高级程序设计
---Python与深度学习
8. numpy & matplotlib

李冰
副研究员
交叉科学研究院

课程内容

- Numpy
  - 创建数组
  - 索引、切片
  - 基本操作
    - 计算
    - 广播
    - 拷贝
    - 迭代
  - 形状变换
  - 随机数
  - 存取函数
  - 其他操作
- Matplotlib
  - 绘制
  - 显示

Numpy

- NumPy 是非常著名的 Python 科学计算工具包
  - 包含了大量有用的概念，比如数组对象（用来表示向量、矩阵、图像等）以及线性代数函数。
  - NumPy 中的数组对象可以实现数组中重要的操作
    - 矩阵乘积、转置、解方程系统、向量乘积和归一化
  - 为图像识别和深度学习等提供了基础。
- NumPy 官方快速入门教程 Quickstart tutorial

Numpy

- NumPy 是非常著名的 Python 科学计算工具包
- 包含了大量有用的概念，比如数组对象（用来表示向量、矩阵、图像等）以及线性代数函数。
- NumPy 中的数组对象可以实现数组中重要的操作
  - 矩阵乘积、转置、解方程系统、向量乘积和归一化
  - 为图像识别和深度学习等提供了基础。
- NumPy 官方快速入门教程 Quickstart tutorial

使用 import 命令导入 NumPy包：

```
import numpy as np
```

Numpy ndarray数组

- NumPy 包
  - 核心是 ndarray 对象，用来表示 N 维数组

```
1  import numpy as np
2  np.array([[ 1., 0., 0.],
3  [ 0., 1., 2.]])
```

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

Numpy ndarray数组

- NumPy 数组对象和标准 Python 序列之间有几个重要的区别：
  - NumPy 数组的大小在创建时就已经固定。
    - Python list 可以动态增长
  - NumPy 数组对大数据量的高级数学和其他类型的操作进行了优化
    - 与Python 序列对象相比，此类操作的**执行效率更高**，代码更少
  - 基于 Python 的科学/数学库大多使用 NumPy 数组对象;
    - 输入转换为NumPy数组，输出 NumPy 数组
    - Python 序列对象需要转换为Numpy数组才能执行
  - NumPy 数组中的元素都需要具有相同的数据类型，因此在内存中的大小相同。

创建numpy数组

- 使用 np.array() 函数从常规 Python 列表或元组中创建数组

```
origin_list = [2, 3, 4]
a = np.array(origin_list)
a
```

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

创建numpy数组

- 使用 np.array()函数从常规 Python 列表或元组中创建数组

```
origin_list = [2, 3, 4]
a = np.array(origin_list)
a
```

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

一个常见的错误在于使用多个数值参数调用 np.array() 函数，而不是提供一个数字列表（List）作为参数。

```
# a = np.array(1, 2, 3, 4)    # Wrong
a = np.array([1, 2, 3, 4])  # Right
```

创建numpy数组

```
1 import numpy as np
2 a=np.array([[ 1., 0., 0.]])
3 type(a)
```

```
numpy.ndarray
```

ndarray 的关键属性

- **ndarray.ndim**
  - 数组的轴（维度）的个数。在 Python 中，维度的数量被称为 rank。
- **ndarray.shape**
  - 数组的形状。shape是一个整数的元组，表示每个维度中数组的大小。对于有 n 行和 m 列的矩阵，shape 将是 (n,m)。因此，shape 元组的长度就是 rank 或维度的个数 ndim。
- **ndarray.size**
  - 数组元素的总数，等于 shape 的元素的乘积。
- **ndarray.dtype**
  - 一个描述数组中元素的类型。可以使用标准的 Python 类型创建或指定 dtype。
  - NumPy 提供它自己的类型，例如 numpy.int32、numpy.int16 和 numpy.float64。
- **ndarray.itemsize**
  - 数组中每个元素的字节大小。例如，元素为 float64 类型的数组的 itemsize 为8（=64/8），而 int32 类型的数组的 itemsize 为4（=32/8）。它等于 ndarray.dtype.itemsize。

创建numpy数组

```
import numpy as np
origin_list = [2, 3, 4]
a = np.array(origin_list)
```

```
print(a.dtype)
print(a.shape)
print(a.ndim)
```

```
int64
(3,)
1
```

数组中元素类型的对象。

数组的维度。

数组的轴（维度）的个数

创建numpy数组

```
import numpy as np
origin_list = [2, 3, 4]
a = np.array(origin_list)
```

```
print(a.dtype)
print(a.shape)
print(a.ndim)
```

```
int64
(3,)
1
```

```
b = np.array([(1.5, 2, 3), (4, 5, 6)])
b
```

```
array([[1.5, 2. , 3. ],
       [4. , 5. , 6. ]])
```

```
print(b.dtype, b.shape, b.ndim)
```

```
float64 (2, 3) 2
```

创建numpy数组

```
import numpy as np
origin_list = [2, 3, 4]
a = np.array(origin_list)
```

```
print(a.dtype)
print(a.shape)
print(a.ndim)
```

```
int64
(3,)
1
```

```
c = np.array([[1, 2], [3, 4]], dtype=complex)
c
```

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

在创建时明确指定数组的类型

```
print(c.dtype, c.shape, c.ndim)
```

numpy内置数组

```
np.zeros((3, 4))
np.ones((2, 3, 4), dtype=np.int16)
```

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])

array([[[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]],

       [[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]]], dtype=int16)
```

- **zeros()**
  - 创建一个由 0 组成的数组
- **ones()**
  - 创建一个由 1 组成的数组，

