

Data Analysis

Velen Kong

摘要

基于Python的数据分析入门笔记，希望能自学掌握基本的数据分析能力，毕竟是统计学的基础之一。

内容安排主要参考

(美) Wes McKinney 著. Python for Data Analysis:2nd Edition [M]
USA: O'Reilly Media 2017

该书作者同时也是pandas库的作者，同时借用其在GitHub上的数据资料。

第三方库与Python入门

IPython & Jupyter

用到的库: ipython, jupyter, numpy, matplotlib, pandas, scipy, scikit-learn, statsmodels。

jupyter确实好用，扩展了`Tab`, `?`, `*`的功能。

还有一些快捷键仅限IPython使用。

另外就是一些常用的Magic Command，比如

```
1 %run
2 %load
3 %debug
4 %xmode Plain # less detailed
5 %xmode Verbose # more detailed
6 %matplotlib inline # draw in Jupyter
7 import numpy as np
8 import matplotlib.pyplot as plt
9 plt.plot(np.random.rand(50).cumsum())
```

```
1 # ----- Day 0 -----
```

Python基础

#注释。

```
1 # this is comment
```

所有变量都是对象(object)。

基本的函数调用，利用函数批量处理。

```
1 def append_element(some_list, element):
2     some_list.append(element)
3     return
4
5 data = [1, 2, 3]
6 append_element(data, 4)
7 data
8 # Out: [1, 2, 3, 4]
```

变量赋值类似引用(reference)。

动态类型，利用type()和isinstance()进行类型检查。后者可以传入tuple代替逻辑或操作。

```
1 a = 1.5
2 isinstance(a, (int, float))
3 # Out: True
```

print配合format输出。

```
1 a = 1; b = 1.5
2 print('a is {1}, b is {0}'.format(type(b),
3                                     type(a)))
3 # Out: a is <class 'int'>, b is <class 'float'>
```

Python中的object拥有各自的属性(attributes)和方法(methods), 可通过getattr, setattr, setattr操作。其中getattr可以直接使用返回对象，setattr不改变原class。

```
1 class A(object):
2     def set(self, a, b):
3         x = a
4         a = b
5         b = x
6         print a, b
7
8 a = A()
9 c = getattr(a, 'set')
10 c(a='1', b='2')
11 # Out: 2 1
```

可以利用try()检查容器是否是顺序，同时生成list。

```
1 def isiterable(obj):
2     try:
3         iter(obj)
4         return True
5     except TypeError:
```

```
6         return False
7     return
8
9 x = 'string'
10 iterable(x)
11 # Out: True
12 iterable(5)
13 # Out: False
14
15 if not isinstance(x, list) and iterable(x):
16     x = list(x)
17 x
18 # Out: ['s', 't', 'r', 'i', 'n', 'g']
```

import和as的使用。

is和is not的使用。

```
1 a = [1, 2, 3]
2 b = a
3 c = list(a)
4 d = None
5 print(a is b)
6 # Out: True
7 print(a is not c)
8 # Out: True
9 print(a == c)
10 # Out: True
11 print(d is None)
12 # Out: True
```

大部分Python的容器都是可变的(mutable)，而strings和tuples不可变(immutable)。

```
1 a = ['foo', 2, [4, 5]]
2 a[2] = (3, 4)
3 a
4 # Out: ['foo', 2, (3, 4)]
```

```
1 # ----- Day 1 -----
```

标量类型(scalar types)一般有`None`, `str`, `bytes`, `float`, `bool`, `int`。

多行字符串用三个引号。

```
1 c = """
2 This is a longer string that
3 spans multiple lines
4 """
5 c.count('\n')
6 # Out: 3
```

`str()`方法可以生成字符串, `+`可以直接拼接字符串。而`replace()`方法可以替换字符串内容, 但不改变原串。

```
1 a = 'this is a string'
2 b = a.replace('string', 'longer string')
3 print(b)
4 # Out: this is a longer string
5 print(a)
6 # Out: this is a string
```

Python采用`Unicode`编码, 支持中文。对特殊字符可以采用`\`或者`r`进行转义。

```
1 s = '12\\34'
2 print(s)
3 # Out: 12\34
4 s = r'this\has\no\special\characters'
5 s
6 # Out: 'this\\has\\no\\special\\characters'
```

format和{}可以进行格式化输出。

```
1 template = '{0:.2f} {1:s} are worth US${2:d}'
2 template.format(4.5560, 'Argentine Pesos', 1)
3 # Out: '4.56 Argentine Pesos are worth US$1'
```

利用encode()和decode()函数可以对字符进行加解码。利用b生成二进制串。

```
1 val = "español"
2 val_utf8 = val.encode('utf-8')
3 val_utf8
4 # Out: b'espa\xc3\xb1ol'
5 type(val_utf8)
6 # Out: bytes
7 val_utf8.decode('utf-8')
8 # Out: 'español'
9 bytes_val = b'this is bytes'
10 bytes_val
11 # Out: b'this is bytes'
12 decoded = bytes_val.decode('utf8')
13 decoded
14 # Out: 'this is bytes'
```

None是一个单独的类型，常见于函数的默认参数中。

```
1 def add_and_maybe_multiply(a, b, c=None):
2     result = a + b
3     if c is not None:
4         result = result * c
5     return result
6
7 type(None)
8 # Out: NoneType
```

and和or运算。注意Python没有位运算操作符，&, |, ~都是逻辑运算。

Python自带时间与日期库。datetime是不可变类型，所以所有的操作都会产生新的对象而不改变原对象。

```
1 from datetime import datetime, date, time
2 dt = datetime(2011, 10, 29, 20, 30, 21)
3 dt.day
4 # Out: 29
5 dt.date()
6 # Out: datetime.date(2011, 10, 29)
7 dt.strftime('%m/%d/%Y %H:%M')
8 # Out: '10/29/2011 20:30'
9 datetime.strptime('20091031', '%Y%m%d')
10 # Out: datetime.datetime(2009, 10, 31, 0, 0)
11 dt.replace(minute=0, second=0)
12 # Out: datetime.datetime(2011, 10, 29, 20, 0)
```

时间差用timedelta表示，单位是天和秒。

```
1 dt2 = datetime(2011, 11, 15, 22, 30)
2 delta = dt2 - dt
3 delta
4 # Out: datetime.timedelta(17, 7179)
5 type(delta)
6 # Out: datetime.timedelta
7 dt + delta
8 # Out: datetime.datetime(2011, 11, 15, 22, 30)
```

