Powerful data structure and software ecosystem

强大的数据结构和Python扩展库

Dazhuang@NJU
Department of Computer Science and Technology
Department of University Basic Computer Teaching

用Python玩转数据

1 为什么需要字典

为什么要使用字典?

某公司人事部门让技术部门用Python构建一个简易的员工信息表，包含员工的姓名和工资信息。根据信息表查询员工牛云的工资。

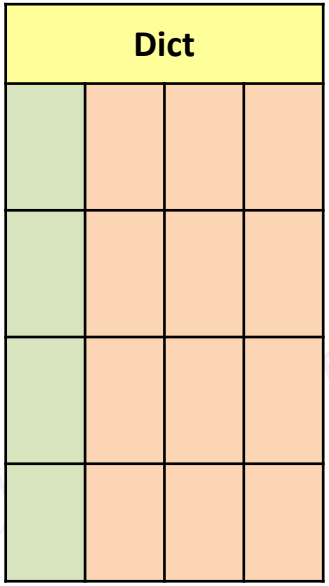
File

```
# Filename: info.py
names = ['Wangdachui', 'Niuyun', 'Linling', 'Tianqi']
salaries = [3000, 2000, 4500, 8000]
print(salaries[names.index('Niuyun')])
```

Output:
2000

salaries['Niuyun']

字典



- 什么是字典?

一种映射类型

- 键（key）
- 值（value）
- key-value对

创建字典

	Info
0	'Wangdachui'
1	'Niuyun',
2	'Linling'
3	'Tianqi'

- **创建字典**
  - 直接
  - 利用dict函数

cInfo['Niuyun']

Source

```
>>> aInfo = {'Wangdachui': 3000, 'Niuyun':2000, 'Linling':4500, 'Tianqi':8000}
>>> info = [('Wangdachui',3000), ('Niuyun',2000), ('Linling',4500), ('Tianqi',8000)]
>>> bInfo = dict(info)
>>> cInfo = dict([['Wangdachui',3000], ['Niuyun',2000], ['Linling',4500], ['Tianqi',8000]])
>>> dInfo = dict(Wangdachui=3000, Niuyun=2000, Linling=4500, Tianqi=8000)
```

```
{'Tianqi': 8000, 'Wangdachui': 3000, 'Linling': 4500, 'Niuyun': 2000}
```

创建字典

创建员工信息表时如何将所有员工的工资默认值设置为3000?

Source

```
>>> aDict = {}.fromkeys(('Wangdachui', 'Niuyun', 'Linling', 'Tianqi'),3000)
>>> aDict
{'Tianqi': 3000, 'Wangdachui': 3000, 'Niuyun': 3000, 'Linling': 3000}
```

sorted(aDict) = ?

['Linling', 'Niuyun', 'Tianqi', 'Wangdachui']

生成字典

已知有姓名列表和工资列表，如何生成字典类型的员工信息表?


```
>>> names = ['Wangdachui', 'Niuyun', 'Linling', 'Tianqi']
>>> salaries = [3000, 2000, 4500, 8000]
>>> dict(zip(names,salaries))
{'Tianqi': 8000, 'Wangdachui': 3000, 'Niuyun': 2000, 'Linling': 4500}
```

生成字典

对于几个公司的财经数据，如何构造公司代码和股票价格的字典？

```
{'AXP': '78.51', 'BA': '184.76', 'CAT ': '96.39', 'CSCO': '33.71', 'CVX': '106.09'}
```

```
lf = [('AXP', 'American Express Company', '78.51'),
      ('BA', 'The Boeing Company', '184.76'),
      ('CAT', 'Caterpillar Inc.', '96.39'),
      ('CSCO', 'Cisco Systems,Inc.', '33.71'),
      ('CVX', 'Chevron Corporation', '106.09')]
```