```
1 temp = np.empty((2, 3))
2 print(temp)
```

```
[[1. 0. 0.]
 [0. 1. 2.]]
```

```
np.full((2, 3), 5.2)
```

```
array([[5.2, 5.2, 5.2],
       [5.2, 5.2, 5.2]])
```

- **empty()**
  - 内容是随机的并且取决于存储器的状态
- **full()**
  - 创建一个由指定数值组成的数组。默认情况下，创建的数组的 dtype 是 float64。

numpy内置数组

```
np.zeros_like(temp)
np.ones_like(temp)
np.full_like(temp, 2.34)
```

```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

```
array([[2.34, 2.34, 2.34],
       [2.34, 2.34, 2.34]])
```

- **zeros_like()**
  生成一个和指定数组一样大小的 0 数组。
- **ones_like()**
  生成一个和指定数组一样大小的全是 1 的数组。
- **full_like()**
  生成一个和指定数组一样大小，由指定数值组成的数组。

创建数组

- NumPy 提供了一个类似于 range 的函数 arange
  - 返回数组序列。

```
np.arange(6)
```

```
array([0, 1, 2, 3, 4, 5])
```

默认数值范围[0,6],递增1

创建数组

- NumPy 提供了一个类似于 range 的函数 arange
  - 返回数组序列。

```
np.arange(10, 30, 5)
```

```
array([10, 15, 20, 25])
```

数值范围[10,30],递增5

创建数组

- NumPy 提供了一个类似于 range 的函数 arange
  - 返回数组序列。

```
np.arange(10, 30, 5)
```

```
array([10, 15, 20, 25])
```

数值范围[10,30],递增5

```
# it accepts float arguments
np.arange(0, 2, 0.3)
```

```
array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8])
```

数值范围[0, 2],递增0.3

当 arange 与浮点参数一起使用时，由于浮点数的精度是有限的，通常不容易预测获得的元素数量。

创建数组

- NumPy 提供linspace 创建数字序列，指定范围和长度。

```
# 9 numbers from 0 to 2
np.linspace(0, 2, 9)
```

```
array([0.  , 0.25, 0.5 , 0.75, 1.  , 1.25, 1.5 , 1.75, 2.  ])
```

数值范围[0,2], 长度为9.

用reshape函数，从一维变换到多维

创建数组

- NumPy 提供linspace 创建数字序列，指定范围和长度。

```
# 9 numbers from 0 to 2
np.linspace(0, 2, 9)
```

```
array([0.  , 0.25, 0.5 , 0.75, 1.  , 1.25, 1.5 , 1.75, 2.  ])
```

数值范围[0,2], 长度为9.

用reshape函数，从一维变换到多维

```
b = np.arange(12).reshape(4, 3)
print(b)
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

创建数组

- NumPy 提供linspace 创建数字序列，指定范围和长度。

```
# 9 numbers from 0 to 2
np.linspace(0, 2, 9)
```

```
array([0.  , 0.25, 0.5 , 0.75, 1.  , 1.25, 1.5 , 1.75, 2.  ])
```

数值范围[0,2], 长度为9.

用reshape函数，从一维变换到多维

```
c = np.arange(24).reshape(2, 3, 4)
print(c)
```

```
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

数组的显示

- NumPy 数组以与嵌套列表类似的方式显示，但是具有以下布局：
  - 最后一个轴从左到右打印，
  - 倒数第二个从上到下打印，
  - 其余的也从上到下打印，每个切片与下一个用空行分开。
- 一维数组被打印为行、二维为矩阵和三维为矩阵列表。

```
1  a = np.arange(6)
2  print(a)
```

```
[0 1 2 3 4 5]
```

```
1  b = np.arange(12).reshape(4, 3)
2  print(b)
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

```
1  c = np.arange(24).reshape(2, 3, 4)
2  print(c)
```

```
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

课程内容

- **Numpy**
  - 创建numpy 对象
  - 索引、切片
  - 基本操作
  - 形状变换
  - 随机数函数
  - 存取操作
  - 其他操作
- **Matplotlib**

索引、切片

- 一维数组可以被索引，切片和迭代
  - 与 Python 序列类似

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

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

```
a[2]                    2
a[2:5]                  array([2, 3, 4])
```

索引、切片

- 一维数组可以被索引，切片和迭代
  - 与 Python 序列类似

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

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

```
a[2]                    2
a[2:5]                  array([2, 3, 4])
```

```
a[:6:2] = -1000   array([-1000,     1, -1000,     3, -1000,     5,     6,     7,     8,
a[::-1]                          9])

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

索引、切片

- 一维数组可以被索引，切片和迭代
  - 与 Python 序列类似

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

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

```
a[2]                    2
a[2:5]                  array([2, 3, 4])
```

```
a[:6:2] = -1000   array([-1000,     1, -1000,     3, -1000,     5,     6,     7,     8,
a[::-1]                          9])
```

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

使用列表作为切片的参数，索引指定元素：

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

索引、切片

- 多维（Multidimensional）数组
  - 每个轴可以有一个索引，索引是一个数组，数组每个元素对应一个轴

```
b = np.arange(0,20).reshape(5,4)
b
```

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

