**Python 基础总结**

**Python 中的六大基本数据类型：**

1. **数字（int\float\bool\complex）举例如下：**

| **int** | **long** | **float** | **complex** |
| --- | --- | --- | --- |
| 10 | 51924361L | 0.0 | 3.14j |
| 100 | -0x19323L | 15.20 | 45.j |
| -786 | 0122L | -21.9 | 9.322e-36j |
| 080 | 0xDEFABCECBDAECBFBAEl | 32.3+e18 | .876j |
| -0490 | 535633629843L | -90. | -.6545+0J |
| -0x260 | -052318172735L | -32.54e100 | 3e+26J |
| 0x69 | -4721885298529L | 70.2-E12 | 4.53e-7j |

* 长整型也可以使用小写"L"，但是还是建议使用大写"L"，避免与数字"1"混淆。
* Python还支持复数，复数由实数部分和虚数部分构成，可以用a + bj,或者complex(a,b)表示， 复数的实部a和虚部b都是浮点型

1. **字符串**

字符串str用单引号(' ')或双引号(" ")括起来

字符串或串(String)是由数字、字母、下划线组成的一串字符。

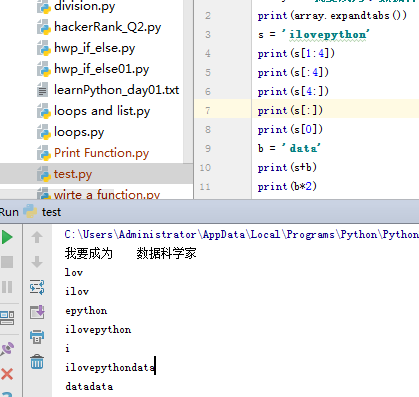
一般记为 : s = ‘abc123

python的字串列表有2种取值顺序:

* 从左到右索引默认0开始的，最大范围是字符串长度少1
* 从右到左索引默认-1开始的，最大范围是字符串开头

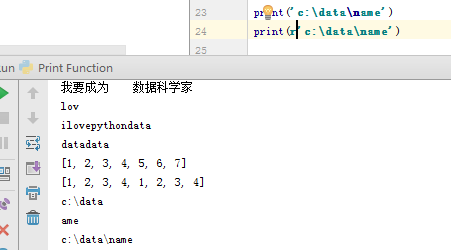
S = ‘Ilovepython’,S[1:5]=’love’,不包含5，切片操作的时候，**注意：[1:5],[:5],[5:],[:]**

实际操作：

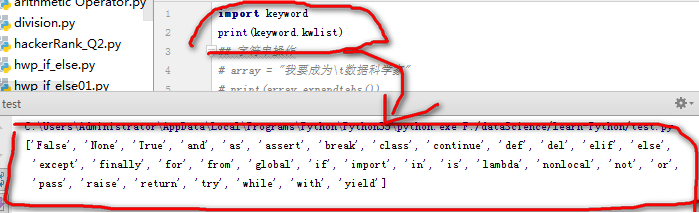


加号（+）是字符串连接运算符，星号（\*）是重复操作

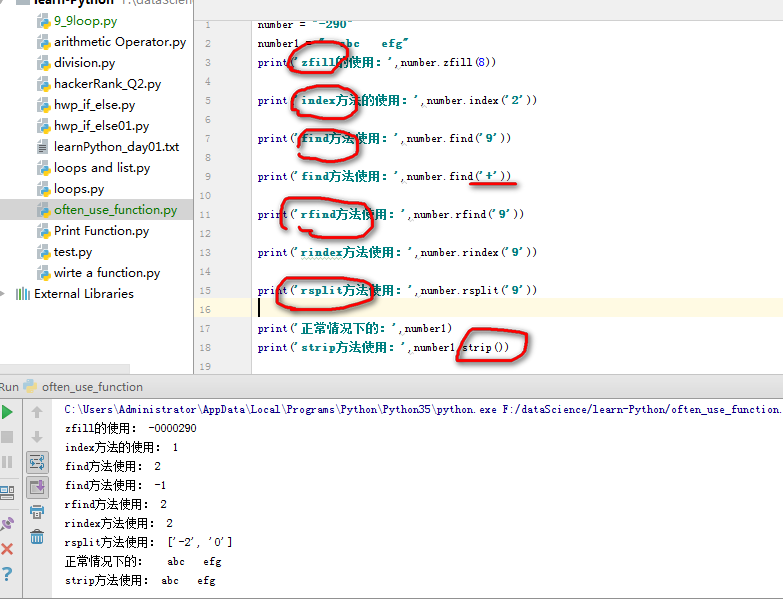
使用反斜杠()转义特殊字符,如果不用反斜杠可用r

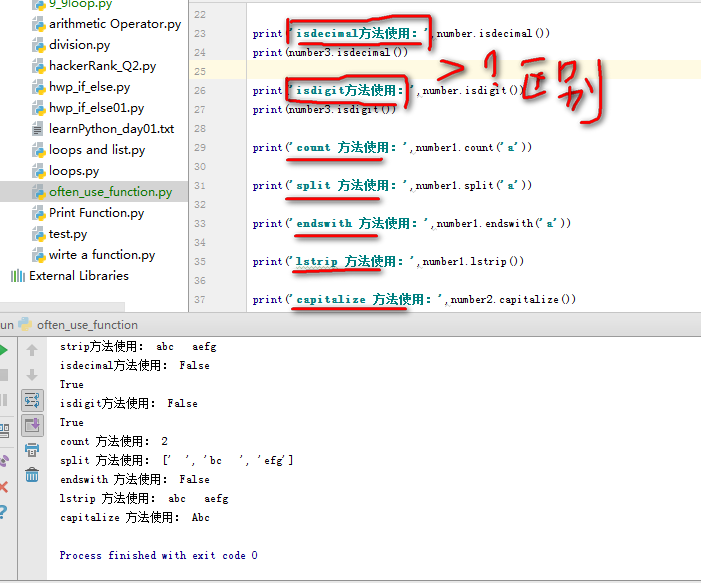


系统自带的一些字符



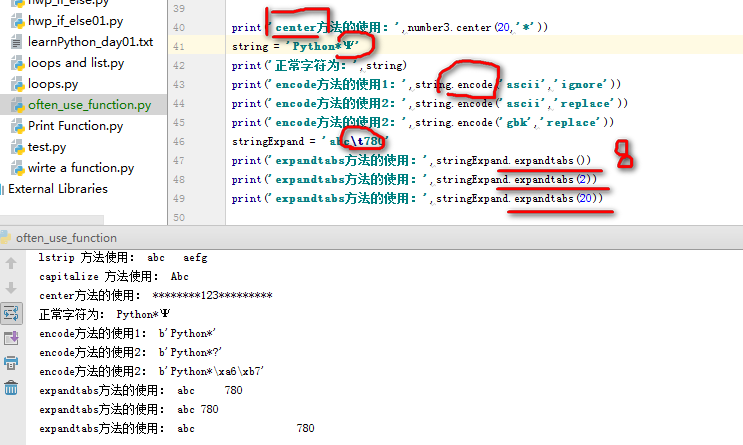
常用的函数操作

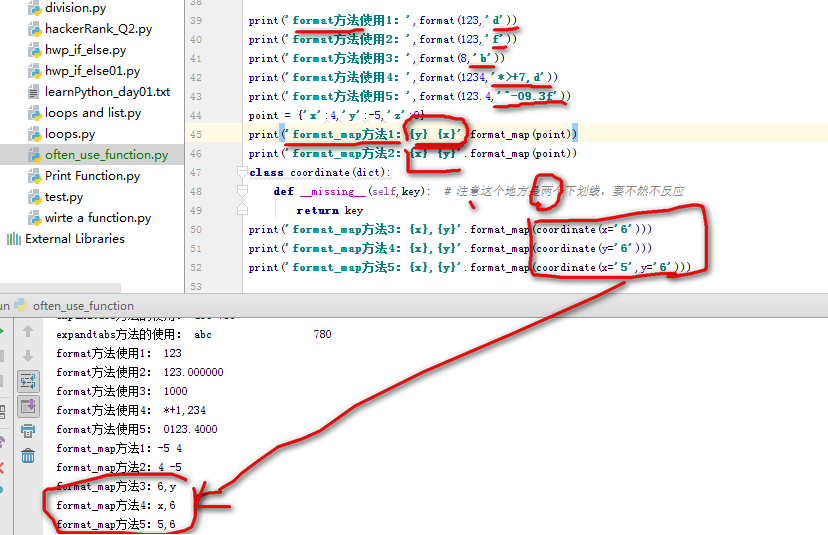


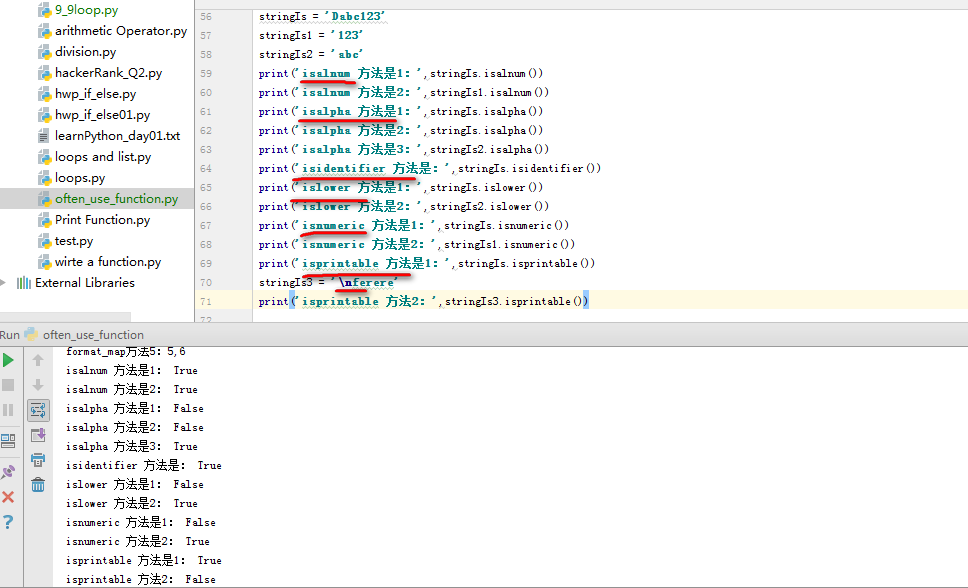


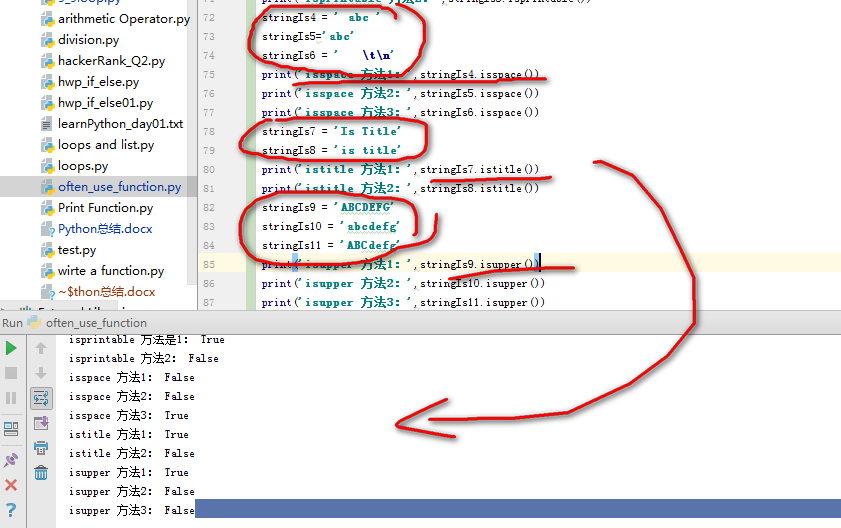
Isdigit,isdecimal,isnumeric 不同

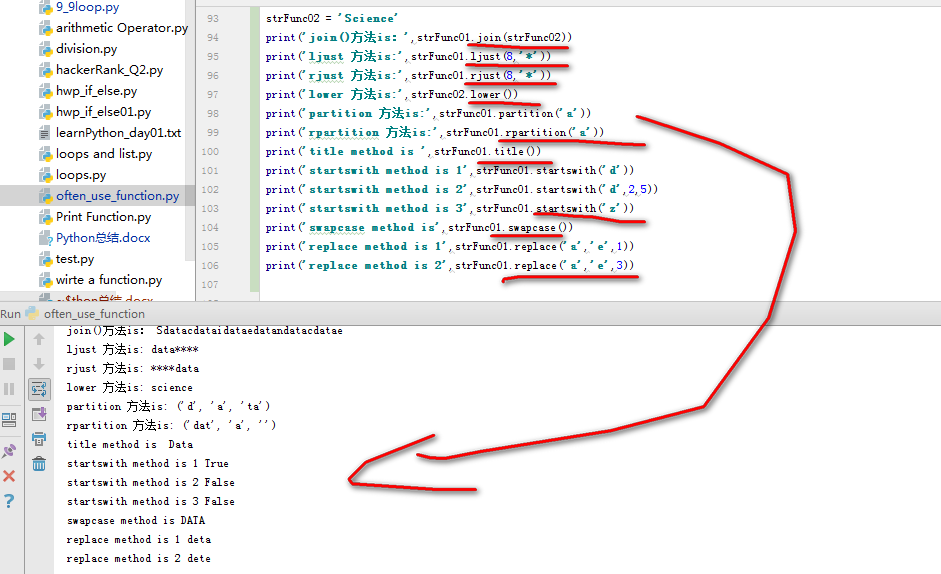
<https://stackoverflow.com/questions/22789392/str-isdecimal-and-str-isdigit-difference-example>