时间的格式符号(format specification)如下

```
1 %Y #4位年数
2 %y #2位年数
3 %m #2位月数
4 %d #2位天数
5 %H #24小时制(2位)
6 %I #12小时制(2位)
7 %M #2位分钟数
8 %S #2位秒数，由于闰秒存在，范围为[00, 61]
```

```
9 %w #星期，范围为[0, 6]
10 %U #一年的第几个星期，以周日为标准，第一个周日之前的算00
11 %W #同上，以周一为标准
12 %z #用+HHMM或-HHMM表示相对于格林威治的时区偏移
13 %Z #时区名称，默认为空
14 %F #等价于%Y-%m-%d
15 %D #等价于%m/%d/%y
```

if, elif, else语句。单行if。

```
1 a = [1,2,3]
2 b = a if len(a) != 0 else ""
3 print(b)
4 # Out: [1, 2, 3]
```

for in语句。多变量for循环。

```
1 starts = [0,1,2,3,4]
2 ends = [5,6,7,8,9]
3 for start, end in zip(starts, ends):
4     print((start, end))
5 '''
6 Out:
7 (0, 5)
8 (1, 6)
9 (2, 7)
10 (3, 8)
11 (4, 9)
12 '''
```

while语句。

range()函数。

```
1 # ----- Day 2 -----
```


数据结构

Tuple是Python的基本数据结构之一，可用逗号分隔直接创建，或者分解顺序容器，也可以通过range构造序列。

```
1 tup = 3, (4, 5, 6), (7, 8)
2 tup
3 # Out: (3, (4, 5, 6), (7, 8))
4 tuple([4, 0, 2])
5 # Out: (4, 0, 2)
6 tuple('string')
7 # Out: ('s', 't', 'r', 'i', 'n', 'g')
8 tuple(['foo', [1, 2], True])
9 # Out: ('foo', [1, 2], True)
10 tuple(range(10))
11 # Out: (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

tuple也可以直接进行+,*运算，和解压操作。

```
1 (4, None, 'foo') + (6, 0) + ('bar',)
2 # Out: (4, None, 'foo', 6, 0, 'bar')
3 ('foo', 'bar') * 4
4 # Out: ('foo', 'bar', 'foo', 'bar', 'foo', 'bar',
5      'foo', 'bar')
5 tup = 4, 5, (6, 7)
6 a, b, (c, d) = tup
7 d
8 # Out: 7
9 a, b = 1, 2
10 b, a = a, b
11 print(a,b)
12 # Out: 2 1
```

也可以配合多变量for循环。

```
1 seq = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]
2 for a, b, c in seq:
3     print('a={0}, b={1}, c={2}'.format(a, b, c))
4 '''
5 Out:
6 a=1, b=2, c=3
7 a=4, b=5, c=6
8 a=7, b=8, c=9
9 '''
```

用*可以代指tuple的剩余部分。

```
1 values = 1, 2, 3, 4, 5
2 a, b, *rest = values
3 rest
4 # Out: [3, 4, 5]
```

count是tuple的常用方法。

```
1 a = (1, 2, 2, 2, 3, 4, 2)
2 a.count(2)
3 # Out: 4
```

List近似于可变的tuple，定义方法和tuple类似，也可以进行+,*操作。

利用append, insert, extend, pop, remove可以对list进行修改。但extend比+速度快。

```
1 tup = ('foo', 'bar', 'baz')
2 b_list = list(tup)
3 b_list
4 # Out: ['foo', 'bar', 'baz']
5 b_list.append('dwarf')
6 b_list
7 # Out: ['foo', 'bar', 'baz', 'dwarf']
8 b_list.insert(1, 'red')
```

```

9 b_list
10 # Out: ['foo', 'red', 'bar', 'baz', 'dwarf']
11 b_list.extend([7, 8, (2, 3)])
12 b_list
13 # Out: ['foo', 'red', 'bar', 'baz', 'dwarf', 7,
14         8, (2, 3)]
15 b_list.pop(2)
16 b_list
17 # Out: ['foo', 'red', 'baz', 'dwarf', 7, 8, (2,
18         3)]
19 b_list.append('foo')
20 b_list.remove('foo')
21 b_list
22 # Out: ['red', 'baz', 'dwarf', 7, 8, (2, 3),
23         'foo']

```

sort可以对一个list排序，bisect库可以进行二分操作。但bisect需要预先排序。sorted可以生成排序后的list。还可以设置排序的key。

```

1 import bisect
2 a = [7, 2, 2, 5, 1, 3]
3 a.sort()
4 a
5 # Out: [1, 2, 2, 3, 5, 7]
6 b = ['saw', 'small', 'He', 'foxes', 'six']
7 b.sort(key=len)
8 b
9 # Out: ['He', 'saw', 'six', 'small', 'foxes']
10 import bisect
11 bisect.bisect(a, 2)
12 # Out: 3
13 bisect.bisect(a, 6)
14 # Out: 5
15 bisect.insort(a, 6)
16 a
17 # Out: [1, 2, 2, 3, 5, 6, 7]

```

```

18 sorted('horse race')
19 # Out: [' ', 'a', 'c', 'e', 'e', 'h', 'o', 'r',
        'r', 's']
20 sorted('horse race', key = lambda x : -ord(x))
21 # Out: ['s', 'r', 'r', 'o', 'h', 'e', 'e', 'c',
        'a', ' ']

```

利用[begin: end: step]可以轻松实现部分list的选取和修改，其中step也可以为负数。reverse能反转整个容器。

```

1 list(reversed(range(10)))
2 # Out: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

```

enumerate和zip可以用于生成变量组合。

```

1 some_list = ['foo', 'bar', 'baz']
2 mapping = {}
3 for i, v in enumerate(some_list):
4     mapping[v] = i
5 mapping
6 # Out: {'foo': 0, 'bar': 1, 'baz': 2}
7 seq1 = ['foo', 'bar', 'baz']
8 seq2 = ['one', 'two', 'three']
9 zipped = zip(seq1, seq2)
10 list(zipped)
11 # Out: [('foo', 'one'), ('bar', 'two'), ('baz',
        'three')]
12 seq3 = [False, True]
13 list(zip(seq1, seq2, seq3))
14 # Out: [('foo', 'one', False), ('bar', 'two',
        True)]
15 for i, (a, b) in enumerate(zip(seq1, seq2)):
16     print('{0}: {1}, {2}'.format(i, a, b))
17 '''
18 Out:
19 0: foo, one
20 1: bar, two

```

```
21 2: baz, three
22 '''
```

zip也可以反过来进行解压。

```
1 pitchers = [('Nolan', 'Ryan'), ('Roger',
  'Clemens'),
2             ('Schilling', 'Curt')]
3 first_names, last_names = zip(*pitchers)
4 first_names
5 # Out: ('Nolan', 'Roger', 'Schilling')
```

```
1 # ----- Day 3 -----
```

字典(dict)是Python最重要的数据结构之一，存储键值对(key-value pair)，基本操作如下。

```
1 d1 = {'a' : 'some value', 'b' : [1, 2, 3, 4]}
2 d1
3 # Out: {'a': 'some value', 'b': [1, 2, 3, 4]}
4 d1['b']
5 # Out: [1, 2, 3, 4]
6 d1['dummy'] = 'another value'
7 d1
8 # Out: {'a': 'some value', 'b': [1, 2, 3, 4],
  'dummy': 'another value'}
9 del d1['a']
10 d1
11 # Out: {'b': [1, 2, 3, 4], 'dummy': 'another
  value'}
12 ret = d1.pop('dummy')
13 ret
14 # Out: 'another value'
15 d1
16 # Out: {'b': [1, 2, 3, 4]}
17 'b' in d1
18 # Out: True
```