```
b[2, 3]
```
```
11
```

```
b[0:5, 1]
```
```
array([ 1, 5, 9, 13, 17])
```

```
b[:, 1]
```
```
array([ 1, 5, 9, 13, 17])
```

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

```
b[-1]
```
```
array([16, 17, 18, 19])
```

缺失的索引被认为是一个完整切片，不建议这种用法

索引、切片

- 三个点（ ... ）表示产生完整索引元组所需的冒号。

```
# a 3D array (two stacked 2D arrays)
c = np.array( [[[  0,  1,  2],
                [ 10, 12, 13]],
               [[100,101,102],
                [110,112,113]]])
c.shape
```

```
(2, 2, 3)
```

```
c[1,...]
```

```
array([[100, 101, 102],
       [110, 112, 113]])
```

```
c[...,2]
```

```
array([[  2,  13],
       [102, 113]])
```

基本操作

- 数组上的算术运算符按元素运算的
  - 运算后创建一个新的数组并填充结果。
  - 加、减、乘、除、幂运算等

```
a = np.array([20, 30, 40, 50])
b = np.arange(4)
```

```
b
```

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

```
c = a - b
c
```

```
array([20, 29, 38, 47])
```

```
b**2
```

```
array([0, 1, 4, 9])
```

基本操作

- 数组上的算术运算符按元素运算的
  - 运算后创建一个新的数组并填充结果。
  - 加、乘、除

```
x = np.array([1.0, 2.0, 3.0])
y = np.array([2.0, 4.0, 6.0])
x + y                                    # 对应元素的加法
x * y                                    # element-wise product
x / y
```

```
array([3., 6., 9.])

array([ 2.,  8., 18.])

array([0.5, 0.5, 0.5])
```

基本操作

- 数组上的算术运算符按元素运算的
  - 运算后创建一个新的数组并填充结果。
  - 布尔运算

```
a = np.array([20, 30, 40, 50])
```

使用不等号运算符, 会得到一个布尔型数组

```
a < 35
```

```
array([ True,  True, False, False])
```

基本操作

- 数组上的一元计算
  - 计算数组中所有元素的总和、查找最值、及最值索引

```
a = np.array([20, 30, 40, 50])
b = np.arange(4)
```

```
a.sum()                    # 所有元素求和
a.min()                    # 返回数组最小值
a.max()                    # 返回数组最大值
a.argmax()                 # 返回数组最大值的索引
```

```
140
20
50
3
```

基本操作

- 数组上的一元计算
  - 计算数组中所有元素的总和、查找最值、及最值索引
  - 对于高维数组，指定某一个轴进行操作

```
b = np.arange(12).reshape(3, 4)
b
```

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

```
b.sum(axis=0)                   # sum of each column
b.sum(axis=1)                   # sum of each row
b.min(axis=1)                   # min of each row
b.cumsum(axis=1)                # cumulative sum along each row
```

```
array([12, 15, 18, 21])
array([ 6, 22, 38])
array([0, 4, 8])
```

```
array([[ 0,  1,  3,  6],
       [ 4,  9, 15, 22],
       [ 8, 17, 27, 38]], dtype=int32)
```

基本操作

- 数组上的一元计算
  - 计算数组中所有元素的总和、查找最值、及最值索引
  - 对于高维数组，指定某一个轴进行操作

```
c = np.arange(24).reshape(2, 3, 4)
c
```

```
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]]])
```

```
c.argmax(axis=0)          # 沿轴axis=0方向数值最大值索引
c.argmax(axis=1)          # 沿轴axis=1方向数值最大值索引
c.argmax(axis=2)          # 沿轴axis=2方向数值最大值索引
```

基本操作

- 数组上的一元计算
  - 计算数组中所有元素的总和、查找最值
  - 对于高维数组，指定某一个轴进行操作

```
c = np.arange(24).reshape(2, 3, 4)
c
```

```
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]]])
```

```
c.argmax(axis=0)          # 沿轴axis=0方向数值最大值索引
c.argmax(axis=1)          # 沿轴axis=1方向数值最大值索引
c.argmax(axis=2)          # 沿轴axis=2方向数值最大值索引
```

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

常见数学函数

- Numpy 提供了常见的数学函数，如 sin，cos 和 exp。
- 这些函数在数组上按元素级别操作，产生一个数组作为输出。

```
B = np.arange(3)
B
```

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