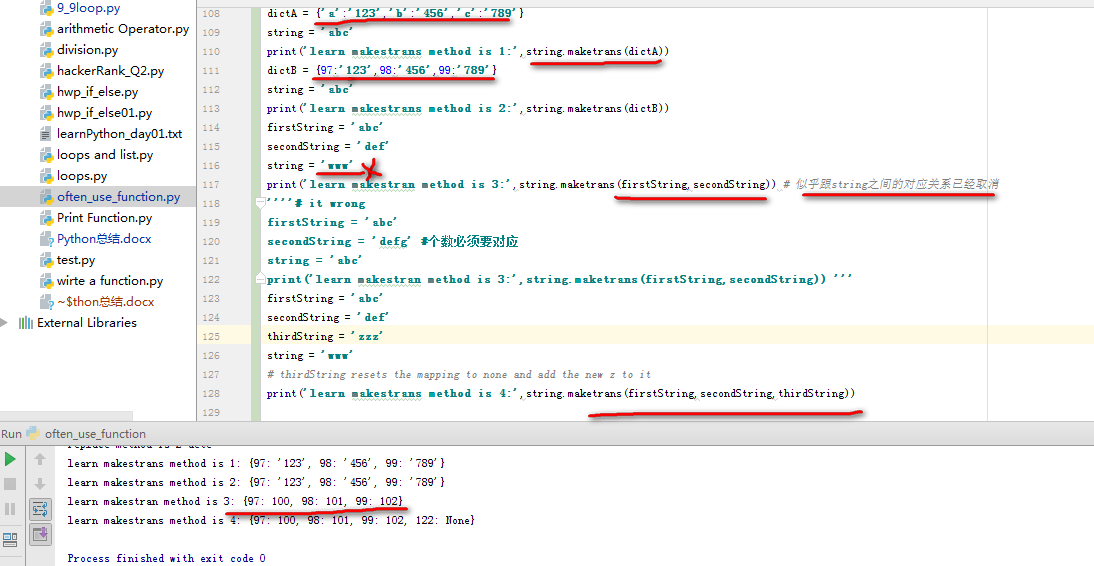










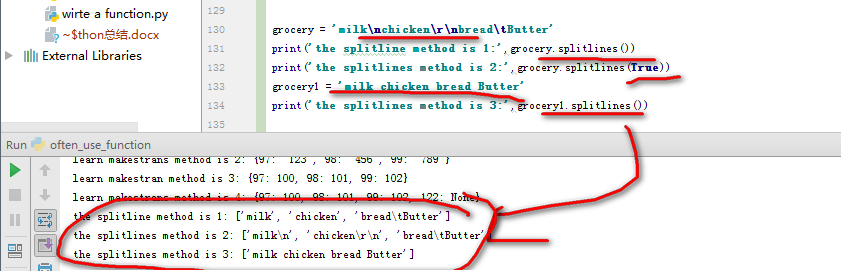


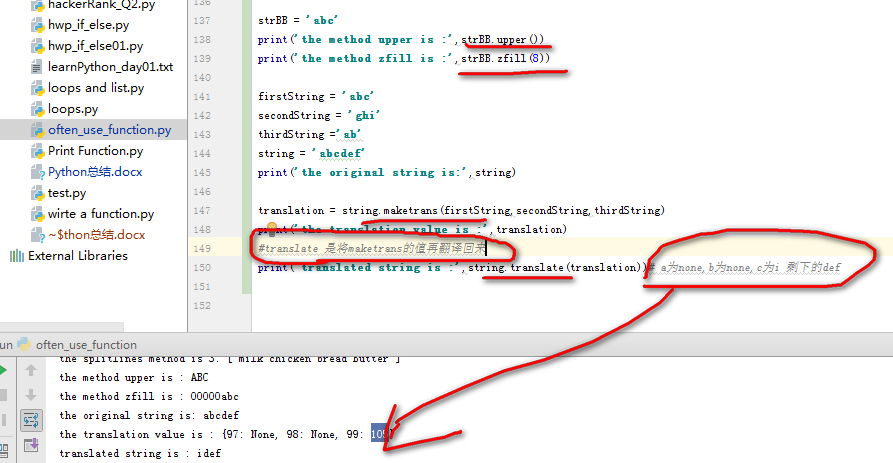
The different about split vs splitlines vs rsplit

<https://www.jquery-az.com/python-split-string-methods-split-rsplit-and-splitlines-6-examples/>

splitline 符号包括

| Representation | Description |
| --- | --- |
| **\n** | Line Feed |
| **\r** | Carriage Return |
| **\r\n** | Carriage Return + Line Feed |
| **\v** or **\x0b** | Line Tabulation |
| **\f** or **\x0c** | Form Feed |
| **\x1c** | File Separator |
| **\x1d** | Group Separator |
| **\x1e** | Record Separator |
| **\x85** | Next Line (C1 Control Code) |
| **\u2028** | Line Separator |
| **\u2029** | Paragraph Separator |





最后附上，字符串经常操作的函数定义：

|  |  |
| --- | --- |
| 字符串函数功能 | |
| array.index() | 查找某一个字符在字符串中的位置索引值，如果没有报错 |
| array.find() | 查找某一个字符在字符串中的位置索引值，如果没有返回-1 |
| array.rfind() | 从右边开始查找 |
| array.rindex() | 从右边开始查找索引 |
| array.rsplit() | 从右边开始以某一个自定义的符号切分 |
| array.strip() | 去空格 |
| array.isdecimal() | 是否是数字 |
| array.isdigit() | 是否是数字 |
| array.count() | 统计字符串中字符数量 |
| array.split() | 以某一个自定义的符号切分 |
| array.endswith() | 以什么结尾 |
| array.lstrip() | 左边去空格 |
| array.capitalize() | It returns a copy of the string with only its first character capitalized |
| array.center() | The method center() returns centered in a string of length width. Padding is done using the specified fillchar. Default filler is a space. |
| array.encode() | 编码 |
| array.expandtabs() | It returns a copy of the string in which tab characters ie. '\t' are expanded using spaces, optionally using the given tabsize (default 8).. |
| array.format() | 格式化数据，可以传入字典的值、不同位置 |
| array.format\_map() | str.format(\*\*mapping) |
| array.isalnum() | The method isalnum() checks whether the string consists of alphanumeric characters.either alphabets or numbers |
| array.isalpha() | The method isalpha() checks whether the string consists of alphabetic characters only |
| array.isidentifier() | The isidentifier() method returns True if the string is a valid identifier in Python. If not, it returns False |
| array.islower() | The method islower() checks whether all the case-based characters (letters) of the string are lowercase. |
| array.isnumeric() | The method isnumeric() checks whether the string consists of only numeric characters. This method is present only on unicode objects. |
| array.isprintable() | The isprintable() methods returns True if all characters in the string are printable or the string is empty. If not, it returns False. |
| array.isspace() | The isspace() method returns True if there are only whitespace characters in the string. If not, it return False. |
| array.istitle() | The istitle() returns True if the string is a titlecased string. If not, it returns False |
| array.isupper() | The string isupper() method returns whether or not all characters in a string are uppercased or not. |
| array.join() | The join() is a string method which returns a string concatenated with the elements of an iterable. |
| array.ljust() | The string ljust() method returns a left-justified string of a given minimum width. |
| array.lower() | The string lower() method converts all uppercase characters in a string into lowercase characters and returns it. |
| array.maketrans() | The string maketrans() method returns a mapping table for translation usable for translate() method. |
| array.partition() | The partition() method splits the string at the first occurrence of the argument string and returns a tuple containing the part the before separator, argument string and the part after the separator. |
| array.replace() | The replace() method returns a copy of the string where all occurrences of a substring is replaced with another substring. |
| array.rjust() | The string rjust() method returns a right-justified string of a given minimum width. |
| array.rpartition() | The rpartition() splits the string at the last occurrence of the argument string and returns a tuple containing the part the before separator, argument string and the part after the separator. |
| array.splitlines() | The splitlines() method splits the string at line breaks and returns a list of lines in the string. |
| array.startswith() | The startswith() method returns True if a string starts with the specified prefix(string). If not, it returns False. |
| array.swapcase() | The string swapcase() method converts all uppercase characters to lowercase and all lowercase characters to uppercase characters of the given string, and returns it. |
| array.title() | The title() method returns a string with first letter of each word capitalized; a title cased string. |
| array.translate() | The string translate() method returns a string where each character is mapped to its corresponding character in the translation table. |
| array.upper() | The string upper() method converts all lowercase characters in a string into uppercase characters and returns it. |
| array.zfill() | The zfill() method returns a copy of the string with '0' characters padded to the left |

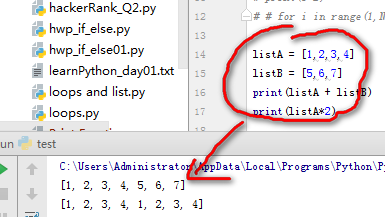
1. **列表类型 list;**

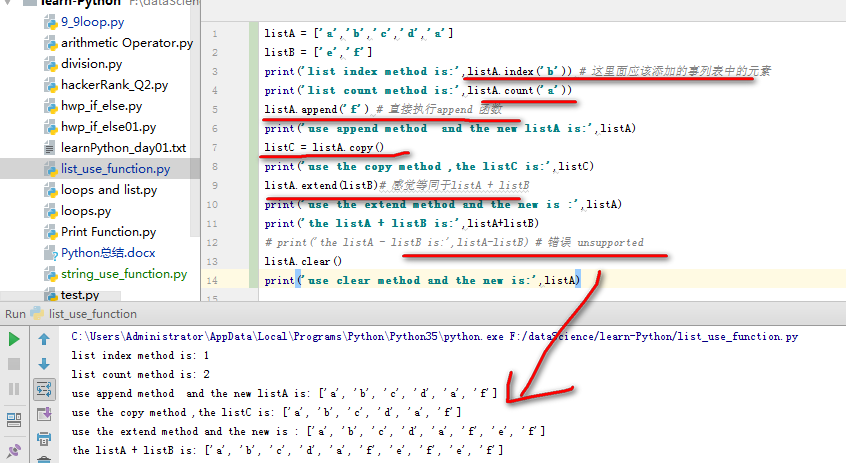
使用最频繁的数据类型

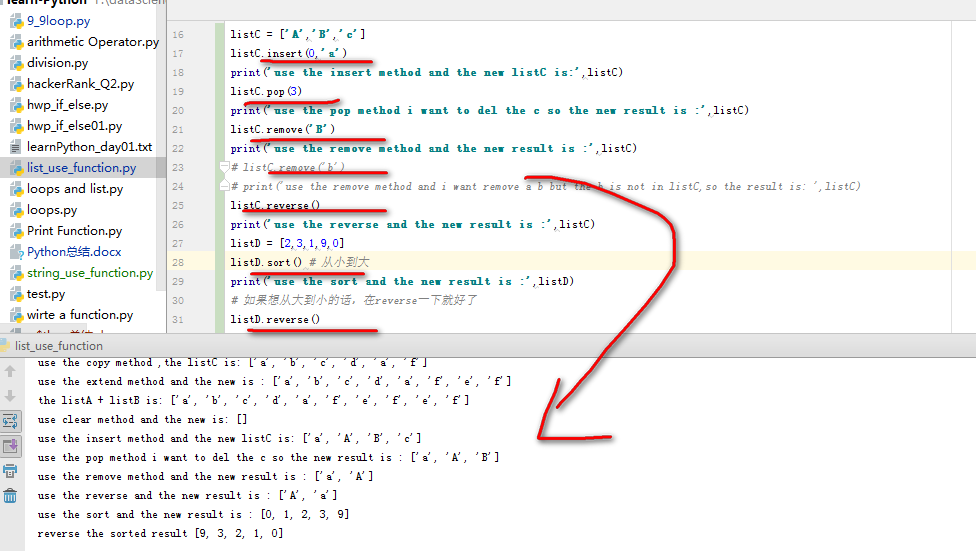
表示方法[1,2,3]

加号（+）是列表连接运算符，星号（\*）是重复操作

实际操作：







The different about pop\ remove \del\

<https://stackoverflow.com/questions/11520492/difference-between-del-remove-and-pop-on-lists>

pop返回值，pop\del 是索引，remove 是值，error mode 也不同

|  |  |
| --- | --- |
| 列表部分函数功能 | |
| listA.index() |  |
| listA.count() |  |
| listA.append() |  |
| listA.clear() |  |
| listA.copy() |  |
| listA.extend() |  |
| listA.insert() |  |
| listA.pop() |  |
| listA.remove() |  |
| listA.reverse() |  |
| listA.sort() |  |

1. **元祖 tuple;**

元组（tuple）与列表类似，不同之处在于元组的元素不能修改;

