Table of Contents ¶

In [187]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

numpy重点:

数组的属性
数组的创建:内置的一些方法
数组的索引
数组形态变换
数组运算:广播机制
数组常用统计函数

1 查看版本信息

In [1]:
```
import numpy as np      #导入numpy并简写为np
import pandas as pd     #导入pandas并简写为pd
```

In [2]:
```
print(np.__version__)#查看版本
```

```
1.20.3
```

2 数组属性

2.1 一维数组

In [3]:
```
a=np.array([1,2,3,4])#把列表转换为数组
a
```

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

In [4]:
```
type(a)#类型,自己定义的数据类型,所以用numpy.
```

Out[4]:
```
numpy.ndarray
```

In [5]:
```
a.ndim#查看维度(维数,是一维数组还是几维数组)
```

Out[5]:
```
1
```

In [7]:
```
a.shape  #形状,尺寸  (4,)这里面只有一个数字,代表一维数组,4代表有4个元素
```

Out[7]:
```
(4,)
```

In [8]:
```
a
```

Out[8]:
```
array([1, 2, 3, 4])
```

In [9]:
```
a.T  #做一个转置,结果跟原来的结果一样,因为一维数组不分行和列
```

Out[9]:
```
array([1, 2, 3, 4])
```

In [21]:
```
a.size  #元素数量
```

Out[21]:
```
4
```

In [11]:
```
a.dtype  #元素类型
#代表数组中的元素的数据类型,int代表整数型,32代表这个数据只能存储32位的宽度
#会空出一位预留符号位,实际存储31位,所以范围在-2^31到2^31(-2的31次方到2的31次方)
#一个字节是8位
```

Out[11]:
```
dtype('int32')
```

In [12]:
```
a.itemsize  #元素的大小，字节为单位（一个字节8位），
#所以对于a来说,一个元素占一个字节,4个元素占4个字节,输出4  ,
```

Out[12]:
```
4
```

In [10]:
```
b=np.array([1,2.0,3,4])   #转换成数组
b        #数据类型要一样,所以输出后都会转换成浮点数,所以每个数后面都会加个点
```

Out[10]:
```
array([1., 2., 3., 4.])
```

In [11]:
```
b.dtype   #64位的浮点数
```

Out[11]:
```
dtype('float64')
```

In [12]:
```
b.itemsize
```

Out[12]:
```
8
```

In [ ]:

2.2 多维数组

大于等于2就称为多维数组

In [13]:
```
a=np.array([[1,2,3,4],[5,6,7,8],[9,11,12,13]])    #二维列表转换为数组
a
#多维数组就会有行和列.   这个数组是3(行)*4(列)的一个二维列表
#行,被称为0轴,正方向向下(类似于纵轴),   列,被称为1轴,正方向向右(类似于横轴)
#沿着行进行计算,其实是算列的值,沿着列进行计算,其实是算行的值
```

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

In [14]:
```
a.ndim#查看维度
```

Out[14]: 2

In [15]:
```
a.shape
```

Out[15]: (3, 4)

In [16]:
```
a.size
```

Out[16]: 12

In [17]:
```
a=np.array([[[1,2,3,4],[5,6,7,8],[9,11,12,13]],[[1,2,3,4],[5,6,7,8],[9,11,12,13]]])#二维列表转换为数组
a
```

Out[17]:
```
array([[[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 11, 12, 13]],

       [[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 11, 12, 13]]])
```

In [18]:
```
a.ndim#查看维度,三维(类似于空间)
```

Out[18]: 3

In [19]:
```
a.shape  # 查看数组的形状大小,是一个2*3*4的三维数组
```

Out[19]: (2, 3, 4)

2.3 函数和方法的理解

3 数组的创建

3.1 基础数据类型转换为数组

In [ ]:
```
#列表，元组，range对象等转换为数组
```

In [2]:
```
np.array([1,2,3,4])
```

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

In [3]:
```
a=np.array((1,2,3,4))#把元组转换为数组
a
```

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

In [4]:
```
np.array(range(5))#range对象转换成数组
```

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

3.2 数组内置的一些方法

3.2.1 np.arange

In [20]:
```
np.arange(8)  #有点类似于range函数
```

Out[20]:
```
array([0, 1, 2, 3, 4, 5, 6, 7])
```

In [21]:
```
range(8)  #可迭代对象 从0到7的一列数,左闭右开
```

Out[21]:
```
range(0, 8)
```

In [22]:
```
np.arange(0,1,0.2)  #生成数组,左闭右开.  第一个参数是开始值\第二个参数是结束值\第三个参数是步长值
```

Out[22]:
```
array([0. , 0.2, 0.4, 0.6, 0.8])
```

3.2.2 np.linspace

In [23]:
```
np.linspace(0,10,21)
#等差数组，第一个参数是起始数,第二个参数是终止数，第三个参数是这个数组内元素的个数
#整体闭区间,左闭右闭的区间
```

Out[23]:
```
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5,  5. ,
        5.5,  6. ,  6.5,  7. ,  7.5,  8. ,  8.5,  9. ,  9.5, 10. ])
```

In [24]:
```
np.linspace(0,1,6)  #从0开始,到1结束,步长为0.2的一组等差数组,所以总共6个数,后面第三个参数写6
```

Out[24]:
```
array([0. , 0.2, 0.4, 0.6, 0.8, 1. ])
```

3.2.3 np.logspace

In [25]:
```
np.logspace(0,100,11)    #等比数组，默认以10位底数
#等比数组的指数是等差数列
#第一个参数是指数从0开始,第二个参数是到指数为100的时候截止,底数默认为10
#第三个参数是这个数组里元素的个数,通过个数计算每个指数之间的差值,指数是符合等差数列的
```

Out[25]:
```
array([1.e+000, 1.e+010, 1.e+020, 1.e+030, 1.e+040, 1.e+050, 1.e+060,
       1.e+070, 1.e+080, 1.e+090, 1.e+100])
```

