



Multi-dimensional View of Python

Python面面观

Department of Computer Science and Technology
Department of University Basic Computer Teaching

用Python玩转数据

1

条件

if 语句

语 法

```
if expression :  
    expr_true_suite
```

expression

条件表达式：
• 比较运算符
• 成员运算符
• 逻辑运算符

expr_true_suite

- expression 条件为 True时执行的代码块
- 代码块必须缩进（通常为4个空格）

File

```
# Filename: ifpro.py  
sd1 = 3  
sd2 = 3  
if sd1 == sd2:  
    print("the square's area is", sd1*sd2)
```

else 语句

语法

```
if expression:  
    expr_true_suite  
else:  
    expr_false_suite
```

expr_false_suite

- expression 条件为 False 时执行的代码块
- 代码块必须缩进
- else语句不缩进

F ile

```
# Filename: elsepro.py  
sd1 = int(input('the first side: '))  
sd2 = int(input('the second side: '))  
if sd1 == sd2:  
    print("the square's area is", sd1*sd2)  
else:  
    print("the rectangle's area is", sd1*sd2)
```

I nput and O utput

```
the first side: 4  
the second side: 4  
the square's area is 16
```

elif 语句

语 法

```
if expression :  
    expr_true_suite  
elif expression2:  
    expr2_true_suite  
    :  
    :  
elif expressionN :  
    exprN_true_suite  
else:  
    none_of_the_above_suite
```

expr2_true_suite

- expression2为True时执行的代码块
- expressionN 为 True 时执行的代码块

exprN_true_suite

- none_of_the_above_suite 是以上所有条件都不满足时执行的代码块

else

elif 语句

F ile

```
# Filename: elifpro.py
k = input('input the index of shape: ')
if k == '1':
    print('circle')
elif k == '2':
    print('oval')
elif k == '3':
    print('rectangle')
elif k == '4':
    print('triangle')
else:
    print('you input the invalid number')
```

I nput and
O utput

input the index of shape: 3
rectangle

I nput and
O utput

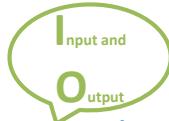
input the index of shape: 8
you input the invalid number

条件嵌套

- 同等缩进为同一条件结构



```
input the index of shape: 3  
the first side: 3  
the second side: 4  
the rectangle's area is 12
```



```
input the index of shape: 2  
oval
```

F ile

```
# Filename: ifnestpro.py  
k = input('input the index of shape: ')  
if k == '1':  
    print('circle')  
elif k == '2':  
    print('oval')  
elif k == '3':  
    sd1 = int(input('the first side: '))  
    sd2 = int(input('the second side : '))  
    if sd1 == sd2:  
        print("the square's area is", sd1*sd2)  
    else:  
        print("the rectangle's area is", sd1*sd2)  
elif k == '4':  
    print('triangle')  
else:  
    print('you input the invalid number')
```

猜数字游戏

- 程序随机产生一个0~300间的整数，玩家竞猜，系统给出“猜中”、“太大了”或“太小了”的提示。

F ile

```
# Filename: guessnum1.py
from random import randint

x = randint(0, 300)
digit = int(input('Please input a number between 0~300: '))
if digit == x:
    print('Bingo!')
elif digit > x:
    print('Too large, please try again.')
else:
    print('Too small, please try again.')
```

用Python玩转数据

RANGE函数

2
3

range()

语 法

`range (start, end, step=1)`

`range (start, end)`

`range (end)`

- 产生一系列整数，返回一个range对象

S_{ource}

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

start

- 起始值（包含）

end

- 终值（不包含）

step

- 步长（不能为0）

range (start, end, step=1)

- 不包含end的值

range (start, end)

- 缺省step值为1

range (end)

- 缺省了start值为0，step为1

range()

异同	range()	xrange()
语法	基本一致	
返回	列表	生成器（类似）
生成	真实列表	用多少生成多少

Python 2.x

异同	range()
语法	与Python 2.x中类似
返回	生成器（类似）
生成	用多少生成多少

Python 3.x

用Python玩转数据

3

循环

while 循环

语法

`while expression:`

`suite_to_repeat`

`expression`

- 条件表达式
- 当 expression 值为 True 时执行 suite_to_repeat 代码块



```
>>> sumA = 0
>>> j = 1
>>> while j < 10:
        sumA += j
        j += 1
>>> sumA
45
>>> j
10
```

for 循环 (一)

语法

```
for iter_var in iterable_object:  
    suite_to_repeat
```

可以明确循环的次数

- 遍历一个数据集内的成员
- 在列表解析中使用
- 生成器表达式中使用

iterable_object

- String
- List
- Tuple
- Dictionary
- File

for 循环 (二)

- 字符串就是一个 *iterable_object*
- range()返回的也是 *iterable_object*

S_{ource}

```
>>> s = 'python'  
>>> for c in s:  
    print(c)
```

p
y
t
h
o
n

```
>>> for i in range(3,11,2):  
    print(i, end = ' ')
```

3 5 7 9

猜数字游戏

- 程序随机产生一个0~300间的整数，玩家竞猜，允许猜多次，系统给出“猜中”、“太大了”或“太小了”的提示。

F ile

```
# Filename: guessnum2.py
from random import randint

x = randint(0, 300)
for count in range(5):
    digit = int(input('Please input a number between 0~300: '))
    if digit == x:
        print('Bingo!')
    elif digit > x:
        print('Too large, please try again.')
    else:
        print('Too small, please try again.') 
```

用Python玩转数据

循环中的 BREAK,CONTINUE和ELSE

4

break 语句

- break语句终止当前循环，转而执行循环之后的语句

F
ile

```
# Filename: breakpro.py
sumA = 0
i = 1
while True:
    sumA += i
    i += 1
    if sumA > 10:
        break
print('i={},sum={}'.format(i, sumA))
```

Output:

i=6, sum=15

while 循环和break

- 输出2-100之间的素数

Output:

```
2 3 5 7 11 13 17 19  
23 29 31 37 41 43  
47 53 59 61 67 71  
73 79 83 89 97
```

F_{ile}

```
# Filename: prime.py  
from math import sqrt  
j = 2  
while j <= 100:  
    i = 2  
    k= sqrt(j)  
    while i <= k:  
        if j%i == 0: break  
        i = i+1  
    if i > k:  
        print(j, end = ' ')  
    j += 1
```

for 循环和break

- 输出2-100之间的素数

Output:

```
2 3 5 7 11 13 17 19  
23 29 31 37 41 43  
47 53 59 61 67 71  
73 79 83 89 97
```

F ile

```
# Filename: prime.py  
from math import sqrt  
for i in range(2,101):      flag = 1  
    k = int(sqrt(i))  
    for j in range(2,k+1):  
        if i%j == 0:  
            flag = 0  
            break  
    if( flag ):  
        print(i, end = ' ')
```

continue 语句

- 在while和for循环中，continue语句的作用：
 - 停止当前循环，重新进入循环
 - while循环则判断循环条件是否满足
 - for循环则判断迭代是否已经结束

continue语句

- 循环中的break :

F ile

```
# Filename: breakpro.py
sumA = 0
i = 1
while i <= 5:
    sumA += i
    if i == 3:
        break
    print('i={},sum={}'.format(i,sumA))
    i += 1
```

- 循环中的continue :

F ile

```
# Filename: continuepro.py
sumA = 0
i = 1
while i <= 5:
    sumA += i
    i += 1
    if i == 3:
        continue
    print('i={},sum={}'.format(i,sumA))
```

猜数字游戏（想停就停，非固定次数）

- 程序随机产生一个0~300间的整数，玩家竞猜，允许玩家自己控制游戏次数，如果猜中系统给出提示并退出程序，如果猜错给出“太大了”或“太小了”的提示，如果不继续玩可以退出并说再见。

F ile

```
# Filename: guessnum3.py
from random import randint
x = randint(0, 300)
go = 'y'
while (go == 'y'):
    digit = int(input('Please input a number between 0~300: '))
    if digit == x:
        print('Bingo!')
        break
    elif digit > x:
        print('Too large, please try again.')
    else:
        print('Too small, please try again.')
    print('Input y if you want to continue.')
    go = input()
    print(go)
else:
    print('Goodbye!')
```

循环中的else语句

- 循环中的else：
 - 如果循环代码从
break处终止，跳出
循环
 - 正常结束循环，则
执行else中代码



```
# Filename: prime.py
from math import sqrt
num = int(input('Please enter a number: '))
j = 2
while j <= int(sqrt(num)):
    if num % j == 0:
        print('{:d} is not a prime.'.format(num))
        break
    j += 1
else:
    print('{:d} is a prime.'.format(num))
```

用Python玩转数据

5

自定义函数

函数

内置
函数

函数调用之前必须先定义

自定
义
函
数

自定义函数的创建

语 法

```
def function_name([arguments]):  
    "optional documentation string"  
  
    function_suite
```

S_{ource}

```
>>> def addMe2Me(x):  
        'apply operation + to argument'  
        return (x+x)
```

自定义函数的调用

- 函数名加上函数运算符，一对小括号
 - 括号之间是所有可选的参数
 - 即使没有参数，小括号也不能省略

S_{ource}

```
>>> addMe2Me()
```

Traceback (most recent call last):

File "<pyshell#6>", line 1, in <module>
 addMe2Me()
TypeError: addMe2Me() takes exactly 1 argument (0 given)

S_{ource}

```
>>> addMe2Me(3.7)
```

7.4

```
>>> addMe2Me(5)
```

10

```
>>> addMe2Me('Python')  
'PythonPython'
```

自定义函数

- 输出1-100之间的素数

Output:

```
2 3 5 7 11 13 17 19  
23 29 31 37 41 43  
47 53 59 61 67 71  
73 79 83 89 97
```

F ile

```
# Filename: prime.py  
from math import sqrt  
def isprime(x):  
    if x == 1:  
        return False  
    k = int(sqrt(x))  
    for j in range(2,k+1):  
        if x%j == 0:  
            return False  
    return True  
for i in range(2,101):  
    if isprime(i):  
        print( i, end = ' ')
```

默认参数（一）

- 函数的参数可以有一个默认值，如果提供有默认值，在函数定义中，默认参数以赋值语句的形式提供

S_{ource}

```
>>> def f(x = True):  
    '''whether x is a correct word or not'''  
    if x:  
        print('x is a correct word')  
    print('OK')  
>>> f()  
x is a correct word  
OK  
>>> f(False)  
OK
```

默认参数 (二)

- 默认参数的值可以改变

S_{ource}

```
>>> def f(x, y = True):  
    "x and y both correct words or not"  
    if y:  
        print(x, 'and y both correct')  
        print(x, 'is OK')  
>>> f(68)  
68 and y both correct  
68 is OK  
>>> f(68, False)  
68 is OK
```

默认参数（三）

- 默认参数一般需要放置在参数列表的最后

S_{ource}

```
def f(y = True, x):  
    '''x and y both correct words or not'''  
    if y:  
        print(x, 'and y both correct ')  
    print(x, 'is OK')
```

SyntaxError: non-default argument follows default argument

关键字参数

- 关键字参数是让调用者通过使用参数名区分参数。允许改变参数列表中的参数顺序

S
ource

```
>>> def f(x, y):  
    "x and y both correct words or not"  
    if y:  
        print(x, 'and y both correct ')  
        print(x, 'is OK')  
>>> f(68, False)  
68 is OK  
>>> f(y = False, x = 68)  
68 is OK  
>>> f(y = False, 68)  
SyntaxError: non-keyword arg after keyword arg  
>>> f(x = 68, False)  
SyntaxError: non-keyword arg after keyword arg
```

传递函数

- 函数可以像参数一样传递给另外一个函数

S_{ource}

```
>>> def addMe2Me(x):  
        return x+x  
>>> def self(f, y):  
        print(f(y))  
>>> self(addMe2Me, 2.2)  
4.4
```

lambda函数

- 匿名函数

S_{ource}

```
>>> def addMe2Me(x):
    'apply operation + to argument'
    return (x + x)
>>> addMe2Me(5)
10
```

S_{ource}

```
>>> r = lambda x : x + x
>>> r(5)
10
```

lambda函数

```
def my_add(x, y) : return x + y
```

```
lambda      x, y : x + y
```

```
my_add = lambda x, y : x + y
```

```
>>> my_add(3, 5)
```

8

6

用Python玩转数据

递归

递归



循环

递归

生成斐波那契数列的方法

递归是最能表现计算思维的算法之一

循环和递归

- 递归必须要有边界条件，即停止递归的条件
 - $n == 0$ or $n == 1$
- 递归的代码更简洁，更符合自然逻辑，更容易理解

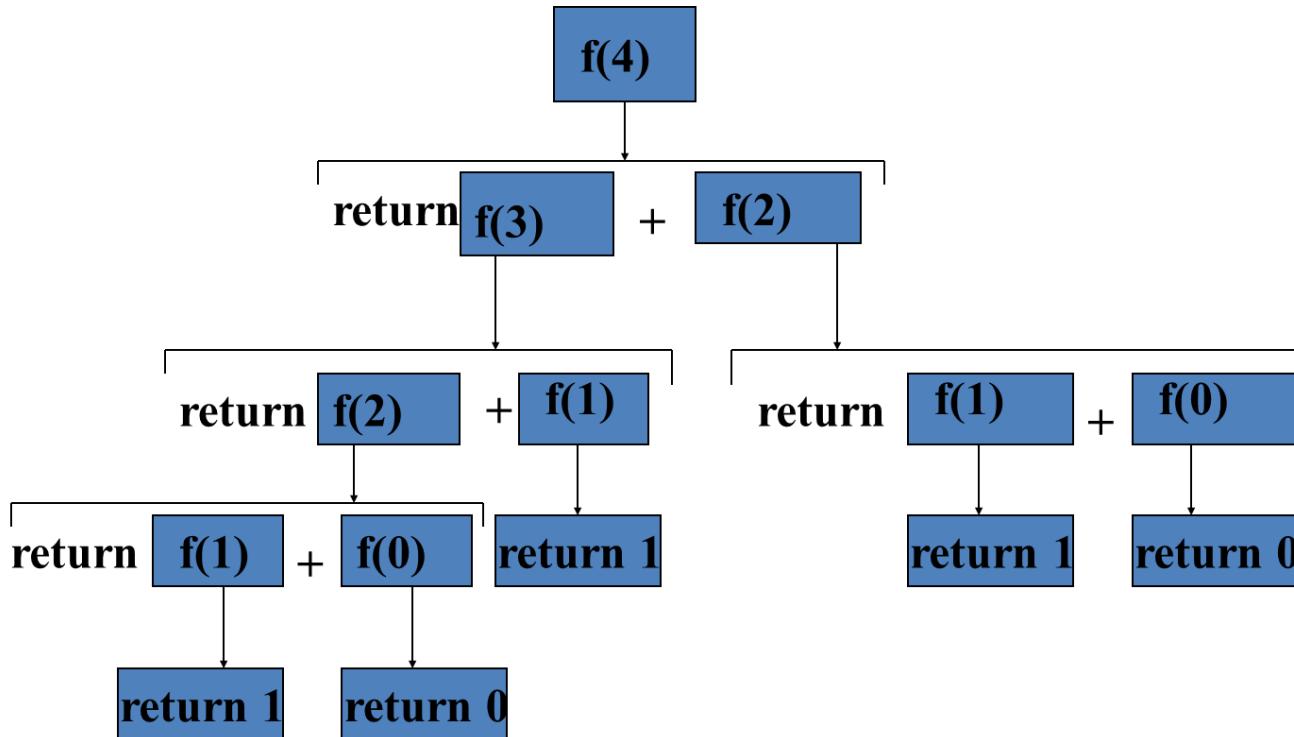
S_{ource}

```
# the nth Fibonacci number
def fib(n):
    a, b = 0, 1
    count = 1
    while count < n:
        a, b = b, a+b
        count = count + 1
    print(a)
```

S_{ource}

```
# the nth Fibonacci number
def fib(n):
    if n == 0 or n == 1:
        return n
    else:
        return fib(n - 1) + fib(n - 2)
```

递归



递归

- 递归的执行



汉诺塔

- 汉诺塔游戏

三个塔座A、B、C上各有一根针，通过C把64个盘子从A针移动到B针上，移动时必须遵循下列规则：

- (1) 圆盘可以插入在A、B或C塔座的针上
- (2) 每次只能移动一个圆盘
- (3) 任何时刻都不能将一个较大的圆盘压在较小的圆盘之上



```
# Filename: Hanoi.py
def hanoi(a,b,c,n):
    if n==1:
        print(a,'->',c)
    else:
        hanoi(a,c,b,n-1)
        print(a,'->', c)
        hanoi(b,a,c,n-1)

hanoi('a','b','c',4)
```

Output:

```
a -> b
a -> c
b -> c
a -> b
c -> a
c -> b
a -> b
a -> c
b -> c
b -> a
c -> a
b -> c
a -> b
a -> c
b -> c
```

用Python玩转数据

7

变量作用域

变量作用域

- 全局变量
- 局部变量

Filename: global.py

```
global_str = 'hello'  
def foo():  
    local_str = 'world'  
    return global_str + local_str
```

File

Source

```
>>> foo()  
'helloworld'
```

同名变量

- 全局变量和局部变量用同一个名字

F ile

```
# Filename: samename.py
a = 3
def f( ):
    a = 5
    print(a ** 2)
```

改变全局变量的值

- 方法是否可行？

F_{ile}

```
# Filename: scopeofvar.py
def f(x):
    print(a)
    a = 5
    print(a + x)

a = 3
f(8)
```

UnboundLocalError: local variable 'a'
referenced before assignment

global语句

- global语句强调全局变量

F_{ile}

```
# Filename: scopeofvar.py
def f(x):
    global a
    print(a)
    a = 5
    print(a + x)
```

```
a = 3
f(8)
print(a)
```

Output:

```
3
13
5
```



Where are data from? How to represent data?

数据 获取与表示

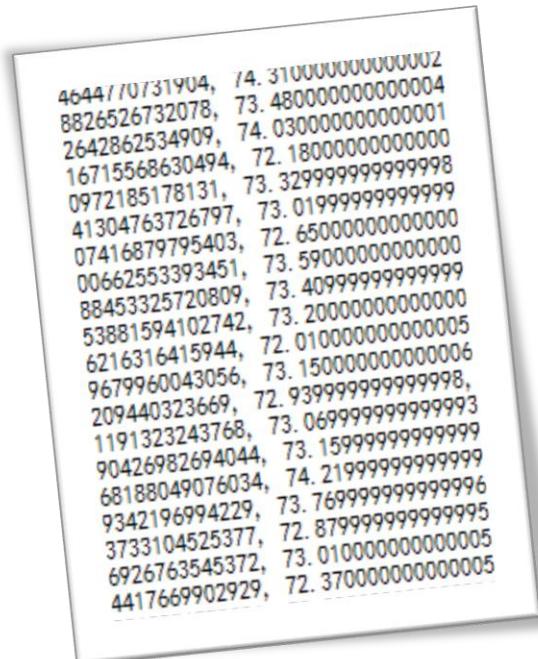
Department of Computer Science and Technology
Department of University Basic Computer Teaching

用Python玩转数据

1

本地数据获取

用Python获取数据



本地数据如何获取？

文件的打开，读写和关闭

- 打开后才能进行读写
- 读文件
- 写文件
- 文件为什么需要关闭？

文件的打开

S_{ource}

```
>>> f1 = open('d:\\infile.txt')
>>> f2 = open(r'd:\\outfile.txt', 'w')
>>> f3 = open('record.dat', 'wb', 0)
```

file_obj = open(filename, mode='r', buffering=-1, ...)

- mode为可选参数，默认值为r
- buffering也为可选参数，默认值为-1（0代表不缓冲，1或大于1的值表示缓冲一行或指定缓冲区大小）

open()函数-mode

Mode	Function
r	以读模式打开
w	以写模式打开 (清空原内容)
a	以追加模式打开 (从EOF开始 , 必要时创建新文件)
r+	以读写模式打开
w+	以读写模式打开 (清空原内容)
a+	以读和追加模式打开
rb	以二进制读模式打开
wb	以二进制写模式打开 (参见w)
ab	以二进制追加模式打开 (参见a)
rb+	以二进制读写模式打开 (参见r+)
wb+	以二进制读写模式打开 (参见w+)
ab+	以二进制读写模式打开 (参见a+)

文件相关函数

返回值

- `open()`函数返回一个文件（file）对象
- 文件对象可迭代
- 有关闭和读写文件相关的函数/方法
 - `f.read()`, `f.write()`, `f.readline()`, `f.readlines()`, `f.writelines()`
 - `f.close()`
 - `f.seek()`

写文件-f.write()

- **file_obj.write(str)**

- 将一个字符串写入文件

S
ource

```
>>> f = open('firstpro.txt', 'w')
>>> f.write('Hello, World!')
>>> f.close()
```

firstpro.txt :
Hello, World!

S
ource

```
>>> with open('firstpro.txt', 'w') as f:
    f.write('Hello, World!')
```

读文件-f.read()

- **file_obj.read(size)**
 - 从文件中至多读出size字节数据，返回一个字符串
- **file_obj.read()**
 - 读文件直到文件结束，返回一个字符串

S
ource

```
>>> with open('firstpro.txt') as f:  
    p1 = f.read(5)  
    p2 = f.read()  
    print(p1,p2)
```

Output:
Hello, World!

其他读写函数

F
ile

```
# Filename: companies_a.py
with open('companies.txt') as f:
    cNames = f.readlines()
    print(cNames)
```

- file_obj.readlines()
- file_obj.readline()
- file_obj.writelines()

Output:

```
['GOOGLE Inc.\n', 'Microsoft Corporation\n', 'Apple Inc.\n',
'Facebook, Inc.']
```

文件读写例子



将文件companies.txt 的字符串前加上序号1、2、3、...后写到另一个文件scompanies.txt中。

F
ile

```
# Filename: revcopy.py
with open('companies.txt') as f1:
    cNames = f1.readlines()
    for i in range(0, len(cNames)):
        cNames[i] = str(i+1) + ' ' + cNames[i]
with open('scompanies.txt', 'w') as f2:
    f2.writelines(cNames)
```

Output:

```
1 GOOGLE Inc.
2 Microsoft Corporation
3 Apple Inc.
4 Facebook, Inc.
```

其他文件相关函数



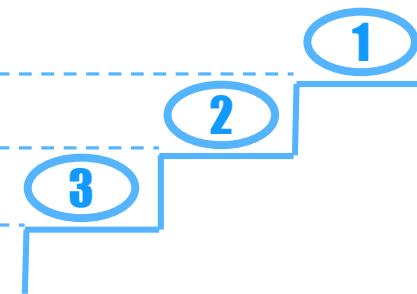
```
# Filename: companies_b.py
s = 'Tencent Technology Company Limited'
with open('companies.txt', 'a+') as f:
    f.writelines('\n')
    f.writelines(s)
    cNames = f.readlines()
    print(cNames)
```

- **file_obj.seek(offset , whence=0)**
 - 在文件中移动文件指针，从 *whence* (0表示文件头部，1表示当前位置，2表示文件尾部) 偏移*offset*个字节
 - *whence*参数可选，默认值为0

标准文件

- 当程序启动后，以下三种标准文件有效

- `stdin` 标准输入
- `stdout` 标准输出
- `stderr` 标准错误



S
ource

```
>>> newcName = input('Enter the name of new company: ')
Enter the name of new company: Alibiabia
>>> print(newcName)
Alibiabia
```

```
>>> import sys
>>> sys.stdout.write('hello')
```

用Python玩转数据

2
3

网络数据获取

用Python获取数据

Symbol	Company Name	Price
OK	Nokia Corporation	8.4
APL	Apple Inc.	104.7
AC	Bank of America Cor...	16.6
	AT&T, Inc.	33.5
BR	Petr	12.5
QQ	PowerShares QQQ	97.9
B	Facebook, Inc.	80.0
IIO	iBio, Inc.	1.4
ELP	Yelp, Inc.	58.5
IFN	Infinera Corporation	13.0
SFT	Microsoft Corporation	44.7
O	The Coca-Cola Comp...	41.2
PF	Pfizer Inc	28.8

网络数据如何获取（爬取）？

抓取网页，解析网页内容

- 抓取
 - urllib内建模块
 - urllib.request
 - Requests第三方库
 - Scrapy框架
- 解析
 - BeautifulSoup库
 - re模块

第三方
API抓取
+解析

Requests库

- Requests库是更简单、方便和人性化的Python HTTP第三方库
- Requests官网：<http://www.python-requests.org/>
- 基本方法 `requests.get()` 请求获取指定URL位置的资源，对应HTTP协议的GET方法

遵循网站爬虫协议 robots. txt

S
ource

```
>>> import requests
>>> r = requests.get('https://book.douban.com/subject/1084336/comments/')
>>> r.status_code
200
>>> print(r.text)
```

道指成分股数据

<http://finance.yahoo.com/q/cp?s=%5EDJI+Component>

Symbol	Company Name	Last Price	Change	% Change	Volume
UTX	United Technologies Corporation	111.80	-0.04	-0.04%	2,249,660
VZ	Verizon Communications Inc.	49.68	0.04	0.08%	9,310,665
CAT	Caterpillar Inc.	92.15	-0.08	-0.09%	3,634,569
DIS	The Walt Disney Company	112.14	-0.10	-0.09%	4,707,297
CVX	Chevron Corporation	107.99	0.12	0.11%	6,339,900
JPM	JPMorgan Chase & Co.	87.29	-0.10	-0.11%	16,093,237
KO	The Coca-Cola Company	42.12	-0.05	-0.12%	13,804,411

Company	Price	Change	% Change	Volume	YTD change
MMM 3M	194.86	+0.38	+0.19%	49,315	+0.12%
AXP American Express	76.94	+0.56	+0.73%	99,320	+3.86%
AAPL Apple	153.78	+1.34	+0.81%	619,445	+12.77%
BA Boeing	178.93	+1.53	+0.86%	29,107	+14.93%
CAT Caterpillar	102.28	+2.06	+2.06%	132,054	+10.26%
CVX Chevron	105.94	+0.77	+0.73%	75,547	+0.99%
CSCO Cisco	31.16	-0.21	-0.69%	617,510	+3.13%
KO Coca-Cola	43.84	+0.04	+0.08%	124,282	+5.73%
DIS Disney	107.28	+0.80	+0.75%	138,082	+2.94%
DD E.I. du Pont de Nemours and Co	77.96	+0.92	+1.19%	48,776	+0.21%
XOM Exxon Mobil	81.91	+0.16	+0.20%	219,457	+0.25%
GE General Electric	28.06	+0.58	+2.11%	900,772	+1.20%
GS Goldman Sachs	216.84	+1.68	+0.78%	64,862	+0.44%
HD Home Depot	156.14	+0.44	+0.28%	68,635	+16.45%
IBM IBM	151.75	+0.97	+0.64%	56,785	+0.58%

<http://money.cnn.com/data/dow30/>

利用Requests库获取道指成分股数据

```

<tr>
    <td class="wsod_firstCol"><a href="/quote/quote.html?symb=INTC" class="ws
    <td class="wsod_aRight"><span stream="last_167459" class="wsod_stream">35
    <td class="wsod_aRight"><span stream="change_167459" class="wsod_stream">
    <td class="wsod_aRight"><span stream="changePct_167459" class="wsod_strea
    <td class="wsod_aRight">17,171,872</td>
    <td class="wsod_aRight"><span class="negData">-3.39%</span></td>
</tr>

<tr>
    <td class="wsod_firstCol"><a href="/quote/quote.html?symb=JNJ" class="ws
    <td class="wsod_aRight"><span stream="last_174239" class="wsod_stream">12
    <td class="wsod_aRight"><span stream="change_174239" class="wsod_stream">
    <td class="wsod_aRight"><span stream="changePct_174239" class="wsod_strea
    <td class="wsod_aRight">6,571,254</td>
    <td class="wsod_aRight"><span class="posData">+10.21%</span></td>
</tr>

```



```

# Filename: dji.py
import requests
re = requests.get('http://money.cnn.com/data/dow30/') # the url may change
print(re.text)

```

- 包含多个字符串

- 'AXP', 'American Express Company', '77.77'
- 'BA', 'The Boeing Company', '177.83'
- 'CAT', 'Caterpillar Inc.', '96.39'
- ...

