


# Chapter 2 – Python基本操作

Frank Ziwei Zhang  
School of Finance



上海對外經貿大學  
SHANGHAI UNIVERSITY OF INTERNATIONAL BUSINESS AND ECONOMICS

# Contents



Q1	变量赋值
Q2	数据类型
Q3	数据结构
Q4	运算符号
Q5	内置函数
Q6	自定义函数
Q7	Python 语句

## 2.1 变量赋值

## 2.1 Python变量赋值

在Python中每个变量在使用前都必须赋值，赋值方法如下所示（左边是一个变量名，右边是存储在该变量中的值）：

变量=值

Python的变量名可以用英文字母数字和下划线构成，但是必须记住以下四点注意事项：

- 1、可以单独使用英文字母或下划线作为变量名，同时英文字母需要注意区分大小写；
- 2、数字不能单独用于表示变量，比如3不能作为变量名；
- 3、变量不能以数字开头，比如1a不能作为变量名a1却可以作为变量名；
- 4、变量的命名应该简洁易懂，比如在金融领域对于变量命名，尽可能运用该变量的英文名英文缩写，或者英文名的首字母。

## 2.1 Python变量赋值

【例2-1】假定利率是8%，可以采用如下代码对利率变量进行赋值：

```
rate=0.08#利率等于8%
```

或者：

```
r=0.08
```

以符号#开始输入的内容，属于注释语句，Python是不会读取的。

## 2.1 Python变量赋值

一些约定成俗的程序员标准：

1、变量命名推荐使用蛇形体，即一律小写，如有多个单词，用下划线隔开：

`student_number=30`      # 蛇形体是一种比较常见的变量命名方式

当然，也有很多人喜欢使用驼峰体。但是，考虑到金融数据分析中常常遇到和另一个常用软件stata的交互，我们并不推荐：

`studentNumberPython=30`    #这种方式就是大名鼎鼎的驼峰体(CamelCase)

2、函数命名一般用驼峰体，即首字母小写，如有多个单词，则其他单词首字母大写，无下划线连接：

`def runWithFunc ():`

私有函数，我们习惯于在函数名前加一个my\_

`def myPrivateFunc():`

3、常量（全局变量）采用全大写，如有多个单词，使用下划线隔开：

`MAX_CLIENT = 100`

`MAX_CONNECTION = 1000`

4、类名一般使用首字母大写的驼峰命名风格：

`class Farm():`              #这种命名也被称为大驼峰体

或者：

`class AnimalFarm(Farm):`

## 2.2 Python的数据类型

## 2.2 主要数据类型





## 2.2.1 整型 int/long

【例2-2】假定A投资者持有一家上市公司100股股票，通过 Python进行赋值，并且通过type函数判断数据类型，具体的代码如下：

```
share=100  
type(share)
```

```
Out[3]: int
```

## 2.2.2 浮点型

【例2-3】运用前面例22中的相关信息，在 Python中输入数字时，用户在100后面增加了一个小数点，则数据类型结果就是浮点型而非整型，具体的代码如下：

```
Share=100.  
type(share)
```

```
Out[6]:float
```

【例2-4】一个变量`a`等于复数`2+3i`，在 Python 中进行输入，并且判断数据类型，具体的代码如下：

```
a=2+3j  
type(a)
```

```
Out[7]: complex
```

特征1: 输入时需用引号

在Python中，用英文的单引号''、双引号""以及三引号"""来标识字符串。其中，三引号往往是用于多行文本。此外，引号内无论中文、英文字母、符号、数字甚至是空格，均被视为字符串。

【例2-5】在Python中，以字符串的数据类型依次输入finance、risk management、金融风险管理、888、1+1，具体的代码如下：

```
a="finance"
```

```
type(a)
```

```
Out[8]:str
```

```
e="1+1"
```

```
type(e)
```

```
Out[12]:str
```

### 特征2：可索引性

字符串是可以被索引的，并且如果针对字符串是从左向右的索引，则默认是从0开始，最大范围就是字符串长度减去1，并且需要用到中括号；相反，从右向左的索引，默认是从-1开始，最大范围是字符串开头。

**【例2-6】**运用例2-5中输入的字符串finance，依次索引finance这个单词的首字母f、单词的第4个字母a以及单词最后一个字母e，具体的代码如下：

```
a="finance"#索引首字母
```

```
a[0]
```

```
Out[14]:'f'
```

```
a[3]#索引第四个字母
```

```
Out[15]:'a'
```

```
a[-1]#索引最后一个字母
```

```
Out[16]:'e'
```

### 特征3：可截取性

假如用户要从字符串“I love risk”选取子字符串“isk”，就需要用到字符串的截取功能。截取也称为切片（slice），是从一个字符串中获取子字符串（也就是字符串的一部分），需要使用中括号、起始索引值（start）、终止索引值（end）以及可选的步长（step）来定义。截取的输入格式如下：字符串[start:end:step]

**【例2-7】**假定用户在Python中输入字符串“I love risk”，并且在该字符串中截取子字符串“love”，具体的代码如下

```
x="I love risk"  
x[2:6]#截取字符串love  
Out[17]:'love'
```

**【例2-10】**假定用户在Python中输入字符串“I love risk management”，并且希望从该字符串中截取从第3位至第9位并且步长为3（也就是跳2格选取）的子字符串，具体的代码如下：

```
x="I love risk management"  
x[2:9:3]  
Out[20]:'lei'
```