In [26]:
```
np.logspace(1,6,6,base=2) #后面的base=2代表指定了底数为2,这是一个以2为底数且指数从1到6的数组,总共有6个元素
```

Out[26]:
```
array([ 2.,  4.,  8., 16., 32., 64.])
```

3.2.4 np.zeros,np.ones

In [5]:
```
np.zeros(3)  #全零的一维数组
```

Out[5]:
```
array([0., 0., 0.])
```

In [6]:
```
np.ones(3)#全一的一维数组
#全零或全一的数组可以用来初始化,实际就是进行占位,后期根据需要进行修改
```

Out[6]:
```
array([1., 1., 1.])
```

In [37]:
```
a=np.ones((3,3))#全一的二维数组
a
```

Out[37]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

3.2.5 np.zeros_like

In [39]:
```
a=np.array([[1,2,3,4],[6,7,8,9]])
a
```

Out[39]:
```
array([[1, 2, 3, 4],
       [6, 7, 8, 9]])
```

In [55]:
```
np.zeros_like(a) #生成一个和a的shape一样的全零的数组
```

Out[55]:
```
array([[0, 0, 0, 0],
       [0, 0, 0, 0]])
```

In [56]:
```
np.ones_like(a) #生成一个和a的shape一样的全一的数组
```

Out[56]:
```
array([[1, 1, 1, 1],
       [1, 1, 1, 1]])
```

3.2.6 np.full

In [40]:
```
np.full((3,2),6)   #全为某个值的数组,full((3,2),6) 代表生成一个3*2的值全部为6的二维数组
```

Out[40]:
```
array([[6, 6],
       [6, 6],
       [6, 6]])
```

3.2.7 np.identity,np.eye

In [60]:
```
np.identity(3)   #单位数组 ， 单位矩阵
```

Out[60]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [61]:
```
np.eye(3)  #单位数组 ， 单位矩阵,以上两种方法都可以
```

Out[61]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

3.2.8 np.diag

In [62]:
```
np.diag([1,2,3,4]) #对角数组 ， 这个是自己设定的值,所以需要将对角线上的值写入参数的位置
```

Out[62]:
```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

In [27]:
```
np.diag?  #diag有两个参数
```

In [28]:
```
np.diag([1,2,3,4],k=1)
#k为整数,
#若k的值为正数是在左侧和下方各增加对应数字的列数和行数,值全部为0
#若k的值为负数,则是在右侧和上方各增加对应数字的列数和行数,值全部为0
```

Out[28]:
```
array([[0, 1, 0, 0, 0],
       [0, 0, 2, 0, 0],
       [0, 0, 0, 3, 0],
       [0, 0, 0, 0, 4],
       [0, 0, 0, 0, 0]])
```

In [30]:
```
np.diag([1,2,3,4],k=2)
```

Out[30]:
```
array([[0, 0, 1, 0, 0, 0],
       [0, 0, 0, 2, 0, 0],
       [0, 0, 0, 0, 3, 0],
       [0, 0, 0, 0, 0, 4],
       [0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0]])
```

In [29]:
```
np.diag([1,2,3,4],k=-1)
```

Out[29]:
```
array([[0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0],
       [0, 2, 0, 0, 0],
       [0, 0, 3, 0, 0],
       [0, 0, 0, 4, 0]])
```

In [31]:
```
np.diag([1,2,3,4],k=-2)
```

Out[31]:
```
array([[0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0],
       [0, 2, 0, 0, 0, 0],
       [0, 0, 3, 0, 0, 0],
       [0, 0, 0, 4, 0, 0]])
```

3.2.9 np.empty

In [41]:
```
np.empty((6,3)) #空数组，元素取值随机 ， 用来占位
```

Out[41]:
```
array([[1.07097411e-311, 5.13828272e-322, 0.00000000e+000],
       [0.00000000e+000, 1.69119873e-306, 1.15998412e-028],
       [2.44171989e+232, 8.00801729e+159, 6.19319847e-071],
       [1.05161522e-153, 6.01391519e-154, 1.05208577e-153],
       [1.19683897e+141, 3.77780862e+180, 1.15998412e-028],
       [6.48224638e+170, 3.67145870e+228, 1.60945238e+295]])
```

3.2.10 随机数的产生：random模块

In [36]:
```
np.random.randint(0,10,5)  #随机整数，从0到10之间但是不包括10的5个随机整数,左闭右开
```

Out[36]:
```
array([5, 7, 2, 9, 2])
```

In [38]:
```
np.random.rand(5)          #0到1之间随机浮点数,左闭右开
```

Out[38]:
```
array([0.79555499, 0.70792983, 0.04756526, 0.43053013, 0.38818561])
```

In [55]:
```
np.random.seed(1)    #seed是固定种子，表示随机数生成的方式,当seed()的参数固定为某一个值的时候每次执行后输出的结果都一样
np.random.rand(5)    #seed()后面的参数范围是1到2^32-1
#rand(5)表示执行的结果输出5个数
```

Out[55]:
```
array([4.17022005e-01, 7.20324493e-01, 1.14374817e-04, 3.02332573e-01,
       1.46755891e-01])
```

In [47]:
```
np.random.seed(2)
np.random.rand(6)
```

Out[47]:
```
array([0.4359949 , 0.02592623, 0.54966248, 0.43532239, 0.4203678 ,
       0.33033482])
```

In [99]:
```
np.random.standard_normal(size=(4,))
#一维数组，标准正态分布,size=(4,)参数里只有一个数字代表是一维数组,4代表这个数组里有4个元素
```

Out[99]:
```
array([-0.88778575, -1.98079647, -0.34791215,  0.15634897])
```

In [100]:
```
np.random.standard_normal(size=(4,4))   #二维数组，标准正态分布
#每一行符合正态分布,每一列也符合正态分布
```

Out[100]:
```
array([[ 1.23029068,  1.20237985, -0.38732682, -0.30230275],
       [-1.04855297, -1.42001794, -1.70627019,  1.9507754 ],
       [-0.50965218, -0.4380743 , -1.25279536,  0.77749036],
       [-1.61389785, -0.21274028, -0.89546656,  0.3869025 ]])
```

In [57]:
```
np.random.normal(0.5,0.1,size=(3, 4)) #均值，标准差，shape,       正态分布
```

Out[57]:
```
array([[0.38940649, 0.33454845, 0.26365314, 0.61353453],
       [0.39829859, 0.56373618, 0.41400934, 0.67726076],
       [0.38896369, 0.51812143, 0.55643449, 0.44334898]])
```

4 数据类型

4.1 数据类型的转换

In [6]:
```
np.float64(43)  #浮点数转换
```

Out[6]:
```
43.0
```

In [7]:
```
np.int32(78.0)
```

Out[7]:
```
78
```

In [48]:
```
a=np.array([1,2,3])
a.dtype
```

Out[48]:
```
dtype('int32')
```

In [49]:
```
a.astype(np.int64)  #数组类型的转换     新生成一个数组,值不变,但是数据类型变成了int64
```

Out[49]:
```
array([1, 2, 3], dtype=int64)
```

In [50]:
```
a=np.array([1,2,3,6,7,11,12,13,14])
a**a
#因为默认是32位的位宽,当位宽超过了32位的时候会出现数据结果的错误
```

Out[50]:
```
array([          1,           4,          27,       46656,      823543,
        1843829075,  -251658240, -1692154371, -1282129920], dtype=int32)
```

In [51]:
```
a=np.array([1,2,3,6,7,11,12,13,14])
b=a.astype(np.int64)
b**b  #进行数组类型的转换,转换成64位的位宽后会容纳更多的位数
```

Out[51]:
```
array([                1,                 4,                27,
                   46656,            823543,       285311670611,
            8916100448256,    302875106592253, 11112006825558016],
      dtype=int64)
```

In [ ]:
```

```

4.2 缺失值:特殊的浮点型

None #在Jupyter里代表空 在Numpy中缺失值用np.nan表示

#需要掌握的知识点 1.缺失的判断 2.缺失值进行任何运算的结果都为缺失值

In [52]:
```
#缺失值，空值
a=np.nan
a
```

Out[52]: nan

In [53]:
```
type(a)       #特殊的浮点型
```

Out[53]: float

In [54]:
```
np.isnan(a)     #空值的判断，只能这样判断  ,是空值返回True,不是空值返回False
```

Out[54]: True

In [69]:
```
a=np.nan
a==np.nan    #不能这样判断,缺失值不等于任何一个东西
```

Out[69]: False

In [55]:
```
np.nan==np.nan   #缺失值不等于任何一个东西,所以不能用==来定义,就会返回False
```

Out[55]: False

In [56]:
```
2*np.nan   #空值任何运算的结果均为空值
```

Out[56]: nan

In [57]:
```
np.nan-np.nan
```

Out[57]: nan

5 数组元素的索引

5.1 一维数组的索引

In [58]:
```
a=np.random.randint(0,10,5)#一维数组索引  与列表索引类似
a
```

Out[58]:
```
array([5, 4, 4, 5, 7])
```

In [59]:
```
a[0]
```

Out[59]:
```
5
```

In [60]:
```
a[-1]
```

Out[60]:
```
7
```

In [61]:
```
a[1::2]    #开始，结束，步长，左闭右开
```

Out[61]:
```
array([4, 5])
```

#索引和切片的时候一般是中括号,用函数的时候一般是圆括号

5.2 多维数组的索引

In [62]:
```
a = np.array([[1, 2, 3, 4, 5],[6, 7, 8,9,10], [17, 18, 19, 20, 21]])
a
```

Out[62]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [17, 18, 19, 20, 21]])
```

