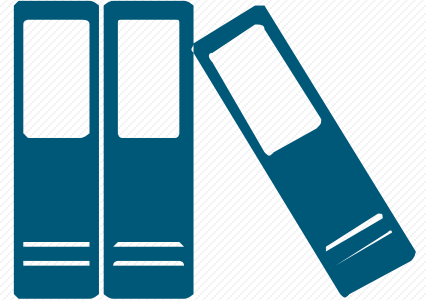


序列数据类型：列表与元组



问题:

- 输入三个成绩，计算平均值。

```
a, b, c = eval(input())  
aver = (a + b + c) / 3
```

- 输入一组数据，计算平均值？

```
n = eval(input())  
s=0  
for i in range(n):  
    x = eval(input("请输入第{}个数:".format(i+1)))  
    s += x  
aver = s/n
```

- **输出所有高于平均分的成绩?**

需要在计算出平均成绩后，将每一个成绩与平均成绩进行比较。

如何存储? ----- 列表 (list)—— 一种序列类型

解决：从一个数据 ----> 一组数据

```
nlst = eval(input())
s=0
for item in nlst:
    s += item
aver = s/len(nlst)
for item in nlst:
    if item>=aver:
        print(item,end=' ')
```

序列类型sequence

序列是具有先后关系的一组元素

- 序列是一维元素向量，元素类型可以不同
- 类似数学元素序列： s_0, s_1, \dots, s_{n-1}
- 元素间由序号引导，通过下标访问序列的特定元素 index
- 序列的最大特点是元素的有序性，所以序列都是通过序号索引来访问元素的。

