi, 400)
y = np.sin(x ** 2)
#创建 4 个子图
fig, axs = plt.subplots(2, 2, subplot_kw = dict(projection = "polar"))
axs[0, 0].plot(x, y)
axs[1, 1].scatter(x, y)
plt.show()
```

运行程序，subplots()方法创建多子图效果如图 5-9 所示。

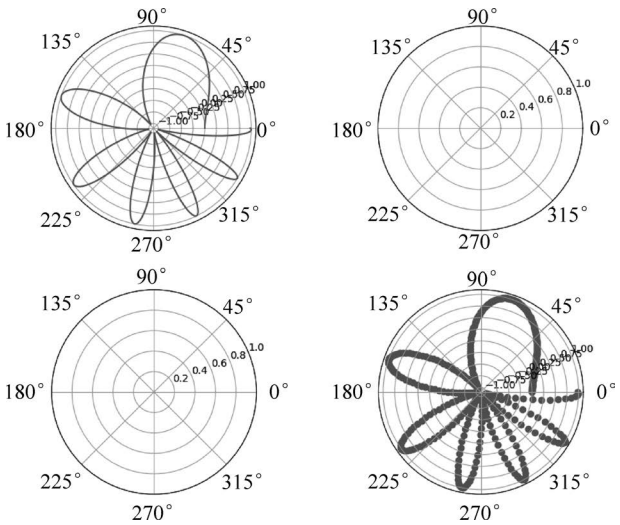

图 5-9　subplots()方法创建多子图效果

5.1.4　散点图

在 Matplotlib 中，可以使用 Pyplot 中的 scatter()函数来绘制散点图。scatter()函数的语法格式如下：

```
matplotlib.pyplot.scatter(x, y, s = None, c = None, marker = None, cmap = None, norm = None, vmin = None, vmax = None, alpha = None, linewidths = None, edgecolors = None, plotnonfinite = False, data = None, ** kwargs)
```

各参数含义如下。

- x,y：长度相同的数组，也就是绘制散点图的数据点(输入数据)。

- s：点的大小，默认为 20，也可以是个数组，数组的每个参数为对应点的大小。
- c：点的颜色，默认为蓝色'b'，也可以是 RGB 或 RGBA 的二维行数组。
- marker：点的样式，默认为小圆圈'o'。
- cmap：默认为 None，标量或者一个 colormap(颜色条)的名称，只有 c 是一个浮点数数组时才使用。如果没有申明即为 image.cmap。
- norm：默认为 None，数据亮度在 0～1，只有 c 是一个浮点数的数组时才使用。
- vmin，vmax：亮度设置，在 norm 参数存在时会忽略。
- alpha：透明度设置，为 0～1，默认为 None，即不透明。
- linewidths：标记点的长度。
- edgecolors：颜色或颜色序列，默认为'face'，可选值有'face'、'none'、None。
- plotnonfinite：布尔值，设置是否使用非限定的 c(inf，-inf 或 nan) 绘制点。
- ** kwargs：其他参数，如透明度、线形颜色、边框边缘颜色等。

【例 5-8】 使用随机数来设置散点图。

```
import numpy as np
import matplotlib.pyplot as plt
#随机数生成器的种子
np.random.seed(19680801)
#显示中文
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = (30 * np.random.rand(N)) ** 2                    #0 到 15 点半径
plt.scatter(x, y, s = area, c = colors, alpha = 0.5)    #设置颜色及透明度
plt.title("散点图")                                      #设置标题
plt.show()
```

运行程序，散点图效果如图 5-10 所示。

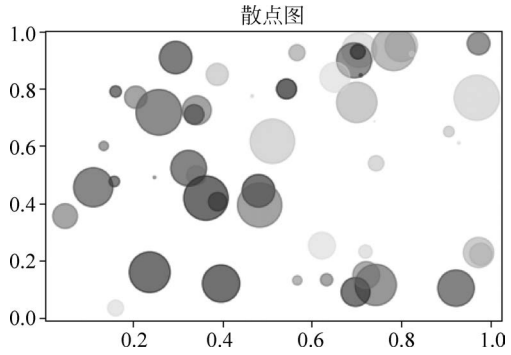

图 5-10 散点图效果

Matplotlib 模块提供了很多可用的颜色条(colormap)，颜色条就像一个颜色列表，设置颜色条需要使用 cmap 参数，默认值为'viridis'，之后颜色值设置为 0～100 的数组。如果要显示颜色条，需要使用 plt.colorbar()函数。

【例 5-9】 显示带颜色条的散点图。

