In [2]:
```
import numpy as np
data = np.random.randn(2,3)     #随机生成2行3列的数组
data
```

Out[2]:
```
array([[-0.58251903, -1.66431728, -0.33349477],
       [-0.15880157, -2.74270947,  0.36085614]])
```

In [2]:
```
data*10
data + data
```

Out[2]:
```
array([[ 0.7518058 , -0.21571538,  2.16032398],
       [ 1.29948852, -0.47847865, -2.85928574]])
```

In [4]:
```
data.shape     #查看数组的行列数
data.dtype     #查看数组内元素的数据类型
```

Out[4]:
```
dtype('float64')
```

In [5]:
```
data1 = [985, 211, 5.55, 0, 1]
array1 = np.array(data1)     # array函数接收任意序列型对象，生成一个新的包含传递数据
array1
```

Out[5]:
```
array([985.  , 211.  ,   5.55,   0.  ,   1.  ])
```

In [7]:
```
data2 = [[1,2,3],[4,5,6]]
array2 = np.array(data2)     #嵌套序列，例如同等长度的列表，将会自动转换成多维数组
array2
```

Out[7]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [8]:
```
array2.ndim     #检查几维数组
```

Out[8]:
```
2
```

In [9]:
```
array2.dtype     #array自动推断生成数组的数据类型
```

Out[9]:
```
dtype('int32')
```

In [11]:
```
np.zeros(5)     #生成全0数组
np.ones(5)     #生成全1数组
np.empty((3,2))     #生成没有初始化数值的数组（可能会如下返回垃圾数值），多维数组得在
```

Out[11]:
```
array([[0.7518058 , 0.21571538],
       [2.16032398, 1.29948852],
       [0.47847865, 2.85928574]])
```

In [12]:
```
np.arange(5)     #arange相当于range，默认生成数据类型是float64
```

Out[12]:
```
array([0, 1, 2, 3, 4])
```

In [13]:
```
arr = np.array([1,2,3])
float_arr = arr.astype(np.float64)     # astype改变array的数据类型
float_arr.dtype
```

Out[13]:
```
dtype('float64')
```

In [14]:
```
arr = np.array([[1.,2.,3.],[4.,5.,6]])
arr*arr     #数组任何批量操作都不用for循环，任何两个等尺寸数组的算数操作都应用了逐元
```

Out[14]:
```
array([[ 1.,  4.,  9.],
       [16., 25., 36.]])
```

In [15]:
```
1/arr
```

Out[15]:
```
array([[1.        , 0.5       , 0.33333333],
       [0.25      , 0.2       , 0.16666667]])
```

In [16]:
```
arr2 = np.array([[0.,4.,1.],[7.,2.,12.]])
arr2 > arr     #数组比较是每个元素一一对应比较，产生Boolean数组
```

Out[16]:
```
array([[False,  True, False],
       [ True, False,  True]])
```

In [17]:
```
arr = np.arange(5)
arr[3:5]     #索引取arr中第3-4位的元素，py中取值范围依旧前闭后开，且从第0位开始
```

Out[17]:
```
array([3, 4])
```

In [19]:
```
arr[3:5] = 9     #将3-4位元素替代为9
arr
```

Out[19]:
```
array([0, 1, 2, 9, 9])
```

In [20]:
```
arr_silce = arr[3:5]
arr_silce[1] = 123456
arr     #数组的切片也是原视图，任何对于视图的修改都会反映到原数组
        #如果想要拷贝而不是切片视图，则arr[3:5].copy()
```

Out[20]:
```
array([     0,      1,      2,      9, 123456])
```

In [21]:
```
arr2d = np.array([[1,2,3],[4,5,6],[7,8,9]])
arr2d[2]     #在二维数组中，每个索引值对应的是一个一维数组，py数组中也是00开始
```

Out[21]:
```
array([7, 8, 9])
```

In [22]:
```
arr2d[0,2]    #在高维数组索引取一个元素得写出它的坐标
```

Out[22]:
```
3
```

In [23]:
```
arr3d = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
arr3d    #arr3d是一个三维数组，索引首位定位一个二维数组，0指123456这个数组，1指7891
```