```

19 list(d1.keys())
20 # Out: ['b']
21 list(d1.values())
22 # Out: [[1, 2, 3, 4]]
23 d1['b'] = 1
24 d1
25 # Out: {'b': 1}
26 d1.update({'b': 2, 2: ['a', 'b']})
27 d1
28 # Out: {'b': 2, 2: ['a', 'b']}

```

dict()可以直接生成dict。

```

1 map1 = dict(zip(range(5), range(5, -5, -1)))
2 map1
3 # Out: {0: 5, 1: 4, 2: 3, 3: 2, 4: 1}
4 map2 = dict(zip(range(10), range(5, 0, -1)))
5 map2
6 # Out: {0: 5, 1: 4, 2: 3, 3: 2, 4: 1}

```

读取dict值时还可以利用get设置默认值。如果不使用get, 则会发生KeyError, 比如pop()和del。所以使用dict前务必用in检查key是否存在。

```

1 # value = some_dict.get(key, default_value)

```

在使用list作为值时, 要注意将default设为空list。可以使用setdefault方法或者defaultdict类型。

```

1 words = ['apple', 'bat', 'bar', 'atom', 'book']
2 by_letter = {}
3 for word in words:
4     letter = word[0]
5     by_letter.setdefault(letter, []).append(word)
6 by_letter
7 # Out: {'a': ['apple', 'atom'], 'b': ['bat',
    'bar', 'book']}

```

```

8 from collections import defaultdict
9 by_letter = defaultdict(list)
10 by_letter
11 # Out: defaultdict(list, {})
12 for word in words:
13     by_letter[word[0]].append(word)
14 by_letter
15 # Out: defaultdict(list, {'a': ['apple', 'atom'],
    'b': ['bat', 'bar', 'book']})

```

dict利用了hash(), 所以key必须是可以被hash的对象。例如list无法作为key使用, 需要转化为tuple。因为在list变化后, hash值也被改变, 因而dict中保存的list将找不到对应的value产生错误。

set是无序无重复的容器。利用set()或者{}来生成set。

set可以利用union(), intersection(), difference(), symmetric_difference()来进行操作, 也可以使用|, &, -, ^。

常见方法有:

add(), clear(), remove(), pop(), copy(), len();

union(), update(), intersection(), intersection_update(), difference(), difference_update(), symmetric_difference(), symmetric_difference_update();

issubset(), issuperset(), isdisjoint()。

和dict类似, remove()和pop()需要先用in检查, 否则会抛出KeyError。

同样, set也利用了hash, 所以也不能加入list。

for in if语句常被用于list, dict, set中。可以轻松进行条件筛选。

```

1 strings = ['a', 'as', 'bat', 'car', 'dove',
  'python']
2 [x.upper() for x in strings if len(x) > 2]

```

```

3 # Out: ['BAT', 'CAR', 'DOVE', 'PYTHON']
4 unique_lengths = {len(x) for x in strings}
5 unique_lengths
6 # Out: {1, 2, 3, 4, 6}
7 loc_mapping = {val : index for index, val in
    enumerate(strings)}
8 loc_mapping
9 # Out: {'a': 0, 'as': 1, 'bat': 2, 'car': 3,
    'dove': 4, 'python': 5}
10 all_data = [['John', 'Emily', 'Michael', 'Mary',
    'Steven'],
11             ['Maria', 'Juan', 'Javier',
    'Natalia', 'Pilar']]
12 result = [name for names in all_data for name in
    names
13            if name.count('e') >= 1]
14 result
15 # Out: ['Michael', 'Steven', 'Javier']

```

也可以使用map函数，对顺序容器做一个映射。

```

1 set(map(len, strings))
2 # Out: {1, 2, 3, 4, 6}
3 list(map(lambda k: k ** 2, [1, 2, 3, 4, 5]))
4 # Out: [1, 4, 9, 16, 25]
5 dict(map(lambda x, y: (x, y), [1, 3, 5, 7, 9],
    [2, 4, 6, 8, 10]))
6 # Out: {1: 2, 3: 4, 5: 6, 7: 8, 9: 10}

```

多重for in语句。

```

1 some_tuples = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]
2 [[x for x in tup] for tup in some_tuples]
3 # Out: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

```

```

1 # ----- Day 4 -----

```


函数

函数定义时可以设置默认参数。调用时可以无视默认参数。也可以乱序设定参数。

定义在函数内部的本地变量会在return时被销毁。但可以使用global来定义。但不建议过多使用全局变量，那说明需要使用class和面向对象编程(object-oriented programming)。

Python可以返回多个参数。

```
1 def f():
2     a = 5
3     b = 6
4     c = 7
5     return a, b, c
6
7 c = 1
8 a, b, d = f()
9 print(a, b, c)
10 # Out: 5 6 1
```

Python中函数也是一种对象。由此可以将不同函数进行组合。

```
1 import re
2 states = ['Alabama ', 'Georgia!', 'Georgia',
3           'georgia',
4           'FLorida', 'south carolina##', 'West
5           virginia?']
6
7 def remove_punctuation(value):
8     return re.sub('[!#?]', '', value)
9
10 clean_ops = [str.strip, remove_punctuation,
11              str.title]
12
13 def clean_strings(strings, ops):
```

```

11     result = []
12     for value in strings:
13         for function in ops:
14             value = function(value)
15             result.append(value)
16     return result
17
18 clean_strings(states, clean_ops)
19 '''
20 Out:
21 ['Alabama',
22  'Georgia',
23  'Georgia',
24  'Georgia',
25  'Florida',
26  'South Carolina',
27  'West Virginia']
28 '''

```

在数据分析中，还会经常使用匿名函数(anonymous function)。其本身在创建时没有__name__属性。

```

1 anon = lambda x: x * 2
2 anon(4)
3 # Out: 8
4 def apply_to_list(some_list, f):
5     return [f(x) for x in some_list]
6
7 ints = [4, 0, 1, 5, 6]
8 apply_to_list(ints, lambda x: x * 2)
9 # Out: [8, 0, 2, 10, 12]
10 strings = ['foo', 'card', 'bar', 'aaaa', 'abab']
11 strings.sort(key=lambda x: len(set(list(x))))
12 strings
13 # Out: ['aaaa', 'foo', 'abab', 'bar', 'card']

```

偏函数用于对函数进行局部套用，可以理解为封装。

```
1 # functools.partial(func, *args, **keywords)
2 def add(*args, **kwargs):
3     for n in args:
4         print(n)
5     print("-"*10)
6     for k, v in kwargs.items():
7         print('%s=%s' % (k, v))
8
9 add(1, 2, 3, k1=10, k2=20)
10 '''
11 Out:
12 1
13 2
14 3
15 -----
16 k1=10
17 k2=20
18 '''
19 from functools import partial
20 add_partial = partial(add, 10, k1=10, k2=20)
21 add_partial(1, 2, 3, k3=20)
22 '''
23 Out:
24 10
25 1
26 2
27 3
28 -----
29 k1=10
30 k2=20
31 k3=20
32 '''
33 add_partial(1, k1=0)
34 '''
35 Out:
36 10
37 1
```