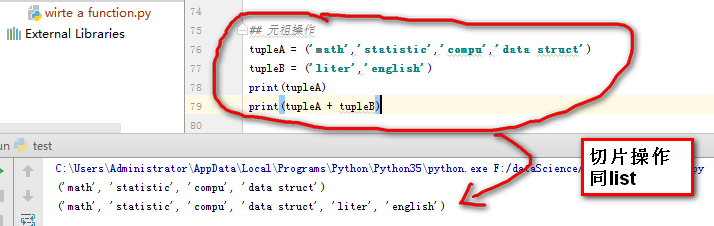
表示方法（1,2,3）

string、list和tuple都属于sequence（序列）

tup1 = () # 空元组

tup2 = (20,) # 一个元素，需要在元素后添加逗号

虽然tuple的元素不可改变，但它可以包含可变的对象，比如list列表



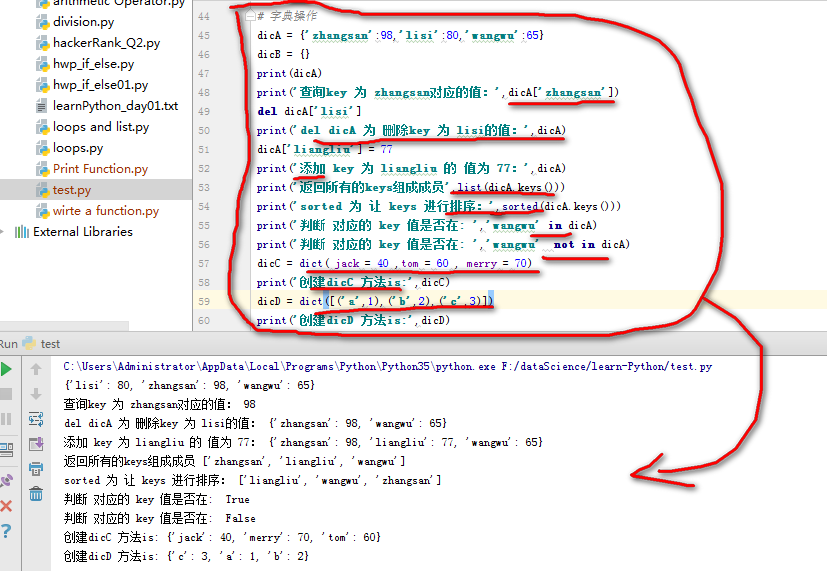
1. **字典;**

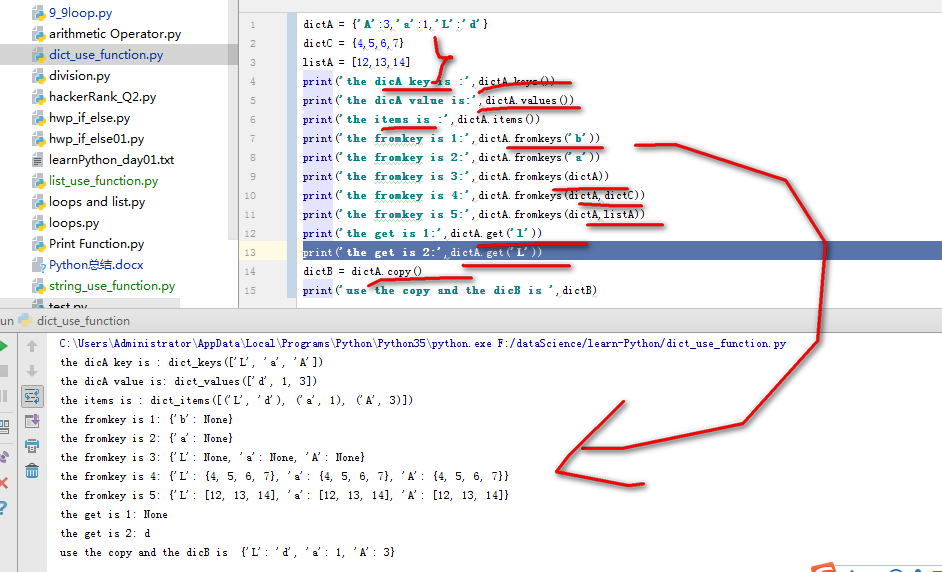
字典是一种映射类型（mapping type），它是一个无序的键 : 值对集合；

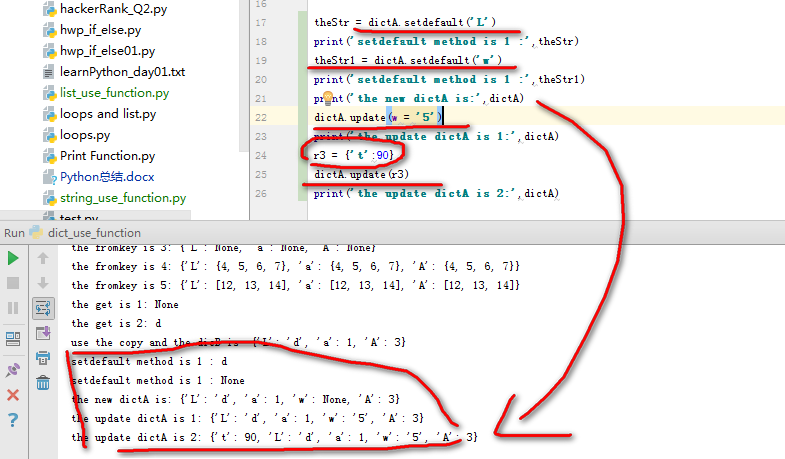
表示方法：{‘jack’:1221,’tom’:2323}

关键字必须使用不可变类型，也就是说list和包含可变类型的tuple不能做关键字

在同一个字典中，关键字还必须互不相同







|  |  |
| --- | --- |
| 字典部分函数功能 | |
| dicA.keys() |  |
| dicA.pop() |  |
| dicA.copy() |  |
| dicA.clear() |  |
| dicA.fromkeys() | The fromkeys() method creates a new dictionary from the given sequence of elements with a value provided by the user. |
| dicA.get() |  |
| dicA.items() |  |
| dicA.popitem() |  |
| dicA.setdefault() | The setdefault() method returns the value of a key (if the key is in dictionary). If not, it inserts key with a value to the dictionary. |
| dicA.update() | The update() method updates the dictionary with the elements from the another dictionary object or from an iterable of key/value pairs. |
| dicA.values() |  |

1. **集合；**

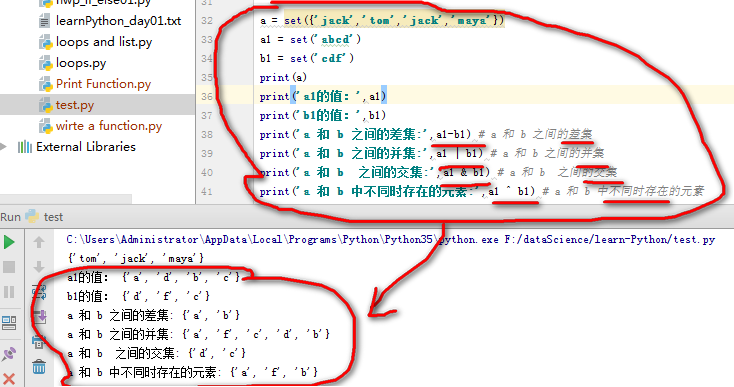
集合（set）是一个无序不重复元素的集；

基本功能是进行成员关系测试和消除重复元素

可以使用大括号 或者 set()函数创建set集合，注意：创建一个空集合必须用 set() 而不是 { }，因为{ }是用来创建一个空字典；

表示方法：{1,2,3,4}

差集，并集，交集，不同时存在的元素等

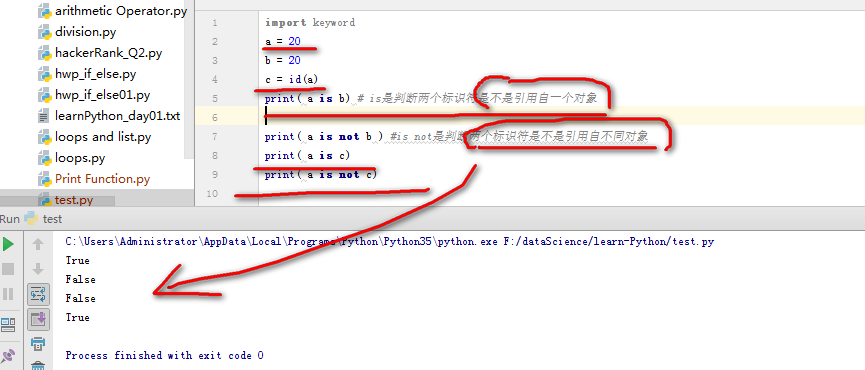


**基类运算包括：**

1. 算术运算符
   1. + ，-，\*，/，%，//,\*\*
2. 位运算符
   1. &(按位与运算) , |(按位或运算),^(按位异或运算),~(按位取反运算),<<(左移动运算符),>>(右移动运算符)



1. 赋值运算符
   1. =，+=，-=，\*=，/=,%=,\*\*=,//=
2. 逻辑运算符
   1. and , or , not
3. 身份运算符
   1. is ， is not



1. 成员运算符
   1. in，not in
2. 比较运算符
   1. ==,!=,>,<,>=,<=，<>

**以下表格列出了从最高到最低优先级的所有运算符**：

| **运算符** | **描述** |
| --- | --- |
| \*\* | 指数 (最高优先级) |
| ~ + - | 按位翻转, 一元加号和减号 (最后两个的方法名为 +@ 和 -@) |
| \* / % // | 乘，除，取模和取整除 |
| + - | 加法减法 |
| >> << | 右移，左移运算符 |
| & | 位 'AND' |
| ^ | | 位运算符 |
| <= < > >= | 比较运算符 |
| <> == != | 等于运算符 |
| = %= /= //= -= += *= \**= | 赋值运算符 |
| is is not | 身份运算符 |
| in not in | 成员运算符 |
| not or and | 逻辑运算符 |

**Python 中的流程控制：**

1. If , if-elif,if-elif-else,if-else

当if有多个条件时可使用括号来区分判断的先后顺序，括号中的判断优先执行

此外 and 和 or 的优先级低于>（大于）、<（小于）等判断符号，即大于和小于在没有括号的情况下会比与或要优先判断

由于 python 并不支持 switch 语句，所以多个条件判断，只能用 elif 来实现，

如果判断需要多个条件需同时判断时，可以使用 or （或），表示两个条件有一个成立时判断条件成功；

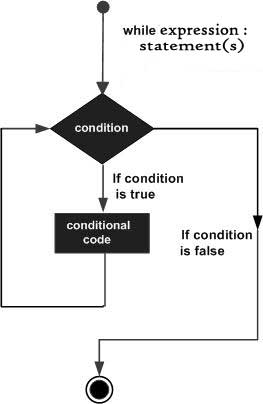
使用 and （与）时，表示只有两个条件同时成立的情况下，判断条件才成功。

9\*9乘法表看编写代码的思维方式

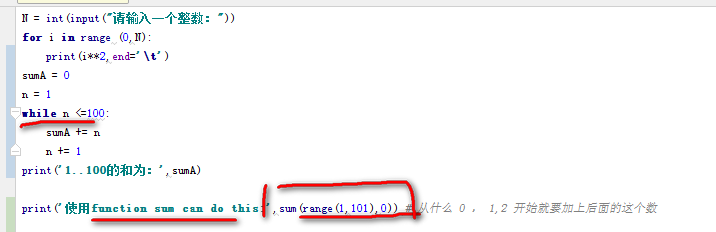
*#* ***错误1，用列的方式来思考是错误的*** *# for j in range(1,10):  
# for i in range(1,10):  
# if (i >= j):  
# print(i,"\*",j,"=",i\*j)  
#* ***1. 横向看计算机的输出方式；2.如何打印出2\*1,2\*2的方式 表达；3.i的值决定了j的循环方式；4.i打印完就会换行；5.同时解决print 换行问题*****for** i **in** range(1,10):  
 **for** j **in** range(1,i+1):  
 print(i,**"\*"**,j,**"="**,i\*j,end=**'\t'**)  
 print(**'\r'**)  
*# 解决取消print 回车问题  
# for i in range(1,5):  
# print(i,end=' ')*

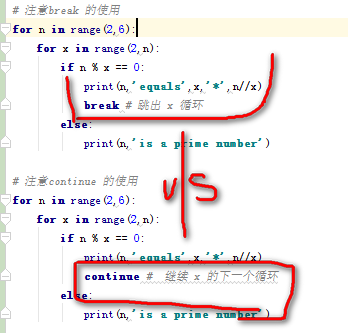
**Python 中的循环语句**

* + 1. **for**
    2. **while**



* + 1. **配合 break , continue , pass**

****

****

|  |  |
| --- | --- |
| **break 部分** | **continue 部分** |
| 3 is a prime number | 3 is a prime number |
| 4 equals 2 \* 2 | 4 equals 2 \* 2 |
| 5 is a prime number | 4 is a prime number |
| 5 is a prime number | 5 is a prime number |
| 5 is a prime number | 5 is a prime number |
|  | 5 is a prime number |

**Python 函数**

函数存在的目的在于减少代码的重复编写，通过调用函数来快速实现代码整合。

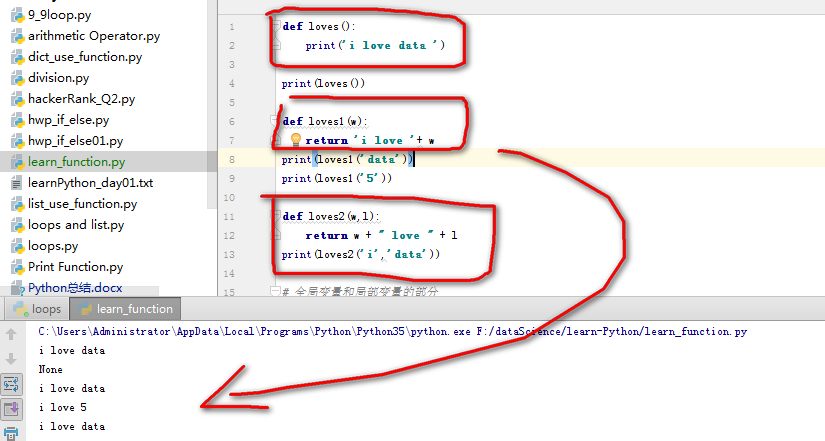
Python 定义函数使用 def 关键字，一般格式如下：

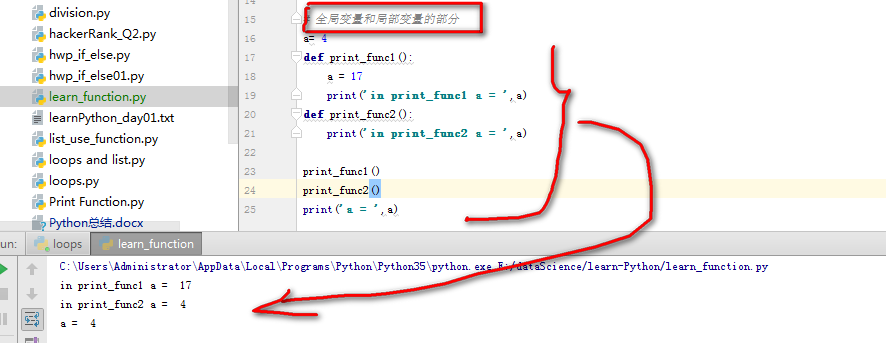
def 函数名（参数列表）：

函数体

定义在函数内部的变量拥有一个局部作用域，定义在函数外的拥有全局作用域。

通过以下实例，你可以清楚了解Python函数变量的作用域：

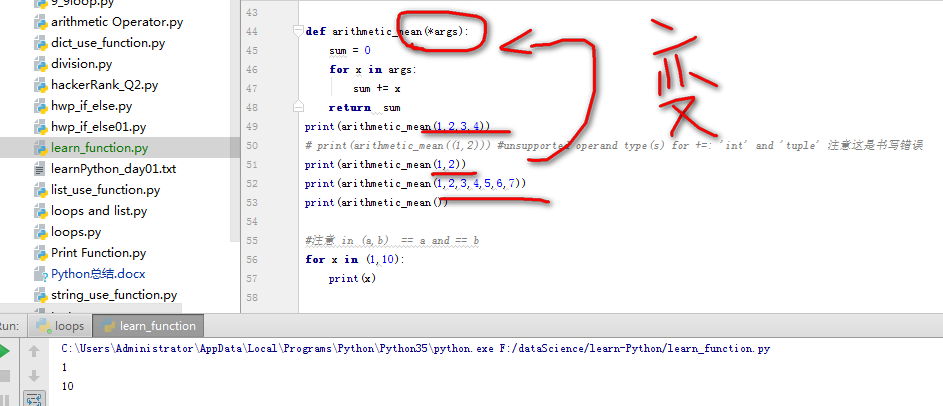


****

****

**可变参数列表**

最后,一个最不常用的选择是可以让函数调用可变个数的参数.这些参数被包装进一个元组(查看元组和序列).在这些可变个数的参数之前,可以有零到多个普通的参数:

****

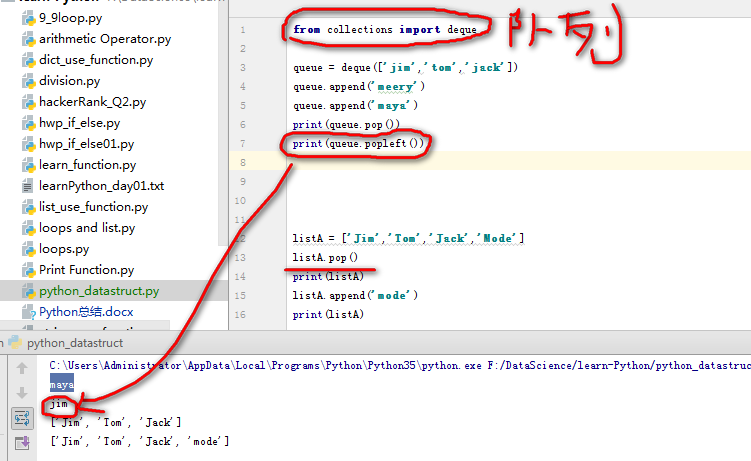
**Python 数据结构**

**将列表当做堆栈使用**

列表方法使得列表可以很方便的作为一个堆栈来使用，堆栈作为特定的数据结构，最先进入的元素最后一个被释放（后进先出）。用 append() 方法可以把一个元素添加到堆栈顶。用不指定索引的 pop() 方法可以把一个元素从堆栈顶释放出来。

**将列表当作队列使用**

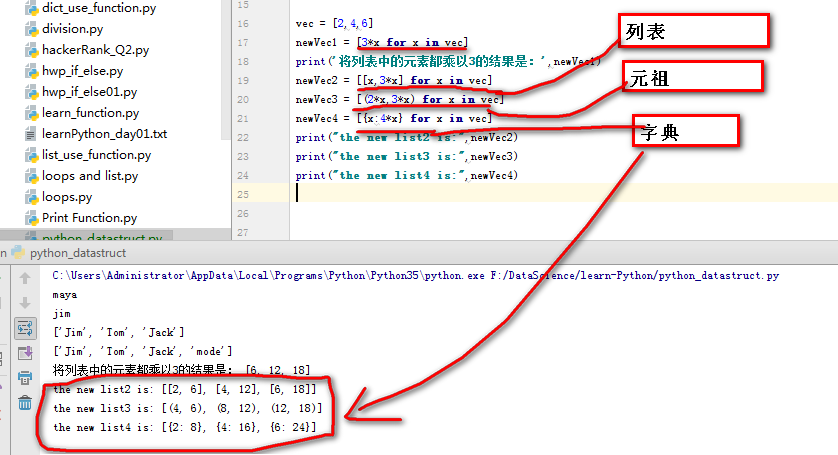
也可以把列表当做队列用，只是在队列里第一加入的元素，第一个取出来；但是拿列表用作这样的目的效率不高。在列表的最后添加或者弹出元素速度快，然而在列表里插入或者从头部弹出速度却不快（因为所有其他的元素都得一个一个地移动）

****

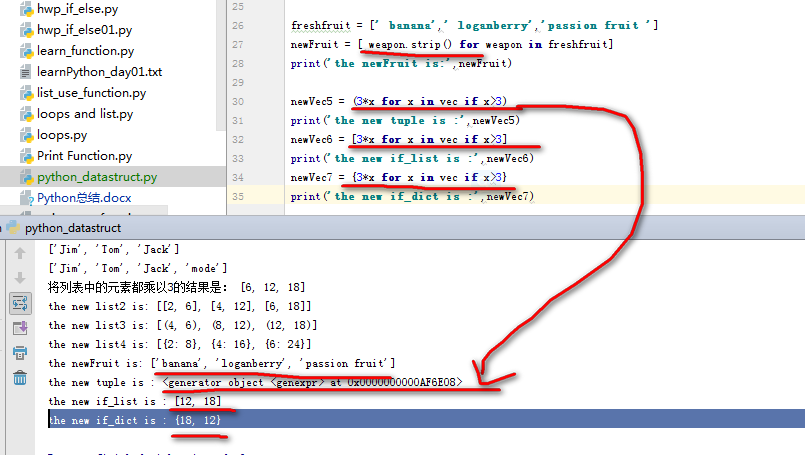
**列表推导式**

列表推导式提供了从序列创建列表的简单途径。通常应用程序将一些操作应用于某个序列的每个元素，用其获得的结果作为生成新列表的元素，或者根据确定的判定条件创建子序列。

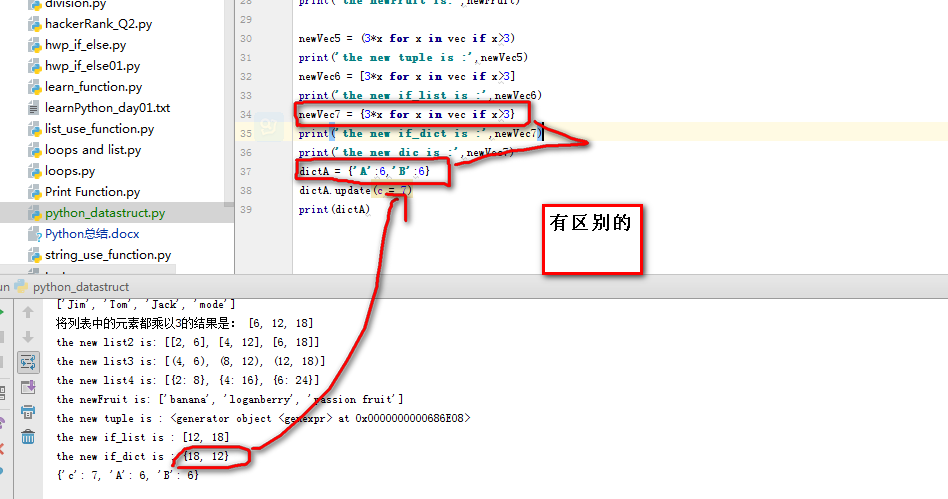
每个列表推导式都在 for 之后跟一个表达式，然后有零到多个 for 或 if 子句。返回结果是一个根据表达从其后的 for 和 if 上下文环境中生成出来的列表。如果希望表达式推导出一个元组，就必须使用括号。

****

可以用 if 子句作为过滤器

****

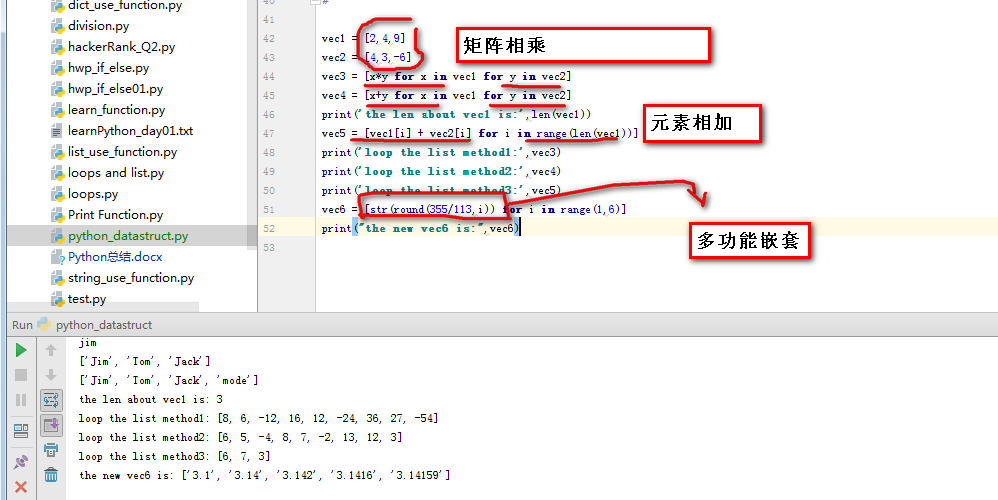
变化括号方式-尝试

****

**Pycharm 调试程序的方法**

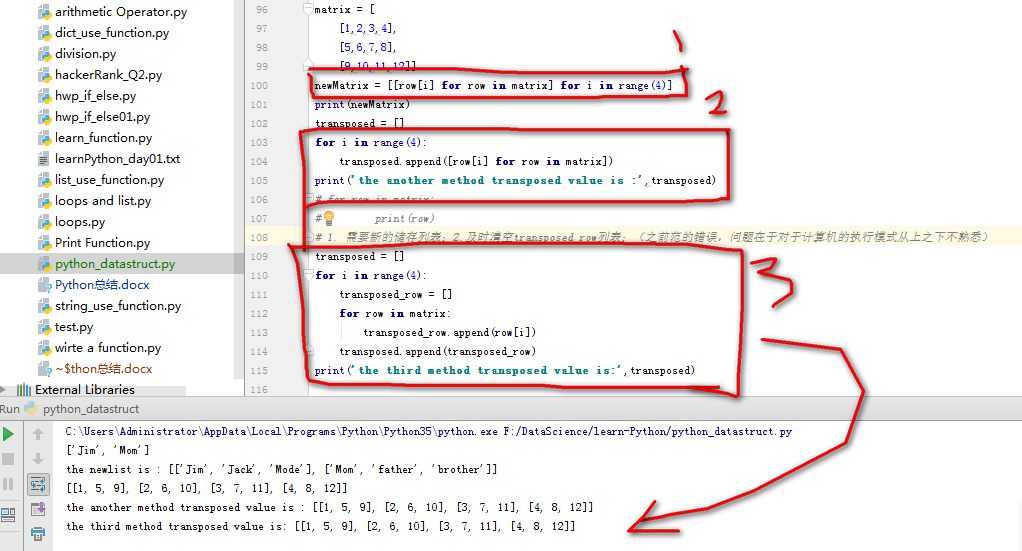
[**http://blog.csdn.net/u013088062/article/details/50130991**](http://blog.csdn.net/u013088062/article/details/50130991)