```
np.exp(B)
```

```
array([1.        , 2.71828183, 7.3890561 ])
```

```
np.sqrt(B)
```

```
array([0.        , 1.        , 1.41421356])
```

```
C = np.array([2., -1., 4.])
np.add(B, C)
```

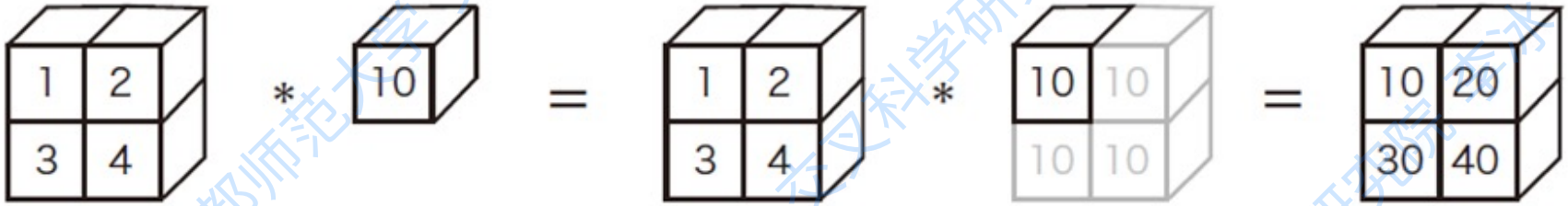
```
array([2., 0., 6.])
```

广播

- 形状不同的数组之间也可以进行运算。

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

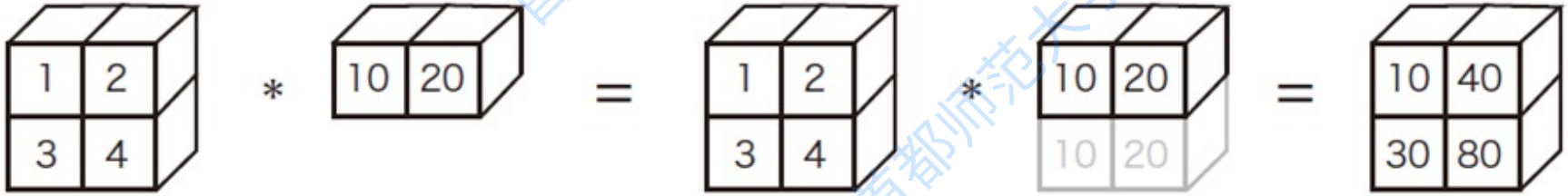
```
array([[10, 20], [30, 40]])
```




```
A = np.array([[1, 2], [3, 4]])
B = np.array([10, 20])
print(B.shape)
A * B
```

```
(2,)

array([[10, 40], [30, 80]])
```



浅拷贝和深拷贝

```
list1 = [2, 3, 4, 5]
list2 = list1
list2[0] = -1
print(list1, list2)
```

```
[-1, 3, 4, 5] [-1, 3, 4, 5]
```

```
list1 = [2, 3, 4, 5]
list2 = list1[:]
list2[0] = -1
print(list1, list2)
```

```
[2, 3, 4, 5] [-1, 3, 4, 5]
```

浅拷贝和深拷贝

```
a = np.arange(12).reshape(3, 4)
b = a[0,:]
print(a)
print(b)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[0 1 2 3]
```

```
b[0] = 99
print(a)
```

```
[[99  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

浅拷贝和深拷贝

```
a = np.arange(12).reshape(3, 4)
b = a[0,:]
print(a)
print(b)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[0 1 2 3]
```

```
b[0] = 99
print(a)
```

```
[[99  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

- 如果需要进行对象深拷贝，需要调用 copy 函数：

```
a = np.arange(12).reshape(3, 4)
b = a[0,:].copy()
b[0] = 99
print(a)
print(b)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[99  1  2  3]
```

迭代

- 迭代 Numpy 多维数组是相对于第一个轴完成的：

```
b = np.arange(12).reshape(3, 4)
for row in b:
    print(row, end='\n\n')
```

```
[0 1 2 3]

[4 5 6 7]

[ 8  9 10 11]
```

```
c = np.arange(24).reshape(2, 3, 4)
for row in c:
    print(row, end='\n\n')
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

[[12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
```

想要对数组中的每个元素执行操作，怎么做？

迭代

- 迭代 Numpy 多维数组是相对于第一个轴完成的：

```
b = np.arange(12).reshape(3, 4)
for row in b:
    print(row, end='\n\n')
```

```
[0 1 2 3]

[4 5 6 7]

[ 8  9 10 11]
```

```
for element in b.flat:
    print(element)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
```

迭代

- 迭代 Numpy 多维数组是相对于第一个轴完成的：

```
b = np.arange(12).reshape(3, 4)
for row in b:
    print(row, end='\n\n')
```

```
[0 1 2 3]

[4 5 6 7]

[ 8  9 10 11]
```

```
for element in b.flat:
    print(element)
```

```
for element in b.flatten():
    print(element)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
```

迭代

- numpy.nditer 提供了一种灵活访问一个或者多个数组的方式。
  - 单个数组的迭代（Single Array Iteration）
  - 完成对数组元素的访问，迭代器接口可以一个接一个地提供的每一个元素。

```
a = np.arange(10, 16).reshape(2,3)
print(a)
for x in np.nditer(a):
    print(x)
```

```
[[10 11 12]
 [13 14 15]]
10
11
12
13
14
15
```

```
it = np.nditer(a, flags=['multi_index'])    # 多重索引
while not it.finished:
    print(it.multi_index, it.value)
    it.iternext()
```

```
(0, 0) 10
(0, 1) 11
(0, 2) 12
(1, 0) 13
(1, 1) 14
(1, 2) 15
```

```
print(type(it))
dir(it)
```

```
<class 'numpy.nditer'>
…,'finished', …'has_index', 'has_multi_index',
'index', …'multi_index', 'ndim', 'nop',
'operands', …'shape', 'value']
```

迭代

- numpy.nditer 提供了一种灵活访问一个或者多个数组的方式。
  - 单个数组的迭代（Single Array Iteration）
  - 完成对数组元素的访问，迭代器接口可以一个接一个地提供的每一个元素。

```
it = np.nditer(a, flags=['c_index'])    # 单个索引
while not it.finished:
    print(it.index, it.value)
    it.iternext()
```