```
38  -----
39  k1=0
40  k2=20
41  '''
```

Python的生成器(generator)在数据科学领域有着广泛的应用，占用计算机资源较小。生成器也可以进行迭代，但与普通的顺序容器不同的是，它不会把所有数据存入内存中，而是只能被读取一次，实时生成数据。

生成器可以通过()构建，也可以使用关键字yield。每次生成器返回一个值时就会被冻结，直到下次调用再返回一个值。

```
1  gen = (x for x in range(5))
2  print(gen)
3  # Out: <generator object <genexpr> at
    0x0000023E97215830>
4  for item in gen:
5      print(item)
6  '''
7  Out:
8  0
9  1
10 2
11 3
12 4
13 '''
14 def createGenerator() :
15     mylist = range(3)
16     for i in mylist :
17         yield i*i
18
19 mygenerator = createGenerator()
20 print(mygenerator)
21 # Out: <generator object createGenerator at
    0x0000023E97215938>
22 for res in mygenerator:
```

```

23     print(res)
24     '''
25 Out:
26 0
27 1
28 4
29     '''
30 mygenerator = createGenerator()
31 print(next(mygenerator))
32 # Out: 0

```

生成器可以用来代替列表推导式(list comprehension)。

```

1 dict((x, x ** 2) for x in range(5))
2 # Out: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

```

标准库itertools提供了许多常用生成器。

```

1 import itertools as it
2 list(it.combinations(range(3), 2))
3 # Out: [(0, 1), (0, 2), (1, 2)]
4 list(it.permutations(range(3), 2))
5 # Out: [(0, 1), (0, 2), (1, 0), (1, 2), (2, 0),
6         (2, 1)]
7 import itertools
8 first_letter = lambda x: x[0]
9 allnames = ['Alan', 'Adam', 'wes', 'will',
10             'Albert', 'Steven']
11 for letter, names in itertools.groupby(allnames,
12                                         first_letter):
13     print(letter, list(names)) # names is a
14                                 generator
15     '''
16 Out:
17 A ['Alan', 'Adam']
18 w ['wes', 'will']
19 A ['Albert']

```

```
16 s ['Steven']
17 '''
18 list(it.product(range(2), ['a', 'b'], repeat =
19     1))
19 # Out: [(0, 'a'), (0, 'b'), (1, 'a'), (1, 'b')]
```

```
1 # ----- Day 5 -----
```

异常处理

可以通过try expect语句进行异常抛出。同时可以设置异常类型，还能使用as和Exception。

```
1 def attempt_float(x):
2     try:
3         return float(x)
4     except (ValueError, TypeError) as e:
5         print(e)
6         return x
7     except Exception:
8         return x
9 attempt_float('1.234')
10 # Out: 1.234
11 attempt_float('1')
12 # Out: 1.0
13 attempt_float('1.a')
14 '''
15 Out:
16 could not convert string to float: '1.a'
17 '1.a'
18 '''
```

同时还可以使用else和finally。

```
1 try:
2     f = open(path, 'w')
3 except:
4     print('Not Found')
5 else:
6     try:
7         write_to_file(f)
8     except:
9         print('Failed')
10    else:
11        print('Succeeded')
12    finally:
13        f.close()
```

还可以使用raise触发异常。

```
1 # raise [Exception [, args [, traceback]]]
2 def ThorwErr():
3     raise Exception("抛出一个异常")
4 ThorwErr()
```

捕捉到了异常，但是又想重新引发它(传递异常)，可以使用不带参数的raise语句。

```

1 class MuffledCalculator:
2     muffled = False
3     def calc(self, expr):
4         try:
5             return eval(expr)
6         except ZeroDivisionError:
7             if self.muffled:
8                 print('Division by zero is
illegal')
9             else:
10                raise
11 a = MuffledCalculator()
12 # a.muffled = True
13 a.calc('2/0')

```

也可以自定义异常类型，对Exception类进行继承即可。

```

1 class ShortInputException(Exception):
2     def __init__(self, length, atleast):
3         self.length = length
4         self.atleast = atleast
5 try:
6     s = input()
7     if len(s) < 3:
8         raise ShortInputException(len(s), 3)
9 except ShortInputException as e:
10     print('输入长度是%s,长度至少是%s' %(e.length,
e.atleast))
11 else:
12     print(s)
13 # In: ab
14 # Out: 输入长度是2,长度至少是3

```

文件系统

利用open()和close()打开关闭文件，可以设置编码方式，r只读，w只写，x新建只写，a添加，r+读写，b二进制模式，t文本模式。打开后tell()表示当前位置，不同编码tell()效果不同。

```
1 path = 'segismundo.txt'
2 f1 = open(path, 'rb')
3 lines = (x.rstrip() for x in open(path))
4 a = f1.read(10)
5 a.decode(encoding = 'utf-8')
6 # Out: 'Sueña e'
7 f2 = open(path)
8 lines = (x.rstrip() for x in open(path))
9 a = f2.read(10)
10 a.encode(encoding = 'gbk').decode('utf-8')
11 # Out: 'Sueña e'
12 print(f1.tell(), f2.tell())
13 # Out: 10 11
14 f1.close()
15 f2.close()
```

也可以用with open读取文件。

```
1 with open('segismundo.txt', 'r', encoding='gbk')
  as f:
2     lines =
      [x.rstrip().encode('cp936').decode('utf-8')\
3         for x in open(path)]
4 lines[0]
5 # Out: 'Sueña el rico en su riqueza,'
```

文件读取也与系统默认编码有关，可以通过sys和locale库查看系统默认编码和Python默认编码。

```
1 import sys
2 sys.getdefaultencoding()
3 # Out: 'utf-8'
4 import locale
5 locale.getpreferredencoding()
6 # Out: 'cp936'
7 # cp936 = gbk
```

seek()和read()分别用来移动到指定位置和读取字符。慎用seek(), 因为seek()转移到某个多字节字符中间时进行read()会发生UnicodeDecodeError。

```
1 f = open(path, 'r')
2 f.seek(3)
3 # Out: 3
4 f.read(1).encode('gbk').decode('utf-8')
5 # Out: 'ñ'
```

文件的常用方法还有readlines(), write(), writelines(), flush(), closed。

```
1 # ----- Day 6 -----
```

NumPy库

N维数组

numpy库全称Numerical Python, 专用于科学计算, 由于底层C语言的实现, 速度比纯python快几十倍。rand()返回[0, 1)上的均匀分布, randn()返回标准正态分布。

```
1 import numpy as np
2 data = np.random.randn(2, 3)
3 data
4 '''
5 Out:
6 array([[ -0.97835473,  0.41601724, -0.75555622],
7        [ 1.90504659,  0.22870654, -0.45414597]])
8 '''
```

可以直接对数组进行数学运算。shape()和dtype()可以查看数组信息。