range(4) = 0 ，1,2,3

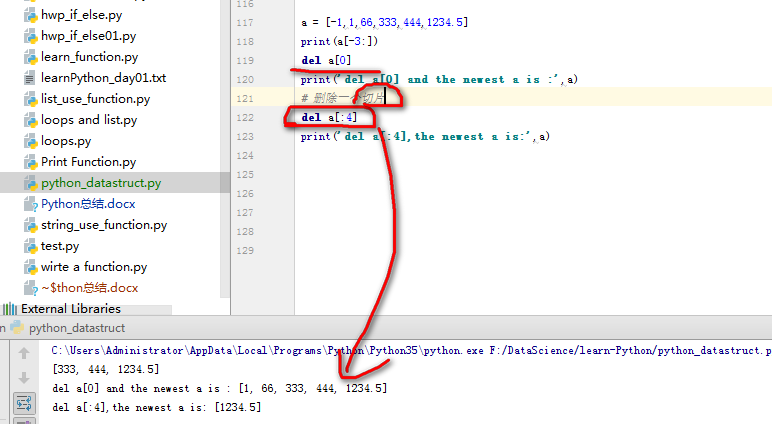


嵌套列表解析

以下实例将3X4的矩阵列表转换为4X3列表：

****

**del 语句**

使用 del 语句可以从一个列表中依索引而不是值来删除一个元素。这与使用 pop() 返回一个值不同。可以用 del 语句从列表

**元组和序列**

元组由若干逗号分隔的值组成，在输入时可能有或没有括号， 不过括号通常是必须的（如果元组是更大的表达式的一部分）



**集合**

集合是一个无序不重复元素的集。基本功能包括关系测试和消除重复元素。

可以用大括号({})创建集合。注意：如果要创建一个空集合，你必须用 set() 而不是 {} ；后者创建一个空的字典

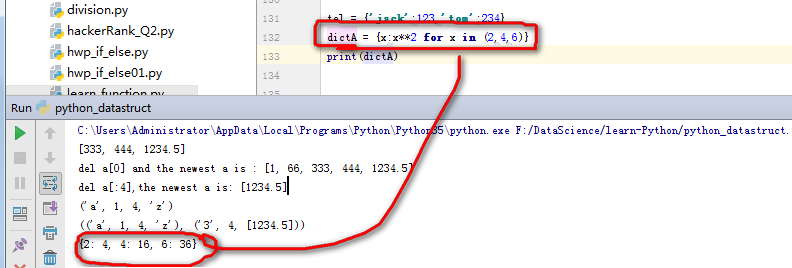
**字典**

另一个非常有用的 Python 内建数据类型是字典。

序列是以连续的整数为索引，与此不同的是，字典以关键字为索引，关键字可以是任意不可变类型，通常用字符串或数值。

理解字典的最佳方式是把它看做无序的键=>值对集合。在同一个字典之内，关键字必须是互不相同。

一对大括号创建一个空的字典：{}



**遍历技巧**

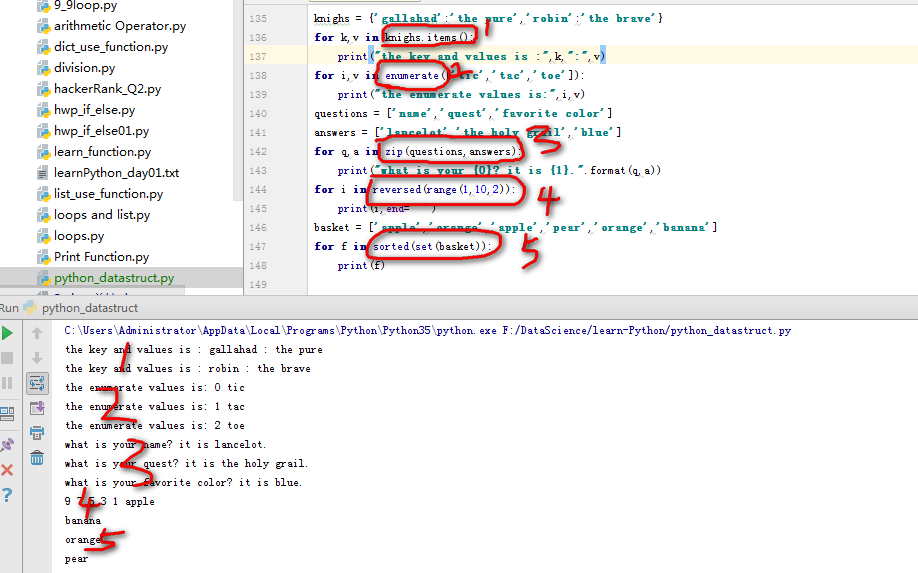
在字典中遍历时，关键字和对应的值可以使用 items() 方法同时解读出来：

在序列中遍历时，索引位置和对应值可以使用 enumerate() 函数同时得到：

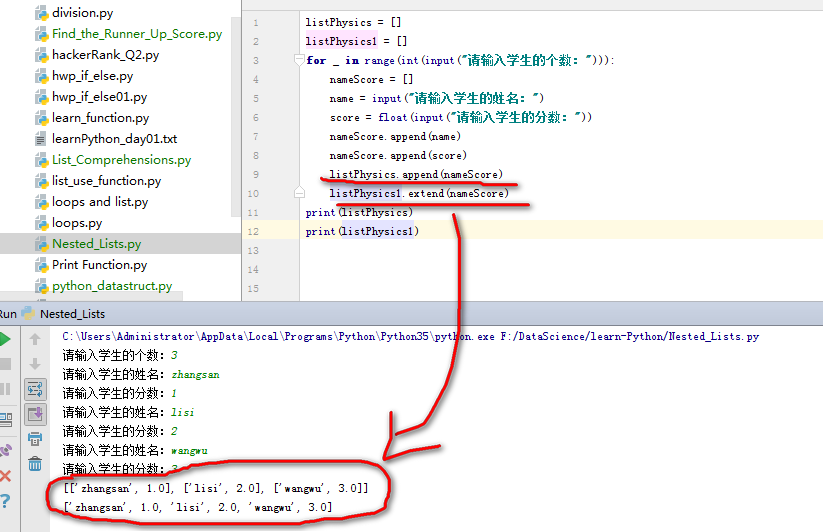
同时遍历两个或更多的序列，可以使用 zip() 组合：

要反向遍历一个序列，首先指定这个序列，然后调用 reversesd() 函数：

要按顺序遍历一个序列，使用 sorted() 函数返回一个已排序的序列，并不修改原值：

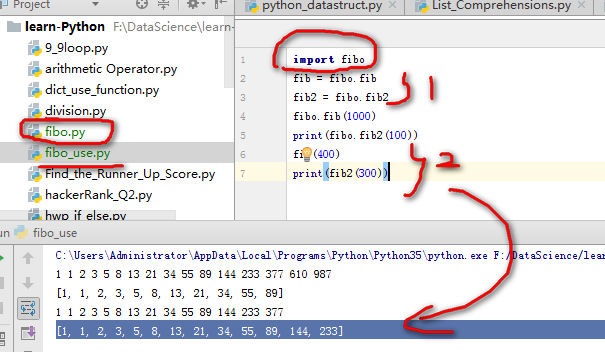


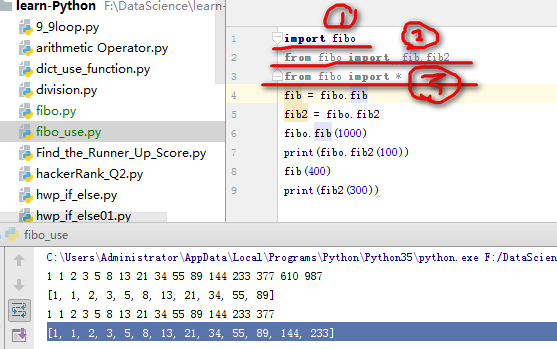
**注意比较 list中的append 和 extend 区别**

****

# Python 模块

模块是一个包含所有你定义的函数和变量的文件，其后缀名是.py

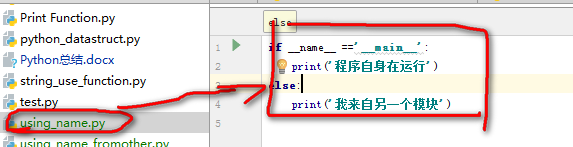
****

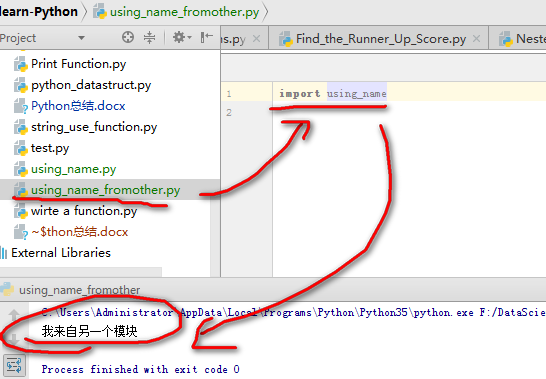
****

From 文件名 import\* 这将把所有的名字都导入进来，但是那些由**单一下划线（\_）开头的名字不在此例**。大多数情况， P**ython程序员不使用这种方法**，因为引入的其它来源的命名，很可能覆盖了已有的定义。

**name属性**

一个模块被另一个程序第一次引入时，其主程序将运行。如果我们想在模块被引入时，模块中的某一程序块不执行，我们可以用**name**属性来使该程序块仅在该模块自身运行时执行



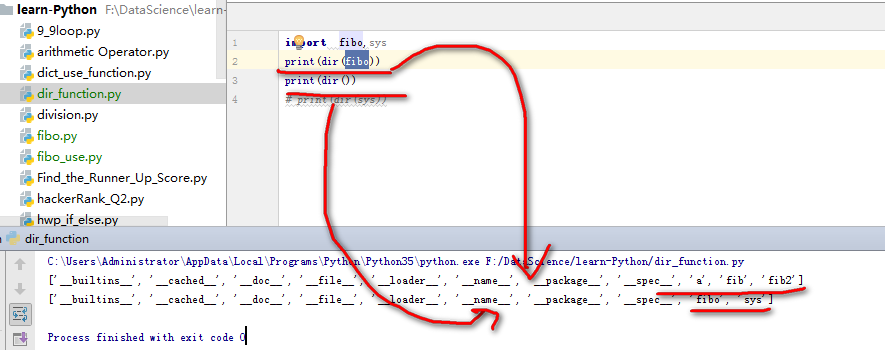
****

**说明：** 每个模块都有一个**name**属性，当其值是'**main**'时，表明该模块自身在运行，否则是被引入。

**dir() 函数**

内置的函数 dir() 可以找到模块内定义的所有名称。

如果没有给定参数，那么 dir() 函数会罗列出当前定义的所有名称:



**标准模块**

Python 本身带着一些标准的模块库

**包**

在导入一个包的时候，Python 会根据 sys.path 中的目录来寻找这个包中包含的子目录。

目录只有包含一个叫做 **init**.py 的文件才会被认作是一个包，主要是为了避免一些滥俗的名字（比如叫做 string）不小心的影响搜索路径中的有效模块。

最简单的情况，放一个空的 :file:**init**.py就可以了。当然这个文件中也可以包含一些初始化代码或者为（将在后面介绍的） **all**变量赋值。

用户可以每次只导入一个包里面的特定模块，比如

sound/ Top-level package

\_\_init\_\_.py Initialize the sound package

formats/ Subpackage for file format conversions

\_\_init\_\_.py

wavread.py

wavwrite.py

aiffread.py

aiffwrite.py

auread.py

auwrite.py

...

effects/ Subpackage for sound effects

\_\_init\_\_.py

echo.py

surround.py

reverse.py

...

filters/ Subpackage for filters

\_\_init\_\_.py

equalizer.py

vocoder.py

karaoke.py

...

import sound.effects.echo

from sound.effects import echo

from sound.effects.echo import echofilter

注意当使用from package import item这种形式的时候，对应的item既可以是包里面的子模块（子包），或者包里面定义的其他名称，比如函数，类或者变量。

import语法会首先把item当作一个包定义的名称，如果没找到，再试图按照一个模块去导入。如果还没找到，恭喜，一个:exc:ImportError 异常被抛出了。

反之，如果使用形如import item.subitem.subsubitem这种导入形式，除了最后一项，都必须是包，而最后一项则可以是模块或者是包，但是不可以是类，函数或者变量的名字。

**从一个包中导入\***

设想一下，如果我们使用 from sound.effects import \*会发生什么？

Python 会进入文件系统，找到这个包里面所有的子模块，一个一个的把它们都导入进来。

但是很不幸，这个方法在 Windows平台上工作的就不是非常好，因为Windows是一个大小写不区分的系统。

在这类平台上，没有人敢担保一个叫做 ECHO.py 的文件导入为模块:mod:echo还是:mod:Echo甚至:mod:ECHO。

为了解决这个问题，只能烦劳包作者提供一个精确的包的索引了。

导入语句遵循如下规则：如果包定义文件 **init**.py 存在一个叫做 **all** 的列表变量，那么在使用 from package import \* 的时候就把这个列表中的所有名字作为包内容导入。

作为包的作者，可别忘了在更新包之后保证 **all** 也更新了啊。你说我就不这么做，我就不使用导入\*这种用法，好吧，没问题，谁让你是老板呢。这里有一个例子，在:file:sounds/effects/**init**.py中包含如下代码:

\_\_all\_\_ = ["echo", "surround", "reverse"]

**通常我们并不主张使用\*这种方法来导入模块，因为这种方法经常会导致代码的可读性降低**。不过这样倒的确是可以省去不少敲键的功夫，而且一些模块都设计成了只能通过特定的方法导入。

记住，使用from Package import specific\_submodule这种方法永远不会有错。事实上，这也是推荐的方法。除非是你要导入的子模块有可能和其他包的子模块重名。

无论是隐式的还是显式的相对导入都是从当前模块开始的。主模块的名字永远是"**main**"，一个Python应用程序的主模块，应当总是使用绝对路径引用。

包还提供一个额外的属性，:attr:**path**。这是一个目录列表，里面每一个包含的目录都有为这个包服务的:file:**init**.py，你得在其他:file:**init**.py被执行前定义哦。可以修改这个变量，用来影响包含在包里面的模块和子包。

这个功能并不常用，一般用来扩展包里面的模块

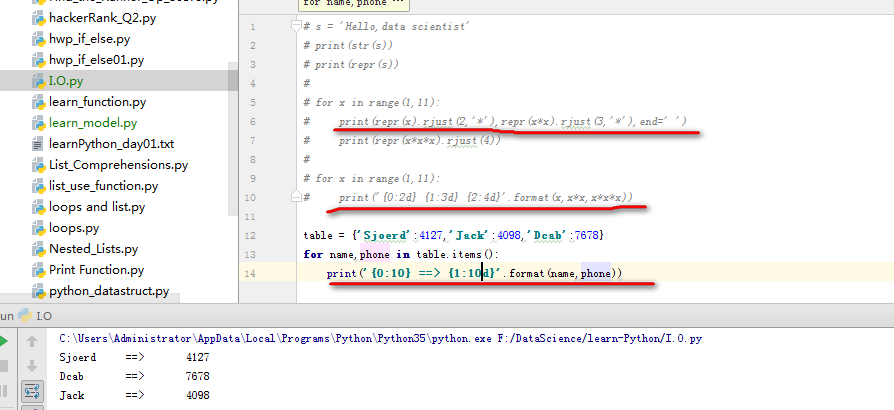
# Python 输入和输出

如果你希望输出的形式更加多样，可以使用 str.format() 函数来格式化输出值。

如果你希望将输出的值转成字符串，可以使用 repr() 或 str() 函数来实现。

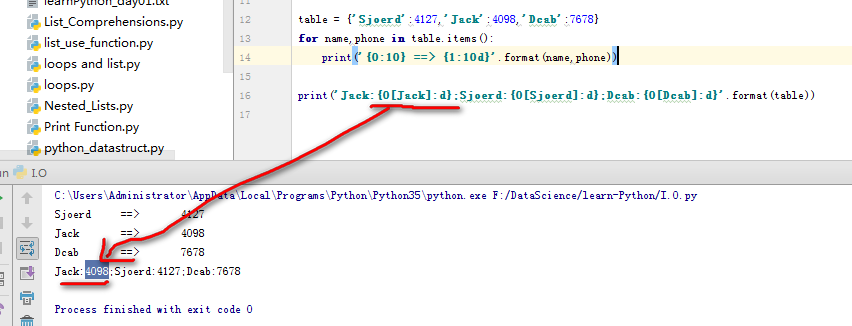
str() 函数返回一个用户易读的表达形式。

repr() 产生一个解释器易读的表达形式。



如果你有一个很长的格式化字符串, 而你不想将它们分开, 那么在格式化时通过变量名而非位置会是很好的事情。

最简单的就是传入一个字典, 然后使用方括号 '[]' 来访问键值 :