In [63]:
```
a[0,3:5]  #索引第0行中第3和第4列的元素,, 第一个参数是0轴(纵轴)的参数,第二个参数是1轴(横轴)的参数
#逗号前面是0轴,数字为0代表取索引位置为0的行(即第1行),    逗号后面是1轴,3:5代表索引位置起始位3,终止位置为5,步长没写默认为1
```

Out[63]:
```
array([4, 5])
```

In [64]:
```
a[1:,2:] #索引第2和第3行中第3列、第4列和第5列的元素
#代表取索引位置为1的行(也就是第2行)到剩余全部的行,列取索引位置为2的列(即第3列)到剩下所有的列,然后取它们的公共部分
```

Out[64]:
```
array([[ 8,  9, 10],
       [19, 20, 21]])
```

In [65]:
```
a[::2,::2]
```

Out[65]:
```
array([[ 1,  3,  5],
       [17, 19, 21]])
```

In [66]:
```
a
```

Out[66]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [17, 18, 19, 20, 21]])
```

In [67]:
```
a[::2,(0,2,3)]
```

Out[67]:
```
array([[ 1,  3,  4],
       [17, 19, 20]])
```

In [68]:
```
a
```

Out[68]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [17, 18, 19, 20, 21]])
```

In [69]:
```
a[(0,0,1,2),(0,1,1,2)] #从两个序列的对应位置取出两个整数组成下标 花式索引
#逗号前代表行,逗号后代表列
#行的第一个数和列的第一个数组成一个位置,这个位置是索引为0的行与索引为0的列相交的那个位置,输出这个位置对应的值
#按照点坐标来理解,a[纵坐标,横坐标]
#剩下的以此类推
```

Out[69]:
```
array([ 1,  2,  7, 19])
```

In [70]:
```
a=np.arange(18).reshape(2,3,3)
a  #三维数组,reshape的第一个参数代表层级为0轴,第二个参数代表行是1轴,第三个参数代表列是2轴
```

Out[70]:
```
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8]],

       [[ 9, 10, 11],
        [12, 13, 14],
        [15, 16, 17]]])
```