网页数据解析

- BeautifulSoup是一个可以从HTML或XML文件中提取数据的Python库
- 官方网站：

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

```
soup.find_all('p', 'comment-content')
```

<p class="comment-content">不知道第几次重读。
每过一段时间再读，都有新的收获。心变得很柔软，脑里的
迷雾被驱散。更多的关注他人，关心这个世界，自私是
多么无趣的事情啊。我想，写一本能温暖人心，帮助困难
的人们的书，比世界上很多事情都有意义。</p>



- re正则表达式模块进行各类正则表达式处理

- 参考网站：

<https://docs.python.org/3.5/library/re.html>

```
'<span class="user-stars allstar(.*)? rating"'
```

```
<span class="user-stars allstar50 rating" title="力荐"></span>
```

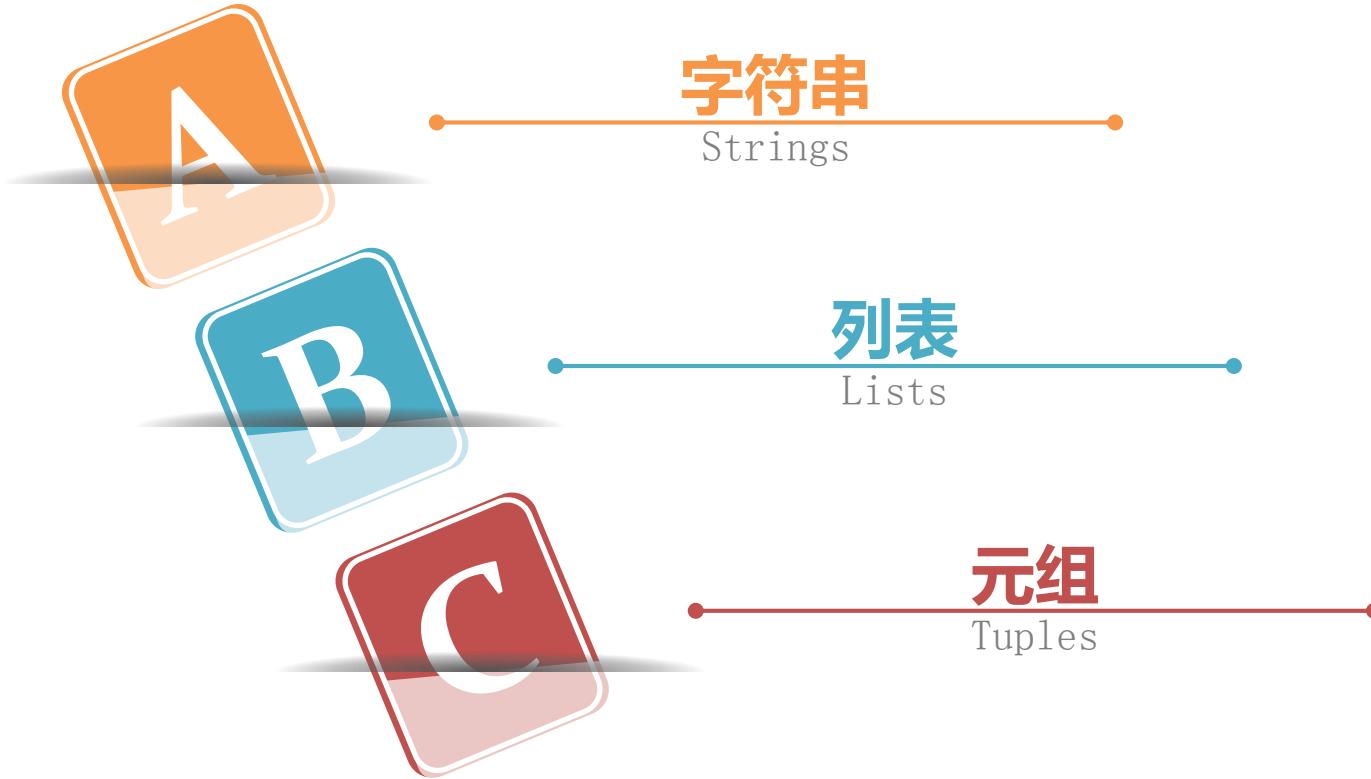
用Python玩转数据

3

序列

序列

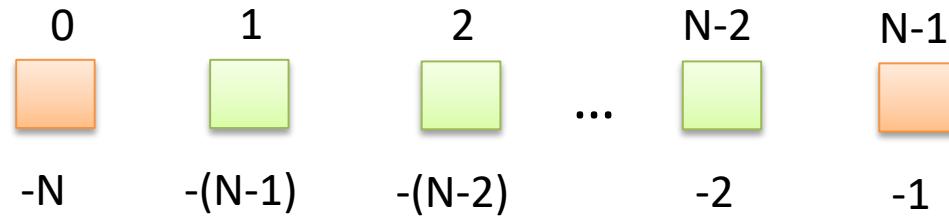
- aStr = 'Hello, World!'
- aList = [2, 3, 5, 7, 11]
- aTuple = ('Sunday', 'happy')
- pList = [('AXP', 'American Express Company', '78.51'),
('BA', 'The Boeing Company', '184.76'),
('CAT', 'Caterpillar Inc.', '96.39'),
('CSCO', 'Cisco Systems, Inc.', '33.71'),
('CVX', 'Chevron Corporation', '106.09')]



Python中的序列

	0	1	2	3	4	5	6
week	'Monday'	'Tuesday'	'Wednesday'	'Thursday'	'Friday'	'Saturday'	'Sunday'
	-7	-6	-5	-4	-3	-2	-1

序列



访问模式

- 元素从0开始通过下标偏移量访问
- 一次可访问一个或多个元素

序列相关操作

标准
类型
运算符

值比较
对象身份比较
布尔运算

序列
类型
运算符

获取
重复
连接
判断

内建
函数

序列类型转换内建函数
序列类型可用内建函数

标准类型运算符

S
ource

```
>>> 'apple' < 'banana'
```

```
True
```

```
>>> [1,3,5] != [2,4,6]
```

```
True
```

```
>>> aTuple = ('BA', 'The Boeing Company', '184.76')
```

```
>>> bTuple = aTuple
```

```
>>> bTuple is not aTuple
```

```
False
```

```
>>> ('86.40' < '122.64') and ('apple' > 'banana')
```

```
False
```

标准类型运算符

值比较

<	>
<=	>=
==	!=

对象身份比较

is
is not

布尔运算

not
and
or

序列类型运算符

S
ource

```
>>> week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
>>> print(week[1], week[-2], '\n', week[1:4], '\n', week[:6], '\n', week[::-1])
Tuesday Saturday
['Tuesday', 'Wednesday', 'Thursday']
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
['Sunday', 'Saturday', 'Friday', 'Thursday', 'Wednesday', 'Tuesday', 'Monday']
>>> 'apple' * 3
'appleappleapple'
>>> 'pine' + 'apple'
'pineapple'
>>> 'BA' in ('BA', 'The Boeing Company', '184.76')
True
```

序列类型运算符

x in s

x not in s

s + t

s * n, n * s

s[i]

s[i:j]

s[i:j:k]

序列类型转换内建函数

`list()`

`str()`

`tuple()`

S
ource

```
>>> list('Hello, World!')
['H', 'e', 'l', 'l', 'o', ',', ' ', 'W', 'o', 'r', 'l', 'd', '!']
>>> tuple("Hello, World!")
('H', 'e', 'l', 'l', 'o', ',', ' ', 'W', 'o', 'r', 'l', 'd', '!')
```

序列类型其他常用内建函数

<code>enumerate()</code>	<code>reversed()</code>
<code>len()</code>	<code>sorted()</code>
<code>max()</code>	<code>sum()</code>
<code>min()</code>	<code>zip()</code>



```
>>> aStr = 'Hello, World!'
>>> len(aStr)
13
>>> sorted(aStr)
[' ', '!', ',', 'H', 'W', 'd', 'e', 'I', 'l', 'o', 'o', 'r']
```

用Python玩转数据

4

字符串

字符串的不同表示形式

```
If = [('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')]
```

S_{ource}

```
>>> aStr = 'The Boeing Company'  
>>> bStr = "The Boeing Company "  
>>> cStr = "I'm a student."  
>>> dStr = ""The Boeing  
company""
```

字符串小例子



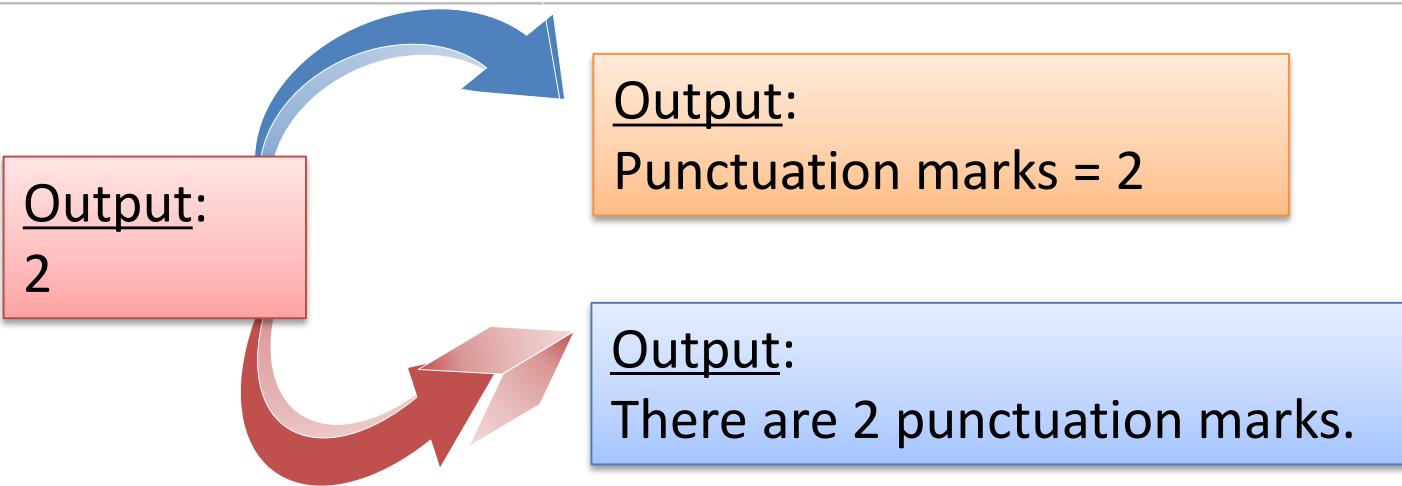
将字符串 “Hello, World!” 中的 “World” 替换成 “Python”，并计算其包含的标点符号（由逗号、句号、感叹号和问号组成）的个数。



```
# Filename: punctcount.py
aStr = "Hello, World!"
bStr = aStr[:7] + "Python!"
count = 0
for ch in bStr[:]:
    if ch in ',.!?':
        count += 1
print(count)
```

Output:
2

字符串与输出形式



```
print('There are %d punctuation marks.' % (count))
```

format_string % (arguments_to_convert)

```
print('There are {0:d} punctuation marks.'.format(count))
```

format_string.format(arguments_to_convert)

类型说明符

字符	描述
b	二进制，以2为基数输出数字
o	八进制，以8为基数输出数字
x	十六进制，以16为基数输出数字，9以上的数字用小写字母（类型符为x时用大写字母）表示
c	字符，将整数转换成对应的Unicode字符输出
d	十进制整数，以10为基数输出数字
f	浮点数，以浮点数输出数字
e	指数记法，以科学计数法输出数字，用e（类型符是E时用大写E）表示幂

其他常用格式说明符

符号	描述
+m.nf	输出带符号（若正整数输出“+”号）的数，保留n位小数，整个输出占m列（若实际宽度超过m则突破m的限制）
<	左对齐，默认用空格填充右边
0>5d	右对齐，用0填充左边，宽度为5
^	居中对齐
{}	输出一个{}

[对齐说明符][符号说明符][最小宽度说明符][.精度说明符][类型说明符]

```
>>> age, height = 21, 1.758
>>> print("Age:{0:<5d}, Height:{1:5.2f}".format(age, height))
Age:21    , Height: 1.76
```

用format函数格式化字符串例

S
ource

```
>>> cCode = ['AXP' , 'BA' , 'CAT' , 'CSCO' , 'CVX' ]
>>> cPrice = ['78.51' , '184.76' , '96.39' , '33.71' , '106.09' ]
>>> for i in range(5):
    print('{:<8d}{:8s}{:8s}'.format(i, cCode[i], cPrice[i]))
0    AXP    78.51
1    BA     184.76
2    CAT    96.39
3    CSCO   33.71
4    CVX    106.09
>>> print('I get {:d}{}'.format(32))
I get 32 {}!
```

字符串的应用



有一个字符串“acdhdca”，判断其是否是回文串。接着判断一个数字354435是否是回文数字。

F ile

```
# Filename: compare.py
sStr = "acdhdca"
if (sStr == ".join(reversed(sStr))):
    print('Yes')
else:
    print('No')
```

F ile

```
# Filename: compare.py
import operator
sStr = "acdhdca"
if operator.eq(sStr, ".join(reversed(sStr)))==0:
    print('Yes')
else:
    print('No')
```

sStr == sStr[::-1]

字符串常用方法

<code>capitalize()</code>	<code>center()</code>	<code>count()</code>	<code>encode()</code>	<code>endswith()</code>	<code>find()</code>
<code>format()</code>	<code>index()</code>	<code>isalnum()</code>	<code>isalpha()</code>	<code>isdigit()</code>	<code>islower()</code>
<code>isspace()</code>	<code>istitle()</code>	<code>isupper()</code>	<code>join()</code>	<code>ljust()</code>	<code>lower()</code>
<code>lstrip()</code>	<code>maketrans()</code>	<code>partition()</code>	<code>replace()</code>	<code>rfind()</code>	<code>rindex()</code>
<code>rjust()</code>	<code>rpartition()</code>	<code>rstrip()</code>	<code>split()</code>	<code>splitlines()</code>	<code>startswith()</code>
<code>strip()</code>	<code>swapcase()</code>	<code>title()</code>	<code>translate()</code>	<code>upper()</code>	<code>zfill()</code>

字符串的应用



有一些从网络上下载的类似如下形式的一些句子：

What do you think of this saying "No pain, No gain"?

对于句子中双引号中的内容，首先判断其是否满足标题格式，不管满足与否最终都将其转换为标题格式输出。

F
ile

```
# Filename: totitle.py
aStr = 'What do you think of this saying "No pain, No gain"?
lindex = aStr.index('"',0,len(aStr))
rindex = aStr.rindex('"',0,len(aStr))
tempStr = aStr[lindex+1:rindex]
if tempStr.istitle():
    print('It is title format.')
else:
    print('It is not title format.')
print(tempStr.title())
```

tempstr= aStr.split('"')[1]

转义字符

字符	说明
\0	空字符
\a	响铃
\b	退格
\t	横向制表符
\n	换行
\v	纵向制表符
\f	换页
\r	回车
\e	转义
\"	双引号
\'	单引号
\\"	反斜杠
\(在行尾时)	续行符

\ooo 八进制数ooo代表的字符
 \xXX 十六进制数XX代表的字符



```
>>> aStr = '\101\t\x41\n'
>>> bStr = '\141\t\x61\n'
>>> print(aStr, bStr)
A      A
a      a
```

用Python玩转数据

5

列表

列表

可扩展的
容器对象

S
ource

```
>>> aList = list('Hello.')  
>>> aList  
['H', 'e', 'l', 'l', 'o', '.']  
>>> aList = list('hello.')  
>>> aList  
['h', 'e', 'l', 'l', 'o', '.']  
>>> aList[0] = 'H'  
>>> aList  
['H', 'e', 'l', 'l', 'o', '.']
```

包含不同
类型对象

S
ource

```
>>> bList = [1, 2, 'a', 3.5]
```

列表的形式

- aList = [1, 2, 3, 4, 5]
- names = ['Zhao', 'Qian', 'Sun', 'Li']
- bList = [3, 2, 1, 'Action']
- pList = [('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')]

列表



某学校组织了一场校园歌手比赛，每个歌手的得分由10名评委和观众决定，最终得分的规则是去掉10名评委所打分数的一个最高分和一个最低分，再加上所有观众评委分数后的平均值。评委打出的10个分数为：9、9、8.5、10、7、8、8、9、8和10，观众评委打出的综合评分为9，请计算该歌手的最终得分。

F ile

```
# Filename: scoring.py
jScores = [9, 9, 8.5, 10, 7, 8, 8, 9, 8, 10]
aScore = 9
jScores.sort()
jScores.pop()
jScores.pop(0)
jScores.append(aScore)
aveScore = sum(jScores)/len(jScores)
print(aveScore)
```

[7, 8, 8, 8.5, 9, 9, 9, 10, 10]

[8, 8, 8, 8.5, 9, 9, 9, 9, 10]

[8, 8, 8, 8.5, 9, 9, 9, 9, 10, 9]

8.72222222222

列表



将工作日（['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']）和周末（['Saturday', 'Sunday']）的表示形式合并，并将它们用序号标出并分行显示。

F
ile

```
# Filename: week.py
week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
weekend = ['Saturday', 'Sunday']
week.extend(weekend)
for i,j in enumerate(week):
    print(i+1, j)
```

Output:

```
1 Monday
2 Tuesday
3 Wednesday
4 Thursday
5 Friday
6 Saturday
7 Sunday
```

列表方法

append()

copy()

count()

extend()

index()

insert()

pop()

remove()

reverse()

sort()

参数的作用：

`list.sort(key=None, reverse=False)`

S
ource

```
>>> numList = [3, 11, 5, 8, 16, 1]
>>> fruitList = ['apple', 'banana', 'pear', 'lemon', 'avocado']
>>> numList.sort(reverse = True)
>>> numList
[16, 11, 8, 5, 3, 1]
>>> fruitList.sort(key = len)
>>> fruitList
['pear', 'apple', 'lemon', 'banana', 'avocado']
```

列表解析

List comprehensions ,
list comps

动态创建列表
简单灵活有用

生成器表达式

Generator expression

```
>>> sum(x for x in range(10))
```

45

lazy evaluation

S
ource

```
>>> [x for x in range(10)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [x ** 2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> [x ** 2 for x in range(10) if x ** 2 < 50]
[0, 1, 4, 9, 16, 25, 36, 49]
>>> [(x+1, y+1) for x in range(2) for y in range(2)]
[(1, 1), (1, 2), (2, 1), (2, 2)]
```

[expression for expr in sequence1
 for expr2 in sequence2 ...
 for exprN in sequenceN
 if condition]

6

用Python玩转数据

元组

元组

- 元组的一般使用与列表类似

S_{ource}

```
>>> 2014  
2014  
>>> 2014,  
(2014,)
```

S_{ource}

```
>>> bTuple = ('Monday', 1, 2, 3)  
>>> bTuple  
(['Monday', 1], 2, 3)  
>>> bTuple[0][1]  
1  
>>> len(bTuple)  
3  
>>> bTuple[1:]  
(2, 3)
```

元组

- 列表元素可以改变
- 元组元素不可以改变

S
ource

```
>>> aList = ['AXP', 'BA', 'CAT']
>>> aTuple = ('AXP', 'BA', 'CAT')
>>> aList[1] = 'Alibiabia'
>>> print(aList)
['AXP', 'Alibiabia', 'CAT']
>>> aTuple[1]= 'Alibiabia'
>>> print(aTuple)
aTuple[1]='Alibiabia'
TypeError: 'tuple' object does not support item assignment
```

元组

- 函数的适用类型

S_{ource}

```
>>> aList = [3, 5, 2, 4]
>>> aList
[3, 5, 2, 4]
>>> sorted(aList)
[2, 3, 4, 5]
>>> aList
[3, 5, 2, 4]
>>> aList.sort()
>>> aList
[2, 3, 4, 5]
```

S_{ource}

```
>>> aTuple = (3, 5, 2, 4)
>>> sorted(aTuple)
[2, 3, 4, 5]
>>> aTuple
(3, 5, 2, 4)
```

```
>>> aTuple.sort()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'sort'
```

元组的作用

元组用在什么地方？



可变长位置参数（元组）

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)



```
>>> def foo(args1, args2 = 'World!'):
        print(args1, args2)
>>> foo('Hello,')
Hello, World!
>>> foo('Hello,', args2 = 'Python!')
Hello, Python!
>>> foo(args2 = 'Apple!', args1 = 'Hello,')
Hello, Apple!
```

```
>>> def foo(args1, *argst):
        print(args1)
        print(argst)
```

可变长位置参数（元组）



```
>>> def foo(args1, *argst):
        print(args1)
        print(argst)
>>> foo('Hello,', 'Wangdachui', 'Niuyun', 'Linling')
Hello,
('Wangdachui', 'Niuyun', 'Linling')
```

元组作为函数特殊返回类型

返回对象的个数	返回类型
0	None
1	object
>1	tuple

S
ource

```
>>> def foo():  
        return 1, 2, 3  
>>> foo()  
(1, 2, 3)
```



Powerful data structure and software ecosystem

强大的数据结构和 Python 扩展库

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。

F
ile

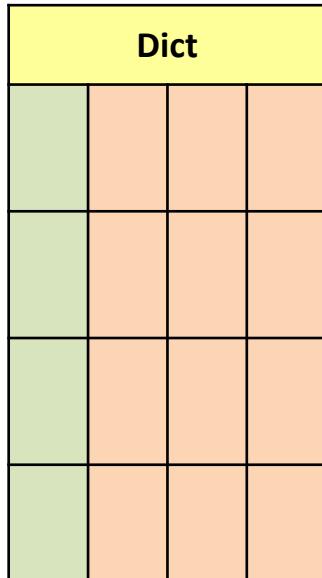
```
# Filename: info.py
names = ['Wangdachui', 'Niuyun', 'Linling', 'Tianqi']
salaries = [3000, 2000, 4500, 8000]
print(salaries[names.index('Niuyun')])
```

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'

S_{ource}

```
>>> 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**

- 直接
- 利用dict函数

cInfo['Niuyun']

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 ?

S
ource

```
>>> 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？

S
ource

```
>>> 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'}
```

```
If = [('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

F_{ile}

```
# 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

2
3

字典的使用

字典的基本操作

S_{ource}

```
>>> alInfo = {'Wangdachui': 3000, 'Niuyun':2000, 'Linling':4500, 'Tianqi':8000}
>>> alInfo['Niuyun'] → 键值查找
2000
>>> alInfo['Niuyun'] = 9999 → 更新
>>> alInfo
{'Tianqi': 8000, 'Wangdachui': 3000, 'Linling': 4500, 'Niuyun': 9999}
>>> alInfo['Fuyun'] = 1000 → 添加
>>> alInfo
{'Tianqi': 8000, 'Fuyun': 1000, 'Wangdachui': 3000, 'Linling': 4500, 'Niuyun': 9999}
>>> 'Mayun' in alInfo → 成员判断
False
>>> del alInfo['Fuyun'] → 删
>>> alInfo
{'Tianqi': 8000, 'Wangdachui': 3000, 'Linling': 4500, 'Niuyun': 9999}
```

字典的内建函数

dict()

len()

hash()

S_{ource}

```
>>> 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

S_{ource}

```
>>> 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？

S_{ource}

```
>>> 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？

S
ource

```
>>> 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？

S_{ource}

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

S_{ource}

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

字典方法

- 删除字典

S_{ource}

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

S_{ource}

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

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

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"}  
}
```

```
>>> 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)

S
ource

```
>>> 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}
```



3

集合

用Python玩转数据

集合

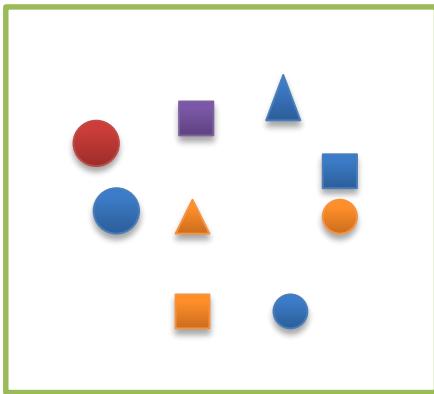


人事部门的一份工资信息表登记时由于工作人员的疏忽有部分姓名重复登记了，如何快速解决这个问题？

S_{ource}

```
>>> 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

S_{ource}