```
1 data * 10
2 '''
3 Out:
4 array([[ -9.7835,  4.1602, -7.5556],
5        [19.0505,  2.2871, -4.5415]])
6 '''
7 data + data
8 '''
9 Out:
10 array([[ -1.9567,  0.832 , -1.5111],
11        [ 3.8101,  0.4574, -0.9083]])
12 '''
13 data.shape
14 # Out: (2, 3)
15 data.dtype
16 # Out: dtype('float64')
```

array()方法可以将传入参数转化为数组，而如果参数是数组那么会产生一个副本，而使用asarray()会直接返回传入的数组。

用ndim返回第一维的维数。

```
1 data1 = [[1, 2, 3], [2, 4, 6]]
2 arr1 = np.array(data1)
3 arr1.ndim
4 # Out: 2
```

zeros(), ones(), zeros_like(), ones_like(), full(), full_like()用来创建新数组。

最好不要使用没有初始化的empty()。

arange()创建一个一维顺序数组。

eye()和identity()用来创建单位矩阵。

默认数据类型均为float64。

通过设定dtype参数改变数据类型。

```
1 arr2 = np.identity(3)
2 arr2
3 '''
4 Out:
5 array([[1., 0., 0.],
6        [0., 1., 0.],
7        [0., 0., 1.]])
8 '''
9 arr3 = np.full_like(arr2, 3, dtype=np.int32)
10 arr3
11 '''
12 Out:
13 array([[3, 3, 3],
14        [3, 3, 3],
15        [3, 3, 3]])
16 '''
```

numpy支持的数据类型有：

int8, uint8, int16, uint16, int64, uint64;

float16, float32, float64, float128;

complex64, complex128, complex256;

bool, object, string_, unicode_.

缩写为i1, u1, i2, u2, i4, u4, i8, u8;

f2, f4 or f, f8 or d, f16 or g;

c8, c16, c21;

?, O, S, U.

对于非数学类型最好使用pandas。

astype()可以进行类型转换，调用时总会创建一个新的array。

```
1 arr4 = arr3.astype('f8')
2 arr4
3 '''
4 Out:
5 array([[3., 3., 3.],
6         [3., 3., 3.],
7         [3., 3., 3.]])
8 '''
```

除了array相互加减乘之外，numpy还支持常数乘除法和指数运算。

对size相同的array使用不等号还可以返回一个bool类型的数组。

```

1 arr5 = np.random.randn(2, 3)
2 arr6 = 1 / arr5
3 arr6 < arr5
4 '''
5 Out:
6 array([[False, False, False],
7        [ True,  True, False]])
8 '''

```

和list类似，array可以使用[begin1: end1, begin2: end2, ...]来获得子数组。注意获得子数组并未新建，利用copy()来获得副本。

```

1 arr0 = np.array([[1, 2, 3],
2                  [4, 5, 6],
3                  [7, 8, 9]])
4 arr7 = arr0[0]
5 arr7
6 # Out: array([1, 2, 3])
7 arr7[:] = 0
8 arr0
9 '''
10 Out:
11 array([[0, 0, 0],
12        [4, 5, 6],
13        [7, 8, 9]])
14 '''
15 arr8 = arr0[1:3, :2].copy()
16 arr8
17 '''
18 Out:
19 array([[4, 5],
20        [7, 8]])
21 '''

```

```

1 # ----- Day 7 -----

```

除了基本索引之外，还可以使用布尔索引，但要保证维数相同，Python并不会报错。也可以两者结合使用。

```
1 import numpy as np
2 names = np.array(['A', 'B', 'A', 'C'])
3 data = np.eye(4, dtype='i4')
4 names == 'A'
5 # Out: array([ True, False,  True, False])
6 data[names == 'A']
7 '''
8 Out:
9 array([[1, 0, 0, 0],
10        [0, 0, 1, 0]])
11 '''
12 data[names == 'A', :2]
13 '''
14 Out:
15 array([[1, 0],
16        [0, 0]])
17 '''
18 data[~((names == 'A') | (names == 'C')), 2:]
19 # Out: array([[1, 0, 0]])
```

也可以利用逻辑表达式来改变array的内容。

```
1 data[data > 0] = -1
2 data
3 '''
4 Out:
5 array([[ -1,  0,  0,  0],
6        [ 0, -1,  0,  0],
7        [ 0,  0, -1,  0],
8        [ 0,  0,  0, -1]])
9 '''
10 data[names != 'A'] = 1
11 data
12 '''
```

```
13 Out:
14 array([[ -1,  0,  0,  0],
15         [ 1,  1,  1,  1],
16         [ 0,  0, -1,  0],
17         [ 1,  1,  1,  1]])
18 '''
```

花式索引(fancy index)是一种智能索引。通过传递索引数组来一次获得多个元素。

```
1 arr = np.zeros((8, 4))
2 for i in range(8):
3     arr[i] = i
4 arr[[4, 3, -1]]
5 ''
6 Out:
7 array([[4., 4., 4., 4.],
8        [3., 3., 3., 3.],
9        [7., 7., 7., 7.]])
10 ''
11 arr = np.arange(20).reshape(5, 4) + 1
12 arr
13 ''
14 Out:
15 array([[ 1,  2,  3,  4],
16        [ 5,  6,  7,  8],
17        [ 9, 10, 11, 12],
18        [13, 14, 15, 16],
19        [17, 18, 19, 20]])
20 ''
21 arr[[1, 2, -1], [3, 0, -1]]
22 # Out: array([ 8,  9, 20])
```

也可以多重嵌套fancy index。


```

1 arr[[1, 2]]
2 '''
3 Out:
4 array([[ 5,  6,  7,  8],
5         [ 9, 10, 11, 12]])
6 '''
7 arr[[1, 2][:, [1, 0]]
8 '''
9 Out:
10 array([[ 6,  5],
11         [10,  9]])
12 '''

```

可用dot()来计算array的点积。T获得转置，也可以使用transpose()，高维建议使用后者，前者相当于整个array倒置。swapaxes()则能交换两个维度。

```

1 arr = np.arange(24).reshape((2, 4, 3)) + 1
2 arr
3 '''
4 Out:
5 array([[[ 1,  2,  3],
6          [ 4,  5,  6],
7          [ 7,  8,  9],
8          [10, 11, 12]],
9
10        [[13, 14, 15],
11         [16, 17, 18],
12         [19, 20, 21],
13         [22, 23, 24]]])
14 '''
15 arr.T
16 '''
17 Out:
18 array([[[ 1, 13],
19          [ 4, 16],

```

```
20         [ 7, 19],
21         [10, 22]],
22
23         [[ 2, 14],
24          [ 5, 17],
25          [ 8, 20],
26          [11, 23]],
27
28         [[ 3, 15],
29          [ 6, 18],
30          [ 9, 21],
31          [12, 24]]])
32     '''
33     arr.transpose((1, 0, 2))
34     '''
35     Out:
36     array([[[ 1,  2,  3],
37             [13, 14, 15]],
38
39            [[ 4,  5,  6],
40             [16, 17, 18]],
41
42            [[ 7,  8,  9],
43             [19, 20, 21]],
44
45            [[10, 11, 12],
46             [22, 23, 24]]])
47     '''
48     arr.swapaxes(0, 1)
49     '''
50     Out:
51     array([[[ 1,  2,  3],
52             [13, 14, 15]],
53
54            [[ 4,  5,  6],
55             [16, 17, 18]],
56
```

