

Python及其应用

主讲人：钱惠敏

E-mail: amandaqian@hhu.edu.cn

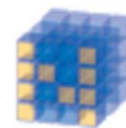
第10讲 Python科学计算生态圈

10.1 科学计算生态圈

10.2 Numpy

10.1 科学计算生态圈

➤ Python在科学计算方面有很多不断改良的库，使其在数据处理、交互探索性计算以及数据可视化方面深受广大编程者的喜爱。



NumPy
Base N-dimensional
array package



SciPy library
Fundamental
library for scientific
computing



Matplotlib
Comprehensive 2D
Plotting

IP[y]:
IPython

IPython
Enhanced
Interactive Console



Sympy
Symbolic
mathematics



pandas
Data structures &
analysis

➤ Python有着一个强大的科学计算生态圈，已经完全可以媲美MATLAB、R等特定编程语言/工具。

10.2 Numpy



- Numpy 是一个专门用于矩阵化运算、科学计算的开源Python库
- NumPy将Python相当于变成一种免费的更强大的Matlab系统
 - ✓ 强大的 ndarray 多维数组结构
 - ✓ 成熟的函数库
 - ✓ 用于整合C/C++和Fortran代码的工具包
 - ✓ 实用的线性代数、傅里叶变换和随机数模块
 - ✓ Numpy 和稀疏矩阵运算包scipy 配合使用非常方便

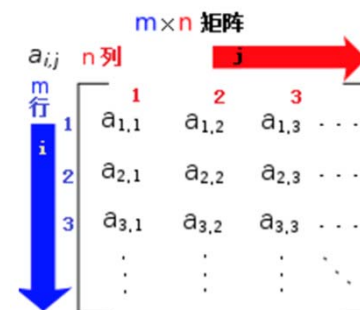
10.2 Numpy

➤ 基本数据结构 ndarray

✓ 矩阵表示：使用Numpy，易得到二维矩阵

```
import numpy as np
np_ar=np.array([[1,2,3],[4,5,6]])
print (type(np_ar))
print(np_ar)
```

```
<class 'numpy.ndarray' >
[[1 2 3]
 [4 5 6]]
```



10.2 Numpy

➤ 基本数据结构 ndarray

- ✓ 作为ndarray对象里的数据有时并不是所需要的，那么可以使用ndarray对象的`astype()` 方法转为指定的数据类型

```
import numpy as np
a=np.array(["1","2","3"])
print (a)
['1' '2' '3']
```

```
a_=a.astype("float")
print(a_)
[1.  2.  3.]
```

10.2 Numpy

➤ ndarray相关操作：索引

- ✓ 将数据转为ndarray对象后，会需要按某种方式来抽取数据
- ✓ ndarray对象提供了两种索引方式：
 - 切片索引：切片索引和对列表list的切片索引相似，不过由原本的一维切片变为多维
 - 布尔值索引：通过添加条件判断数组中每个值的 真/假 转为布尔值，再对原数组进行索引，为真 True 时会被抽取出来

10.2 Numpy

➤ ndarray相关操作：切片索引

```
import numpy as np
np_ar=np.array([[1,2,3],[4,5,6]])
print (type(np_ar))
print(np_ar)
<class 'numpy.ndarray' >      #通过shape属性得到数组的行数和列数
[[1 2 3]                      print(np_ar.shape)
 [4 5 6]                      (2, 3)
print (np_ar[:,2]) #切片索引
[[1 2]
 [4 5]]

np_ar[:,2]=0 #重新赋值
np_ar
array([[0, 0, 3],
       [0, 0, 6]])
```