生成字典

File

```
# Filename: createdict.py
pList = …
aList = []
bList = []
for i in range(5):
    aStr = pList[i][0]
    bStr = pList[i][2]
    aList.append(aStr)
    bList.append(bStr)
aDict = dict(zip(aList,bList))
print(aDict)
```

```
pList = [('AXP', 'American Express Company', '78.51'),
         ('BA', 'The Boeing Company', '184.76'),
         ('CAT', 'Caterpillar Inc.', '96.39'), …]
```

```
{'AXP': '78.51', 'BA': '184.76', 'CAT ': '96.39', 'CSCO': '33.71', 'CVX': '106.09'}
```

用Python玩转数据

字典的使用

字典的基本操作

Source

```
>>> aInfo = {'Wangdachui': 3000, 'Niuyun':2000, 'Linling':4500, 'Tianqi':8000}
>>> aInfo['Niuyun']
2000
>>> aInfo['Niuyun'] = 9999
>>> aInfo
{'Tianqi': 8000, 'Wangdachui': 3000, 'Linling': 4500, 'Niuyun': 9999}
>>> aInfo['Fuyun'] = 1000
>>> aInfo
{'Tianqi': 8000, 'Fuyun': 1000, 'Wangdachui': 3000, 'Linling': 4500, 'Niuyun': 9999}
>>> 'Mayun' in aInfo
False
>>> del aInfo['Fuyun']
>>> aInfo
{'Tianqi': 8000, 'Wangdachui': 3000, 'Linling': 4500, 'Niuyun': 9999}
```

键值查找

更新

添加

成员判断

删除字典成员

字典的内建函数

dict()
len()
hash()

Source

```
>>> names = ['Wangdachui', 'Niuyun', 'Linling', 'Tianqi']
>>> salaries = [3000, 2000, 4500, 8000]
>>> aInfo = dict(zip(names, salaries))
>>> aInfo
{'Wangdachui': 3000, 'Linling': 4500, 'Niuyun': 2000, 'Tianqi': 8000}
>>> len(aInfo)
4
```

字典的内建函数

Source

```
>>> hash('Wangdachui')
7716305958664889313
>>> testList = [1, 2, 3]
>>> hash(testList)
Traceback (most recent call last):
  File "<pyshell#127>", line 1, in <module>
    hash(testList)
TypeError: unhashable type: 'list'
```

字典方法

已知有员工姓名和工资信息表{'Wangdachui':3000, 'Niuyun':2000, 'Linling':4500, 'Tianqi':8000}，如何单独输出员工姓名和工资金额?

Source

```
>>> aInfo = {'Wangdachui': 3000, 'Niuyun': 2000, 'Linling': 4500, 'Tianqi': 8000}
>>> aInfo.keys()
['Tianqi', 'Wangdachui', 'Niuyun', 'Linling']
>>> aInfo.values()
[8000, 3000, 2000, 4500]
>>> for k, v in aInfo.items():
        print(k, v)
```

字典方法

人事部门有两份人员和工资信息表，第一份是原有信息，第二份是公司中有工资更改人员和新进人员的信息，如何处理可以较快地获得完整的信息表?

Source

```
>>> aInfo = {'Wangdachui': 3000, 'Niuyun': 2000, 'Linling': 4500}
>>> bInfo = {'Wangdachui': 4000, 'Niuyun': 9999, 'Wangzi': 6000}
>>> aInfo.update(bInfo)
>>> aInfo
{'Wangzi': 6000, 'Linling': 4500, 'Wangdachui': 4000, 'Niuyun': 9999}
```

字典方法

下面两个程序都通过键查找值，区别在哪里？你更喜欢哪一个？

Source

```
>>> stock = {'AXP': 78.51, 'BA': 184.76}
>>> stock['AAA']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'AAA'
```

Source

```
>>> stock = {'AXP': 78.51, 'BA': 184.76}
>>> print(stock.get('AAA'))
None
```

字典方法

- 删除字典

Source

```
>>> aStock = {'AXP': 78.51, 'BA':184.76}
>>> bStock = aStock
>>> aStock = {}
>>> bStock
{'BA': 184.76, 'AXP': 78.51}
```

Source

```
>>> aStock = {'AXP': 78.51, 'BA': 184.76}
>>> bStock = aStock
>>> aStock.clear()
>>> aStock
{}
>>> bStock
{}
```

clear()	copy()	fromkeys()	get()	items()
keys()	pop()	setdefault()	update()	values()