In [71]:
```
a[0][0,:]
```

Out[71]:
```
array([0, 1, 2])
```

In [78]:
```
a[0,0,:]
```

Out[78]:
```
array([0, 1, 2])
```

In [72]:
```
a[1][0,:]
```

Out[72]:
```
array([ 9, 10, 11])
```

In [79]:
```
a[1,0,:]
```

Out[79]:
```
array([ 9, 10, 11])
```

In [73]:
```
a[0][:,(1)]
```

Out[73]:
```
array([1, 4, 7])
```

In [82]:
```
a[0,:,1]
```

Out[82]:
```
array([1, 4, 7])
```

In [83]:
```
a[0,:,(1)]
```

Out[83]:
```
array([1, 4, 7])
```

In [74]:
```
a[1][:,(1)]
```

Out[74]:
```
array([10, 13, 16])
```

In [84]:
```
a[1,:,1]
```

Out[84]:
```
array([10, 13, 16])
```

In [75]:
```
a[:,0,:]
```

Out[75]:
```
array([[ 0,  1,  2],
       [ 9, 10, 11]])
```

In [76]:
```
a[:,:,1]
```

Out[76]:
```
array([[ 1,  4,  7],
       [10, 13, 16]])
```

In [77]:
```
a[:,::2,::2]
```

Out[77]:
```
array([[[ 0,  2],
        [ 6,  8]],

       [[ 9, 11],
        [15, 17]]])
```

6 变换数组的形态

6.1 reshape

In [85]:
```
#变换数组的形态
a=np.arange(12)
a
```

Out[85]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

In [86]:
```
a.reshape(3,4)  #元素个数要一致，是一个新生成的数组
#将上面一维的数组形态变换成3*4的一个二维数组的形态
```

Out[86]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [89]:
```
a  #原数据不改变
```

Out[89]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

In [87]:
```
a.reshape(3,5)
#报错: 原因是3*5的数组需要15个元素,而原数据只有12个元素,不够,会导致行列元素个数不一致的情况,不符合转换格式要求
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_27704/1214200498.py in <module>
----> 1 a.reshape(3,5)

ValueError: cannot reshape array of size 12 into shape (3,5)
```

In [88]:
```
a.reshape(2,6)    #2*6的数组的个数是12个,可以满足行列元素一致的要求
```

Out[88]:
```
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]])
```

In [92]:
```
a.reshape(-1,2)  #-1表示自动计算, 要求能够整除
#第一个参数代表分成几行,第二参数代表分成几列,如果有一个为负数,则另一个必须是元素总个数的约数,要求可以整除
#如果第一个参数为正,第二个参数为负,则第二个参数没有实质性的意义,默认用总数/行数的结果去匹配列数
#如果第一个参数为负,第二个参数为正,则第一个参数没有实质性的意义,默认用总数/列数的结果去匹配行数     这就是自动计算的解释
```

Out[92]:
```
array([[ 0,  1],
       [ 2,  3],
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11]])
```

In [102]:
```
a.reshape(-2,2)
```

Out[102]:
```
array([[ 0,  1],
       [ 2,  3],
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11]])
```

In [103]:
```
a.reshape(3,-2)
```

Out[103]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

6.2 resize

In [90]:
```
a
```

Out[90]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

In [92]:
```
a.resize(3,4)  #原地修改, 原数据会进行改变,也就是最初的值被修改之后就没有了,而且不需要给a.resize(3,4)定义一个变量
```

In [93]:
```
a
```

Out[93]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [94]:
```
b=a.resize(3,4)
b   #执行后没有输出,代表b为空的,因为a.resize(3,4)是原地操作,不需要赋值一个变量
```

In [95]:
```
a
```

Out[95]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

6.3 flatten

In [96]:
```
a
```

Out[96]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [97]:
```
a.flatten()#展平，降维，横着拼
```

Out[97]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

In [98]:
```
a.flatten(order='F')#展平，降维，竖着拼
```

Out[98]:
```
array([ 0,  4,  8,  1,  5,  9,  2,  6, 10,  3,  7, 11])
```

In [99]:
```
a.reshape(1,12)  # 改变数组的行列数,代表1行12列
```

Out[99]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11]])
```

In [101]:
```
a.reshape(3,4)  #改变数组的行列数,代表3行4列
```

Out[101]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

6.4 ravel

In [104]:
```
a
```

Out[104]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [105]:
```
a.ravel()#展平,它没有参数,只能横着拼
```

Out[105]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

6.5 swapaxes

In [106]:
```
a=np.arange(12).reshape(3,4)
a
```

Out[106]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [107]:
```
a.swapaxes(0,1)  #0轴和1轴交换  相当于做了一个转置
```

Out[107]:
```
array([[ 0,  4,  8],
       [ 1,  5,  9],
       [ 2,  6, 10],
       [ 3,  7, 11]])
```

In [108]:
```
a=np.arange(12).reshape(3,2,2)
a
```

Out[108]:
```
array([[[ 0,  1],
        [ 2,  3]],

       [[ 4,  5],
        [ 6,  7]],

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

In [109]:
```
a.swapaxes(1,2)  #1轴和2轴互换,三维数组1轴是行,2轴是列
```

Out[109]:
```
array([[[ 0,  2],
        [ 1,  3]],

       [[ 4,  6],
        [ 5,  7]],

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

6.6 stack

In [110]:
```
a=np.arange(12).reshape(3,4)
a
```

Out[110]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [111]:
```
b=a*3
b
```

Out[111]:
```
array([[ 0,  3,  6,  9],
       [12, 15, 18, 21],
       [24, 27, 30, 33]])
```

In [112]:
```
np.hstack([a,b])#横向堆叠,b拼在a的右边,行数需要一样
```

Out[112]:
```
array([[ 0,  1,  2,  3,  0,  3,  6,  9],
       [ 4,  5,  6,  7, 12, 15, 18, 21],
       [ 8,  9, 10, 11, 24, 27, 30, 33]])
```

In [113]:
```
np.hstack([b,a])  #横向堆叠,b拼在a的左边
```

Out[113]:
```
array([[ 0,  3,  6,  9,  0,  1,  2,  3],
       [12, 15, 18, 21,  4,  5,  6,  7],
       [24, 27, 30, 33,  8,  9, 10, 11]])
```

In [114]:
```
np.vstack([a,b])#纵向堆叠,列数需要一样,b拼在a的下边
```

Out[114]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [ 0,  3,  6,  9],
       [12, 15, 18, 21],
       [24, 27, 30, 33]])
```