```
0 10
1 11
2 12
3 13
4 14
5 15
```

```
it = np.nditer(a, flags=['multi_index'])    # 多重索引
while not it.finished:
    print(it.multi_index, it.value)
    it.iternext()
```

```
(0, 0) 10
(0, 1) 11
(0, 2) 12
(1, 0) 13
(1, 1) 14
(1, 2) 15
```

```
print(type(it))
dir(it)
```

```
<class 'numpy.nditer'>
…,'finished', …'has_index', 'has_multi_index',
'index', …'multi_index', 'ndim', 'nop',
'operands', …'shape', 'value']
```

数组的形状变换

- 数组的形状可以通过各种函数进行更改。
  - ravel、reshape、T
  - 返回一个修改后的数组，但不会更改原始数组

```
arr = np.arange(12).reshape(3,4)
arr
```

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

```
arr.ravel()
```

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

```
arr.reshape(6, 2)
```

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

数组的形状变换

- 数组的形状可以通过各种函数进行更改。
  - ravel、reshape、T
  - 返回一个修改后的数组，但不会更改原始数组

```
arr = np.arange(12).reshape(3,4)
arr
```

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

```
arr.T
```

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

```
arr.T.shape
```

```
(4, 3)
```

```
arr.reshape(2,-1)
```

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

数组的形状变换

```
arr = np.arange(12).reshape(3,4)
arr
```

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

- 数组的形状可以通过各种函数进行更改。
  - ravel、reshape、T
- ravel() 与 flatten()的区别

```
second = arr.ravel()
third = arr.flatten()
```

```
[ 0 1 2 3 4 5 6 7 8 9 10 11]
```

```
second[0] = 88
print(arr)
print(second)
```

```
[[88  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[88  1  2  3  4  5  6  7  8  9 10 11]
```

```
third[0] = 99
print(arr)
print(third)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[99  1  2  3  4  5  6  7  8  9 10 11]
```

- ravel()：一般不会产生源数据的副本
- flatten()：返回源数据的副本

数组的堆叠

- 不同数组可以使用 vstack()、hstack()、dstack() 等函数沿不同的轴堆叠在一起。
  - 一般来说，对于具有两个以上维度的数组，vstack 沿第一轴堆叠，hstack 沿第二轴堆叠，dstack 沿第三轴堆叠。

数组的堆叠

- 不同数组可以使用 vstack()、hstack()、dstack() 等函数沿不同的轴堆叠在一起。

```
a = np.arange(0,6).reshape(2,3)
b = np.arange(6,12).reshape(2,3)
print(a)
print(b)
```

```
[[0 1 2]
 [3 4 5]]
[[ 6  7  8]
 [ 9 10 11]]
```

```
c = np.vstack((a,b))
print(c.shape)
print(c)
```

沿第一轴堆叠

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

```
d = np.hstack((a,b))
print(d.shape)
print(d)
```

沿第二轴堆叠

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

```
e = np.dstack((a,b))
print(e.shape)
print(e)
```

沿第三轴堆叠
没有第三轴，拼接

```
(2, 3, 2)
[[[ 0  6]
  [ 1  7]
  [ 2  8]]

 [[ 3  9]
  [ 4 10]
  [ 5 11]]]
```

课程内容

- **Numpy**
  - 创建numpy 对象
  - 索引、切片
  - 基本操作
  - 形状变换
  - 随机数函数
  - 存取操作
  - 其他操作
- **Matplotlib**

随机数函数

- numpy.random 模块提供了非常全的产生随机数据方法
  - 随机种子

```
np.random.seed(1234)      #设置随机种子为1234
```

- 随机数是由随机种子根据一定的计算方法计算出来的数值。所以，只要计算方法一定，随机种子一定，那么产生的随机数就不会变。
- 只要用户不设置随机种子，那么在默认情况下随机种子来自系统时钟（即定时/计数器的值）
- 随机数产生的算法与系统有关。即便是随机种子一样，Windows 系统和 Linux 系统产生的随机数也不一样。

随机数函数

- numpy.random 模块提供了非常全的产生随机数据方法
  - 随机种子

```
np.random.seed(1234)          #设置随机种子为1234
```

`np.random.rand(2, 3)` 产生一个给定形状的数组，数组中的值服从 [0, 1)之间的均匀分布。

`np.random.random((2,3))` 产生从 [0, 1)之间均匀抽样的数组。

`np.random.randn(3, 2)` 生成一个指定形状的数组，数组中的值服从标准正态分布。

随机数函数

- numpy.random 模块提供了非常全的产生随机数据方法
  - 随机种子

```
np.random.seed(1234)    #设置随机种子为1234
```

```
np.random.uniform(low=1, high=10, (3, 2))
```

•在区间 [low, high) 中均匀分布的数组。

```
np.random.normal(loc=1, scale=0.2, (3, 2))
```

•均值 μ=loc, 标准差 σ=scale 的正态分布数组。

```
np.random.randint(low=1, high=10, size=(3, 2))
```

在区间 [low, high)中离散均匀抽样的数组

```
np.random.choice(9)
```

返回产生一个随机数。

```
np.random.choice([0,2,4,6,8,10,12])
```

若a为单个int类型数，则选取range(a)中的数
a为数组，则从a中选取元素

存取操作

- NumPy 提供了存取数组内容的文件操作函数，存取数组数据的文件可以是二进制格式或者文本格式。
  - **tofile()：** 二进制格式写进文件，不保存数组形状和元素类型等信息
  - **fromfile()：** 用户指定元素类型，并对数组的形状进行适当的修改