```
57         [[ 7,  8,  9],
58          [19, 20, 21]],
59
60         [[10, 11, 12],
61          [22, 23, 24]]])
62     '''
```

```
1 | # ----- Day 8 -----
```

通用函数

array有许多通用函数都可以使用，比如sqrt, exp等。常用的函数有：

abs, fabs;

sqrt, square, exp, expm1, log, log10, log2, log1p;

其中expm1相当于 $\exp(x)-1$ ，与log1p相当于 $\ln(x+1)$ 互逆，常用来数据预处理，比如求RMSLE(均方根对数误差)。

sign, ceil, floor, rint, modf;

分别是符号函数，和四个取整函数(返回float)，其中modf会返回两个数组。

```

1 arr = np.random.randn(3) * 5
2 arr
3 # Out: array([1.40907511, 4.3517101 ,
4           2.37764427])
5 remained, whole_part = np.modf(arr)
6 remained
7 # Out: array([0.40907511, 0.3517101 ,
8           0.37764427])
9 whole_part
10 # Out: array([1., 4., 2.])
11 np.ceil(whole_part)
12 whole_part
13 # Out: array([1., 4., 2.])

```

isnan, isfinite, isinf;

注意nan不能使用==来判断，而且会被同时当作极大和极小。

```

1 np.nan == np.nan
2 # Out: False
3 np.nan is np.nan
4 # Out: True
5 import math
6 math.nan is np.nan
7 # Out: False
8 np.isnan(math.nan)
9 # Out: True

```

cos, cosh, sin, sinh, tan, tanh, arccos, arccosh, arcsin, arcsinh, arctan, arctanh;

logical_not(等价于~array);

还有一些带有两个参数的通用函数：

add(), subtract(), multiply(), divide(), floor_divide(), power();

这些函数可以接受单个值也可以接受array。

```

1 arrd = np.divide(arr, 2)
2 arrd
3 # Out: array([0.70453755, 2.17585505,
  1.18882213])
4 arrd = np.divide(arr, whole_part)
5 arrd
6 # Out: array([1.40907511, 1.08792752,
  1.18882213])
7 arrp = np.power(arr, 2)
8 arrp
9 # Out: array([ 1.98549266, 18.93738078,
  5.65319227])
10 arrp = np.power(arr, remained)
11 arrp
12 # Out: array([1.15060233, 1.67734792,
  1.38691458])

```

maximum, fmax, minimum, fmin;

其中fmax和fmin无视NaN。

```

1 arr0 = np.array([np.nan, 1, 3, 1])
2 arr1 = np.array([2, 1, 1, 1])
3 print(np.maximum(arr0, arr1))
4 # Out: [nan  1.  3.  1.]
5 print(np.fmax(arr0, arr1))
6 # Out: [2.  1.  3.  1.]

```

copysign();

greater(), greater_equal(), less(), less_equal(), equal(),
not_equal();

等价于不等号判断，返回boolean数组。

logical_and(), logical_or(), logical_xor();

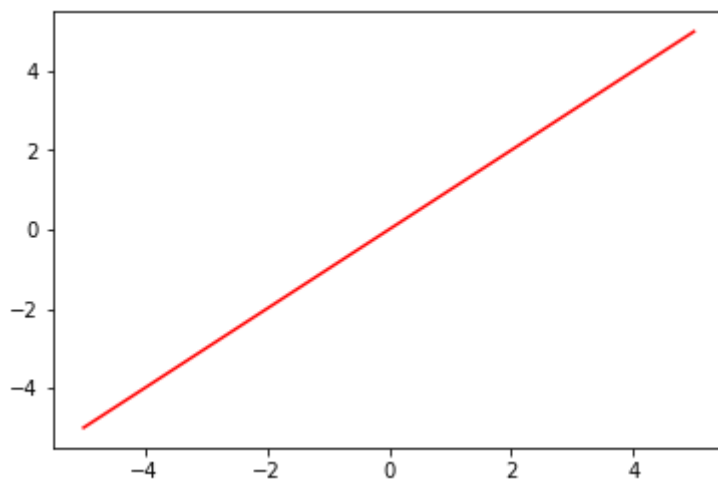
等价于&, |, ^。

面向数组编程

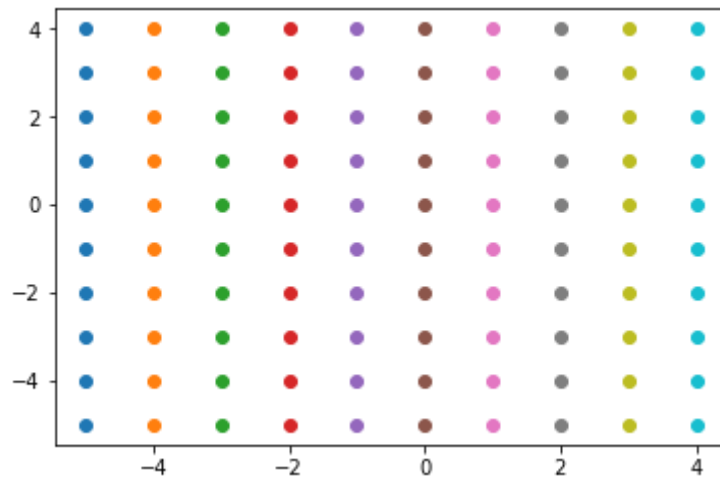
NumPy提供了许多面向数组的函数和方法，能更快速地对array进行操作。

比如当我们需要网格点时，可以利用meshgrid()对x, y坐标数组做笛卡儿积。

```
1 points = np.arange(-5, 5, 1)
2 plt.plot(points, points, 'r')
3 plt.show()
```



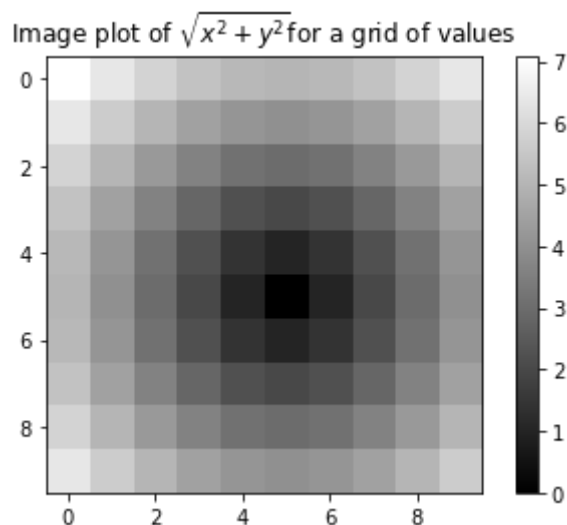
```
1 xs, ys = np.meshgrid(points, points)
2 plt.plot(xs, ys, 'o')
3 plt.show()
```



```

1 z = np.sqrt(xs**2 + ys**2)
2 plt.imshow(z, cmap=plt.cm.gray)
3 plt.colorbar()
4 plt.title('Image plot of  $\sqrt{x^2 + y^2}$  for a grid of values')
5
6 plt.show()

```



where()是一个三元函数，用来进行类似if else的操作，需要传入一个boolean数组和两个其他数组。也可以传入值。

