

同济大学
TONGJI UNIVERSITY

王睿智

ruizhiwang@tongji.edu.cn

人工智能技术与应用

2.1 Numpy 简介

2.2 pandas 简介

- 主流的人工智能编程语言：Python、R、Java、Julia、C/C++等。
- Python最受欢迎，已形成AI开发生态系统，有丰富的AI相关库：
 - 用于机器学习的SciKit-learn,
 - 用于表格数据处理的Pandas,
 - 用于可视化的Matplotlib,
 - 用于深度学习的Pytorch、Keras和TensorFlow等。

- **Python代码**是以 .py 为扩展名的文本文件。可利用集成开发环境编写、调试、运行Python代码。
- 常用的Python集成开发环境有：
 - **IDLE** Python内置的一个简洁开发环境。没有集成第三方库。
 - **Anaconda** 一个开源项目，内含**Spyder**和**Jupyter Notebook**开发环境，集成了大量常用第三方库。
 - **PyCharm** 由JetBrains开发的Python IDE, 支持Django和Google App Engine。
 - **Vs code** 由Microsoft开发的免费、开源、跨平台的轻量级代码编辑器，具有强大的功能和灵活性。

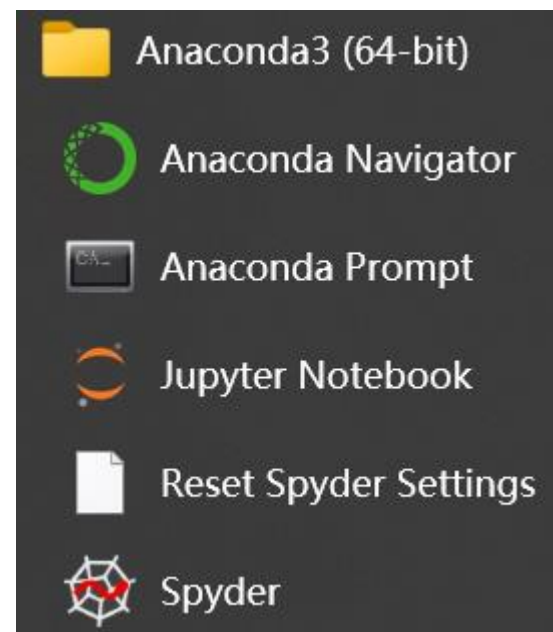
推荐

Anaconda是Python的一个科学计算发行版，内置了数百个Python常用库，包括机器学习库、科学计算等库，如 **Scikit-learn**、**NumPy**、**Pandas**、**Matplotlib**、**SciPy**等。并提供集成开发环境 **Spyder** 和 **Jupyter Notebook**。

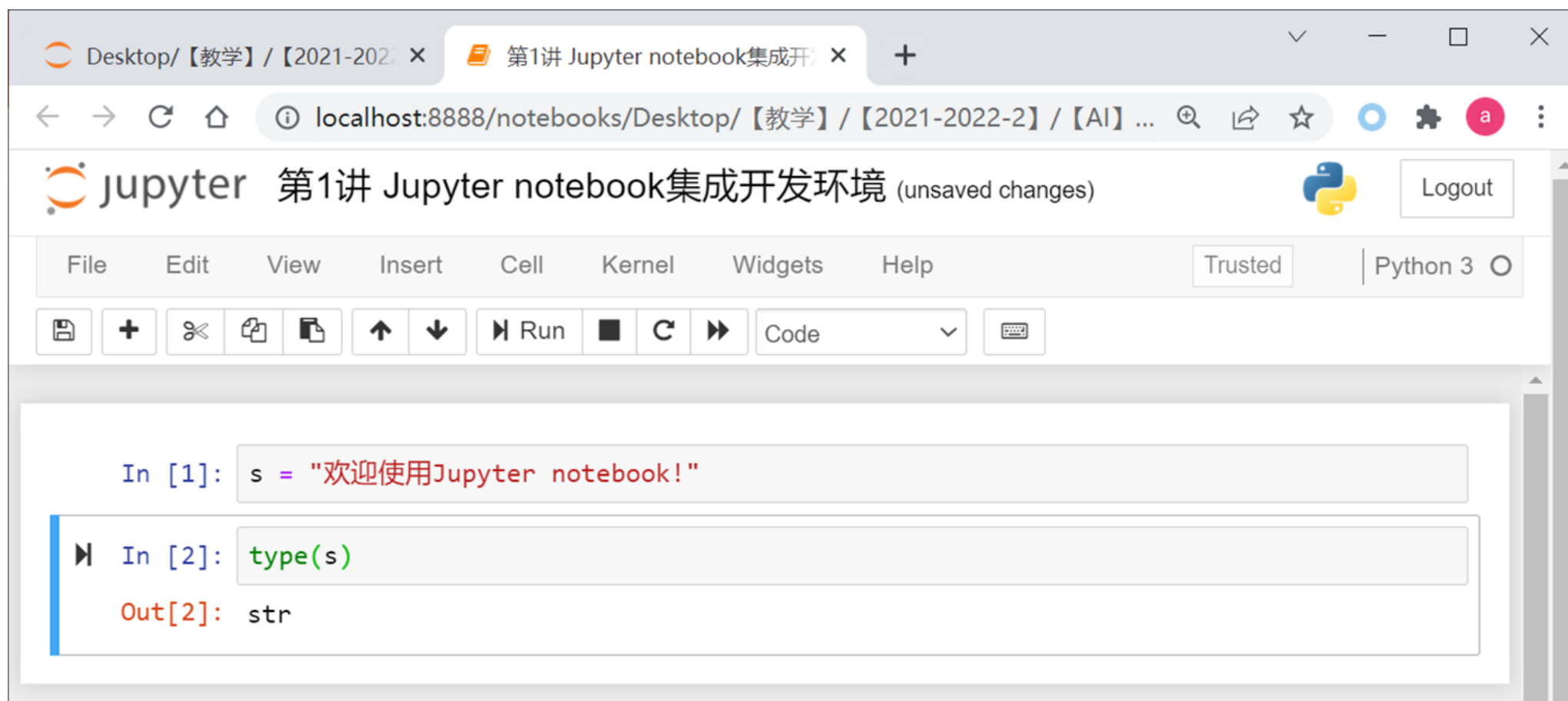
- 清华镜像: <https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/>