### 特征4：可替换性

如果用户发现在Python输入字符串中的部分子字符串出现错误这时用户就可以运用replace函数实现修正和替换，下面就通过一道例题进行演示。

**【例2-11】**假定用户在事后检查中发现，原本希望在Python中输入“I love finance”，但是却误写成了“I love management”，此时运用replace函数进行修改，具体的代码如下：

```
y="I love management"#错误的更正  
y.replace("management","finance")
```

```
Out[21]:'I love finance'
```

## 2.3 Python的数据结构



数据结构类型	简介和特征	特征性标识	金融领域运用
元组 (tuple)	一种高级的数据结构，可索引，但不可修改	用小括号 ( ) 标识	较少运用
列表 (list)	与元组类似，除了可索引以外，更重要的是可以修改，能够理解为元组的升级版，而元组则是稳定版的列表	用中括号 [] 标识	经常运用
集合 (set)	类似于数学的集合，每个集合中的元素具有无序性、不重复性的特征	用大括号 {} 标识	很少运用
字典 (dict)	与日常生活中的字典用法很类似，通过“名称（键 <b>key</b> ）--内容（值 <b>value</b> ）来构建	用大括号 {} 标识	较少运用
数组 (array)	科学计算和代数运算常用的数据类型，类似于线性代数的向量和矩阵	用 <b>array</b> ，小括号，中括号共同标识	经常运用
数据框 (dataframe)	数据分析处理常用的数据类型，带有索引 ( <b>index</b> ) 和列名 ( <b>column</b> )，类似于Excel的工作表	用 <b>DataFrame</b> 、小括号，中括号共同标识	经常运用

【例2-12】在 Python 中，创建一个空元组，具体的代码如下：

```
tup1=()
type(tup1)
Out[22]: tuple
```

需要注意的是，如果元组中只包含一个元素时，需要在该元素后面添加一个逗号，否则就无法构成一个元组。

【例2-13】在 Python 中，创建一个仅包含一个元素3的元组，具体的代码如下：

```
tup2=(3,) #在元素后面加上一个逗号
type(tup2)
Out[23]: tuple
```

```
tup3=(3) #没有在元素后面加一个逗号
type(tup3)
Out[25]: int
```

## 2.3.1 元组

【例2-14】在 Python中创建一个元组，该元组包含的元素包括 finance、风险管理2019、8.88，相关的代码如下：

```
tup4=('finance','风险管理',2019,8.88)
```

```
type(tup4)
```

```
Out[26]: tuple
```

元组一旦创建以后，元组中的元素是不可修改的，只能进行访问。访问的方式与前面介绍的字符串索引是非常类似的。

【例2-15】沿用例2-14中的信息，分别访问元组的第1个元素、最后一个元素以及同时访问第2个和第3个元素，相关的代码如下：

```
tup4[0]
```

```
Out[27]: 'finance'
```

```
tup4[-1]      #访问元祖最后一个元素
```

```
Out[28]: 8.88
```

```
tup4[1:3]      #访问元祖第2个至第3个元素——注意这个较为搞笑的规则，  
索引的时候，包括位置1，不包括位置3
```

```
Out[29]: ('风险管理', 2019)
```

## 2.3.2 列表

	不同点	相同点
元组	每个元素都是不可修改的，用小括号标识	1.可以容纳Python的任何对象 2.元素是有序的，即每个元素都有一个索引值
列表	每个元素是可修改的，用中括号标识	

【例2-16】在 Python中创建一个空列表，相关的代码如下：

```
list1=[]  
type(list1)  
Out[30]: list
```

【例2-17】在 Python中创建一个列表，该列表包含的元素包括 finance、risk management、金融风险管理、2020、8.88，具体的代码如下：

```
list2=['finance','risk management',"金融风险管理",2020,8.88]
```

```
type(list2)
```

```
Out[32]: list
```

访问列表与访问元组的方式是类似的。

【例2-18】对例2-17中创建的列表进行访问，分别访问该列表中的第一个元素、最后一个元素以及从第3个至第4个元素，相关的代码如下：

```
list2[0]
```

```
Out[34]: 'finance'
```

```
list2[-1]
```

```
Out[35]: 8.88
```

```
list2[2:4]
```

```
Out[36]: ['金融风险管理', 2020]
```

对于已有列表，在新增元素时可以运用 `append`函数，并且新元素是添加到列表的结尾。

**【例2-20】**在例2-16创建的空列表中添加新的元素，新的元素是2018年1月15日至19日这一个交易周沪深300指数（本书第8章的8.1节将介绍主要的股票指数）每日的涨跌幅，分别是-0.54%、0.77%、0.24%、0.87%以及0.38%，具体的代码如下：

```
list1.append(-0.0054)
list1.append(0.0077)
list1.append(0.0024)
list1.append(0.0087)
list1.append(0.0038)
list1
Out[38]: [-0.0054, 0.0077, 0.0024, 0.0087, 0.0038]
```

对列表中元素的删除分为两类：第一类是删除指定的元素，运用 `remove` 函数；第二类是删除列表中全部的元素，也就是清空列表，需要运用 `clear` 函数。

【例2-21】 对例2-20的列表`list1`，删除列表中第3个元素0.024，具体的代码如下：

```
list1.remove(0.0024)
```

```
list1
```

```
Out[39]: [-0.0054, 0.0077, 0.0087, 0.0038]
```