```
import matplotlib.pyplot as plt
import numpy as np
```

```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])
plt.scatter(x, y, c = colors, cmap = 'afmhot_r') #设置颜色条颜色为 afmhot_r
plt.colorbar()                                   #有该语句则显示颜色图,没有则不显示
plt.show()
```

运行程序,显示带颜色条的散点图效果如图 5-11 所示。

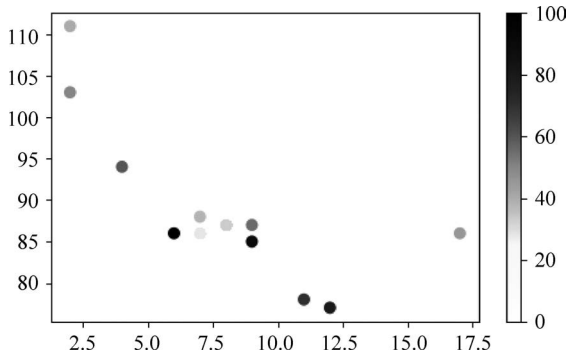

图 5-11　显示带颜色条的散点图效果

5.1.5　柱形图

在 Matplotlib 中,可以使用 Pyplot 中的 bar()函数来绘制柱形图。bar()函数语法格式如下:

```
matplotlib.pyplot.bar(x, height, width = 0.8, bottom = None, align = 'center', data = None,
**kwargs)
```

各参数的含义如下。

- x:浮点型数组,柱形图的 x 轴数据。
- height:浮点型数组,柱形图的高度。
- width:浮点型数组,柱形图的宽度。
- bottom:浮点型数组,底座的 y 坐标,默认为 0。
- align:柱形图与 x 坐标的对齐方式。'center'表示以 x 位置为中心,是默认值。'edge'表示将柱形图的左边缘与 x 位置对齐。要对齐右边缘的条形,可以传递负数的宽度值及设置 align='edge'。

此外,垂直方向的柱形图可以使用 barh()函数来设置。需要注意的是,设置柱形图宽度,bar()函数使用 width 设置,barh()函数使用 height 设置 height。

【例 5-10】 绘制柱形图,并设置相应的宽度。

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(["柱形 1", "柱形 2", "柱形 3", "C-柱形"])
y = np.array([12, 22, 6, 18])
#显示中文
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.subplot(1,2,1);plt.bar(x, y, width = 0.1)
plt.title('垂直柱形图')
plt.subplot(1,2,2);plt.barh(x, y, height = 0.1)
```

```
plt.title('水平柱形图')
plt.show() # 显示图像
```

运行程序，柱形图效果如图 5-12 所示。

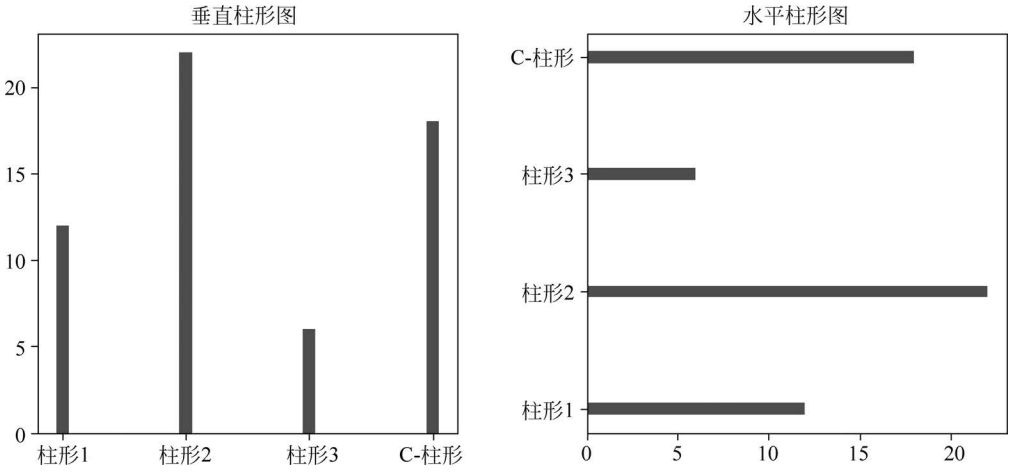

图 5-12　柱形图效果

5.1.6　饼图

饼图是一种常用的数据可视化图形，用来展示各类别在总体中所占的比例。在 Matplotlib 中可以使用 Pyplot 中的 pie()函数来绘制饼图。函数的语法格式如下：

```
matplotlib.pyplot.pie(x, explode = None, labels = None, colors = None, autopct = None, pctdistance = 0.6, shadow = False, labeldistance = 1.1, startangle = 0, radius = 1, counterclock = True, wedgeprops = None, textprops = None, center = 0, 0, frame = False, rotatelabels = False, normalize = None, data = None)[source]
```