**读和写文件**

open(filename, mode)

f = open('/tmp/workfile', 'w')

* 第一个参数为要打开的文件名。
* 第二个参数描述文件如何使用的字符。 mode 可以是 'r' 如果文件只读, 'w' 只用于写 (如果存在同名文件则将被删除), 和 'a' 用于追加文件内容; 所写的任何数据都会被自动增加到末尾. 'r+' 同时用于读写。 mode 参数是可选的; 'r' 将是默认值。

### f.read()

为了读取一个文件的内容，调用 f.read(size), 这将读取一定数目的数据, 然后作为字符串或字节对象返回。

size 是一个可选的数字类型的参数。 当 size 被忽略了或者为负, 那么该文件的所有内容都将被读取并且返回。

>>> f.read()

'This is the entire file.\n'

>>> f.read()

''

### f.readline()

f.readline() 会从文件中读取单独的一行。换行符为 '\n'。f.readline() 如果返回一个空字符串, 说明已经已经读取到最后一行。

>>> f.readline()

'This is the first line of the file.\n'

>>> f.readline()

'Second line of the file\n'

>>> f.readline()

''

### f.readlines()

f.readlines() 将返回该文件中包含的所有行。

如果设置可选参数 sizehint, 则读取指定长度的字节, 并且将这些字节按行分割。

>>> f.readlines()

['This is the first line of the file.\n', 'Second line of the file\n']

另一种方式是迭代一个文件对象然后读取每行:

>>> for line in f:

... print(line, end='')

...

This is the first line of the file.

Second line of the file

这个方法很简单, 但是并没有提供一个很好的控制。 因为两者的处理机制不同, 最好不要混用。

### f.write()

f.write(string) 将 string 写入到文件中, 然后返回写入的字符数。

>>> f.write('This is a test\n')

15

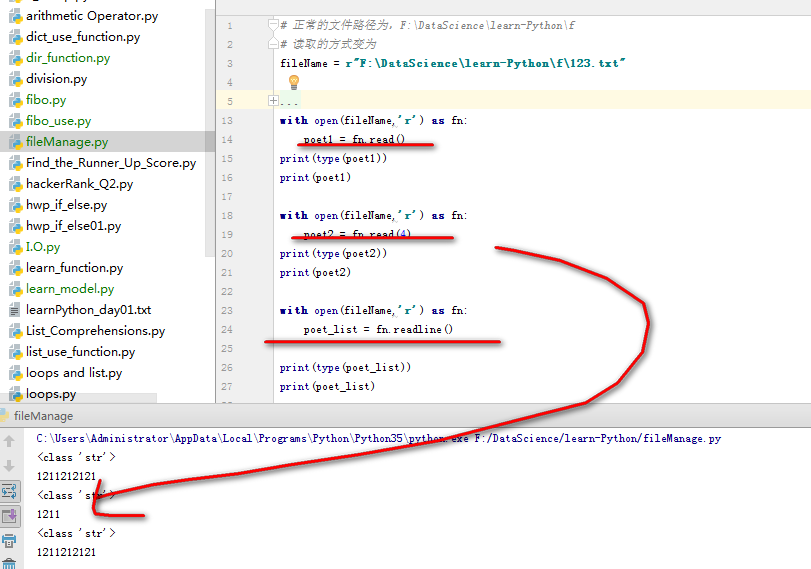
如果要写入一些不是字符串的东西, 那么将需要先进行转换:

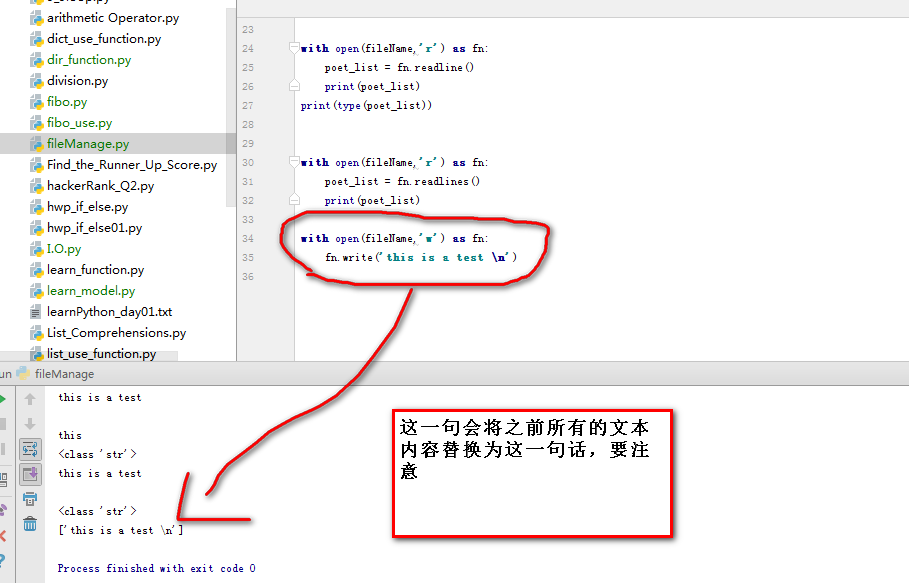
>>> value = ('the answer', 42)

>>> s = str(value)

>>> f.write(s)

18

****

****

### f.tell()

f.tell() 返回文件对象当前所处的位置, 它是从文件开头开始算起的字节数。

### f.close()

在文本文件中 (那些打开文件的模式下没有 b 的), 只会相对于文件起始位置进行定位。

当你处理完一个文件后, 调用 f.close() 来关闭文件并释放系统的资源，如果尝试再调用该文件，则会抛出异常。

当处理一个文件对象时, 使用 with 关键字是非常好的方式。在结束后, 它会帮你正确的关闭文件。

**pickle 模块**

python的pickle模块实现了基本的数据序列和反序列化。

通过pickle模块的序列化操作我们能够将程序中运行的对象信息保存到文件中去，永久存储。

通过pickle模块的反序列化操作，我们能够从文件中创建上一次程序保存的对象。

基本接口：

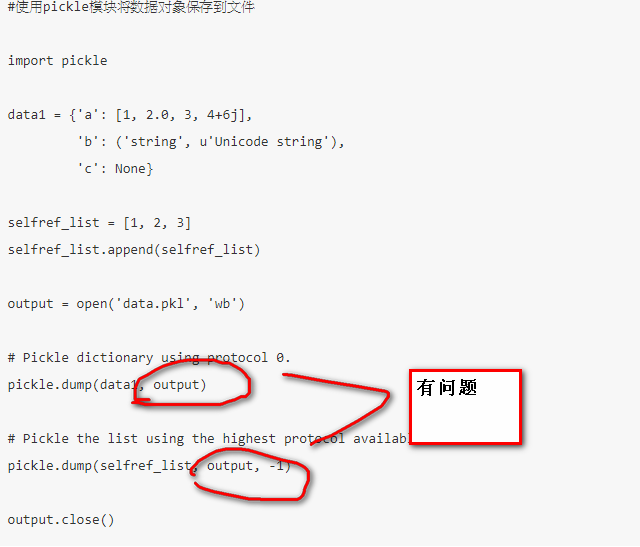
pickle.dump(obj, file, [,protocol])

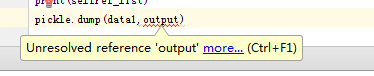
有了 pickle 这个对象, 就能对 file 以读取的形式打开:

x = pickle.load(file)

**注解：**从 file 中读取一个字符串，并将它重构为原来的python对象。

**file:** 类文件对象，有read()和readline()接口。

****

****

# Python 错误和异常

Python有两种错误很容易辨认：语法错误和异常。

**python标准异常**

| **异常名称** | **描述** |
| --- | --- |
| BaseException | 所有异常的基类 |
| SystemExit | 解释器请求退出 |
| KeyboardInterrupt | 用户中断执行(通常是输入^C) |
| Exception | 常规错误的基类 |
| StopIteration | 迭代器没有更多的值 |
| GeneratorExit | 生成器(generator)发生异常来通知退出 |
| SystemExit | Python 解释器请求退出 |
| StandardError | 所有的内建标准异常的基类 |
| ArithmeticError | 所有数值计算错误的基类 |
| FloatingPointError | 浮点计算错误 |
| OverflowError | 数值运算超出最大限制 |
| ZeroDivisionError | 除(或取模)零 (所有数据类型) |
| AssertionError | 断言语句失败 |
| AttributeError | 对象没有这个属性 |
| EOFError | 没有内建输入,到达EOF 标记 |
| EnvironmentError | 操作系统错误的基类 |
| IOError | 输入/输出操作失败 |
| OSError | 操作系统错误 |
| WindowsError | 系统调用失败 |
| ImportError | 导入模块/对象失败 |
| KeyboardInterrupt | 用户中断执行(通常是输入^C) |
| LookupError | 无效数据查询的基类 |
| IndexError | 序列中没有没有此索引(index) |
| KeyError | 映射中没有这个键 |
| MemoryError | 内存溢出错误(对于Python 解释器不是致命的) |
| NameError | 未声明/初始化对象 (没有属性) |
| UnboundLocalError | 访问未初始化的本地变量 |
| ReferenceError | 弱引用(Weak reference)试图访问已经垃圾回收了的对象 |
| RuntimeError | 一般的运行时错误 |
| NotImplementedError | 尚未实现的方法 |
| SyntaxError | Python 语法错误 |
| IndentationError | 缩进错误 |
| TabError | Tab 和空格混用 |
| SystemError | 一般的解释器系统错误 |
| TypeError | 对类型无效的操作 |
| ValueError | 传入无效的参数 |
| UnicodeError | Unicode 相关的错误 |
| UnicodeDecodeError | Unicode 解码时的错误 |
| UnicodeEncodeError | Unicode 编码时错误 |
| UnicodeTranslateError | Unicode 转换时错误 |
| Warning | 警告的基类 |
| DeprecationWarning | 关于被弃用的特征的警告 |
| FutureWarning | 关于构造将来语义会有改变的警告 |
| OverflowWarning | 旧的关于自动提升为长整型(long)的警告 |
| PendingDeprecationWarning | 关于特性将会被废弃的警告 |
| RuntimeWarning | 可疑的运行时行为(runtime behavior)的警告 |
| SyntaxWarning | 可疑的语法的警告 |
| UserWarning | 用户代码生成的警告 |

**异常处理**

try语句按照如下方式工作；

* 首先，执行try子句（在关键字try和关键字except之间的语句）
* 如果没有异常发生，忽略except子句，try子句执行后结束。
* 如果在执行try子句的过程中发生了异常，那么try子句余下的部分将被忽略。如果异常的类型和 except 之后的名称相符，那么对应的except子句将被执行。最后执行 try 语句之后的代码。
* 如果一个异常没有与任何的except匹配，那么这个异常将会传递给上层的try中。

一个 try 语句可能包含多个except子句，分别来处理不同的特定的异常。最多只有一个分支会被执行。

处理程序将只针对对应的try子句中的异常进行处理，而不是其他的 try 的处理程序中的异常。

一个except子句可以同时处理多个异常，这些异常将被放在一个括号里成为一个元组，例如:

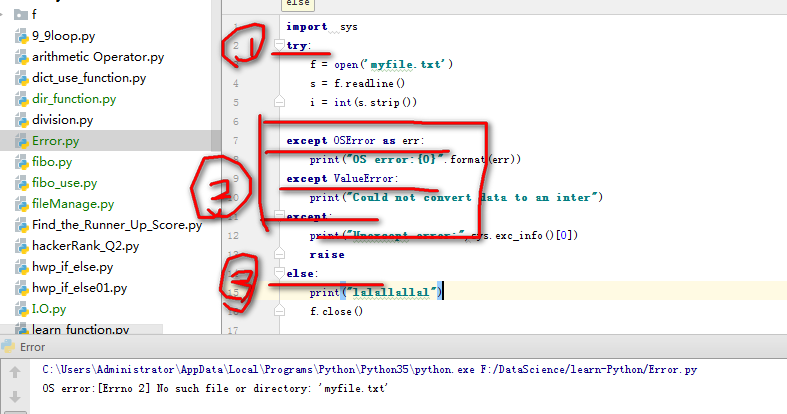
except (RuntimeError, TypeError, NameError):

pass

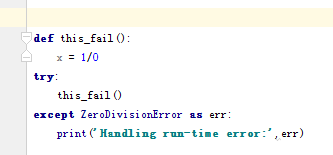
最后一个except子句可以忽略异常的名称，它将被当作通配符使用。你可以使用这种方法打印一个错误信息，然后再次把异常抛出。

try except 语句还有一个可选的else子句，如果使用这个子句，那么必须放在所有的except子句之后。这个子句将在try子句没有发生任何异常的时候执行。