需要注意的是，如果在一个列表中有多个相同值的元素`x`，则 `remove(x)` 是删除列表中值为`x`的第一个元素，而不是全部`x`的元素。

【例2-22】 针对包括2、4、6、8、10、2、4、2元素的列表，删除列表中数值是2的第1个元素，具体的代码如下：

```
list3=[2,4,6,8,10,2,4,2]
```

```
list3.remove(2)
```

```
list3
```

```
Out[40]: [4, 6, 8, 10, 2, 4, 2]
```

## 2.3.2 列表

针对列表的指定位置插入元素，就需要运用`insert`函数，该函数需要输入两个参数：第1个参数是位置参数，相当于索引值；第2个参数就是需要插入的元素值。

**【例2-24】**针对例2-21中的列表`list1`，在列表第3个元素的位置重新插入元素0.0024，具体的代码如下：

```
list1.insert(2,0.0024)
```

```
list1
```

```
Out[42]: [-0.0054, 0.0077, 0.0024, 0.0087, 0.0038]
```



排序的问题，由小到大排序需要运用sort函数，由大到小排序则运用 reverse 函数。

【例2-25】针对例2-24的列表list1，分别按照由小到大、由大到小进行排序，具体操的代码如下：

```
list1.sort()          #从小到大排序  
list1  
Out[43]: [-0.0054, 0.0024, 0.0038, 0.0077, 0.0087]
```

```
list1.reverse()       #由大到小排序  
list1  
Out[44]: [0.0087, 0.0077, 0.0038, 0.0024, -0.0054]
```

计算该元素出现的次数，需要用到count函数

【例2-26】假定有一个列表:[1,2,3,1,5,1,6,9,1,2,7]，需要计算该列表中，数字1和2出现的次数，具体的代码如下：

```
list4=[1,2,3,1,5,1,6,2,9,1,2,7]
```

```
list4.count(1)
```

```
Out[45]: 4
```

```
list4.count(2)    #计算列表中数字2出现的次数
```

```
Out[46]: 3
```

## 2.3.3 集合

Python中，集合（set）的概念接近数学上的集合。每个集合中的元素是无序且不重复的，因此就可以通过集合去判断数据的从属关系，有时还可以通过集合把数据结构中重复的元素过滤掉。

集合的创建方式如下：

变量={元素1，元素2，元素3，...}

需要注意的是，集合不可以被截取，也不能被索引，只能包括并集、差集、交集等集合运算，同时，集合元素可以被添加和删除。

**【例2-27】** 分别创建两个集合，一个集合包含上证综指、深圳成指、恒生指数、日经225指数、道琼斯指数等元素，另一个集合则包含标普500指数、道琼斯指数、沪深300指数、日经225指数、法国CAC40指数、德国DAX指数等元素，具体的代码如下：

```
set1={"上证综指","深圳成指","恒生指数","日经225指数","道琼斯指数"}  
type(set1)  
Out[1]: set
```

```
set2={"标普500指数","道琼斯指数","沪深300指数","日经225指数","法国CAC指数","德国DAX指数"}  
type(set2)  
Out[2]: set
```

## 2.3.3 集合

针对集合求并集时，需要运用到符号“|”，下面来看一个例题。

【例2-28】针对例2-27中创建的两个集合，求这两个集合的并集，具体的代码如下：

```
set1|set2  
Out[3]:  
{'上证综指',  
'德国DAX指数',  
'恒生指数',  
'日经225指数',  
'标普500指数',  
'沪深300指数',  
'法国CAC指数',  
'深圳成指',  
'道琼斯指数'}
```

针对集合求交集时，需要运用到符号“&”或者运用 Intersection函数。

【例2-29】针对例2-27中创建的两个集合，求这两个集合的交集，具体的代码如下：

```
set1&set2  
Out[4]: {'日经225指数', '道琼斯指数'}
```

针对集合求差集时，需要运用到数学中的减号“-”

【例2-30】针对例227中创建的两个集合，分别求set1对set2的差集、set2对set1差集，具体的代码如下：

```
set1-set2          #set1对set2的差集  
Out[6]: {'上证综指', '恒生指数', '深圳成指'}
```

```
set2-set1          #set2对set1的差集  
Out[7]: {'德国DAX指数', '标普500指数', '沪深300指数', '法国CAC指数'}
```

用户可以在已经创建的集合中添加新的元素，需要运用add函数，并且输出的结果可能会自行排列。

【例2-31】针对例2-27中创建的集合set1，在集合中增加元素“德国DAX指数”，具体的代码如下：

```
set1.add('德国DAX指数')  
set1  
Out[9]: {'上证综指', '德国DAX指数', '恒生指数', '日经225指数', '深圳成指', '道琼斯指数'}
```