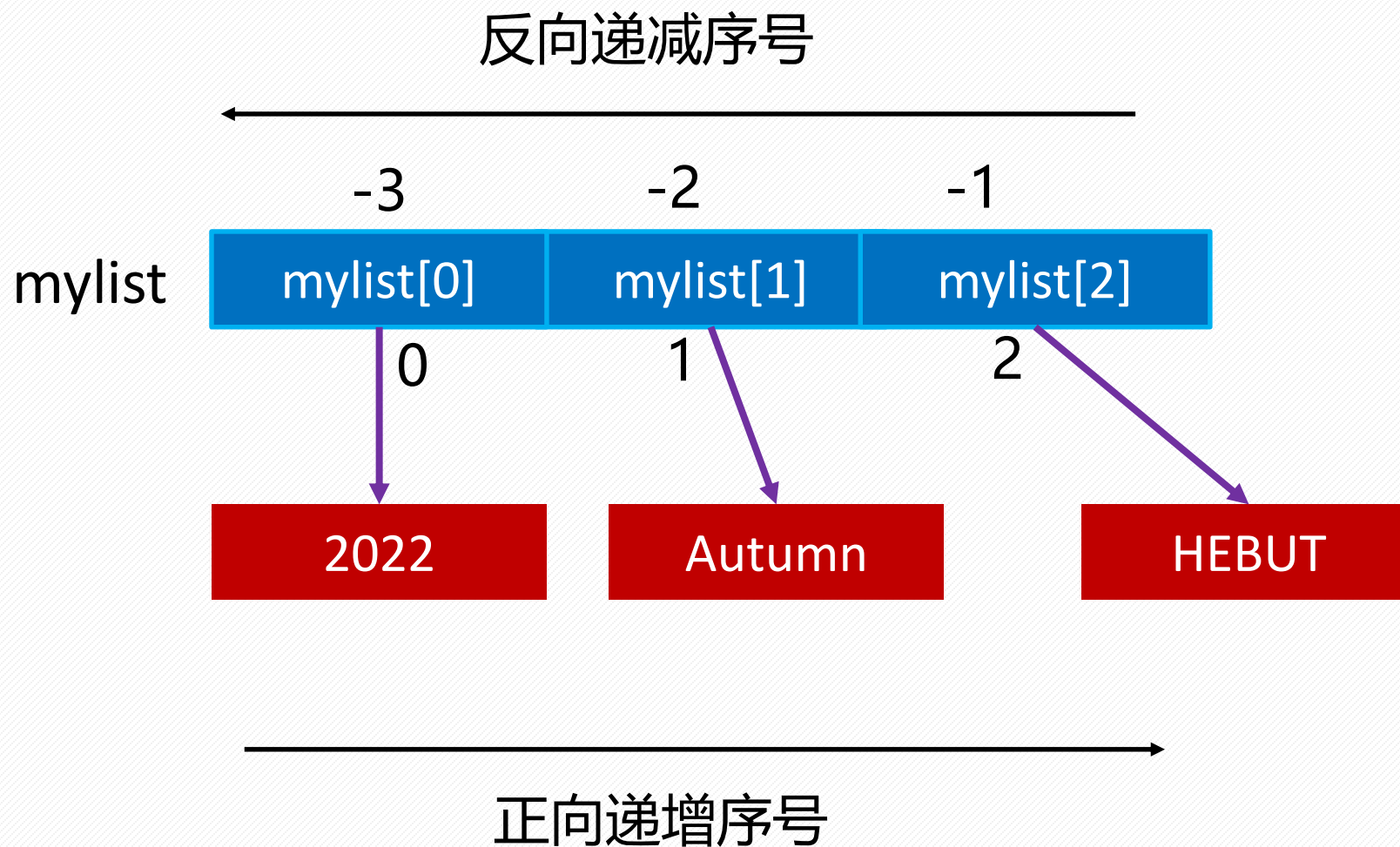
序列类型

字符串类型str mystr = "2022 Autumn HEBUT"

列表类型list mylist = [2022, "Autumn", "HEBUT"]

元组类型tuple mytuple = (2022, "Autumn", "HEBUT")

```
mylist = [2022, "Autumn", "HEBUT"]
```



序列类型通用操作符

6个操作符

操作符及应用	描述
<code>x in s</code>	如果x是序列s的元素，返回True，否则返回False
<code>x not in s</code>	如果x是序列s的元素，返回False，否则返回True
<code>s + t</code>	连接两个序列s和t
<code>s*n</code> 或 <code>n*s</code>	将序列s复制n次
<code>s[i]</code>	索引，返回s中的第i个元素
<code>s[i: j]</code> 或 <code>s[i: j: k]</code>	切片，返回s中第i到j(不包含第j个元素) 以k为步长的元素子序列

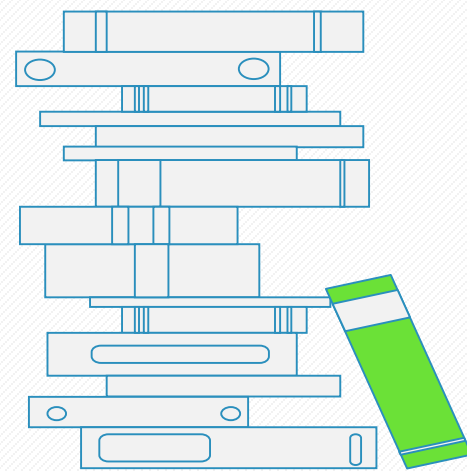
序列类型通用函数和方法

5个函数和方法

函数和方法	描述
<code>len(s)</code>	返回序列s的长度，即元素个数
<code>min(s)</code>	返回序列s的最小元素（s中元素需要可比较）
<code>max(s)</code>	返回序列s的最大元素（s中元素需要可比较）
<code>s.index(x)</code> 或 <code>s.index(x, i, j)</code>	返回序列s从i开始到j位置中第一次出现元素x的位置
<code>s.count(x)</code>	返回序列s中出现x的总次数



列表与列表操作



- 列表 (list) 用来**有序**存放一组相关数据，以便进行统一的处理。
- Python中，将**一组数据**放在一对方括号 “[]” 中即定义了一个列表。



- 列表的长度和内容都是**可变**的，可自由对列表中数据项进行增加、删除或替换。
列表没有长度限制。
- 一个列表中的**元素数据类型可以不同**，使用非常灵活。
- 最主要作用：**表示一组有序数据**，进而操作它们。