常用字典方法

字典相关使用小案例

- **JSON格式**
  - JavaScript Object Notation，JS对象标记)
  - 一种轻量级的数据交换格式

```
'{"name":"Niuyun","address":
        {"city":"Beijing","street":"Chaoyang Road"}
 }'
```

解析后

```
>>> x = {"name":"Niuyun","address":
        {"city":"Beijing","street":"Chaoyang Road"}
        }
>>> x['address']['street']
'Chaoyang Road'
```

- **搜索引擎关键词查询**

百度
http://www.baidu.com/s?wd=%s
谷歌
http://www.googlestable.com/search/?q=%us
Bing
中国：http://cn.bing.com/search?q=%us
美国：http://www.bing.com/search?q=%us

```
>>> import requests
>>> kw = {'q': 'Python dict'}
>>> r = requests.get('http://cn.bing.com/search',
                     params = kw)
>>> r.url
>>> print(r.text)
```

可变长关键字参数（字典）

Python中函数的参数形式

- 位置或关键字参数
- 仅位置的参数
- 可变长位置参数
- 可变长关键字参数

（参数可以设定默认值）

Source

```
>>> def func(args1, *argst, **argsd):
        print(args1)
        print(argst)
        print(argsd)
>>> func('Hello,','Wangdachui','Niuyun','Linling',a1= 1,a2=2,a3=3)
Hello,
('Wangdachui', 'Niuyun', 'Linling')
{'a1': 1, 'a3': 3, 'a2': 2}
```

也可写成

```
>>> names = ['Wangdachuan', 'Liuyun', 'Linling']
>>> info = {'a1' : 1, 'a2' : 2, 'a3' : 3}
>>> greeting('Hello,', *names, **info)
```

用Python玩转数据

集合

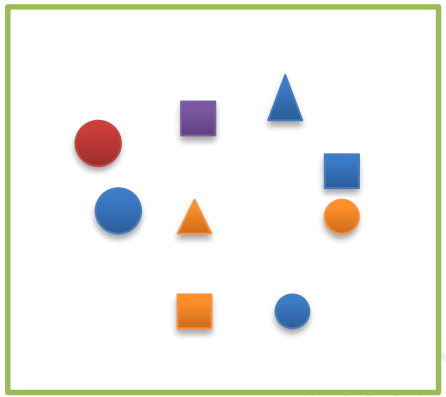
集合

人事部门的一份工资信息表登记时由于工作人员的疏忽有部分姓名重复登记了，如何快速解决这个问题?

Source

```
>>> names = ['Wangdachui', 'Niuyun', 'Wangzi', 'Wangdachui', 'Linling', 'Niuyun']
>>> namesSet = set(names)
>>> namesSet
{'Wangzi', 'Wangdachui', 'Niuyun', 'Linling'}
```

集合



- **什么是集合?**

一个无序不重复的元素的组合

- 可变集合（set）
- 不可变集合（frozenset）

集合的创建

Source

```
>>> aSet = set('hello')
>>> aSet
{'h', 'e', 'l', 'o'}
>>> fSet = frozenset('hello')
>>> fSet
frozenset({'h', 'e', 'l', 'o'})
>>> type(aSet)
<class 'set'>
>>> type(fSet)
<class 'frozenset'>
```

集合比较

Source

```
>>> aSet = set('sunrise')
>>> bSet = set('sunset')
>>> 'u' in aSet
True
>>> aSet == bSet
False
>>> aSet < bSet
False
>>> set('sun') < aSet
True
```

数学符号	Python符号
∈	in
∉	not in
=	==
≠	!=
⊂	<
⊆	<=
⊃	>
⊇	>=

标准类型运算符

集合关系运算

Source

```
>>> aSet = set('sunrise')
>>> bSet = set('sunset')
>>> aSet & bSet
{'u', 's', 'e', 'n'}
>>> aSet | bSet
{'e', 'i', 'n', 's', 'r', 'u', 't'}
>>> aSet - bSet
{'i', 'r'}
>>> aSet ^ bSet
{'i', 'r', 't'}
>>> aSet -= set('sun')
>>> aSet
{'e', 'i', 'r'}
```