## 2.3.4 字典

指数名称	证券代码	交易日期	涨跌幅
沪深300	000300	2019-01-08	0.22%

字典的形式如下：

变量={键1： 值1， 键2： 值2， 键3： 值3， .....}

需要注意的是，字典有3个特征：

一是字典中的元素必须以键（**key**）和值（ **value**）的形式成对出现，也就是所谓的键-值存储；

二是键不可以重复，但是值可以重复；

三是键不可修改，但是值可以修改，并且数值可以是任意的数据类型。

字典的创建可以采用两种不同的方式：一是直接法，就是一次输入全部的键与值；二是间接法，也就是先创建一个空字典，然后逐对输入键与值。

【例2-33】将表2-3中的信息以字典的形式在 Python中输入，并且分别运用直接法和间接法进行输入，具体的代码如下：

```
dict1={'指数名称':'沪深300','证券代码':'000300','交易日期':'2019-01-08','涨跌幅':-0.0022} #直接法创建
```

```
dict1
```

```
Out[12]: {'指数名称': '沪深300', '证券代码': '000300', '交易日期': '2019-01-08', '涨跌幅': -0.0022}
```

```
type(dict1)
```

```
Out[13]: dict
```

```
dict2={}          #间接法创建
```

```
dict2["指数名称"]="沪深300"
```

```
dict2["证券代码"]="000300"
```

```
dict2["交易日期"]="2019-01-08"
```

```
dict2["涨跌幅"]=-0.0022
```

```
dict2
```

```
Out[20]: {'指数名称': '沪深300', '证券代码': '000300', '交易日期': '2019-01-08', '涨跌幅': -0.0022}
```

【例2-34】针对例2-33中创建的字典，访问并输出字典中的全部键和值，具体的代码如下：

```
dict1.keys()
```

```
Out[22]: dict_keys(['指数名称', '证券代码', '交易日期', '涨跌幅'])
```

```
dict1.values()
```

```
Out[23]: dict_values(['沪深300', '000300', '2019-01-08', -0.0022])
```

也可以通过 `Items` 遍历字典的全部元素，也就是将字典中的每个元素（即每对键与值）组成一个元组并放在列表中输出。

【例2-35】针对例2-33中创建的字典，遍历字典的全部元素，具体的代码如下：

```
dict1.items()
```

```
Out[24]: dict_items([('指数名称', '沪深300'), ('证券代码', '000300'), ('交易日期', '2019-01-08'), ('涨跌幅', -0.0022)])
```

如果仅仅是查询某个键对应的值，可以直接通过在中括号内输入键码的方式完成。

【例2-36】针对例2-33中创建的字典，查找并输出涨跌幅对应的具体金额，具体的代码如下：

```
dict1["涨跌幅"]    #注意是用中括号
```

```
Out[25]: -0.0022
```



【例2-37】针对例2-33中创建的字典，用户希望将字典中的“交易日期”对应的“2019-01-08”修改为“2019-01-07”，涨跌幅对应的-0.22%相应调整为0.61%，具体的代码如下：

```
dict1["交易日期"]="2019-01-07"
```

```
dict1["涨跌幅"]="0.0061"
```

```
dict1
```

```
Out[26]: {'指数名称': '沪深300', '证券代码': '000300', '交易日期': '2019-01-07', '涨跌幅': '0.0061'}
```

如果在已经创建的字典中，新增加键与值，可以运用 update 函数。

【例2-38】针对例2-33中创建的字典，增加当日的收盘价3054.3以及成交额1057.04亿元的信息，具体的代码如下：

```
dict1.update({"收盘价":3054.3,"成交额(亿元)":1057.04})
```

```
dict1
```

```
Out[27]:
```

```
{'指数名称': '沪深300',  
'证券代码': '000300',  
'交易日期': '2019-01-07',  
'涨跌幅': '0.0061',  
'收盘价': 3054.3,  
'成交额(亿元)': 1057.04}
```

## 2.4 Python运算符

## 2.4.1 基本符号

运算符号	描述	示例	在金融领域的运用情况
+	加法	$1+1\rightarrow2$	经常使用
-	减法	$1-1\rightarrow0$	
*	乘法	$1*2\rightarrow2$	
/	除法	$1/2\rightarrow0.5$	
**	幂运算	$2**3\rightarrow8$	
%	模运算（取余数）	$5\%2\rightarrow1$	较少使用
//	整除（商的整数部分）	$9//4\rightarrow2$	

## 2.4.2 关系符号

运算符	描述	Python的示例	在金融领域的运用情况
==	相等	In : 2==2 Out: True In : 2==3 Out: False	经常使用
!=	不等于	In : 2!=3 Out: True In : 2!=2 Out: False	
>	大于	In : 2>1 Out: True In : 2>3 Out: False	
>=	大于等于	In : 2>=-1 Out: True In : 2>=-3 Out: False	
<	小于	In : 2<3 Out: True In : 2<1 Out: False	
<=	小于等于	In : 2<=3 Out: True In : 2<=1 Out: False	

## 2.4.3 赋值符号

运算符	描述	举例	Python的示例	在金融领域的运用情况
+=	加法赋值运算符，等价于 $y=y+x$ 注：相当于对等式左边的变量 $y$ 进行新的赋值（下同）	$y+=x$	In : $x1=3$ In : $y1=10$ In : $y1+=x1$ In : $y1$ Out: 13	经常使用
-=	减法赋值运算符，等价于 $y=y-x$	$y-=x$	In : $x2=5$ In : $y2=12$ In : $y2-=x2$ In : $y2$ Out: 7	
*=	乘法赋值运算符，等价于 $y=y*x$	$y*=x$	In : $x2=5$ In : $y2=12$ In : $y2*=x2$ In : $y2$ Out: 60	有时使用
/=	除法赋值运算符，等价于 $y=y/x$	$y/=x$	In : $x2=5$ In : $y2=12$ In : $y2/=x2$ In : $y2$ Out: 2.4	
**=	幂赋值运算符，等价于 $y=y**x$	$y**x$	In : $x2=5$ In : $y2=12$ In : $y2**=x2$ In : $y2$ Out: 79.6263	
%=	模赋值运算符，等价于 $y=y\%x$	$y\%=x$	In : $x2=5$ In : $y2=12$ In : $y2\%=x2$ In : $y2$ Out: 4.6262	较少使用
//=	整除赋值运算符，等价于 $y=y//x$	$y//x$	In : $x2=5$ In : $y2=12$ In : $y2//=x2$ In : $y2$ Out: 0	