各参数的含义如下。

- x：浮点型数组或列表，绘制饼图的数据，表示每个扇形的面积。
- explode：数组，表示各个扇形之间的间隔，默认值为 0。
- labels：列表，表示各个扇形的标签，默认值为 None。
- colors：数组，表示各个扇形的颜色，默认值为 None。
- autopct：设置饼图内各个扇形百分比的显示格式，“%d%%”表示整数百分比，“%0.1f”表示一位小数，“%0.1f%%”表示一位小数百分比，“%0.2f%%”表示两位小数百分比。
- labeldistance：标签标记的绘制位置，是相对于半径的比例，默认值为 1.1，如“<1”表示绘制在饼图内侧。
- pctdistance：类似于 labeldistance，指定 autopct 的位置刻度，默认值为 0.6。
- shadow：布尔值 True 或 False，设置饼图的阴影，默认为 False，不设置阴影。
- radius：设置饼图的半径，默认为 1。
- startangle：用于指定饼图的起始角度，默认为从 x 轴正方向逆时针画起，如设定为 90 则从 y 轴正方向画起。
- counterclock：布尔值，用于指定是否逆时针绘制扇形。默认为 True，即逆时针绘制；False 为顺时针绘制。
- wedgeprops：字典类型，默认值为 None。用于指定扇形的属性，比如边框线颜色、边

框线宽度等。例如：wedgeprops={'linewidth':5}表示设置 wedge 线宽为 5。

- textprops：字典类型，用于指定文本标签的属性，比如字体大小、字体颜色等，默认值为 None。
- center：浮点类型的列表，用于指定饼图的中心位置，默认值为(0,0)。
- frame：布尔类型，用于指定是否绘制饼图的边框，默认值为 False。如果为 True，则绘制带有表的轴框架。
- rotatelabels：布尔类型，用于指定是否旋转文本标签，默认为 False。如果为 True，则表示旋转每个 label 到指定的角度。
- data：用于指定数据。如果设置了 data 参数，则可以直接使用数据框中的列作为 x、labels 等参数的值，无须再次传递。

除此之外，pie()函数还可以返回以下三个参数。

- wedges：一个包含扇形对象的列表。
- texts：一个包含文本标签对象的列表。
- autotexts：一个包含自动生成的文本标签对象的列表。

【例 5-11】 饼图的绘制。

```
import matplotlib.pyplot as plt
import numpy as np
#显示中文
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
y = np.array([35, 25, 25, 15])
plt.subplot(1,2,1);
plt.pie(y,labels=['A','B','C','D'],                    #设置饼图标签
        colors=["#d5695d", "#5d8ca8", "#65a479", "#a564c9"],      #设置饼图颜色
        )
plt.title("带标签饼图")                                 #设置标题
plt.subplot(1,2,2)
plt.pie(y,labels=['A','B','C','D'],                    #设置饼图标签
        colors=["#d5695d", "#5d8ca8", "#65a479", "#a564c9"],      #设置饼图颜色
        explode=(0, 0.2, 0, 0),                        #第二部分突出显示,值越大,距离中心越远
        autopct='%.2f%%',                              #格式化输出百分比
        )
plt.title("第二部分突出显示")
plt.show()
```

运行程序，饼图效果如图 5-13 所示。

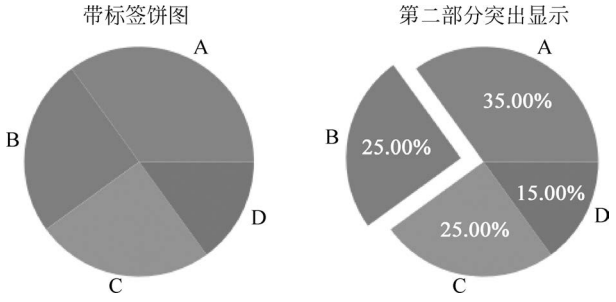

图 5-13　饼图效果

5.1.7 直方图

在 Matplotlib 中，可以使用 Pyplot 中的 hist()函数来绘制直方图。hist()函数可以用于可视化数据的分布情况，例如观察数据的中心趋势、偏态和异常值等。函数语法格式如下：

```
matplotlib.pyplot.hist(x, bins = None, range = None, density = False, weights = None, cumulative = False, bottom = None, histtype = 'bar', align = 'mid', orientation = 'vertical', rwidth = None, log = False, color = None, label = None, stacked = False, ** kwargs)
```

各参数的含义如下。

- x：绘制直方图的数据，可以是一个一维数组或列表。
- bins：可选参数，表示直方图的箱数，默认为 10。
- range：可选参数，表示直方图的值域范围，可以是一个二元组或列表，默认为 None，即使用数据中的最小值和最大值。
- density：可选参数，表示是否将直方图归一化，默认为 False，即直方图的高度为每个箱子内的样本数，而不是频率或概率密度。
- weights：可选参数，表示每个数据点的权重，默认为 None。
- cumulative：可选参数，表示是否绘制累积分布图，默认为 False。
- bottom：可选参数，表示直方图的起始高度，默认为 None。
- histtype：可选参数，表示直方图的类型，可以是'bar'、'barstacked'、'step'、'stepfilled'等，默认为'bar'。
- align：可选参数，表示直方图箱子的对齐方式，可以是'left'、'mid'、'right'，默认为'mid'。
- orientation：可选参数，表示直方图的方向，可以是'vertical'、'horizontal'，默认为'vertical'。
- rwidth：可选参数，表示每个箱子的宽度，默认为 None。
- log：可选参数，表示是否在 y 轴上使用对数刻度，默认为 False。
- color：可选参数，表示直方图的颜色。
- label：可选参数，表示直方图的标签。
- stacked：可选参数，表示是否堆叠不同的直方图，默认为 False。
- ** kwargs：可选参数，表示其他绘图参数。