```
>>> 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

S_{ource}

```
>>> 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	in
\notin	not in
$=$	$==$
\neq	$!=$
\subset	<
\subseteq	\leq
\supset	>
\supseteq	\geq

标准类型运算符

集合关系运算

S
ource

```
>>> 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
\cap	&
\cup	
- 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()

S_{ource}

```
>>> 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()

S_{ource}

```
>>> 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()
```

4



用Python玩转数据

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



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？

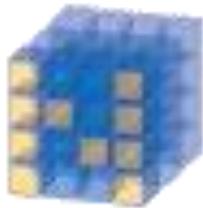
- **SciPy中的数据结构**



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

S_{ource}

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

SciPy库

特征



- Python中科学计算程序的核心包
- 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

S_{ource}

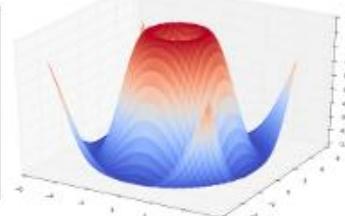
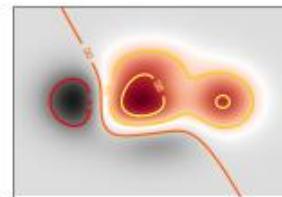
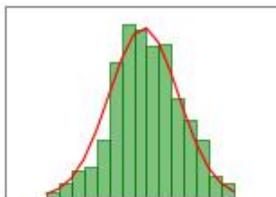
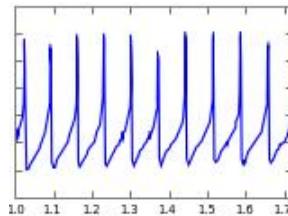
```
>>> import numpy as np  
>>> from scipy import linalg  
>>> arr = np.array([[1,2],[3,4]])  
>>> linalg.det(arr)  
-2.0
```

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

S
ource

...

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



用Python玩转数据

5

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

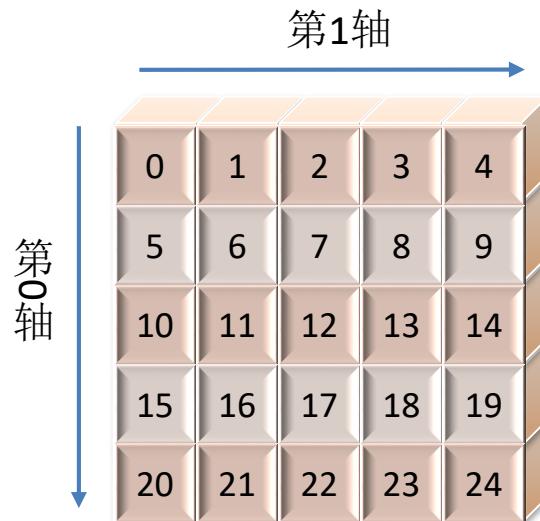
0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

- **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 (axis)，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

S
ource

```
>>> 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

S_{ource}

```
>>> 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



S
ource

```
>>> 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

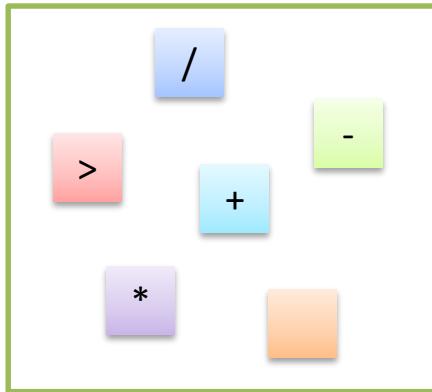
S_{ource}

```
>>> 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]])
```

S_{ource}

```
>>> 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

S
ource

```
>>> 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

S
ource

```
>>> 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

S_{ource}

```
>>> 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, ...

F
ile

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

6

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

S_{ource}

```
from pandas import Series
>>> aSer = pd.Series([1,2.0,'a'])
>>> aSer
0    1
1    2
2    a
dtype: int64
```

自定义Series的index

S_{ource}

```
>>> 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

S_{ource}

```
>>> 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

S_{ource}

```
>>> 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
 - 在算术运算中自动对齐不同索引的数据

S_{ource}

```
>>> 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name属性

- **重要功能**

- Series对象本身及其索引均有一个name属性
- Series的name属性与其他重要功能关系密切

S_{ource}

```
>>> aSer = pd.Series(data, index = sindex)
>>> aSer.name = 'cnames'
>>> aSer.index.name = 'volume'
>>> aSer
volume
      AXP    86.40
      CSCO   122.64
      BA     99.44
      AAPL    NaN
Name: cnames, dtype: object
```

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

S
ource

```
>>> data = {'name': ['Wangdachui', 'Linling', 'Niuyun'], 'pay': [4000, 5000, 6000]}\n>>> frame = pd.DataFrame(data)\n>>> frame\n      name    pay\n0  Wangdachui  4000\n1      Linling  5000\n2      Niuyun  6000
```

DataFrame的索引和值

S_{ource}

```
>>> 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

S_{ource}

```
>>> 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

S_{ource}

```
>>> 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

S
ource

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

S
ource

```
>>> 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

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

S_{ource}

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

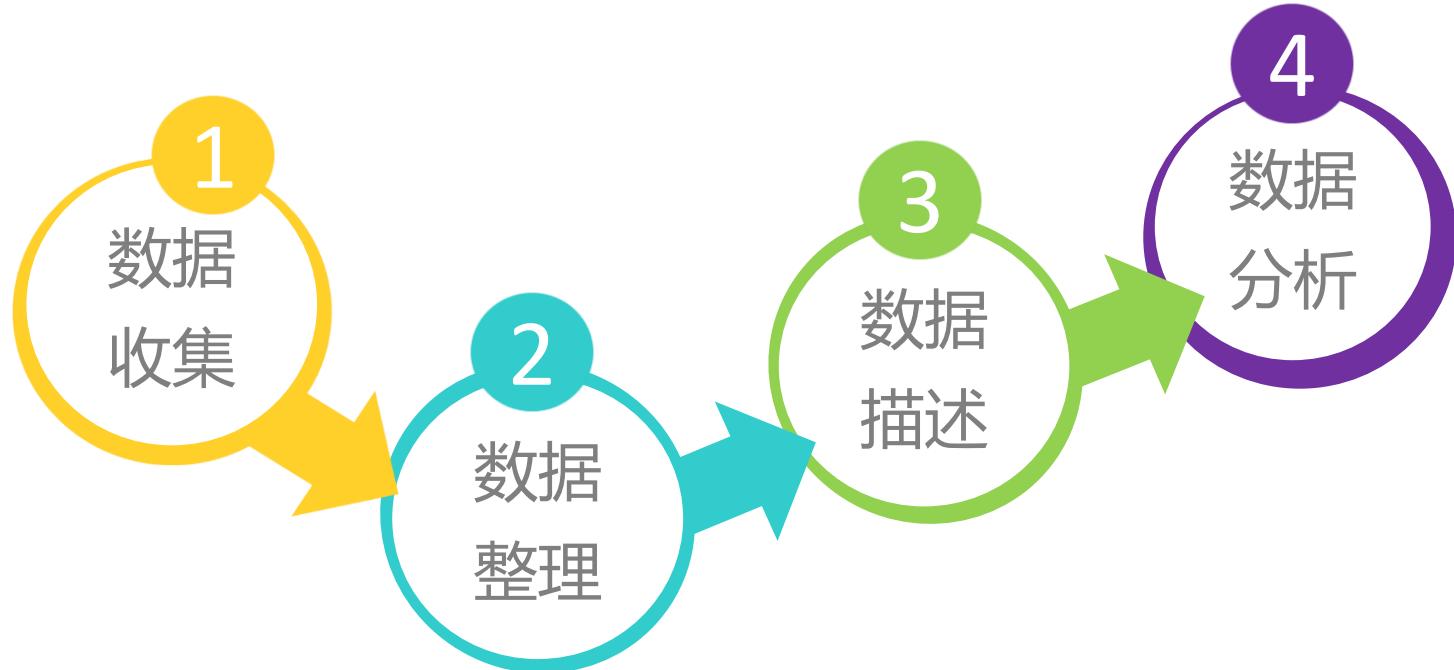


Basic data processing of Python

Python基本数据统计

Department of Computer Science and Technology
Department of University Basic Computer Teaching

简单数据处理过程

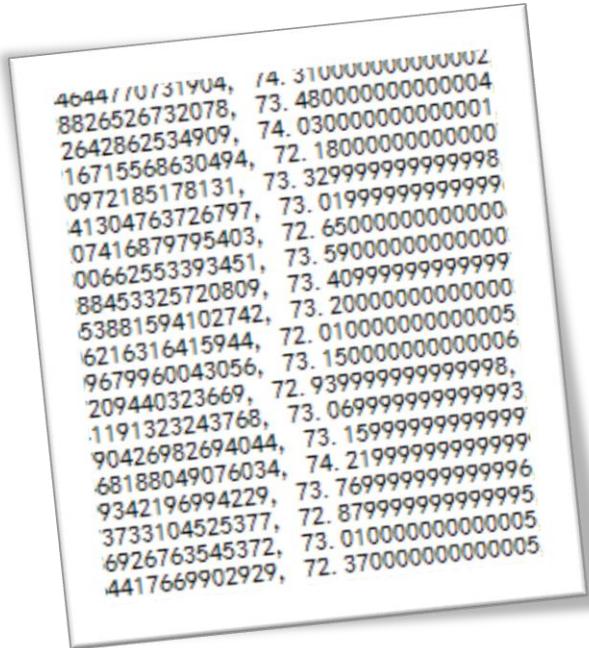


用Python玩转数据

1

便捷数据获取

用Python获取数据



本地数据如何获取？

文件的打开，读写和关闭

- 文件打开
- 读文件
- 写文件
- 文件关闭

用Python获取数据

Symbol	Company Name	Price
OK	Nokia Corporation	8.4
APL	Apple Inc.	104.7
AC	Bank of America Cor...	16.6
	AT&T, Inc.	33.5
BR	Petr	12.5
QQ	PowerShares QQQ	97.9
B	Facebook, Inc.	80.0
IIO	iBio, Inc.	1.4
ELP	Yelp, Inc.	58.5
IFN	Infinera Corporation	13.0
SFT	Microsoft Corporation	44.7
O	The Coca-Cola Comp...	41.2
PF	Pfizer Inc	28.8

网络数据如何获取（爬取）？

抓取网页，解析网页内容

- 抓取
 - urllib内建模块
 - urllib.request
 - Requests第三方库
 - Scrapy框架
- 解析
 - BeautifulSoup库
 - re模块

道指成分股数据

Dow 30 Companies - CNN Money

money.cnn.com/data/dow30/

DOW JONES INDU AVERAGE INDEX
(Dow Jones Global Indices: INDX)

20,804.84 +141.82 / +0.69% 17,063 21,169 +5.27%

Delayed Data As of May 19
Today's Change 52-Week Range Year-to-Date

Companies in the Dow Jones Industrial Average

Company	Price	Change	% Change	Volume	YTD change
MMM 3M	195.80	+0.56	+0.29%	1,894,690	+9.65%
AXP American Express	76.80	+0.42	+0.55%	3,283,915	+3.67%
AAPL Apple	153.06	+0.52	+0.34%	26,960,788	+32.15%
BA Boeing	180.76	+3.36	+1.89%	4,038,809	+16.11%
CAT Caterpillar	102.43	+2.21	+2.21%	4,636,549	+10.45%
CVX Chevron	106.52	+1.35	+1.28%	8,383,478	-9.50%
CSCO Cisco	31.21	-0.17	-0.54%	39,859,249	+3.28%
KO Coca-Cola	43.90	+0.10	+0.23%	11,796,762	+5.89%
DIS Disney	107.52	+0.84	+0.79%	6,336,651	+3.17%
DD E I du Pont de Nemours and Co	77.82	+0.78	+1.01%	2,649,385	+6.02%
XOM Exxon Mobil	81.93	+0.18	+0.22%	13,028,511	-9.23%
GE General Electric	28.05	+0.57	+2.07%	48,488,671	-11.23%
GS Goldman Sachs	215.39	+0.23	+0.11%	3,747,072	-10.05%
HD Home Depot	156.30	+0.60	+0.39%	4,781,806	+16.57%
IBM IBM	151.98	+1.20	+0.80%	5,643,264	-8.44%

CFD losses can exceed deposits. Refer to our PDS. Not directed at persons in any jurisdiction where it would be contrary to local laws or regulations. Issued by IG Markets Limited AFSL 220446.

AXP Historical Prices | YAHOO! FINANCE

<https://finance.yahoo.com/quote/AXP/history?p=AXP>

Search for news, symbols or companies

Finance Home Originals Events Personal Finance Technology Markets Industries My Screeners ...

Apr 24, 2017 80.64 80.92 80.24 80.45 80.45 3, **Financial** Consumer Good

Apr 21, 2017 79.88 80.47 79.49 79.59 79.59 5, **Healthcare** Technology

Apr 20, 2017 77.50 80.28 77.46 80.02 80.02 11, **Services**

Apr 19, 2017 76.19 76.29 75.51 75.55 75.55 5, **Utilities**

Apr 18, 2017 76.45 76.49 75.53 75.79 75.79 3, **Industrial Goods**

Apr 17, 2017 75.83 76.71 75.63 76.67 76.67 4, **Basic Materials**

Apr 13, 2017 76.33 76.81 75.70 75.80 75.80 3, **Conglomerates**

Apr 12, 2017 77.39 77.48 76.58 76.68 76.68 3, **Automate IT WITH IG MT4.**

Apr 11, 2017 77.53 77.57 76.69 77.26 77.26 4,642,300 20, **Buy**

Apr 10, 2017 77.92 78.11 77.36 77.49 77.49 2,217,100 10, **Hold**

Apr 07, 2017 77.48 78.24 77.37 77.77 77.77 2,203,000 10, **Unk**

Apr 06, 2017 77.76 78.30 77.15 77.92 77.92 2,914,800 10, **Sell**

Apr 05, 2017 78.59 79.03 77.66 77.76 77.76 2,858,400 10, **Recommendation Rating >**

Apr 04, 2017 **0.32 Dividend** 25

Apr 04, 2017 78.49 78.61 78.15 78.26 78.26 2,563,700 1, **Strong Buy**

Apr 03, 2017 79.17 79.18 77.97 78.59 78.59 3,022,700 2, **Buy**

1 Strong Buy 2 Buy 3 Hold 4 Underperform

dji

quotes

数据形式

0	1	2
0	MMM	3M
1	AXP	American Express
2	AAPL	Apple
3	BA	Boeing
4	CAT	Caterpillar
5	CVX	Chevron
6	CSCO	Cisco
7	KO	Coca-Cola
8	DIS	Disney
9	DD	E I du Pont de Nemours
10	XOM	Exxon Mobil
11	GE	General Electric
12	GS	Goldman Sachs
13	HD	Home Depot
14	IBM	IBM
15	INTC	Intel
16	JNJ	Johnson & Johnson
17	JPM	JPMorgan Chase
18	MCD	McDonald's
19	MRK	Merck
20	MSFT	Microsoft
21	NKE	Nike
22	PFE	Pfizer
23	PG	Procter & Gamble
24	TRV	Travelers Companies Inc
25	UTX	United Technologies
26	UNH	UnitedHealth
27	VZ	Verizon
28	V	Visa
29	WMT	Wal-Mart

djidf

	close	date	high	low	open	volume
0	76.8	1495200600	77.35	76.3	76.55	3278200
1	76.38	1495114200	76.85	75.97	76.27	3545700
2	76.37	1495027800	78.13	76.24	78.13	4441600
3	78.13	1494941400	78.64	77.84	78.6	2457500
4	78.33	1494855000	78.62	77.48	77.48	3327000
5	77.49	1494595800	77.81	77.22	77.7	2865800
6	77.92	1494509400	78.45	77.25	78.2	3780600
7	78.65	1494423000	78.66	78.14	78.28	2396900
8	78.44	1494336600	78.74	78.09	78.16	2570600
9	78.16	1494250200	78.74	77.95	78.5	2608600
10	78.32	1493991000	78.73	77.88	78.61	2936700
11	78.33	1493904600	79.42	77.99	79.23	3902200
12	78.83	1493818200	79.51	78.69	79.23	3800600
13	79.54	1493731800	79.66	79.15	79.15	3334900
14	79.23	1493645400	79.49	78.88	79.22	3458100
15	79.25	1493386200	80.17	79.05	79.94	5313200
16	80.33	1493299800	80.87	80.08	80.77	2922700
17	80.52	1493213400	80.92	80.15	80.62	3661600
18	80.63	1493127000	81.4	80.63	81.06	5061300
19	80.45	1493040600	80.92	80.24	80.64	3563200
20	79.59	1492781400	80.47	79.49	79.88	5837800

quotesdf

便捷网络数据获取



是否能够简单方便并且快速的方式获得财经网站上公司股票的历史数据？

Time Period: May 20, 2016 - May 20, 2017

Show: Historical Prices

Frequency: Daily

Apply

Currency in USD

Date	Open	High	Low	Close	Adj Close	Volume
2016/5/20	63.16	64.14	62.95	63.92	63.92	5278200
2016/5/23	63.86	64.1	63.56	63.59	63.59	3074100
2016/5/24	63.79	65.1	63.79	64.87	64.87	3946100
2016/5/25	65.04	65.76	65.01	65.31	65.31	5755900
2016/5/26	65.29	65.37	64.95	65.23	65.23	3593500
2016/5/27	65.39	65.7	65.33	65.52	65.52	3925700
2016/5/31	65.7	65.92	65.4	65.76	65.76	5256000

Download Data

F
ile

```
# Filename: quotes_fromcsv.py
import pandas as pd
quotesdf = pd.read_csv('axp.csv')
print(quotesdf)
```

便捷网络数据获取



图书Api V2

回Api V2 首页

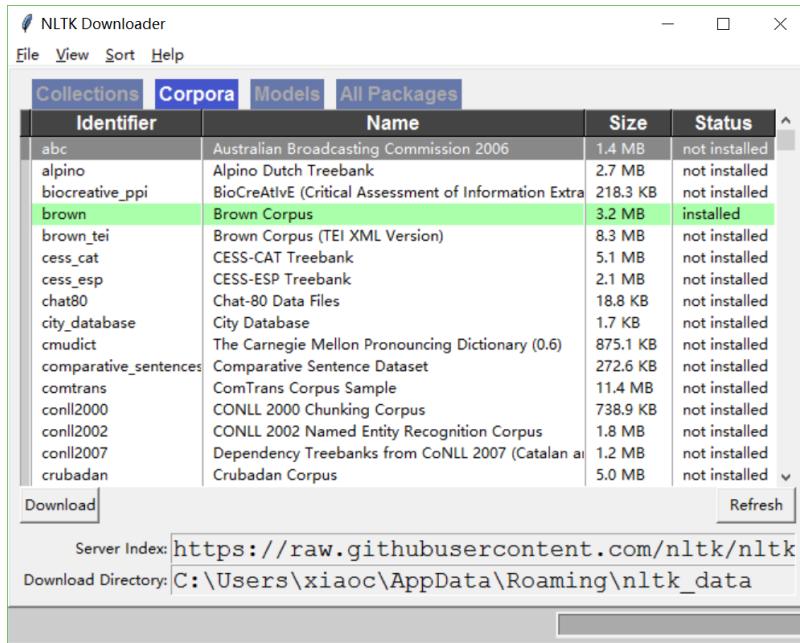
注意：1. 下文中提到的图书并不包括杂志。2. count最大为100，大于100的
scope: book_basic_r

获取图书信息	GET	/v2/book/:id
根据isbn获取图书信息	GET	/v2/book/isbn/:name
搜索图书	GET	/v2/book/search



```
>>> r = requests.get('https://api.douban.com/v2/book/1084336')
>>> r.text
'{"rating":{"max":10,"numRaters":218148,"average":"9.0","min":0
},"subtitle":"","author":["[法] 圣埃克苏佩里"],"pubdate":"2003-8","tags":[{"count":52078,"name":"小王子","title":"小王子"}, {"count":43966,"name":"童话", ... , "price":"22.00元"}]
```

NLTK语料库



便捷网络数据



```
>>> from nltk.corpus import gutenberg → brown
>>> import nltk
>>> print(gutenberg.fileids())
['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', 'bible-kjv.txt', 'blake-
poems.txt', 'bryant-stories.txt', 'burgess-busterbrown.txt', 'carroll-alice.txt',
'chesterton-ball.txt', 'chesterton-brown.txt', 'chesterton-thursday.txt', 'edgeworth-
parents.txt', 'melville-moby_dick.txt', 'milton-paradise.txt', 'shakespeare-caesar.txt',
'shakespeare-hamlet.txt', 'shakespeare-macbeth.txt', 'whitman-leaves.txt']
>>> texts = gutenberg.words('shakespeare-hamlet.txt')
>>> print(texts)
[['The', 'Tragedie', 'of', 'Hamlet', 'by', ...]]
```

用Python玩转数据

数据准备

2
3

数据形式

30支道指成分股
(dji) 股票数据的
逻辑结构

公司代码	公司名	最近一次成交价

美国运通公司
(quotes) 股票历
史数据的逻辑结构

收盘价	日期	最高价	最低价	开盘价	成交量

数据整理

djidf加列索引 (columns)

F
ile

```
# Filename: stock.py
import requests
import re
import pandas as pd
def retrieve_dji_list():
    ...
    return dji_list
dji_list = retrieve_dji_list()
djidf = pd.DataFrame(dji_list)
cols = ['code', 'name', 'lasttrade']
djidf.columns = cols
print(quotesdf)
```

0	1	2	
0 MMM	3M	195. 8	
1 AXP	American Express	76. 8	
2 AAPL	Apple	153. 06	
3 BA	Boeing	180. 76	
4 CAT	Caterpillar	102. 43	
5 CVX	Chevron	106. 52	
6 CSCO	Cisco	31. 21	
7 KO	Coca-Cola	43. 9	
8 DIS	Disney	107. 52	
9 DD	E I du Pont de Nemours	77. 82	
10 XOM	Exxon Mobil	81. 93	
11 GE	General Electric	28. 05	
12 GS	Goldman Sachs	215. 39	
13 HD	Home Depot	156. 3	
14 IBM	IBM	151. 98	
15 INTC	Intel	35. 4	
16 JNJ	Johnson & Johnson	127	
17 JPM	JPMorgan Chase	84. 78	
18 MCD	McDonald's	148. 15	
19 MRK	Merck	63. 78	
20 MSFT	Microsoft	67. 69	
21 NKE	Nike	51. 77	
22 PFE	Pfizer	32. 46	
23 PG	Procter & Gamble	86. 24	
24 TRV	Travelers Companies Inc	120. 79	
25 UTX	United Technologies	121. 16	
26 UNH	UnitedHealth	172. 59	
27 VZ	Verizon	45. 42	
28 V	Visa	92. 48	
29 WMT	Wal-Mart	78. 77	

数据整理

djidf数据：加完
columns的形式

code	name	lasttrade
MMM		
AXP		
AAPL		
...		
WMT		

quotesdf数据：
原始数据中已有
columns

close	date	high	low	open	volume
	1464010200				
	1464096600				
	1464183000				
	...				
	1495200600				

数据整理

用1,2,...作为index (行索引)

```
quotesdf = pd.DataFrame(quotes)
```

```
quotesdf.index = range(1,len(quotes)+1)
```

	close	date	high	low	open	volume
0	63.590000	1464010200	64.099998	63.560001	63.860001	3074100
1	64.870003	1464096600	65.099998	63.790001	63.790001	3946100
2	65.309998	1464183000	65.760002	65.010002	65.040001	5755900
3	65.230003	1464269400	65.370003	64.949997	65.290001	3593500
4	65.519997	1464355800	65.699997	65.330002	65.389999	3925700

	close	date	high	low	open	volume
1	63.590000	1464010200	64.099998	63.560001	63.860001	3074100
2	64.870003	1464096600	65.099998	63.790001	63.790001	3946100
3	65.309998	1464183000	65.760002	65.010002	65.040001	5755900
4	65.230003	1464269400	65.370003	64.949997	65.290001	3593500
5	65.519997	1464355800	65.699997	65.330002	65.389999	3925700

数据整理



如果可以直接用date作为索引，quotes的时间能否转换成普通日期形式（如下图中的效果）？

	close	high	low	open	volume
2016-05-23	63.590000	64.099998	63.560001	63.860001	3074100
2016-05-24	64.870003	65.099998	63.790001	63.790001	3946100
2016-05-25	65.309998	65.760002	65.010002	65.040001	5755900
2016-05-26	65.230003	65.370003	64.949997	65.290001	3593500
2016-05-27	65.519997	65.699997	65.330002	65.389999	3925700
2016-05-31	65.760002	65.919998	65.400002	65.699997	5256000
2016-06-01	65.910004	65.959999	65.180000	65.760002	3816000
2016-06-02	66.410004	66.410004	65.599998	65.860001	3052200
2016-06-03	65.489998	65.820000	64.769997	65.529999	4336100
2016-06-06	65.940002	66.199997	65.500000	65.550003	3915200
2016-06-07	65.889999	66.599998	65.879997	66.150002	3779500
2016-06-08	66.260002	66.580002	65.940002	65.940002	2601100

S
ource

```
>>> from datetime import date
>>> firstday = date.fromtimestamp(1464010200)
>>> lastday = date.fromtimestamp(1495200600)
>>> firstday
datetime.date(2016, 5, 23)
>>> lastday
datetime.date(2017, 5, 19)
```