```

1 xarr = [1, 2, 3]
2 yarr = [-1, -2, -3]
3 cond = [True, False, True]
4 result = np.where(cond, xarr, yarr)
5 result
6 # Out: array([ 1, -2,  3])
7 arr = np.random.randn(3, 3)

```

```

8 np.where(arr > 0, 1, arr)
9 '''
10 Out:
11 array([[ 1.          ,  1.          , -0.49359797],
12        [ 1.          , -1.88837102, -1.03168442],
13        [-0.04999441, -1.3569411 ,  1.          ]])
14 '''

```

NumPy还自带了很多数学函数。比如mean()和sum(), 可以使用axis来控制计算的维度, axis=0即表示将除了0号维度之外全部相同的元素进行计算, 从而消去0号维度。

```

1 arr.mean()
2 # Out: -0.04499350890122711
3 np.sum(arr)
4 # Out: -0.404941580111044
5 arr.sum(axis=0)
6 # Out: array([ 2.13961385, -1.55606646,
7               -0.98848896])
8 arr = np.ones((2, 3, 4))
9 arr.sum(axis=0)
10 '''
11 Out:
12 array([[2., 2., 2., 2.],
13        [2., 2., 2., 2.],
14        [2., 2., 2., 2.]])
15 '''

```

还有累加和累乘, cumsum(), cumprod(), 也可以用axis控制维度。如果不控制纬度则会顺序遍历返回一个一维数组。


```

1 arr.cumsum(axis = 1)
2 '''
3 Out:
4 array([[1., 1., 1., 1.],
5        [2., 2., 2., 2.],
6        [3., 3., 3., 3.]],
7
8        [[1., 1., 1., 1.],
9         [2., 2., 2., 2.],
10        [3., 3., 3., 3.]])
11 '''

```

除此以外还有var(), std(), argmin(), argmax()等函数。其中后两者返回最值第一次出现的位置。

```

1 # ----- Day 10 -----

```

boolean数组也可以执行sum(), 同时还有any(), all()来进行整体与或。这也能用在其他array中。

array还可以执行sort(), 利用axis参数将给定维度坐标不同, 其余坐标相同的元素排序。

```

1 arr = np.random.randn(3, 3, 3) * 5
2 arr = np.ceil(arr)
3 arr
4 '''
5 Out:
6 array([[[ -3.,  -0.,  -0.],
7         [  5.,   8.,  -3.],
8         [  1.,   1.,   5.]],
9
10        [[  5.,   4.,  -3.],
11         [-1.,   2.,   1.],
12         [-2.,  -6.,  -8.]],
13
14        [[ -1.,   6.,   6.],

```

```

15         [ 3., -4.,  6.],
16         [ 3., -11., -0.]])
17     '''
18     arr.sort(axis=1)
19     arr
20     '''
21     Out:
22     array([[[-3., -0., -3.],
23             [ 1.,  1., -0.],
24             [ 5.,  8.,  5.]],
25
26           [[-2., -6., -8.],
27            [-1.,  2., -3.],
28            [ 5.,  4.,  1.]],
29
30           [[-1., -11., -0.],
31            [ 3., -4.,  6.],
32            [ 3.,  6.,  6.]])
33     '''

```

利用sort()可以很方便地计算q分位数。

```

1 arr = np.random.randn(1000)
2 arr.sort()
3 arr[int(0.05 * len(arr))]
4 # Out: -1.5488900316485399

```

另外还有一些常用函数：

unique(x), intersect1d(x, y), union1d(x, y), in1d(x, y), setdiff1d(x, y), setxor1d(x, y)。

其中intersect1d等返回的都是一维值数组。in1d()返回一个一维的boolean数组。

```

1 arr1 = np.array([[3, 4, 5],
2                  [4, 2, 3],

```

```

3         [5, 3, 5]])
4 arr2 = np.array([[1, 2, 5],
5                 [3, 4, 3],
6                 [5, 3, 5]])
7 np.unique(arr1)
8 # Out: array([0, 2, 3, 4, 5])
9 np.union1d(arr1, arr2)
10 # Out: array([0, 1, 2, 3, 4, 5])
11 np.in1d(arr1, arr2)
12 # Out: array([ True,  True,  True,  True,  True,
13              True,  True,  True, False])
14 np.setdiff1d(arr1, arr2)
15 # Out: array([0])

```

文件读写

NumPy拥有自己的文件读写函数。

通过save(), load(), savez(), savez_compressed()可以进行基本的读写。其中savez类似于dict, 而compressed进行了压缩。

```

1 arr = np.arange(5)
2 np.save('some_array', arr)
3 np.load('some_array.npy')
4 np.savez('array_archive.npz', a=arr, b=arr)
5 arch = np.load('array_archive.npz')
6 arch['b']
7 # Out: array([0, 1, 2, 3, 4, 5])
8 np.savez_compressed('arrays_compressed.npz',
9                    a=arr, b=arr)

```

```

1 # ----- Day 11 -----

```

线性代数

在NumPy中，*并不能进行矩阵乘法，而是单纯地将各个位置上的元素相乘。进行点乘需要使用dot()。

```
1 x = np.array([[2, 3],
2               [0, 1]])
3 y = np.array([[0, 1],
4               [1, 0]])
5 x * y
6 '''
7 Out:
8 array([[0, 3],
9        [0, 0]])
10 '''
11 x.dot(y)
12 '''
13 Out:
14 array([[3, 2],
15        [1, 0]])
16 '''
```

另外，在点乘中一个二维矩阵和一个一维数组相乘，后者会被转化成向量，并返回一个一维数组。这个操作也可以用运算符@实现。

```
1 x.dot(np.array([1, 2]))
2 # Out: array([8, 2])
3 x @ np.array([2, 1])
4 # Out: array([7, 1])
```

numpy.linalg拥有许多常用函数，比如inv(), qr()用来获得逆矩阵和QR分解。

```
1 from numpy.linalg import inv, qr
2 x_ = inv(x)
3 x_
```

```

4  '''
5  Out:
6  array([[ 0.5, -1.5],
7         [ 0. ,  1. ]])
8  '''
9  mat = np.random.randn(3, 3)
10 q, r = qr(mat)
11 r
12 '''
13 Out:
14 array([[ 0.79869179,  1.18877944,  0.19382584],
15        [ 0.          , -2.24303773,  0.08872422],
16        [ 0.          ,  0.          ,  0.86608074]])
17 '''

```

trace()获得矩阵的迹，而diag()对于二维矩阵可以获取对角线元素，对于一维数组则会创建对应的对角阵。

```

1  x.trace()
2  # Out: 3
3  np.diag(x)
4  # Out: array([2, 1])
5  np.diag(np.diag(x))
6  '''
7  Out:
8  array([[2, 0],
9         [0, 1]])
10 '''

```

linalg.det()用来计算array的行列式，但要求最后两个维度大小相同。

linalg.eig()用来计算矩阵的特征值和特征向量。