```
a = np.arange(12).reshape(3, 4)
print(a)
a.tofile('array.raw')
b = np.fromfile('array.raw', dtype=np.int)
b
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

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

```
b = np.fromfile('array.raw', dtype=np.int).reshape(3,4)

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

存取操作

- NumPy 提供了存取数组内容的文件操作函数，存取数组数据的文件可以是二进制格式或者文本格式。
  - save()，load()
    - 存取 NumPy 专用的二进制格式保存数据（默认文件后缀名.npy），它们会自动处理元素类型和形状等信息。

```
np.save('arr.npy', a)
```

```
c = np.load('arr.npy')
c
```

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

存取操作

- NumPy 提供了存取数组内容的文件操作函数，存取数组数据的文件可以是二进制格式或者文本格式。
  - savetxt()，loadtxt()
    - 只能存取一维或二维数组的文本文件
    - 可以用来读写csv格式文件

```
# 默认按照'%.18e'格式保存数值，空格分隔
np.savetxt('a1.txt', a)
```

```
np.loadtxt('a1.txt')
```

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

```
np.savetxt('a2.txt', a, fmt='%d', delimiter=',')
```

```
np.loadtxt('a2.txt', delimiter=',')
```

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

其他操作

- 数据类型转换：astype()

```
arr = np.arange(24).reshape(2, 3, 4)
print(arr.dtype)
print(arr)
b = arr.astype(np.float64)
print(b.dtype)
print(b)
```

```
int64
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]

float64
[[[ 0.  1.  2.  3.]
  [ 4.  5.  6.  7.]
  [ 8.  9. 10. 11.]]

 [[12. 13. 14. 15.]
  [16. 17. 18. 19.]
  [20. 21. 22. 23.]]]
```

其他操作

- 矩阵转置 T、维度排序 transpose 和交换轴 swapaxes，
  - transpose
  - swapaxes 交换一对轴编号

```
arr = np.arange(24).reshape(2, 3, 4)
print(arr.T.shape)
print(arr.T)
```

```
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]

(4, 3, 2)
[[[ 0 12]
  [ 4 16]
  [ 8 20]]

 [[ 1 13]
  [ 5 17]
  [ 9 21]]

 [[ 2 14]
  [ 6 18]
  [10 22]]

 [[ 3 15]
  [ 7 19]
  [11 23]]]
```

其他操作

- 矩阵转置 T、维度排序 transpose 和交换轴 swapaxes，
  - transpose
  - swapaxes 交换一对轴编号

```
arr = np.arange(24).reshape(2, 3, 4)
```

```
arr.transpose(2, 1, 0)
```

```
arr.swapaxes(0, 1)
```

```
array([[[ 0, 12],
        [ 4, 16],
        [ 8, 20]],

       [[ 1, 13],
        [ 5, 17],
        [ 9, 21]],

       [[ 2, 14],
        [ 6, 18],
        [10, 22]],

       [[ 3, 15],
        [ 7, 19],
        [11, 23]]])
```

其他操作

- 矩阵转置 T、维度排序 transpose 和交换轴 swapaxes，
  - transpose
  - swapaxes 交换一对轴编号

```
arr = np.arange(24).reshape(2, 3, 4)
```

```
arr.transpose(2, 1, 0)
```

```
array([[[ 0,  1,  2,  3],
        [12, 13, 14, 15]],

       [[ 4,  5,  6,  7],
        [16, 17, 18, 19]],

       [[ 8,  9, 10, 11],
        [20, 21, 22, 23]]])
```

```
arr.swapaxes(0, 1)
```

课程内容

- Numpy
  - 创建数组
  - 索引、切片
  - 基本操作
    - 计算
    - 广播
    - 拷贝
    - 迭代
  - 形状变换
  - 随机数
  - 存取函数
  - 其他操作
- Matplotlib
  - 绘制
  - 显示

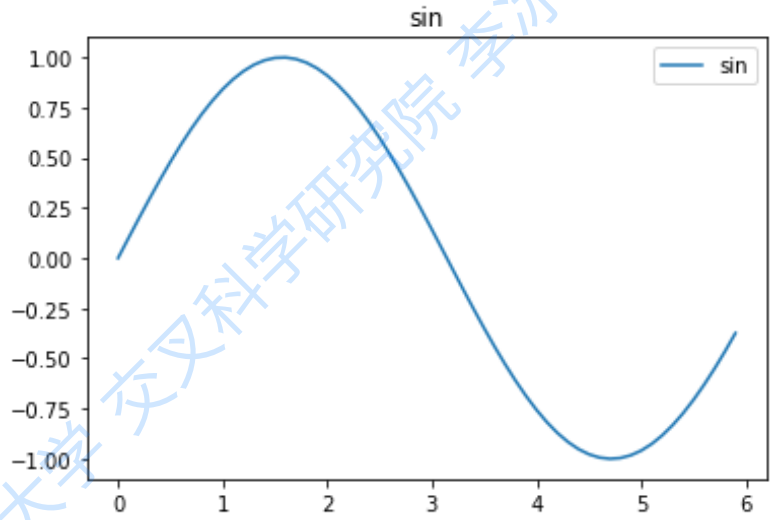
Matplotlib

- 绘制简单图形
  - 使用 matplotlib 的 pyplot 模块绘制图形

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


# 生成数据
x = np.arange(0, 6, 0.1)
y = np.sin(x)

# 绘制图形
plt.plot(x, y, label="sin")
plt.title('sin')          # 标题
plt.legend()
plt.show()
```