时间序列

F_{ile}

```
# Filename: quotes_history_v2.py
def retrieve_quotes_historical(stock_code):
    ...
    return [item for item in quotes if not 'type' in item]
quotes = retrieve_quotes_historical('AXP')
list1 = []
for i in range(len(quotes)):
    x = date.fromtimestamp(quotes[i]['date'])  
|—————→ 转换成常规时间
    y = date.strftime(x, '%Y-%m-%d')  
|—————→ 转换成固定格式
    list1.append(y)
quotesdf_ori = pd.DataFrame(quotes, index = list1)
quotesdf_m = quotesdf_ori.drop(['unadjclose'], axis = 1)  
|—————→ 删除原unadjclose列
quotesdf = quotesdf_m.drop(['date'], axis = 1)  
|—————→ 删除原date列
print(quotesdf)
```

创建时间序列



```
>>> import pandas as pd
>>> dates = pd.date_range('20170520', periods=7)
>>> dates
<class 'pandas.tseries.index.DatetimeIndex'>
[2017-05-20, ..., 2017-05-26]
Length: 7, Freq: D, Timezone: None
>>> import numpy as np
>>> datesdf = pd.DataFrame(np.random.randn(7,3), index=dates, columns = list('ABC'))
>>> datesdf
          A         B         C
2017-05-20  1.302600 -1.214708  1.411628
2017-05-21 -0.512343  2.277474  0.403811
2017-05-22 -0.788498 -0.217161  0.173284
2017-05-23  1.042167 -0.453329 -2.107163
2017-05-24 -1.628075  1.663377  0.943582
2017-05-25 -0.091034  0.335884  2.455431
2017-05-26 -0.679055 -0.865973  0.246970
```



用Python玩转数据

3

数据显示



数据显示

	code		name	lasttrade
0	MMM		3M	195.80
1	AXP	American Express		76.80
2	AAPL		Apple	153.06
3	BA		Boeing	180.76
4	CAT	Caterpillar		102.43
5	CVX		Chevron	106.52
6	CSCO		Cisco	31.21
7	KO		Coca-Cola	43.90
8	DIS		Disney	107.52
9	DD	E I du Pont de Nemours and Co		77.82
10	XOM		Exxon Mobil	81.93
11	GE	General Electric		28.05
12	GS		Goldman Sachs	215.39
13	HD		Home Depot	156.30
14	IBM		IBM	151.98
15	INTC		Intel	35.40
16	JNJ	Johnson & Johnson		127.00
17	JPM		JPMorgan Chase	84.78
18	MCD		McDonald's	148.15
19	MRK		Merck	63.78
20	MSFT		Microsoft	67.69
21	NKE		Nike	51.77
22	PFE		Pfizer	32.46
23	PG	Procter & Gamble		86.24
24	TRV	Travelers Companies Inc		120.79
25	UTX	United Technologies		121.16
26	UNH		UnitedHealth	172.59
27	VZ		Verizon	45.42
28	V		Visa	92.48
29	WMT		Wal-Mart	78.77

djidf

	close	high	low	open	volume
2016-05-23	63.590000	64.099998	63.560001	63.860001	3074100
2016-05-24	64.870003	65.099998	63.790001	63.790001	3946100
2016-05-25	65.309998	65.760002	65.010002	65.040001	5755900
2016-05-26	65.230003	65.370003	64.949997	65.290001	3593500
2016-05-27	65.519997	65.699997	65.330002	65.389999	3925700
2016-05-31	65.760002	65.919998	65.400002	65.699997	5256000
2016-06-01	65.910004	65.959999	65.180000	65.760002	3816000
2016-06-02	66.410004	66.410004	65.599998	65.860001	3052200
2016-06-03	65.489998	65.820000	64.769997	65.529999	4336100
2016-06-06	65.940002	66.199997	65.500000	65.550003	3915200
2016-06-07	65.889999	66.599998	65.879997	66.150002	3779500
2016-06-08	66.260002	66.580002	65.940002	65.940002	2601100
2016-06-09	65.709999	65.779999	64.900002	65.720001	3883800
2016-06-10	64.970001	65.480003	64.709999	65.260002	3939100
2016-06-13	63.669998	64.889999	63.630001	64.800003	5883400
2016-06-14	61.070000	63.660000	60.380001	63.590000	12323200
2016-06-15	61.419998	62.160000	60.860001	61.470001	5979900

quotesdf

数据显示

显示方式：

- 显示行索引
- 显示列索引
- 显示数据的值
- 显示数据描述



```
>>> list(djidf.index)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
23, 24, 25, 26, 27, 28, 29]
>>> list(djidf.columns)
['code', 'name', 'lasttrade']
>>> dijdf.values
array([['MMM', '3M', 195.8],
       ...,
       ['WMT', 'Wal-Mart', 78.77]], dtype=object)
>>> djidt.describe
<bound method NDFrame.describe of   code          name  lasttrade
0                      MM            3M    195.80
...
29
WMT  Wal-Mart    78.77>
```

数据显示

数据的格式

S_{ource}

```
>>> djidf.lasttrade  
1  199.54  
2  77.44  
3  153.87  
...  
30  78.31  
Name: lasttrade, dtype: float64
```

```
dji_list = []  
for item in dji_list_in_text:  
    dji_list.append([item[0], item[1], float(item[2])])
```

数据显示



查看道指成分股
中前5只和后5
只的股票基本信
息？

显示方式：

- 显示行
 - 专用方式
 - 切片
 - 显示列

Source

>>> djidf.head(5)

	code	name	lasttrade
0	MMM	3M	195.80
1	AXP	American Express	76.80
2	AAPL	Apple	153.06
3	BA	Boeing	180.76
4	CAT	Caterpillar	102.43

djidf[:5]

>>> djidf.tail(5)

	code	name	lasttrade
25	UTX	United Technologies	121.16
26	UNH	UnitedHealth	172.59
27	VZ	Verizon	45.42
28	V	Visa	92.48
29	WMT	Wal-Mart	78.77

djidf[-5:]



用Python玩转数据

4

数据选择



数据选择

	code		name	lasttrade
0	MMM		3M	195.80
1	AXP	American Express		76.80
2	AAPL		Apple	153.06
3	BA		Boeing	180.76
4	CAT	Caterpillar		102.43
5	CVX		Chevron	106.52
6	CSCO		Cisco	31.21
7	KO	Coca-Cola		43.90
8	DIS		Disney	107.52
9	DD	E I du Pont de Nemours and Co		77.82
10	XOM		Exxon Mobil	81.93
11	GE	General Electric		28.05
12	GS		Goldman Sachs	215.39
13	HD	Home Depot		156.30
14	IBM		IBM	151.98
15	INTC		Intel	35.40
16	JNJ	Johnson & Johnson		127.00
17	JPM	JPMorgan Chase		84.78
18	MCD		McDonald's	148.15
19	MRK		Merck	63.78
20	MSFT		Microsoft	67.69
21	NKE		Nike	51.77
22	PFE	Pfizer		32.46
23	PG	Procter & Gamble		86.24
24	TRV	Travelers Companies Inc		120.79
25	UTX	United Technologies		121.16
26	UNH	UnitedHealth		172.59
27	VZ	Verizon		45.42
28	V	Visa		92.48
29	WMT	Wal-Mart		78.77

选择方式：

- 选择行
- 选择列
- 选择区域
- 筛选 (条件选择)

	close	high	low	open	volume
2016-05-23	63.590000	64.099998	63.560001	63.860001	3074100
2016-05-24	64.870003	65.099998	63.790001	63.790001	3946100
2016-05-25	65.309998	65.760002	65.010002	65.040001	5755900
2016-05-26	65.230003	65.370003	64.949997	65.290001	3593500
2016-05-27	65.519997	65.699997	65.330002	65.389999	3925700
2016-05-31	65.760002	65.919998	65.400002	65.699997	5256000
2016-06-01	65.910004	65.959999	65.180000	65.760002	3816000
2016-06-02	66.410004	66.410004	65.599998	65.860001	3052200
2016-06-03	65.489998	65.820000	64.769997	65.529999	4336100
2016-06-06	65.940002	66.199997	65.500000	65.550003	3915200
2016-06-07	65.889999	66.599998	65.879997	66.150002	3779500
2016-06-08	66.260002	66.580002	65.940002	65.940002	2601100
2016-06-09	65.709999	65.779999	64.900002	65.720001	3883800
2016-06-10	64.970001	65.480003	64.709999	65.260002	3939100
2016-06-13	63.669998	64.889999	63.630001	64.800003	5883400
2016-06-14	61.070000	63.660000	60.380001	63.590000	12323200
2016-06-15	61.419998	62.160000	60.860001	61.470001	5979900

数据选择



美国运通公司
2017年5月1日至
2017年5月5日间
的股票交易信息？

选择方式：

- 选择行
 - 切片
 - 索引

S
ource

```
>>> quotesdf['2017-05-01':'2017-05-05']
```

	close	high	low	open	volume
2017-05-01	79.230003	79.489998	78.879997	79.220001	3458100
2017-05-02	79.540001	79.660004	79.150002	79.150002	3334900
2017-05-03	78.830002	79.510002	78.690002	79.230003	3800600
2017-05-04	78.330002	79.419998	77.989998	79.230003	3902200
2017-05-05	78.320000	78.730003	77.879997	78.610001	2936700

数据选择



道指成分股公司
代码？

选择方式：

- 选择列
 - 列名

S
ource

```
>>> djidf['code']  
0    MMM  
1    AXP  
2    AAPL  
...  
29   WMT  
Name: code, dtype: object  
>>> djidf.code  
0    MMM  
1    AXP  
2    AAPL  
...  
29   WMT  
Name: code, dtype: object
```

不支持

djidf['code', 'lasttrade']
djidf['code':'lasttrade']

数据选择



道指成分股中行
索引是1至5的股
票信息以及所有
股票的代码和最
近一次交易价？

选择方式：

- 行、列
 - 标签label (loc)



```
>>> djidf.loc[1:5,]
   code
1 AXP
2 AAPL
3 BA
4 CAT
5 CVX
>>> djidf.loc[:, ['code', 'lasttrade']]
   code lasttrade
0  MMM    195.80
1  AXP     76.80
2  AAPL    153.06
...
29 WMT     78.77
```

	name	lasttrade
American Express	76.80	
Apple	153.06	
Boeing	180.76	
Caterpillar	102.43	
Chevron	106.52	

数据选择



道指成分股中行索引
是1至5的股票代码和
最近一次交易价?行
索引是1的股票的最
近一次交易价?

选择方式：

- 行和列的区域
 - 标签label (loc)
- 单个值
 - at

S
ource

```
>>> djidf.loc[1:5, ['code','lasttrade']]
   code  lasttrade
1    AXP    76.80
2    AAPL   153.06
3     BA   180.76
4    CAT   102.43
5    CVX   106.52
>>> djidf.loc[1, 'lasttrade']
76.79999999999999
>>> djidf.at[1, 'lasttrade']
76.79999999999999
```

数据选择

选择方式：

- 行、列和区域
 - 用`iloc`（位置）
- 取某个值
 - `iat`

Source

```
>>> djidf.loc[1:5,['code','lasttrade']]
```

	code	lasttrade
1	AXP	76.80
2	AAPL	153.06
3	BA	180.76
4	CAT	102.43
5	CVX	106.52

如果直接写成
[1:6, 0:2]则表
示列索引即第
0和第1列

Source

```
>>> djidf.iloc[1:6,[0,2]]
```

	code	lasttrade
1	AXP	76.80
2	AAPL	153.06
3	BA	180.76
4	CAT	102.43
5	CVX	106.52

Source

```
>>> djidf.loc[1,'lasttrade']
```

76.799999999999997

```
>>> djidf.at[1,'lasttrade']
```

76.799999999999997

Source

```
>>> djidf.iloc[1,2]
```

76.799999999999997

```
>>> djidf.iat[1,2]
```

76.799999999999997

数据选择

美国运通公司本年度3月份的股票信息？进一步寻找美国运通公司本年度一季度收盘价大于等于80的记录？

选择方式：

- 条件筛选

S
ource

```
>>> quotesdf[(quotesdf.index >= '2017-03-01') & (quotesdf.index <= '2017-03-31')]
```

	close	high	low	open	volume
2017-03-01	81.919998	82.000000	81.019997	81.050003	4746400
2017-03-02	80.099998	81.660004	80.059998	81.660004	4409800
...					
2017-03-31	79.110001	79.430000	78.800003	78.930000	5228400

```
>>> quotesdf[(quotesdf.index >= '2017-01-01') & (quotesdf.index <= '2017-03-31') & (quotesdf.close >= 80)]
```

	open	close	high	low	volume
2017-02-23	80.050003	80.449997	79.769997	79.870003	3339500
2017-02-27	80.169998	80.309998	79.589996	79.750000	2619400
2017-02-28	80.059998	80.489998	79.769997	80.120003	4415300
2017-03-01	81.919998	82.000000	81.019997	81.050003	4746400
2017-03-02	80.099998	81.660004	80.059998	81.660004	4409800



用Python玩转数据

5

简单统计与处理



简单统计与筛选

求道指成分股中
30只股票最近一
次成交价的平均值 ?
股票最近一次成交
价大于等于180的
公司名 ?



```
>>> djidf.lasttrade.mean()  
101.26500000000001  
>>> djidf[djidf.lasttrade >= 180].name  
0           3M  
3          Boeing  
12  Goldman Sachs  
Name: name, dtype: object
```

简单统计与筛选



统计美国运通公司近一年股票涨和跌分别的天数？



统计美国运通公司近一年相邻两天收盘价的涨跌情况？



```
>>> len(quotesdf[quotesdf.close > quotesdf.open])  
123  
>>> len(quotesdf)-123  
128
```



```
>>> status = np.sign(np.diff(quotesdf.close))  
>>> status  
array([ 1.,  1., -1., ..., -1.,  1.,  1.])  
>>> status[np.where( status == 1.)].size  
132  
>>> status[np.where( status == -1.)].size  
118
```

排序



按最近一次成交价对30只道指成分股股票进行排序。根据排序结果列出前三甲公司名。

S
ource

```
>>> tempdf = djidf.sort_values(by = 'lasttrade', ascending = False)
   code           name  lasttrade
12   GS    Goldman Sachs    215.39
0   MMM            3M    195.80
3   BA        Boeing    180.76
26  UNH  UnitedHealth    172.59
...
>>> tempdf[:3].name
12  Goldman Sachs
0          3M
3        Boeing
Name: name, dtype: object
```

计数统计



统计本年度1月份的股票开盘天数？

S_{ource}

```
>>> t = quotesdf[(quotesdf.index >= '2017-01-01') & (quotesdf.index < '2017-02-01')]  
>>> len(t)  
20
```

计数统计



统计近一年每个月的股票开盘天数？

F
ile

```
# Filename: quotes_month.py
import time
...
listtemp = []
for i in range(len(quotesdf)):
    temp = time.strptime(quotesdf.index[i], "%Y-%m-%d")
    listtemp.append(temp.tm_mon)
tempdf = quotesdf.copy()
tempdf['month'] = listtemp
print(tempdf['month'].value_counts())
```

Output:

8	23
3	23
6	22
12	21
11	21
10	21
9	21
5	21
7	20
1	20
4	19
2	19

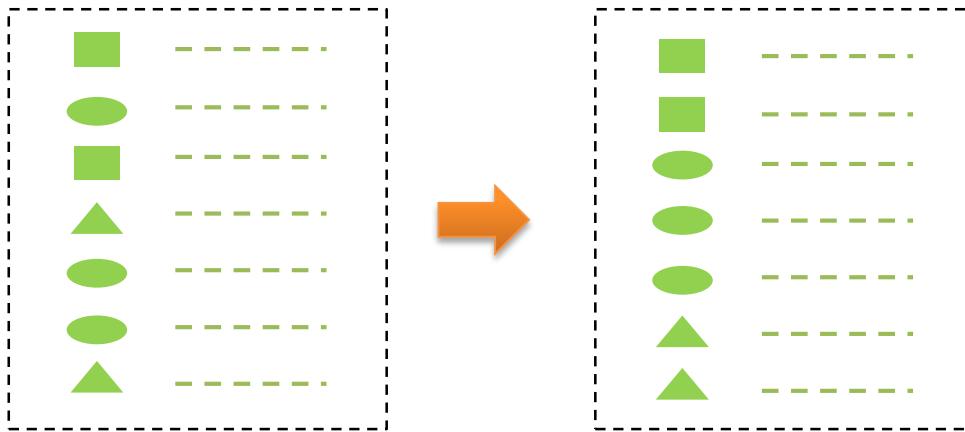
Name: month,
dtype: int64

6

用Python玩转数据

GROUPING

分组



Grouping的顺序

- ① Splitting
- ② Applying
- ③ Combining

分组



统计近一年每个月的股票开盘天数？

S
ource

```
>>> x = tempdf.groupby('month').count()
           close  high  low  open  volume
month
1          20    20   20    20     20
2          19    19   19    19     19
3          23    23   23    23     23
...
11         21    21   21    21     21
12         21    21   21    21     21
```

```
>>> x.close
```

Output:

month	
1	20
2	19
3	23
4	19
5	21
6	22
7	20
8	23
9	21
10	21
11	21
12	21

Name: month, dtype: int64

分组



统计近一年每个月的总成交量？

S
ource

```
>>> tempdf.groupby('month').sum().volume  
month  
1    103887100  
2    65816600  
3    98700800  
4    77893800  
...  
10   116243400  
11   99527200  
12   75948200  
Name: volume, dtype: float64
```

mean()

min()

max()

...

分组



如果更高效统计近一年每个月的总成交量？

S
ource

```
tempdf.groupby('month').sum().volume
```



```
>>> tempdf.groupby('month').volume.sum()
month
1    103887100
2     65816600
3    98700800
4    77893800
...
12   75948200
Name: volume, dtype: float64
```



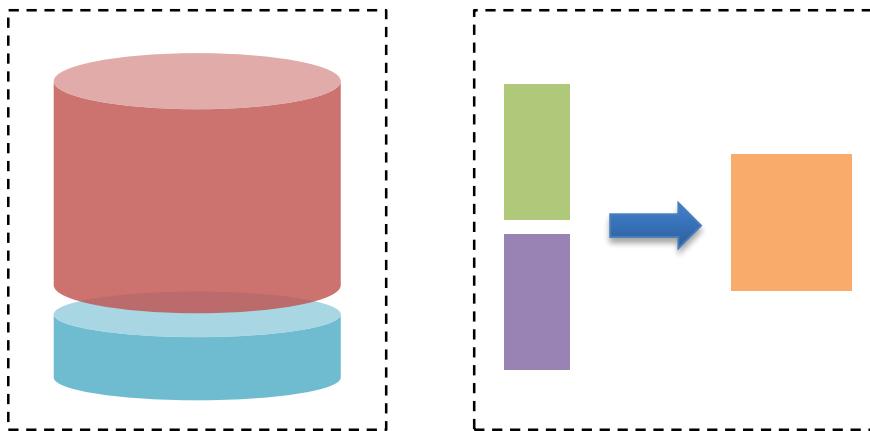
用Python玩转数据

7

MERGE



合并



Merge的形式

- Append
 - 加行到DataFrame
- Concat
 - 连接pandas对象
- Join
 - SQL类型的连接

Append



把美国运通公司
本年度1月1日
至1月5日间的
股票交易信息合
并到近一年中前
两天的股票信息
中？

S
ource

```
>>> p = quotesdf[:2]
>>> p
```

	open	close	high	low	volume
2016-05-23	63.590000	64.099998	63.560001	63.860001	3074100
2016-05-24	64.870003	65.099998	63.790001	63.790001	3946100

```
>>> q = quotesdf['2017-01-01':'2017-01-05']
>>> q
```

	open	close	high	low	volume
2017-01-03	75.349998	75.750000	74.739998	74.889999	5853900
2017-01-04	76.260002	76.550003	75.059998	75.260002	4635800
2017-01-05	75.320000	76.180000	74.820000	76.000000	3383000

```
>>> p.append(q)
```

	open	close	high	low	volume
2016-05-23	63.590000	64.099998	63.560001	63.860001	3074100
2016-05-24	64.870003	65.099998	63.790001	63.790001	3946100
2017-01-03	75.349998	75.750000	74.739998	74.889999	5853900
2017-01-04	76.260002	76.550003	75.059998	75.260002	4635800
2017-01-05	75.320000	76.180000	74.820000	76.000000	3383000

Concat

将美国运通公司近一年股票数据中的前5个和后5个合并。

S
ource

```
>>> pieces = [tempdf[:5], tempdf[len(tempdf)-5:]]  
>>> pd.concat(pieces)
```

	open	close	high	low	volume	month
2016-05-23	63.590000	64.099998	63.560001	63.860001	3074100	5
2016-05-24	64.870003	65.099998	63.790001	63.790001	3946100	5
2016-05-25	65.309998	65.760002	65.010002	65.040001	5755900	5
2016-05-26	65.230003	65.370003	64.949997	65.290001	3593500	5
2016-05-27	65.519997	65.699997	65.330002	65.389999	3925700	5
2017-05-15	78.330002	78.620003	77.480003	77.480003	3327000	5
2017-05-16	78.129997	78.639999	77.839996	78.599998	2457500	5
2017-05-17	76.370003	78.129997	76.239998	78.129997	4441600	5
2017-05-18	76.379997	76.849998	75.970001	76.269997	3545700	5
2017-05-19	76.800003	77.349998	76.300003	76.550003	3278200	5

Concat



两个不同逻辑结构
的对象能否连接？

objs	axis
join	join_axes
keys	levels
names	verify_integrity
ignore_index	

S
ource

```
>>> piece1 = quotesdf[:3]
>>> piece2 = tempdf[:3]
>>> pd.concat([piece1,piece2], ignore_index = True)
      close      high      low month      open    volume
0  63.590000  64.099998  63.560001   NaN  63.860001  3074100
1  64.870003  65.099998  63.790001   NaN  63.790001  3946100
2  65.309998  65.760002  65.010002   NaN  65.040001  5755900
3  63.590000  64.099998  63.560001  5.0  63.860001  3074100
4  64.870003  65.099998  63.790001  5.0  63.790001  3946100
5  65.309998  65.760002  65.010002  5.0  65.040001  5755900
```

Join

code	name
AXP	
KO	



code	name	volume	month
AXP			
AXP			
KO			
KO			

volume	code	month
	AXP	
	AXP	
	KO	
	KO	

Join



将美国运通公司
和可口可乐公司
近一年中每个月
的交易总量表
(包含公司代码)
与30只道琼斯
成分股股票信息
合并。

code | name | volume | month

	code		name	lasttrade
0	MMM		3M	195.80
1	AXP	American Express		76.80
2	AAPL		Apple	153.06
3	BA		Boeing	180.76
4	CAT		Caterpillar	102.43
5	CVX		Chevron	106.52
6	CSCO		Cisco	31.21
7	KO		Coca-Cola	43.90
8	DIS		Disney	107.52
9	DD	E I du Pont de Nemours and Co		77.82
10	XOM		Exxon Mobil	81.93
11	GE	General Electric		28.05
12	GS		Goldman Sachs	215.39
13	HD		Home Depot	156.30
14	IBM		IBM	151.98
15	INTC		Intel	35.40
16	JNJ	Johnson & Johnson		127.00
17	JPM		JPMorgan Chase	84.78
18	MCD		McDonald's	148.15
19	MRK		Merck	63.78
20	MSFT		Microsoft	67.69
21	NKE		Nike	51.77
22	PFE		Pfizer	32.46
23	PG	Procter & Gamble		86.24
24	TRV	Travelers Companies Inc		120.79
25	UTX		United Technologies	121.16
26	UNH		UnitedHealth	172.59
27	VZ		Verizon	45.42
28	V		Visa	92.48
29	WMT		Wal-Mart	78.77

month	volume	code	month
1	103887100	AXP	1
2	65816600	AXP	2
3	98700800	AXP	3
4	77893800	AXP	4
5	76209200	AXP	5
6	121788800	AXP	6
7	90064900	AXP	7
8	77514100	AXP	8
9	95572800	AXP	9
10	116243400	AXP	10
11	99527200	AXP	11
12	75948200	AXP	12
1	240321400	KO	1
2	333983800	KO	2
3	339185400	KO	3
4	232465400	KO	4
5	239687800	KO	5
6	265483400	KO	6
7	235959400	KO	7
8	235118300	KO	8
9	251007200	KO	9
10	264839100	KO	10
11	316557000	KO	11
12	283871000	KO	12

djidf

AKdf

Join

S
ource

```
>>> pd.merge(djidf.drop(['lasttrade'], axis = 1), AKdf, on = 'code')
```

	code	name	volume	month
0	AXP	American Express	103887100	1
1	AXP	American Express	65816600	2
2	AXP	American Express	98700800	3
3	AXP	American Express	77893800	4
4	AXP	American Express	76209200	5
...				
19	KO	Coca-Cola	235118300	8
20	KO	Coca-Cola	251007200	9
21	KO	Coca-Cola	264839100	10
22	KO	Coca-Cola	316557000	11
23	KO	Coca-Cola	283871000	12

	code	name	volume	month
0	AXP	American Express	103887100	1
1	AXP	American Express	65816600	2
2	AXP	American Express	98700800	3
3	AXP	American Express	77893800	4
4	AXP	American Express	76209200	5
5	AXP	American Express	121788800	6
6	AXP	American Express	90064900	7
7	AXP	American Express	77514100	8
8	AXP	American Express	95572800	9
9	AXP	American Express	116243400	10
10	AXP	American Express	99527200	11
11	AXP	American Express	75948200	12
12	KO	Coca-Cola	240321400	1
13	KO	Coca-Cola	333983800	2
14	KO	Coca-Cola	339185400	3
15	KO	Coca-Cola	232465400	4
16	KO	Coca-Cola	239687800	5
17	KO	Coca-Cola	265483400	6
18	KO	Coca-Cola	235959400	7
19	KO	Coca-Cola	235118300	8
20	KO	Coca-Cola	251007200	9
21	KO	Coca-Cola	264839100	10
22	KO	Coca-Cola	316557000	11
23	KO	Coca-Cola	283871000	12

merge函数的参数

left	right	how
on	left_on	right_on
left_index	right_index	sort
suffixes	copy	



Advanced Data Processing and Visualization of Python

Python高级数据处理与可视化

Department of Computer Science and Technology

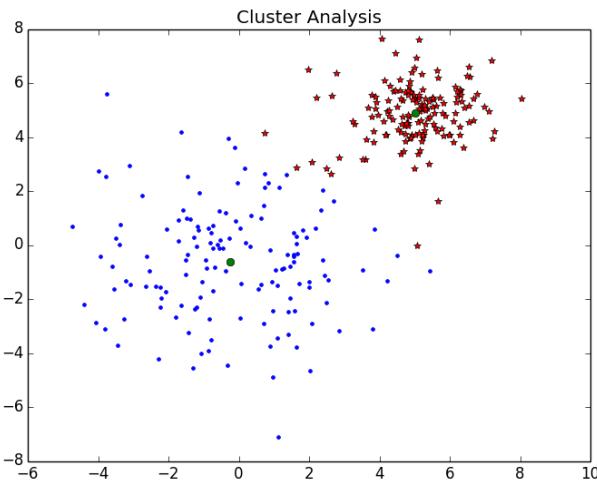
Department of University Basic Computer Teaching