## 2.4.4 成员运算符

成员运算符	具体描述
<b>in</b>	如果一个变量在指定的另一个变量（列表、元组、字符串等）中找到相应的值，则返回 <b>True</b> ，否则返回 <b>False</b>
<b>not in</b>	如果一个变量在指定的另一个变量中没有找到相应的值，返回 <b>True</b> ，否则返回 <b>False</b>

## 2.4.4 成员运算符

【例2-48】在 Python中考察相应的数字是否在一个列表中，相关的代码如下：

```
a=1  
b=3  
c=[1,2,4,8,16]  
a in c  
Out[59]: True
```

```
b in c  
Out[60]: False
```

【例2-49】在 Python中考察相应的字符串是否在一个列表中，相关的代码如下：

```
d='金融'  
e='风险管理'  
f=['finance','风险管理','波动率']  
d in f  
Out[61]: False
```

```
e in f  
Out[62]: True
```

## 2.5 Python主要内置函数



## 2.5 主要内置函数

函数名称	功能	Python代码示例
<b>abs</b>	求绝对值 (abs 是绝对值英文单词 absolute 的缩写)	In : a=-5 In : abs (a) Out: 5
<b>enumerate</b>	将对象 (如列表、元组或字符串) 组合为一个带有索引的序列, 一般用于for循环中	In : stock=['A股', 'B股', 'H股', 's股'] In : list (enumerate (stock,start=1)) Out: [(1, 'A股'), (2, 'B股'), (3, 'H股'), (4, 's股')] 注意: 要输出结果需要用到list函数, start=1代表1是索引的起始值, 也可以任意设定索引起始值
<b>float</b>	数字或字符串转换为浮点型	整型转化为浮点型的举例 In : b=6 In : float (6) Out: 6.0 字符串转化为浮点型的举例 In : c='28' In : float (c) Out: 28.0
<b>int</b>	数字或字符串转换为整型	浮点型转为整型的举例 In : d=4.6 In : int (d) Out: 4 字符串转为整型的举例 In : e= '12' In : int (e) Out: 12

## 2.5 主要内置函数

len	<p>输出字符串长度的举例  <b>In : a='finance'</b>  <b>In : len (a)</b>  <b>Out: 7</b></p> <p>输出对象（包括字符串、列表、元组等）长度或元素个数，len 是长度英文单词length的缩写</p> <p>输出列表的元素个数的举例。  <b>In : list2=['finance', 'risk management', '金融风险管理', 2020, 88.8]</b>  <b>In : len (list2)</b>  <b>Out:5</b></p>
max	<p>求最大值（max是最大值英文单词maximum的缩写）  以输出2018年1月15日至1月19日沪深300指数每日涨跌幅的最大值作为举例：  <b>In : list1=[-0.0054,-0.0077,0.0024,0.0087,0.0038]</b>  <b>In : max (list1)</b>  <b>Out:0.0087</b></p>
min	<p>求最小值（min是最小值英文单词minimum的缩写）  以输出2018年1月15日至1月19日沪深300指数每日涨跌幅的最小值作为举例：  <b>In : min (list1)</b>  <b>Out: -0.0054</b></p>
pow	<p>幂函数（pow是取了幂英文单词power的缩写）  pow (x,y) 是输出<math>x^y</math> (x的y次方) 的值  <b>In : pow (2,5)</b>  <b>Out: 32</b>  <b>In : pow (5,2)</b>  <b>Out: 25</b></p>
print	<p>输出字符串和变量等  <b>In : Print ('2018年1月15日至1月19日沪深300 指数每日涨跌幅: ', list1)</b>  2018年1月15日至19日沪深300指数每日涨跌幅: [-0.0054, 0.0077,0.0024,0.0087,0.0038]  注：输入的字符串需要放在引号的中间，并且字符串与输出的变量之间需要用逗号隔开</p>

## 2.5 主要内置函数

range	输出一个整数数列,一般用于 for 循环中	<p>函数语法结构:range(x,y, z)</p> <p>参数x:计数的起始值,默认(不填)是从0开始:</p> <p>参数y:计数的终止值,但不包括y</p> <p>参数z:步长,默认为1;</p> <pre>In : list(range(0,10)) Out: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] In : list(range(10)) Out: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] In : list(range(2,13,3)) Out: [2, 5, 8, 11] In : list(range(3,13,5)) Out: [3, 8]</pre> <p>注:在Python3.X版本中,输出结果需要用 list 函数</p>
reversed	输出一个逆序排列的序列(包括数列,元组,字符串等)	<pre>In: list1 = [-0.0054, 0.0077, 0.0024, 0.0087, 0.0038] In: List1_reversed = reversed(list1) In: list(list1_reversed) Out: [0.0038, 0.0087, 0.0024, 0.0077, -0.0054]</pre>
sorted	输出一个顺序排列的序列(包括数列、元组、字符串等)	<pre>In : list1_sorted = sorted(list1) In : list(list1_sorted) Out: [-0.0054, 0.0024, 0.0038, 0.0077, 0.0087]</pre>
sum	求和	<pre>In : sum(list1) Out: 0.0172</pre>
zip	将对象中对应的元素打包成一个个元组,并返回由这些元组组成的列表	<pre>In : code = ['000001', '000002', '000004', '000005'] In : stock = ['平安银行', '万科A', '国农科技', '世纪星源'] In : list(zip(code, stock)) Out: [( '000001', '平安银行'), ('000002', 万科 A ), ( '000004', '国农科技'), ('000005', '世纪星源')]</pre>