Matplotlib

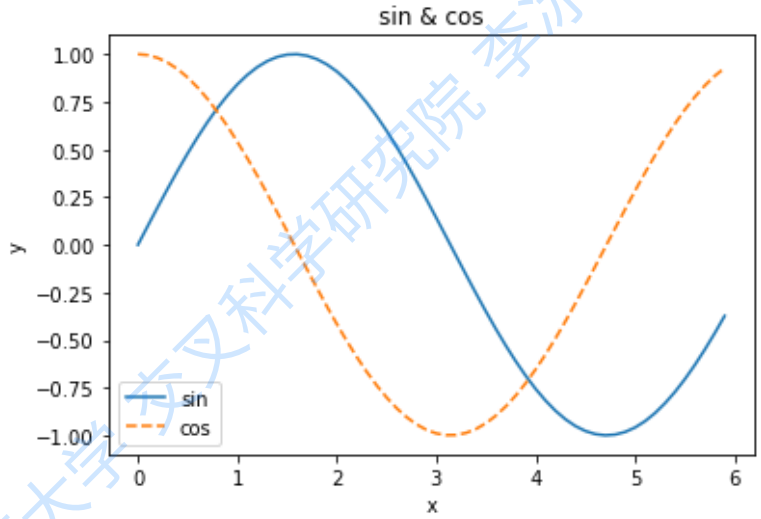
- 绘制简单图形
  - 使用 matplotlib 的 pyplot 模块绘制图形

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


# 生成数据
x = np.arange(0, 6, 0.1)
y1 = np.sin(x)
y2 = np.cos(x)

# 绘制图形
plt.plot(x, y1, label="sin")
plt.plot(x, y2, linestyle = "--", label="cos")
plt.xlabel("x")            # x轴标签
plt.ylabel("y")            # y轴标签

plt.title('sin & cos')  # 标题
plt.legend()
plt.show()
```



Matplotlib

- 绘制简单图形
  - 使用 matplotlib 的 pyplot 模块绘制图形

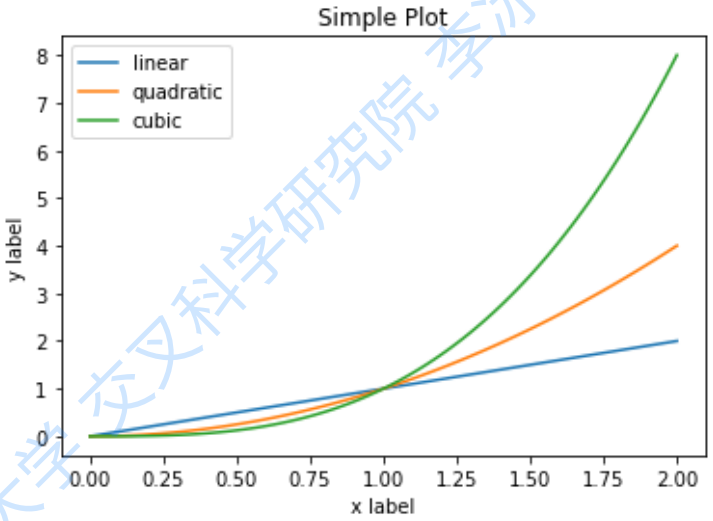
```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(0, 2, 100)

plt.plot(x, x, label='linear')
plt.plot(x, x**2, label='quadratic')
plt.plot(x, x**3, label='cubic')

plt.xlabel('x label')
plt.ylabel('y label')

plt.title("Simple Plot")
plt.legend()
plt.show()
```



Matplotlib

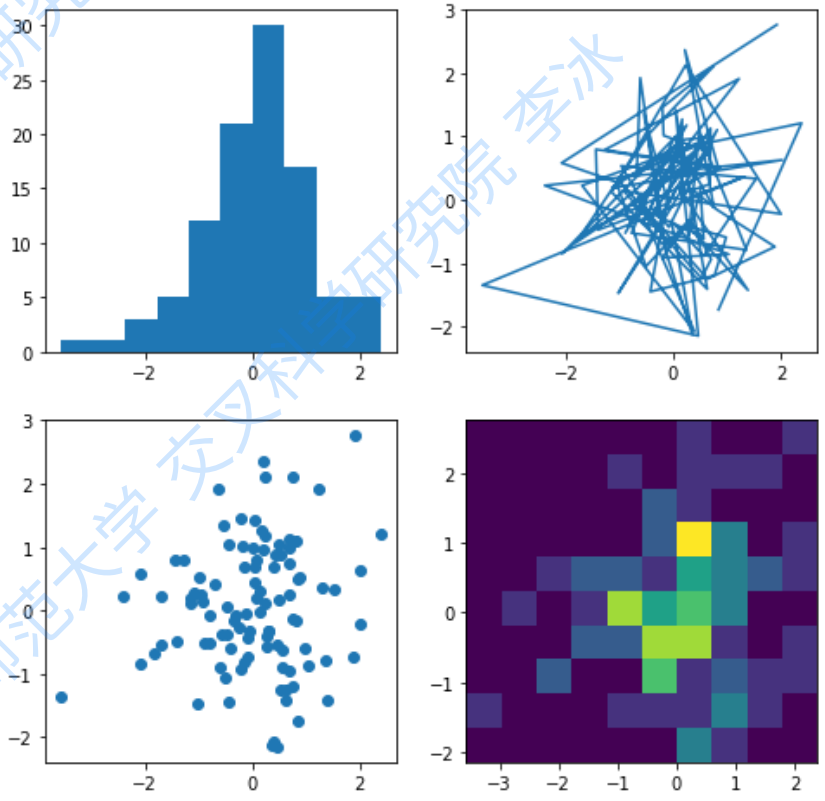
- 绘制简单图形
  - 在同一张图绘制多幅图

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


data = np.random.randn(2, 100)

fig, axs = plt.subplots(2, 2, figsize=(8, 8))
axs[0, 0].hist(data[0])
axs[1, 0].scatter(data[0], data[1])
axs[0, 1].plot(data[0], data[1])
axs[1, 1].hist2d(data[0], data[1])

plt.show()
```