用Python玩转数据

聚类分析

1

聚类



- 聚类分析(cluster analysis)

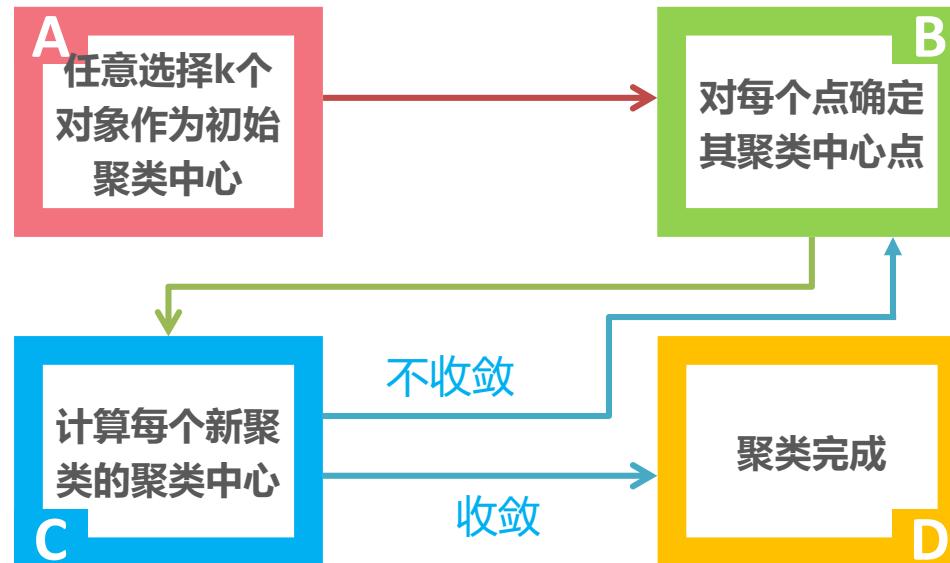
以相似性为基础把相似的对象通过静态分类的方法分成不同的组别或者更多的子集

- 特性

- 基于相似性
 - 有多个聚类中心

K-MEANS

K-均值算法表示以空间中k个点为中心进行聚类，对最靠近他们的对象归类。



一个日常小例子

	高数	英语	Python	音乐
小明	88	64	96	85
大明	92	99	95	94
小朋	91	87	99	95
大朋	78	99	97	81
小萌	88	78	98	84
大萌	100	95	100	92

Output:

[1 0 0 1 1 0]

F
ile

```
# Filename: kmeansStu1.py
import numpy as np
from scipy.cluster.vq import vq, kmeans, whiten
list1 = [88.0, 74.0, 96.0, 85.0]
list2 = [92.0, 99.0, 95.0, 94.0]
list3 = [91.0, 87.0, 99.0, 95.0]
list4 = [78.0, 99.0, 97.0, 81.0]
list5 = [88.0, 78.0, 98.0, 84.0]
list6 = [100.0, 95.0, 100.0, 92.0]
data = np.array([list1, list2, list3, list4, list5, list6])
whiten = whiten(data)
centroids, _ = kmeans(whiten, 2)
result, _ = vq(whiten, centroids)
print(result)
```

用专业工具解决

F
ile

```
# Filename: kmeansStu2.py
import numpy as np
from sklearn.cluster import KMeans
list1 = [88.0,74.0,96.0,85.0]
list2 = [92.0,99.0,95.0,94.0]
list3 = [91.0,87.0,99.0,95.0]
list4 = [78.0,99.0,97.0,81.0]
list5 = [88.0,78.0,98.0,84.0]
list6 = [100.0,95.0,100.0,92.0]
X = np.array([list1,list2,list3,list4,list5,list6])
kmeans = KMeans(n_clusters = 2).fit(X)
pred = kmeans.predict(X)
print(pred)
```



```
from sklearn import datasets
from sklearn import svm
clf = svm.SVC(gamma=0.001, C=100.)
digits = datasets.load_digits()
clf.fit(digits.data[:-1], digits.target[:-1])
clf.predict(digits.data[-1])
```

Output:

[0 1 1 1 0 1]

另一个例子



基于10只道指成分股股票近一年来相邻两天的收盘价涨跌数据规律
对它们进行聚类

File

```
['MMM','AXP','AAPL','BA','CAT','CVX','CSCO','KO','DIS','DD']
```

```
# Filename: kmeansDJI.py
listDji = ['MMM','AXP','AAPL','BA','CAT','CVX','CSCO','KO','DIS','DD']
listTemp = [0] * len(listDji)
for i in range(len(listTemp)):
    listTemp[i] = create_df(listDji[i]).close    # a function for creating a DataFrame
status = [0] * len(listDji)
for i in range(len(status)):
    status[i] = np.sign(np.diff(listTemp[i]))
kmeans = KMeans(n_clusters = 3).fit(status)
pred = kmeans.predict(status)
print(pred)
```

Output:

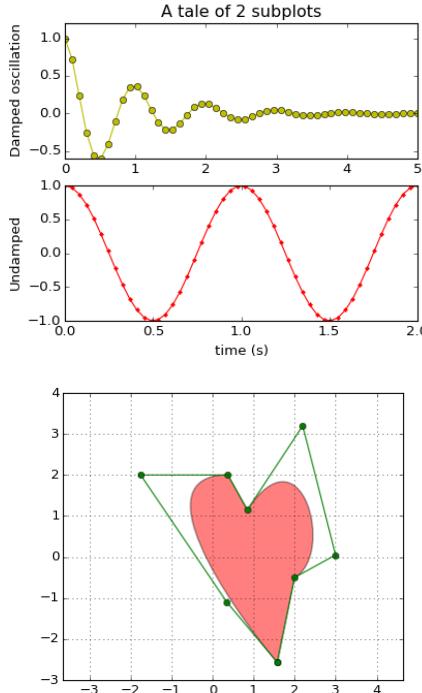
[2 0 2 2 0 0 2 2 1 1]

用Python玩转数据

2
3

MATPLOTLIB 绘图基础

Matplotlib绘图



- **Matplotlib绘图**

**最著名Python绘图库，
主要用于二维绘图**

- 画图质量高
- 方便快捷的绘图模块
 - 绘图API——pyplot模块
 - 集成库——pylab模块（包含NumPy和pyplot中的常用函数）

数据源

可口可乐公司近一年来股票收盘价的月平均价



S
ource

```
>>> closeMeansKO = tempkodf.groupby('month').close.mean()  
>>> closeMeansKO  
month  
1    41.440500  
2    41.350526  
3    42.241304  
4    42.934210  
...  
10   41.979524  
11   41.523809  
12   41.345714
```

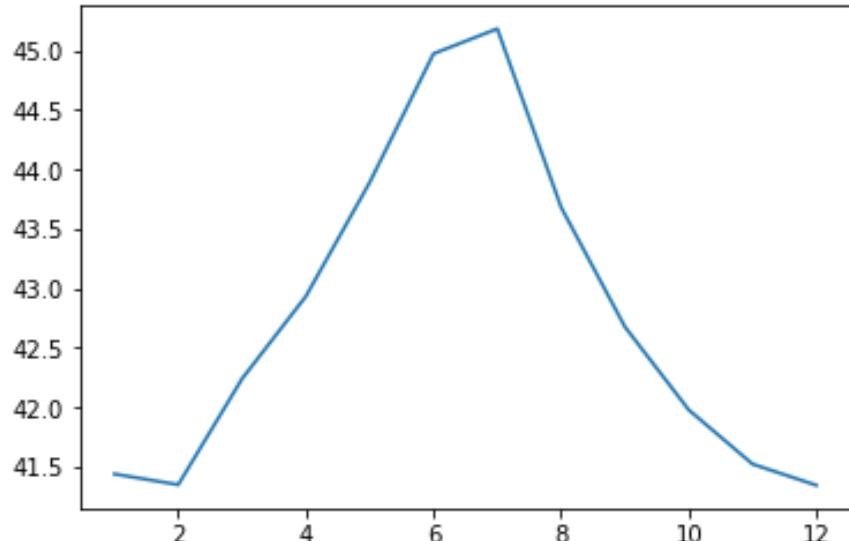
折线图



将可口可乐公司近一年来股票收盘价的月平均价绘制成折线图

F_{ile}

```
# Filename: plotKO.py
import matplotlib.pyplot as plt
...
x = closeMeansKO.index
y = closeMeansKO.values
plt.plot(x, y)
```

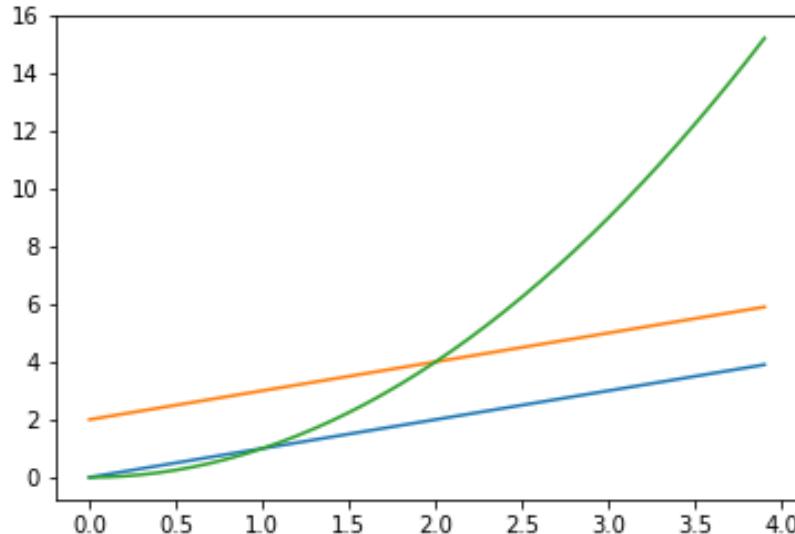


折线图

NumPy数组也可以作为
Matplotlib的参数

S
ource

```
>>> import numpy as np  
>>> import matplotlib.pyplot as plt  
>>> t=np.arange(0.,4.,0.1)  
>>> plt.plot(t, t, t, t+2, t, t**2)
```



散点图

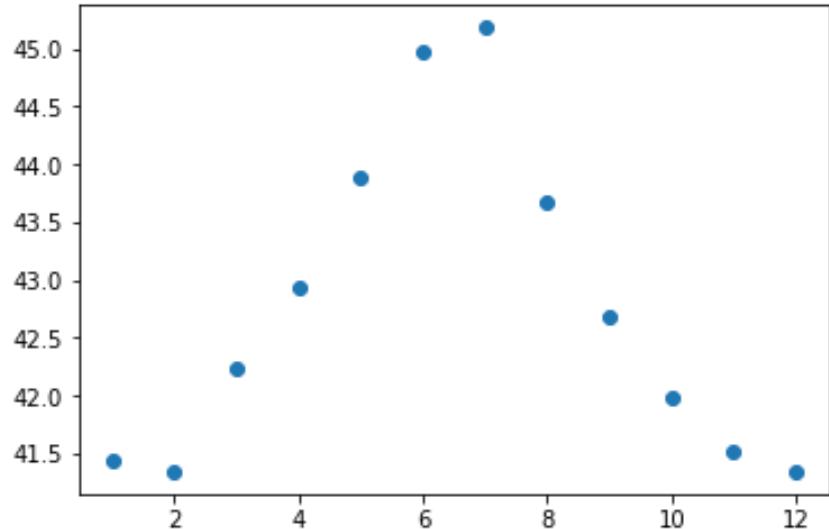


将可口可乐公司近一年来
股票收盘价的月平均价绘
制成散点图

`plt.plot(x, y)`



`plt.plot(x, y, 'o')`



柱状图

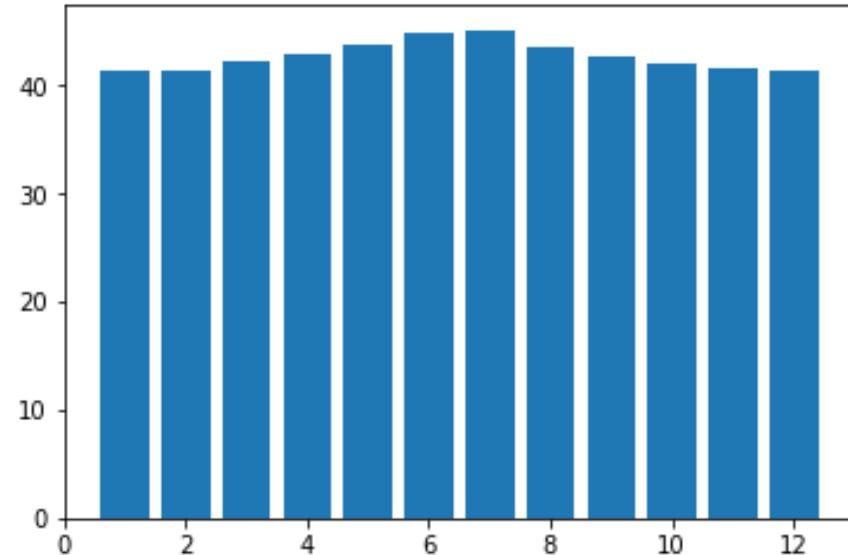


将可口可乐公司近一年来
股票收盘价的月平均价绘
制成柱状图

plt.plot(x, y)



plt.bar(x, y)

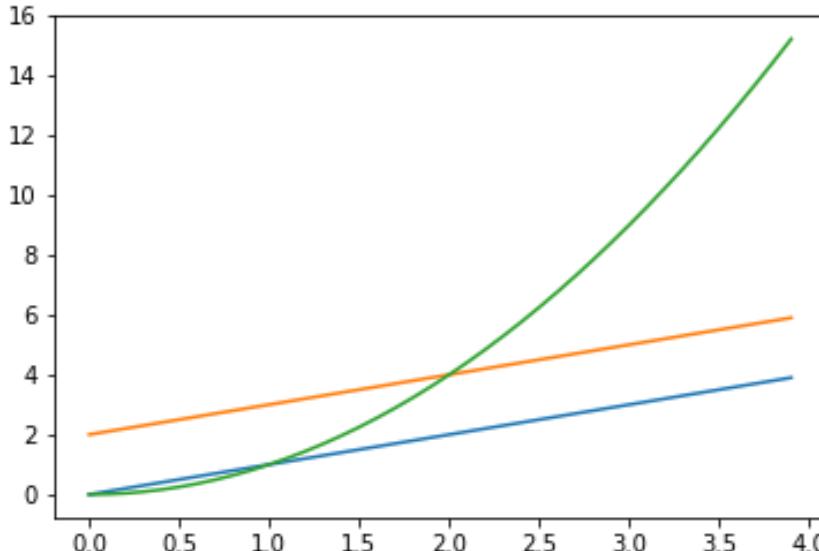


pylab绘图

NumPy数组也可以作为
Matplotlib的参数

S_{ource}

```
>>> import numpy as np  
>>> import pylab as pl  
>>> t=np.arange(0.,4.,0.1)  
>>> pl.plot(t,t,t,t+2,t,t**2)
```





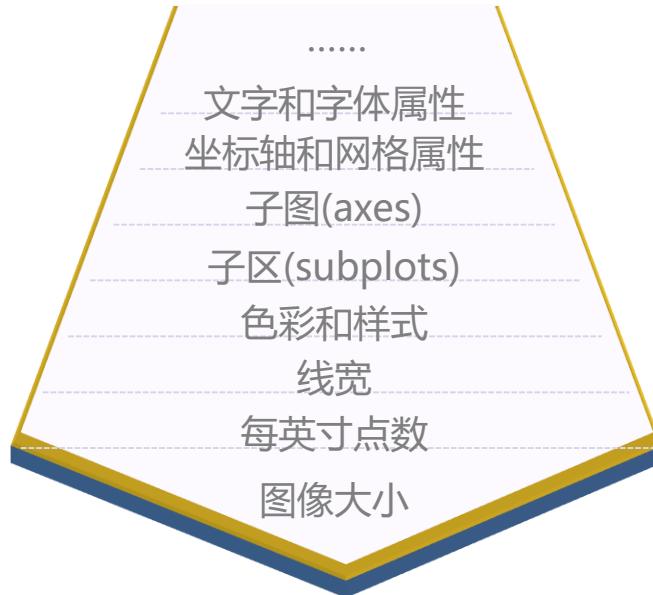
用Python玩转数据

3

MATPLOTLIB 图像属性控制



Matplotlib属性

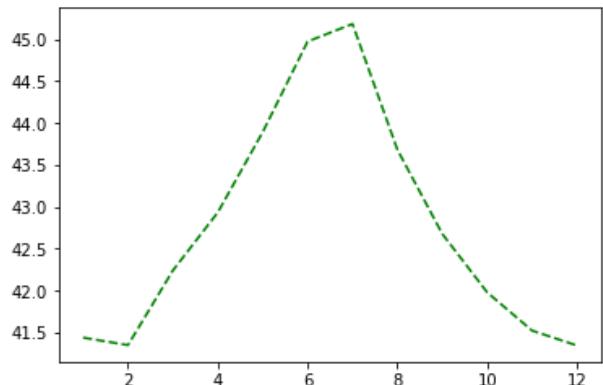


Matplotlib可以控制的默认属性

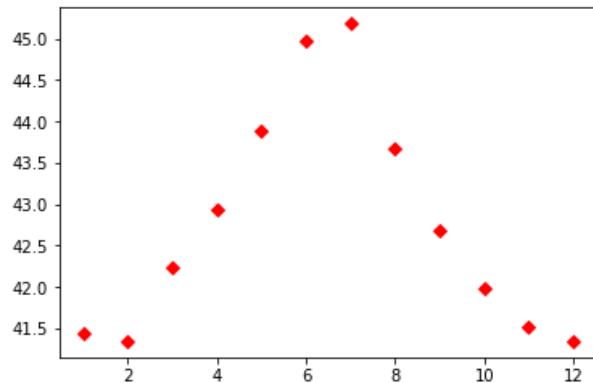
色彩和样式



绘图颜色
和线条类
型和样式
可以更改
吗？



```
plt.plot(x, y, 'g--')
```



```
plt.plot(x, y, 'rD')
```

色彩和样式

符号	颜色
b	blue
g	green
r	red
c	cyan
m	magenta
Y	yellow
k	black
w	white

线型	描述
'-'	solid
'--'	dashed
'-.'	dash_dot
::	dotted
'None'	draw nothing
' '	draw nothing
" "	draw nothing

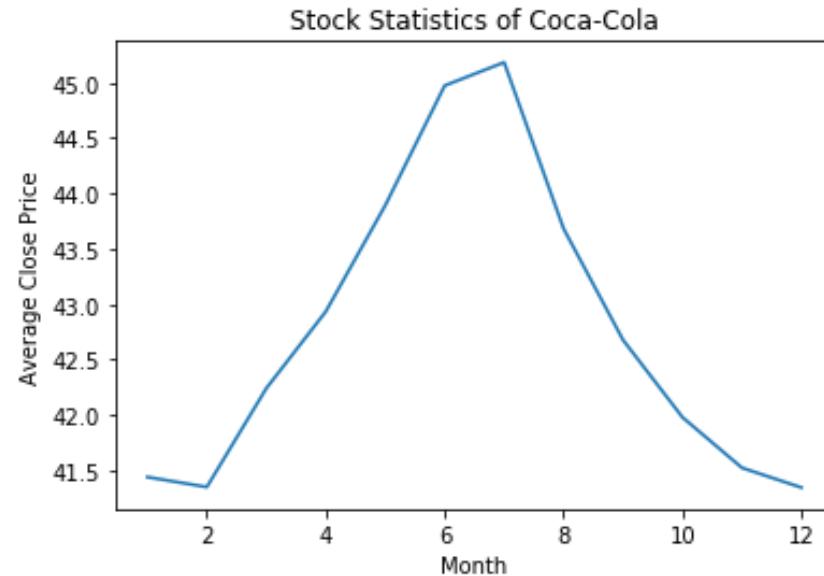
标记	描述
"o"	circle
"v"	triangle_down
"s"	square
"p"	pentagon
"*"	star
"h"	hexagon1
"+"	plus
"D"	diamond
...	...

文字

加标题：图、横轴和纵轴

F ile

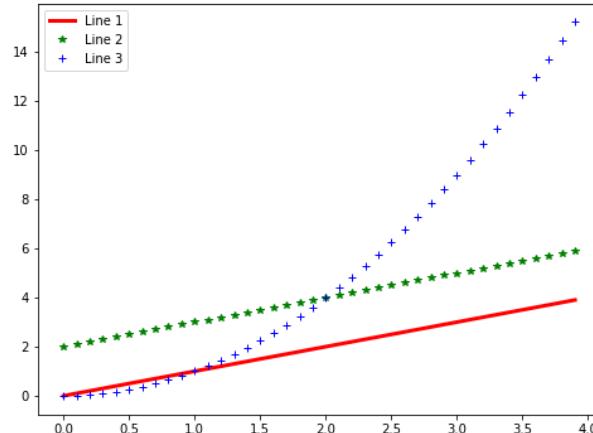
```
# Filename: plotKO.py
import matplotlib.pyplot as plt
...
x = closeMeansKO.index
y = closeMeansKO.values
plt.title('Stock Statistics of Coca-Cola')
plt.xlabel('Month')
plt.ylabel('Average Close Price')
plt.plot(x, y)
```



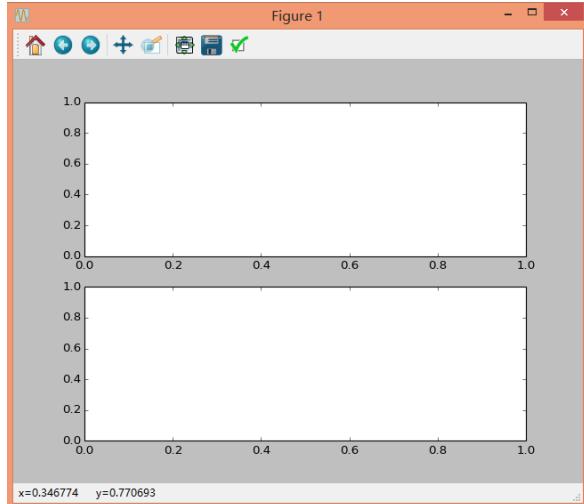
其他属性

F
ile

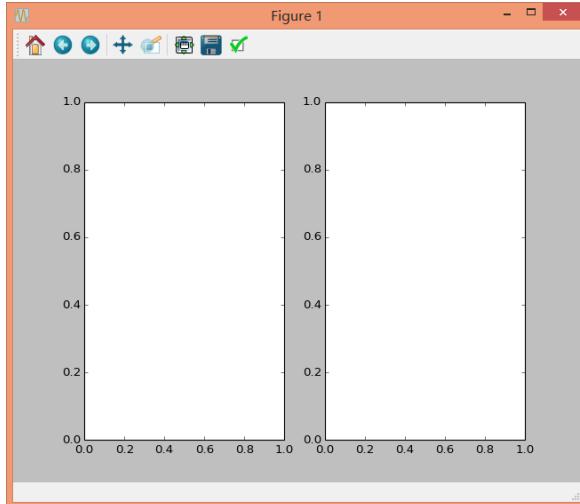
```
# Filename: multilines.py
import pylab as pl
import numpy as np
pl.figure(figsize=(8,6),dpi=100)
t=np.arange(0.,4.,0.1)
pl.plot(t,t,color='red',linestyle='-',linewidth=3,label='Line 1')
pl.plot(t,t+2,color='green',linestyle=' ',marker='*',linewidth=3,label='Line 2')
pl.plot(t,t**2,color='blue',linestyle=' ',marker='+',linewidth=3,label='Line 3')
pl.legend(loc='upper left')
```



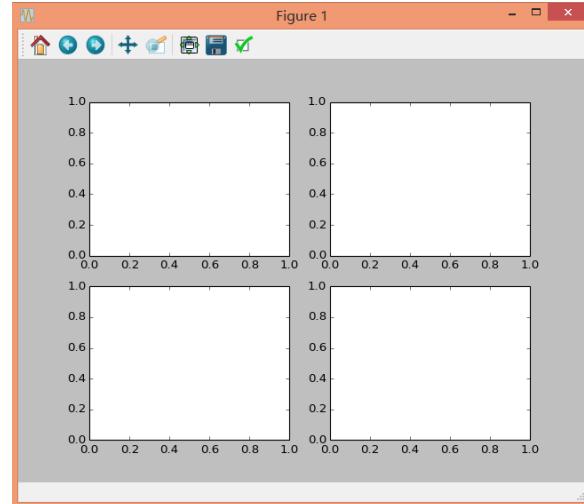
多子图-subplots



```
plt.subplot(211)  
plt.subplot(212)
```



```
plt.subplot(121)  
plt.subplot(122)
```



```
plt.subplot(221)  
plt.subplot(222)  
plt.subplot(223)  
plt.subplot(224)
```