Out[23]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 7,  8,  9],
        [10, 11, 12]]])
```

In [24]:
```
arr3d[1,0]    #指第1个二维数组，第0行
```

Out[24]:
```
array([7, 8, 9])
```

In [25]:
```
arr3d[1,0,1]    #指第1个二维数组，第0行，第1列元素
```

Out[25]:
```
8
```

In [26]:
```
arr2d[:2]    #选择arr2d的前2行进行切片
```

Out[26]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [27]:
```
arr2d[:2,1:]    #选择arr2d的前两行中的1、2列
                #一个：表示选择整个轴上的数组
```

Out[27]:
```
array([[2, 3],
       [5, 6]])
```

In [33]:
```
names = np.array(['Bob','Joe','Will','Bob'])
data = np.random.randn(4,3)
data
```

Out[33]:
```
array([[ 1.30865186, -2.12142588, -2.06364165],
       [ 0.2851082 ,  0.28548592,  0.1483923 ],
       [-0.03486523,  1.56439757,  1.46028828],
       [-0.25387171, -1.30909419, -0.62879963]])
```

In [34]:
```
names == 'Bob'    #==是比较操作，生成一个Boolean数组
data[names == 'Bob']    #Boolean数组会索引出一个数组，前提是布尔数组长度必须和数组
                        #即布尔列一一对应着数组的行，ture对应的就显示，false就不显
```

Out[34]:
```
array([[ 1.30865186, -2.12142588, -2.06364165],
       [-0.25387171, -1.30909419, -0.62879963]])
```

In [35]:
```
data[~(names == 'Bob')]    # ~表示取反，即剩余的行
```

Out[35]:
```
array([[ 0.2851082 ,  0.28548592,  0.1483923 ],
       [-0.03486523,  1.56439757,  1.46028828]])
```

In [36]:
```
mask = (names == 'Bob') | (names == 'Will')    # &(and) |(or)
data[mask]
```

Out[36]:
```
array([[ 1.30865186, -2.12142588, -2.06364165],
       [-0.03486523,  1.56439757,  1.46028828],
       [-0.25387171, -1.30909419, -0.62879963]])
```

In [40]:
```
data[data < 0] = 0    #data中小于0的元素全替换成0、
data
```

Out[40]:
```
array([[1.30865186, 0.        , 0.        ],
       [0.2851082 , 0.28548592, 0.1483923 ],
       [0.        , 1.56439757, 1.46028828],
       [0.        , 0.        , 0.        ]])
```

In [41]:
```
arr = np.empty((8,4))
for i in range(8):
    arr[i] = i
arr
```

Out[41]:
```
array([[0., 0., 0., 0.],
       [1., 1., 1., 1.],
       [2., 2., 2., 2.],
       [3., 3., 3., 3.],
       [4., 4., 4., 4.],
       [5., 5., 5., 5.],
       [6., 6., 6., 6.],
       [7., 7., 7., 7.]])
```

In [42]:
```
arr[[4,3,0,6]]    #arr中括号套中括号，里面的表示取多少行，即取4306行
```

Out[42]:
```
array([[4., 4., 4., 4.],
       [3., 3., 3., 3.],
       [0., 0., 0., 0.],
       [6., 6., 6., 6.]])
```

In [43]:
```
arr[[-3,-5,-7]]    #使用负索引，将从尾部开始选择，尾部从-1开始，没有0
```

Out[43]:
```
array([[5., 5., 5., 5.],
       [3., 3., 3., 3.],
       [1., 1., 1., 1.]])
```

In [45]:
```
arr = np.arange(32).reshape((8,4))     #从0-31按顺序生成8行4列的数组，reshape用于改变
arr
```

Out[45]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23],
       [24, 25, 26, 27],
       [28, 29, 30, 31]])
```

In [46]:
```
arr[[1,5,7,2],[0,3,1,2]]     #取（1，0），（5，3），（7，2），（2，2）对应的元素
                             #两组【】，即其中一一对应组成坐标，取对应坐标的元素
```

