Matplotlib

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。

plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签

plt.rcParams['axes.unicode\_minus']=False #用来正常显示负号

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

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

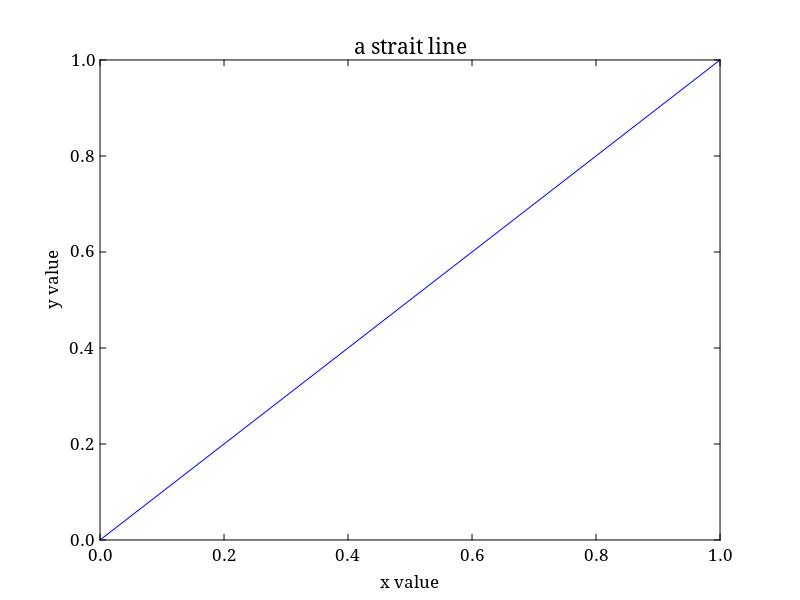
title("a strait line")

xlabel("x value")

ylabel("y value")

savefig("demo.jpg")

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：



### 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。

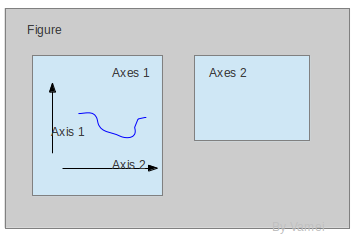
4) 每件事情都可以有至少两种方式完成，用户很容易混淆。

而对于开发者来说，了解对象是参与到Matplotlib项目的第一步。

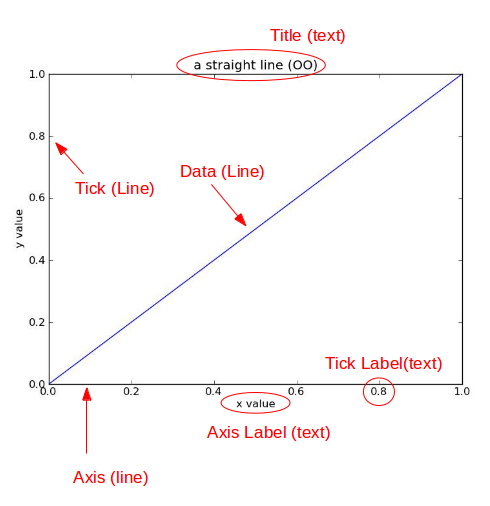
我们将上面的直线绘图更改为面向对象式(OO, object-oriented)的，为此，我们引入两个类: Figure和FigureCanvas。(函数式编程也调用了这些类，只是调用的过程被函数调用所遮掩。)

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)。

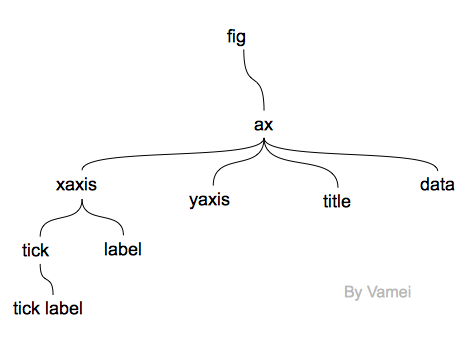
我们先来看什么是Figure和Axes对象。在matplotlib中，整个图像为一个Figure对象。在Figure对象中可以包含一个，或者多个Axes对象。每个Axes对象都是一个拥有自己坐标系统的绘图区域。其逻辑关系如下：