根据操作系统，下载相应的安装包，安装即可。
如 64位windows可下载：

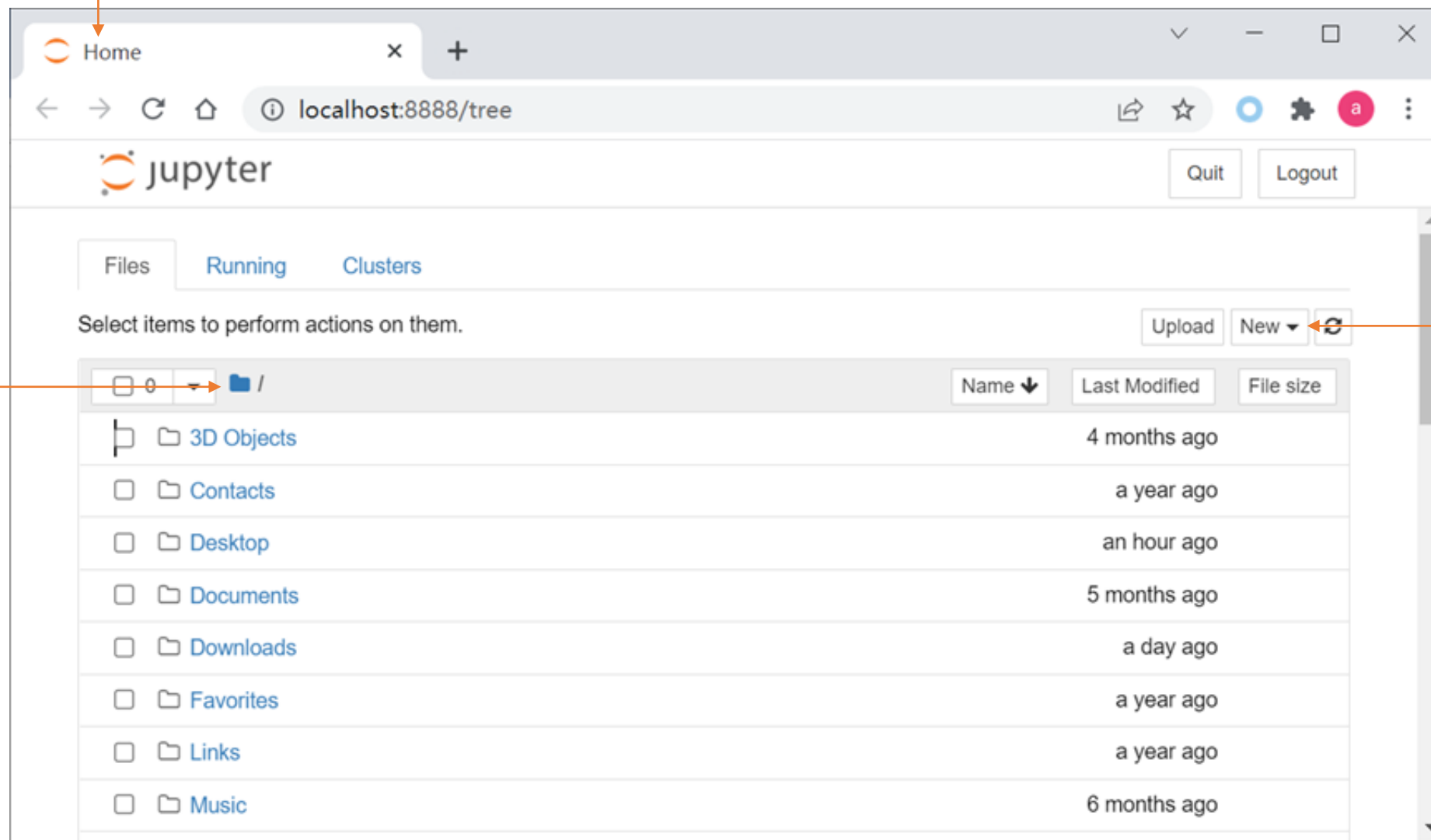
```
Anaconda3-5.0.1-Windows-x86.exe  
Anaconda3-5.0.1-Windows-x86_64.exe  
Anaconda3-5.1.0-Linux-ppc64le.sh
```