- 可以存放一组成绩

```
>>> scores = [98, 96, 95, 94, 92]
```

- 可以存放一组人名

```
>>> names = ["萧峰", "杨过", "令狐冲", "张无忌", "郭靖"]
```

- 可以存放一个学生的基本信息

```
>>> student = ['张三', 18, '江苏南京']
```

允许列表元素类型互不相同

允许列表嵌套定义

- 可以存放若干学生的基本信息

```
>>> students = [['张三', 18, '江苏南京'], ['李四', 19, '山东济南']]
```

- 可以定义嵌套的列表

```
>>> player1 = ["萧峰", 98]
>>> player2 = ["杨过", 96]
>>> player3 = ["令狐冲", 95]
>>> player4 = ["张无忌", 94]
>>> player5 = ["郭靖", 92]
>>> players = [player1, player2, player3, player4, player5]
>>> players
[['萧峰', 98], ['杨过', 96], ['令狐冲', 95], ['张无忌', 94], ['郭靖', 92]]
```

列表操作方法

方法	描述
ls.append(x)	在列表ls最后增加一个元素x
ls.insert(i,x)	在列表ls的第i位置增加元素x
ls.clear()	删除列表ls中所有元素
ls.pop(i)	将列表ls中第i位置元素取出并删除该元素
ls.remove(x)	将列表ls中出现的第一个元素x删除
ls.reverse()	将列表ls中的元素反转
ls.copy()	生成一个新列表，赋值ls中所有元素
ls.sort()	将列表ls进行升序排序(原地排序)

创建列表

➤ 创建空列表： `list()` , `[]`

```
>>> t = list()  
>>> t  
[]
```

```
>>> t=[]  
>>> type(t)  
<class 'list'>
```

list函数可以将range对象转换为列表

➤ 由list函数和字符串得到一个列表:

```
>>> t = list("abc")  
>>> t  
['a', 'b', 'c']
```

➤ 由list函数和range函数得到一个列表:

```
>>> range(3)  
range(0, 3)  
>>> list(range(3))  
[0, 1, 2]
```

range函数无法直接生成列表,
可利用list函数将range对象转换为列表



思考: 1. 用list()可以将字符串转换成列表
2. 如果对一篇英文短文分词, 用list函数可以吗?
那用什么方法呢?

➤ 通过字符串的split方法，可将由某个字符间隔的字符串分解为列表

```
>>> frtstr = 'apple,banana,peach'
>>> frtlist = frtstr.split(',') #逗号分割
>>> frtlist
['apple', ' banana', ' peach']
```

```
>>> nums = '1,2,3,4,5'
>>> numlist = nums.split(',') #逗号分割
>>> numlist
['1', '2', '3', '4', '5']
```

```
>>> frtstr = 'apple banana peach'
>>> frtlist = frtstr.split() #空白字符：空格，回车符，制表符
>>> frtlist
['apple', 'banana', 'peach']
```

提醒：使用字符串的split方法时，常用的是：str.split()
此时是将空白字符作为分隔符：包括空格，\t, \n。
尽量不要用str.split(' ')。

```
>>> x = "a  b\tc"  
>>> x.split(' ')  
['a', '', '', '', 'b\tc']
```



```
>>> x = "a  b\tc"  
>>> x.split()  
['a', 'b', 'c']
```

➤ 输入列表

- 按列表的形式直接输入，而后使用eval函数进行转换：

```
>>> values = eval(input())
```

```
[1,2,3,4,5]
```

```
>>> values
```

```
[1, 2, 3, 4, 5]
```

- 数字类型：使用逗号分隔输入一组数据，而后利用eval函数和list函数转换为列表

```
>>> values = eval(input())
```

```
1,2,3,4,5
```

```
>>> values
```

```
(1, 2, 3, 4,5)
```

```
>>> vlst = list(values)
```

```
>>> vlst
```

```
[1, 2, 3, 4, 5]
```

