Practice 2：Python Numerical Method

GongYi龚怡

2167570874@qq.com

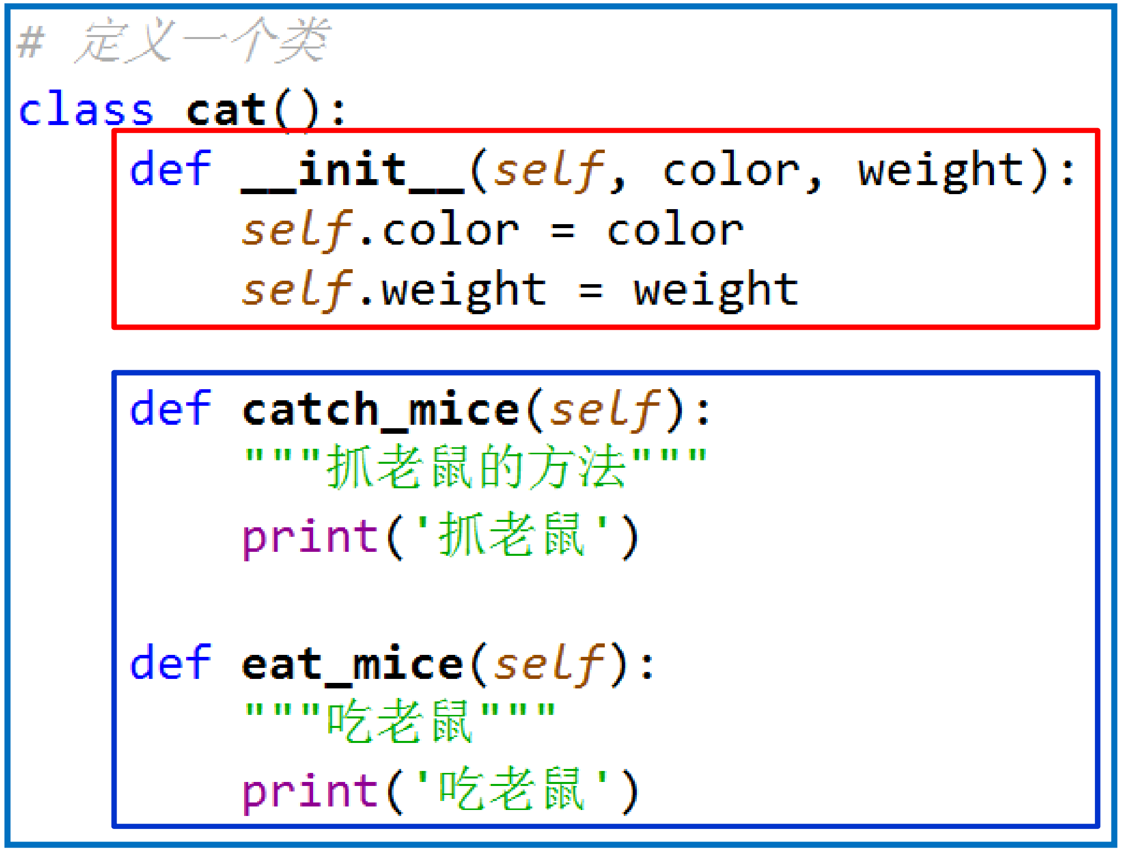
- Python 2：Python programming
- Exercise 1： Drawing the simulation process of solving nonlinear equations
- Exercise 2： Solve the system of Hilbert coefficient matrix equations

Python 2

- 2.1 Class and Instance
- 2.2 Iteration
- 2.3 Data Structure of NumPy
- 2.4 Matrix Operations of NumPy
- 2.5 Linear Algebra of NumPy线性代数
- 2.6 Drawing by using python

2.1 Class and Instance

- Classes are the foundation of object-oriented programming
- A class consists of two parts: properties and methods



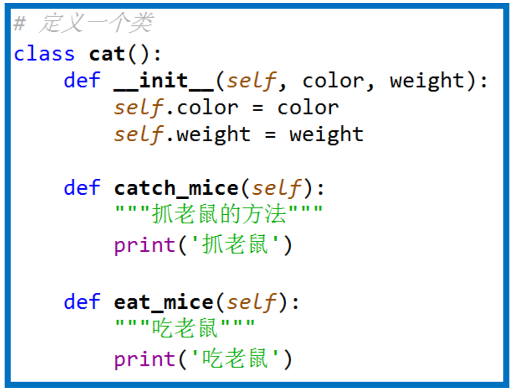
```
# 定义一个类
class cat():
    def __init__(self, color, weight):
        self.color = color
        self.weight = weight

    def catch_mice(self):
        """抓老鼠的方法"""
        print('抓老鼠')

    def eat_mice(self):
        """吃老鼠"""
        print('吃老鼠')
```

__init__(self) initialization：Every class needs initialization, set the properties of the class, execute some methods when instantiating a class

The methods of this class are functions

Definition and Usage of Class

- 1）Class definition

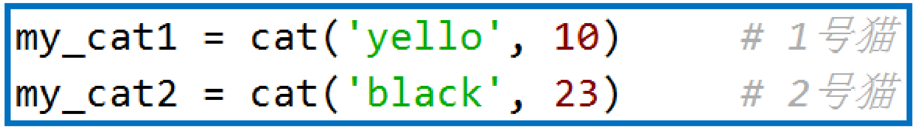


```
# 定义一个类
class cat():
    def __init__(self, color, weight):
        self.color = color
        self.weight = weight

    def catch_mice(self):
        """抓老鼠的方法"""
        print('抓老鼠')

    def eat_mice(self):
        """吃老鼠"""
        print('吃老鼠')
```

Properties

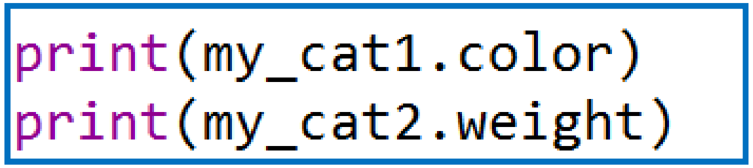
Methods

- 2）Instantiating class



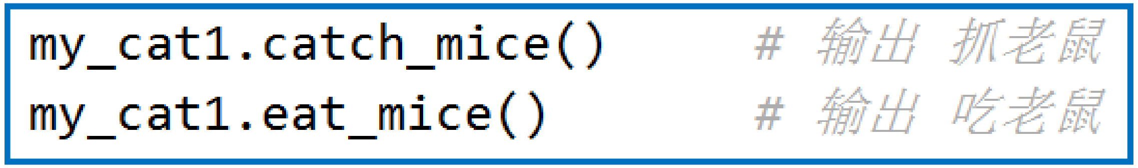
```
my_cat1 = cat('yello', 10)      # 1号猫
my_cat2 = cat('black', 23)      # 2号猫
```

- 3）Query the properties of the class



```
print(my_cat1.color)
print(my_cat2.weight)
```

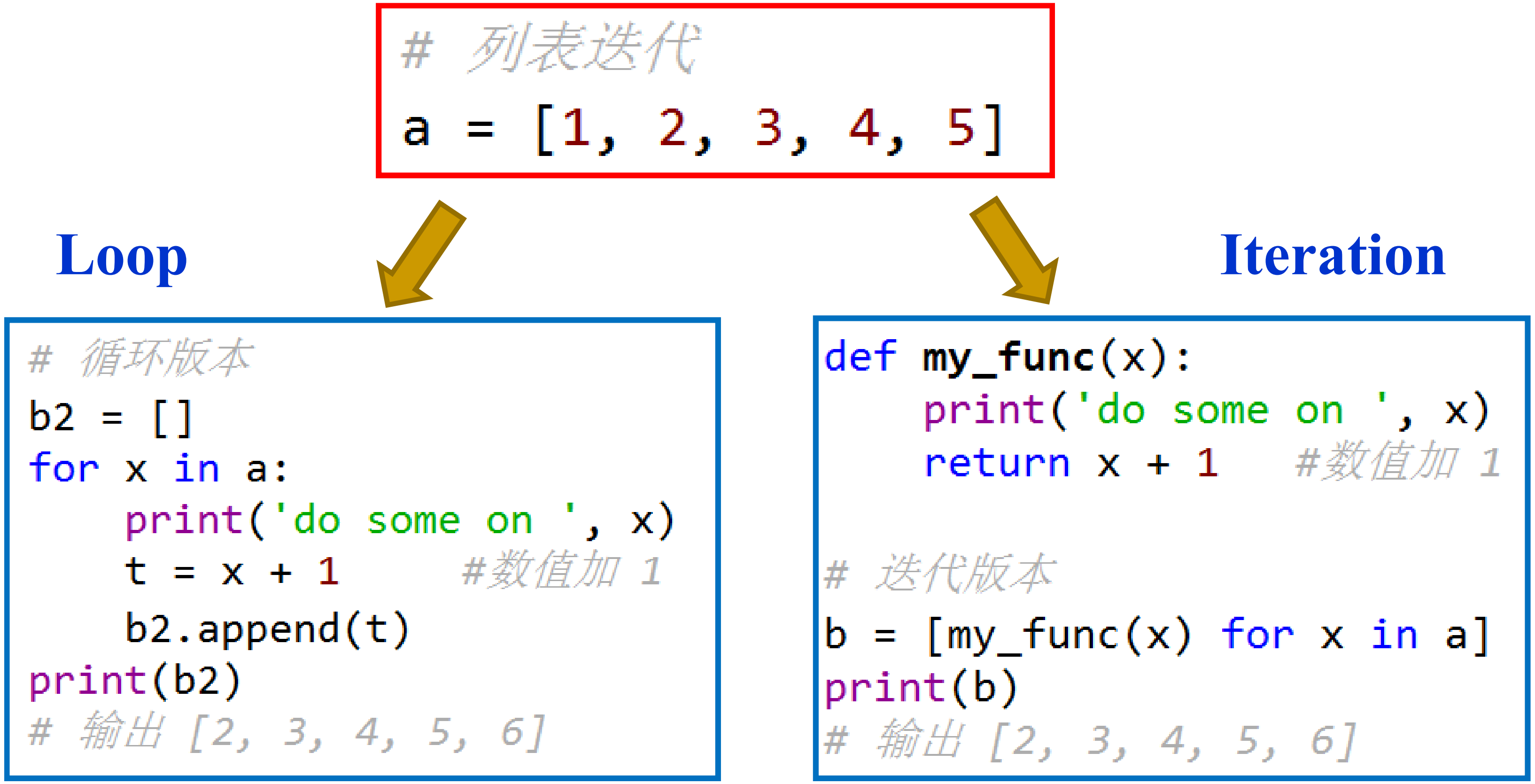
- 4）Call the methods



```
my_cat1.catch_mice()      # 输出 抓老鼠
my_cat1.eat_mice()        # 输出 吃老鼠
```

2.2 Iteration in Python

- Iteration can be seen as an efficient loop
- Improve programming efficiency and code readability



```
#  列表迭代
a = [1, 2, 3, 4, 5]
```