【例 5-12】 绘制直方图。

```
import matplotlib.pyplot as plt
import numpy as np
#显示中文
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
#生成 3 组随机数据
data1 = np.random.normal(0, 1, 1000)
data2 = np.random.normal(2, 1, 1000)
data3 = np.random.normal(-2, 1, 1000)
# 绘制直方图，bins 参数为 30，表示将数据范围分成 30 个等宽的区间
plt.hist(data1, bins = 30, alpha = 0.5, label = 'data1')
#alpha = 0.5，表示每个直方图的颜色透明度为 50%
plt.hist(data2, bins = 30, alpha = 0.5, label = 'data2')
plt.hist(data3, bins = 30, alpha = 0.5, label = 'data3')
#设置图表属性
plt.title('直方图')
```

```
plt.xlabel('值')
plt.ylabel('频率')
plt.legend()
plt.show()      #显示图表
```

运行程序，直方图效果如图 5-14 所示。

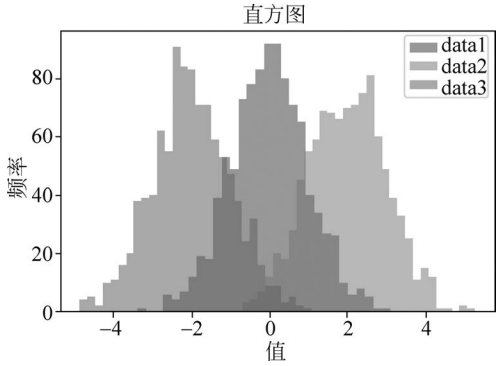

图 5-14　直方图效果

从图 5-14 可以清晰地看出这 3 组数据的分布情况，其中 data1 和 data2 分布接近正态分布，而 data3 分布偏态。这种绘制直方图的方式可以帮助我们分析和比较不同数据组的分布情况。

5.1.8　图像显示与保存

显示与保存图像是每个编程语言都有的机制，下面介绍这两种机制。

1. 显示图像

imshow()函数是 Matplotlib 库中的一个函数，用于显示图像，它常用于绘制二维的灰度图像或彩色图像，也可用于绘制矩阵、热力图、地图等。imshow()函数语法格式如下：

```
imshow(X, cmap = None, norm = None, aspect = None, interpolation = None, alpha = None, vmin = None,
vmax = None, origin = None, extent = None, shape = None, filternorm = 1, filterrad = 4.0, imlim =
None, resample = None, url = None, data = None, **kwargs)
```

各参数的含义如下。

- X：输入数据。可以是二维数组、三维数组、PIL 图像对象、Matplotlib 路径对象等。
- cmap：颜色映射。用于控制图像中不同数值所对应的颜色。可以选择内置的颜色映射，如 gray、hot、jet 等，也可以自定义颜色映射。
- norm：用于控制数值的归一化方式。可以选择 Normalize、LogNorm 等归一化方法。
- aspect：控制图像纵横比(aspect ratio)。可以设置为 auto 或一个数字。
- interpolation：插值方法。用于控制图像的平滑程度和细节程度。可以选择 nearest、bilinear、bicubic 等插值方法。
- alpha：图像透明度，取值范围为 0～1。
- origin：坐标轴原点的位置。可以设置为 upper 或 lower。
- extent：控制显示的数据范围。可以设置为[xmin，xmax，ymin，ymax]。
- vmin、vmax：控制颜色映射的值域范围。
- filternorm 和 filterrad：表示图像滤波的对象。可以设置为 None、antigrain、freetype 等。
- imlim：用于指定图像显示范围。

- resample：用于指定图像重采样方式。
- url：用于指定图像链接。

【例 5-13】 创建一个 4×4 的二维 numpy 数组，并对其进行三种不同的 imshow 图像展示。

```
import matplotlib.pyplot as plt
import numpy as np
#显示中文
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
n = 4
#创建一个 n×n 的二维 numpy 数组
a = np.reshape(np.linspace(0,1,n**2), (n,n))
plt.figure(figsize=(12,4.5))
#第一张图展示灰度的色彩映射方式，并且没有进行颜色的混合
plt.subplot(131)
plt.imshow(a, cmap='gray', interpolation='nearest')
plt.xticks(range(n))
plt.yticks(range(n))
#灰度映射，无混合
plt.title('灰度映射，无混合', y=1.02, fontsize=12)
#第二张图展示使用 Viridis 颜色映射的图像，同样没有进行颜色的混合
plt.subplot(132)
plt.imshow(a, cmap='viridis', interpolation='nearest')
plt.yticks([])
plt.xticks(range(n))
# Viridis 映射，无混合
plt.title('Viridis 映射，无混合', y=1.02, fontsize=12)
#第 3 张图展示使用 Viridis 颜色映射的图像，且使用了双立方插值方法进行颜色混合
plt.subplot(133)
plt.imshow(a, cmap='viridis', interpolation='bicubic')
plt.yticks([])
plt.xticks(range(n))
# Viridis 映射，双立方混合
plt.title('Viridis 映射，双立方混合', y=1.02, fontsize=12)
plt.show()
```