Out[46]:
```
array([ 4, 23, 29, 10])
```

In [47]:
```
arr = np.arange(15).reshape((3,5))
arr.T    #arr.T即转置矩阵（数组），行列对换
```

Out[47]:
```
array([[ 0,  5, 10],
       [ 1,  6, 11],
       [ 2,  7, 12],
       [ 3,  8, 13],
       [ 4,  9, 14]])
```

In [48]:
```
np.dot(arr.T, arr)     #dot.求矩阵arr和其倒置的内积
                       #内积即，a00 = （0行对应元素*0列对应元素）加和，其实就是矩阵自
```

Out[48]:
```
array([[125, 140, 155, 170, 185],
       [140, 158, 176, 194, 212],
       [155, 176, 197, 218, 239],
       [170, 194, 218, 242, 266],
       [185, 212, 239, 266, 293]])
```

In [50]:
```
arr = np.arange(16).reshape((2,2,4))
arr
```

Out[50]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7]],

       [[ 8,  9, 10, 11],
        [12, 13, 14, 15]]])
```

In [51]:
```
arr.transpose((1,0,2))     #用于多维数组轴的转换，第一、二轴互换
```

Out[51]:
```
array([[[ 0,  1,  2,  3],
        [ 8,  9, 10, 11]],

       [[ 4,  5,  6,  7],
        [12, 13, 14, 15]]])
```

In [52]:
```
arr.swapaxes(1,2)     #swapaxes接收一对轴编号作为参数，并对轴进行调整用于重组数据
                      #（1，2）的意思是arr(2,2,4)中1、2轴互换，即（2，4，2），表示2个
                       #二维数组中只有两个参数，swapaxes(0,1)即等于倒置
```

Out[52]:
```
array([[[ 0,  4],
        [ 1,  5],
        [ 2,  6],
        [ 3,  7]],

       [[ 8, 12],
        [ 9, 13],
        [10, 14],
        [11, 15]]])
```

In [53]:
```
arr = np.arange(10)
np.sqrt(arr)     #对数组中每个元素开根号
np.exp(arr)     #对数组中每个元素求e的某次方
                #np.maximum(a,b)， 逐个把a,b两个数组中对应元素的最大值计算出来
```

Out[53]:
```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
       2.98095799e+03, 8.10308393e+03])
```

In [55]:
```
arr = np.random.randn(7)
remainder,whole_part = np.modf(arr)     #modf将arr中的每个元素分为小数部分，整数部分
remainder     #小数部分
```

Out[55]:
```
array([ 0.19309404,  0.86844082, -0.24225644, -0.49334057, -0.72104755,
       -0.5884988 , -0.77036242])
```

In [56]:
```
whole_part     #整数部分
```

Out[56]:
```
array([ 0.,  0., -0., -0., -1., -2., -0.])
```

In [57]:
```
np.sqrt(arr,arr)     #通用函数接收一个可选函数out,即双选arr,允许对数组按位置操作，而
                     #通用函数有abs, fabs(两者都绝对值)， square(平方)， rint(保留到整
                     #二元通用函数add,subtract,multiply,divide,power(幂)， mod,floor_
arr
```

```
C:\Users\12232\AppData\Local\Temp\ipykernel_19332\3432068196.py:1: RuntimeWarnin
g: invalid value encountered in sqrt
  np.sqrt(arr,arr)
```

Out[57]:
```
array([0.43942467, 0.93190172,        nan,        nan,        nan,
              nan,        nan])
```

In [61]:
```
points = np.arange(-5,5,0.01)
xs,ys = np.meshgrid(points,points)     #meshgrid函数接收两个一维数组（points），并根
                                       #x,y互为倒置，x中取值随列数（横坐标）增大而增
                                       #这样可以实现（x,y），在行列取值规律上模拟直角