## 2.6 Python自定义函数

## 2.6.1 def语句

运用def语法时,函数的基本框架如下:

```
def 函数名(参数):  
    """函数说明文档"""  
    函数主体  
    return 返回对象
```

【例2-50】通过 Python自定义一个计算算术平均收益率的函数, 具体的代码如下:

```
def mean_a(r):  
    """定义一个求解算术平均收益率的函数  
    r:代表收益率的一个列表"""  
    total=sum(r)  
    return total/len(r)
```

## 2.6.1 def语句

【例2-51】运用在例2-50中自定义的函数 `mean_a`求解2018年1月15日至1月19日沪深300指数每日涨跌幅(收益率)的算数平均值,具体的代码如下:

```
list1=[-0.0054,0.0077,0.0024,0.0087,0.0038]  
mean_a(r=list1)  
Out[71]: 0.00344
```

通过以上的计算得到算术平均收益率是0.344%.此外,可以通过另一种方法验算该结果是否正确:

```
sum(list1)/5  
Out[73]: 0.00344
```

## 2.6.2 lambda语句

lambda函数在 Python中被称为是匿名函数，具体的函数基本格式如下：

函数名= lambda参数:表达式

用 lambda定义函数时，撰写的代码通常是控制在一行以内，因此可以用 lambda写相对简单的函数，或者是复杂函数的一个组成部分。

**【例2-52】**用 lambda定义计算算术平均收益率的函数,并且依然用新定义的函数求解2018年1月15日至1月19日沪深300指数每日涨跌幅的算数平均值,具体的代码如下：

```
mean_A=lambda r:sum(r)/len(r)
```

```
#用lambda定义函数
```

```
mean_A(r=list1)
```

```
Out[75]: 0.00344
```

## 2.7 Python语句



## 2.7.1 条件语句

条件语句是通过一条或多条语句的执行结果（True或者False）来决定执行的代码块：

```
if 判断语句 1:  
    执行语句 1  
elif 判断语句 2:  
    执行语句 2  
elif 判断语句 3:  
    执行语句 3  
...  
elif 判断语句 n:  
    执行语句 n  
else:  
    执行语句 n+1
```

## 2.7.1 条件语句

【例2-53】2019年1月4日，沪深300指数当天收盘上涨了2.4%，需要通过Python的条件语句并设定一个判断条件，用于判断是正收益还是非正收益，具体的代码如下：

```
r1=0.024
if r1>0:
    print("正收益",r1)
else:
    print("非正收益",r1)
正收益 0.024
```

【例2-54】2019年1月14日，沪深300指数当天收盘下跌了0.74%，需要设定两个判断条件，用于区分是正收益、零收益或负收益，具体的代码如下：

```
r2=-0.0074
if r2>0:
    print("正收益:",r2)
elif r2==0:
    print("零收益:",r2)
else:
    print("负收益:",r2)
负收益: -0.0074
```

## 2.7.2 循环语句

**for**循环可以遍历各种序列的项目，如一个列表或者字符串，**for**循环的语法结构如下：

```
for 迭代变量 (iterating_var) in 序列 (比如列表、字符串等):  
    陈述 (statements)
```

$$\bar{R} = \sqrt[n]{\prod_{i=1}^n (1 + R_i)} - 1$$

【例2-55】通过 Python定义一个计算几何平均收益率的函数,具体的代码如下：

```
def mean_g(r):  
    """定义一个计算几何平均收益率的函数  
    r:代表收益率的一个列表"""  
    total=1  
    for i in r:  
        total=total*(1+i)  
    return pow(total,1/len(r))-1
```