Loop

```
#  循环版本
b2 = []
for x in a:
    print('do some on ', x)
    t = x + 1      #数值加 1
    b2.append(t)
print(b2)
#  输出 [2, 3, 4, 5, 6]
```

Iteration

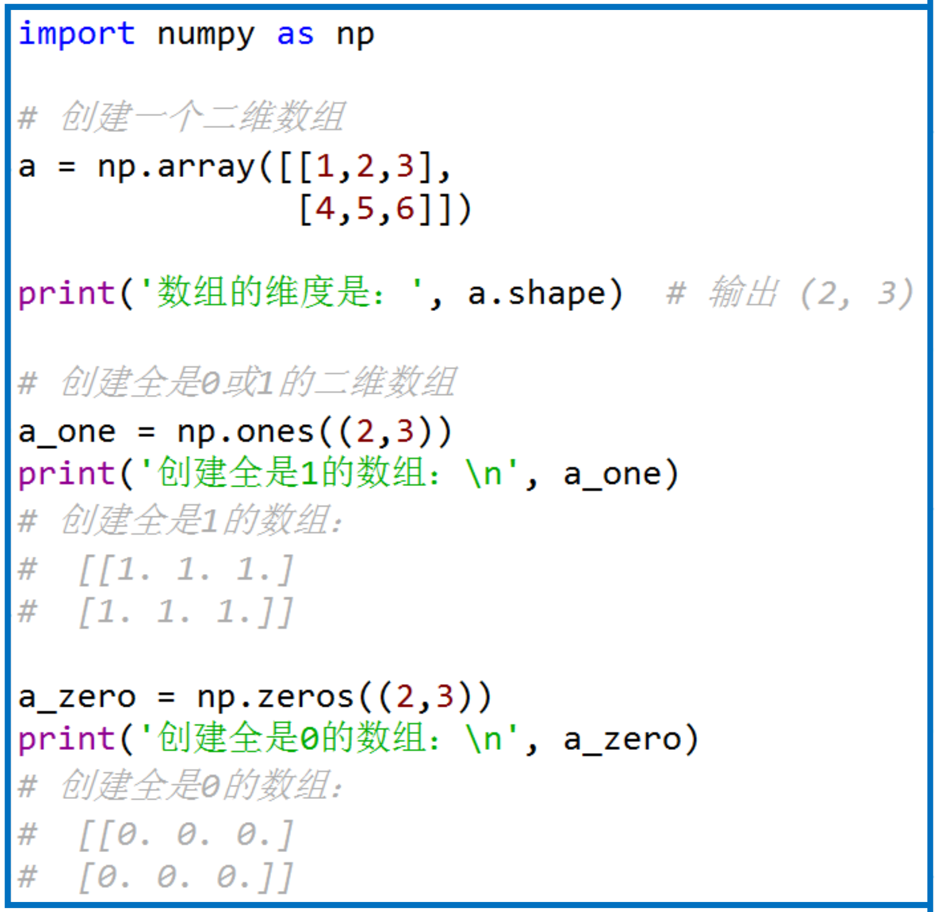
```
def my_func(x):
    print('do some on ', x)
    return x + 1    #数值加 1

#  迭代版本
b = [my_func(x) for x in a]
print(b)
#  输出 [2, 3, 4, 5, 6]
```

2.3 NumPy Data Structure

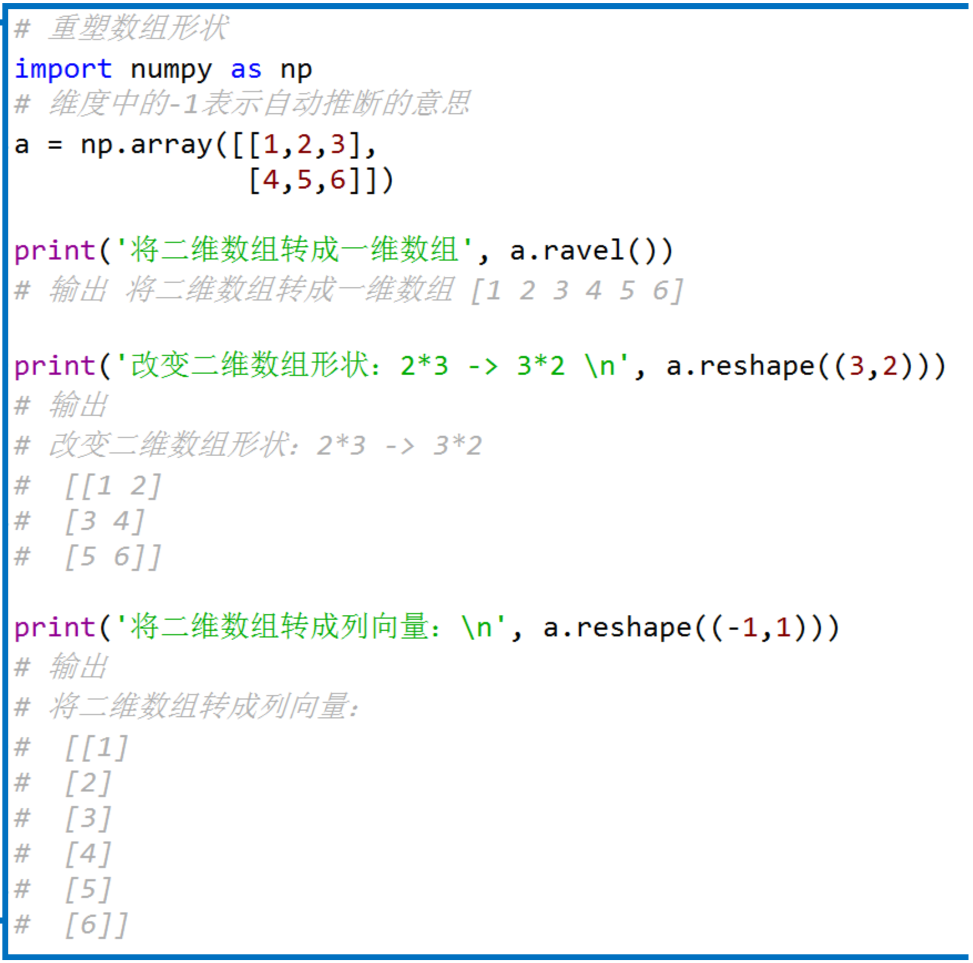
- Array
  - One-dimensional array： row vector
  - Two-dimensional array ： column vector or matrix
  - Three-dimensional array ： a two-dimensional array on the timeline
  - High-dimensional array
- Codes
  - Array creation np.array([ ])
  - Show the shape .shape
  - All 0s & 1s array np.ones((x,y))和 np.zeros ((x,y))
  - Modify the shape .ravel()和.reshape((x,y))

*x,y are the rows and coloumns

Programming 1：



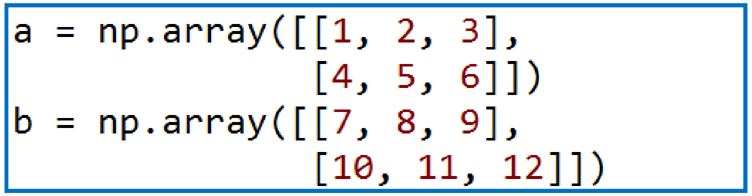
```python
import numpy as np

# 创建一个二维数组
a = np.array([[1,2,3],
              [4,5,6]])

print('数组的维度是： ', a.shape)  # 输出 (2, 3)

# 创建全是0或1的二维数组
a_one = np.ones((2,3))
print('创建全是1的数组： \n', a_one)
# 创建全是1的数组:
#  [[1. 1. 1.]
#  [1. 1. 1.]]

a_zero = np.zeros((2,3))
print('创建全是0的数组： \n', a_zero)
# 创建全是0的数组:
#  [[0. 0. 0.]
#  [0. 0. 0.]]
```



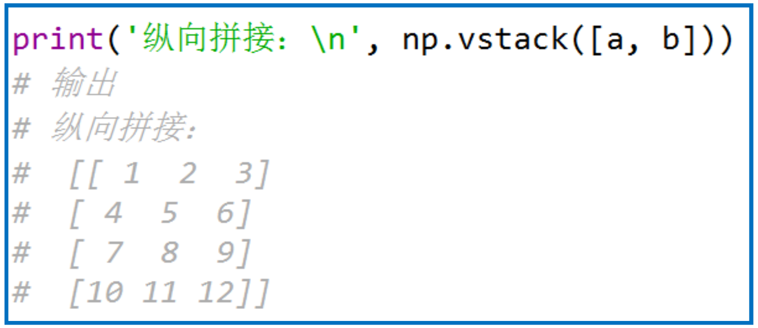
```python
# 重塑数组形状
import numpy as np
# 维度中的-1表示自动推断的意思
a = np.array([[1,2,3],
              [4,5,6]])

print('将二维数组转成一维数组', a.ravel())
# 输出 将二维数组转成一维数组 [1 2 3 4 5 6]

print('改变二维数组形状： 2*3 -> 3*2 \n', a.reshape((3,2)))
# 输出
# 改变二维数组形状: 2*3 -> 3*2
#  [[1 2]
#  [3 4]
#  [5 6]]

print('将二维数组转成列向量： \n', a.reshape((-1,1)))
# 输出
# 将二维数组转成列向量:
#  [[1]
#  [2]
#  [3]
#  [4]
#  [5]
#  [6]]
```

Codes

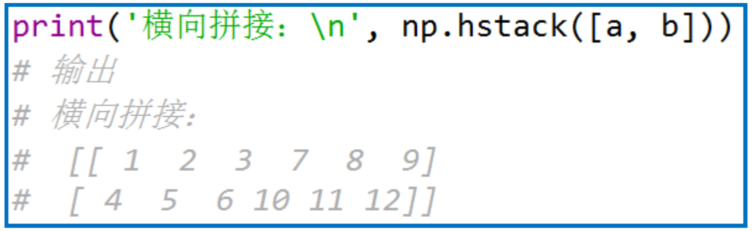
- Stack arrays in sequence vertically np.vstack([array 1,array 2])
- **Stack** arrays in sequence horizontally np.hstack([array1,array 2])
- Array slicing