运行程序，显示图像效果如图 5-15 所示。

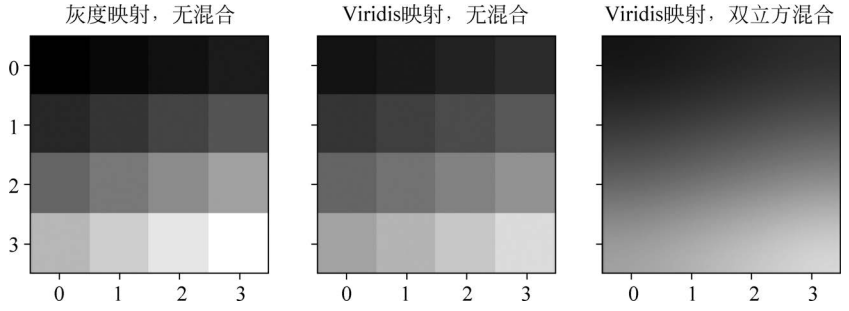

图 5-15　显示图像效果

2. 保存图像

imsave()函数是 Matplotlib 库中将图像数据保存到磁盘上的函数，通过 imsave()函数可以轻松将生成的图像保存到指定的目录中。imsave()函数支持多种图像格式，如 PNG、JPEG、BMP 等。函数的语法格式如下：

```
matplotlib.pyplot.imsave(fname, arr, **kwargs)
```

其中，参数 fname 为保存图像的文件名，可以是相对路径，也可以是绝对路径；arr 表示图像的 NumPy 数组；kwargs 为可选参数，用于指定保存的图像格式以及图像质量等参数。

【例 5-14】 使用 imsave()函数将一幅灰度图像和一幅彩色图像保存到磁盘上。

```
import matplotlib.pyplot as plt
import numpy as np
#创建一幅灰度图像
img_gray = np.random.random((100, 100))
#创建一幅彩色图像
img_color = np.zeros((100, 100, 3))
img_color[:, :, 0] = np.random.random((100, 100))
img_color[:, :, 1] = np.random.random((100, 100))
img_color[:, :, 2] = np.random.random((100, 100))
#显示灰度图像
plt.subplot(1,2,1),plt.imshow(img_gray, cmap = 'gray')
#保存灰度图像到磁盘上
plt.subplot(1,2,2),plt.imsave('test_gray.png', img_gray, cmap = 'gray')
#显示彩色图像
plt.imshow(img_color)
#保存彩色图像到磁盘上
plt.imsave('test_color.jpg', img_color)
plt.show()
```

运行程序，灰度图像与彩色图像效果如图 5-16 所示。

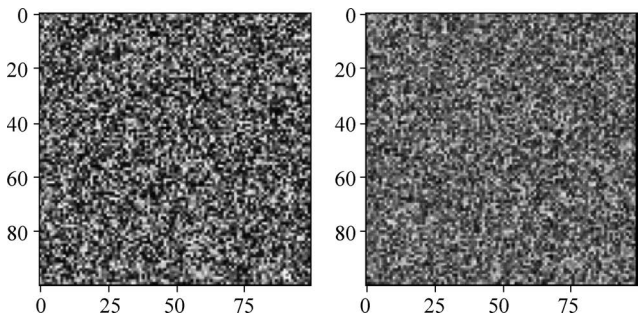

图 5-16　灰度图像与彩色图像效果

5.1.9　读取图像

imread()是 Matplotlib 库中的一个函数，用于从图像文件中读取图像数据。imread()函数返回一个 numpy.ndarray 对象，其形状是(nrows，ncols，nchannels)，表示读取的图像的行数、列数和通道数。如果图像是灰度图像，则 nchannels 为 1；如果是彩色图像，则 nchannels 为 3 或 4，为 3 时表示红、绿、蓝 3 个颜色通道，为 4 时表示红、绿、蓝 3 个通道和 1 个 alpha 通道。

imread()函数的语法如下：

```
matplotlib.pyplot.imread(fname, format = None)
```

其中，参数 fname 指定要读取的图像文件的文件名或文件路径，可以是相对路径，也可以是绝对路径；format 参数指定图像文件的格式，如果不指定，则默认根据文件后缀名来自动识别格式。

【例 5-15】 利用 imread()读取图像，并将数组乘以一个 0～1 的数，将图像变暗。

```
import matplotlib.pyplot as plt
#读取图像文件，下载地址：https://static.jyshare.com/images/mix/tiger.jpeg
img_array = plt.imread('tiger.jpeg')
tiger = img_array/255
#显示图像
plt.figure(figsize=(10,6))
for i in range(1,5):
    plt.subplot(2,2,i)
    x = 1 - 0.2*(i-1)
    plt.axis('off')      #隐藏坐标轴
    plt.title('x={:.1f}'.format(x))
    plt.imshow(tiger*x)
plt.show()
```