In [115]:
```
np.vstack([b,a])  #纵向堆叠,列数需要一样,b拼在a的上边
```

Out[115]:
```
array([[ 0,  3,  6,  9],
       [12, 15, 18, 21],
       [24, 27, 30, 33],
       [ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

6.7 concatenate

In [116]:
```
np.concatenate([a,b],axis = 1)  #concatenate函数横向组合,沿着1轴的方向拼接,右拼接(b拼接到a的右边)
```

Out[116]:
```
array([[ 0,  1,  2,  3,  0,  3,  6,  9],
       [ 4,  5,  6,  7, 12, 15, 18, 21],
       [ 8,  9, 10, 11, 24, 27, 30, 33]])
```

In [117]:
```
np.concatenate([a,b],axis = 0)  #concatenate函数纵向组合，沿着0轴的方向拼接， 下拼接(b拼接到a的下边)
```

Out[117]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [ 0,  3,  6,  9],
       [12, 15, 18, 21],
       [24, 27, 30, 33]])
```

6.8 split

In [118]:
```
#分割
a=np.arange(16).reshape(4,4)
a
```

Out[118]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [119]:
```
np.hsplit(a,2)#等分，返回值为列表，  横向分割,将列分成2部分,形成两个数组,前两列一组,后两列一组,行数不变
```

Out[119]:
```
[array([[ 0,  1],
        [ 4,  5],
        [ 8,  9],
        [12, 13]]),
 array([[ 2,  3],
        [ 6,  7],
        [10, 11],
        [14, 15]])]
```

In [120]:
```
np.vsplit(a,2)#纵向分割 ， 将行分成2部分,形成两个数组,前两行一组,后两行一组,列数不变
```

Out[120]:
```
[array([[0, 1, 2, 3],
        [4, 5, 6, 7]]),
 array([[ 8,  9, 10, 11],
        [12, 13, 14, 15]])]
```

7 数组的运算

7.1 数组与数值的运算

In [121]:
```
a=np.arange(16).reshape(4,4)
a
```

Out[121]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [122]:
```
a*2  #数组内部每一个元素进行相同的操作,就是乘以2
```

Out[122]:
```
array([[ 0,  2,  4,  6],
       [ 8, 10, 12, 14],
       [16, 18, 20, 22],
       [24, 26, 28, 30]])
```

In [123]:
```
a//2  #整除
```

Out[123]:
```
array([[0, 0, 1, 1],
       [2, 2, 3, 3],
       [4, 4, 5, 5],
       [6, 6, 7, 7]], dtype=int32)
```

In [124]:
```
a%2#取余数，取模
```

Out[124]:
```
array([[0, 1, 0, 1],
       [0, 1, 0, 1],
       [0, 1, 0, 1],
       [0, 1, 0, 1]], dtype=int32)
```

In [125]:
```
a+2
```

Out[125]:
```
array([[ 2,  3,  4,  5],
       [ 6,  7,  8,  9],
       [10, 11, 12, 13],
       [14, 15, 16, 17]])
```

7.2 数组与数组的运算

In [126]:
```
a=np.arange(16).reshape(4,4)
b=np.arange(16).reshape(4,4)
```

In [127]:
```
a
```

Out[127]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [128]:
```
b
```

Out[128]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [129]:
```
a*b
#a和b的形状相同，对应的位置上的元素进行操作,所以必须能够进行一一对应.如果形状不一样,则不能进行一一对应就会进行广播处理
```

Out[129]:
```
array([[  0,   1,   4,   9],
       [ 16,  25,  36,  49],
       [ 64,  81, 100, 121],
       [144, 169, 196, 225]])
```

In [131]:
```
a+b
```

Out[131]:
```
array([[ 0,  2,  4,  6],
       [ 8, 10, 12, 14],
       [16, 18, 20, 22],
       [24, 26, 28, 30]])
```

In [130]:
```
a**a  #a的a次方,  后面会有数据不对的是因为数位不够的原因
```

Out[130]:
```
array([[          1,           1,           4,          27],
       [        256,        3125,       46656,      823543],
       [   16777216,   387420489,  1410065408, 1843829075],
       [ -251658240, -1692154371, -1282129920, 1500973039]], dtype=int32)
```

7.3 数组的广播机制

In [132]:
```
#数组的广播机制
a1 = np.array([[0,0,0],[1,1,1],[2,2,2],[3,3,3]])
print(a1.shape)
a2 = np.array([1,2,3])
print(a2.shape)
```

```
(4, 3)
(3,)
```

In [134]:
```
a1
```

Out[134]:
```
array([[0, 0, 0],
       [1, 1, 1],
       [2, 2, 2],
       [3, 3, 3]])
```

In [135]:
```
a2
```

Out[135]:
```
array([1, 2, 3])
```

In [136]:
```
a1+a2
```

Out[136]:
```
array([[1, 2, 3],
       [2, 3, 4],
       [3, 4, 5],
       [4, 5, 6]])
```

In [137]:
```
a1 = np.array([[0,0,0],[1,1,1],[2,2,2],[3,3,3]])
print(a1.shape)
a2 = np.array([1,2,3,4]).reshape(4,1)
print(a2.shape)
```

```
(4, 3)
(4, 1)
```

In [138]:
```
a1
```

Out[138]:
```
array([[0, 0, 0],
       [1, 1, 1],
       [2, 2, 2],
       [3, 3, 3]])
```