元组

在末尾增加元素：append


```
>>> t.append(1)
>>> t
[1]
>>> t.append(2)
>>> t
[1, 2]
```

```
>>> t = [1, 2, 3]
>>> t.append("abc")
>>> t
[1, 2, 3, 'abc']
>>> t.append(["def"])
>>> t
[1, 2, 3, 'abc', ['def']]
```

向列表中的某个位置增加元素

列表名.insert(位置, 新元素值)

```
>>> t = [1, 2, 3, 'abc', ['def']]
>>> t.insert(3, 4)
>>> t
[1, 2, 3, 4, 'abc', ['def']]
```



第一个参数指定新元素插入位置

```
>>> t
[1, 2, 3, 4, 'abc', ['def']]
>>> t.insert(3, [1, 2])
>>> t
[1, 2, 3, [1, 2], 4, 'abc', ['def']]
```

对列表进行扩充

extend, +

```
>>> t = [1, 2, 3]
>>> s = ['cat', 'dog', 'duck']
>>> t + s
[1, 2, 3, 'cat', 'dog', 'duck']
>>> t
[1, 2, 3]
>>> s
['cat', 'dog', 'duck']
```

“+” 运算将两个列表连接**生成一个新列表**，而不改变参与运算的列表本身。

将参数列表添加至原列表中

```
>>> t = []
>>> t.extend(['a', 'b', 'c'])
>>> t
['a', 'b', 'c']
>>> t.extend("abc")
>>> t
['a', 'b', 'c', 'a', 'b', 'c']
>>> t.extend(123)
```

```
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: 'int' object is not iterable
```

提醒： extend方法中的**参数**必须为可迭代类型。

对列表进行扩充：*

"*" 运算，通过重复指定遍数扩充列表长度

与 "+" 运算类似， "*" 运算也生成新列表、而不改变参与运算的列表本身。

```
>>> t = [1, 2, 3]
>>> t * 2
[1, 2, 3, 1, 2, 3]
>>> t
[1, 2, 3]
```

访问列表元素

列表元素按下标 (index序号, 位置) 访问, 注意不能越界, 下标的变化范围为 $0 \sim \text{len}(\text{list}) - 1$ $-1 \sim -\text{len}(\text{list})$

正向递增序号 → 0 1 2 3 4

scores = [98, 96, 95, 94, 92]

 -5 -4 -3 -2 -1 ← 反向递减序号

scores [2] == ==score [-3]===== 95

遍历列表元素

- 直接遍历元素
表达更加直观

```
t = [1, 2, 3, 'abc', ['def']]
for item in t:
    print(item)
```

```
1
2
3
abc
['def']
```

- 使用range函数, 通过索引 (位置) 进行访问, 注意不要越界
IndexError: list index out of range

```
t = [1, 2, 3, 'abc', ['def']]
for i in range(len(t)):
    print(t[i])
```

- 若需要同时访问下标和元素,
可使用enumerate函数

```
frtlist = ['apple', ' banana', ' peach']
for i, item in enumerate(frtlist):
    print(i, item)
```

```
0 apple
1 banana
2 peach
```



【思考题2】

```
t = [1, 2, 3, 'abc', ['def']]  
for item in t:  
    if item == 3:  
        item = "T"  
print(t)
```

输出结果为？

直接遍历元素
表达更加直观

遍历列表元素

对于包含子列表的列表，可以使用两个下标
访问子列表中的内容

```
lst = [['cat', 3], ['dog', 2], ['duck', 1]]
```

```
for item in lst:  
    print(item)
```

```
['cat', 3]
```

```
lst[0][0]
```

```
['dog', 2]
```

```
lst[1][0]
```

```
['duck', 1]
```

```
lst[2][0]
```