运行程序，读取图像并显示的效果如图 5-17 所示。

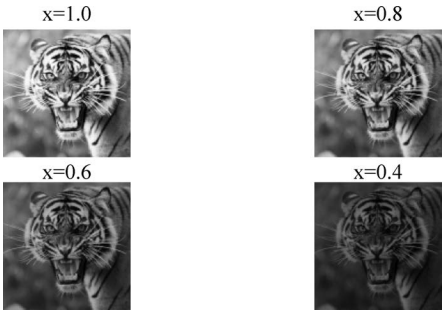

图 5-17 读取图像并显示的效果

5.2 海龟绘图

海龟绘图(Turtle Graphics)是 Python 语言内置的绘图模块，是早期的 LOGO 编程语言在 Python 语言中的实现。使用这个模块绘图时，可以把屏幕当成一块画布，通过控制一个小三角形(或小海龟)画笔在画布上移动，在它前进的路径上绘制出图形。

turtle 模块提供了用于绘图的函数，在使用前要先导入 turtle 模块：

```
import turtle
```

5.2.1 turtle 绘图的基础知识

画布是 turtle 用于绘图的区域，可以设置它的大小和初始位置。

1. 设置画布大小

设置画布大小可以用以下函数。

turtle.screensize(canvwidth=None,canvheight=None,bg=None)：参数分别对应画布的宽(单位像素)、高、背景颜色。

turtle.screensize(1000,800,"blue")。

turtle.screensize() #返回默认大小(500,400)。

turtle.setup(width=0.5,height=0.75,startx=None,starty=None)。

当 width 和 height 输入宽和高为整数时，表示像素；为小数时，表示占据计算机屏幕的比例。(startx,starty)这一坐标表示矩形窗口左上角顶点的位置，如果为空，则窗口位于屏幕

中心。

2. 画笔的状态

在画布上，默认有一个坐标原点为画布中心的坐标轴，坐标原点上有一只面朝 x 轴正方向小乌龟。这里描述小乌龟时使用了坐标原点(位置)和面朝 x 轴正方向(方向)。turtle 绘图中，就是使用位置和方向描述小乌龟(画笔)的状态。

```
turtle.pensize()          #设置画笔的宽度
turtle.pencolor()         #没有参数传入，返回当前画笔颜色，传入参数设置画笔颜色，可以是字符串，如"green"、"red"，也可以是 RGB 的三元组
turtle.speed(speed)       #设置画笔移动速度，画笔绘制的速度范围是 0～10 的整数，数字越大，速度越快
```

3. 使用 turtle 绘图

使用 turtle 绘图主要有 3 种命令代码，分别为画笔运动命令、画笔控制命令和全局控制命令。

1) 画笔运动命令

画笔运动命令的主要包含的函数如下。

turtle.forward(distance)：向当前画笔方向移动 distance 像素长度。

turtle.backward(distance)：向当前画笔相反方向移动 distance 像素长度。

turtle.right/left(degree)：顺时针/逆时针移动 degree 度(°)。

turtle.pendown()：移动时绘制图形。

turtle.goto(x,y)：将画笔移动到坐标为(x,y)的位置。

turtle.penup()：提起笔移动，不绘制图形，用于另起一个地方绘制。

turtle.circle()：画圆，半径为正(负)，表示圆心在画笔的左边(右边)画圆。

setx( )：将当前 x 轴移动到指定位置。

sety( )：将当前 y 轴移动到指定位置。

setheading(angle)：设置当前朝向为 angle 角度。

home()：设置当前画笔位置为原点，朝向东。

dot：绘制一个指定直径和颜色的圆点。

2) 画笔控制命令

画笔控制命令主要包含的函数如下。

turtle.fillcolor(colorstring)：绘制图形的填充颜色。

turtle.color(color1,color2)：设置 pencolor=color1，fillcolor=color2。

turtle.filling()：返回当前是否在填充状态，turtle.begin_fill()表示准备开始填充图形。

turtle.end_fill()：填充完成。

turtle.hideturtle()：隐藏画笔的 turtle 形状。

turtle.showturtle()：显示画笔的 turtle 形状。

3) 全局控制命令

全局控制命令主要包含的函数如下。

turtle.clear()：清空 turtle 窗口，但是 turtle 的位置和状态不会改变。

turtle.reset()：清空窗口，重置 turtle 状态为起始状态。

turtle.undo()：撤销上一个 turtle 动作。

turtle.isvisible()：返回当前 turtle 是否可见。

stamp()：复制当前图形。

turtle.write(s[,font=("fontname",font_size,"font_type")])：s 为文本内容；font 是字体的参数，分别为字体名称、大小和类型，font 为可选项，font 参数也是可选项。