xs
ys
```

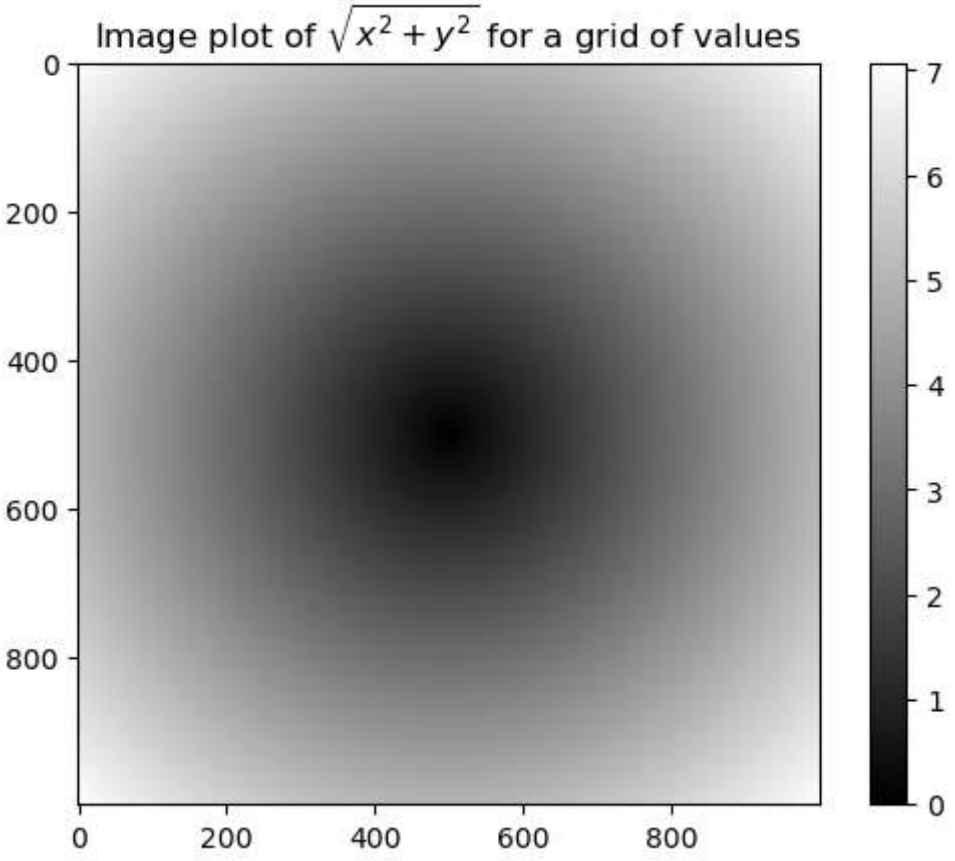
Out[61]:
```
array([[-5.  , -5.  , -5.  , ..., -5.  , -5.  , -5.  ],
       [-4.99, -4.99, -4.99, ..., -4.99, -4.99, -4.99],
       [-4.98, -4.98, -4.98, ..., -4.98, -4.98, -4.98],
       ...,
       [ 4.97,  4.97,  4.97, ...,  4.97,  4.97,  4.97],
       [ 4.98,  4.98,  4.98, ...,  4.98,  4.98,  4.98],
       [ 4.99,  4.99,  4.99, ...,  4.99,  4.99,  4.99]])
```

In [62]:
```
z = np.sqrt(xs**2 + ys**2)     #表示距离原点距离公式函数，x取列数，y取行数，即可找到
z
```

Out[62]:
```
array([[7.07106781, 7.06400028, 7.05693985, ..., 7.04988652, 7.05693985,
        7.06400028],
       [7.06400028, 7.05692568, 7.04985815, ..., 7.04279774, 7.04985815,
        7.05692568],
       [7.05693985, 7.04985815, 7.04278354, ..., 7.03571603, 7.04278354,
        7.04985815],
       ...,
       [7.04988652, 7.04279774, 7.03571603, ..., 7.0286414 , 7.03571603,
        7.04279774],
       [7.05693985, 7.04985815, 7.04278354, ..., 7.03571603, 7.04278354,
        7.04985815],
       [7.06400028, 7.05692568, 7.04985815, ..., 7.04279774, 7.04985815,
        7.05692568]])
```

In [65]:
```
import matplotlib.pyplot as plt
plt.imshow(z,cmap=plt.cm.gray)  #使用 imshow() 函数绘制图像。z 是要显示的图像数据，
plt.colorbar()  #添加一个颜色条，用于显示图像上不同灰度值对应的颜色。
plt.title("Image plot of $\sqrt{x^2 + y^2}$ for a grid of values")
#将z函数利用matplot数据可视化
```

Out[65]:
```
Text(0.5, 1.0, 'Image plot of $\\sqrt{x^2 + y^2}$ for a grid of values')
```



In [3]:
```
xarr = np.array([1.1,1.2,1.3,1.4,1.5])
yarr = np.array([2.1,2.2,2.3,2.4,2.5])
cond = np.array([True,False,True,True,False])
result = np.where(cond,xarr,yarr)     #类似于三元表达式 x if condition else y, cond
result
```

Out[3]:
```
array([1.1, 2.2, 1.3, 1.4, 2.5])
```

In [4]:
```
arr = np.random.randn(4,4)
np.where(arr>0, 2, -2)     #arr元素中大于0的就替换成2，否则替换成-2
```

Out[4]:
```
array([[ 2, -2,  2,  2],
       [-2,  2, -2, -2],
       [-2,  2, -2,  2],