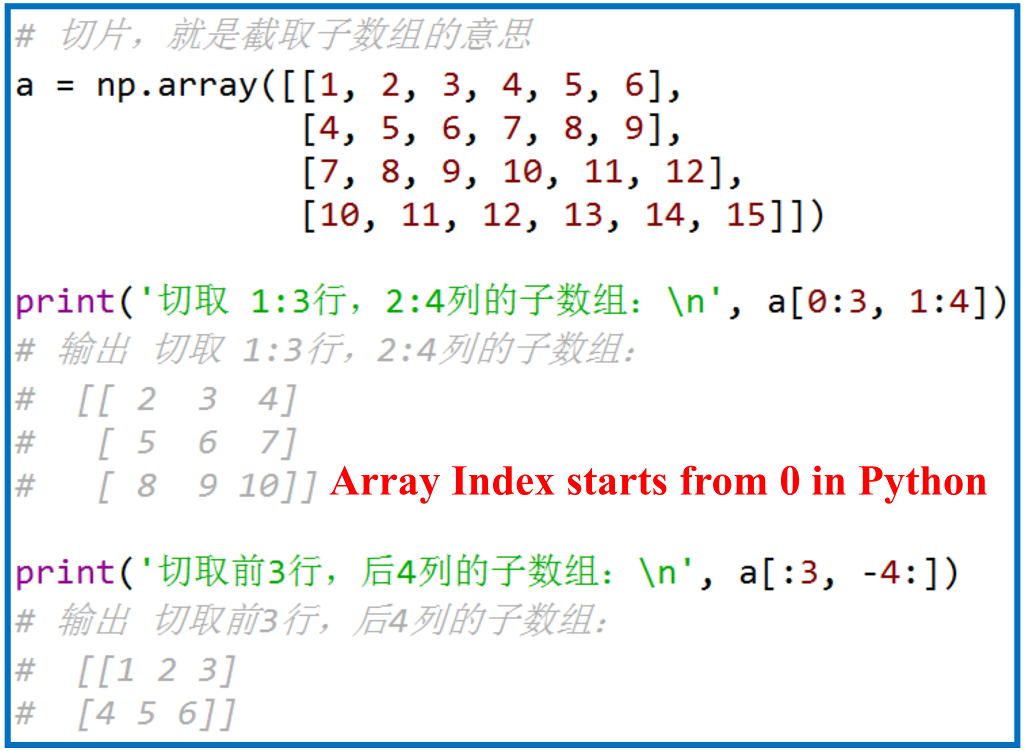
```
a = np.array([[1, 2, 3],
              [4, 5, 6]])
b = np.array([[7, 8, 9],
              [10, 11, 12]])
```



```
print('纵向拼接：\n', np.vstack([a, b]))
# 输出
# 纵向拼接:
#  [[ 1  2  3]
#  [ 4  5  6]
#  [ 7  8  9]
#  [10 11 12]]
```



```
print('横向拼接：\n', np.hstack([a, b]))
# 输出
# 横向拼接:
#  [[ 1  2  3  7  8  9]
#  [ 4  5  6 10 11 12]]
```

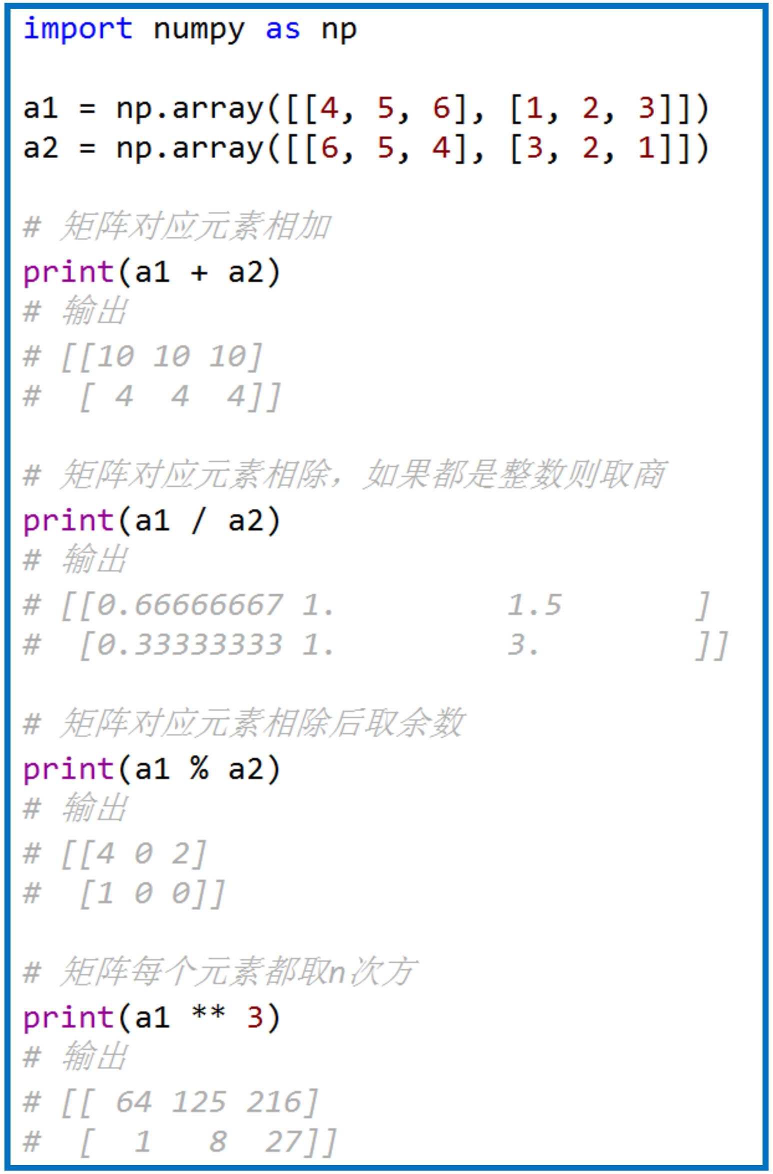


```
# 切片，就是截取子数组的意思
a = np.array([[1, 2, 3, 4, 5, 6],
              [4, 5, 6, 7, 8, 9],
              [7, 8, 9, 10, 11, 12],
              [10, 11, 12, 13, 14, 15]])

print('切取 1:3行，2:4列的子数组：\n', a[0:3, 1:4])
# 输出 切取 1:3行，2:4列的子数组:
#  [[ 2  3  4]
#   [ 5  6  7]
#   [ 8  9 10]]

print('切取前3行，后4列的子数组：\n', a[:3, -4:])
# 输出 切取前3行，后4列的子数组:
#  [[1 2 3]
#  [4 5 6]]
```

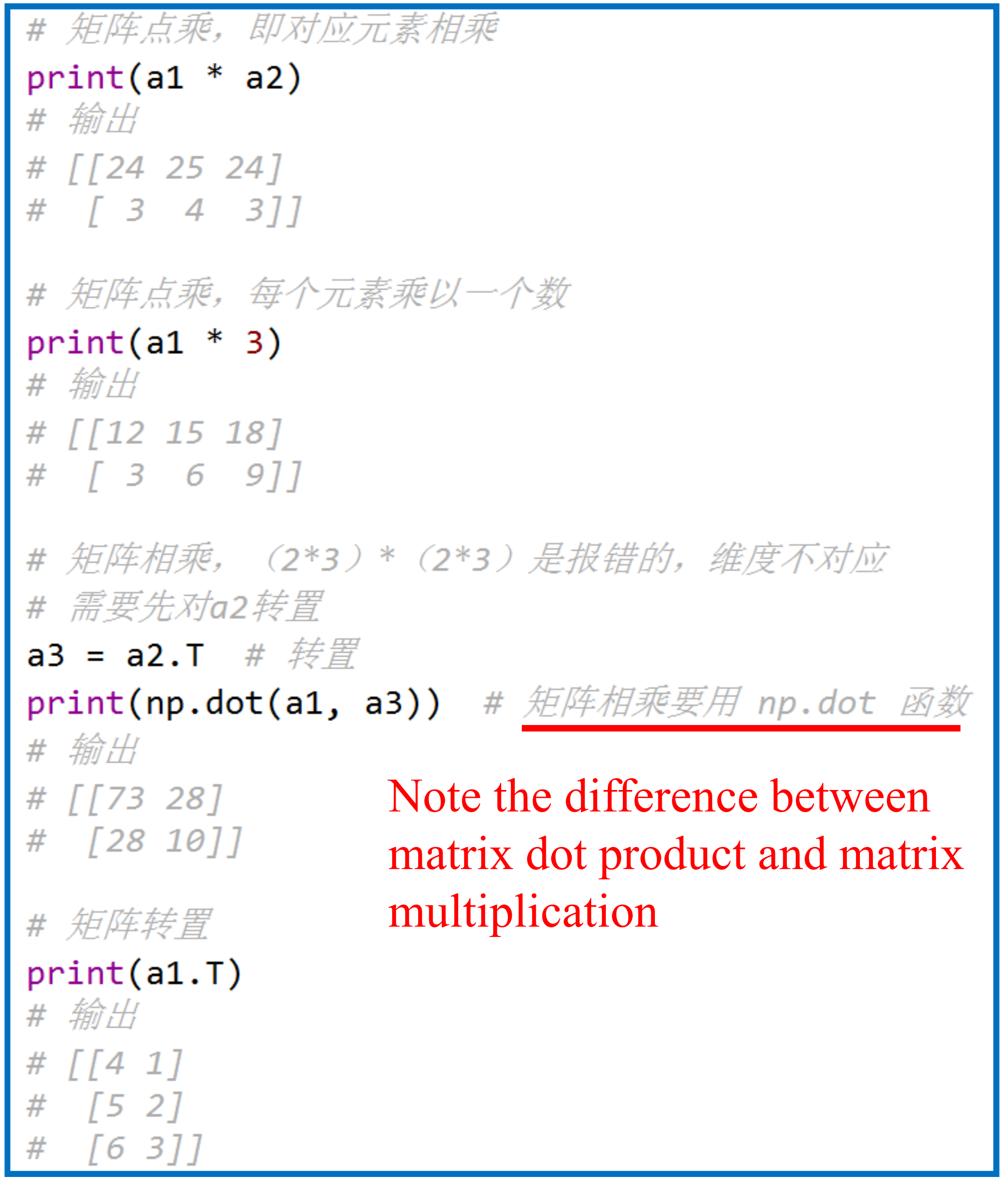
Array Index starts from 0 in Python

Exercise: Slice and stack columns 1,3 of array A and column 2 of array B

2.4 NumPy Matrix operations



```
import numpy as np

a1 = np.array([[4, 5, 6], [1, 2, 3]])
a2 = np.array([[6, 5, 4], [3, 2, 1]])

# 矩阵对应元素相加
print(a1 + a2)
# 输出
# [[10 10 10]
#  [ 4  4  4]]

# 矩阵对应元素相除，如果都是整数则取商
print(a1 / a2)
# 输出
# [[0.66666667 1.         1.5       ]
#  [0.33333333 1.         3.        ]]

# 矩阵对应元素相除后取余数
print(a1 % a2)
# 输出
# [[4 0 2]
#  [1 0 0]]

# 矩阵每个元素都取n次方
print(a1 ** 3)
# 输出
# [[ 64 125 216]
#  [  1   8  27]]
```

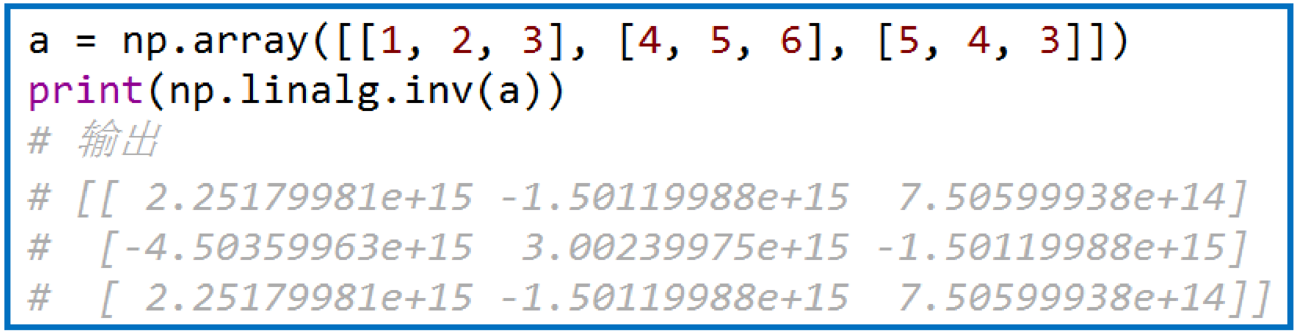


```
# 矩阵点乘，即对应元素相乘
print(a1 * a2)
# 输出
# [[24 25 24]
#  [ 3  4  3]]

# 矩阵点乘，每个元素乘以一个数
print(a1 * 3)
# 输出
# [[12 15 18]
#  [ 3  6  9]]

# 矩阵相乘，（2*3）*（2*3）是报错的，维度不对应
# 需要先对a2转置
a3 = a2.T  # 转置
print(np.dot(a1, a3))  # 矩阵相乘要用 np.dot 函数
# 输出
# [[73 28]
#  [28 10]]

# 矩阵转置
print(a1.T)
# 输出
# [[4 1]
#  [5 2]
#  [6 3]]
```

Note the difference between matrix dot product and matrix multiplication

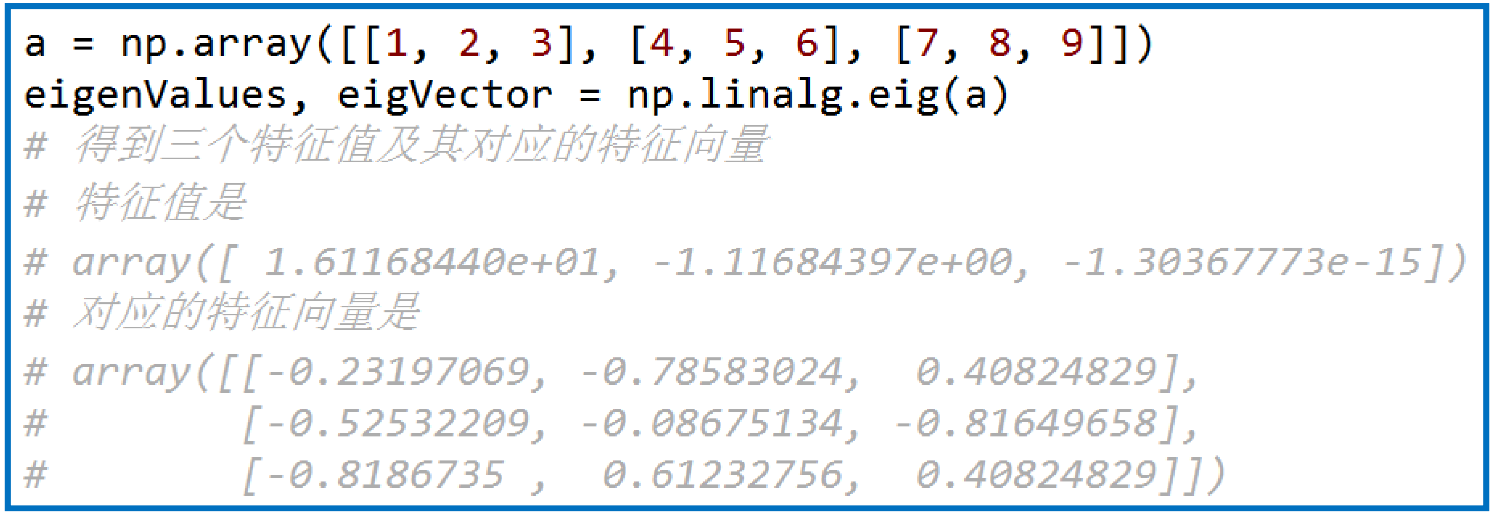
2.4 NumPy Matrix Operations

- Inverse a matrix



```
a = np.array([[1, 2, 3], [4, 5, 6], [5, 4, 3]])
print(np.linalg.inv(a))
# 输出
# [[ 2.25179981e+15 -1.50119988e+15  7.50599938e+14]
#  [-4.50359963e+15  3.00239975e+15 -1.50119988e+15]
#  [ 2.25179981e+15 -1.50119988e+15  7.50599938e+14]]
```

- Eigenvalues and eigenvectors



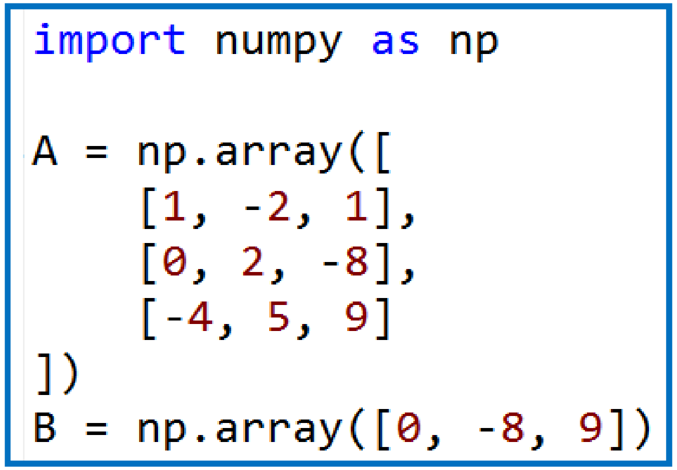
```
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
eigenValues, eigVector = np.linalg.eig(a)
# 得到三个特征值及其对应的特征向量
# 特征值是
# array([ 1.61168440e+01, -1.11684397e+00, -1.30367773e-15])
# 对应的特征向量是
# array([[-0.23197069, -0.78583024,  0.40824829],
#        [-0.52532209, -0.08675134, -0.81649658],
#        [-0.8186735 ,  0.61232756,  0.40824829]])
```

2.5 NumPy Solving equations directly

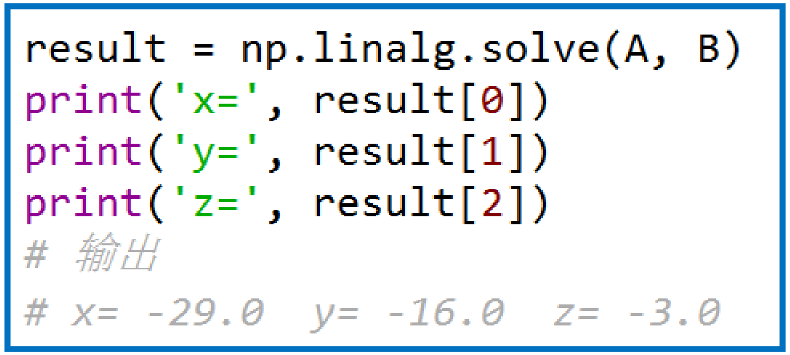
- General form Linear equations

$$\boldsymbol{Ax} = \boldsymbol{b}$$

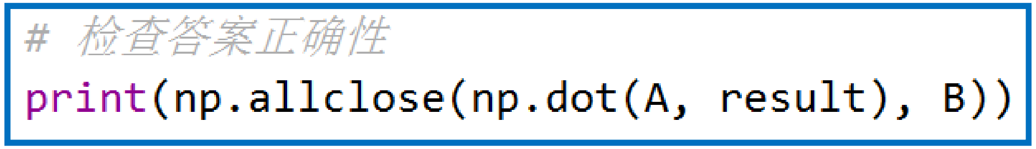
$$A = \begin{bmatrix} 1 & -2 & 1 \\ 0 & 2 & -8 \\ -4 & 5 & 9 \end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix} 0 \\ -8 \\ 9 \end{bmatrix}$$



```python
import numpy as np

A = np.array([
     [1, -2, 1],
     [0, 2, -8],
     [-4, 5, 9]
])
B = np.array([0, -8, 9])
```



```python
result = np.linalg.solve(A, B)
print('x=', result[0])
print('y=', result[1])
print('z=', result[2])
# 输出
# x= -29.0  y= -16.0  z= -3.0
```



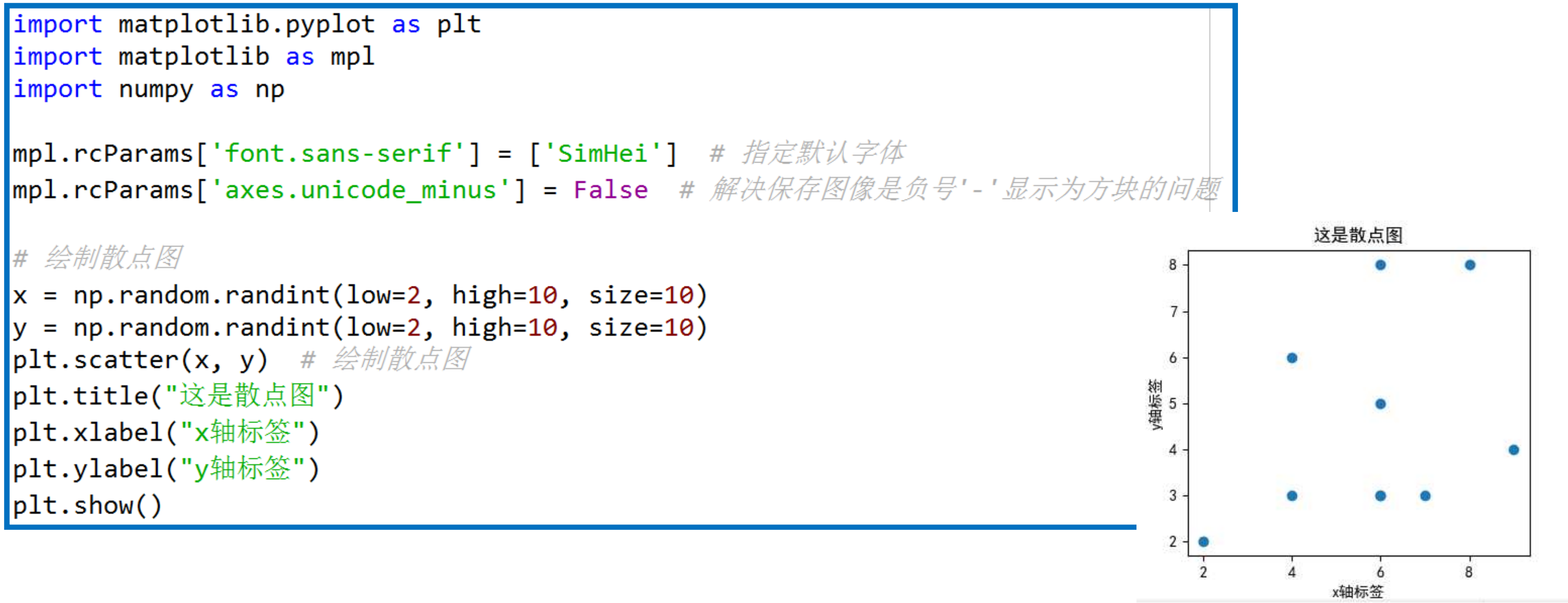
```python
# 检查答案正确性
print(np.allclose(np.dot(A, result), B))
```

https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html

2.6 Python Drawing

- Numerical Python -- Numpy
- Visualization library in Python for 2D plots of arrays -- Matplotlib
  - **Seaborn** is a **Python** data visualization library based on matplotlib

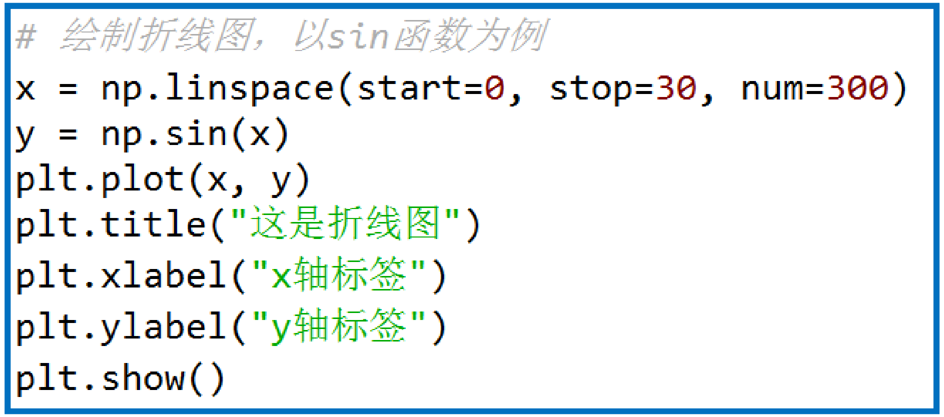
1） Scatter plot plt.scatter(x,y)



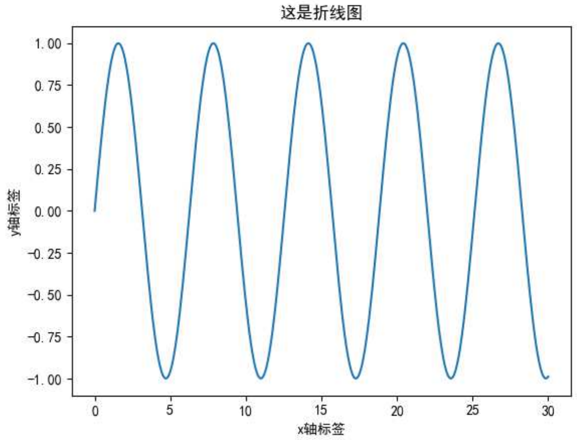
```
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np

mpl.rcParams['font.sans-serif'] = ['SimHei']  # 指定默认字体
mpl.rcParams['axes.unicode_minus'] = False  # 解决保存图像是负号'-'显示为方块的问题

# 绘制散点图
x = np.random.randint(low=2, high=10, size=10)
y = np.random.randint(low=2, high=10, size=10)
plt.scatter(x, y)  # 绘制散点图
plt.title("这是散点图")
plt.xlabel("x轴标签")
plt.ylabel("y轴标签")
plt.show()
```

2.6 Python Drawing

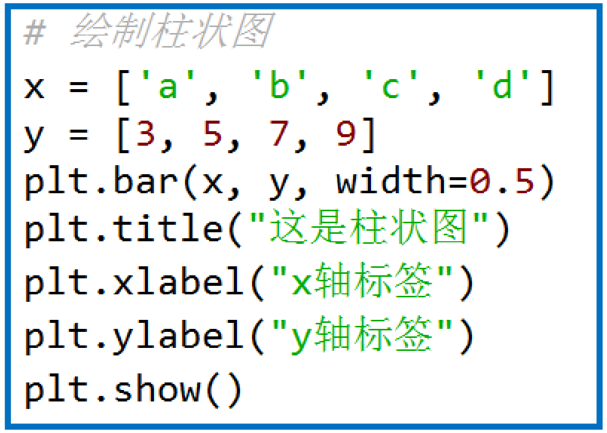
2） Lines image plt.plot(x,y)



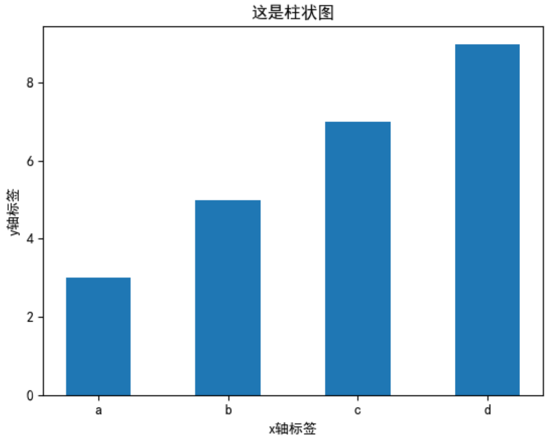
```
# 绘制折线图，以sin函数为例
x = np.linspace(start=0, stop=30, num=300)
y = np.sin(x)
plt.plot(x, y)
plt.title("这是折线图")
plt.xlabel("x轴标签")
plt.ylabel("y轴标签")
plt.show()
```



3）Bar image plt.bar(x,y)

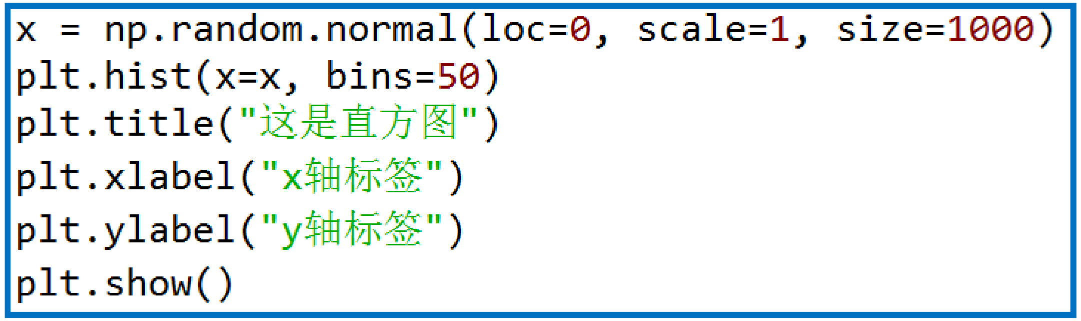


```
# 绘制柱状图
x = ['a', 'b', 'c', 'd']
y = [3, 5, 7, 9]
plt.bar(x, y, width=0.5)
plt.title("这是柱状图")
plt.xlabel("x轴标签")
plt.ylabel("y轴标签")
plt.show()
```

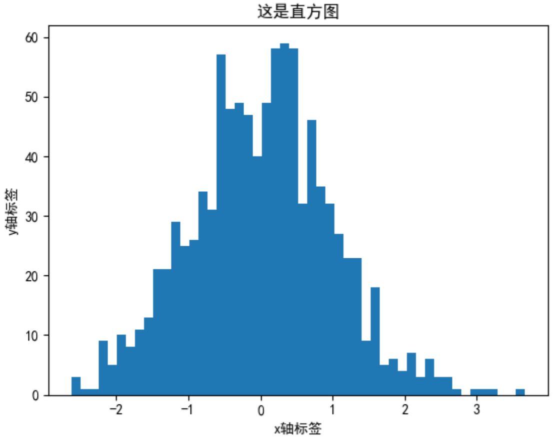


2.6 Python Drawing

4） **Histogram** plt.hist(x=x,bins=n)



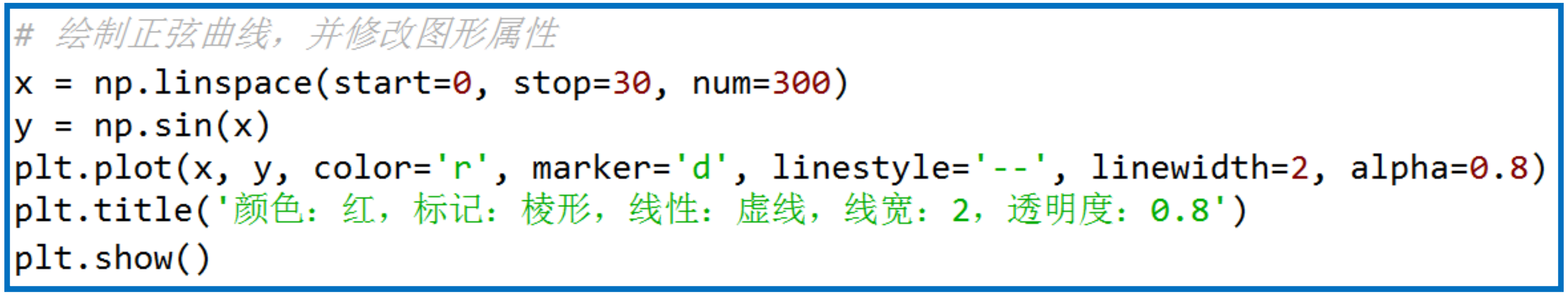
```
x = np.random.normal(loc=0, scale=1, size=1000)
plt.hist(x=x, bins=50)
plt.title("这是直方图")
plt.xlabel("x轴标签")
plt.ylabel("y轴标签")
plt.show()
```



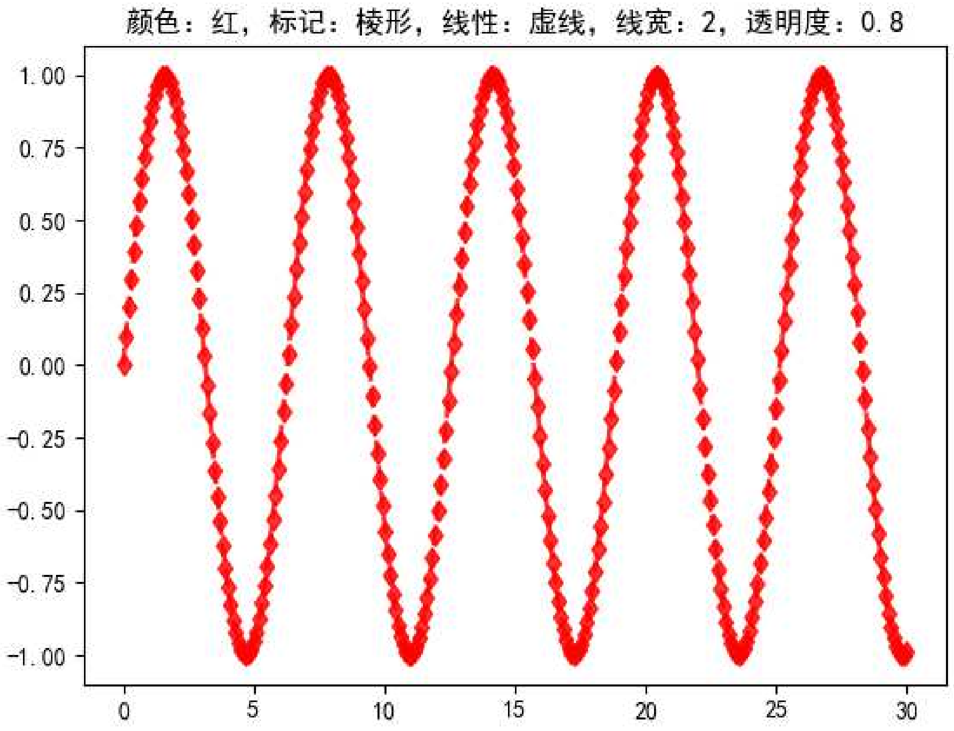
5） Graph properties

- Color r-red， b-blue， g-green， y-yellow
- o-circle marker， .-dot marker， d-diamond marker
- -- dashed line (虚线)， -solid line(实线)

2.6 Python Drawing

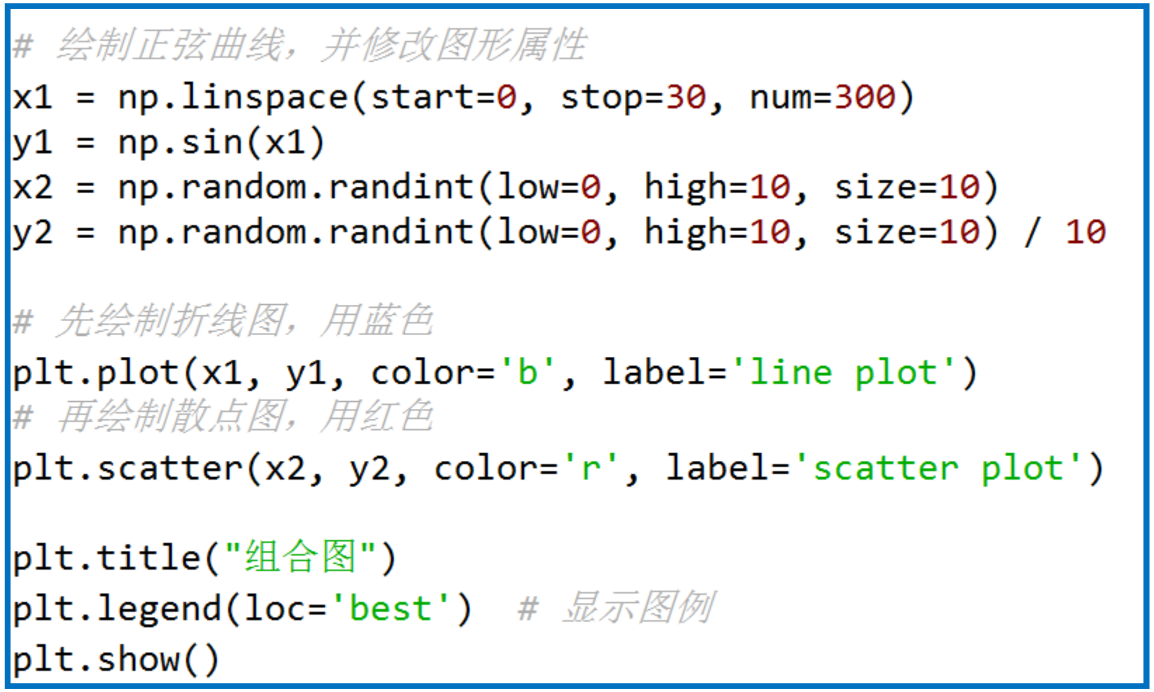


```
# 绘制正弦曲线，并修改图形属性
x = np.linspace(start=0, stop=30, num=300)
y = np.sin(x)
plt.plot(x, y, color='r', marker='d', linestyle='--', linewidth=2, alpha=0.8)
plt.title('颜色：红，标记：棱形，线性：虚线，线宽：2，透明度：0.8')
plt.show()
```

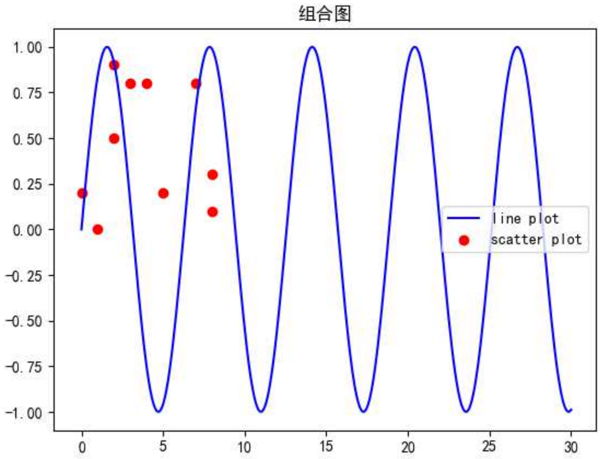


2.6 Python Drawing

6）Combination graph

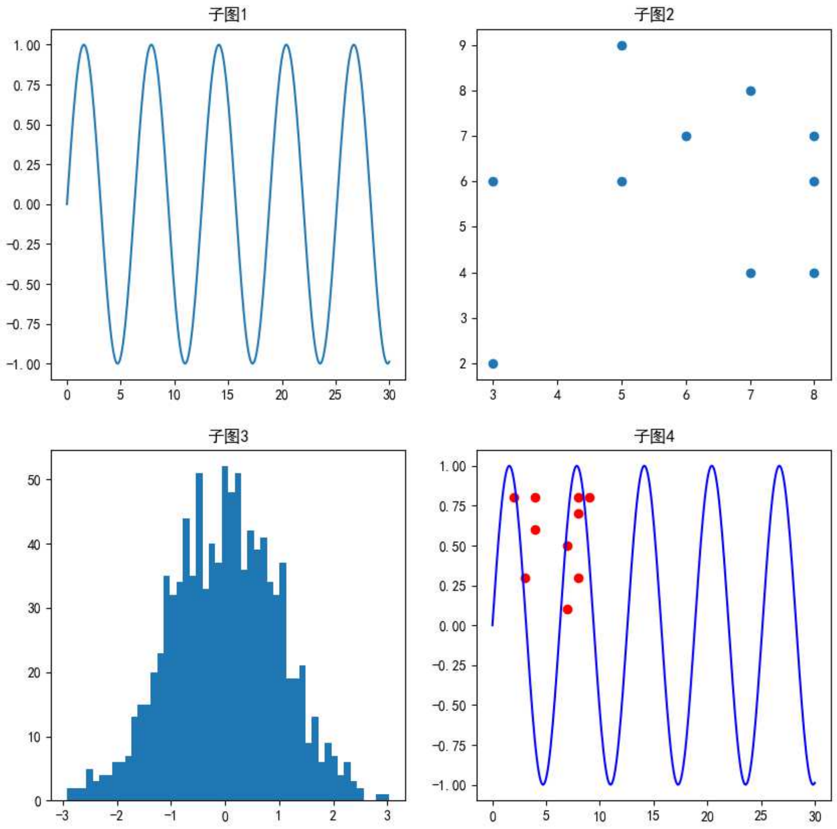


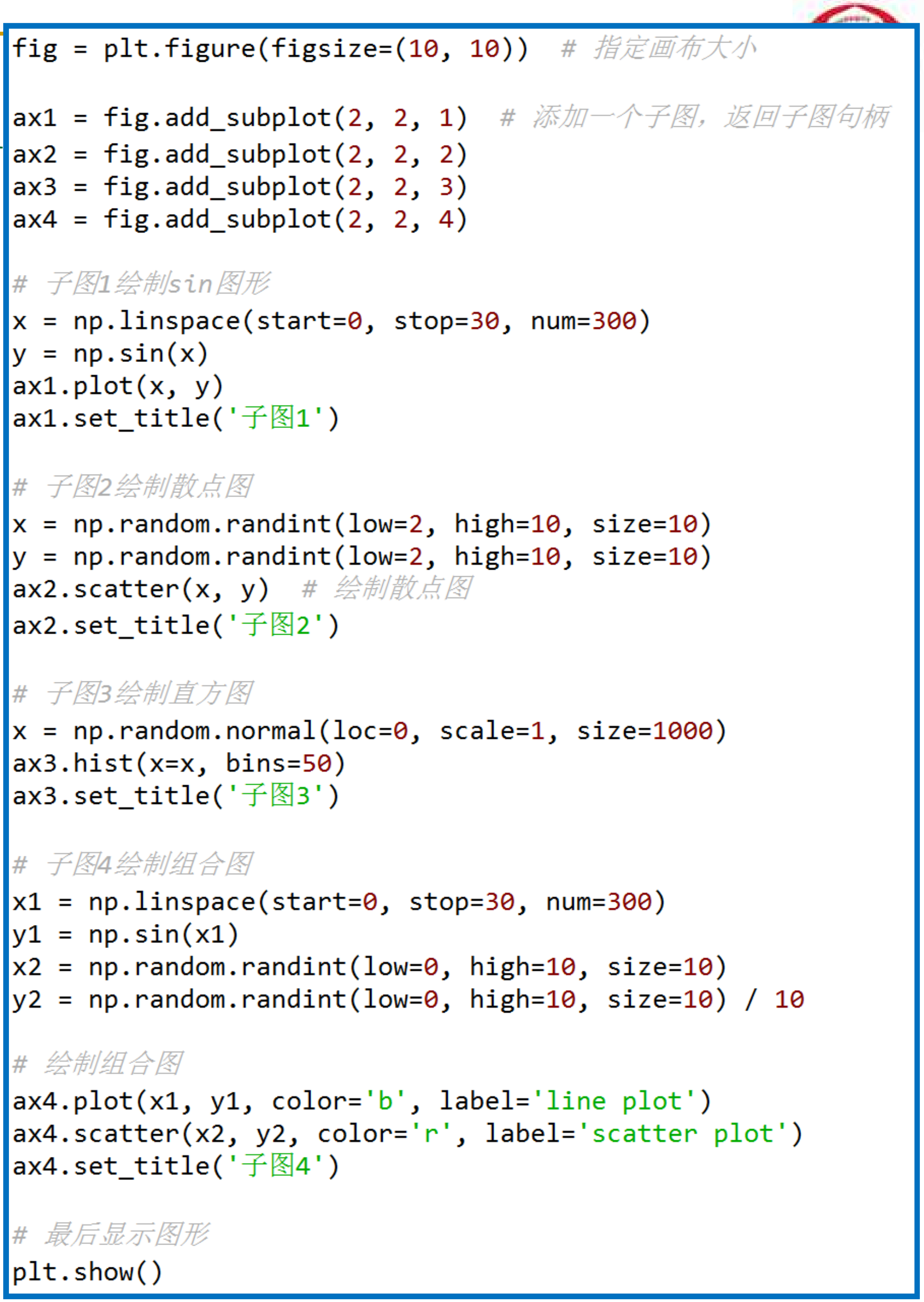
```
# 绘制正弦曲线，并修改图形属性
x1 = np.linspace(start=0, stop=30, num=300)
y1 = np.sin(x1)
x2 = np.random.randint(low=0, high=10, size=10)
y2 = np.random.randint(low=0, high=10, size=10) / 10

# 先绘制折线图，用蓝色
plt.plot(x1, y1, color='b', label='line plot')
# 再绘制散点图，用红色
plt.scatter(x2, y2, color='r', label='scatter plot')

plt.title("组合图")
plt.legend(loc='best')  # 显示图例
plt.show()
```



2.6 Python Drawin

7） Subgraph axi.plot( )

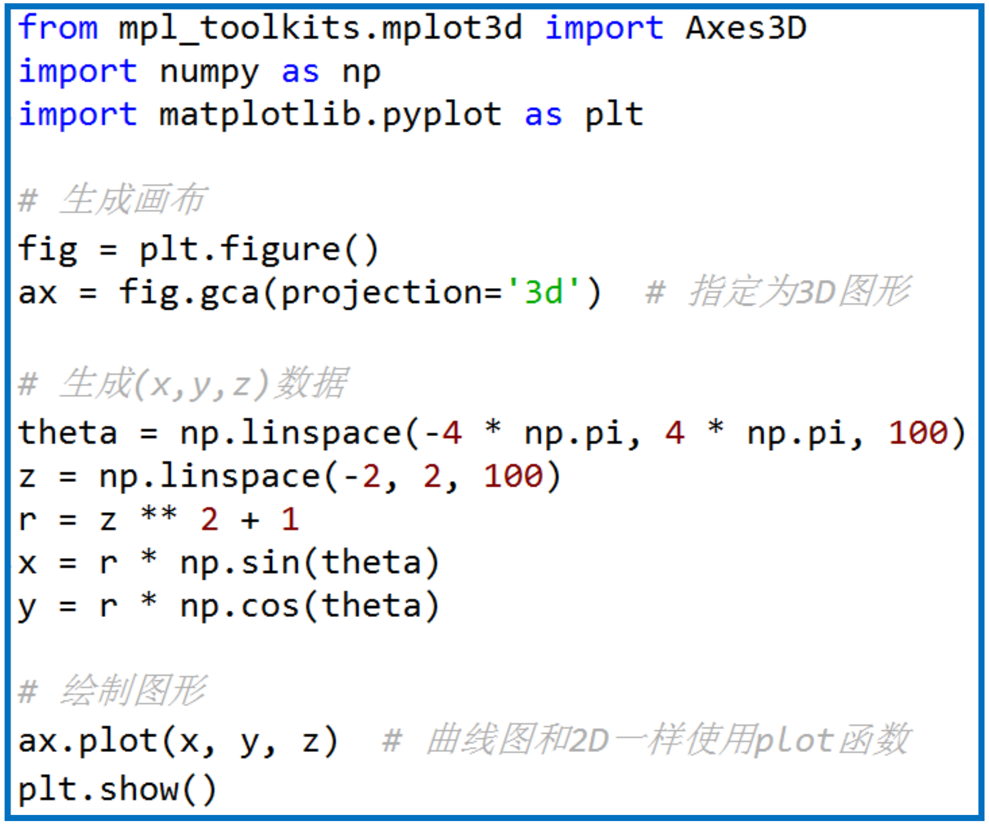




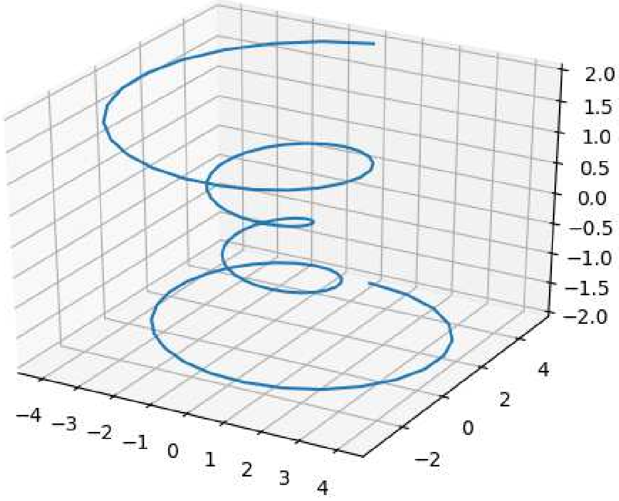
```
fig = plt.figure(figsize=(10, 10))  # 指定画布大小

ax1 = fig.add_subplot(2, 2, 1)  # 添加一个子图，返回子图句柄
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
ax4 = fig.add_subplot(2, 2, 4)

# 子图1绘制sin图形
x = np.linspace(start=0, stop=30, num=300)
y = np.sin(x)
ax1.plot(x, y)
ax1.set_title('子图1')

# 子图2绘制散点图
x = np.random.randint(low=2, high=10, size=10)
y = np.random.randint(low=2, high=10, size=10)
ax2.scatter(x, y)  # 绘制散点图
ax2.set_title('子图2')

# 子图3绘制直方图
x = np.random.normal(loc=0, scale=1, size=1000)
ax3.hist(x=x, bins=50)
ax3.set_title('子图3')

# 子图4绘制组合图
x1 = np.linspace(start=0, stop=30, num=300)
y1 = np.sin(x1)
x2 = np.random.randint(low=0, high=10, size=10)
y2 = np.random.randint(low=0, high=10, size=10) / 10

# 绘制组合图
ax4.plot(x1, y1, color='b', label='line plot')
ax4.scatter(x2, y2, color='r', label='scatter plot')
ax4.set_title('子图4')

# 最后显示图形
plt.show()
```

2.6 Python Drawing

8）3D graph import library mpl_toolkits.mplot3d

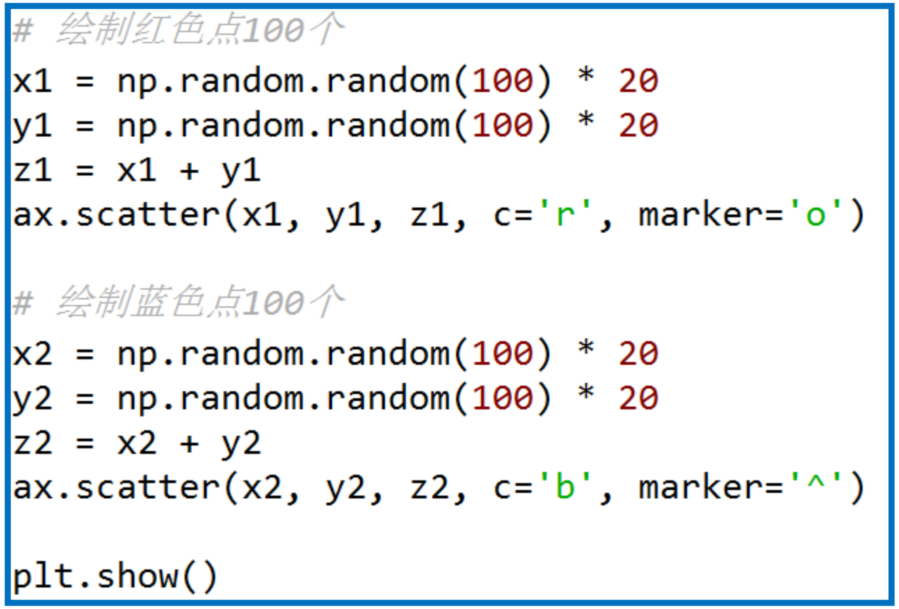


```
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt

# 生成画布
fig = plt.figure()
ax = fig.gca(projection='3d')  # 指定为3D图形

# 生成(x,y,z)数据
theta = np.linspace(-4 * np.pi, 4 * np.pi, 100)
z = np.linspace(-2, 2, 100)
r = z ** 2 + 1
x = r * np.sin(theta)
y = r * np.cos(theta)

# 绘制图形
ax.plot(x, y, z)  # 曲线图和2D一样使用plot函数
plt.show()
```

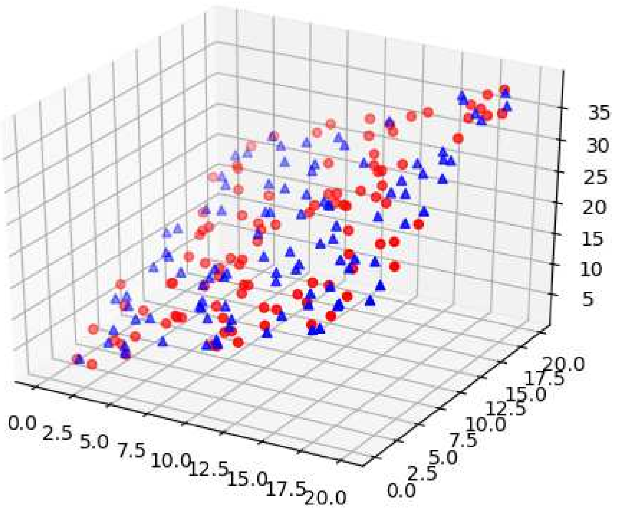


2.6 Python Drawing

9）3D scatter plot import library mpl_toolkits.mplot3d



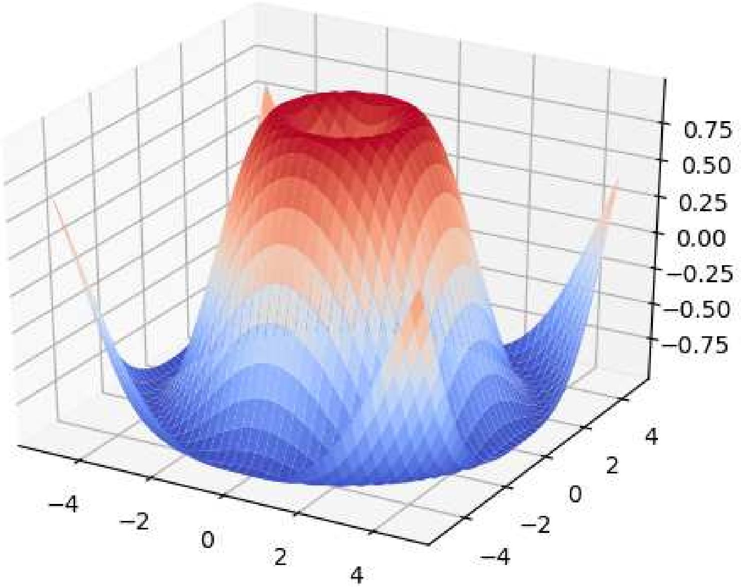
```python
#  绘制红色点100个
x1 = np.random.random(100) * 20
y1 = np.random.random(100) * 20
z1 = x1 + y1
ax.scatter(x1, y1, z1, c='r', marker='o')

#  绘制蓝色点100个
x2 = np.random.random(100) * 20
y2 = np.random.random(100) * 20
z2 = x2 + y2
ax.scatter(x2, y2, z2, c='b', marker='^')

plt.show()
```



2.6 Python Drawing

10） 3D surface plot import library pl_toolkits.mplot3d

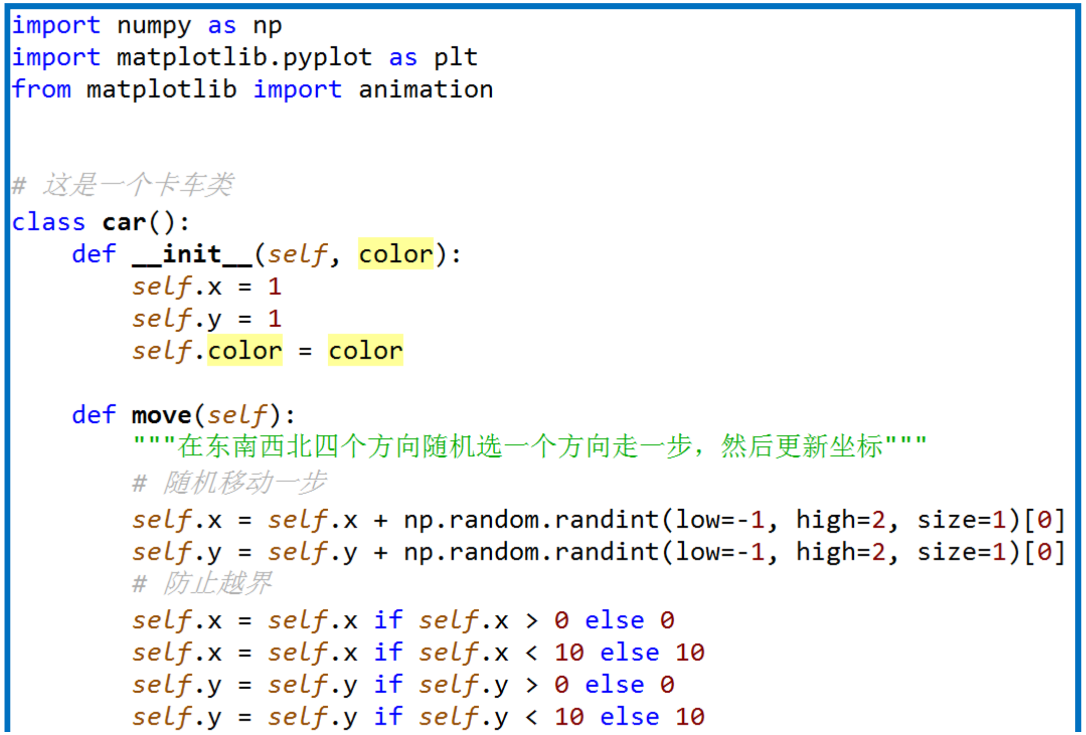
```
# 生成数据(x,y,z)
x = np.arange(-5, 5, 0.25)
y = np.arange(-5, 5, 0.25)
x, y = np.meshgrid(x, y)  # 重点，用np.meshgrid生成坐标网格矩阵
z = np.sin(np.sqrt(x ** 2 + y ** 2))

# 使用plot_surface函数
# cmap=cm.coolwarm 是颜色属性
surf = ax.plot_surface(x, y, z, cmap=cm.coolwarm)
plt.show()
```



2.6 Python Drawing

11) live graph, import library-animation

- Suppose a scene there are 3 trucks driving in a square. Draw the real time location of 3 trucks.
- The truck class consists of 3 properties, x and y represent its position, and marker represents its shape

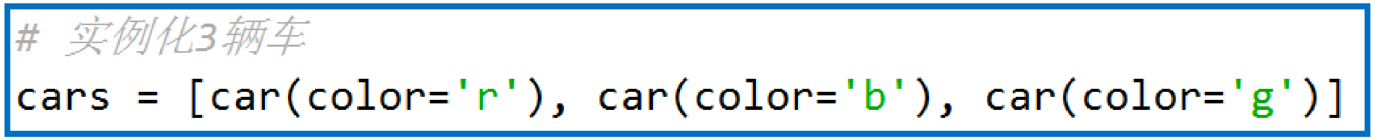


```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation


# 这是一个卡车类
class car():
    def __init__(self, color):
        self.x = 1
        self.y = 1
        self.color = color

    def move(self):
        """在东南西北四个方向随机选一个方向走一步，然后更新坐标"""
        # 随机移动一步
        self.x = self.x + np.random.randint(low=-1, high=2, size=1)[0]
        self.y = self.y + np.random.randint(low=-1, high=2, size=1)[0]
        # 防止越界
        self.x = self.x if self.x > 0 else 0
        self.x = self.x if self.x < 10 else 10
        self.y = self.y if self.y > 0 else 0
        self.y = self.y if self.y < 10 else 10
```

Define truck class

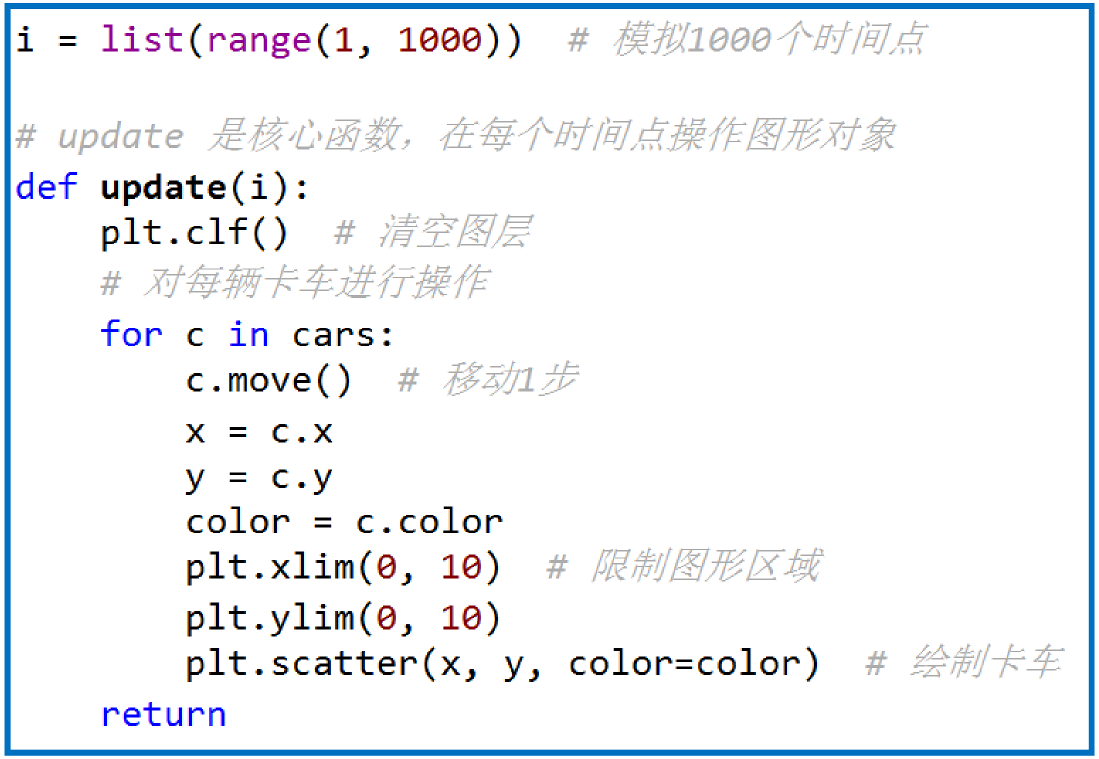
2.6 Python Drawing

- Instantiate 3 truck



```
# 实例化3辆车
cars = [car(color='r'), car(color='b'), car(color='g')]
```

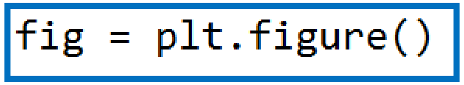
- Simulate 1000 time points, manipulate graphical objects at each time point



```
i = list(range(1, 1000))  # 模拟1000个时间点

# update 是核心函数，在每个时间点操作图形对象
def update(i):
    plt.clf()  # 清空图层
    # 对每辆卡车进行操作
    for c in cars:
        c.move()  # 移动1步
        x = c.x
        y = c.y
        color = c.color
        plt.xlim(0, 10)  # 限制图形区域
        plt.ylim(0, 10)
        plt.scatter(x, y, color=color)  # 绘制卡车
    return
```

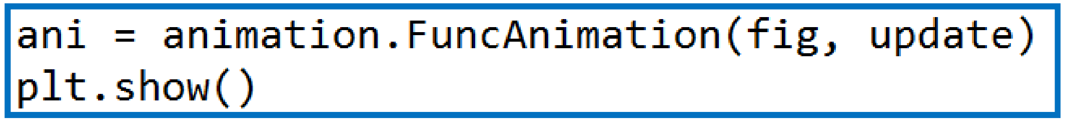
2.6 Python Drawing

- Draw a canvas

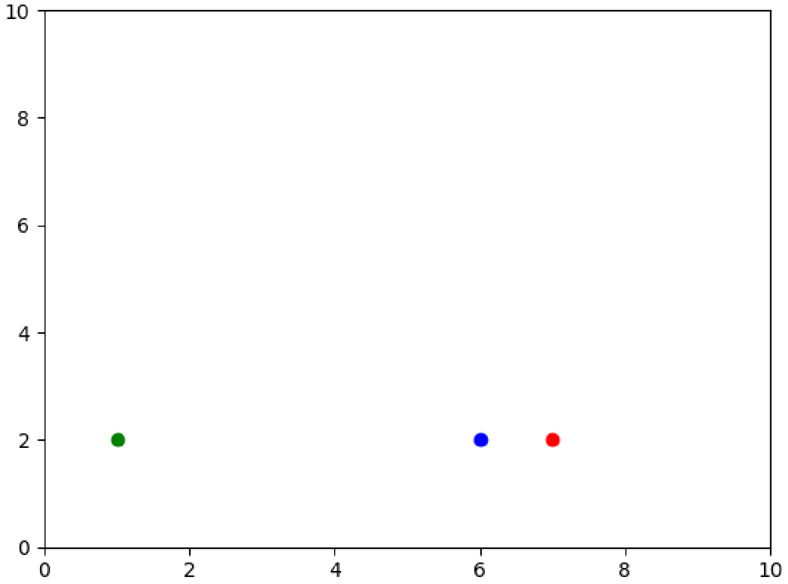


```
fig = plt.figure()
```

- Draw live graph



```
ani = animation.FuncAnimation(fig, update)
plt.show()
```



- Python 2： Python programming
- Exercise 1： Drawing the simulation process of solving nonlinear equations
- Exercise 2： Solve the system of Hilbert coefficient matrix equations

Exercise 1： Drawing the simulation process of solving nonlinear equations

- **Objective ：**

The iteration for the solution is shown in a live graph

- **Requirements：**

Modify fixPointFig.py， such that

1) The graph coordinates are automatically adjusted according to the reduction of the iteration range (Consider plt.xlim, ylim)；
2) Displays the iteration index and the value obtained by the corresponding iteration on the image (Modify title)

Exercise 2： Solve the system of Hilbert coefficient matrix equations

- **Objective：**

Compare the characteristics of direct and iterative methods to solve systems of ill-conditioned equations

- **Requirments：**

The coefficient matrix is a Hilbert matrix, a system of equations with all 1 solutions, with n = 2,3,......, programming, testing, and analyzing systems of equations by using the direct and iterative methods.

Hilbert matrix $H_n = \begin{bmatrix} 1 & \frac{1}{2} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & & \vdots \\ \vdots & \vdots & \ddots & \\ \frac{1}{n} & \frac{1}{n+1} & \cdots & \frac{1}{2n-1} \end{bmatrix}$

Practice 2 contents

- Complete exercise 1 and exercise 2， write “practice 2.docx”

Submit practice2.pdf on CG.