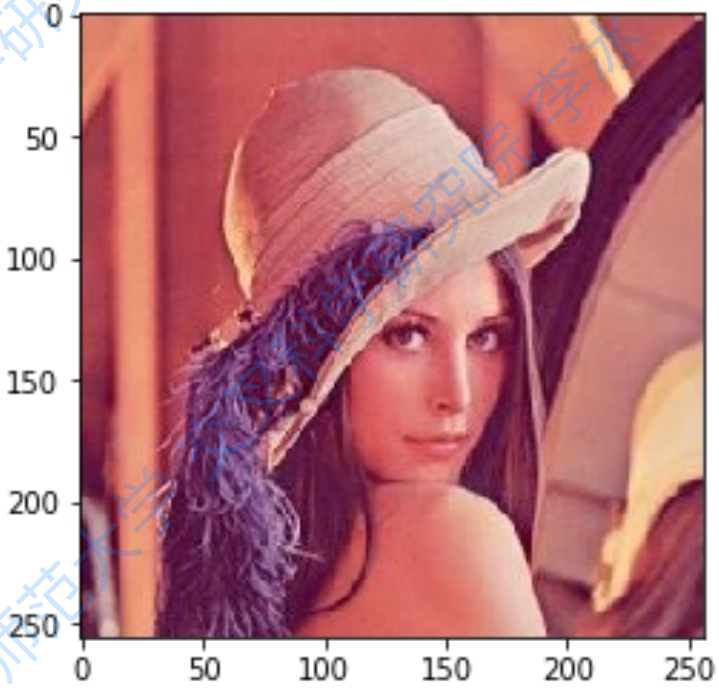
Matplotlib

- 显示图像
  - pyplot 中还提供了用于显示图像的方法 imshow()。
  - 可以使用 matplotlib.image 模块的 imread() 方法读入图像。

```
from matplotlib.image import imread

img = imread('images/chapter08/lena.jpg')

plt.imshow(img)
```



Matplotlib

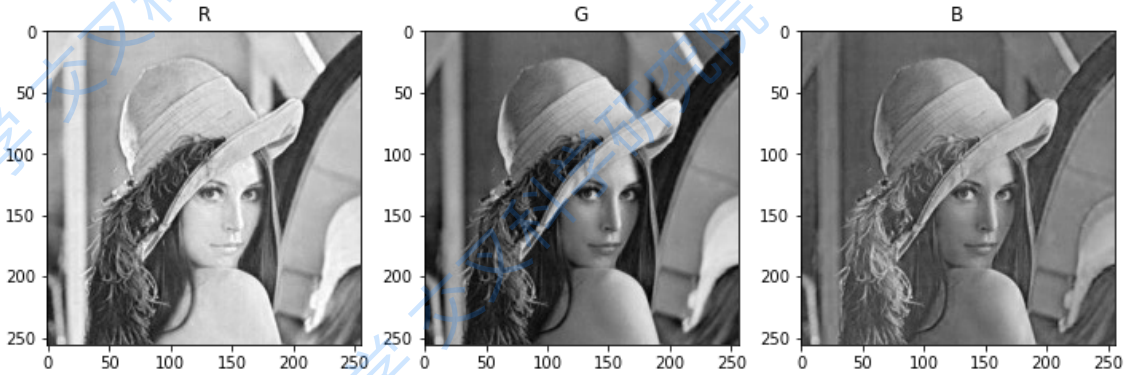
- 显示图像
  - pyplot 中还提供了用于显示图像的方法 imshow()。
  - 可以使用 matplotlib.image 模块的 imread() 方法读入图像。

```
fig = plt.figure(figsize=(12, 12))
fig.add_subplot(1, 3, 1)
plt.imshow(img[:,:,0], cmap='gray')
plt.title('R')

fig.add_subplot(1, 3, 2)
plt.imshow(img[:,:,1], cmap='gray')
plt.title('G')

fig.add_subplot(1, 3, 3)
plt.imshow(img[:,:,2], cmap='gray')
plt.title('B')

plt.show()
```



练习

1.交换二维矩阵的第一行和第二行
2.使用numpy.ndarray分片操作将图像水平镜像和垂直镜像
3.归一化二维矩阵，将其中每个元素都量化到0-1区间
4.将二维矩阵的每一行的元素都减去该行的平均值