In [139]:
```
a2
```

Out[139]:
```
array([[1],
       [2],
       [3],
       [4]])
```

In [140]:
```
#4x3
a1 + a2
```

Out[140]:
```
array([[1, 1, 1],
       [3, 3, 3],
       [5, 5, 5],
       [7, 7, 7]])
```

In [142]:
```
a1 = np.array([1,2,3,4])
print(a1.shape)
a2=np.array([4,5,6,7]).reshape(4,1)
print(a2.shape)
a1+a2
```

```
(4,)
(4, 1)
```

Out[142]:
```
array([[ 5,  6,  7,  8],
       [ 6,  7,  8,  9],
       [ 7,  8,  9, 10],
       [ 8,  9, 10, 11]])
```

In [204]:
```
a1 = np.array([[0,0,0],[1,1,1],[2,2,2],[3,3,3]])
print(a1.shape)
a2 = np.array([1,2,3]).reshape(3,1)
print(a2.shape)
#进行计算的话会报错,因为0轴和1轴的长度都不一样
```

```
(4, 3)
(3, 1)
```

In [205]:
```
a1
a2
```

Out[205]:
```
array([[0, 0, 0],
       [1, 1, 1],
       [2, 2, 2],
       [3, 3, 3]])
```

Out[205]:
```
array([[1],
       [2],
       [3]])
```

In [145]:
```
a1+a2 #会报错,因为上面数组0轴和1轴的长度都不一样,无法计算
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_27704/4064069160.py in <module>
----> 1 a1+a2

ValueError: operands could not be broadcast together with shapes (4,3) (3,1)
```

8 numpy数据的保存和读取

8.1 一个数组情况

In [146]:
```
arr1 = np.arange(9).reshape(3,3)  #创建一个数组,一个3*3的数组
np.save('a1.npy',arr1)#保存的是单个数组  ,  后缀npy是固定的,没有写保存路径就是保存到当前的路径,查看当前路径可以用import
#想要保存到指定路径,只需要在a1.npy前面引号里面加上要保存的路径的地址即可
#a1是文件名,  .npy是后缀
```

In [147]:
```
import os
os.getcwd()  #获取当前工作路径的代码
```

Out[147]: 'C:\\Users\\99253\\Desktop\\CDA数据分析\\课件\\Pyhon\\Python资料\\正课~函数,面向对象,连接SQL\\数据清洗课件和代码\\代码'

In [148]:
```
b=np.load('a1.npy')#读取文件, 如果保存到其他路径了,也是在a1.npy前面引号里面加上要保存的路径的地址即可
b
```

Out[148]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

8.2 多个数组情况

In [149]:
```
a1 = np.arange(12).reshape(3,4)
a2 = np.arange(10)
np.savez('arr12.npz',a1=a1,a2=a2)#保存的是两个数组到一个文件中(把这两个数组都保存在同一个文件中)
#arr12是文件名,  .npz是后缀
```

In [150]:
```
arr12=np.load('arr12.npz')  #读取
```

In [151]:
```
arr12['a1']
```

Out[151]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [152]:
```
arr12['a2']
```

Out[152]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

9 使用numpy进行统计分析

9.1 排序

In [153]:
```
np.random.seed(42) #设置随机种子，固定种子
a = np.random.randint(1,10,size = 10)  #生成随机数1~10
a
```

Out[153]:
```
array([7, 4, 8, 5, 7, 3, 7, 8, 5, 4])
```

In [154]:
```
a.sort() #原地排序，没有返回值，    默认从小到大排序
```

In [155]:
```
a
```

Out[155]:
```
array([3, 4, 4, 5, 5, 7, 7, 7, 8, 8])
```

In [206]:
```
a = np.random.randint(1,10,size=5) #生成随机数1~10，shape为10
a
```

Out[206]:
```
array([9, 4, 1, 2, 1])
```

In [207]:
```
a.argsort()      #从小到大排序，返回其相对应的索引值
#从小到大排序之后看这个数字在原数据的位置,显示原数据位置的索引值
#排序之后的顺序是 1  1  2  4  9  ,这里第一个1在原数据的位置是第3个数,索引值为2,第二个1在原数据的位置是第5个数,索引值为4
#以此类推,输出他们的索引值为 2  4  3  1  0
```

Out[207]:
```
array([2, 4, 3, 1, 0], dtype=int64)
```

9.2 去重复和重复

In [159]:
```
a = np.random.randint(1,10,size = 10) #生成随机数1~10，shape为10
a
```

Out[159]:
```
array([2, 8, 6, 2, 5, 1, 6, 9, 1, 3])
```

In [160]:
```
np.unique(a)#去重复(去重)
```

Out[160]:
```
array([1, 2, 3, 5, 6, 8, 9])
```

In [161]:
```
a=np.arange(10)
a
```

Out[161]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [162]:
```
np.tile(a,3) #对整个数组进行重复,后面的参数3代表整体重复3次,全部的数作为一个整体
```

Out[162]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1,
       2, 3, 4, 5, 6, 7, 8, 9])
```

In [163]:
```
a.repeat(3)#对单个的元素进行重复,即每个元素都重复3次,一个重复结束再到另一个
```

Out[163]:
```
array([0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 7,
       7, 7, 8, 8, 8, 9, 9, 9])
```

9.3 填充

In [164]:
```
#np.pad(数组，边界宽度，常量，常量值)，用常量值填充边界
a=np.ones((2,2))
a
```

Out[164]:
```
array([[1., 1.],
       [1., 1.]])
```

In [165]:
```
np.pad(a,pad_width=2,mode='constant',constant_values=0)
#括号里面的参数的解释
#a代表往哪个数组里填充
#pad_width代表填充的宽度,上下左右增加几行几列
#mode='constant',constant_values=0   代表用数字0进行填充
```

Out[165]:
```
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 1., 0., 0.],
       [0., 0., 1., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