数学符号	Python符号
∩	&
∪	|
- 或 \	-
Δ	^

集合类型运算符

运算符可复合

&=　|=　-=　^=

集合内建函数

- 函数也能完成以上的任务
  - 面向所有集合

s.issubset(t)
issuperset(t)
union(t)
intersection(t)
difference(t)
symmetric_difference(t)
copy()

Source

```
>>> aSet = set('sunrise')
>>> bSet = set('sunset')
>>> aSet.issubset(bSet)
False
>>> aSet.intersection(bSet)
{'u', 's', 'e', 'n'}
>>> aSet.difference(bSet)
{'i', 'r'}
>>> cSet = aSet.copy()
>>> cSet
{'s', 'r', 'e', 'i', 'u', 'n'}
```

集合内建函数

- 函数也能完成以上的任务
  - 面向可变集合

update(t)
intersection_update(t)
difference_update(t)
symmetric_difference_update(t)
add(obj)
remove(obj)
discard(obj)
pop()
clear()

Source

```
>>> aSet = set('sunrise')
>>> aSet.add('!')
>>> aSet
{'!', 'e', 'i', 'n', 's', 'r', 'u'}
>>> aSet.remove('!')
>>> aSet
{'e', 'i', 'n', 's', 'r', 'u'}
>>> aSet.update('Yeah')
>>> aSet
{'a', 'e', 'i', 'h', 'n', 's', 'r', 'u', 'Y'}
>>> aSet.clear()
>>> aSet
set()
```

用Python玩转数据

4 扩展库SCIPY

SciPy

特征

- 基于Python的软件生态系统
- 开源
- 主要为数学、科学和工程服务

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常用的数据结构

其他数据结构?

- **SciPy中的数据结构**

Python原有数据结构的变化

– ndarray（N维数组）

– Series（变长字典）

– DataFrame（数据框）

NumPy

特征

- 强大的ndarray对象和ufunc函数
- 精巧的函数
- 比较适合线性代数和随机数处理等科学计算
- 有效的通用多维数据，可定义任意数据类型
- 无缝对接数据库

Source

```
>>> import numpy as np
>>> xArray = np.ones((3,4))
```

SciPy库

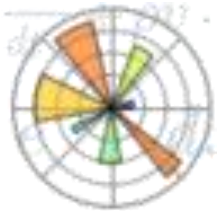
特征

- 基于NumPy，是Python中科学计算程序的核心包，与NumPy中的函数有交集，但更丰富，有些功能更强
- 有效计算numpy矩阵，让NumPy和SciPy协同工作
- 致力于科学计算中常见问题的各个工具箱，其不同子模块有不同的应用，如插值、积分、优化和图像处理等

Source

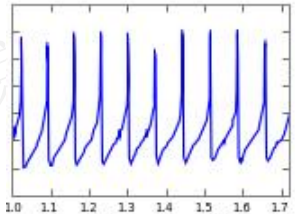
```
>>> import numpy as np
>>> from scipy import linalg
>>> arr = np.array([[1,2],[3,4]])
>>> linalg.det(arr)
-2.0
```

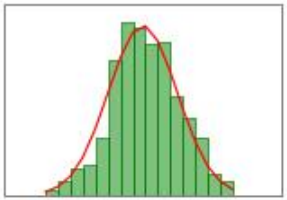
Matplotlib

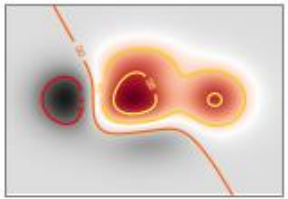


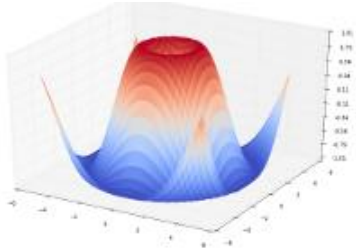
特征

- 基于NumPy
- 二维绘图库，简单快速地生成曲线图、直方图和散点图等形式的图
- 常用的pyplot是一个简单提供类似MATLAB接口的模块









pandas

特征

- 基于 SciPy 和 NumPy
- 高效的Series和DataFrame数据结构
- 强大的可扩展数据操作与分析的Python库
- 高效处理大数据集的切片等功能
- 提供优化库功能读写多种文件格式，如CSV、 HDF5