       [-2, -2,  2, -2]])
```

In [5]:
```
arr = np.random.randn(5,4)
arr.mean(1)      #计算每一列的平均值，输出一个一维数组
```

Out[5]:
```
array([-0.56122201, -0.02396455, -0.43128295, -0.51062888, -0.37531298])
```

In [10]:
```
arr.sum(0)     #计算行轴向的累和,每一列的累和
arr.sum(1)     #计算列轴向的累和，每一行的累和，输出一个一维数组
               #mean,sum可用于计算给定轴向上的统计值，形成一个下降一维度的数组
```

Out[10]:
```
array([-2.24488806, -0.09585819, -1.72513178, -2.04251552, -1.50125194])
```

In [14]:
```
arr = np.arange(8)
arr.cumsum(axis=0)      #cumsum和cumprod不会聚合，它们产生一个中间结果
                        #即0位=0+0，1位=0+1，2位=0+1+2，以此显示累加的过程中间结果，
```

Out[14]:
```
array([ 0,  1,  3,  6, 10, 15, 21, 28])
```

In [15]:
```
arr = np.array([[0,1,2],[3,4,5],[6,7,8]])
arr.cumsum(axis=0)     #每一列元素的累和，返回相同长度的数组，但是可以在指定轴向上根
```

Out[15]:
```
array([[ 0,  1,  2],
       [ 3,  5,  7],
       [ 9, 12, 15]])
```

In [16]:
```
arr.cumprod(axis=1)     #每一行元素的累积，在谁的位置上即到谁时的乘积
```

Out[16]:
```
array([[  0,   0,   0],
       [  3,  12,  60],
       [  6,  42, 336]])
```

In [17]:
```
arr = np.random.randn(100)
(arr > 0).sum()     #由于true=1,false=0,即可以利用布尔累加计算100中大于0的数的个数
```

Out[17]:
```
55
```

In [18]:
```
bools = np.array([False,False,True,False])
bools.any()     #any相当于全部析取，有一个true就是true
bools.all()     #all相当于全部合取，有一个false就是false
```

Out[18]:
```
False
```

In [20]:
```
arr = np.random.randn(6)
arr.sort()     #numpy中也可以用sort排序（升序）
arr
```

Out[20]:
```
array([-0.7615957 , -0.59919173, -0.41871271,  0.3579097 ,  0.60990547,
        0.70176918])
```

In [21]:
```
arr = np.random.randn(5,3)
arr.sort(1)     #沿着行排序，sort返回的是已经排序好的数组拷贝，不是数组
arr
```

Out[21]:
```
array([[-1.92664999, -0.75376617,  1.79608887],
       [-0.40380621,  0.17680378,  1.15586677],
       [-0.72206324, -0.64370996, -0.03448301],
       [-1.02897135,  0.51902556,  0.8116753 ],
       [-1.03845051, -0.95772553,  1.03831763]])
```

In [22]:
```
large_arr = np.random.randn(1000)
large_arr.sort()