9.4 常用统计函数

In [166]:
```
a=np.arange(20).reshape(4,5)
a
```

Out[166]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

In [167]:
```
a.sum()  #调用函数进行求和,将数组里面的数全部加起来,即对数组里的所有数进行求和
```

Out[167]: 190

In [168]:
```
np.sum([1,2,3])  #列表也可以用求和函数,求和函数是通用的
```

Out[168]: 6

In [169]:
```
a.sum()  #优先考虑调用方法,代码运行更方便快捷
```

Out[169]: 190

In [170]:
```
a.sum(axis=0)#纵轴求和
```

Out[170]: array([30, 34, 38, 42, 46])

In [171]:
```
a.sum(axis=1)#横轴求和
```

Out[171]: array([10, 35, 60, 85])

In [172]:
```
a.mean()        #求平均,不指定参数默认求所有数的平均值
```

Out[172]: 9.5

In [173]:
```
a.mean(axis=0)#求平均，求按纵轴的数的平均值(即每一列的平均值)
```

Out[173]: array([7.5, 8.5, 9.5, 10.5, 11.5])

In [174]:
```
a.mean(axis=1)#求平均，求按横轴的数的平均值(即每一行的平均值)
```

Out[174]: array([2., 7., 12., 17.])

In [175]:
```
a.var()#方差  不指定参数默认求所有数的方差
```

Out[175]: 33.25

In [176]:
```
a.std()#标准差  不指定参数默认求所有数的标准差
```

Out[176]: 5.766281297335398

In [177]:
```
a.max()#最大值    不指定参数默认求所有数中的最大值
```

Out[177]: 19

In [178]:
```
a.min()#最小值  不指定参数默认求所有数中的最小值
```

Out[178]:
```
0
```

In [181]:
```
a=np.random.randint(1,10,5)  #生成从1到10的随机数,5个数
a
```

Out[181]:
```
array([3, 7, 5, 9, 7])
```

In [220]:
```
a=np.random.randint(1,10,5)
a
```

Out[220]:
```
array([6, 8, 2, 5, 1])
```

In [221]:
```
a.argmin()   #求最小元素的索引
```

Out[221]:
```
4
```

In [222]:
```
a.argmax()#求最大元素的索引
```

Out[222]:
```
1
```

In [184]:
```
a=np.arange(2,10)
a
```

Out[184]:
```
array([2, 3, 4, 5, 6, 7, 8, 9])
```

In [185]:
```
a.cumsum()#累计和
```

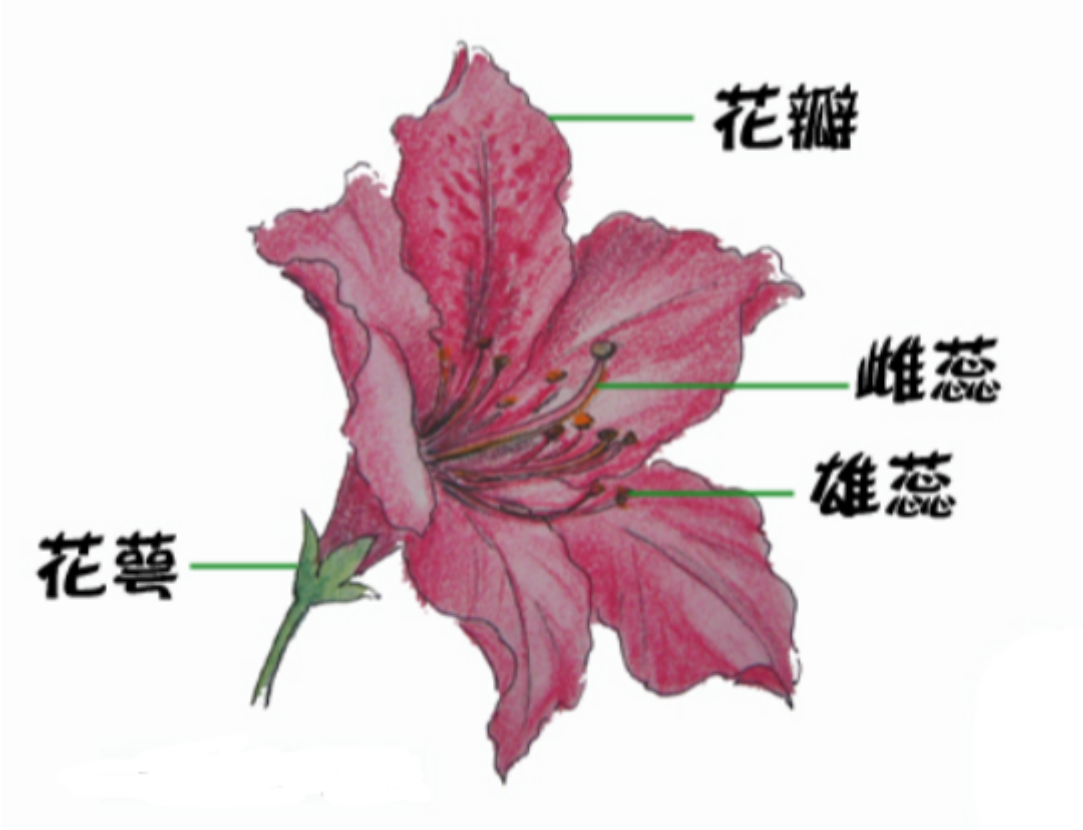
Out[185]:
```
array([ 2,  5,  9, 14, 20, 27, 35, 44], dtype=int32)
```

In [186]:
```
a.cumprod()#累计积
```

Out[186]:
```
array([     2,      6,     24,    120,    720,   5040,  40320, 362880],
      dtype=int32)
```

10 案例实战

10.1 对鸢尾花数据进行分析



1.读取数据
2.排序
3.去重复
4.数组总和
5.累计和
6.均值
7.标准差

8.方差
9.最大值
10.最小值

1. os.chdir(path):这个函数用来修改当前目录 2. os.getcwd() # 获取当前工作路径 3. os.listdir(path):这个函数用来列出目录中的内容，目录下的所有内容（包括文件和目录）会以字符串列表的形式返回 获取路径下所有文件的文件名,以列表的形式呈现出来 4. os.mkdir(path[,mode]): 创建文件夹