Source

```
...
>>> df[2 : 5]
>>> df.head(4)
>>> df.tail(3)
```

用Python玩转数据

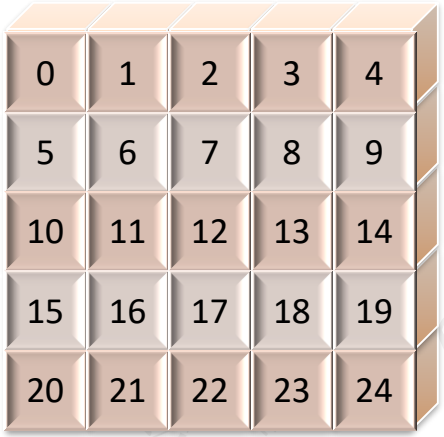
NDARRAY

Python中的数组

形式

- 用list和tuple等数据结构表示数组
  - 一维数组 list = [1,2,3,4]
  - 二维数组 list = [[1,2,3],[4,5,6],[7,8,9]]
- array模块
  - 通过array函数创建数组， array.array("B", range(5))
  - 提供append、 insert和read等方法

ndarray

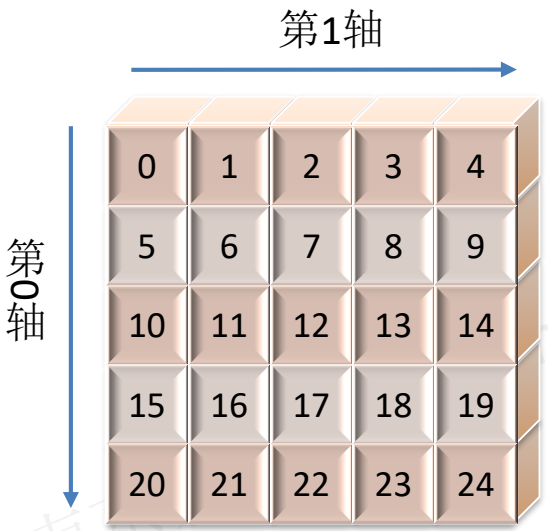


- **ndarray是什么?**

  N维数组

  - NumPy中基本的数据结构
  - 所有元素是同一种类型
  - 别名为array
  - 利于节省内存和提高CPU计算时间
  - 有丰富的函数

ndarray基本概念



- **ndarray数组属性**

  N维数组

  - 维度（dimensions）称为轴（axes），轴的个数称为秩（rank）
  - 基本属性
    - ndarray.ndim（秩）
    - ndarray.shape（维度）
    - ndarray.size（元素总个数）
    - ndarray.dtype（元素类型）
    - ndarray.itemsize（元素字节大小）

ndarray的创建

Source

```
>>> import numpy as np
>>> aArray = np.array([1,2,3])
>>> aArray
array([1, 2, 3])
>>> bArray = np.array([(1,2,3),(4,5,6)])
>>> bArray
array([[1, 2, 3],
       [4, 5, 6]])
>>> np.arange(1,5,0.5)
array([ 1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5])
>>> np.random.random((2,2))
array([[ 0.79777004,  0.1468679 ],
       [ 0.95838379,  0.86106278]])
>>> np.linspace(1, 2, 10, endpoint=False)
array([ 1. ,  1.1,  1.2,  1.3,  1.4,  1.5,  1.6,  1.7,  1.8,  1.9])
```

arange	**array**
copy	**empty**
empty_like	**eye**
fromfile	**fromfunction**
identity	**linspace**
logspace	**mgrid**
ogrid	**ones**
ones_like	**r**
zeros	**zeros_like**

ndarray创建函数

ndarray的创建

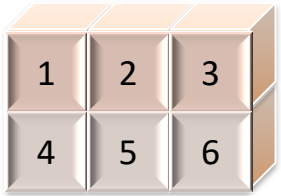
Source

```
>>> np.ones([2,3])
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> np.zeros((2,2))
array([[ 0.,  0.],
       [ 0.,  0.]])
>>> np.fromfunction(lambda i,j:(i+1)*(j+1), (9,9))
array([[  1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.],
       [  2.,   4.,   6.,   8.,  10.,  12.,  14.,  16.,  18.],
       [  3.,   6.,   9.,  12.,  15.,  18.,  21.,  24.,  27.],
       [  4.,   8.,  12.,  16.,  20.,  24.,  28.,  32.,  36.],
       [  5.,  10.,  15.,  20.,  25.,  30.,  35.,  40.,  45.],
       [  6.,  12.,  18.,  24.,  30.,  36.,  42.,  48.,  54.],
       [  7.,  14.,  21.,  28.,  35.,  42.,  49.,  56.,  63.],
       [  8.,  16.,  24.,  32.,  40.,  48.,  56.,  64.,  72.],
       [  9.,  18.,  27.,  36.,  45.,  54.,  63.,  72.,  81.]])
```

arange	array
copy	empty
empty_like	eye
fromfile	fromfunction
identity	linspace
logspace	mgrid
ogrid	ones
ones_like	r
zeros	zeros_like

ndarray创建函数

ndarray的操作



Source

```
>>> aArray = np.array([(1,2,3),(4,5,6)])
array([[1, 2, 3],
       [4, 5, 6]])
>>> print(aArray[1])
[4 5 6]
>>> print(aArray[0:2])
[[1 2 3]
 [4 5 6]]
>>> print(aArray[:,[0,1]])
[[1 2]
 [4 5]]
>>> print(aArray[1,[0,1]])
[4 5]
>>> for row in aArray:
        print(row)
[1 2 3]
[4 5 6]
```

ndarray的操作

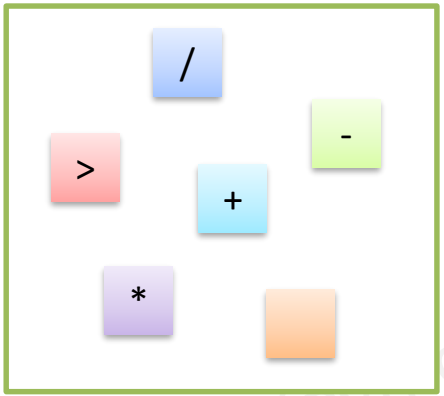
Source

```
>>> aArray = np.array([(1,2,3),(4,5,6)])
>>> aArray.shape
(2, 3)
>>> bArray = aArray.reshape(3,2)
>>> bArray
array([[1, 2],
       [3, 4],
       [5, 6]])
>>> aArray
array([[1, 2, 3],
       [4, 5, 6]])
```

Source

```
>>> aArray.resize(3,2)
>>> aArray
array([[1, 2],
       [3, 4],
       [5, 6]])
>>> bArray = np.array([1,3,7])
>>> cArray = np.array([3,5,8])
>>> np.vstack((bArray, cArray))
array([[1, 3, 7],
       [3, 5, 8]])
>>> np.hstack((bArray, cArray))
array([1, 3, 7, 3, 5, 8])
```

ndarray的运算



利用基本运算符

Source

```
>>> aArray = np.array([(5,5,5),(5,5,5)])
>>> bArray = np.array([(2,2,2),(2,2,2)])
>>> cArray = aArray * bArray
>>> cArray
array([[10, 10, 10],
       [10, 10, 10]])
>>> aArray += bArray
>>> aArray
array([[7, 7, 7],
       [7, 7, 7]])
>>> a = np.array([1,2,3])
>>> b = np.array([[1,2,3],[4,5,6]])
>>> a + b
array([[2, 4, 6],
       [5, 7, 9]])
```

ndarray的运算

Source

```
>>> aArray = np.array([(1,2,3),(4,5,6)])
>>> aArray.sum()
21
>>> aArray.sum(axis = 0)
array([5, 7, 9])
>>> aArray.sum(axis = 1)
array([ 6, 15])
>>> aArray.min()      # return value
1
>>> aArray.argmax()     # return index
5
>>> aArray.mean()
3.5
>>> aArray.var()
2.9166666666666665
>>> aArray.std()
1.707825127659933
```

sum	**mean**
std	**var**
min	**max**
argmin	**argmax**
cumsum	**cumprod**

利用基本数组统计方法

ndarray的专门应用—线性代数

Source

```
>>> import numpy as np
>>> x = np.array([[1,2], [3,4]])
>>> r1 = np.linalg.det(x)
>>> print(r1)
-2.0
>>> r2 = np.linalg.inv(x)
>>> print(r2)
[[-2.   1. ]
 [ 1.5 -0.5]]
>>> r3 = np.dot(x, x)
>>> print(r3)
[[ 7 10]
 [15 22]]
```

dot	矩阵内积
linalg.det	行列式
linalg.inv	逆矩阵
linalg.solve	多元一次方程组求根
linalg.eig	求特征值和特征向量

常用函数示例

ndarray的ufunc函数

- ufunc（universal function）是一种能对数组的每个元素进行操作的函数。NumPy内置的许多ufunc函数都是在C语言级别实现的，计算速度非常快。

add, all, any, arange, apply_along_axis, argmax, argmin, argsort, average, bincount, ceil, clip, conj, corrcoef, cov, cross, cumprod, cumsum, diff, dot, exp, floor, …

File

```
# Filename: math_numpy.py
import time
import math
import numpy as np
x = np.arange(0, 100, 0.01)
t_m1 = time.process_time()
for i, t in enumerate(x):
    x[i] = math.pow((math.sin(t)), 2)
t_m2 = time.process_time()
y = np.arange(0,100,0.01)
t_n1 = time.process_time()
y = np.power(np.sin(y), 2)
t_n2 = time.process_time()
print('Running time of math:', t_m2 - t_m1)
print('Running time of numpy:', t_n2 - t_n1)
```

用Python玩转数据

SERIES

Series

- **基本特征**
  - 类似一维数组的对象
  - 由数据和索引组成

Source

```
>>> import pandas as pd
>>> aSer = pd.Series([1,2.0,'a'])
>>> aSer
0   1
1   2
2   a
dtype: object
```

自定义Series的index

Source

```
>>> import pandas as pd
>>> bSer = pd.Series(['apple','peach','lemon'], index = [1,2,3])
>>> bSer
1    apple
2    peach
3    lemon
dtype: object
>>> bSer.index
Int64Index([1, 2, 3], dtype='int64')
>>> bSer.values
array(['apple', 'peach', 'lemon'], dtype=object)
```

Series的基本运算

Source

```
>>> aSer = pd.Series([3,5,7],index = ['a','b','c'])
>>> aSer['b']
5
>>> aSer * 2
a     6
b    10
c    14
dtype: int64
>>> import numpy as np
>>> np.exp(aSer)
a       20.085537
b      148.413159
c     1096.633158
dtype: float64
```

Series的数据对齐

Source

```
>>> data = {'AXP':'86.40','CSCO':'122.64','BA':'99.44'}
>>> sindex = ['AXP','CSCO','BA','AAPL']
>>> aSer = pd.Series(data, index = sindex)
>>> aSer
 AXP          86.40
CSCO         122.64
   BA          99.44
AAPL            NaN
dtype: object
>>> pd.isnull(aSer)
 AXP         False
CSCO         False
   BA         False
AAPL          True
dtype: bool
```