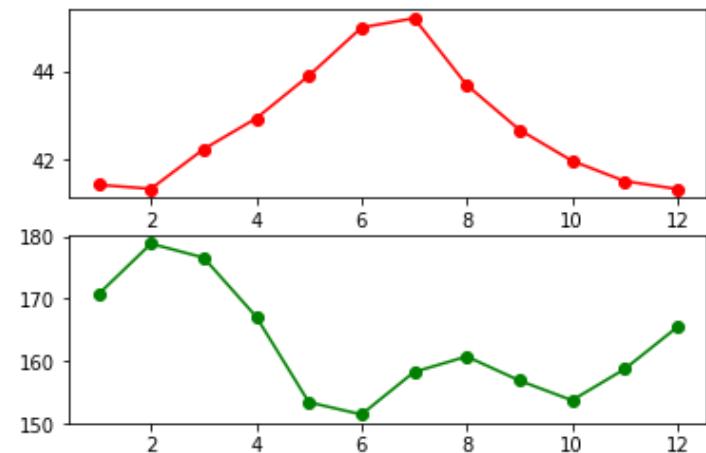
多子图-subplots



将可口可乐公司和IBM公司近一年来股票收盘价的月平均价绘制在
一张图中

F_{ile}

```
#The data of Coca-Cola and IBM is ready
plt.subplot(211)
plt.plot(x,y,color='r',marker='o')
plt.subplot(212)
plt.plot(xi,yi,color='green',marker='o')
```



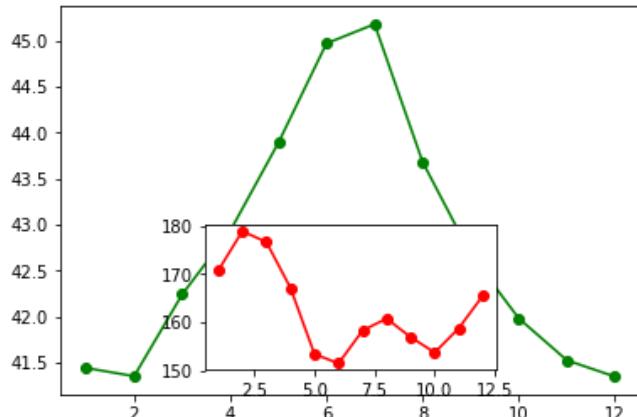
子图-axes



将可口可乐公司和IBM公司近一年来股票收盘价的月平均价绘制在
一张图中

F_{ile}

```
#The data of Coca-Cola and IBM is ready
plt.axes([.1,.1,0.8,0.8])
plt.plot(x,y,color='green',marker='o')
plt.axes([.3,.15,0.4,0.3])
plt.plot(xi,yi,color='r',marker='o')
plt.savefig('1.jpg')
```



axes([left, bottom, width, height])
参数范围为(0,1)

用Python玩转数据

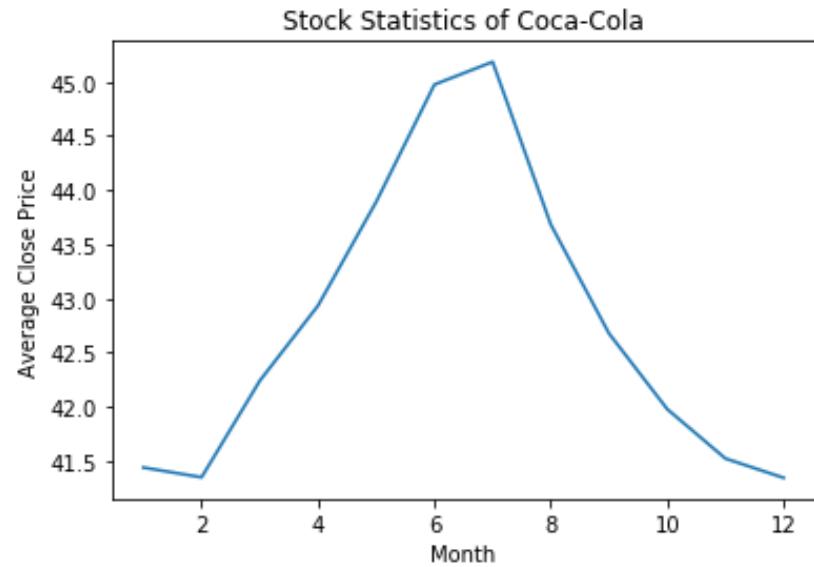
PANDAS作图

4

Python实例

S_{ource}

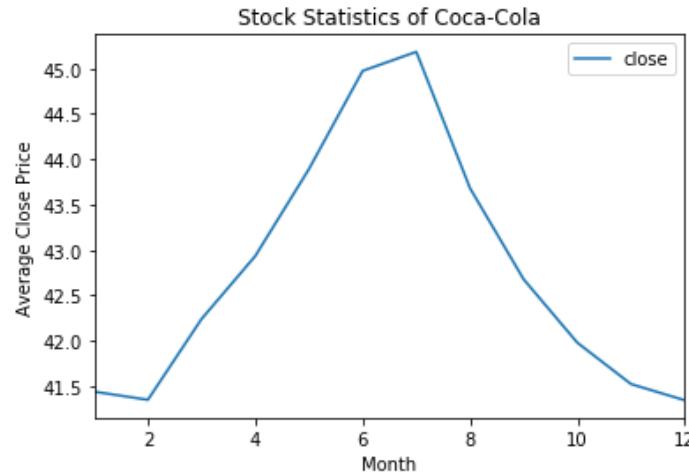
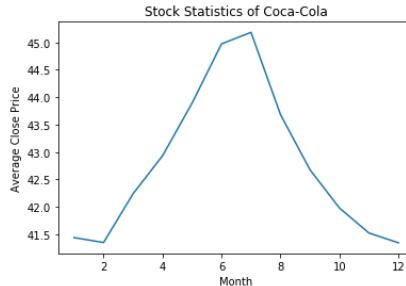
```
>>> plt.title('Stock Statistics of Coca-Cola')  
>>> plt.xlabel('Month')  
>>> plt.ylabel('Average Close Price')  
>>> plt.plot(closeMeansKO)
```



pandas绘图

S_{ource}

```
>>> import pandas as pd  
  
>>> closeMeansKO.plot()  
  
>>> plt.title('Stock Statistics of Coca-Cola')  
  
>>> plt.xlabel('Month')  
  
>>> plt.ylabel('Average Close Price')
```



pandas绘图



绘制IBM公司近一年来的股票收盘价折线图



Filename: quotesdfplot.py

...

```
quotes = retrieve_quotes_historical('IBM')
quotesdfIBM = pd.DataFrame(quotes)
quotesdfIBM.close.plot()
```



pandas控制图像形式



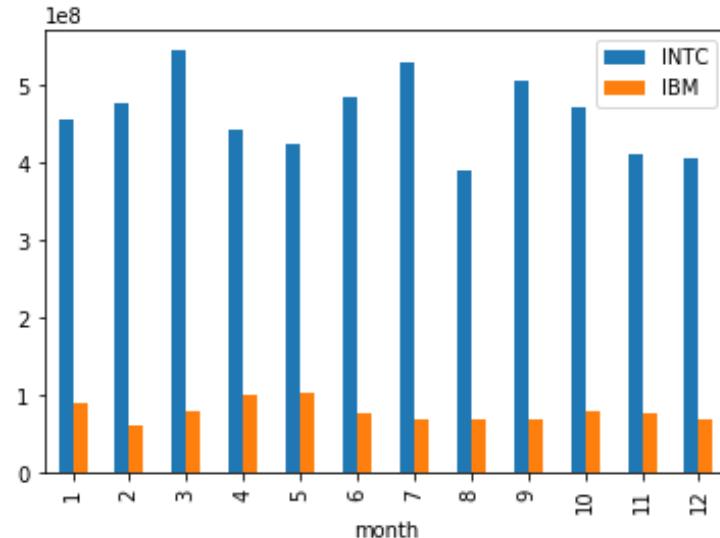
用柱状图比较Intel和IBM这两家科技公司近一年来股票成交量

File

```
# Filename: plot_volumes.py
```

```
...
```

```
INTC_volumes = create_volumes('INTC')
IBM_volumes = create_volumes('IBM')
quotesIIdf = pd.DataFrame()
quotesIIdf['INTC'] = INTC_volumes
quotesIIdf['IBM'] = IBM_volumes
quotesIIdf.plot(kind = 'bar')
```



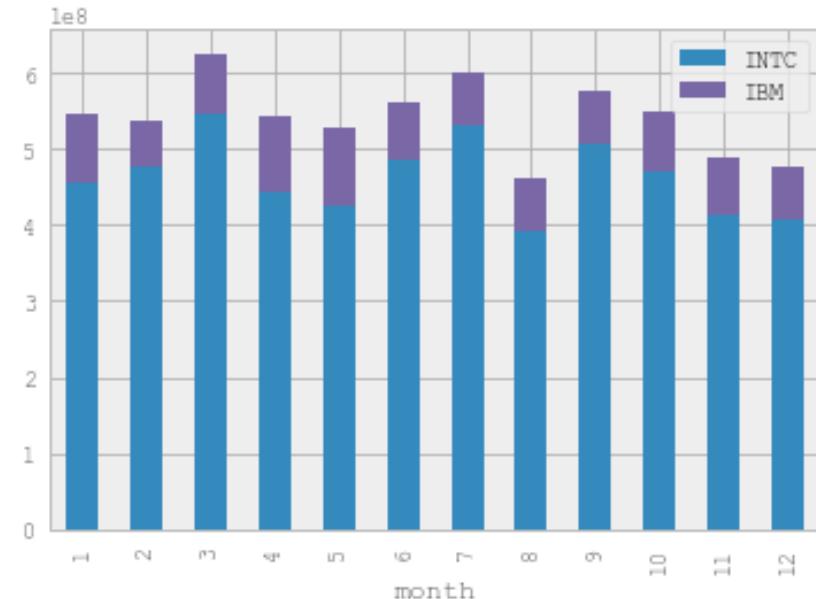
pandas控制图像形式

? 用柱状图比较Intel和IBM这两家科技公司近一年来的股票成交量

```
quotesIIdf.plot(kind='bar')
```



```
quotesIIdf.plot(kind='bar',stacked = True)
```



pandas控制图像形式

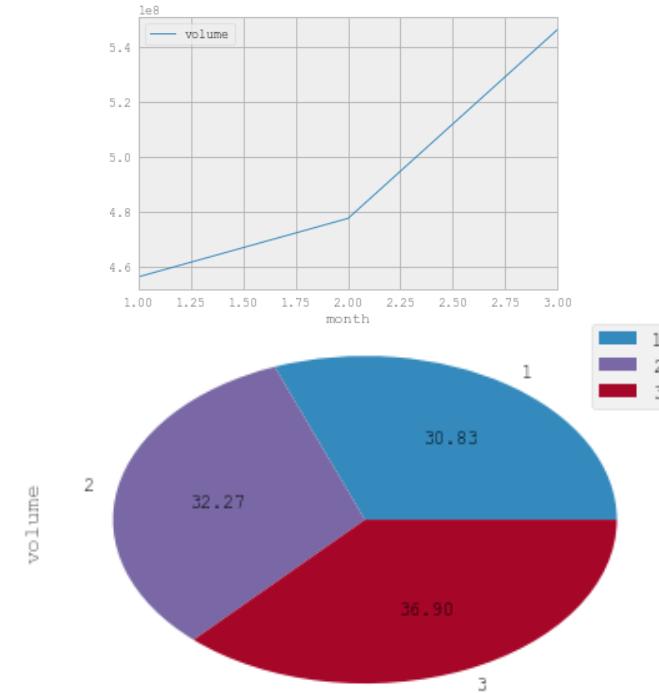


Intel公司本年度前3个月每个月股票收盘价的占比

quotesINTC.plot()



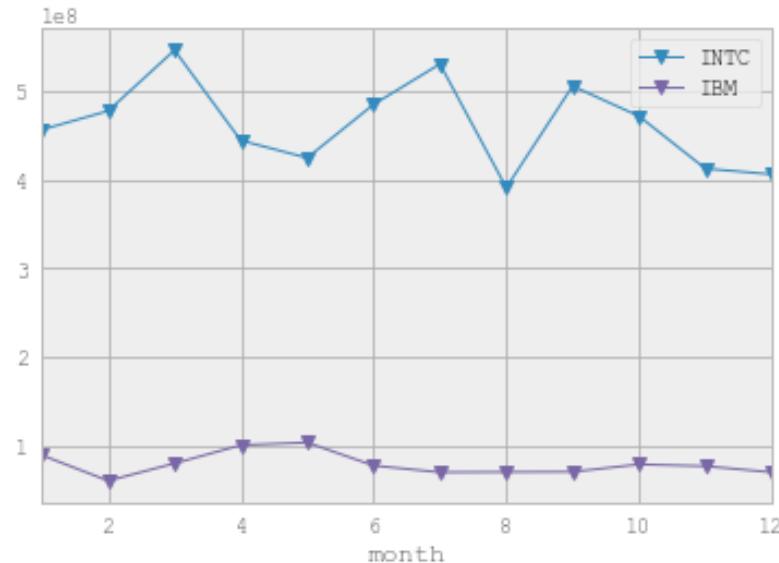
quotesINTC.plot(kind = 'pie',
subplots = True, autopct = '%.2f')



pandas控制图像属性

S_{ource}

#The data of Intel and IBM is ready
>>> quotesIldf.plot(marker='v')



箱形图

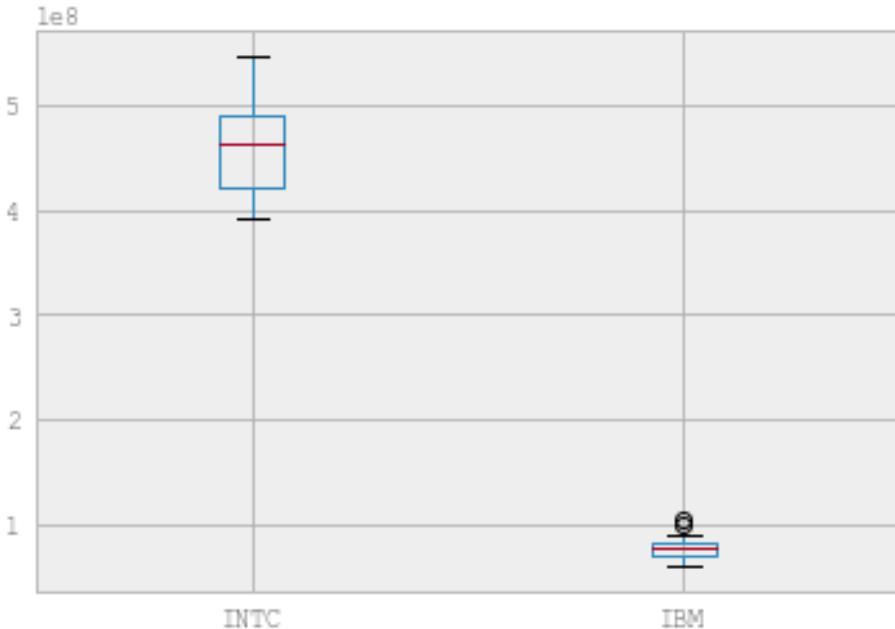


用箱形图比较Intel和IBM这两家科技公司近一年来的股票成交量

```
quotesIIdf.plot(kind='bar')
```



```
quotesIIdf.boxplot()
```



上边缘，上四分位数，中位数，下四分位数，下边缘



用Python玩转数据

5

数据存取



csv格式数据存取



将美国通用公司近一年来的股票基本信息存入文件stockAXP.csv中

File

```
# Filename: to_csv.py
import pandas as pd
...
quotes = retrieve_quotes_historical('AXP')
df = pd.DataFrame(quotes)
df.to_csv('stockAXP.csv')
```

csv格式数据存取

	A	B	C	D	E	F	G	
1		close	date	high	low	open	volume	
2	0	76.8	1495200600	77.35	76.3	76.55	3278200	
3	1	76.38	1495114200	76.85	75.97	76.27	3545700	
4	2	76.37	1495027800	78.13	76.24	78.13	4441600	
5	3	78.13	1494941400	78.64	77.84	78.6	2457500	
6	4	78.33	14948550	close, date, high, low, open, volume				
7	5	77.49	149459580	76.80000305, 1495200600, 77.34999847, 76.30000305, 76.55000305, 3278200				
8	6	77.92	149450941	76.37999725, 1495114200, 76.84999847, 75.97000122, 76.26999664, 3545700				
9	7	78.65	149442302	76.37000275, 1495027800, 78.12999725, 76.23999786, 78.12999725, 4441600				
10	8	78.44	149433663	78.12999725, 1494941400, 78.63999939, 77.83999634, 78.59999847, 2457500				
11	9	78.16	149425024	78.33000183, 1494855000, 78.62000275, 77.48000336, 77.48000336, 3327000				
12	10	78.32	149399105	77.48999786, 1494595800, 77.80999756, 77.22000122, 77.69999695, 2865800				
13	11	78.33	149390466	77.91999817, 1494509400, 78.44999695, 77.25, 78.19999695, 3780600				
				78.65000153, 1494423000, 78.66000366, 78.13999939, 78.27999878, 2396900				
				78.44000244, 1494336600, 78.73999786, 78.08999634, 78.16000366, 2570600				
				78.16000366, 1494250200, 78.73999786, 77.94999695, 78.5, 2608600				
				78.31999969, 1493991000, 78.73000336, 77.87999725, 78.61000061, 2936700				
				78.33000183, 1493904600, 79.41999817, 77.98999786, 79.23000336, 3902200				

csv格式数据存取

S_{ource}

```
>>> result = pd.read_csv('stockAXP.csv')
>>> result
   Unnamed: 0      close      date      high      low      open \
0      0  76.800003 1495200600  77.349998  76.300003  76.550003
1      1  76.379997 1495114200  76.849998  75.970001  76.269997
2      2  76.370003 1495027800  78.129997  76.239998  78.129997
3      3  78.129997 1494941400  78.639999  77.839996  78.599998
...
>>> print(result['close'])
0    76.800003
1    76.379997
2    76.370003
3    78.129997
...
```

excel数据存取

File

```
# Filename: to_excel.py  
...  
quotes = retrieve_quotes_historical('AXP')  
df = pd.DataFrame(quotes)  
df.to_excel('stockAXP.xlsx', sheet_name='AXP')
```

	close	date	high	low	open	volume
0	76.8	1495200600	77.35	76.3	76.55	3278200
1	76.38	1495114200	76.85	75.97	76.27	3545700
2	76.37	1495027800	78.13	76.24	78.13	4441600
3	78.13	1494941400	78.64	77.84	78.6	2457500
4	78.33	1494855000	78.62	77.48	77.48	3327000
5	77.49	1494595800	77.81	77.22	77.7	2865800

File

```
# Filename: read_excel.py  
...  
df = pd.read_excel('stockAXP.xlsx')  
print(df['close'][:3])
```

```
0    76.800003  
1    76.379997  
2    76.370003  
Name: close, dtype: float64
```

6

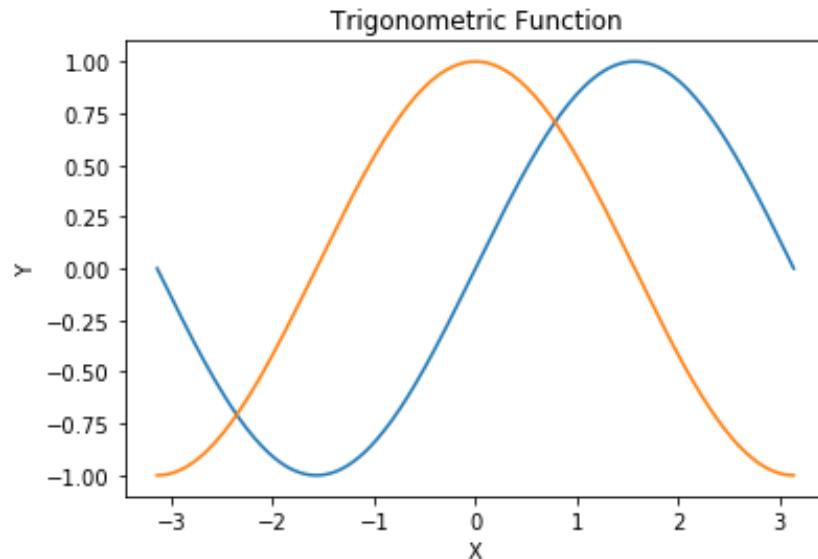
PYTHON的 理工类应用

用Python玩转数据

简单的三角函数计算

F_{ile}

```
# Filename: mathA.py
import numpy as np
import pylab as pl
x = np.linspace(-np.pi, np.pi, 256)
s = np.sin(x)
c = np.cos(x)
pl.title('Trigonometric Function')
pl.xlabel('X')
pl.ylabel('Y')
pl.plot(x,s)
pl.plot(x,c)
```

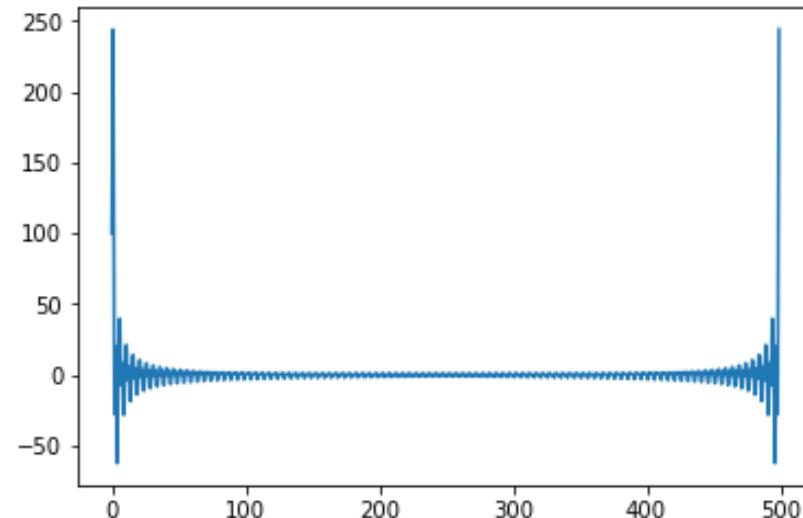


一组数据的快速傅里叶变换

数组 : [1,1,...,1,-1,-1,...,1,1,1...,1]

F ile

```
# Filename: mathB.py
import scipy as sp
import pylab as pl
listA = sp.ones(500)
listA[100:300] = -1
f = sp.fft(listA)
pl.plot(f)
```



图像处理库

- 常用Python图像处理库

- Pillow(PIL)
- OpenCV
- Skimage



File

```
# Filename: pasteimg.py
from PIL import Image
im1 = Image.open('1.jpg')
print(im1.size, im1.format, im1.mode)
Image.open('1.jpg').save('2.png')
im2 = Image.open('2.png')
size = (288, 180)
im2.thumbnail(size)
out = im2.rotate(45)
im1.paste(out, (50,50))
```

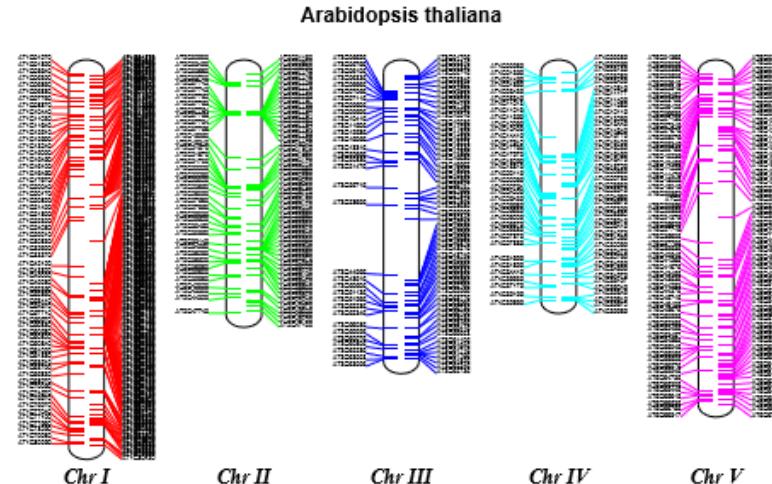
Biopython



- 来源于一个使用Python开发计算分子生物学工具的国际社团Biopython
- 序列、字母表和染色体图

S_{ource}

```
>>> from Bio.Seq import Seq
>>> my_seq = Seq("AGTACACTGGT")
>>> my_seq.alphabet
Alphabet()
>>> print(my_seq)
AGTACACTGGT
```





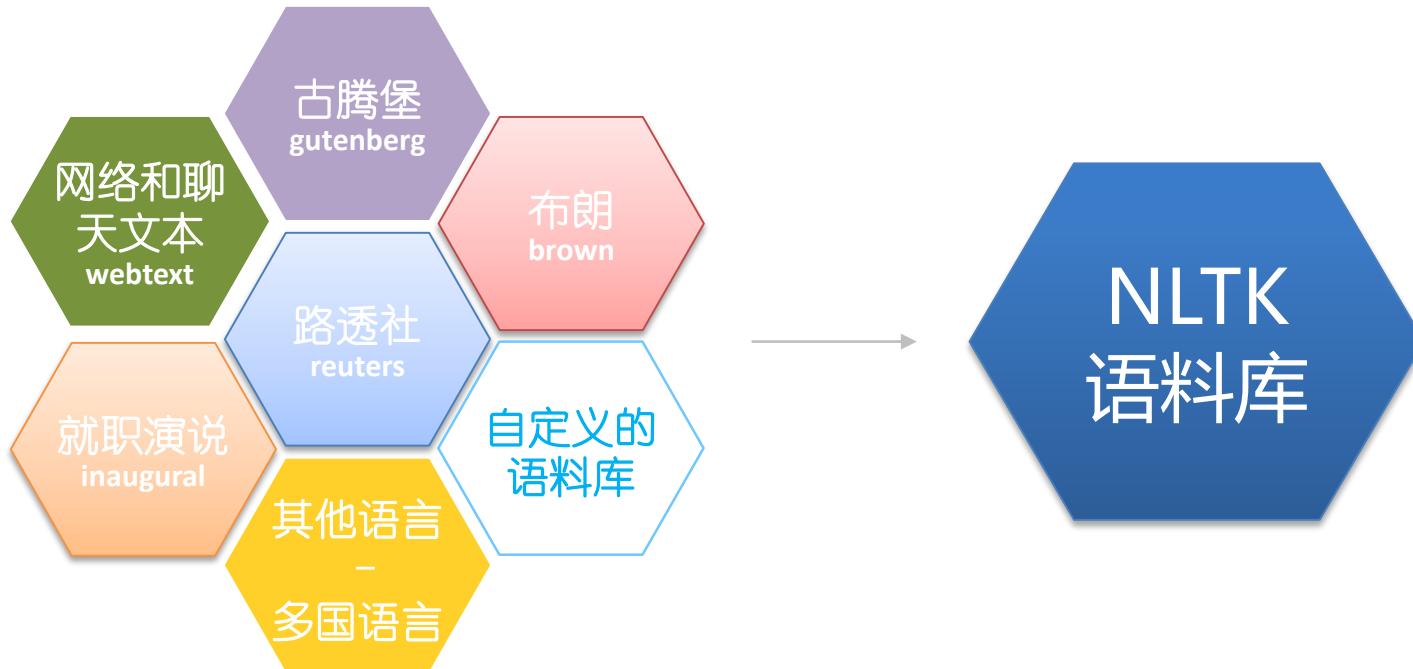
用Python玩转数据

7

PYTHON的 人文社科类应用



NLTK语料库



古滕堡项目

- 计算NLTK中目前收录的古滕堡项目的书

S
ource

```
>>> from nltk.corpus import gutenberg
>>> gutenberg.fileids()
['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', 'bible-kjv.txt', 'blake-poems.txt', 'bryant-stories.txt', 'burgess-busterbrown.txt', 'carroll-alice.txt', 'chesterton-ball.txt', 'chesterton-brown.txt', 'chesterton-thursday.txt', 'edgeworth-parents.txt', 'melville-moby_dick.txt', 'milton-paradise.txt', 'shakespeare-caesar.txt', 'shakespeare-hamlet.txt', 'shakespeare-macbeth.txt', 'whitman-leaves.txt']
```

古滕堡项目

- 一些简单的计算

S
ource

```
>>> from nltk.corpus import gutenberg
>>> allwords = gutenberg.words('shakespeare-hamlet.txt')
>>> len(allwords)
37360
>>> len(set(allwords))
5447
>>> allwords.count('Hamlet')
99
>>> A = set(allwords)
>>> longwords = [w for w in A if len(w) > 12]
>>> print(sorted(longwords))
```

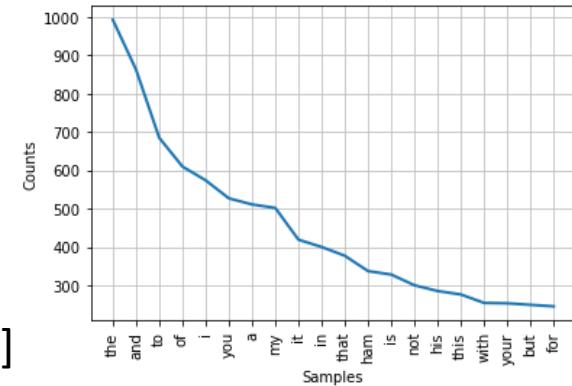
Output:

```
['Circumstances',
'Guildensterne',
'Incontinencie',
'Recognizances',
'Vnderstanding',
'determination',
'encompassement',
'entertainment',
'imperfections',
'indifferently',
'instrumentall',
'reconcilement',
'stubbornnesse',
'transformation',
'venderstanding']
```

古滕堡项目

File

```
# Filename: freqG20.py
from nltk.corpus import gutenberg
from nltk.probability import *
fd2 = FreqDist([sx.lower() for sx in allwords if sx.isalpha()])
print(fd2.B())
print(fd2.N())
fd2.tabulate(20)
fd2.plot(20)
fd2.plot(20, cumulative = True)
```



Output:

4699

30266

the and to of i you a my it in that ham
is not his this with your but for

993 863 685 610 574 527 511 502 419 400

377 337 328 300 285 276 254 253 249 245

就职演说语料库

S_{ource}

```
>>> from nltk.corpus import inaugural  
>>> from nltk.probability import *  
>>> fd3 = FreqDist([s for s in inaugural.words()])  
>>> print(fd3.freq('freedom'))  
0.00119394791917
```

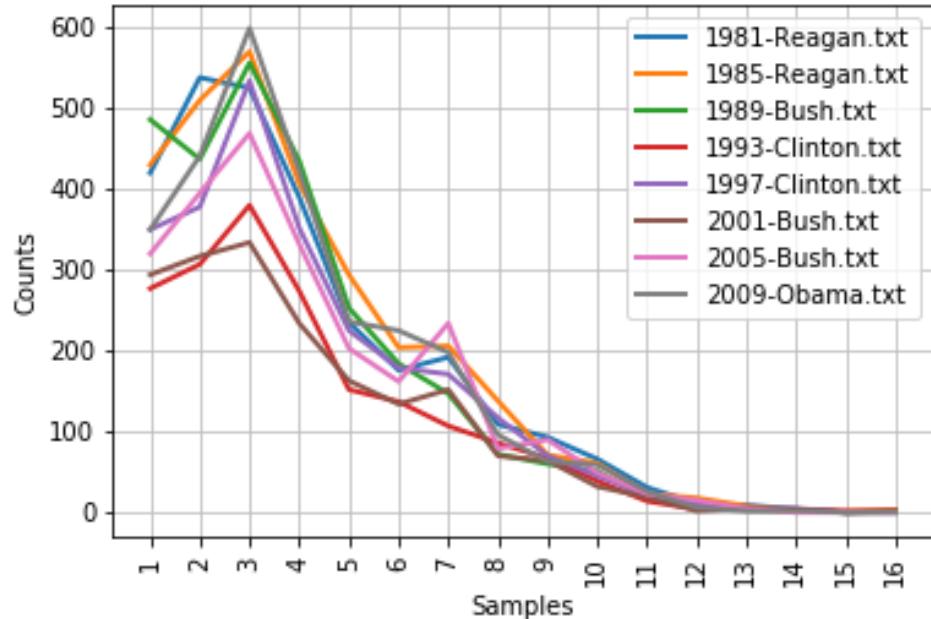
F_{ile}

```
# Filename: inaugural.py  
from nltk.corpus import inaugural  
from nltk.probability import *  
cfid = ConditionalFreqDist(  
    (fileid, len(w))  
    for fileid in inaugural.fileids()  
    for w in inaugural.words(fileid)  
    if fileid > '1980' and fileid < '2010')  
print(cfd.items())  
cfid.plot()
```

就职演说语料库

Output:

```
dict_items([('1981-Reagan.txt',  
FreqDist({2: 538, 3: 525, 1: 420, 4:  
390, 5: 235, 7: 192, 6: 176, 8: 109, 9:  
93, 10: 66, ...})), ... , ('2005-Bush.txt',  
FreqDist({3: 469, 2: 395, 4: 332, 1:  
320, 7: 234, 5: 203, 6: 162, 9: 90, 8:  
79, 10: 49, ...})), ('2009-Obama.txt',  
FreqDist({3: 599, 2: 441, 4: 422, 1:  
350, 5: 236, 6: 225, 7: 198, 8: 96, 9:  
63, 10: 59, ...}))])
```





Object-oriented & Graphical user interface

面向对象 和 图形用户界面

Department of Computer Science and Technology

Department of University Basic Computer Teaching

用Python玩转数据

1

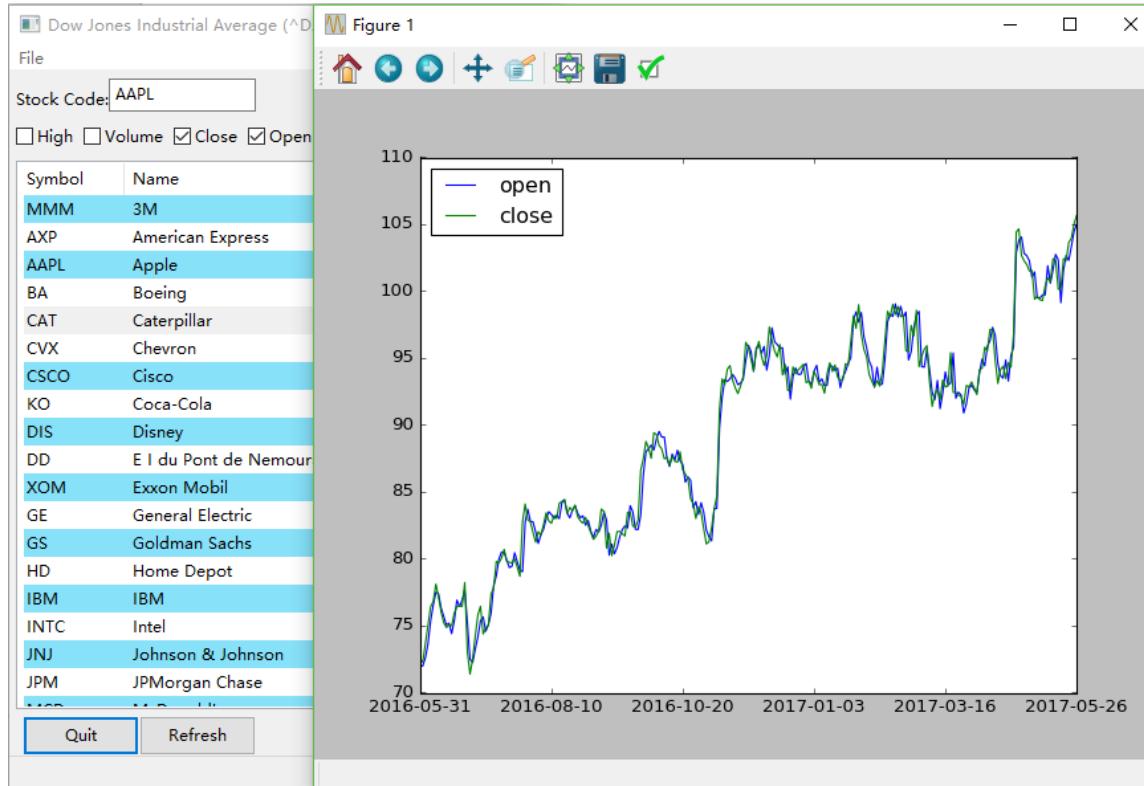
GUI与面向对象

字符用户界面

```
def foo():
    ''' add function'''
    listA = []
    print(' input the numbers: ')
    while True:
        num = input()
        if num == '.':
            break
        listA.append(eval(num))
    sumList = sum(listA)
    return sumList
```

```
>>> foo()
input the numbers:
3
5
6
7
.
21
```

图形用户界面



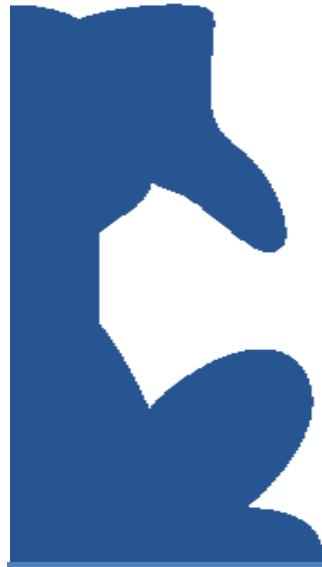
用Python玩转数据

抽象

2
3

面向对象

- 对象（实例）
 - 由数据及能对其实施的操作所构成的封装体
- 类
 - 类描述了对象的特征（数据和操作）

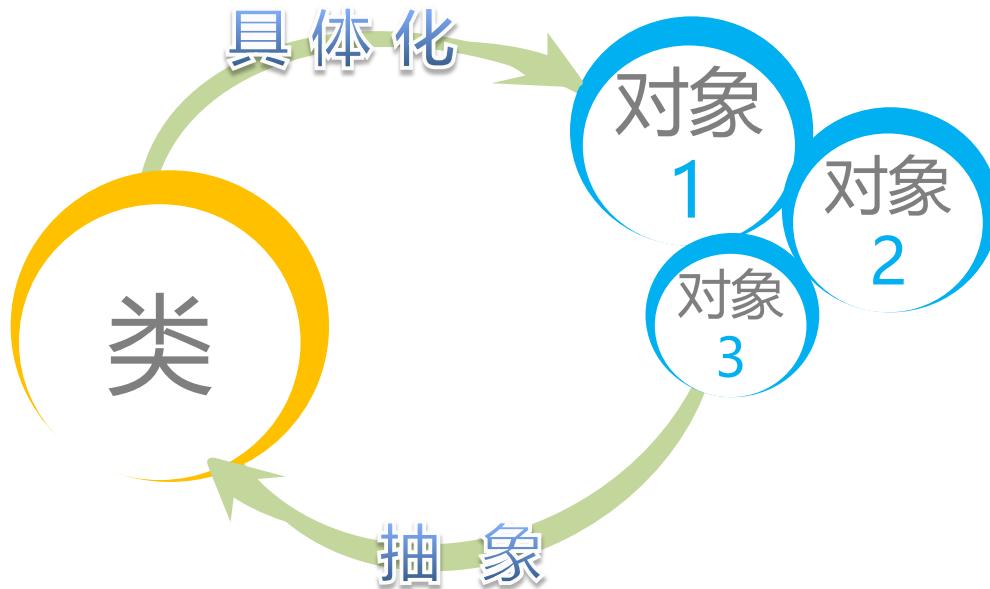




- 有名字
- 有矩形框
- 鼠标点击时有效果
- 功能不同：刷新、退出

面向对象 之 抽象

类与对象的关系



类的定义

S
ource

```
class ClassName(object):  
    'define ClassName class'  
  
class_suite
```

S
ource

```
class MyDate(object):  
    'this is a very simple  
example class'  
    pass
```

万类之源——object

类的方法

- 类的方法定义



```
>>> class Dog(object):  
    def greet(self):  
        print('Hi!')
```

实例 (Instances)

S
ource

```
>>> class Dog(object):  
    def greet(self):  
        print('Hi!')
```

S
ource

```
>>> dog = Dog()  
>>> dog.greet()
```

- 实例的创建——通过调用类对象

- ① 定义类——Dog
- ② 创建一个实例——dog
- ③ 通过实例使用属性或方法——dog.greet

实例属性 (Instance Attributes)

F ile

```
# Filename: doginsta.py
class Dog(object):
    "define Dog class"
    def setName(self, name):
        self.name = name
    def greet(self):
        print("Hi, I am called %s." % self.name)
if __name__ == '__main__':
    dog = Dog()
    dog.setName("Paul")
    dog.greet()
```

Output:

Hi, I am called Paul.

对象的初始化方法 `__init__()`

01

当类被调用后，Python将创建实例对象

02

创建完对象以后，Python自动调用的第一个方法
为`__init__()`

03

实例对象作为方法的第一个参数（`self`）被传递进
去，调用类创建实例对象时的参数都传给`__init__()`

__init__()举例

F ile

```
# Filename: doginsta.py
class Dog(object):
    "define Dog class"
    def __init__(self, name):
        self.name = name
    def greet(self):
        print("Hi, I am called %s." % self.name)
if __name__ == '__main__':
    dog = Dog("Sara")
    dog.greet()
```

Output:

Hi, I am called Sara.

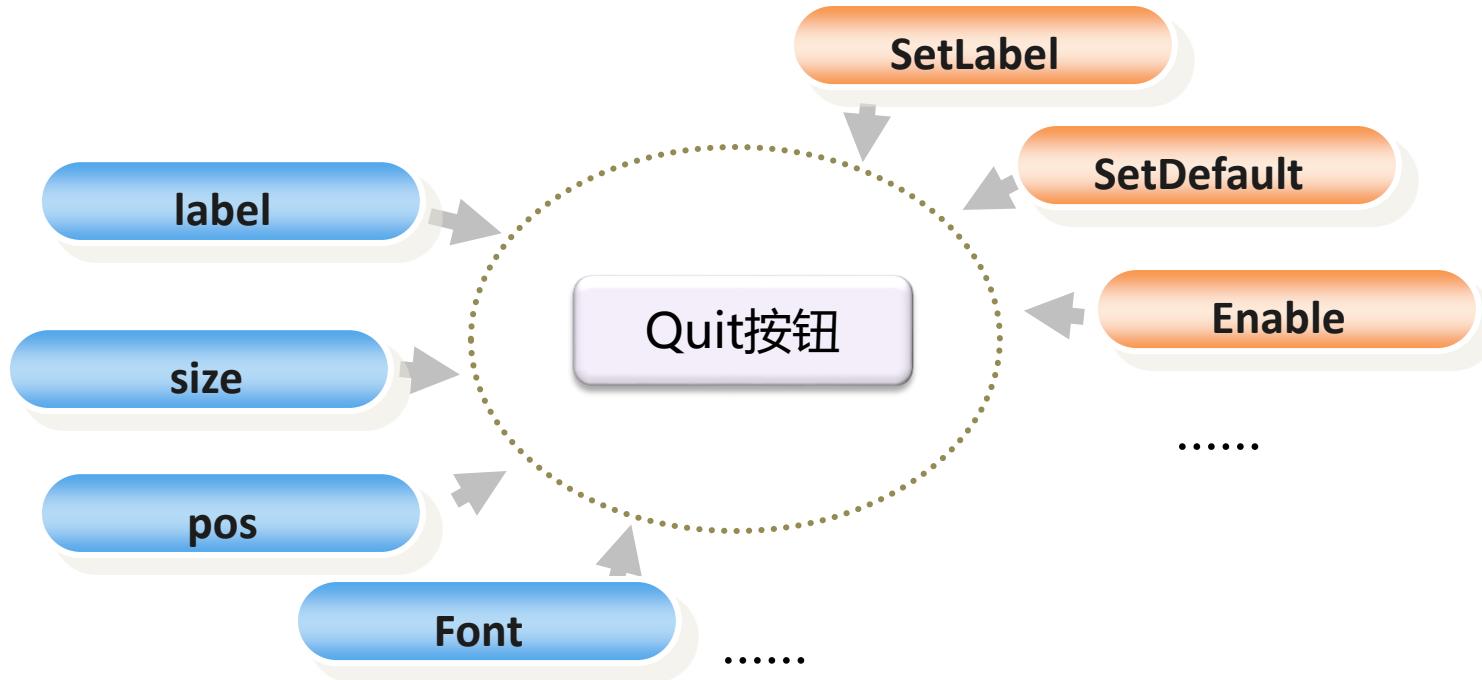
类属性 (Class Attributes)

- 类的数据属性 (静态成员)
仅仅是所定义的类的变量
- 在类创建后被使用
- 可以由类中的方法来更新 , 也
可以在主程序中更新
- 类属性和实例无关 , 修改类属
性需要使用类名

F ile

```
# Filename: doginsta.py
class Dog(object):
    "define Dog class"
    counter = 0
    def __init__(self, name):
        self.name = name
        Dog.counter += 1
    def greet(self):
        print("Hi, I am %s, my number is %d" % (self.name,
Dog.counter))
if __name__ == '__main__':
    dog = Dog("Zara")
    dog.greet()
```

以按钮 (Button) 为例

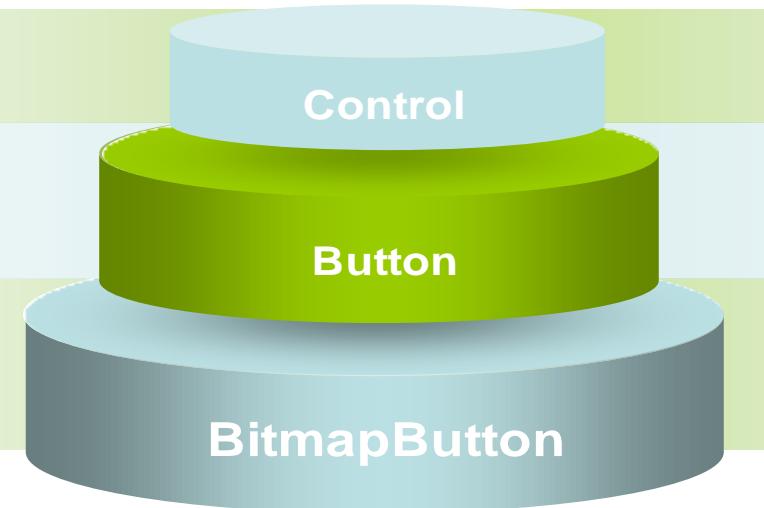
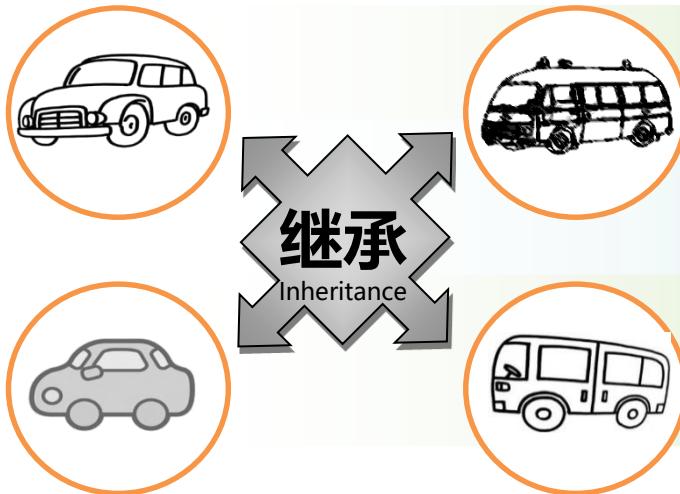


用Python玩转数据

3

继承

父类（基类）子类（派生类）



子类的定义

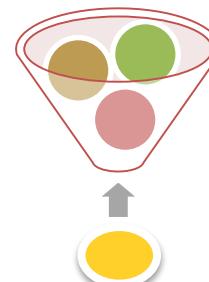
S_{ource}

```
class SubClassName (ParentClass1[, ParentClass2, ...]):  
    'optional class documentation string'  
    class_suite
```

单
继承



多
继承



子类定义举例和重载

F ile

```
# Filename: doginsta.py
class Dog(object):
    "define Dog class"
    counter = 0
    def __init__(self, name):
        self.name = name
        Dog.counter += 1
    def greet(self):
        print("Hi, I am %s, my number is %d" %
              (self.name, Dog.counter))
```

F ile

```
# Filename: overridepro.py
class BarkingDog(Dog):
    "define subclass BarkingDog"
    def greet(self):
        "initial subclass"
        print("Woof! I am %s, my number is
              %d" % (self.name, Dog.counter))
    if __name__ == '__main__':
        dog = BarkingDog("Zoe")
        dog.greet()
```

私有属性和方法

- 默认情况下，Python 类的成员属性与方法都是 “public”
- 提供 “访问控制符” 来限定成员函数的访问
 - 双下划线(_)
_var属性会被_classname_var替换，将防止父类与子类中的同名冲突
 - 单下划线(_)
在属性名前使用一个单下划线字符，防止模块的属性用“from mymodule import *” 来加载

用Python玩转数据

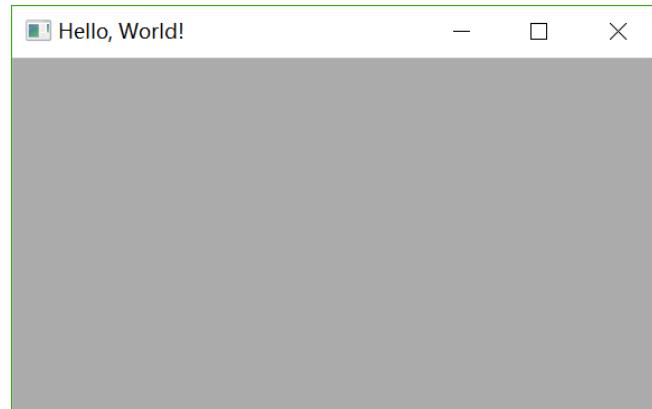
GUI的基本框架

4

创建一个简单的wxPython程序



```
# Filename: firstwxPython.py
import wx
app = wx.App()
frame = wx.Frame(None, title = "Hello, World!")
frame.Show(True)
app.MainLoop()
```



上例也可改为：

```
# Filename: mouse.py
import wx

class MyApp(wx.App):
    def OnInit(self):
        frame = wx.Frame(None, title = "Hello, World!")
        frame.Show()
        return True
    if __name__ == '__main__':
        app = MyApp()
        app.MainLoop()
```

应用程序对象也可以是
wx.App的子类的一个实例

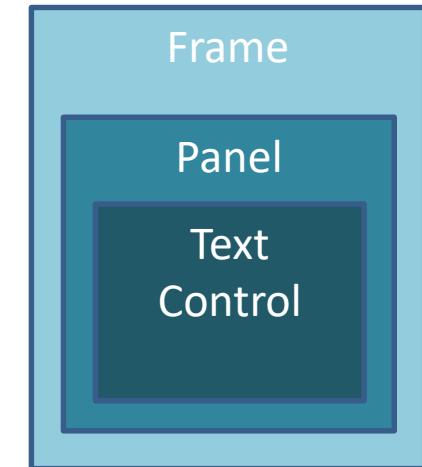
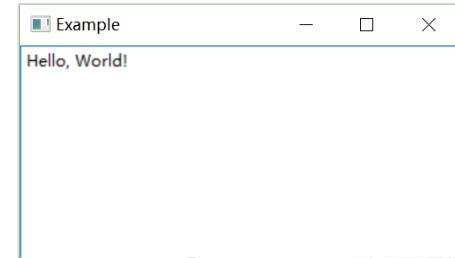
组件

- 组件容器 (Containers) ——用于容纳其他组件
 - 例 : wx.Panel 等
- 动态组件 (Dynamic Widgets) ——可以被用户编辑
 - 例 : wx.Button、wx.TextCtrl、wx.ListBox 等
- 静态组件 (Static Widgets) ——显示信息用，不能被用户编辑
 - 例 : wx.StaticBitmap、wx.StaticText、wx.StaticLine 等
- 其它组件
 - 例 : wxToolBar、wxMenuBar、wxStatusBar

又见 “Hello , World ! ”



```
# Filename: helloworld.py
import wx
class Frame1(wx.Frame):
    def __init__(self, superior):
        wx.Frame.__init__(self, parent = superior, title = "Example", pos =
(100,200), size= (350,200))
        panel = wx.Panel(self)
        text1= wx.TextCtrl(panel, value = "Hello, World!", size = (350,200))
    if __name__ == '__main__':
        app =wx.App()
        frame = Frame1(None)
        frame.Show(True)
        app.MainLoop()
```



事件处理机制(Event Handling)

- GUI程序工作的基本机制之一——事件处理
- 事件
 - 移动鼠标，按下鼠标左键、单击按钮等
 - 可以由用户操作触发产生，也可以在程序中创建对象产生
- wxPython程序将特定类型的事件关联到特定的一块代码（方法），当该类型的事件产生时，相关代码将响应事件被自动执行
 - 例：当产生鼠标移动事件时，OnMove()方法将被自动调用

还是Hello, World!

F
ile

```
# Filename: mouse.py
import wx
class Frame1(wx.Frame):
    def __init__(self,superior):
        ...
        self.panel.Bind(wx.EVT_LEFT_UP, self.OnClick)
    def OnClick(self, event):
        posm = eventGetPosition()
        wx.StaticText(parent = self.panel,label = "Hello, World!",pos = (posm.x, posm.y))
..... #create app and frame, show and execute event loop
```

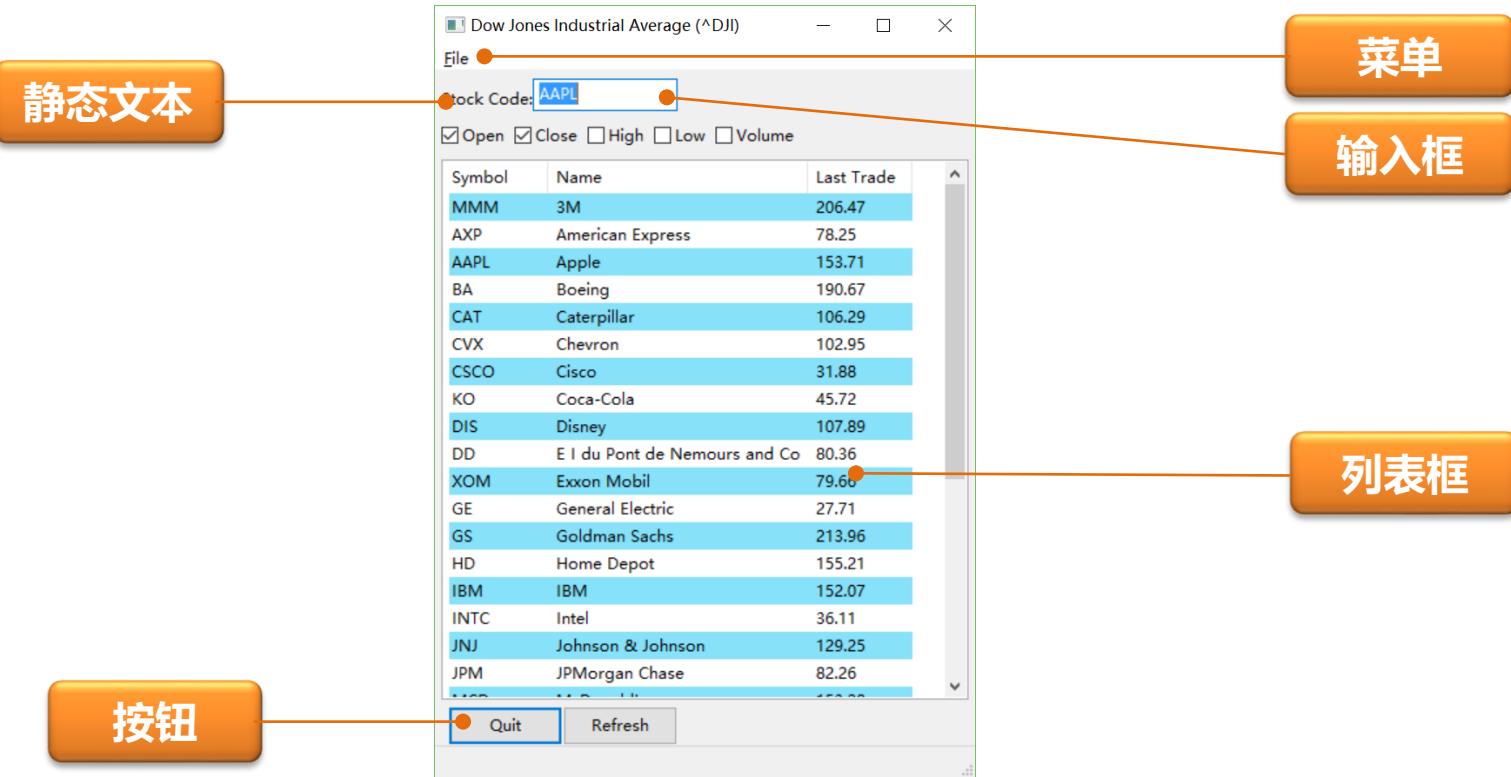


用Python玩转数据

5

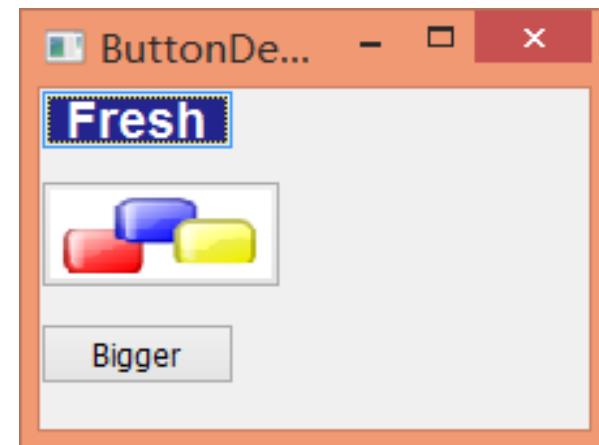
GUI常用组件

应用程序示例



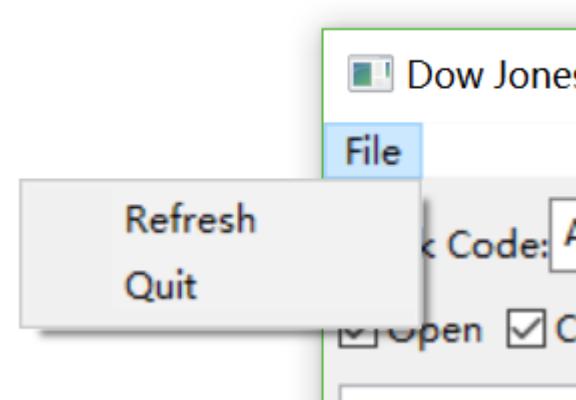
按钮 (Button及其家族)

- 功能：接受用户的点击事件，触发相应的操作。
- 常用按钮：
 - wx.Button : 文本按钮
 - wx.BitmapButton : 位图按钮
 - wx.ToggleButton : 开关按钮
- 绑定处理按钮点击的事件



菜单 (Menu及其组件)

- 菜单
 - 菜单栏
 - 菜单
 - 菜单项命令
- wxPython用于创建菜单的类：
 - wx.MenuBar
 - wx.Menu
 - wx.MenuItem



菜单常用事件

- 菜单事件
 - wx.EVT_MENU

File

```
# Filename: menudemo.py
```

...

```
#绑定事件处理器
```

```
self.Bind(wx.EVT_MENU,self.OnClickBigger,biggerItem)  
self.Bind(wx.EVT_MENU,self.OnClickQuit,id=wx.ID_EXIT)
```

...

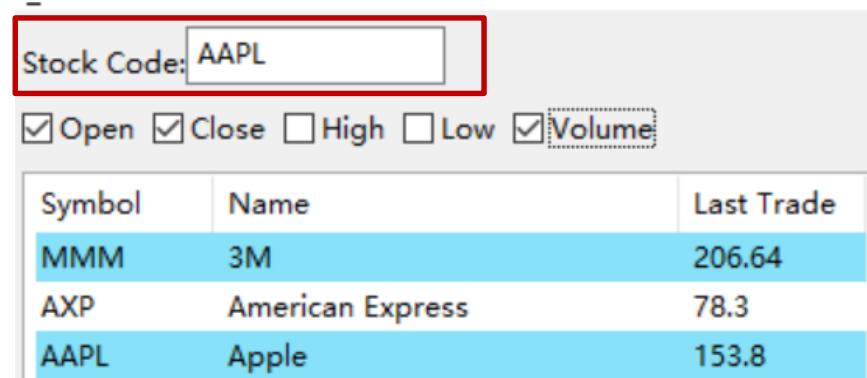
```
#事件处理器
```

```
def OnClickBigger(self,e):  
    pass  
def OnClickQuit(self,e):  
    self.Close()
```

...

静态文本 (StaticText) 和文本框 (TextCtrl)

- 文本框用于接收用户在框内输入的信息，或显示由程序提供的信息
- 静态文本框（标签）：
 - 类：wx.StaticText
- 文本框：
 - 类：wx.TextCtrl
 - 常用形式：单行,多行,富文本框



列表 (ListCtrl)

- 列表用于显示多个条目并且可供用户选择
- 列表能够以下面四种不同模式建造：
 - wx.LC_ICON (图标)
 - wx.LC_SMALL_ICON (小图标)
 - wx.LC_LIST (列表)
 - wx.LC_REPORT (报告)

Col #1	Col #2	Col #3
Row #1	aaaa	1
Row #2	bbbb	2
Row #3	cccc	3
Row #4	dddd	4
Row #5	eeee	5

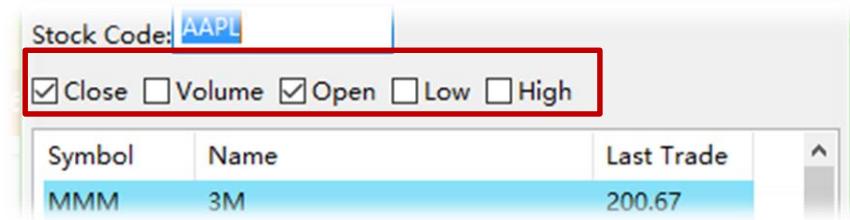
Report 模式

Row #1
Row #2
Row #3
Row #4
Row #5

List模式

单选 (RadioBox) 与复选框 (CheckBox)

- 复选框用于从一组可选项中，同时选中多个选项
- 对应的，单选框用于从一组互斥的选项中，选取其一



程序示例

```
# Filename: helloworldbtn.py
import wx
class Frame1(wx.Frame):
    def __init__(self, superior):
        wx.Frame.__init__(self, parent = superior, title = "Hello World in wxPython")
        panel = wx.Panel(self)
        sizer = wx.BoxSizer(wx.VERTICAL)
        self.text1 = wx.TextCtrl(panel, value = "Hello, World!", size = (200,180), style = wx.TE_MULTILINE)
        sizer.Add(self.text1, 0, wx.ALIGN_TOP | wx.EXPAND)
        button = wx.Button(panel, label = "Click Me")
        sizer.Add(button)
        panel.SetSizerAndFit(sizer)
        panel.Layout()
        self.Bind(wx.EVT_BUTTON, self.OnClick, button)
    def OnClick(self, text):
        self.text1.AppendText("\nHello, World!")
```

6

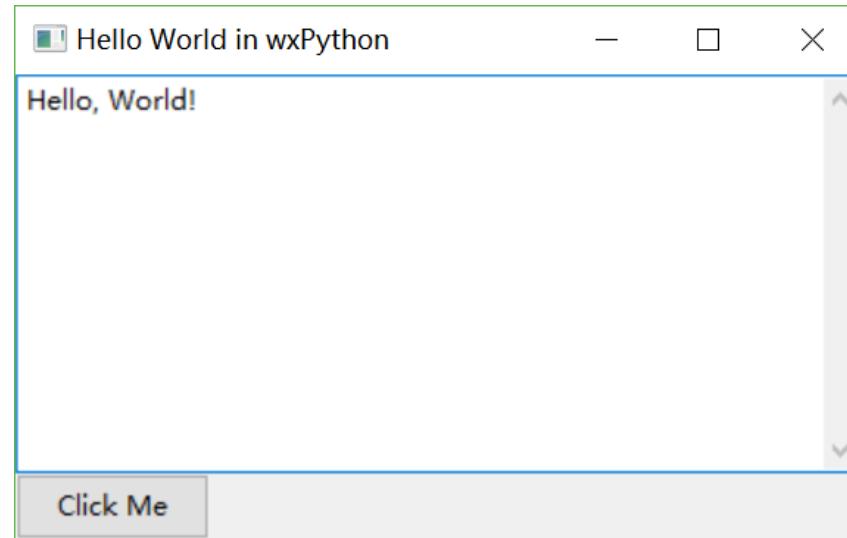
用Python玩转数据 布局管理

布局管理

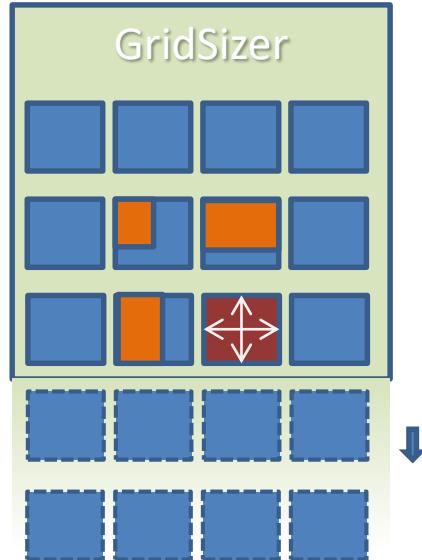
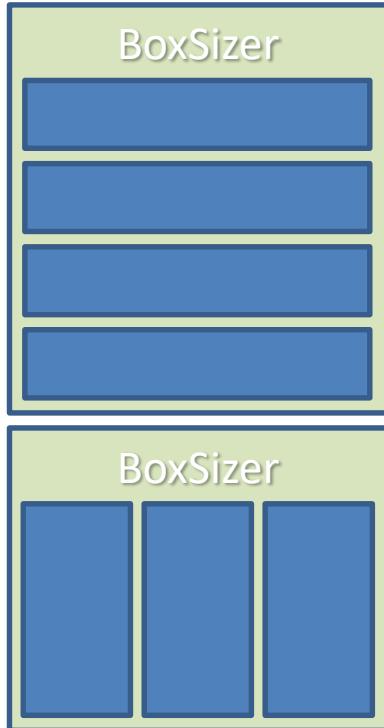
- 绝对定位 - 每个窗口组件被创建时可以显式地指定它的位置和大小
 - 缺点：定位不灵活
 - 调整大小困难
 - 受设备、操作系统甚至字体影响
- 灵活布局的解决方案 - sizer
 - 每个sizer有自己的定位策略
 - 开发者只需要选择合适策略的sizer将窗口组件放入，并且指定好需求即可

sizer

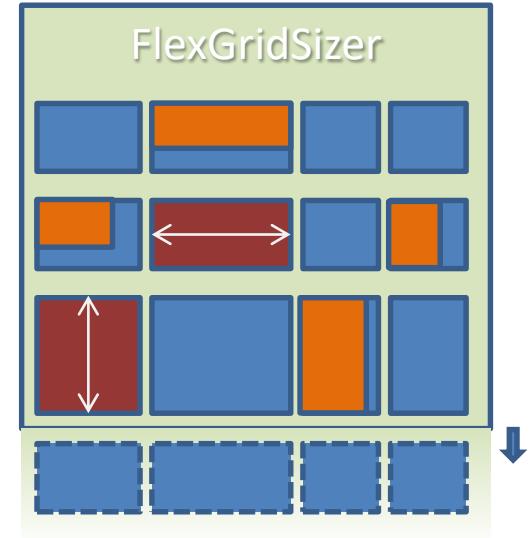
- sizer本身不是一个容器或一个窗口组件。它只是一个屏幕布局的算法
- sizer允许嵌套
- wxPython常用的sizer
 - wx.BoxSizer
 - wx.FlexGridSizer
 - wx.GridSizer
 - wx.GridBagSizer
 - wx.StaticBoxSizer



各种sizer示意



所有组件大小一致，
固定一个方向，在
另外一个方向生长



行高和列宽由最大的
组件决定

使用sizer的步骤

01

创建自动调用尺寸的容器，例如panel

02

创建sizer

03

创建子窗口（窗体组件）

04

使用sizer的Add()方法将每个子窗口添加给sizer

05

调用容器的SetSizer(sizer)方法

示例程序

```
# Filename: helloworldbtn.py
import wx
class Frame1(wx.Frame):
    def __init__(self,superior):
        wx.Frame.__init__(self, parent = superior, title = "Hello World in wxPython")
        panel = wx.Panel(self)
        sizer = wx.BoxSizer(wx.VERTICAL)
        self.text1= wx.TextCtrl(panel, value = "Hello, World!", size = (200,180), style = wx.TE_MULTILINE)
        sizer.Add(self.text1, 0, wx.ALIGN_TOP | wx.EXPAND)
        button = wx.Button(panel, label = "Click Me")
        sizer.Add(button)
        panel.SetSizerAndFit(sizer)
        panel.Layout()
        self.Bind(wx.EVT_BUTTON,self.OnClick,button)
    def OnClick(self, text):
        self.text1.AppendText("\nHello, World!")
```

用Python玩转数据

其他GUI库



Python的GUI实现



wxPython

开源软件，具有优秀的
跨平台能力。官网：

<http://wxpython.org>

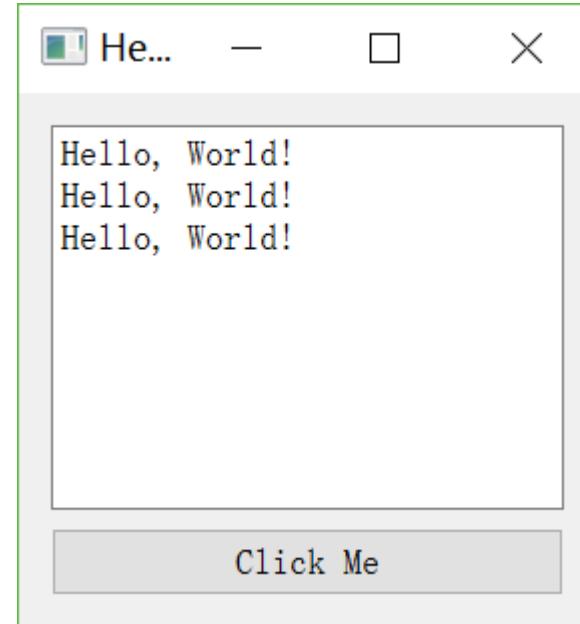
PyQt

- 是Python语言的GUI编程解决方案之一
- 提供了GPL与商业协议两种授权方式，可以免费地用于自由软件的开发
- 跨平台：可以运行于Microsoft Windows、Mac OS X、Linux以及其他类Unix平台上

PyQt简单示例

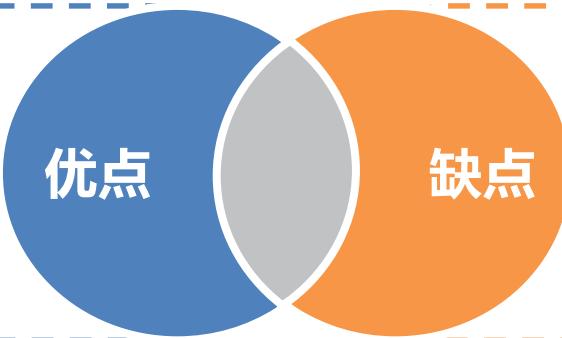
File

```
# Filename: PyQtdemo.py
import sys
from PyQt5 import QtWidgets
class TestWidget(QtWidgets.QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Hello World!")
        self.outputArea=QtWidgets.QTextBrowser()
        self.helloButton=QtWidgets.QPushButton("Click Me")
        self.layout=QtWidgets.QVBoxLayout()
        self.layout.addWidget(self.outputArea)
        self.layout.addWidget(self.helloButton)
        self.setLayout(self.layout)
        self.helloButton.clicked.connect(self.sayHello)
    def sayHello(self):
        self.outputArea.append("Hello, World!")
if __name__ == '__main__':
    app=QtWidgets.QApplication(sys.argv)
    testWidget=TestWidget()
    testWidget.show()
    sys.exit(app.exec_())
```



PyQT的优缺点

- 文档比其他GUI库丰富
- 与Qt、C++开发经验互通
- 可使用大多数为Qt开发的组件
- 有方便的周边工具支持PyQt，如QtDesigner, Eric4



- 要注意避免内存泄露
- 运行时庞大
- 需要学习一些C++知识

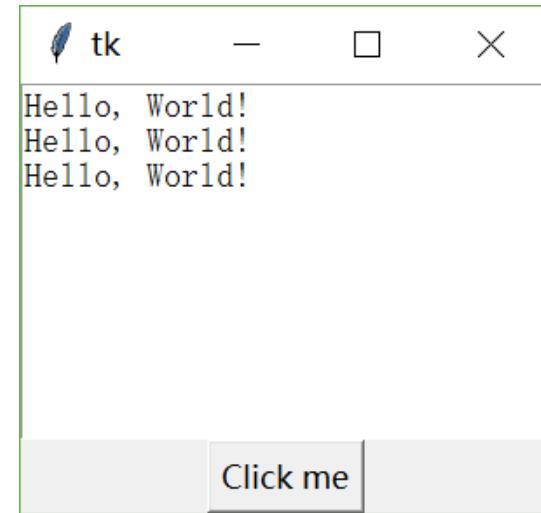
Tkinter

- Tkinter 绑定了 Python 的 Tk GUI 工具集，通过内嵌在 Python 解释器内部的 Tcl 解释器实现
- Tkinter 的调用转换成 Tcl 命令，然后交给 Tcl 解释器进行解释，实现 Python 的 GUI 界面

Tkinter简单示例

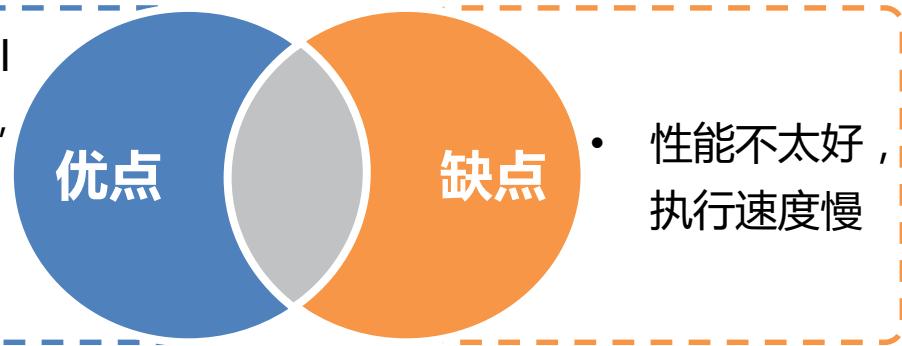
File

```
# Filename: Tkinterdemo.py
import tkinter as tk
class Tkdemo(object):
    def __init__(self):
        self.root=tk.Tk()
        self.txt=tk.Text(self.root,width=30,height=10)
        self.txt.pack()
        self.button=tk.Button(self.root,text='Click me',
                             command=self.sayhello)
        self.button.pack()
    def sayhello(self):
        self.txt.insert(tk.INSERT,"Hello, World!\n")
d=Tkdemo()
d.root.mainloop()
```



Tkinter的优缺点

- 历史最悠久，是Python 事实上的标准 GUI
 - Python 中使用 Tk GUI 工具集的标准接口，已包括在标准的Python Windows 安装中
 - 著名的 IDLE 用 Tkinter 实现 GUI
- 创建的 GUI 简单，学起来和用起来也简单



PyGTK

- PyGTK是一套 GTK+ GUI 库的Python封装
- pyGTK为创建桌面程序提供了一套综合的图形元素和其它使用的编程工具
- PyGTK是基于LGPL协议的免费软件
- 许多 Gnome 下的著名应用程序的 GUI 都是使用 PyGTK 实现的，比如 BitTorrent , GIMP 和 Gedit 都有可选的实现

PyGTK的简单示例



#PyGTKdemo.py

import pygtk

pygtk.require('2.0')

import gtk

class HelloWorld:

def hello(self, widget, data=None):

textbuffer = self.textview.get_buffer()

startiter, enditer = textbuffer.get_bounds()

content_text = textbuffer.get_text(startiter, enditer)

content_text += "Hello, World!\n"

textbuffer.set_text(content_text)

def __init__(self):

self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)

self.window.set_title("A Simple Example of PyGtk")

self.window.connect("delete_event", self.delete_event)

self.window.connect("destroy", self.destroy)

self.window.set_border_width(10)

box1 = gtk.VBox(False, 0)

self.window.add(box1)

```

box1.show()
sw = gtk.ScrolledWindow()
sw.set_policy(gtk.POLICY_AUTOMATIC,
gtk.POLICY_AUTOMATIC)
self.textview = gtk.TextView()
textbuffer = self.textview.get_buffer()
sw.add(self.textview)
sw.show()
self.textview.show()
box1.pack_start(sw)

self.button = gtk.Button("Click Me")
self.button.connect("clicked", self.hello, None)
self.button.show()
box1.pack_start(self.button, expand=False, fill=False)
self.window.show()

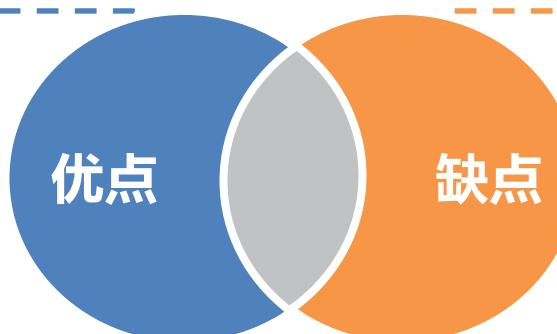
def main(self):
    gtk.main()

if __name__ == "__main__":
    hello = HelloWorld()
    hello.main()

```



PyGTK的优缺点

- 底层的GTK+提供了各式的可视元素和功能
 - 能开发在GNOME桌面系统运行的功能完整的软件
- 
- 在Windows平台表现不太好

8

用Python玩转数据 综合应用

图形用户界面