使用 else 子句比把所有的语句都放在 try 子句里面要好，这样可以避免一些意想不到的、而except又没有捕获的异常。

****

异常处理并不仅仅处理那些直接发生在try子句中的异常，而且还能处理子句中调用的函数（甚至间接调用的函数）里抛出的异常。例如

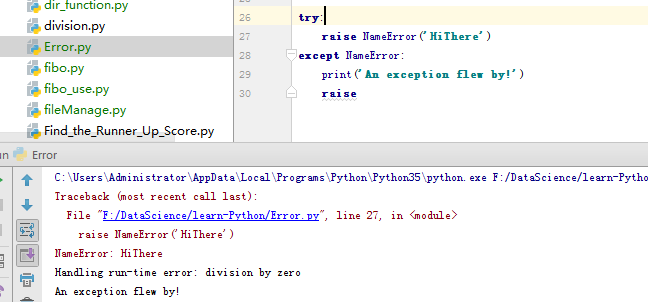
****

**抛出异常**

Python 使用 raise 语句抛出一个指定的异常

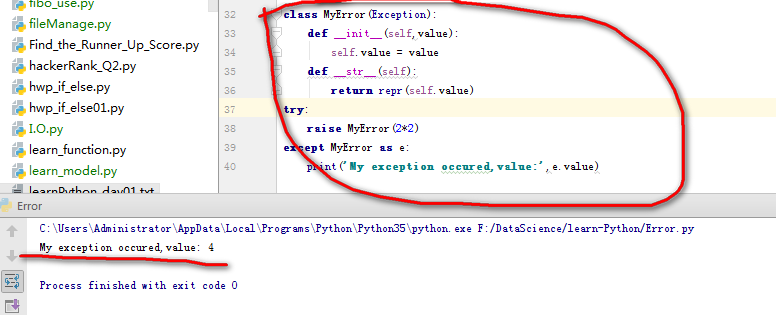
raise 唯一的一个参数指定了要被抛出的异常。它必须是一个异常的实例或者是异常的类（也就是 Exception 的子类）。

如果你只想知道这是否抛出了一个异常，并不想去处理它，那么一个简单的 raise 语句就可以再次把它抛出。

****

**用户自定义异常**

你可以通过创建一个新的exception类来拥有自己的异常。异常应该继承自 Exception 类，或者直接继承，或者间接继承，例如:



当创建一个模块有可能抛出多种不同的异常时，一种通常的做法是为这个包建立一个基础异常类，然后基于这个基础类为不同的错误情况创建不同的子类:

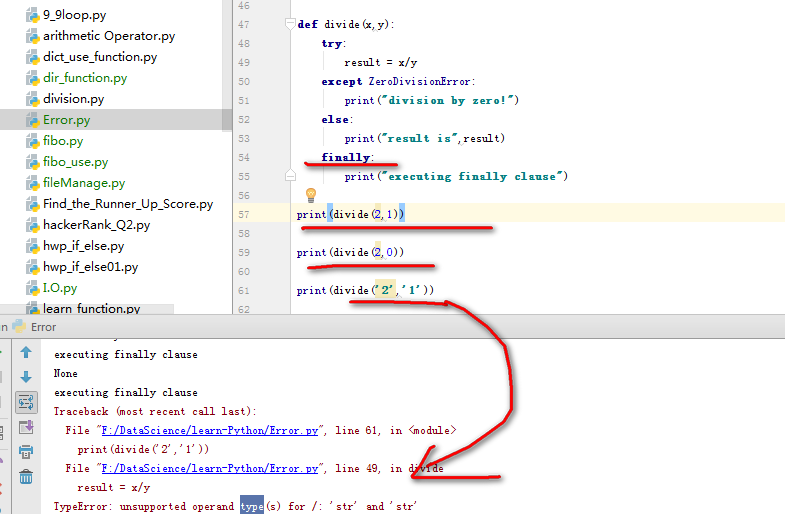
**定义清理行为**

try 语句还有另外一个可选的子句，它定义了无论在任何情况下都会执行的清理行为。 例如

以上例子洪不管try子句里面有没有发生异常，finally子句都会执行。

如果一个异常在 try 子句里（或者在 except 和 else 子句里）被抛出，而又没有任何的 except 把它截住，那么这个异常会在 finally 子句执行后再次被抛出。

下面是一个更加复杂的例子（在同一个 try 语句里包含 except 和 finally 子句）:

****

**预定义的清理行为**

一些对象定义了标准的清理行为，无论系统是否成功的使用了它，一旦不需要它了，那么这个标准的清理行为就会执行。

这面这个例子展示了尝试打开一个文件，然后把内容打印到屏幕上:

for line in open("myfile.txt"):

print(line, end="")

以上这段代码的问题是，当执行完毕后，文件会保持打开状态，并没有被关闭。

关键词 with 语句就可以保证诸如文件之类的对象在使用完之后一定会正确的执行他的清理方法:

with open("myfile.txt") as f:

for line in f:

print(line, end="")

以上这段代码执行完毕后，就算在处理过程中出问题了，文件 f 总是会关闭。

# Python 类

和其它编程语言相比，Python 在尽可能不增加新的语法和语义的情况下加入了类机制。

Python中的类提供了面向对象编程的所有基本功能：类的继承机制允许多个基类，派生类可以覆盖基类中的任何方法，方法中可以调用基类中的同名方法。

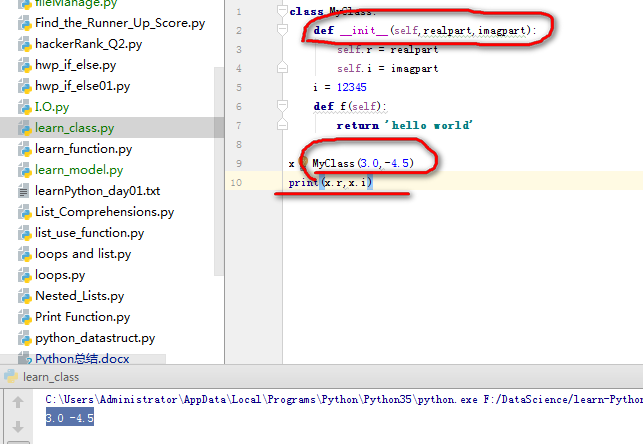
对象可以包含任意数量和类型的数据。

类对象支持两种操作：属性引用和实例化

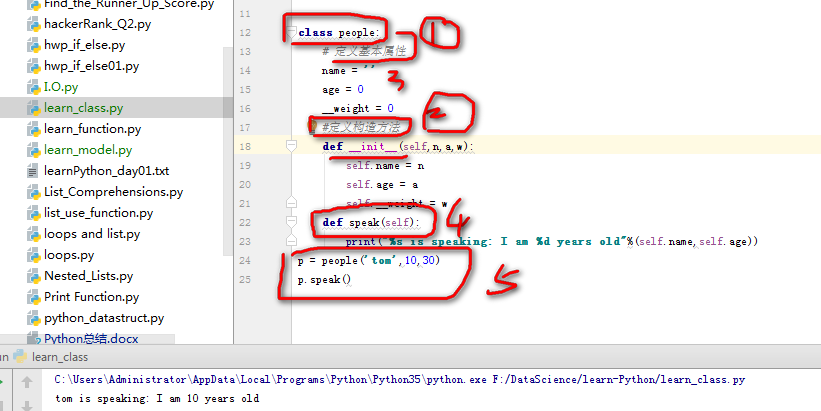
以上创建了一个新的类实例并将该对象赋给局部变量 x，x 为空的对象。

很多类都倾向于将对象创建为有初始状态的。因此类可能会定义一个名为 **init**() 的特殊方法（构造方法），像下面这样：

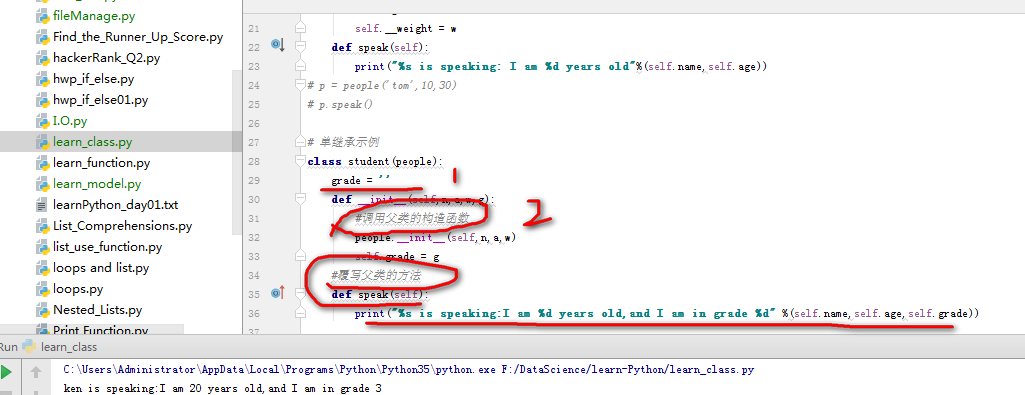
当然， **init**() 方法可以有参数，参数通过 **init**() 传递到类的实例化操作上。例如

****

在类地内部，使用def关键字可以为类定义一个方法，与一般函数定义不同，类方法必须包含参数self,且为第一个参数:

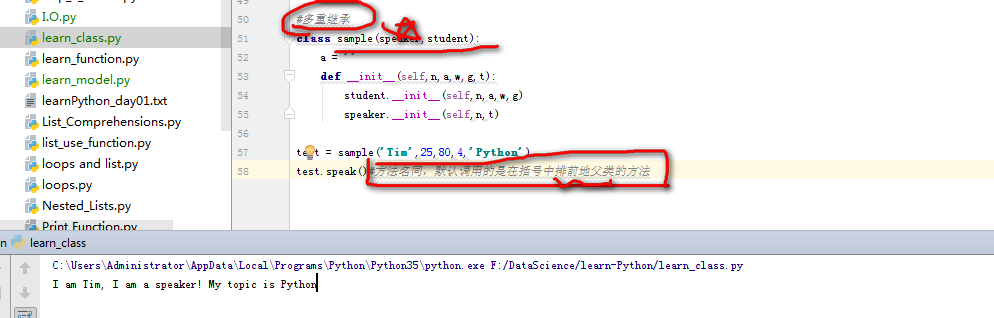
****

**继承**

****

**多重继承**

Python同样有限的支持多继承形式。多继承的类定义形如下例:

****

**类私有方法**

\_\_private\_method 两个下划线开头，声明该方法为私有方法，不能在类地外部调用。

在类的内部调用slef.\_\_private\_methods。

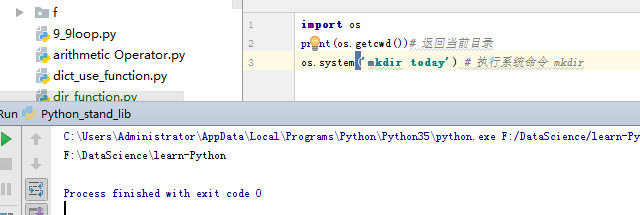
类的专有方法：

* **init** 构造函数，在生成对象时调用
* **del** 析构函数，释放对象时使用
* **repr** 打印，转换
* **setitem**按照索引赋值
* **getitem**按照索引获取值
* **len**获得长度
* **cmp**比较运算
* **call**函数调用
* **add**加运算
* **sub**减运算
* **mul**乘运算
* **div**除运算
* **mod**求余运算
* **pow**称方

# Python 标准库概览

**操作系统接口**

os模块提供了不少与操作系统相关联的函数



建议使用 "import os" 风格而非 "from os import \*"。这样可以保证随操作系统不同而有所变化的 os.open() 不会覆盖内置函数 open()。

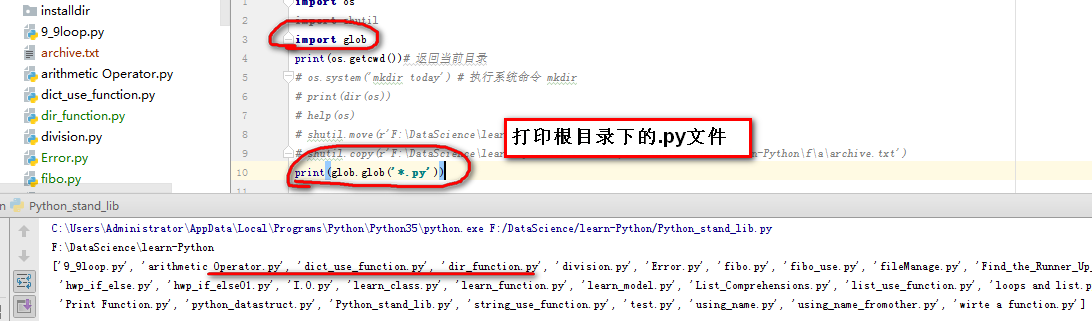
在使用 os 这样的大型模块时内置的 dir() 和 help() 函数非常有用:

针对日常的文件和目录管理任务，:mod:shutil 模块提供了一个易于使用的高级接口:

****

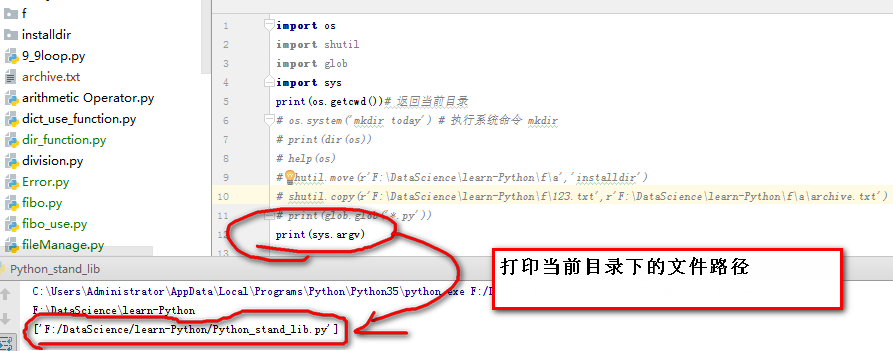
**文件通配符**

glob模块提供了一个函数用于从目录通配符搜索中生成文件列表:

****

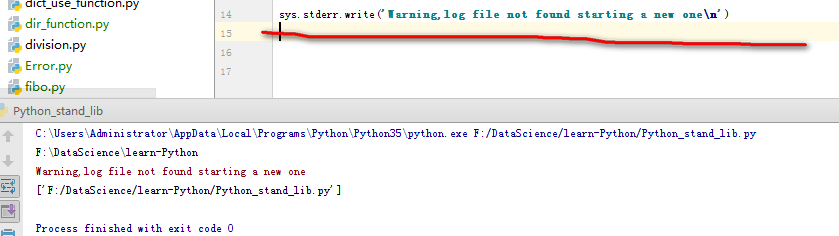
**命令行参数**

通用工具脚本经常调用命令行参数。这些命令行参数以链表形式存储于 sys 模块的 argv 变量。例如在命令行中执行 "python demo.py one two three" 后可以得到以下输出结果:

****

**错误输出重定向和程序终止**

sys 还有 stdin，stdout 和 stderr 属性，即使在 stdout 被重定向时，后者也可以用于显示警告和错误信息。



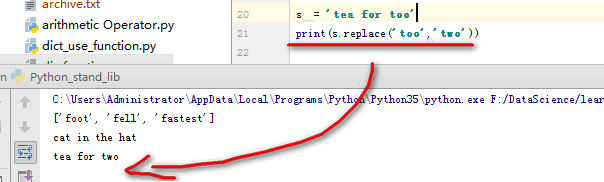
大多脚本的定向终止都使用 "sys.exit()"。

**字符串正则匹配**

re模块为高级字符串处理提供了正则表达式工具。对于复杂的匹配和处理，正则表达式提供了简洁、优化的解决方案:



如果只需要简单的功能，应该首先考虑字符串方法，因为它们非常简单，易于阅读和调试



## 正则表达式模式

模式字符串使用特殊的语法来表示一个正则表达式：

字母和数字表示他们自身。一个正则表达式模式中的字母和数字匹配同样的字符串。

多数字母和数字前加一个反斜杠时会拥有不同的含义。

标点符号只有被转义时才匹配自身，否则它们表示特殊的含义。

反斜杠本身需要使用反斜杠转义。

由于正则表达式通常都包含反斜杠，所以你最好使用原始字符串来表示它们。模式元素(如 r'/t'，等价于'//t')匹配相应的特殊字符。

下表列出了正则表达式模式语法中的特殊元素。如果你使用模式的同时提供了可选的标志参数，某些模式元素的含义会改变。

| **模式** | **描述** |
| --- | --- |
| ^ | 匹配字符串的开头 |
| $ | 匹配字符串的末尾。 |
| . | 匹配任意字符，除了换行符，当re.DOTALL标记被指定时，则可以匹配包括换行符的任意字符。 |
| [...] | 用来表示一组字符,单独列出：[amk] 匹配 'a'，'m'或'k' |
| [...](https://wizardforcel.gitbooks.io/w3school-python/content/25.html#fn_...) | 不在[]中的字符：[abc](https://wizardforcel.gitbooks.io/w3school-python/content/25.html#fn_abc) 匹配除了a,b,c之外的字符。 |
| re\* | 匹配0个或多个的表达式。 |
| re+ | 匹配1个或多个的表达式。 |
| re? | 匹配0个或1个由前面的正则表达式定义的片段，贪婪方式 |
| re{ n} |  |
| re{ n,} | 精确匹配n个前面表达式。 |
| re{ n, m} | 匹配 n 到 m 次由前面的正则表达式定义的片段，贪婪方式 |
| a | b | 匹配a或b |
| (re) | G匹配括号内的表达式，也表示一个组 |
| (?imx) | 正则表达式包含三种可选标志：i, m, 或 x 。只影响括号中的区域。 |
| (?-imx) | 正则表达式关闭 i, m, 或 x 可选标志。只影响括号中的区域。 |
| (?: re) | 类似 (...), 但是不表示一个组 |
| (?imx: re) | 在括号中使用i, m, 或 x 可选标志 |
| (?-imx: re) | 在括号中不使用i, m, 或 x 可选标志 |
| (?#...) | 注释. |
| (?= re) | 前向肯定界定符。如果所含正则表达式，以 ... 表示，在当前位置成功匹配时成功，否则失败。但一旦所含表达式已经尝试，匹配引擎根本没有提高；模式的剩余部分还要尝试界定符的右边。 |
| (?! re) | 前向否定界定符。与肯定界定符相反；当所含表达式不能在字符串当前位置匹配时成功 |
| (?> re) | 匹配的独立模式，省去回溯。 |
| \w | 匹配字母数字 |
| \W | 匹配非字母数字 |
| \s | 匹配任意空白字符，等价于 [\t\n\r\f]. |
| \S | 匹配任意非空字符 |
| \d | 匹配任意数字，等价于 [0-9]. |
| \D | 匹配任意非数字 |
| \A | 匹配字符串开始 |
| \Z | 匹配字符串结束，如果是存在换行，只匹配到换行前的结束字符串。c |
| \z | 匹配字符串结束 |
| \G | 匹配最后匹配完成的位置。 |
| \b | 匹配一个单词边界，也就是指单词和空格间的位置。例如， 'er\b' 可以匹配"never" 中的 'er'，但不能匹配 "verb" 中的 'er'。 |
| \B | 匹配非单词边界。'er\B' 能匹配 "verb" 中的 'er'，但不能匹配 "never" 中的 'er'。 |
| \n, \t, 等. | 匹配一个换行符。匹配一个制表符。等 |
| \1...\9 | 比赛第n个分组的子表达式。 |
| \10 | 匹配第n个分组的子表达式，如果它经匹配。否则指的是八进制字符码的表达式。 |

## 正则表达式实例

#### 字符匹配

| **实例** | **描述** |
| --- | --- |
| python | 匹配 "python". |

#### 字符类

| **实例** | **描述** |
| --- | --- |
| [Pp]ython | 匹配 "Python" 或 "python" |
| rub[ye] | 匹配 "ruby" 或 "rube" |
| [aeiou] | 匹配中括号内的任意一个字母 |
| [0-9] | 匹配任何数字。类似于 [0123456789] |
| [a-z] | 匹配任何小写字母 |
| [A-Z] | 匹配任何大写字母 |
| [a-zA-Z0-9] | 匹配任何字母及数字 |
| [aeiou](https://wizardforcel.gitbooks.io/w3school-python/content/25.html#fn_aeiou) | 除了aeiou字母以外的所有字符 |
| [0-9](https://wizardforcel.gitbooks.io/w3school-python/content/25.html#fn_0-9) | 匹配除了数字外的字符 |

#### 特殊字符类

| **实例** | **描述** |
| --- | --- |
| . | 匹配除 "\n" 之外的任何单个字符。要匹配包括 '\n' 在内的任何字符，请使用象 '[.\n]' 的模式。 |
| \d | 匹配一个数字字符。等价于 [0-9]。 |
| \D | 匹配一个非数字字符。等价于 [0-9](https://wizardforcel.gitbooks.io/w3school-python/content/25.html#fn_0-9)。 |
| \s | 匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [ \f\n\r\t\v]。 |
| \S | 匹配任何非空白字符。等价于 [\f\n\r\t\v](https://wizardforcel.gitbooks.io/w3school-python/content/25.html#fn_ \f\n\r\t\v)。 |
| \w | 匹配包括下划线的任何单词字符。等价于'[A-Za-z0-9\_]'。 |
| \W | 匹配任何非单词字符。等价于 '[A-Za-z0-9\_](https://wizardforcel.gitbooks.io/w3school-python/content/25.html#fn_A-Za-z0-9_)'。 |

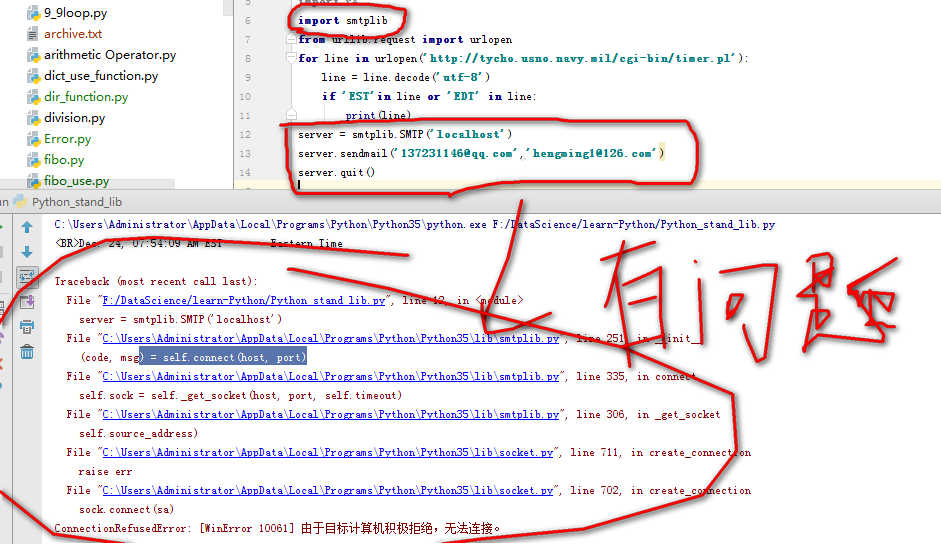
**数学**

random提供了生成随机数的工具

math模块为浮点运算提供了对底层C函数库的访问:

**访问 互联网**

有几个模块用于访问互联网以及处理网络通信协议。其中最简单的两个是用于处理从 urls 接收的数据的 urllib.request 以及用于发送电子邮件的 smtplib:

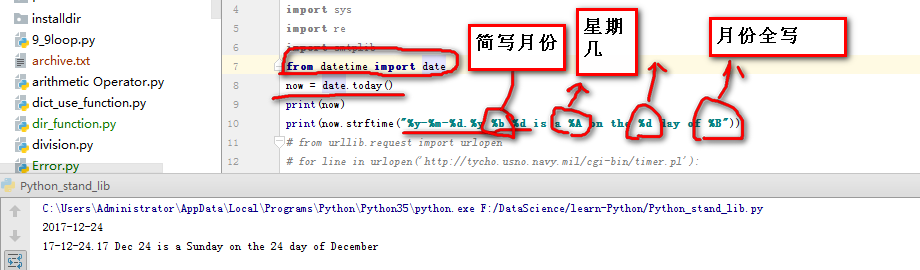


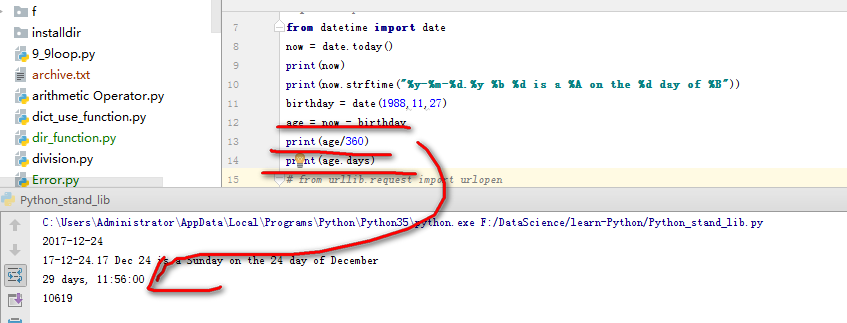
**日期和时间**

datetime模块为日期和时间处理同时提供了简单和复杂的方法。

支持日期和时间算法的同时，实现的重点放在更有效的处理和格式化输出。

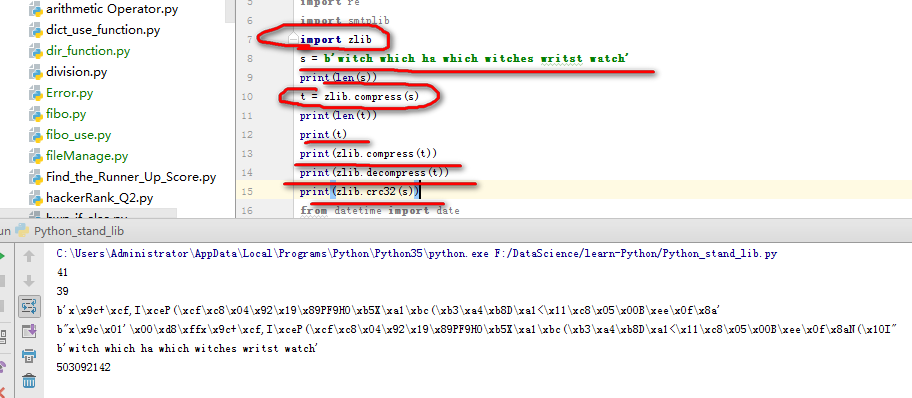
该模块还支持时区处理:





**数据压缩**

以下模块直接支持通用的数据打包和压缩格式：zlib，gzip，bz2，zipfile，以及 tarfile。



**性能度量**

有些用户对了解解决同一问题的不同方法之间的性能差异很感兴趣。Python 提供了一个度量工具，为这些问题提供了直接答案。

例如，使用元组封装和拆封来交换元素看起来要比使用传统的方法要诱人的多,timeit 证明了现代的方法更快一些。



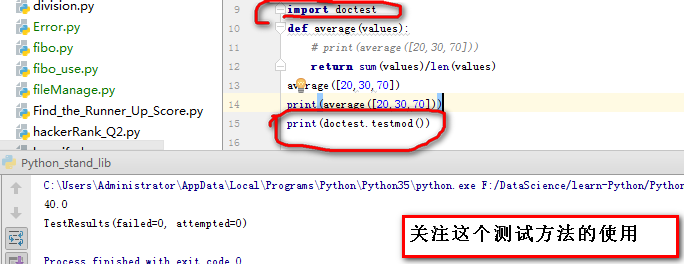
**测试模块**

开发高质量软件的方法之一是为每一个函数开发测试代码，并且在开发过程中经常进行测试

doctest模块提供了一个工具，扫描模块并根据程序中内嵌的文档字符串执行测试。

测试构造如同简单的将它的输出结果剪切并粘贴到文档字符串中。

通过用户提供的例子，它强化了文档，允许 doctest 模块确认代码的结果是否与文档一致:



unittest模块不像 doctest模块那么容易使用，不过它可以在一个独立的文件里提供一个更全面的测试集：