Series的数据对齐

- **重要功能**
  - 在运算中自动对齐不同索引的数据

Source

```
>>> aSer = pd.Series(data, index = sindex)
>>> aSer
 AXP      86.40
CSCO     122.64
  BA      99.44
AAPL        NaN
dtype: object
>>> bSer = {'AXP':'86.40','CSCO':'122.64','CVX':'23.78'}
>>> cSer = pd.Series(bSer)
>>> aSer + cSer
AAPL              NaN
 AXP        86.4086.40
  BA              NaN
CSCO    122.64122.64
 CVX              NaN
dtype: object
```

Series的数据对齐

- **重要功能**
  - 在运算中自动对齐不同索引的数据

Source

```
>>> data = {'AXP':86.40,'CSCO':122.64,'BA':99.44}
>>> aSer = pd.Series(data, index = sindex)
>>> aSer
 AXP         86.40
CSCO        122.64
   BA        99.44
 AAPL          NaN
dtype: object
>>> bSer = {'AXP':86.40,'CSCO':130.64,'CVX':23.78}
>>> cSer = pd.Series(bSer)
>>> (aSer+cSer)/2
AAPL          NaN
AXP         86.40
BA            NaN
CSCO       126.64
CVX           NaN
dtype: float64
```

用Python玩转数据

DATAFRAME

DataFrame

- **基本特征**
  - 一个表格型的数据结构
  - 含有一组有序的列（类似于index）
  - 大致可看成共享同一个index的Series集合

Source

```
>>> data = {'name': ['Wangdachui', 'Linling', 'Niuyun'], 'pay': [4000, 5000, 6000]}
>>> frame = pd.DataFrame(data)
>>> frame
        name   pay
0  Wangdachui  4000
1        Linling  5000
2        Niuyun  6000
```

DataFrame的索引和值

Source

```
>>> data = np.array([('Wangdachui', 4000), ('Linling', 5000), ('Niuyun', 6000)])
>>> frame =pd.DataFrame(data, index = range(1, 4), columns = ['name', 'pay'])
>>> frame
          name   pay
1  Wangdachui  4000
2      Linling  5000
3      Niuyun   6000
>>> frame.index
RangeIndex(start=1, stop=4, step=1)
>>> frame.columns
Index(['name', 'pay'], dtype='object')
>>> frame.values
array([['Wangdachui', '4000'],
       ['Linling', '5000'],
       ['Niuyun', '6000']], dtype=object)
```

DataFrame的基本操作

- 取DataFrame对象的列和行可获得Series

	name	pay
0	Wangdachui	4000
1	Linling	5000
2	Niuyun	6000

Source

```
>>> frame['name']
0   Wangdachui
1          Linling
2          Niuyun
Name: name, dtype: object
>>> frame.pay
0   4000
1   5000
2   6000
Name: pay, dtype: int64
```

Source

```
>>> frame.iloc[ : 2, 1]
0   4000
1   5000
Name: pay, dtype: object
```

DataFrame的基本操作

- DataFrame对象的修改和删除

Source

```
>>> frame['name'] = 'admin'
>>> frame
    name   pay
0  admin  4000
1  admin  5000
2  admin  6000
```

Source

```
>>> del frame['pay']
>>> frame
    name
0  admin
1  admin
2  admin

[3 rows x 1 columns]
```

DataFrame的统计功能

- DataFrame对象成员找最低工资和高工资人群信息

```
    name          pay
0    Wangdachui  4000
1    Linling          5000
2     Niuyun          6000
```

Source

```
>>> frame.pay.min()
'4000'
```

Source

```
>>> frame[frame.pay >= '5000']
     name    pay
1   Linling  5000
2  Niuyun  6000
```

小结


第四周 强大的数据结构和Python扩展库
4.1 为什么需要字典
概念，创建和生成字典
4.2 字典的使用
基本操作，内建函数和方法，应用
4.3 集合
概念，创建，比较，运算，内建函数
4.4 扩展库SciPy
功能简介，SciPy核心包介绍
4.5 ndarray
基本概念，创建，操作，运算，应用，ufunc函数
4.6 Series
概念，基本运算，数据对齐
4.7 DataFrame
概念，基本操作，简单统计
扩展
字典和集合编程示例介绍，NumPy常见应用选讲（单位数组，采样，布尔数组，矢量化运算与广播等）