Jupyter Notebook 是**基于网页**的、用于交互式计算的应用程序，可以网页的形式打开、编写和运行代码，并且运行结果也会直接在代码块下显示。



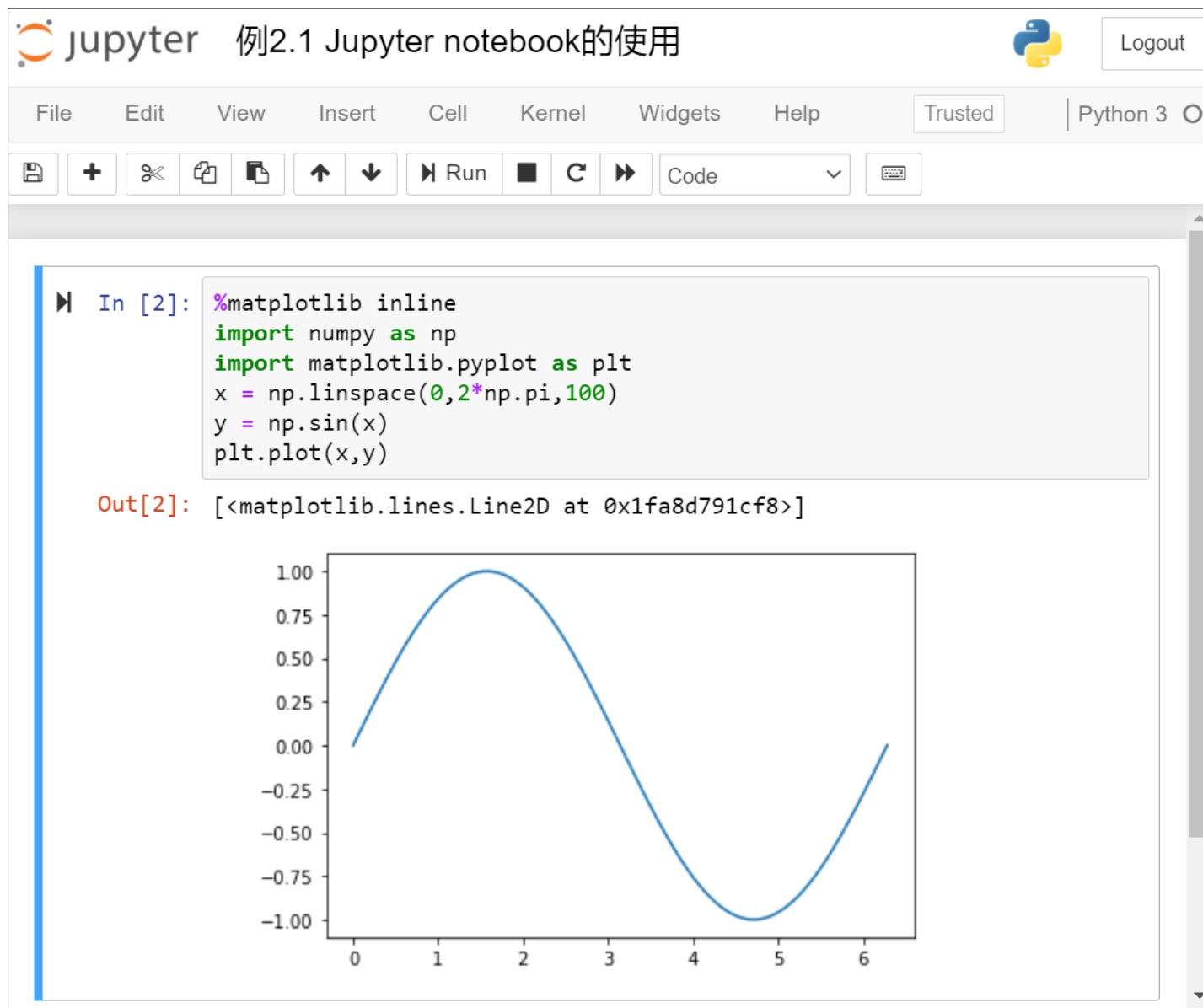
主页



初始
目录

新建
notebook

例 2.1 Jupyter notebook的使用



说明:

(1) `%matplotlib inline` 设置 `inline` 方式, 直接把图片画在网页上。它是 IPython (是交互式的 python 解释器) 的一个魔术命令。凡以 % 为前缀的命令都是 IPython 的魔术命令, 以实现特殊功能。

(2) `np.linspace()` 是 numpy 提供的创建指定元素个数的等距数组的函数。

(3) `np.sin()` 和 `np.pi` 分别是 numpy 提供的正弦函数和 π 常量。

有两种不同的键盘输入模式：

- 编辑模式（将光标移入单元格开启）



由绿色单元格边框指示。编辑模式下，可在单元格中键入代码或文本。
右上角出现一只铅笔的图标。


- 命令模式（将光标移出单元格开启）



由带蓝色左边线的灰色单元格边框指示。命令模式将键盘绑定到命令，
键入的按键作为命令（即快捷键），不作为文本处理。

Windows系统中，命令模式下的常用命令（快捷键）如下：

命令	涵义
Ctrl + Enter	运行当前单元
Shift + Enter	运行当前单元，并把焦点移到下一个单元
A	在当前单元上方插入一个新单元
B	在当前单元下方插入一个新单元
DD	连击两下D，删除选中的单元
Z	恢复最后删除的一个单元

 小技巧

忘记命令时，可在命令模式下，按h键或H键，调出帮助。

2.1 Numpy 简介



- NumPy 是高效科学计算的基础包，不但能完成科学计算的任务，还能用作高效的多维数据容器，可用于存储和处理大型数据。
- **ndarray** (多维数组) 是 NumPy 的核心数据类型。一个 ndarray 对象，是一个快速而灵活的大数据集容器。数组的所有元素是同种数据类型。
- 本节介绍数组创建、访问和操作方法，更多内容见官网：<https://numpy.org/>
- NumPy 是外部库，使用前需导入：**import numpy as np**

```
array([[10, 10, 12, 8],  
       [12, 6, 9, 6],  
       [11, 3, 2, 5]])
```

数组的创建:

- (1) `numpy.array(序列)`, 将序列转换为数组
- (2) `numpy.arange()` 或 `numpy.linspace()`, 创建等距数组
- (3) `numpy.ones()`、`numpy.zeros()` 等函数, 创建特殊的数组
- (4) `numpy.random`模块中的随机函数, 创建随机数组

序列转为数组

```
import numpy as np
x = np.array([[1,3,5,7],[2,4,6,8]], dtype=np.int32)
print(type(x)) # x的类型
print(x.dtype) # x中元素的类型
```

等长的多个列表或
元组创建二维数组

输出：

```
<class 'numpy.ndarray'>
int32
```

(1) numpy.array() 将Python列表、元组等序列对象转换为Numpy数组。

numpy.array(序列对象, dtype = 类型)

创建等距数组

```
x1 = np.arange(1,10,2)
print(x1)
x2 = np.linspace(1,10,5)
print(x2)
```

程序执行结果:

```
[1 3 5 7 9]
[ 1. 3.25 5.5 7.75 10.]
```

(2) numpy.arange()或 numpy.linspace(), 创建等距数组

`numpy.arange(start, stop, step)`

从范围[start, stop), 以步长step取值构成的整型或浮点数组

`numpy.linspace(start, stop, num=50)`

返回指定间隔[start, stop]内的num个等距浮点数构成的数组。

创建特殊数组

```
x3= np.ones((2,4))  
print(x3)  
x4 = np.zeros_like(x3)  
print(x4)
```

输出：

```
[[1.  1.  1.  1.]  
 [1.  1.  1.  1.]]  
[[0.  0.  0.  0.]  
 [0.  0.  0.  0.]]
```

(3) numpy.ones()函数等，创建特殊的数组

- **numpy.zeros**(形状, dtype=float), 返回一个给定形状和类型的新数组，并用**零**填充。
- **numpy.ones**(形状, dtype=None), 返回一个给定形状和类型的新数组，并用**1**填充。
- **numpy.zeros_like**(a, dtype=None), 返回一个与给定数组a形状和类型相同的**零**数组。

创建随机数组

```
a1 = np.random.rand(2,3)
a2 = np.random.randint(0,10,(3,3))
a3 = np.random.randn(10)
```

这里所有函数，若省略形状参数，则返回一个float数。

(4) numpy.random中的随机函数，创建随机数组

函数	涵义
<code>np.random.rand(d0, d1, ..., dn)</code>	从[0,1)上 均匀分布 中随机抽取数，创建一个形状为(d0, d1, ..., dn)的 随机float数组 。
<code>np.random.randint(low, high=None, size=None)</code>	从[low, high)上 离散均匀分布 中随机抽取 整数 ，创建一个形状由参数size指定的 随机整数数组 。如果high为None (默认值)，则采样区间来自[0, low)。
<code>np.random.randn(d0, d1, ..., dn)</code>	从 标准正态 (均值0, 方差1)分布中随机抽取实数，创建一个形状为(d0, d1, ..., dn)的 随机float数组 。
<code>np.random.choice(n,k,replace=False)</code>	从 0~n-1 不放回地随机抽取k个
<code>np.random.uniform(low, high, (d0, d1,...,dn))</code>	创建一个[low,high)之间 随机浮点 数组成的形状为(d0, d1, ..., dn)的 随机float数组
<code>np.random.normal(均值=0.0, 标准差=1.0, 形状=None)</code>	创建一个指定形状的均值为0，标准差为1，符合 正态分布 的 随机float数组