10.2 Numpy

➤ ndarray相关操作：切片索引

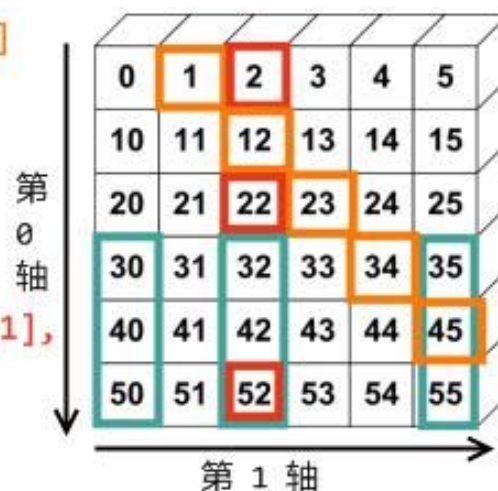
```
>>> a[0,3:5]
array([3,4])
>>> a[4:,:4:]
array([[44,45],[54,55]])
>>> a[:,2]
array([2,12,22,32,42,52])
>>> a[2::2,:2]
array([[20,22,24],
       [40,42,44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

10.2 Numpy

➤ ndarray相关操作：切片索引+布尔值索引

```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]  
array([1,12,23,34,45])  
>>> a[3:,[0,2,5]]  
array([[30,32,35],  
       [40,42,45],  
       [50,52,55]])  
>>> mask=np.array([1,0,1,0,0,1],  
                  dtype=np.bool)  
>>> a[mask,2]  
array([2,22,52])
```



10.2 Numpy

➤ ndarray相关操作：切分

```
sh_ndarray=np.array([[ 'date', 'open ', 'close', 'high'],  
[ '0', '3258', '3350', '3369'],  
[ '0', '3330', '3351', '3364'],  
[ '0', '3330', '3351', '3364']])
```

✓ 使用split将ndarray按照行平均分为几个ndarray

```
# 每两行作为一个新的array  
np.split(sh_ndarray,2)
```

```
[array([[ 'date', 'open ', 'close', 'high'], [ '0', '3258', '3350', '3369']], dtype='<U5'),  
array([[ '0', '3330', '3351', '3364'], [ '0', '3330', '3351', '3364']], dtype='<U5')]
```

10.2 Numpy

➤ ndarray相关操作：切分

```
sh_ndarray=np.array([[ 'date', 'open ', 'close', 'high'],  
[ '0', '3258', '3350', '3369'],  
[ '0', '3330', '3351', '3364'],  
[ '0', '3330', '3351', '3364']])
```

```
# 将第一行、最后一行以及剩下的行分为三个新的array  
np.split(sh_ndarray,[1,-1])
```

```
[array([[ 'date', 'open ', 'close', 'high']], dtype='<U5'),  
array([[ '0', '3258', '3350', '3369'], [ '0', '3330', '3351', '3364']], dtype='<U5'),  
array([[ '0', '3330', '3351', '3364']], dtype='<U5')]
```

10.2 Numpy

➤ ndarray相关操作：重构

```
sh_ndarray=np.array([[ 'date', 'open ', 'close', 'high'],  
[ '0', '3258', '3350', '3369'],  
[ '0', '3330', '3351', '3364'],  
[ '0', '3330', '3351', '3364']])
```

✓ 通过 reshape 方法将所有元素按照指定行指定列进行重构

```
# 将16个元素按照2行8列方式重构array  
sh_ndarray.reshape((2,8))
```

```
array([[ 'date', 'open ', 'close', 'high', '0', '3258', '3350', '3369'],  
[ '0', '3330', '3351', '3364', '0', '3330', '3351', '3364']], dtype='<U5')
```

10.2 Numpy

➤ ndarray相关操作：拼接

```
sh_ndarray=np.array([[ 'date', 'open ', 'close', 'high'],  
[ '0', '3258', '3350', '3369'],  
[ '0', '3330', '3351', '3364'],  
[ '0', '3330', '3351', '3364']])
```

```
# 使用sh_ndarray第一行进行横轴拼接操作  
h = np.hstack((sh_ndarray[0],sh_ndarray[0]))  
# 使用sh_ndarray第一行进行纵轴拼接操作  
v = np.vstack((sh_ndarray[0],sh_ndarray[0]))  
#  
c = np.concatenate((sh_ndarray[0],sh_ndarray[0]))  
print (h)  
print (h.shape)  
print (v)  
print (v.shape)  
print (c)
```

- ✓ 通过 hstack 沿横轴拼接
- ✓ 通过 vstack 沿纵轴拼接
- ✓ 通过 concatenate 进行拼接

```
['date' 'open ' 'close' 'high' 'date' 'open ' 'close' 'high']  
(8,)  
[['date' 'open ' 'close' 'high'] ['date' 'open ' 'close' 'high']]  
(2, 4)  
['date' 'open ' 'close' 'high' 'date' 'open ' 'close' 'high']
```

10.2 Numpy

➤ ndarray相关操作：转置

```
sh_ndarray=np.array([[ 'date', 'open ', 'close', 'high'],  
                     ['0', '3258', '3350', '3369'],  
                     ['0', '3330', '3351', '3364'],  
                     ['0', '3330', '3351', '3364']])
```

✓ 转置： transpose 方法、在数组后加 .T

```
print(sh_ndarray.transpose())
```

```
[[ 'date' '0' '0' '0']  
 ['open ' '3258' '3330' '3330']  
 ['close' '3350' '3351' '3351']  
 ['high' '3369' '3364' '3364']]
```

```
print(sh_ndarray.T)
```

```
[[ 'date' '0' '0' '0']  
 ['open ' '3258' '3330' '3330']  
 ['close' '3350' '3351' '3351']  
 ['high' '3369' '3364' '3364']]
```

10.2 Numpy

➤ ndarray相关操作：翻转

```
sh_ndarray=np.array([[ 'date', 'open ', 'close', 'high'],  
[ '0', '3258', '3350', '3369'],  
[ '0', '3330', '3351', '3364'],  
[ '0', '3330', '3351', '3364']])
```

✓ 翻转： `fliplr` 左右翻转、`flipud` 上下翻转

```
print(np.fliplr(sh_ndarray))
```

```
[[ 'high' 'close' 'open ' 'date']  
[ '3369' '3350' '3258' '0']  
[ '3364' '3351' '3330' '0']  
[ '3364' '3351' '3330' '0']]
```

```
print(np.flipud(sh_ndarray))
```

```
[[ '0' '3330' '3351' '3364']  
[ '0' '3330' '3351' '3364']  
[ '0' '3258' '3350' '3369']  
[ 'date' 'open ' 'close' 'high']]
```


10.2 Numpy

➤ 对位运算

- ✓ 指 ndarray 进行加减乘除运算时，使对应位置的数值进行加减乘除运算

```
a=np.array([[4,5,6],[7,8,9]])  
b=np.array([[2,3,4],[5,6,7]])  
print('a+b','\n',a+b)  
print('a-b','\n',a-b)  
print('a*b','\n',a*b)  
print('a/b','\n',a/b)
```

a+b

```
[[ 6 8 10]  
[12 14 16]]
```

a-b

```
[[2 2 2]  
[2 2 2]]
```

a*b

```
[[ 8 15 24]  
[35 48 63]]
```

a/b

```
[[2. 1.66666667 1.5 ]  
[1.4 1.33333333 1.28571429]]
```

10.2 Numpy

➤ 对位运算

- ✓ 当两个 ndarray 的维度不一致时，则没有对齐的维度上会分别执行对位运算，这种机制叫做广播（broadcasting）

#a与c的维度不一致，c将与a的每一维做运算

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

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

```
print('a+c','\n',a+b)
```

```
print('a-c','\n',a-b)
```

```
print('a*c','\n',a*b)
```

```
print('a/c','\n',a/b)
```

a+c

```
[[ 6  8 10]
```

```
 [12 14 16]]
```

a-c

```
[[2 2 2]
```

```
 [2 2 2]]
```

a*c

```
[[ 8 15 24]
```

```
 [35 48 63]]
```

a/c

```
[[2. 1.66666667 1.5 ]
```

```
 [1.4 1.33333333 1.28571429]]
```

10.2 Numpy

➤ 结构化的数据

数据sh_content

表 1 上证指数数据

	date	open	close	high	low	volume	code
0	2015-01-06 00:00:00	3330.8	3351.45	3394.22	3303.18	501661695.0	sh
1	2015-01-07 00:00:00	3326.65	3373.95	3374.9	3312.21	391918880.0	sh
2	2015-01-08 00:00:00	3371.96	3293.46	3381.57	3285.09	371131170.0	sh
3	2015-01-09 00:00:00	3276.97	3285.41	3404.83	3267.51	410240872.0	sh
4	2015-01-12 00:00:00	3258.21	3229.32	3275.19	3191.58	322064679.0	sh

```
import numpy as np
```

```
dtype = [('date', object), ('open', float), ('close', float), ('high', float), ('low', float), ('volume', float), ('code', str)] #设置数据类型
sh_content_tuple = [tuple(row) for row in sh_content[1:]] #将需要转为numpy.ndarray的数据转为元组tuple类型
sh_np = np.array(sh_content_tuple, dtype=dtype)
sh_np.dtype
```

```
dtype([('date', 'O'), ('open', 'f8'), ('close', 'f8'), ('high', 'f8'), ('low', 'f8'), ('volume', 'f8'), ('code', 'S')])
```

10.2 Numpy

➤ 结构化的数据

✓ 可实现按命名索引

```
sh_np["date"][:5]  #`sh_np["date"]`指抽取命名为`date`的那一列的前五个元素
```

```
array([datetime.datetime(2015, 1, 5, 0, 0),  
       datetime.datetime(2015, 1, 6, 0, 0),  
       datetime.datetime(2015, 1, 7, 0, 0),  
       datetime.datetime(2015, 1, 8, 0, 0),  
       datetime.datetime(2015, 1, 9, 0, 0)], dtype=object)
```

```
sh_np["volume"][:5]
```

```
array([[ 5.31352391e+08,   5.01661695e+08,   3.91918880e+08,  
        3.71131170e+08,   4.10240872e+08])
```

10.2 Numpy

➤ 内置操作函数

✓ 数学函数

- `sin(x[, out])` 正弦计算
- `cos(x[, out])` 余弦计算
- `tan(x[, out])` 正切计算
- `arcsin(x[, out])` 反正弦
- `arccos(x[, out])` 反余弦
- `arctan(x[, out])` 反正切
- `exp(x[, out])` 指数
- `log(x[, out])` 自然对数
- `log10(x[, out])` 10为底的对数
- `log2(x[, out])` 2为底的对数

✓ 运算函数

```
a=np.array([1,2,3,4,5])  
print(np.log(a))
```

✓ 统计函数

```
[0. 0.69314718 1.09861229 1.38629436 1.60943791]
```

10.2 Numpy

➤ 内置操作函数

✓ 数学函数

✓ 运算函数

✓ 统计函数

- 在计算的时候，会用到差分、累加的情况。Numpy提供很多相关的运算函数
- 以计算差分的`numpy.diff()`方法为例：

`numpy.diff()` 方法的语法为：

```
numpy.diff(a, n=1, axis=-1)
```

```
a=np.array([1,3,4,5,8])  
print(np.diff(a))
```

[2 1 1 3]

- `prod(a[, axis, dtype, out, keepdims])` 每个元素相乘
- `sum(a[, axis, dtype, out, keepdims])` 每个元素相加
- `cumprod(a[, axis, dtype, out])` 每个元素累乘
- `cumsum(a[, axis, dtype, out])` 每个元素累加
- `diff(a[, n, axis])` 相减

10.2 Numpy

➤ 内置操作函数

✓ 数学函数

✓ 运算函数

✓ 统计函数

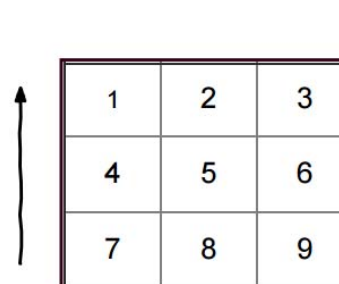
- 对于二维矩阵，需要指明是对哪一个维度进行差分

```
a=np.array([[1,2,3],[4,5,6],[7,8,9]])  
print(np.diff(a,axis=0))
```

```
[[3,3,3]  
 [3,3,3]]
```

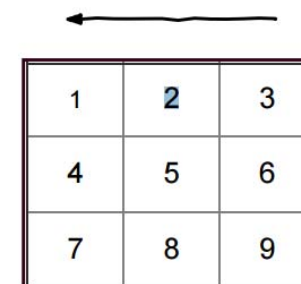
```
print(np.diff(a,axis=1))
```