5.2.2 基本绘图

让海龟前进 90 步：

```
import turtle as tur
tur.forward(90)
```

在显示中新建一个窗口，向东绘制一条线段，如图 5-18 所示，实现海龟画出一条线段，方向朝东。改变海龟的方向，让它向左转 120°(逆时针)：

```
tur.left(120) # 向左旋转 120°效果如图 5-19 所示
```

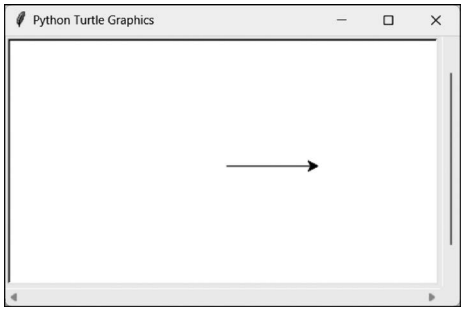

图 5-18 向东绘制一条线段

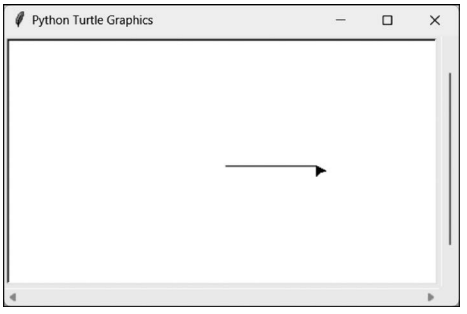

图 5-19 向左旋转 120°效果

继续画一个三角形：

```
tur.forward(90)
tur.left(120)
tur.forward(90)
```

绘制的三角形效果如图 5-20 所示。

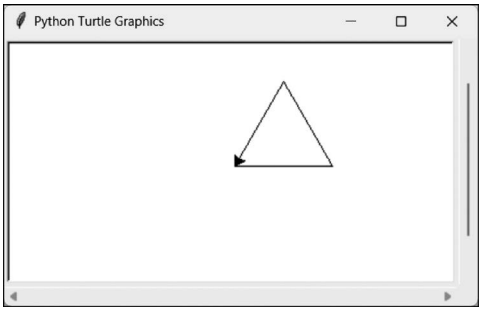

图 5-20 绘制的三角形效果

需要注意的是，图 5-20 中的箭头表示的是海龟是如何随着操纵指向不同方向的。可继续尝试这些命令，还可以使用 backward()和 right()函数进行三角形绘制。

5.2.3 使用算法绘制图案

在海龟绘图中，也可以使用循环构建各种几何图案。例如：

```
import turtle as tur
for steps in range(100):
    for c in ('blue', 'red', 'green'):
```

```
    tur.color(c)
    tur.forward(steps)
    tur.right(30)
```

运行程序，画一个十二边形的效果如图 5-21 所示。

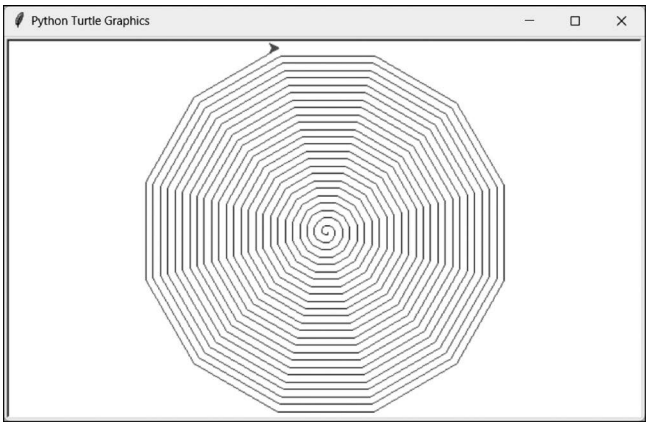

图 5-21　画一个十二边形的效果

如果想要用红色线条绘制一个星形，并用黄色填充（星形效果如图 5-22 所示），实现代码为：

```
import turtle as tur
#设置线条为红色,黄色填充
tur.color('red')
tur.fillcolor('yellow')
tur.begin_fill() #打开填充
#创建一个循环体
while True:
    tur.forward(200)
    tur.left(170)
    if abs(tur.pos()) < 1:          #确定海龟何时回到初始点
        break                       #跳出循环
tur.end_fill()                      #完成填充
```

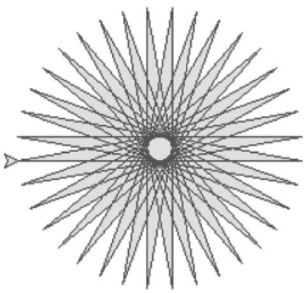

图 5-22　星形效果

注意，只有在给出 end_fill()命令时才会进行填充。

5.2.4　使用 turtle 模块命令空间

海龟模块将其所有基本功能作为函数公开，并通过 from turtle import * 使这一切成为可能。使用 from turtle import * 虽然很方便，但要注意它导入的对象集相当大，如果同时在做海龟绘图以外的事就有可能发生名称冲突。