<数组名>.<属性名> 查看数组的属性

```
x = [[1,2,3,4],[10,20,30,40]]
a = np.array(x)

print(a.shape) # 输出(2, 4)
print(a.size)  # 输出8
print(a.ndim)  # 输出2
print(type(x)) # 输出<class 'list'>
print(type(a)) # <class 'numpy.ndarray'>
print(a.dtype) # 输出 int32
```

属性	涵义
<数组名>.size	查看数组元素的总个数
<数组名>.shape	数组的形状，表示数组维度大小的元组，每个元素对应数组每个维度的大小。 每维也称轴，从0开始编号
<数组名>.ndim	数组有几个维度
<数组名>.dtype	数组元素的类型

type(<数组名>) 查看数组类型



课堂练习

使用 `numpy.random` 模块生成一个形状为 `(3, 4)` 的数组，其中的元素服从标准正态分布（均值为0，方差为1），哪个选项正确：

- A. `random_array = np.random.randint(0,1,size =(3, 4))`
- B. `random_array = np.random.randn(3,4)`
- C. `random_array=np.random.uniform(0,1,size = (3,4))`
- D. `random_array = np.random.rand(3, 4)`



引例 iris数据可视化

鸢尾花数据集有150个样本，每个样本有4个特征，若想取出前2个特征对应的数据，画出数据散点图，怎么做呢？

```
array([[5.1, 3.5, 1.4, 0.2],  
       [4.9, 3. , 1.4, 0.2],  
       [4.7, 3.2, 1.3, 0.2],  
       [4.6, 3.1, 1.5, 0.2],  
       [5. , 3.6, 1.4, 0.2],  
       [5.4, 3.9, 1.7, 0.4],  
       [4.6, 3.4, 1.4, 0.3],  
       [5. , 3.4, 1.5, 0.2],  
       [4.4, 2.9, 1.4, 0.2],  
       [4.9, 3.1, 1.5, 0.1],  
       [5.4, 3.7, 1.5, 0.2],  
       [4.8, 3.4, 1.6, 0.2],  
       [4.8, 3. , 1.4, 0.1],  
       [4.3, 3. , 1.1, 0.1],  
       [5.8, 4. , 1.2, 0.2],  
       [5.7, 4.4, 1.5, 0.4],  
       [5.4, 3.9, 1.3, 0.4],  
       [5.1, 3.5, 1.4, 0.3],  
       [5.7, 3.8, 1.7, 0.3],  
       ...])
```

关键： 获取数组中的**某个**元素或**某些**元素

数组索引操作可获取**某个**元素

数组切片操作可获取**某些**元素

数组索引操作：获取单个元素

- 一维数组a, **a[i]** 获取a中索引为i的元素 (i从0开始)。支持负数索引。
- 多维数组a, **a[i,j,k,...]**, 获取对应位置元素。

注意：数组a有几维，[]中就有几个值。**a[i,j,k]** 等价于 **a[i][j][k]**

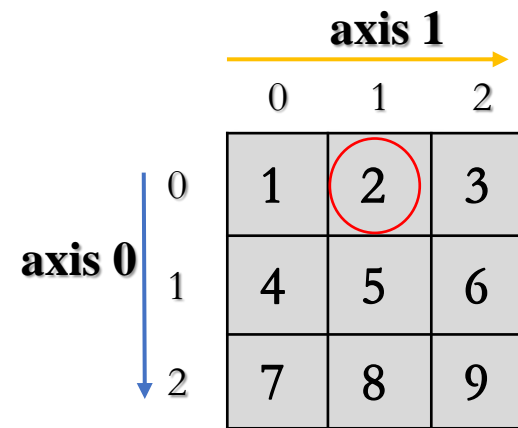
```
a = np.array([[1,2,3], [4,5,6], [7,8,9]])
```

```
# 获取位置0行、1列的元素？
```

```
print (a[0,1],a[0][1])
```

```
# 将最后1行、最后1列的元素修改为0
```

```
a[-1,-1] = 0
```



		axis 1		
		0	1	2
axis 0	0	1	2	3
	1	4	5	6
	2	7	8	9

说明： **a[i,j,k] = x** 对数组元素赋值

数组切片操作: 获取子数组, 维数不变

切片用冒号(:)表示, 语法与Python列表相同 不能省

一维数组切片: $x[\text{start} : \text{stop} : \text{step}]$

表示从 x 的半开区间 $[\text{start}, \text{stop})$ 中, 以 step 为步长取元素, 构成新数组。
若 start 省略, 则取 0; 若 stop 省略, 则取数组大小; 若 step 省略, 则取 1。

对切片赋值: $x[\text{start} : \text{stop} : \text{step}] = \text{值 或 同维数组}$

```
x = np.arange(10)
```

```
x[:5]      # ?
```

```
x[5:]      # ?
```

```
x[:-1]     # ?
```

```
x[1::2]    # ?
```

```
x[5::-2]=10
```

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

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

```
array([5, 6, 7, 8, 9])
```

```
array([0 1 2 3 4 5 6 7 8])
```

```
array([1, 3, 5, 7, 9])
```

```
# 对切片赋值, 哪些值被修改了? array([5, 3, 1])
```

多维数组切片: `x[start1:stop1:step1, start2:stop2:step2, ...]`

先从 `x` 的 0 轴 `[start1, stop1)` 中, 以 `step1` 为步长取元素, 构成新数组; 再从新数组的 1 轴 `[start2, stop2)` 中, 以 `step2` 为步长取元素, ...。所得到的子数组与原数组维数相同。赋值操作与一维数组切片赋值类似。

```
x1 = np.array([[12, 5, 2, 4], [7, 6, 8, 8], [1, 6, 7, 7]])
print(x1[:2, :3])           # 输出? 等价于 x1[:2][:3]
print(x1[:3, ::2])          # 输出?
print(x1[::-1, ::-1])       # 输出?
x1[:1, :] = [10, 20, 30, 40] # 可缩写为 x1[:1], 改了哪些值?
```

```
[ [12,  5,  2,  4],
  [ 7,  6,  8,  8],
  [ 1,  6,  7,  7]]
```

```
[ [12,  5,  2,  4],
  [ 7,  6,  8,  8],
  [ 1,  6,  7,  7]]
```

🔔 注意

数组有几维, 切片列表中就有几项。若最后一项为空切片, 则可省。单独一个冒号(`:`)表示选择整个轴, 称空切片, 如 `x1[0:2,:]` 同 `x1[0:2]`

```
[[10 20 30 40]
 [ 7  6  8  8]]
```


索引与切片混用：将得到比原数组维度低的子数组

```
x2 = np.random.randint(1,13,(3,4))
```

```
x2[:,0] # 索引切片混用，降维
```

```
x2[0,:] # 可简写为 x2[0]
```

```
x2[0:1,:] # 各维都用切片，维数不变
```

```
x2[0:1,:]=0 # x2 ?
```

```
array([[10, 10, 12,  8],  
       [12,  6,  9,  6],  
       [11,  3,  2,  5]])
```

二维

```
array([10, 12, 11])
```

一维

```
array([10, 10, 12,  8])
```

一维

```
array([[10, 10, 12,  8]])
```

二维

说明：二维数组索引与切片混用，获取原数组的单行/单列元素构成的一维数组。



课堂练习 1

二维数组a

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

1. 获取数组a的中间行数据[5 7], 写出操作表达式, 查看其形状。
2. 从a中获取[[5 7]], 写出操作表达式, 查看其形状。



花式索引：通过索引列表 (`[i1,i2,...]`) 来访问元素

可以灵活取出索引列表中指定行或列的元素。

```
np.random.seed(25)
```

```
arr = np.random.randint(1,20,(4,2))
```

```
array([[ 5, 16],  
       [13, 19],  
       [ 9,  5],  
       [ 6,  2]])
```

若希望选取红圈指定元素构成数组，怎么实现？

```
arr[[1,3],1]    array([19,  2])
```

```
arr[[-1,-3],:] # 输出？可简写为arr[[-1,-3]] array([[ 6,  2],  
       [13, 19]])
```

```
arr[[1,3],1] = 10 #修改数组
```

思考：用花式索引实现课堂练习1

布尔索引：从数组中选择满足条件的元素

- NumPy数组比较运算（==, !=, <, >, <=, >=），实现了数组的**逐元素比较**，其结果是一个布尔类型的数组。两个布尔数组的元素可进行**逐位布尔运算**（& 与，| 或，~取反，^异或），其结果仍为布尔数组。
- 用布尔数组作为掩码，通过掩码选择一个数组的子集，称为**布尔索引**。

```
x = np.array([1,2,3,4,5])
print(x)
print(x < 3)  # 返回的布尔数组形状与x同
print(x > 1)
print((x<3) & (x>1)) # 条件逐位“与”
print(x[(x<3) & (x>1)]) # 布尔索引
print(x % 2 == 0) # 先算术运算，再比较运算
print(x[x % 2 == 0]) # 布尔索引
```

```
[1 2 3 4 5]
[ True  True False False False]
[False  True  True  True  True]
[False  True False False False]
[2]
[False  True False  True False]
[2 4]
```



课堂练习 2

成绩二维数组score

```
[[80 62 73 84]
 [95 36 67 78]
 [59 99 81 82]]
```

若改为五级制，即

将在 $[0,60)$ 内的元素，为1；

在 $[60,70)$ 内的元素，为2；

在 $[70,80)$ 内的元素，为3；

在 $[80,90)$ 内的元素，为4；

在 $[90,100]$ 内的元素，为5。怎样实现？

```
import numpy as np
score=np.array([[80,62,73,84],[95,36,67,78],[59,99,81,82]])
score[(score>=0)&(score<60)]=1 # 直接赋值即可
score[(score>=60)&(score<70)]=2
score[(score>=70)&(score<80)]=3
score[(score>=80)&(score<90)]=4
score[(score>=90)&(score<=100)]=5
print(score)
```




课堂练习 3

Iris数据集有4个特征，取前2个特征对应数据，画出散点图，点按类别着色。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

iris = pd.read_csv('iris.csv')
data = iris.values

M = np.array(data[:, :4], dtype='float')
y = np.array(data[:, -1])

y[y=='setosa']='red'
y[y=='versicolor']='blue'
y[y=='virginica']='yellow'
plt.scatter(M[:, 0], M[:, 1], c = y)
```

数组变形，是指只改变数组的形状、不改变元素值的操作。它要求变形后数组的大小与原数组的大小必须一致。

reshape()方法最灵活，flatten()或ravel()用来展平数组。

1. x.reshape(t)方法

将数组x变形为参数t所指定的形状。参数**t是正整数元组,如 (3,3)**。

【例】 将数字1~9放入3×3的矩阵中

```
a1 = np.arange(1,10)
a2 = a1.reshape((3,3))
# 或 a2 = a1.reshape(3,3)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

【例】 将一维数组`x=np.array([1,2,3])`分别转为二维的行矩阵,列矩阵。

```
x = np.array([1,2,3])  
newx1 = x.reshape((1,3)) # 行矩阵  
newx2 = x.reshape((3,1)) # 列矩阵
```

```
array([[1, 2, 3]])
```

```
array([[1],  
       [2],  
       [3]])
```

另一种方法：在切片操作中利用`numpy.newaxis`关键字

```
x[np.newaxis,:] # 加一个新轴作为0轴，获得行矩阵  
x[:,np.newaxis] # 加一个新轴作为1轴，获得列矩阵
```

2. `flatten()`或`ravel()`方法，将高维数组展平为一维向量

```
grid = np.arange(1,10).reshape(3,3)
print(grid)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
v1 = grid.flatten()
print(v1)
```

```
v2 = grid.ravel()
print(v2)
```

```
[1 2 3 4 5 6 7 8 9]
```

```
v3 = grid.reshape( ? )
print(v3)
```

reshape()也可以

数组的拼接（也称合并），指讲多个数组合并

`concatenate([数组a,数组b,...], axis=0)` 按指定轴方向合并多个数组。

沿0轴

```
arr1 = np.array([[1,2,3],[4,5,6]])
arr2 = np.array([[7,8,9],[10,11,12]])
a = np.concatenate([arr1,arr2])
a
```

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

行堆叠

沿1轴

```
arr1 = np.array([[1,2,3],[4,5,6]])
arr2 = np.array([[7,8,9],[10,11,12]])
b = np.concatenate([arr1,arr2],axis=1)
b
```

```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
```

列堆叠

算术运算(+、-、*、/、//、%、**)：向量运算，对应元素的操作

对应元素的操作要求两数组元素个数相同

```
x = np.array([[1,2],[3,4]], dtype=float)
y = np.array([[5,6],[7,8]], dtype=float)
```

```
# 对应元素相加
print(x + y)
```

```
[[ 6.  8.]
 [10. 12.]]
```

```
x= [[ 1.  2.]
     [ 3.  4.]]
```

```
y= [[ 5.  6.]
     [ 7.  8.]]
```

```
# 对应元素相乘
print(x * y)
```

```
[[ 5. 12.]
 [21. 32.]]
```

若两数组元素个数不同，能否算术运算呢？有时可以！通过广播！

数组和标量之间可以进行算术运算

数组与标量的算术运算也是元素级的，将那个标量值**广播**到各个元素。

```
arr = np.array([[1., 2., 3.], [4., 5., 6.]])  
arr
```

```
1 / arr # 除法
```

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

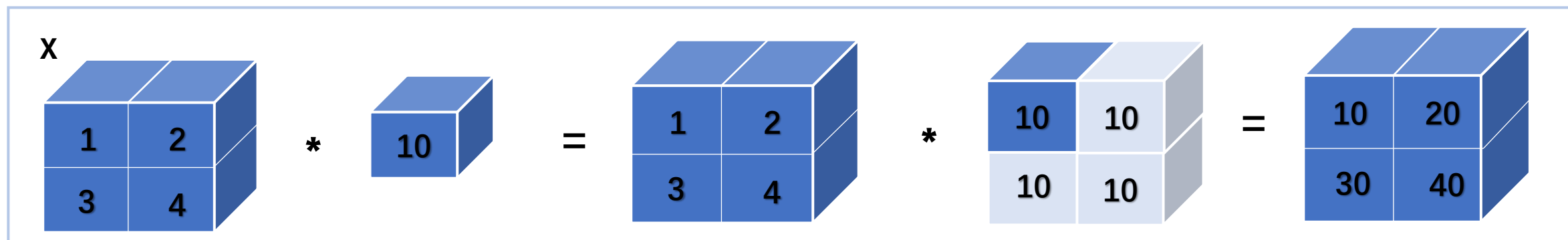
```
arr ** 0.5 # 幂运算
```

```
array([[1.          , 1.41421356, 1.73205081],  
       [2.          , 2.23606798, 2.44948974]])
```

```
arr // 2 # 地板除法
```

```
array([[0., 1., 1.],  
       [2., 2., 3.]])
```


广播功能，使得形状不同的数组之间也可以进行运算



2×2 矩阵 X 和标量10之间进行乘法运算时，可以认为标量10被扩展成2×2的形状，然后再与矩阵 X 进行乘法运算。这种功能被称为**广播 (broadcast)**。

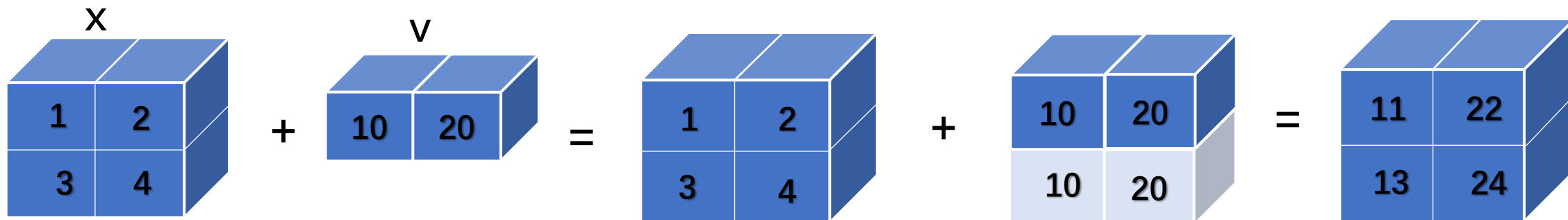
```
X = np.array([[1,2], [3,4]])  
print (X * 10)
```

```
X * 10 = [[10 20]  
          [30 40]]
```

1个数组广播

```
x = np.array([[1,2], [3,4]])  
v = np.array([10, 20])  
y = x + v    # 矩阵x的每行都加上一维数组v  
print(y)
```

```
y = [[11 22]  
     [13 24]]
```

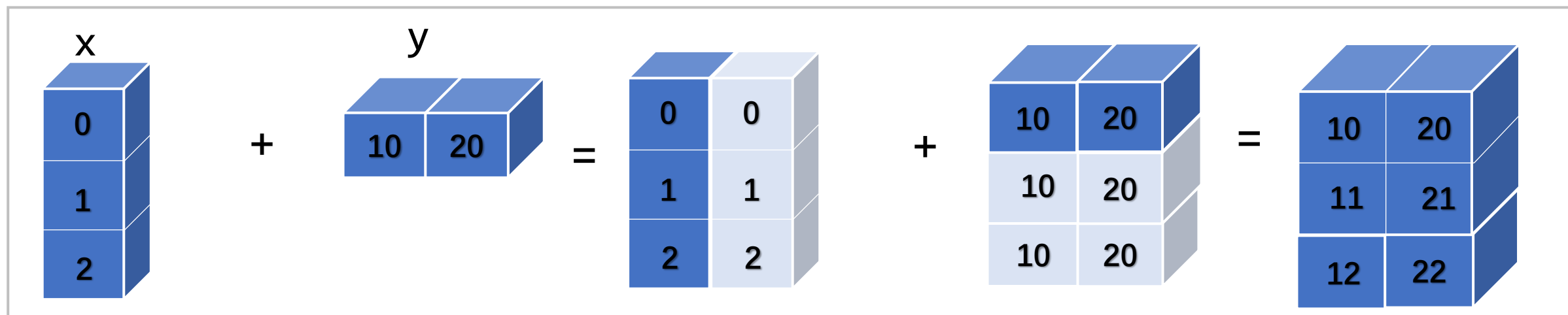


上图，数组 v 被变成和二维数组 x 相同形状，然后再以**对应元素**的方式进行运算。

两个数组同时广播

```
x = np.arange(3).reshape(3,1)
y = np.array([10,20])
v = x + y
print(v)
```

```
[[10 20]
 [11 21]
 [12 22]]
```



上图，数组 x 被广播成 $(3,2)$ ，数组 y 也被广播成与 x 相同形状，然后再做对应元素运算。

例 下列情况能使用广播机制来匹配吗？

```
a = np.arange(6).reshape(2,3)
b = np.arange(6)
v = a + b    # 能正常运算吗 ???
```

- 可用广播匹配的条件：两个数组必须有一个维度可以扩展。数组在可扩展的维度上进行复制，得到两个相同维度的数组，再进行运算。
- 多维数组与标量运算，是广播的一个特例。标量根据多维数组的维度进行扩展，最后复制出与多维数组维度相同的数组后，再进行运算。

矩阵转置T：原先的行变为列，原先的列变为行。用NumPy数组表示矩阵。

矩阵 v 的转置： $v.T$

```
x = np.array([[1,2], [3,4]])
```

```
print(x)
```

```
print(x.T)
```

注意：一维数组转置不会对数组产生任何影响

```
v = np.array([1,2,3])
```

```
print(v)
```

```
print(v.T)
```

$$\mathbf{x} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$\mathbf{x} \cdot \mathbf{T} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

$$\mathbf{v} = [1 \ 2 \ 3]$$

$$\mathbf{v} \cdot \mathbf{T} = [1 \ 2 \ 3]$$

内积：也称两个矩阵或向量的点积。

v 和 w 符合进行点积运算的条件

内积语法有三种：

- `v.dot(w)`
- `np.dot(v,w)`
- `v @ w`

向量内积

```
v = np.array([9,10])  
w = np.array([11, 12])
```

`v = [9 10]`

`w = [11 12]`

```
print(v.dot(w))  
print(np.dot(v, w))  
print(v@w)
```

`v•w = 219`

矩阵/矩阵内积

```
x = np.array([[1,2],[3,4]])  
y = np.array([[5,6],[7,8]])
```

`x = [[1 2]
[3 4]]`

`y = [[5 6]
[7 8]]`

```
print(x.dot(y))  
print(np.dot(x, y))  
print(x@y)
```

`x•y = [[19 22]
[43 50]]`

注意顺序！

聚合函数：对数组元素按轴聚合统计，如 `np.sum(x,axis=0)`

语法有两种：

- `np.聚合函数(<数组名>, axis = None)`
- `<数组名>.聚合函数 (axis = None)`

说明：

- 省略 `axis` 参数，则对数组全体元素做聚合；
- `axis = 0`，表示0轴将被折叠，对于二维数组意味着每列的值都将被聚合。

```
x = np.array([[1,2],[3,4]])
print(np.sum(x))           # 计算数组所有元素之和
print(np.sum(x, axis=0))   # 计算每列元素之和
Print(np.sum(x, axis=1))  # 计算每行元素之和
```

输出：

10

[4 6]

[3 7]


```
arr = np.array([[0,1,2],[3,4,5],[6,7,8]])
print(np.sum(arr%2==0)) # 统计arr中偶数元素个数
print((arr%2==0).sum()) # 另一种语法
print(arr.min(),arr.min(axis=0),arr.min(axis=1))
```

方法	说明
sum	对数组中全部或某轴向的元素求和，不指定axis则求数组全部元素之和。零长度的数组的sum为0。np.sum(<布尔数组>)中布尔值True视为1，False视为0。
mean	求算术平均数。零长度的数组的mean为NaN
std, var	求标准差和方差
min, max	求最小值和最大值
argmin, argmax	求最小和最大元素的索引
cumsum, cumprod	求某轴向元素的累计和、累计积，产生一个由中间结果组成的数组

数组排序: `np.sort()` 和 `np.argsort()`

- `np.sort(a,axis=-1)` 返回排序后的新数组，原数组a不变。
- `a.sort(axis=-1)` 用排好序的数组替代原数组a。
- `np.argsort(a,axis=-1)` 返回原始数组排好序的索引值数组。

数组元素沿参数axis指定轴升序排序，默认沿最后一个轴排序。

```
arr1=np.random.randint(1,10,(3,4))  
print(arr1)
```

```
arr2 = np.sort(arr1,axis=0)  
print(arr2)      # 新数组
```

```
print(arr1)
```

np.sort()不改变原数组

```
arr3 = np.array([42,21,37,56,98])  
idx = np.argsort(arr3)  
print(idx)
```

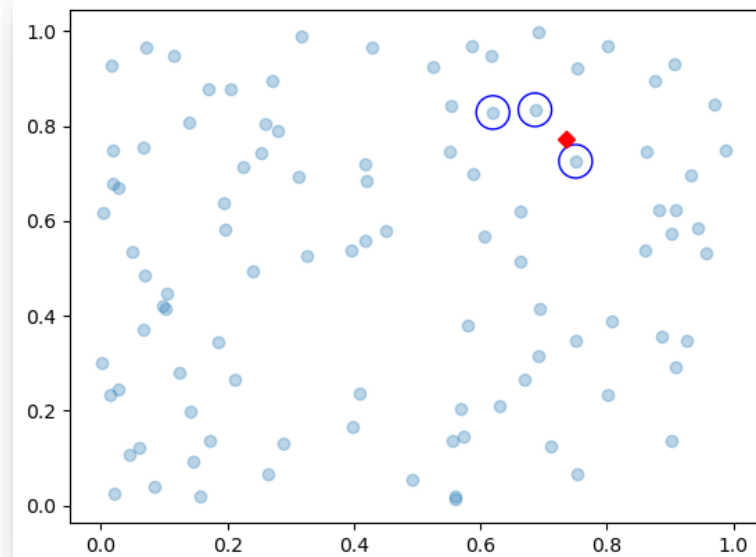
```
[[3 7 3 3]  
 [1 2 3 1]  
 [7 1 3 3]]
```

```
[[1 1 3 1]  
 [3 2 3 3]  
 [7 7 3 3]]
```

```
[[3 7 3 3]  
 [1 2 3 1]  
 [7 1 3 3]]
```

```
[1 2 0 3 4]
```

在二维空间中，创建100个 $[-5, 5)$ 上均匀分布随机点的集合。从中随机选1点，找到它的k个（如 $k=3$ ）最近邻。用平方距离（即欧氏距离的平方）来衡量两点远近。画图显示结果。



```
import numpy as np
import numpy.random as rand
import matplotlib.pyplot as plt
rand.seed(1)

X = rand.rand(100,2)*10-5 # 1. 创建100个点, 每维在[-5,5)随机抽取
i = rand.choice(X.shape[0],1,replace=False) #不放回随机抽1点,返其索引
selected = X[i] # 2. 随机选定点的坐标
dist =np.sum((X - selected)**2,axis=1) # 3.计算该点到各点的平方距离
idx = np.argsort(dist) # 4.距离数组排序, 返回索引值数组
k=3
neighbor = X[idx[1:k+1]] # 5.取k个最近邻的坐标(去掉自己)

plt.scatter(X[:,0],X[:,1],alpha=0.3)
plt.plot((selected[0,0]),(selected[0,1]),'rD')
plt.scatter(neighbor[:,0],neighbor[:,1],facecolor='none',
            edgecolor='b',s=300)
```

iris数据有150个样本，分属三类(setosa,versicolor,virginica)。每个样本由4个特征 (sepal length, sepal width, petal length, petal width)来描述。

4个特征的取值范围有大有小，怎样将每个特征的取值都转换到[0,1]?