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。

### 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。)

# 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

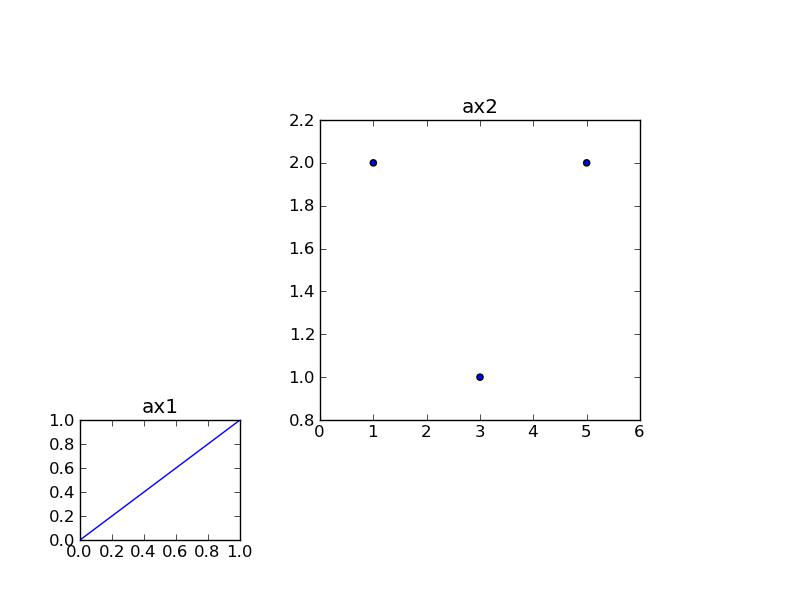
ax2 = fig.add\_axes([0.4, 0.3, 0.4, 0.5])

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

ax2.set\_title("ax2")

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

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。

# 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')

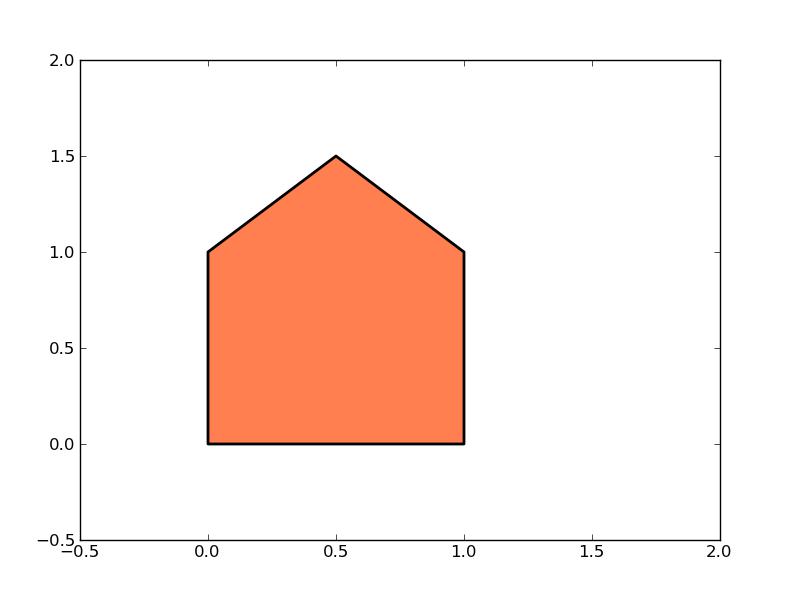
ax.add\_patch(patch)

ax.set\_xlim(-0.5,2)

ax.set\_ylim(-0.5,2)

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

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。