解决办法是使用 import turtle，例如，fd()将变成 turtle.fd()，width()将变成 turtle.width()等，如果觉得反复输入 turtle 太烦琐，还可以改成 import turtle as tur 等。

例如，使用 turtle 模块命名空间的代码如下：

```
import turtle as t
from random import random

for i in range(100):
    steps = int(random() * 100)
    angle = int(random() * 360)
    t.right(angle)
    t.fd(steps)
```

此代码并不十分完善，因为一旦脚本结束，Python 将会同时关闭海龟的窗口。如果想在结束脚本时不关闭海龟的窗口，可添加 t.mainloop()到脚本的末尾，表示不会自动退出而需要人为终止。

5.2.5 使用面向对象的海龟绘图

使用面向对象的方式进行海龟绘图十分常见，功能也很强大。例如，这种方式允许屏幕上同时存在多只海龟。

在这种方式下，各种海龟命令都是对象(主要是 turtle 对象)的方法。可以在 shell 中使用面向对象的方法，但在 Python 脚本中使用的是更为典型的做法。

例如，可使用如下脚本实现 5.2.4 节的实例：

```
from turtle import Turtle
from random import random
t = Turtle()
for i in range(100):
    steps = int(random() * 100)
    angle = int(random() * 360)
    t.right(angle)
    t.fd(steps)
t.mainloop()
```

5.2.6 绘制任意多边形

多边形的内角公式为内角$=(n-2)\times180/n$，其中 n 是多边形的边数。多边形的所有内角之和等于 $180\times(n-2)$，其中 n 是多边形的边数。

以正五边形为例，将 n 设置为 5，代入公式得到$(5-2)\times180/5=3\times180/5=108°$，因此，正五边形的每个内角为 108°。

【例 5-16】 绘制 3 个正五边形。

```
from turtle import *
import time

def draw5(x, y):
    pu()
    goto(x, y)
    pd()
    # set heading: 0
    seth(0)
    for i in range(3):
```

```
        fd(40)
        rt(72)
        time.sleep(1)
for x in range(0, 250, 50):
    draw5(x, 0)
done()
```

【例 5-17】 绘制一个五角星。

解析：根据多边形的内角公式可知，如果要绘制五角星，关键是得知道每次转向多少角度。五角星是 144°。为什么是 144°？这是因为正五边形的内角是 108°，因此它的补角是 72°，五角星的每个角是 180°－72°－72°＝36°，因此每次转向 180°－36°＝144°。

```
#绘制一个五角星
from turtle import *
import time
def drawStar(x, y):
    pu()
    goto(x, y)
    pd()
    #设定航向：0
    seth(0)
    for i in range(5):
        fd(110)
        rt(144)
        time.sleep(1)
drawStar(0,0)
done()
```

利用 turtle 还可以绘制一些更漂亮、复杂的图形。

【例 5-18】 绘制一棵树。

```
from turtle import Turtle, mainloop
from time import perf_counter as clock
def tree(plist, l, a, f):
    """ plist 是笔的列表
    l 是分支的长度
    a 是两个分支之间的角度的一半
    f 是分支
    从一层到另一层缩短的因子"""
    if l > 3:
        lst = []
        for p in plist:
            p.forward(l)
            q = p.clone()
            p.left(a)
            q.right(a)
            lst.append(p)
            lst.append(q)
        for x in tree(lst, l*f, a, f):
            yield None
def maketree():
    p = Turtle()
    p.setundobuffer(None)
    p.hideturtle()
    p.speed(0)
    p.getscreen().tracer(30,0)
    p.left(90)
```

```
    p.penup()
    p.forward(-210)
    p.pendown()
    t = tree([p], 200, 65, 0.6375)
    for x in t:
        pass
def main():
    a = clock()
    maketree()
    b = clock()
    return "done: %.2f sec." % (b-a)
if __name__ == "__main__":
    msg = main()
    print(msg)
    mainloop()
```

运行程序,绘制一棵树的效果如图 5-23 所示。

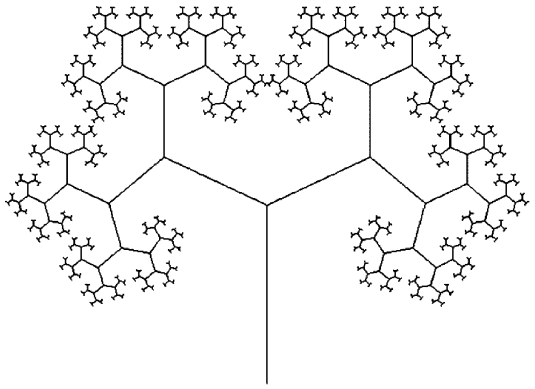

图 5-23　绘制一棵树的效果

5.3　练习

1. 根据二维散点图,试绘制三维散点图。
2. 试用 line 画直线。
3. 参照例 5-16 及例 5-17,尝试绘制一个六角星。
4. 发挥想象,画一幅漂亮的几何拼贴图。