```

1 from numpy.linalg import eig
2 w, v = eig(np.diag((1, 2, 3)))
3 w
4 # Out: array([1., 2., 3.])
5 v
6 '''
7 Out:
8 array([[1., 0., 0.],
9         [0., 1., 0.],
10        [0., 0., 1.]])
11 '''

```

`linalg.pinv()`用来计算Moore-Penrose广义逆矩阵。

`linalg.svd()`对矩阵进行奇异值分解，返回三个分解矩阵。

`linalg.solve()`解线性方程组。

`linalg.lstsq()`求最小二乘解，返回解向量，残差和，秩，奇异值。

```

1 x = np.array([[1, 2, 1],
2               [1, 0, 0],
3               [0, 4, 0]])
4 pinv(x)
5 '''
6 Out:
7 array([[ 2.06415723e-16,  1.00000000e+00,
8         -2.86157572e-16],
9         [ 1.18467981e-16, -1.07459905e-16,
10          2.50000000e-01],
11         [ 1.00000000e+00, -1.00000000e+00,
12          -5.00000000e-01]])
13 '''
14 svd(x)
15 '''
16 Out:
17 (array([[-0.4856289 , -0.70136375, -0.52177912],
18         [-0.02497309, -0.58551396,  0.81027757],
19         [ 0.87184678,  0.42329091,  0.29744358]]),
20  array([0.52177912, 0.70136375, 0.4856289 ]),
21  array([2, 1, 0]))
22 '''

```

```

16         [-0.87380828,  0.40652464,
17         0.26682728]]),
18         array([4.52173541, 1.48251813, 0.59669834]),
19         array([[ -0.11292169, -0.98778246, -0.10739879],
20               [-0.86803506,  0.15067004, -0.4730895 ],
21               [ 0.48349129,  0.03980385,
22                -0.87444373]]))
23 '''
24 solve(x, np.array([1, 1, 1]))
25 # Out: array([ 1.  ,  0.25, -0.5 ])
26 x = np.array([[1, 2],
27               [1, 0],
28               [0, 4]])
29 lstsq(x, np.array([1, 1, 1]), rcond=None)
30 '''
31 Out:
32 (array([0.77777778, 0.22222222]),
33  array([0.11111111]),
34  2,
35  array([4.49661478, 1.33433712]))
36 '''

```

```

1 # ----- Day 12 -----

```

伪随机数的产生

NumPy有许多的随机数生成器，比如可以用`normal()`来生成正态分布。

```

1 sample = np.random.normal(size=(3, 3))

```

通过`normal()`生成的一系列随机数速度很快，比逐个生成要快很多。

```

1 from random import normalvariate
2 N = 1000000
3 %timeit samples = [normalvariate(0, 1) for _ in
4   range(N)]
5 %timeit np.random.normal(size=N)
6 '''
7 Out:
8 882 ms ± 24.3 ms per loop (mean ± std. dev. of 7
9   runs, 1 loop each)
10 38.1 ms ± 449 µs per loop (mean ± std. dev. of 7
11   runs, 10 loops each)
12 '''

```

另外可以通过seed()改变随机种子，RandomState()来创建局部随机生成器。

```

1 np.random.seed(123)
2 sample = np.random.RandomState(123)
3 sample.randn(5)
4 # Out: array([-1.0856306 ,  0.99734545,
5   0.2829785 , -1.50629471, -0.57860025])

```

permutation()和shuffle()都可以用来打乱array，多维数组则只打乱第一维。

但前者还能传入int，相当于随机打乱arange()。而且前者返回一个打乱后的副本，后者会直接改变array并且返回None。

```

1 sample = np.array([[1, 2, 3],
2                    [4, 5, 6],
3                    [7, 8, 9]])
4 np.random.permutation(sample)
5 '''
6 Out:
7 array([[4, 5, 6],
8        [7, 8, 9],
9        [1, 2, 3]])

```



```

10 '''
11 sample
12 '''
13 Out:
14 array([[1, 2, 3],
15        [4, 5, 6],
16        [7, 8, 9]])
17 '''
18 np.random.shuffle(sample)
19 sample
20 '''
21 Out:
22 array([[4, 5, 6],
23        [1, 2, 3],
24        [7, 8, 9]])
25 '''
26 sample = np.random.permutation(5)
27 sample
28 # Out: array([0, 4, 2, 3, 1])

```

rand()返回 $[0, 1)$ 上的均匀随机变量。而randint()得到 $[0, low)$ 或者 $[low, high)$ 的随机整数。randn返回标准正态随机变量。

还有一些常见的分布函数如下：

```

1 # uniform(low=0.0, high=1.0, size=None)
2 # 均匀分布
3 # binomial(n, p, size=None)
4 # 二项分布
5 # negative_binomial(n, p, size=None)
6 # 负二项分布
7 # multinomial(n, pvals, size=None)
8 # 多项分布
9 # normal(loc=0.0, scale=1.0, size=None)
10 # 正态分布
11 # standard_normal(size=None)
12 # 标准正态分布

```

```
13 # lognormal(mean=0.0, sigma=1.0, size=None)
14 # 对数正态分布
15 # multivariate_normal(mean, cov[, size,
    check_valid, tol])
16 # 多元正态分布
17 # wald(mean, scale, size=None)
18 # 逆高斯分布
19 # poisson(lam=1.0, size=None)
20 # 泊松分布
21 # exponential(scale=1.0, size=None)
22 # 指数分布
23 # standard_exponential(size=None)
24 # 标准指数分布
25 # beta(a, b, size=None)
26 #  $\beta$ 分布
27 # gamma(shape, scale=1.0, size=None)
28 #  $\Gamma$ 分布
29 # standard_gamma(shape, size=None)
30 # 标准 $\Gamma$ 分布
31 # chisquare(df, size=None)
32 # 卡方分布
33 # standard_t(df, size=None)
34 # 标准t分布
35 # f(dfnum, dfden, size=None)
36 # F分布
37 # noncentral_chisquare(df, nonc, size=None)
38 # 非中心卡方分布
39 # noncentral_f(dfnum, dfden, nonc, size=None)
40 # 非中心F分布
41 # geometric(p, size=None)
42 # 几何分布
43 # hypergeometric(ngood, nbad, nsample, size=None)
44 # 超几何分布
45 # pareto(a, size=None)
46 # 帕累托分布
47 # dirichlet(alpha, size=None)
48 # 狄利克雷分布
```

```
49 # gumbel(loc=0.0, scale=1.0, size=None)
50 # Gumbel分布(极值分布)
51 # laplace(loc=0.0, scale=1.0, size=None)
52 # 拉普拉斯分布
53 # logistic(loc=0.0, scale=1.0, size=None)
54 # Logistic分布
55 # rayleigh(scale=1.0, size=None)
56 # 瑞利分布
57 # standard_cauchy(size=None)
58 # 标准柯西分布
59 # triangular(left, mode, right, size=None)
60 # 三角分布
61 # weibull(a, size=None)
62 # 韦布尔分布
63 # zipf(a, size=None)
64 #  $\zeta$ 分布
```

```
1 | # ----- Day 13 -----
```

样例：随机游动

Pandas库