In [189]:
```
import numpy as np
iris_sepal_length=np.loadtxt('iris_sepal_length.csv')  #读取文件
```

In [191]:
```
iris_sepal_length
```

Out[191]:
```
array([4.3, 4.4, 4.4, 4.4, 4.5, 4.6, 4.6, 4.6, 4.6, 4.7, 4.7, 4.8, 4.8,
       4.8, 4.8, 4.8, 4.9, 4.9, 4.9, 4.9, 4.9, 4.9, 5. , 5. , 5. , 5. ,
       5. , 5. , 5. , 5. , 5. , 5. , 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1,
       5.1, 5.1, 5.2, 5.2, 5.2, 5.2, 5.3, 5.4, 5.4, 5.4, 5.4, 5.4, 5.4,
       5.5, 5.5, 5.5, 5.5, 5.5, 5.5, 5.5, 5.6, 5.6, 5.6, 5.6, 5.6, 5.6,
       5.7, 5.7, 5.7, 5.7, 5.7, 5.7, 5.7, 5.7, 5.8, 5.8, 5.8, 5.8, 5.8,
       5.8, 5.8, 5.9, 5.9, 5.9, 6. , 6. , 6. , 6. , 6. , 6. , 6.1, 6.1,
       6.1, 6.1, 6.1, 6.1, 6.2, 6.2, 6.2, 6.2, 6.3, 6.3, 6.3, 6.3, 6.3,
       6.3, 6.3, 6.3, 6.3, 6.4, 6.4, 6.4, 6.4, 6.4, 6.4, 6.4, 6.5, 6.5,
       6.5, 6.5, 6.5, 6.6, 6.6, 6.7, 6.7, 6.7, 6.7, 6.7, 6.7, 6.7, 6.7,
       6.8, 6.8, 6.8, 6.9, 6.9, 6.9, 6.9, 7. , 7.1, 7.2, 7.2, 7.2, 7.3,
       7.4, 7.6, 7.7, 7.7, 7.7, 7.7, 7.9])
```

In [192]:
```
iris_sepal_length.sort()#原地排序
```

In [193]:
```
iris_sepal_length=np.unique(iris_sepal_length)  #去重复
iris_sepal_length
```

Out[193]:
```
array([4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5. , 5.1, 5.2, 5.3, 5.4, 5.5,
       5.6, 5.7, 5.8, 5.9, 6. , 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8,
       6.9, 7. , 7.1, 7.2, 7.3, 7.4, 7.6, 7.7, 7.9])
```

In [194]:
```
iris_sepal_length.sum() #求和，优先考虑
```

Out[194]: 210.39999999999998

In [195]:
```
iris_sepal_length.cumsum() #累积和
```

Out[195]:
```
array([  4.3,   8.7,  13.2,  17.8,  22.5,  27.3,  32.2,  37.2,  42.3,
        47.5,  52.8,  58.2,  63.7,  69.3,  75. ,  80.8,  86.7,  92.7,
        98.8, 105. , 111.3, 117.7, 124.2, 130.8, 137.5, 144.3, 151.2,
       158.2, 165.3, 172.5, 179.8, 187.2, 194.8, 202.5, 210.4])
```

In [196]:
```
iris_sepal_length.mean()#平均
```

Out[196]: 6.011428571428571

In [197]:
```
iris_sepal_length.std()#标准差
```

Out[197]: 1.0289443768310533

In [198]:
```
iris_sepal_length.var()#方差
```

Out[198]: 1.0587265306122449

In [199]:
```
iris_sepal_length.min()#最小值
```

Out[199]: 4.3

In [200]:
```
iris_sepal_length.max()#最大值
```

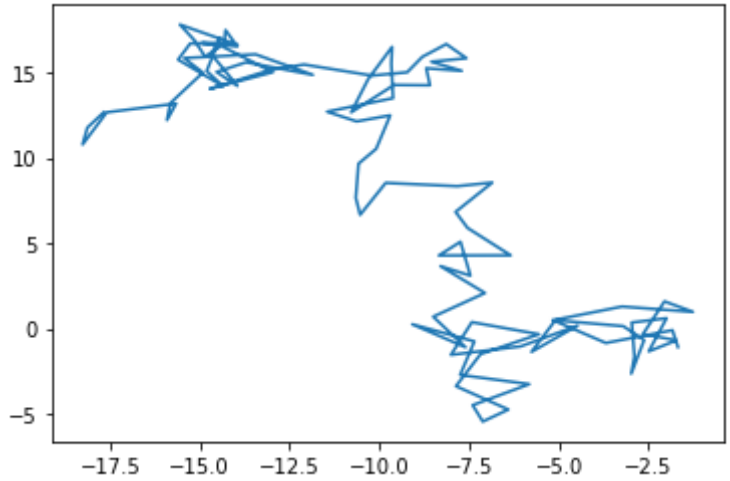
Out[200]: 7.9

10.2 醉汉随机漫步

假设醉汉每一步的距离是1或2，方向也完全随机，360度不确定，然后模拟醉汉走100步的路径

In [201]:
```
r=np.random.randint(1,3,100)      #100个步长，随机1或者2,(第一个参数1为起始值,第二个参数3为终止值,左闭右开,取整数,第三个参数为个数,取100
pi=3.14
theta=np.random.rand(100)*2*pi        #100个随机角度
x=r*np.cos(theta)   #每一次走的x
y=r*np.sin(theta)   #每一次走的y
x=np.cumsum(x)      #走完每步后x的坐标
y=np.cumsum(y)      #走完每步后y的坐标
```

In [202]:
```
import matplotlib.pyplot as plt
plt.plot(x,y)
plt.show();
```



In [ ]: