Data Processing Using Python

Powerful Data Structure and Software Ecosystem

ZHANG Li/Dazhuang
Nanjing University
Department of Computer Science and Technology
Department of University Basic Computer Teaching

Data Processing Using Python

1 WHY DICTIONARY?

Why Dictionary?

Use Python to build a simple employee information table including names and salaries. Use the table to query salary of Niuyun.

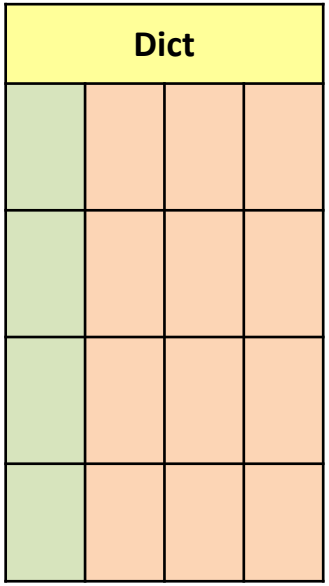
File

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

salaries['Niuyun']

Output:
2000

Dictionary



- **What is dictionary?**

  A mapping type

  - key
  - value
  - key-value pair

Create a Dictionary

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

- **Create a dictionary**
  - directly
  - Use dict()

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}
```

Create a Dictionary

How to set the default value of salary to be 3000?

Source

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

sorted(aDict) = ?

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

Generate a Dictionary

How to generate an employee information dictionary with name and salary list?

Source

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

Generate a Dictionary

How to generate a dictionary of company code and stock price from data?

{'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')]
```

Generate a Dictionary

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'}
```

Data Processing Using Python

USING DICTIONARY

Basic Operation of Dictionary

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}
```

Search by key

update

insert

Member identification

Delete

Built-in Functions of Dictionary

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
```

Built-in Functions of Dictionary

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'
```

Dictionary Methods

An information dictionary is known as {'Wangdachui':3000, 'Niuyun':2000, 'Linling':4500, 'Tianqi':8000}, how to output the name and salary of employee separately?

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)
```

Dictionary Methods

There are two dictionaries, the first one contains original information, while the second one has some new members and updates, how to merge and update information?

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}
```

Dictionary Methods

What's the difference between the two kinds of search operation?

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
```

Dictionary Methods

- Delete a dictiontary

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()

method

Case Study

- **JSON format**
  - JavaScript Object Notation
  - A lightweight data exchange format

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

after decode

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

- **Keyword query with search engine**

Baidu:
http://www.baidu.com/s?wd=%s
Google:
http://www.googlestable.com/search/?q=%us
Bing
China：http://cn.bing.com/search?q=%us
USA：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)
```

Variable Length Keyword Parameter(dict)

Parameter type in Python function:

- Position or keyword parameter
- Only position parameter
- Variable Length Position Parameter
- Variable length keyword parameter with default value

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}
```

Data Processing Using Python

SET

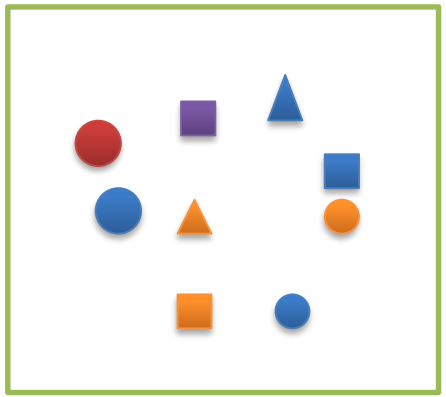
Set

How to remove the duplicate values in information form?

Source

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

Set



- **What is set?**

A combination of several unordered elements with no duplicate

  - Variable set（set）
  - Fixed set（frozenset）

Create a Set

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'>
```

Comparison between Sets

Source

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

Mathematic	Python
∈	in
∉	not in
=	==
≠	!=
⊂	<
⊆	<=
⊃	>
⊇	>=

Standard type operators

Relational Operation

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'}
```

Mathematic	Python
∩	&
∪	|
- or \	-
Δ	^

Set type operator

compound

assignment operators

&= |= -= ^=

Built-in Function for Set

- Function can also be used to do similar work
  - For all sets

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'}
```

Built-in Function for Set

- Function can also be used to do similar work
  - For variable sets

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()
```

Data Processing Using Python

4 SCIPY LIBRARY

SciPy

Feature

- A software ecosystem based on Python
- Open-source
- Serve for math, science and engineering

NumPy
Base N-dimensional array package

SciPy library
Fundamental library for scientific computing

Matplotlib
Comprehensive 2D Plotting

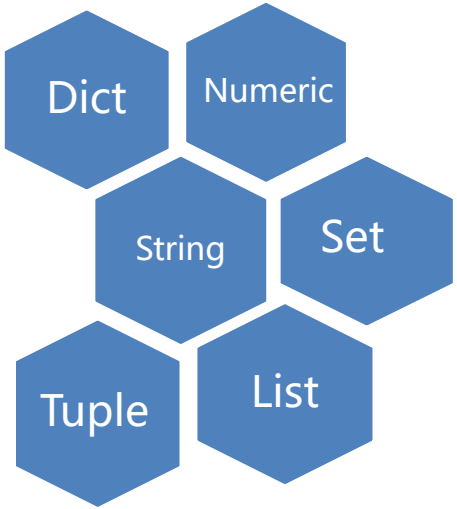
IP[y]: IPython

IPython
Enhanced Interactive Console

Sympy
Symbolic mathematics

pandas
Data structures & analysis

Common Data Type in Python



Other Data Structure

- **Data Structure in SciPy**

Modification based on original

Python data structure

  - ndarray（n-dimension array）
  - Series（dictionary with variable length）
  - DataFrame

NumPy

Feature

- Powerful ndarray object and ufunc() function
- Ingenious funciton
- Suitable for scientific computation like linear algebra and random number handling
- Flexible and available general multi-dimension data structure
- Easy to connect with database

Source

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

SciPy

Feature

- Key package for scientific computation in Python and it is based on NumPy. It includes richer functions and methods than NumPy and it probably has stronger function when they have the same functions or methods.
- Efficiently compute NumPy matrix to benefit collaboration between NumPy and SciPy.
- Toolbox to deal with different fields in scientific computation with modules including interpolation, integration, optimization and image processing.

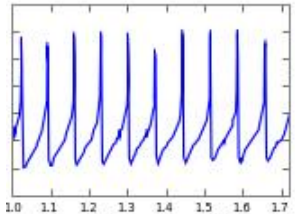
Source

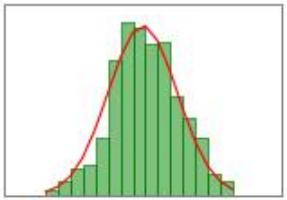
```
>>> import numpy as np
>>> from scipy import linalg
>>> arr = np.array([[1,2],[3,4]])
>>> linalg.det(arr)
-2.0
```

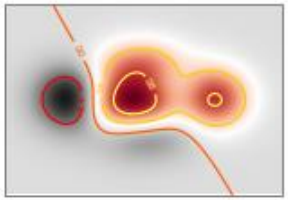
Matplotlib

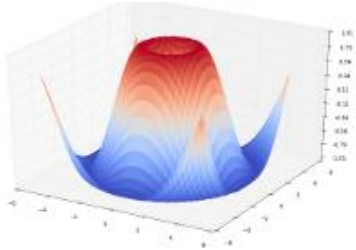
Feature

- Based on NumPy
- 2-dimensional graph library to rapidly generate all kinds of graphs
- Pyplot module provides MATLAB-like interface.









pandas

Feature

- Based on SciPy and NumPy
- Efficient Series and DataFrame structure
- Powerful Python library for scalable data processing
- Efficient solution for large dataset slides
- Optimized library function to read/write many types of files, like CSV and HDF5

Source

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

Data Processing Using Python

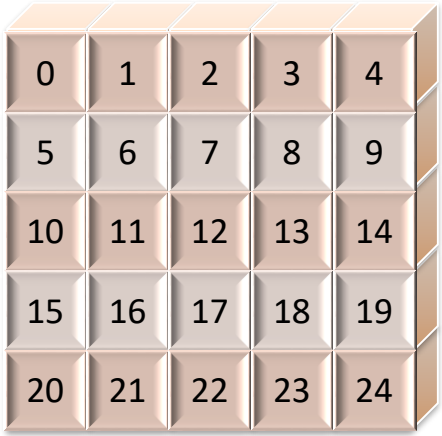
NDARRAY

Array in Python

Format

- Use data structure like **list** and **tuple**
  - One-dimensional array list = [1,2,3,4]
  - Two-dimensional array list = [[1,2,3],[4,5,6],[7,8,9]]
- **Array** module
  - Create array with **array**()， array.array("B", range(5))
  - Provide methods including **append**、 **insert** and **read**

Ndarray

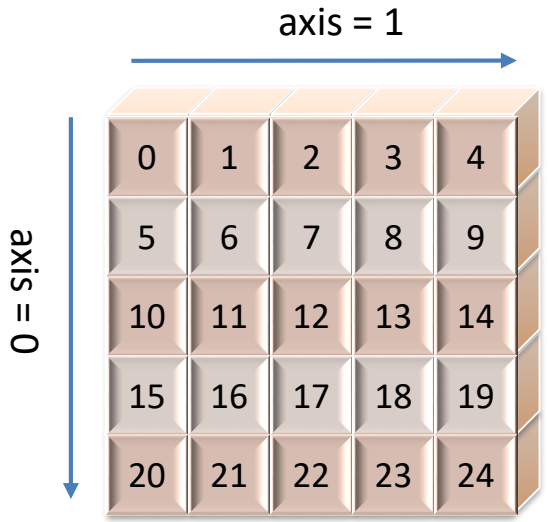


- **What is ndarray?**

  N-dimensional array

  - Basic data type in NumPy
  - Elements are of the same type
  - With another name array
  - Reduce memory cost and improve the computational efficiency
  - Powerful functions

Basic Concepts of Ndarray



- **Ndarray attributes**

  ndarray

  – Dimensions are called **axes**, the number of axes is **rank**.

  – Basic attributes

  - ndarray.ndim (rank)
  - ndarray.shape (dimension)
  - ndarray.size (total size)
  - ndarray.dtype (type of element)
  - ndarray.itemsize (size of item(in byte))

Creation of Ndarray

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 creation funciton

Creation of Ndarray

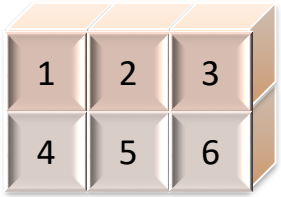
Source

```
>>> 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 creation funciton

Ndarray Operations



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 Operations

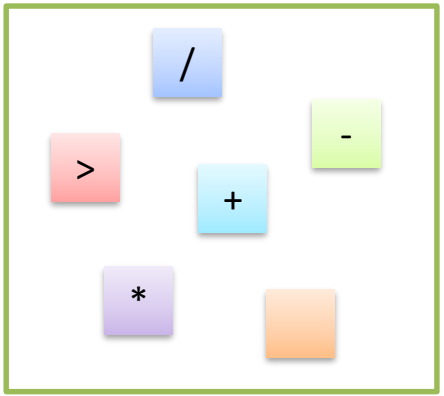
Source

```
>>> 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 Calculation



Use basic operators.

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 Calculation

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**

Use basic array statistic methods

Specific Application—Linear Algebra

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	**Inner product of matrix**
linalg.det	**Determinant**
linalg.inv	**Inverse matrix**
linalg.solve	**Multiple linear equation**
linalg.eig	**Eigenvalue and eigenvector**

Common Functions

ufunc() in Ndarray

- ufunc （universal function）

  can operate each element in

  the array. As many ufunc()s in

  NumPy are implemented by C,

  the speed can be fast.

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)
```

Data Processing Using Python

SERIES

Series

- **Basic feature**
  - Object similar to one-dimensional array
  - Consist of data and index.