large_arr[int(0.05 * len(large_arr))]  #计算出排序后的数组位置的索引，并将该索引应用
```

Out[22]:
```
-1.5742827531086296
```

In [23]:
```
names = np.array(['Bob','Joe','Will','Bob','Will','Joe','Joe'])
np.unique(names)     #unique选出数组中的唯一值，并排序
```

Out[23]:
```
array(['Bob', 'Joe', 'Will'], dtype='<U4')
```

In [24]:
```
values = np.array([6,0,0,3,2,5,6])
np.in1d(values, [2,3,6])      #in1d计算values中的元素是否包含在236中，返回一个布尔数
                              #setdiff1d(x,y)    差集，在x中但不在y中的x的元素
                              #setxor1d(x,y)    异或集，在x或y中，但是不属于x和y的交
                              #intersect1d(x,y)    计算x和y的交集，并排序
                              #union1d(x,y)    计算x和y的并集，并排序
                              #都是相应英文加上1d
```

Out[24]:
```
array([ True, False, False,  True,  True, False,  True])
```

In [27]:
```
arr = np.arange(10)
np.save('some_array',arr)      #以.npy的格式压缩格式进行存储
np.load('some_array.npy')       #硬盘上的数组进行载入，名称后要加.npy
```

Out[27]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [28]:
```
np.savez('array_archive.npz', a = arr, b = arr)    #savez将数组作为参数传递给该函数，
arch =  np.load('array_archive.npz')      #载入文件时，获得一个字典型对象
arch['b']     #通过对象可以方便地载入索引对应的数组
              #np.savez_compressed()可以将已经压缩好的文件直接存入，不用再压缩
```

Out[28]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [29]:
```
x = np.array([[1,2,3],[4,5,6]])
y = np.array([[6,23],[-1,7],[8,9]])
x.dot(y)     #矩阵的点乘，即矩阵相乘
```

Out[29]:
```
array([[ 28,  64],
       [ 67, 181]])
```

In [30]:
```
np.dot(x,np.ones(3))    #二维数组和一个长度（列数）相同的数组之间的乘积，结果是一个一
```

Out[30]:
```
array([ 6.,  15.])
```

In [32]:
```
x @ np.ones(3)     # @ 也表示点乘
```

Out[32]:
```
array([ 6.,  15.])
```

In [82]:
```
from numpy.linalg import inv,qr
X = np.random.randint(0,25,size=(5,5))     #意味着生成一个形状为 (5, 5) 的矩阵，其中
mat = X.T.dot(X)     #与本身倒置矩阵相乘会形成方阵
mat
inv(mat)    # inv计算方阵的逆矩阵
```

Out[82]:
```
array([[ 0.00725694, -0.00122245,  0.00158742, -0.00084385, -0.01119289],