## 2.7.2 循环语句

**while**语句是用于循环执行程序，具体就是在某条件下，循环执行某段程序，以处理需要重复处理的相同任务。基本的语法结构如下：

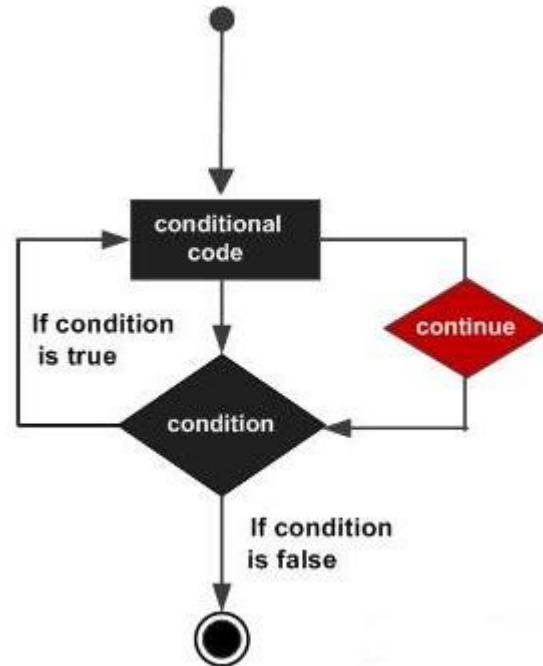
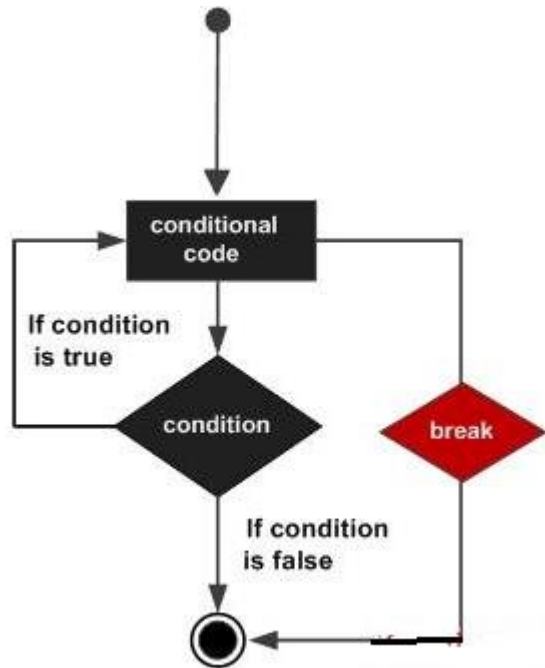
```
while 判断条件:  
    执行语句
```

【例2-57】假定需要依次输出0~10的数字，并且运用 **while**语句编写，具体的代码如下：

```
n=0  
while n<=10:  
    print("输出数字是:",n)  
    n+=1  
print("完")
```

## 2.7.3 循环控制语句

循环控制语句名称	具体功能
<b>break</b>	终止当前循环，且跳出整个循环
<b>continue</b>	终止当次循环，跳出该次循环，直接执行下一次循环
<b>Pass</b>	不执行任何操作，一般用于占据一个位置



## 2.7.4 条件循环

【例2-59】2019年1月2日至1月18日，沪深300指数的每日涨跌幅如下： -1.37%、2.4%、0.61%、-0.22%、1.01%、-0.19%、0.72%、-0.87%、1.96%、0.02%、-0.55%、1.82%。对此，分别完成若干个编程任务。

任务1：在依次访问以上这些涨跌幅数据时，一旦访问到涨幅大于2%时，就终止整个程序，并且输出已经访问的数据。就需要运用到for、if和 break搭配的语句，具体的代码如下：

```
r_list=[-0.0137,-0.0016,0.024,0.0061,-0.0022,0.0101,-0.0019,0.0072,-  
0.0087,0.0196,0.0002,-0.0055,0.0182]  
for i in r_list:  
    if i>0.02:  
        break  
    print("收益率数据:",i)
```

## 2.7.4 条件循环

**任务2：**在依次访问以上这些涨跌幅数据时,一旦访问到涨跌幅为负数时,就跳过这些负数的数据,并且输出全部非负数的数据。这时,可以有两种不同的代码:

第1种:运用到for、if和continue搭配的语句结构,具体的代码如下:

```
for i in r_list:
    if i<0:
        continue
    print("非负收益率数据:",i)
```

第2种: 运用到for、if、pass和else搭配的语句结构, 具体的代码如下:

```
for i in r_list:
    if i<0:
        pass
    else:
        print("非负收益率数据:",i)
```

## 2.7.5 异常捕捉处理语句

### **try...except...else...finally:**

**try**中包含要验证的语句；若出现异常，则执行异常对应的**except**语句，若无异常，则不执行**except**语句，**except**包含各类可能发生异常的类型；所有**except**皆未执行的话，将执行**else**语句；无论如何**finally**语句皆执行。

当在**try**子句中没有异常发生时，异常处理语句将不被执行。

当在**try**子句中有异常发生时，会按顺序搜索执行**except**搜索异常处理语句，直到第一个匹配的处理语句找到为止。

如果没有找到匹配的语句，异常（**e**）就会被临时保存起来，然后去执行**finally**语句；

### **注意:**

1、当执行**try...except**之间的语句序列没有发生异常时，则忽略异常处理部分(**except**)的语句；

2、**Except**引导的语句，只有在产生异常的情况下会被执行；**Finally**引导的语句则是一定会执行，无论是否有异常产生；

3、一般会将一个没有指定异常的**except**语句放在最后，它会匹配任何异常并处理。因此，**finally**部分常常不是必须的。



## 2.7.5 异常捕捉处理语句

```
def func():
    import numpy as np
    try:
        road_num = np.random.randint(1,4)
        print("road_num == " , road_num)
        if road_num == 1:
            a=1
            b=2
            assert a == b
        elif road_num == 2:
            a=1
            b=0
            a/b
        else:
            print(故意输出错误)
    except AssertionError:
        print("断言错误")
    except SyntaxError:
        print("语法错误")
    except ArithmeticError:
        print("数学错误")
    except BaseException:
        print("异常基类")
    else:
        print("我在摸鱼")
    finally:
        print("我一定会露脸的")
```

## 2.7.5 异常捕捉处理语句

用func()引用

结果可能是：

`road_num == 1`

断言错误

我一定会露脸的

实际上，上述语句中 `else` 永远也捕获不到错误，因为错误都是从 `BaseException` 类派生的，它会捕获所有的错误。关于这一点，可以查阅常见的错误类型和继承关系：

<https://docs.python.org/3/library/exceptions.html#exception-hierarchy>

## 2.8 Python模块导入

## 2.8.1 模块导入方法

	Python的代码	具体说明与示例
方法1	<code>import 模块名</code>	直接导入整个模块，这种导入方式比较占用内存，具体的示例如下： <code>import math</code>
方法2	<code>import 模块名 as 名称缩写</code>	导入整个模块的同时给该模块取一个别名，往往是用在模块名字比较长的情况下，这样能节省调用该模块时的输入时间。具体的示例如下：In []: <code>import matplotlib as mp</code>
方法3	<code>import 模块名.子模块名 as 名称缩写</code>	导入某个模块的子模块，并且给该子模块取一个别名，当然是否取别名是一个可选项，这样占用的内存也比较少。具体的示例如下：In []: <code>import matplotlib.pyplot as plt</code>
方法4	<code>from 模块名 import 函数</code>	从特定模块中导入某个或者某几个函数（不同的数之间用英文逗号隔开）这样不仅占用的内存比较少，而且这些函数可以直接以函数名字的方式使用，具体的示例如下： In: <code>from math import exp, log, sqrt</code> In: <code>log(10)</code> Out:2.302585092994046 In: <code>exp(3)</code> Out:20.085536923187668
方法5	<code>from 模块名,子模块名 import 函数</code>	与方法4很相似,只不过是从小模块的子模块中导入某个或者某几个函数。具体的示例如下： In : <code>from matplotlib.pyplot import figure,plot</code>

## 2.8.2 自定义函数和模块

在工程量大的时候，Python的一个py文件如何调用另一个py文件中已经定义好的函数呢？为了解决这个实际问题，我们分成两种情况考虑：在同一个文件夹下的调用、不在一个文件夹下的调用。

### 1、在工作文件夹下

假如一个文件夹（D:\Zhangzw\Python\Python金融数据分析\RawData\第2章）下，同时存在两个py文件：callDemo.py及calledDemo.py。

其中，calledDemo.py文件定义如下：

```
def adds(x,y):  
    print('两值相加和为: %d' %(x+y))  
    print('这是一个测试程序，检测函数同一工作文件夹下函数的相互引用，加法')  
)
```

```
def times(x,y):  
    print('两值相乘积为: %d' %(x*y))  
    print('这是一个测试程序，检测函数同一工作文件夹下函数的相互引用，乘法')
```

## 2.8.2 自定义函数和模块

现在，callDemo.py要引用calledDemo.py里面定义的adds、times函数：

```
import calledDemo #导入模块calledDemo，其中已定义了一个函数adds
calledDemo.adds(100,20000)
calledDemo.times(300,2)
```

```
from calledDemo import adds #直接导入adds函数
adds(1,2)
```

```
from calledDemo import times
times(4,2)
```

## 2.8.2 自定义函数和模块

### 2、不在工作文件夹下

Python在导入模块py文件时，实际上是在sys.path的路径里按顺序查找这个模块的（sys.path是一个列表，里面以字符串的形式存储了许多路径）。因此，要使用模块py文件中的函数，需要先将它的文件路径添加到sys.path中。

假如在另一个文件夹（D:\Zhangzw\Python\Python金融数据分析\RawData\第2章\anotherFile）下，存在另一个py文件：calledDemoAnotherFile.py:

```
def minus(x,y):
    print('两值相减差为: %d' %(x-y))
    print('这是一个测试程序，检测函数不在工作文件夹下函数的相互引用，减法')

def powers(x,y):
    print('两值乘方值为: %d' %(x**y))
    print('这是一个测试程序，检测函数不在工作文件夹下函数的相互引用，乘方')
```

## 2.8.2 自定义函数和模块

现在callDemo.py要引用里面定义的函数minus和powers:

```
import sys
sys.path.append(r'D:\Zhangzw\Python\Book\chpt2\anotherFile') #先将它的文件路
径添加到sys.path中
```

```
import calledDemoAnotherFile
calledDemoAnotherFile.minus(1,2)
calledDemoAnotherFile.powers(3,2)
```

```
from calledDemoAnotherFile import minus #直接导入minus函数
minus(1,2)
from calledDemoAnotherFile import powers
powers(4,2)
```



## 2.8.2 自定义函数和模块

### 3、文件夹操作

```
import os
os.getcwd() # 获取当前工作文件夹

os.chdir('D:\Zhangzw\Python\Python金融数据分析\RawData\第2章') # 指定当前
工作文件夹
os.getcwd() # 获取当前工作文件夹

%run calledDemo.py #运行当前工作文件夹里的指定程序
# 当然也可以直接把文件路径加上去
%run D:\Zhangzw\Python\Python金融数据分析\RawData\第2章\calledDemo.py

%run D:\Zhangzw\Python\Python 金融 数 据 分 析 \RawData\ 第 2 章
\anotherFile\calledDemoAnotherFile.py
```

**Your appreciation makes me a miracle.  
Thank you!**

**Frank Ziwei Zhang  
18117228563  
frank8027@163.com**



**上海對外經貿大學**  
SHANGHAI UNIVERSITY OF INTERNATIONAL BUSINESS AND ECONOMICS