特征规范化 是将数据的各个特征进行缩放处理，使各特征取值均落入一个给定的区间内或服从指定分布。

iris.csv文件

特征 \Rightarrow

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa

\Leftarrow 目标类别

对iris数据进行特征规范化，使得每个特征取值范围都转换为[0,1]，称为**min-max归一化**。

iris.csv文件

$$\text{公式: } M_{scaled} = \frac{M - \min}{\max - \min}$$

读文件

取每行前4列

```
[['5.1', '3.5', '1.4', '0.2'],
 ['4.9', '3', '1.4', '0.2'],
 ['4.7', '3.2', '1.3', '0.2'],
 ['4.6', '3.1', '1.5', '0.2'],
 ['5', '3.6', '1.4', '0.2'],
 ['5.4', '3.9', '1.7', '0.4'],
 ['4.6', '3.4', '1.4', '0.3'],
 ['5', '3.4', '1.5', '0.2'],
 ['4.4', '2.9', '1.4', '0.2'],
 ['4.9', '3.1', '1.5', '0.1'],
 ['5.4', '3.7', '1.5', '0.2'],
 ['4.8', '3.4', '1.6', '0.2'],
 ['4.8', '3', '1.4', '0.1'],
 ['4.3', '3', '1.1', '0.1'],
 ['5.8', '4', '1.2', '0.2'],
 ['5.7', '4.4', '1.5', '0.4'],
 ['5.4', '3.9', '1.3', '0.4'],
 ['5.1', '3.5', '1.4', '0.3'],
 ['5.7', '3.8', '1.7', '0.3'],
 ...]
```

列表转换

为float数组

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       ...])
```

min-max归一化

```
import numpy as np
import pandas as pd

data=pd.read_csv('iris.csv')
X=data.values[:, :-1] # 不要最后一列

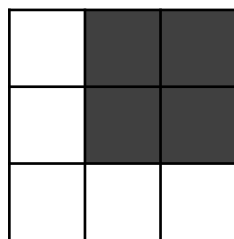
M = np.array(X, dtype='float') # 将Pandas的DataFrame转换为数组
min_M = np.min(M, axis=0) # 沿0轴求每列最小
max_M = M.max(0) # 沿0轴求每列最大
scaledM = (M-min_M)/(max_M - min_M)
print('归一化后数组: \n', scaledM)
```



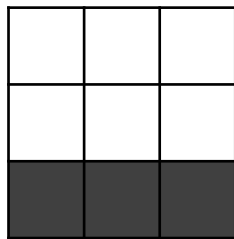

作业

1.1 二维数组的切片与索引

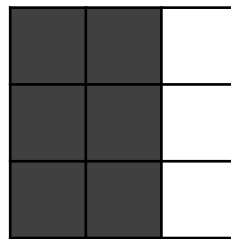
请对数组a进行切片和索引操作，以获取阴影部分结果，如下图(1)~(5)所示。将所有可行的操作表达式及对应结果的shape填入表中。



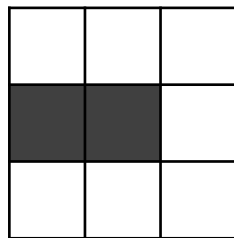
(1)



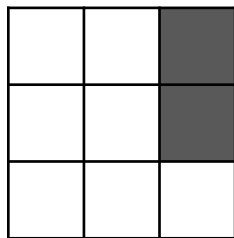
(2)



(3)



(4)



(5)

题号	表达式	shape
(1)		
(2)		
(3)		
(4)	a[1, :2] a[1:2, :2]	(2,) (1, 2)
(5)		

提交word文件



作业

1.2 数组的布尔索引

用arange()函数生成一个形状为(3, 4)的二维数组a，如下所示

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

将其中所有3以下和9以上的元素都修改为5，结果如下图所示。

```
a = [[5 5 3 4]
      [5 6 7 8]
      [9 5 5 5]]
```

提交代码



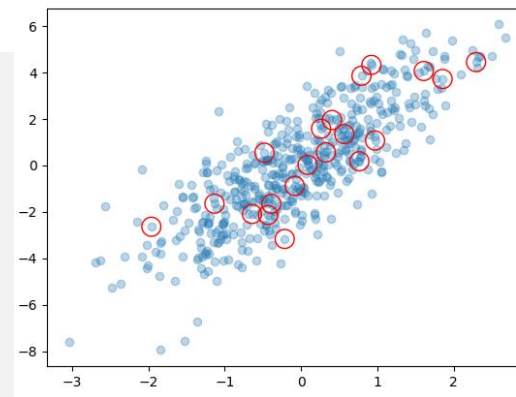
作业

1.3 选择随机点

程序填空：以[0,0]为均值、[[1,2],[2,6]]为协方差，生成二维正态分布的点云，包含500个点。从中随机抽取20个，画出点云、用红圈标出选中的随机点。

```
import numpy.random as rand
import matplotlib.pyplot as plt

mean = [0,0]          # 指定均值
cov=[[1,2],[2,6]]     # 指定方差
X = rand.multivariate_normal(mean, cov,500) # 从二维正态分布中抽取500个
n= X.shape[0]
idx = rand.choice(n,20,replace=False) #从0~n-1不放回地随机抽取20个
selected = _____ # 选定点
plt.scatter(_____(2)_____,_____(3)_____,alpha=0.3) # 画出点云
plt.scatter(_____(4)_____,_____(5)_____,
            facecolor='none',edgecolor='r',s=200) # 标出选中点
```





作业

1.4 求给定数据的最近邻

在二维空间中，创建100个随机点的集合A，坐标点由 $[0, 1)$ 上均匀分布中随机抽取一对实数构成。然后，再从相同分布中随机抽取10个点，构成数据集B，找出B中每个点在A中的最近邻（ $k=1$ ）。

提交代码



作业

1.5 Z-score标准化

生成一个形状为(10,4)、元素在[0,100)中的随机整数数组。对数组进行规范化预处理，使得每列元素的列均值为0，标准差为1（称为**Z-score标准化**）。

$$\text{公式: } x^* = (x - \mu) / \sigma$$

其中， x 为随机数组的某列， μ 为该列均值； σ 为该列的标准差。

Z-score标准化

- Z-score标准化是一种常用的数据规范化方法。
- 适用于特征最大值和最小值未知的情况，或者有超出取值范围的离群数据的情况。

提交代码