       [-0.00122245,  0.01572088, -0.01396225,  0.00263609,  0.00458472],
       [ 0.00158742, -0.01396225,  0.01835219, -0.00287976, -0.01060196],
       [-0.00084385,  0.00263609, -0.00287976,  0.00593099, -0.00150978],
       [-0.01119289,  0.00458472, -0.01060196, -0.00150978,  0.02751336]])
```

In [79]:
```
mat
a = mat.dot(inv(mat))     #矩阵与其逆矩阵相乘得到单位矩阵（对角为1）
np.round(a).astype(int)     #矩阵中的元素取整
```

Out[79]:
```
array([[1, 0, 0, 0, 0],
       [0, 1, 0, 0, 0],
       [0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0],
       [0, 0, 0, 0, 1]])
```

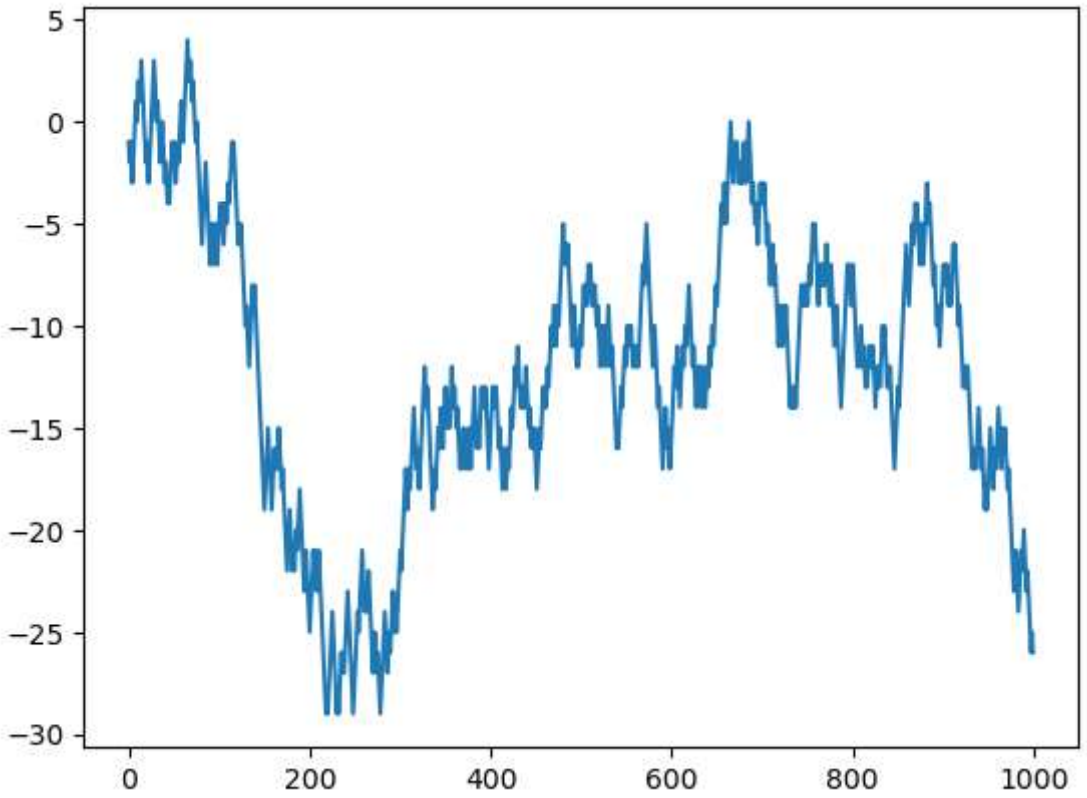
In [83]:
```
q, r = qr(mat)     #计算QR分解，将矩阵分解成一个正交集合，每个列向量都可以用Q（列向量
r
np.round(r).astype(int)
```

Out[83]:
```
array([[-2270, -1016, -1603, -1068, -1446],
       [    0,  -248,  -241,    11,   -47],
       [    0,     0,   -73,  -125,   -40],
       [    0,     0,     0,  -137,     0],
       [    0,     0,     0,     0,    31]])
```

In [ ]:
```
# diag   将方阵对角元素作为一维数组返回
# trace   计算对角元素和
# det   计算矩阵的行列式
# eig   计算方阵的特征值和特征向量
# svd   计算奇异值分解（SVD）
# solve   求解x的线性系统Ax=b，其中A是方阵（求解矩阵方程）
# lstsq   计算Ax=b的最小二乘解
```

In [86]:
```
nsteps = 1000    #进行1000步漫游
draws = np.random.randint(0,2,size=nsteps)    #在这1000步中，随机定义每步为0或1
steps = np.where(draws>0, 1,  -1)    #如果一步为0，则定义为-1（倒退一步），一步为1，
walk = steps.cumsum()    #开始漫游，累加1000次漫步的结果，并显示过程结果
import matplotlib.pyplot as plt
plt.plot(walk[:1000])    #漫游可视化，每次运行结果都不同
```

Out[86]:
```
[<matplotlib.lines.Line2D at 0x194265d07d0>]
```



In [87]:
```
walk.min()    #统计漫步数据中的最小值（倒退到最远的地方）
walk.max()    #统计漫步数据中的最大值（前进到最远的地方）
```

Out[87]:
```
4
```

In [88]:
```
(np.abs(walk) >= 10).argmax()    #找到第一次往一个方向连续走10步的位置
```

Out[88]:
```
129
```

In [1]:
```

```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 1
----> 1 xelatex

NameError: name 'xelatex' is not defined
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: