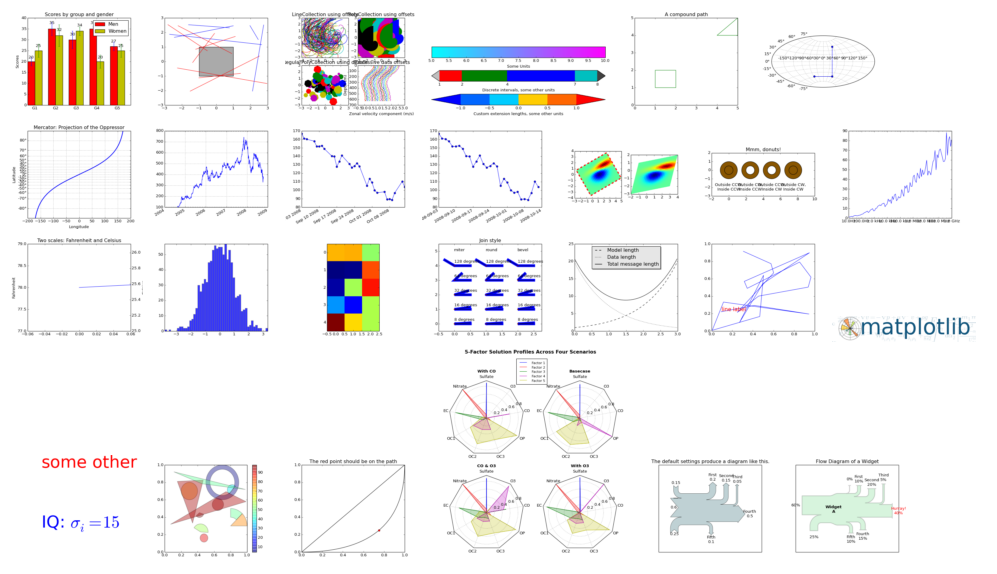
# [绘图: Python matplotlib简介](http://www.cnblogs.com/vamei/archive/2012/09/17/2689798.html)

作者：Vamei 出处：http://www.cnblogs.com/vamei 欢迎转载，也请保留这段声明。谢谢！

[matplotlib](http://matplotlib.org/index.html)是基于numpy的一套Python工具包。这个包提供了丰富的数据绘图工具，主要用于绘制一些统计图形。你可以找到很多各式各样的[例子](http://matplotlib.org/gallery)：



通过数据绘图，我们可以将枯燥的数字转换成容易被人们接受的图表，从而让人留下更加深刻的印象。实际上，早在一百多年前，[南丁格尔](http://zh.wikipedia.org/wiki/%E5%8D%97%E4%B8%81%E6%A0%BC%E5%B0%94)就曾经用统计图形来说服英国政府，以改善军队的卫生状况。

我们将以GDP数据为例子，看看如何绘制经典的饼图和条形图。

### 数据

下面是我们要使用的数据，为2011年GDP前十的国家以及其具体的GDP：

[复制代码](javascript:void(0);)

USA 15094025

China 11299967

India 4457784

Japan 4440376

Germany 3099080

Russia 2383402

Brazil 2293954

UK 2260803

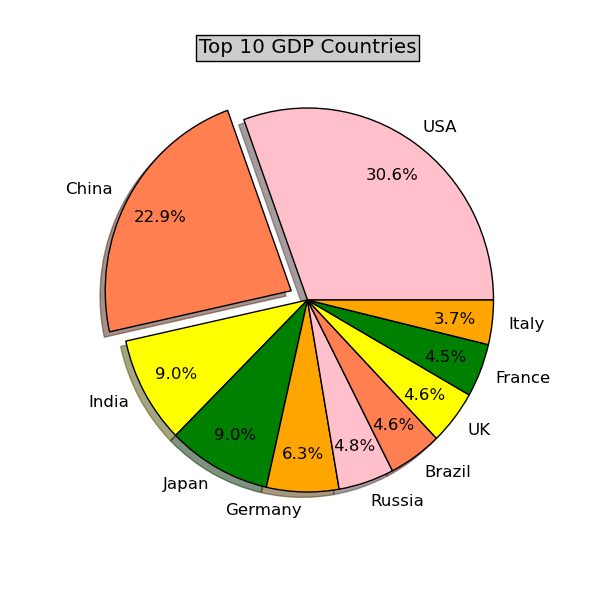
France 2217900

Italy       1846950

[复制代码](javascript:void(0);)

### 饼图

我们先来绘制饼图 (pie plot)。饼图适用于表达各个国家GDP所占的百分比。每一小块的面积代表了占比的多少：



具体代码如下，可以看到我们主要使用了matplotlib.pyplot工具包：

[复制代码](javascript:void(0);)

# Make a pie chart

# This script is written by Vamei, http://www.cnblogs.com/vamei

# you may freely use it.

import matplotlib.pyplot as plt

# quants: GDP

# labels: country name

labels = []

quants = []

# Read data

for line in file('../data/major\_country\_gdp'):

info = line.split()

labels.append(info[0])

quants.append(float(info[1]))

# make a square figure

plt.figure(1, figsize=(6,6))

# For China, make the piece explode a bit

def explode(label, target='China'):

if label == target: return 0.1

else: return 0

expl = map(explode,labels)

# Colors used. Recycle if not enough.

colors = ["pink","coral","yellow","orange"]

# Pie Plot

# autopct: format of "percent" string;

plt.pie(quants, explode=expl, colors=colors, labels=labels, autopct='%1.1f%%',pctdistance=0.8, shadow=True)

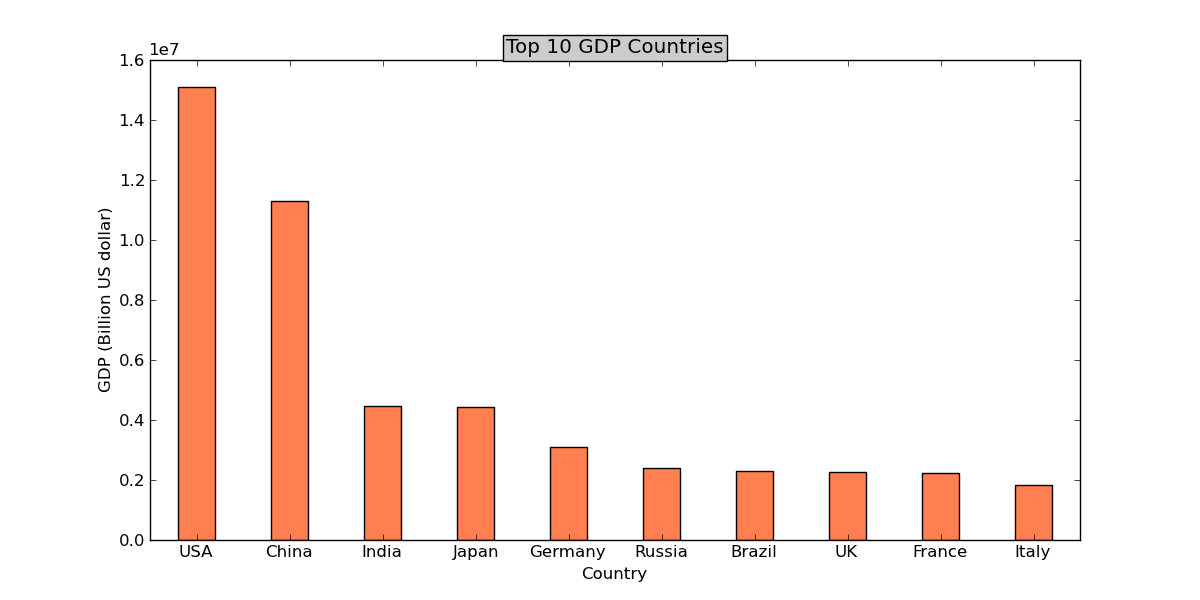
plt.title('Top 10 GDP Countries', bbox={'facecolor':'0.8', 'pad':5})

plt.show()

[复制代码](javascript:void(0);)

### 条形图

下面我们尝试一下条形图(bar plot)。用每个长条的高度代表每个国家的GDP，长条越高，GDP值越高：



代码如下：

[复制代码](javascript:void(0);)

"""

Make a pie chart

This script is written by Vamei, http://www.cnblogs.com/vamei

you may freely use it.

"""

import matplotlib.pyplot as plt

import numpy as np

# quants: GDP

# labels: country name

labels = []

quants = []

# Read data

for line in file('../data/major\_country\_gdp'):

info = line.split()

labels.append(info[0])

quants.append(float(info[1]))

width = 0.4

ind = np.linspace(0.5,9.5,10)

# make a square figure

fig = plt.figure(1, figsize=(12,6))

ax = fig.add\_subplot(111)

# Bar Plot

ax.bar(ind-width/2,quants,width,color='coral')

# Set the ticks on x-axis

ax.set\_xticks(ind)

ax.set\_xticklabels(labels)

# labels

ax.set\_xlabel('Country')

ax.set\_ylabel('GDP (Billion US dollar)')

# title

ax.set\_title('Top 10 GDP Countries', bbox={'facecolor':'0.8', 'pad':5})

plt.show()

[复制代码](javascript:void(0);)

该代码中我们利用了ax对象，以便控制刻度以及刻度所对应的国家名。这与我们在pie plot所做的有些不同(pie plot也可以这样实现，只是没有必要而已)。

从两个图上看，亚洲国家的GDP还是很厉害的。西方的话就是美国一枝独秀了。

### 总结

我们演示了饼图和条性图的绘制方法。matplotlib是一款功能强大的数据绘图工具，非常值得学习。

# [绘图: matplotlib核心剖析](http://www.cnblogs.com/vamei/archive/2013/01/30/2879700.html)

作者：Vamei 出处：http://www.cnblogs.com/vamei 欢迎转载，也请保留这段声明。谢谢！

matplotlib是基于Python语言的开源项目，旨在为Python提供一个数据绘图包。我将在这篇文章中介绍matplotlib API的核心对象，并介绍如何使用这些对象来实现绘图。实际上，matplotlib的对象体系严谨而有趣，为使用者提供了巨大的发挥空间。用户在熟悉了核心对象之后，可以轻易的定制图像。matplotlib的对象体系也是计算机图形学的一个优秀范例。即使你不是Python程序员，你也可以从文中了解一些通用的图形绘制原则。

matplotlib使用numpy进行数组运算，并调用一系列其他的Python库来实现硬件交互。matplotlib的核心是一套由对象构成的绘图API。



matplotlib项目是由John D. Hunter发起的。John D. Hunter由于癌症于去年过世，但他发为社区作出的无比贡献将永远留存。



John D. Hunter

你需要安装Python, numpy和matplotlib。(可以到python.org下载Python编译器。相关Python包的安装，请参看[我的Python小技巧](http://www.cnblogs.com/vamei/archive/2012/11/06/2755503.html))

matplotlib的官网是： <http://matplotlib.org/>  官网有丰富的图例和文档说明。

matplotlib在github的地址为：<https://github.com/matplotlib> 欢迎有兴趣的开发者fork。

### 函数式绘图

matplotlib是受MATLAB的启发构建的。MATLAB是数据绘图领域广泛使用的语言和工具。MATLAB语言是面向过程的。利用函数的调用，MATLAB中可以轻松的利用一行命令来绘制直线，然后再用一系列的函数调整结果。

matplotlib有一套完全仿照MATLAB的函数形式的绘图接口，在matplotlib.pyplot模块中。这套函数接口方便MATLAB用户过度到matplotlib包。下面，我们调用该模块绘制一条直线。

[复制代码](javascript:void(0);)

# a strait line: use pyplot functions  
  
from matplotlib.pyplot import \*

plot([0, 1], [0, 1]) # plot a line from (0, 0) to (1, 1)

title("a strait line")

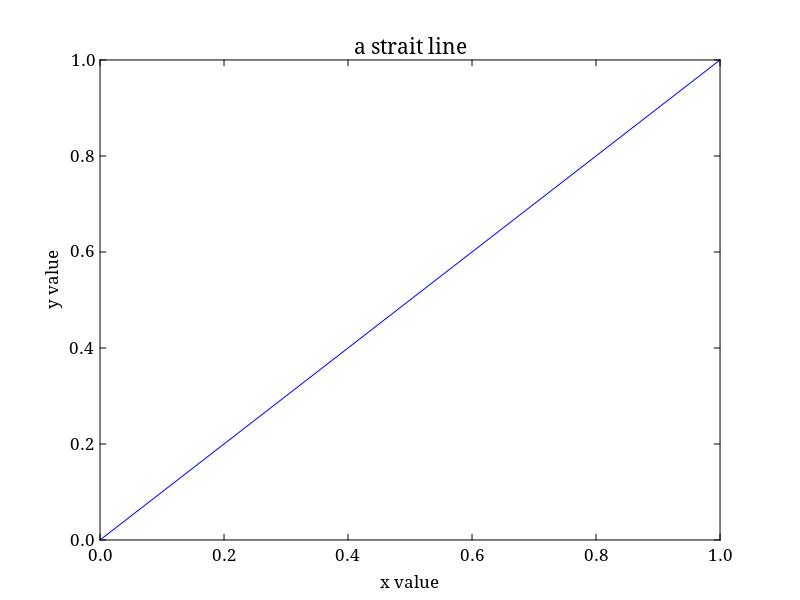
xlabel("x value")

ylabel("y value")

savefig("demo.jpg")

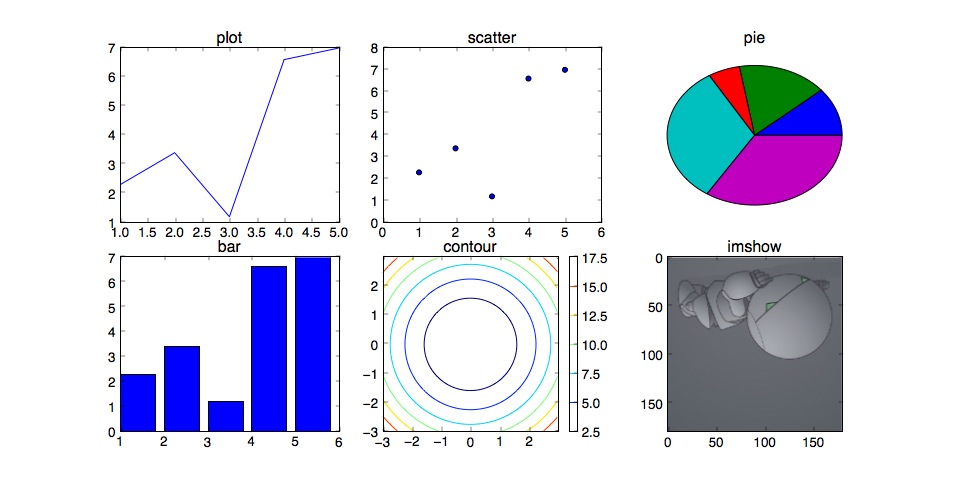
[复制代码](javascript:void(0);)

上面的每一条命令都很简单，你可以从函数名读出该函数所要实现的功能。比如plot为画线，title为增加标题。最终保存的demo.jpg如下：



上面的函数式调用很方便。在[Python特殊方法与多范式](http://www.cnblogs.com/vamei/archive/2012/11/19/2772441.html)中，我们已经谈到，Python中的函数式编程是通过封装对象实现的。matplotlib中的函数式调用其实也是如此。matplotlib本质上还是构建对象来构建图像。函数式编程将构建对象的过程封装在函数中，从而让我们觉得很方便。

在matplotlib.pyplot中，你还可以找到下面的绘图函数。如果你经常使用数据绘图程序，应该会很熟悉这些图形：



绘图程序如下：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

import matplotlib.pyplot as plt

# 1D data

x = [1,2,3,4,5]

y = [2.3,3.4,1.2,6.6,7.0]

plt.figure(figsize=(12,6))

plt.subplot(231)

plt.plot(x,y)

plt.title("plot")

plt.subplot(232)

plt.scatter(x, y)

plt.title("scatter")

plt.subplot(233)

plt.pie(y)

plt.title("pie")

plt.subplot(234)

plt.bar(x, y)

plt.title("bar")

# 2D data

import numpy as np

delta = 0.025

x = y = np.arange(-3.0, 3.0, delta)

X, Y = np.meshgrid(x, y)

Z = Y\*\*2 + X\*\*2

plt.subplot(235)

plt.contour(X,Y,Z)

plt.colorbar()

plt.title("contour")

# read image

import matplotlib.image as mpimg

img=mpimg.imread('marvin.jpg')

plt.subplot(236)

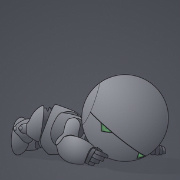
plt.imshow(img)

plt.title("imshow")

plt.savefig("matplot\_sample.jpg")

[复制代码](javascript:void(0);)

上面用到的marvin.jpg是下图，请保存到当地电脑：



函数式编程创造了一个仿真MATLAB的工作环境，并有许多成形的绘图函数。如果只是作为Matplotlib的一般用户(非开发者)，pyplot可以满足大部分的需求。

(当然，matplotlib是免费而开源的，MATLAB昂贵而封闭。这是不“仿真”的地方)

### 面向对象编程

尽管函数式绘图很便利，但利用函数式编程会有以下缺点：

1) 增加了一层“函数”调用，降低了效率。

2) 隶属关系被函数掩盖。整个matplotlib包是由一系列有组织有隶属关系的对象构成的。函数掩盖了原有的隶属关系，将事情变得复杂。

3) 细节被函数掩盖。pyplot并不能完全复制对象体系的所有功能，图像的许多细节调中最终还要回到对象。

4) 每件事情都可以有至少两种方式完成，用户很容易混淆。

而对于开发者来说，了解对象是参与到Matplotlib项目的第一步。

我们将上面的直线绘图更改为面向对象式(OO, object-oriented)的，为此，我们引入两个类: Figure和FigureCanvas。(函数式编程也调用了这些类，只是调用的过程被函数调用所遮掩。)

[复制代码](javascript:void(0);)

# object-oriented plot  
  
from matplotlib.figure import Figure

from matplotlib.backends.backend\_agg import FigureCanvasAgg as FigureCanvas

fig = Figure()

canvas = FigureCanvas(fig)

ax = fig.add\_axes([0.1, 0.1, 0.8, 0.8])

line, = ax.plot([0,1], [0,1])

ax.set\_title("a straight line (OO)")

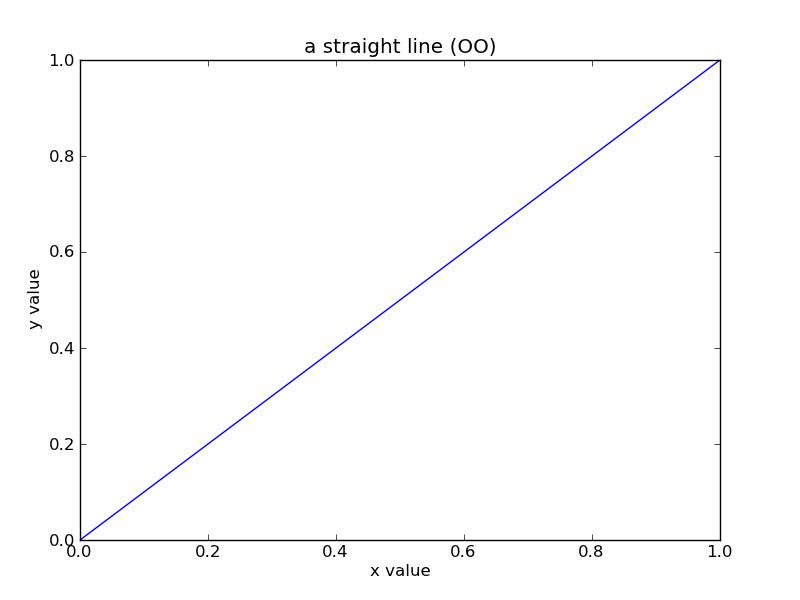
ax.set\_xlabel("x value")

ax.set\_ylabel("y value")

canvas.print\_figure('demo.jpg')

[复制代码](javascript:void(0);)

新的demo.jpg如下：

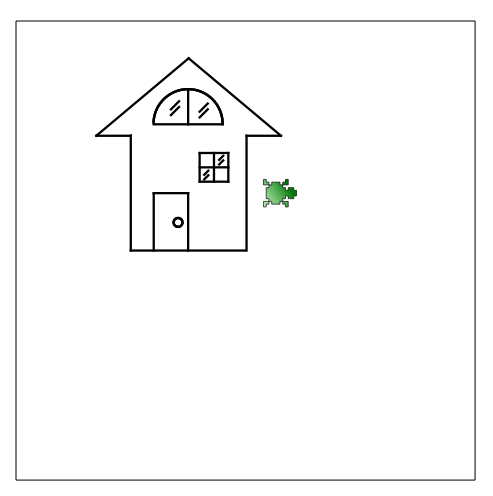


### 

### 理解对象

上面的例子中，我们至少构建了四个对象： fig, canvas, ax, line。它们分别属于Figure类，FigureCanvas类，Axes类和Line2D类。(使用obj.\_\_class\_\_.\_\_name\_\_来查询对象所属的类)

在深入各个对象之前，我们先来做一个比喻。看下面一个图片：

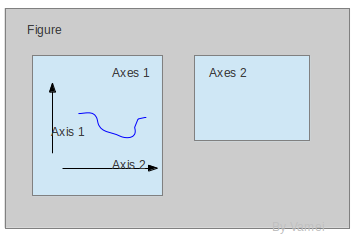


这个图片是用KTurtle绘制。参看[把你的孩子打造成为码农](http://www.cnblogs.com/vamei/archive/2012/09/15/2686966.html)

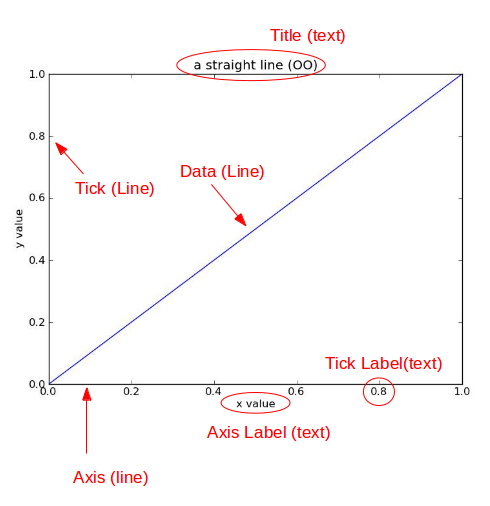
可以看到，图中有一个房子，房子上有窗户和门，窗户上有条纹，门上有把手，此外图像外还有一只小乌龟。我们所提到的房子，窗户，门，条纹，把手，都可以称其为对象。不同的对象之间有依附关系，比如窗户和门属于房子，而把手属于门。乌龟和房子则是并行的两个对象。此外，整个图像外有一个方框，用来表明可绘图的范围，所有上面提到的元素都依附于该方框。

这就是用面向对象的方式来理解一个图像。事实上，对象是描述图像的最自然的方式，面向对象编程最成功的领域就是在计算机图形方面。

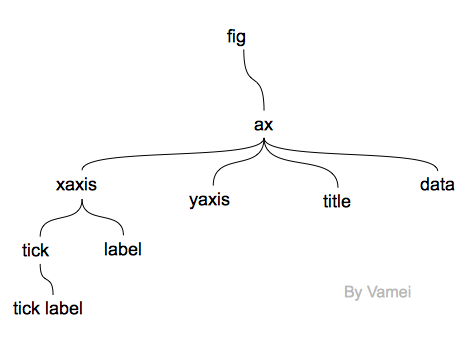
我们先来看什么是Figure和Axes对象。在matplotlib中，整个图像为一个Figure对象。在Figure对象中可以包含一个，或者多个Axes对象。每个Axes对象都是一个拥有自己坐标系统的绘图区域。其逻辑关系如下：



转过头来看直线图。整个图像是fig对象。我们的绘图中只有一个坐标系区域，也就是ax。此外还有以下对象。(括号中表示对象的基本类型)



Title为标题。Axis为坐标轴，Label为坐标轴标注。Tick为刻度线，Tick Label为刻度注释。各个对象之间有下面的对象隶属关系：



(yaxis同样有tick, label和tick label，没有画出)

尽管data是数据绘图的关键部分，也就是数据本身的图形化显示，但是必须和xaxis, yaxis, title一起，才能真正构成一个绘图区域axes。一个单纯的，无法读出刻度的线是没有意义的。xaxis, yaxis, title合起来构成了数据的辅助部分(data guide)。

上面元素又包含有多种图形元素。比如说，我们的data对象是一条线(Line2D)。title, tick label和label都是文本(Text)，而tick是由短线(Line 2D)和tick label构成，xaxis由坐标轴的线和tick以及label构成，ax由xaxis, yaxis, title, data构成，ax自身又构成了fig的一部分。上面的每个对象，无论是Line2D, Text还是fig，它们都来自于一个叫做Artist的基类。

OO绘图的原程序还有一个canvas对象。它代表了真正进行绘图的后端(backend)。Artist只是在程序逻辑上的绘图，它必须连接后端绘图程序才能真正在屏幕上绘制出来(或者保存为文件)。我们可以将canvas理解为绘图的物理(或者说硬件)实现。

在OO绘图程序中，我们并没有真正看到title, tick, tick label, xaxis, yaxis对象，而是使用ax.set\_\*的方法间接设置了这些对象。但这些对象是真实存在的，你可以从上层对象中找到其“真身”。比如，fig.axes[0].xaxis就是我们上面途中的xaxis对象。我们可以通过fig -> axes[0] (也就是ax) -> xaxis的顺序找到它。因此，重复我们刚才已经说过的，一个fig就构成了一个完整的图像。对于每个Artist类的对象，都有findobj()方法，来显示该对象所包含的所有下层对象。

## 坐标

坐标是计算机绘图的基础。计算机屏幕是由一个个像素点构成的。想要在屏幕上显示图像，计算机必须告诉屏幕每个像素点上显示什么。所以，最贴近硬件的坐标体系是以像素为单位的坐标体系。我们可以通过具体说明像素位置来标明显示器上的某一点。这叫做显示坐标(display coordinate)，以像素为单位。

然而，像素坐标不容易被纳入绘图逻辑。相同的程序，在不同的显示器上就要调整像素值，以保证图像不变形。所以一般情况下，还会有图像坐标和数据坐标。

图像坐标将一张图的左下角视为原点，将图像的x方向和y方向总长度都看做1。x方向的0.2就是指20%的图像在x方向的总长，y方向0.8的长度指80%的y方向总长。(0.5, 0.5)是图像的中点，(1, 1)指图像的右上角。比如下面的程序，我们在使用add\_axes时，传递的参数中，前两个元素为axes的左下角在fig的图像坐标上的位置，后两个元素指axes在fig的图像坐标上x方向和y方向的长度。fig的图像坐标称为Figure坐标，储存在为fig.transFigure

(类似的，每个axes，比如ax1，有属于自己的图像坐标。它以ax1绘图区域总长作为1，称为Axes坐标。也就是ax1.transAxes。(0.5, 0.5)就表示在Axes的中心。Axes坐标和Figure坐标原理相似，只是所用的基准区域不同。)

[复制代码](javascript:void(0);)

# object-oriented plot

from matplotlib.figure import Figure

from matplotlib.backends.backend\_agg import FigureCanvasAgg as FigureCanvas

fig = Figure()

canvas = FigureCanvas(fig)

# first axes

ax1 = fig.add\_axes([0.1, 0.1, 0.2, 0.2])

line, = ax1.plot([0,1], [0,1])

ax1.set\_title("ax1")

# second axes

ax2 = fig.add\_axes([0.4, 0.3, 0.4, 0.5])

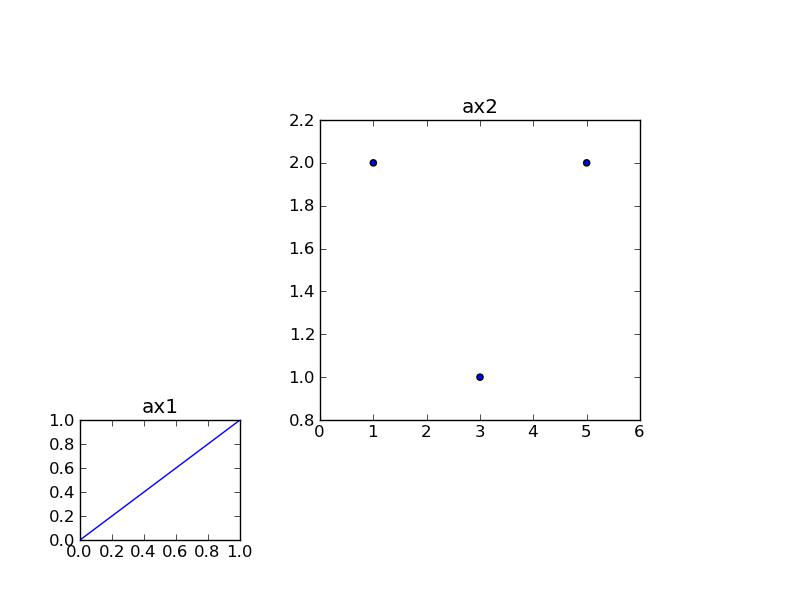
sca = ax2.scatter([1,3,5],[2,1,2])

ax2.set\_title("ax2")

canvas.print\_figure('demo.jpg')

[复制代码](javascript:void(0);)

我们在绘图，比如使用plot的时候，绘制了两点间的连线。这两点分别为(0, 0)和(1, 1)。(plot中的第一个表为两个x坐标，第二个表为两个y坐标)。这时使用的坐标系为数据坐标系(ax1.transData)。我们可以通过绘出的坐标轴读出数据坐标的位置。



如果绘制的是具体数据，那么数据坐标符合我们的需求。如果绘制的是标题这样的附加信息，那么Axes坐标符合符合我们的需求。如果是整个图像的注解，那么Figure坐标更符合需求。每一个Artist对象都有一个transform属性，用于查询和改变所使用的坐标系统。如果为显示坐标，transform属性为None。

### 深入基础

在上面的例子中，无论是使用plot绘制线，还是scatter绘制散点，它们依然是比较成熟的函数。matplotlib实际上提供了更大的自由度，允许用户以更基础的方式来绘制图形，比如下面，我们绘制一个五边形。

[复制代码](javascript:void(0);)

# object-oriented plot

from matplotlib.figure import Figure

from matplotlib.backends.backend\_agg import FigureCanvasAgg as FigureCanvas

fig = Figure()

canvas = FigureCanvas(fig)

ax = fig.add\_axes([0.1, 0.1, 0.8, 0.8])

from matplotlib.path import Path

import matplotlib.patches as patches

verts = [

(0., 0.),

(0., 1.),

(0.5, 1.5),

(1., 1.),

(1., 0.),

(0., 0.),

]

codes = [Path.MOVETO,

Path.LINETO,

Path.LINETO,

Path.LINETO,

Path.LINETO,

Path.CLOSEPOLY,

]

path = Path(verts, codes)

patch = patches.PathPatch(path, facecolor='coral')

ax.add\_patch(patch)

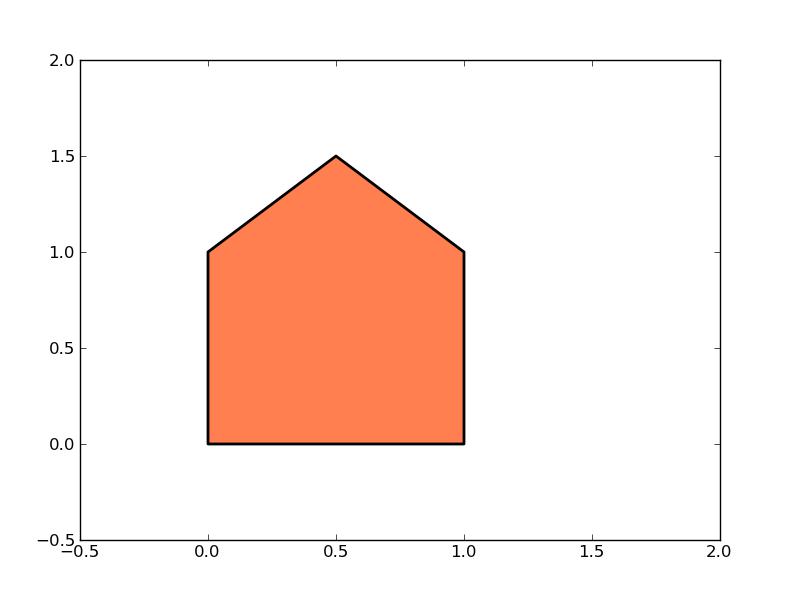
ax.set\_xlim(-0.5,2)

ax.set\_ylim(-0.5,2)

canvas.print\_figure('demo.jpg')

[复制代码](javascript:void(0);)

在上面的程序中。我们首先确定顶点，然后构建了一个path对象，这个对象实际上就是5个顶点的连线。在codes中，我们先使用MOVETO将画笔移动到起点，然后依次用直线连接(LINETO)(我们也可以用曲线来连线，比如CURVE4，但这里没有用到)。 在path建立了封闭的5边形后，我们在path的基础上构建了patch对象，是一个图形块。patch的背景颜色选为coral。最后，我们将这个patch对象添加到预先准备好的ax上，就完成了整个绘图。



上面的过程中，我们就好像拿着一个画笔的小孩，一步步画出心目中的图画。这就是深入理解matplotlib的魅力所在——创造你自己的数据绘图函数！

(将上面的程序封装到函数中，保留顶点以及其它参数接口，就构成了一个五边形绘图函数。O(∩\_∩)O~ 我们也创造了新的“一键绘图”)

可以相像，一个plot函数如何用path对象实现。

### 总结

我们已经了解了matplotlib的最重要的方面，它们是：

1) pyplot函数绘图借口

2) 对象如何组合成为图像

3) 坐标系统

希望我的讲解没有消耗完你对matplotlib的兴趣。事实上，matplotlib是发展相当迅猛的绘图包，而它的开放性也让它成为了解计算机图形学的一个好接口。利用开放的核心对象，你可以随心的定制自己的数据绘图，而不用受制于高层的调用函数。谢谢John D. Hunter。