Source

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

Index of Self-defined Series

Source

```
>>> 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)
```

Basic Operation of Series

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
```

Data Alignment of Series

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
```

Data Alignment of Series

- **Important feature**
  - Align data with different indexes during computation

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
```

Data Alignment of Series

- **Important feature**
  - Align data with different indexes during computation

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
```

Data Processing Using Python

7

DATAFRAME

DataFrame

- **Basic Feature**
  - A form-like data structure
  - Have an ordered column （like index）
  - Can be considered as a set of Series sharing the same index

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
```

Index and Value of Dataframe


```
>>> 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)
```

Basic Operation of DataFrame

- The query for row and column of DataFrame object returns **Series**

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
```

	name	pay
0	Wangdachui	4000
1	Linling	5000
2	Niuyun	6000

Source

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

Basic Operation of DataFrame

- Modification and deletion of DataFrame object

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]
```

Statistics with DataFrame

- Find groups with lowest and high salaries in DataFrame object members

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

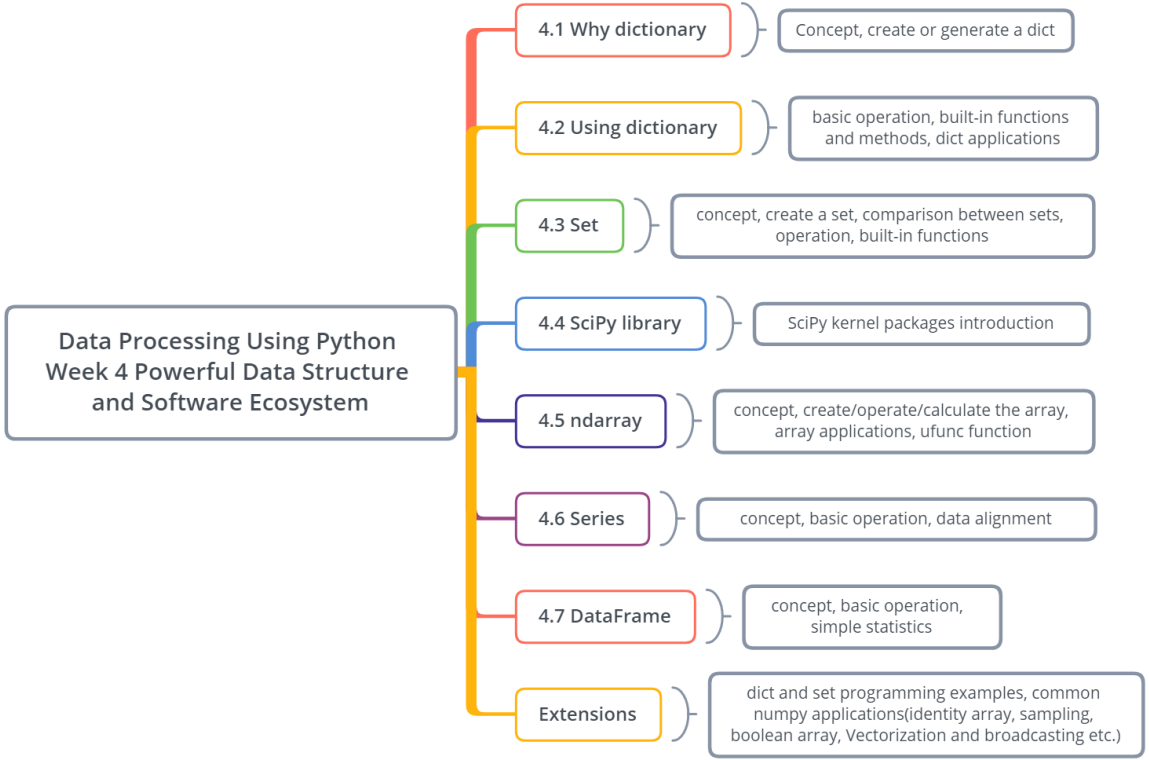
Source

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

Source

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

Summary


Data Processing Using Python
Week 4 Powerful Data Structure and Software Ecosystem
4.1 Why dictionary
Concept, create or generate a dict
4.2 Using dictionary
basic operation, built-in functions and methods, dict applications
4.3 Set
concept, create a set, comparison between sets, operation, built-in functions
4.4 SciPy library
SciPy kernel packages introduction
4.5 ndarray
concept, create/operate/calculate the array, array applications, ufunc function
4.6 Series
concept, basic operation, data alignment
4.7 DataFrame
concept, basic operation, simple statistics
Extensions
dict and set programming examples, common numpy applications(identity array, sampling, boolean array, Vectorization and broadcasting etc.)