```
[[1,1]  
 [1,1]  
 [1,1]]
```



1	2	3
4	5	6
7	8	9

axis=0



1	2	3
4	5	6
7	8	9

axis=1

10.2 Numpy

➤ 内置操作函数

✓ 数学函数

✓ 运算函数

✓ 统计函数

- numpy提供了很多计算最大值、最小值、均值、中位数等统计量的函数

- `amin(a[, axis, out, keepdims])` 返回最小值
- `amax(a[, axis, out, keepdims])` 返回最大值
- `percentile(a, q[, axis, out, ...])` 返回分位数
- `median(a[, axis, out, overwrite_input, keepdims])` 返回中位数
- `average(a[, axis, weights, returned])` 返回平均值，可以设定权重，计算权重平均
- `mean(a[, axis, dtype, out, keepdims])` 返回平均值
- `std(a[, axis, dtype, out, ddof, keepdims])` 计算标准差
- `var(a[, axis, dtype, out, ddof, keepdims])` 计算方差

10.2 Numpy

➤ 线性代数模块linalg

- Numpy 广泛应用于数值计算过程，其线性代数模块 linalg 常用于向量及矩阵的基本运算以及特征值分解、SVD分解等矩阵分解过程
- 特征值分解、Cholesky分解、SVD分解

```
c = np.array([[2,1],[1,2]])  
u,v = np.linalg.eig(c) # 特征值分解  
l = np.linalg.cholesky(c) # Cholesky分解  
U, s, V = np.linalg.svd(c) # SVD分解, 计算出的s需要重建为对角矩阵  
S = np.array([[s[0],0],[0,s[1]]])  
print S  
  
[[ 3.  0.]  
 [ 0.  1.]]
```

10.2 Numpy

➤ 线性代数模块linalg

如要求得 $Ax = b$ 的通解, 则可以使用SVD分解方法, 例如方程组为

$$2x_1 - 2x_2 - 4x_3 = 0$$

$$-x_1 + 3x_2 + 4x_3 = 0$$

$$x_1 - 2x_2 - 3x_3 = 0$$

```
a = np.array([[2,-2,-4],[-1,3,4],[1,-2,-3]])
b = np.array([0,0,0])
U, s, V = np.linalg.svd(a) # SVD分解
np.compress(s < 1e-10, V, axis=0) # 按照 s 对 v 进行切片

array([[ -0.57735027,  0.57735027, -0.57735027]])
```

10.2 Numpy

➤ 随机模块random

- 伪随机数的产生，可从离散分布和连续分布中产生
- 在蒙特卡洛方法、随机积分、随机过程模拟等很多方面都有应用
- 指定随机种子（seed）产生相同的随机数序列

```
from numpy import random

random.seed(666)
print random.rand(2,3) # 产生一个2行3列的矩阵，其中的每一个元素为[0,1)之间的浮点型随机数
print random.randint(0,10) # 产生一个[0,10)之间的整型随机数
```

```
[[ 0.70043712  0.84418664  0.67651434]
 [ 0.72785806  0.95145796  0.0127032 ]]
```

3

- random.seed(666) 指定全局种子，也可以使用 random.RandomState(666) 的方式指定局部种子，在指定局部种子的情况下，使用局部种子产生的随机数不改变

10.2 Numpy

➤ 随机模块random

✓ 常见分布的产生方式

```
print random.binomial(n=5,p=0.5,size=5) # 二项分布: 产生5个服从二项分布B(5,0.5)的样本
print random.uniform(-1,1,5) # 均匀分布: 产生5个服从均匀分布U[-1,1]的样本
print random.normal(size=(2,3)) # 标准正态分布: 产生2x5的标准正态分布样本
print random.normal(loc=0,scale=5,size=(2,3)) # 正态分布: 产生2x5的均值为0, 标准差为5的正态分布样本
```

```
[2 2 3 4 3]
[ 0.98306524 -0.93647313  0.08605806  0.01417688 -0.79867321]
[[-0.57957596 -0.53929631 -0.13996843]
 [-0.605991   -1.23679475 -0.9914476  ]]
[[ -5.06268003  -8.9900398    1.03876241]
 [  0.1611856    6.32885767 -12.79336444]]
```