【例】输入一组成绩，计算平均值，并将高于平均分的成绩存放到一个列表中并输出。

步骤：

1输入成绩

```
sclist = eval(input())
```

————→ 输入数据列表

```
n = len(sclist)
```

————→ 得到列表的长度

```
hlist = []
```

2计算平均值

```
s = 0
```

3遍历成绩列表

```
for sc in sclist:
```

————→ 得到累加和

```
    s += sc
```

s = **sum**(sclist)

```
aver = s / n
```

```
for sc in sclist:
```

————→ 遍历列表

```
    if sc > aver:
```

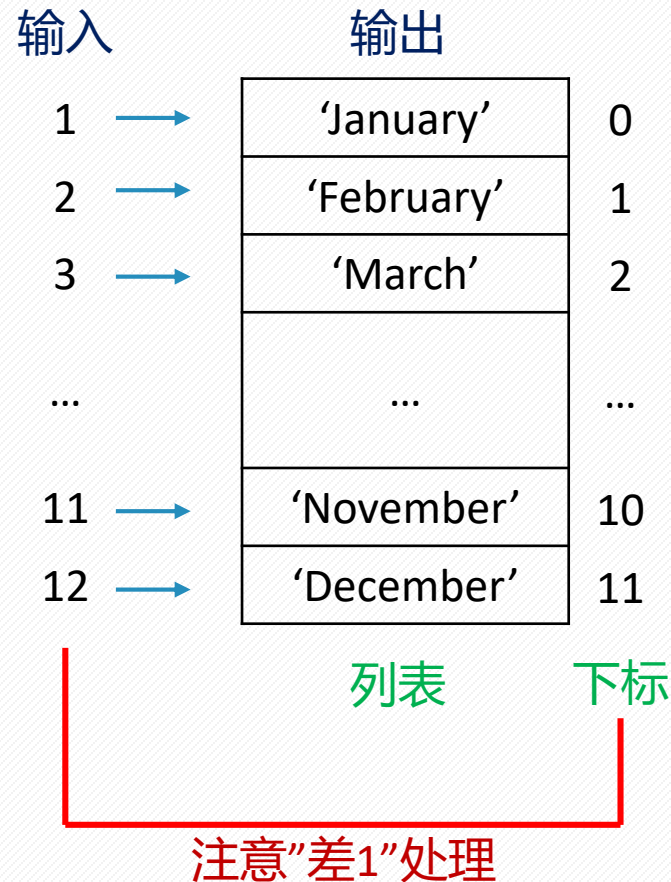
```
        hlist.append(sc)
```

hlist = [sc for sc in sclist if sc>aver]

```
print(hlist)
```

【例】根据输入的数字，输出对应的月份信息。如输入“6”，则输出“It's June.”

- 分析：可以创建列表来存储月份，按照输入的数字索引相应的列表中月份



```
months = ["January", "February", "March", "April", "May", "June",  
          "July", "August", "September", "October", "November", "December"]  
m = int(input("请输入整数月份"))  
print("It's {}".format(months[m-1]))
```

运行结果:

```
请输入整数月份12  
It's December.
```

- 【例】使用while循环，循环输入单词，当输入的单词不是quit时，则将其小写形式加入到列表中；当输入的单词为quit时，则退出循环（quit不加入到列表中）并将该列表输出。如果已经加入到列表中的单词，再次输入时，则输出“Exit”，不应再加入到列表。注意不区分大小写，例如若列表中已有apple，则再次输入的Apple或aPPLe等都不能再次加入到列表中。

步骤：

构造循环

单词加入到列表中

```
wdlist = []
word = input().lower()
while word != "quit":
    if word not in wdlist:
        wdlist.append(word)
    else:
        print("Exit")
        word = input().lower()
print(wdlist)
```

例：列出某个范围内（如1-999之间）的完全数

```
for n in range(1,1000):  
    s=0  
    for i in range(1,n):  
        if n%i==0:  
            s+=i  
    if n == s:  
        print(n,end=' ')
```

```
for n in range(1,1000):  
    s = n  
    for i in range(1,n):  
        if n%i == 0:  
            s -= i  
    if s == 0:  
        print(n)
```

```
for n in range(1,1000):  
    lst = []  
    for i in range(1,n):  
        if n%i==0:  
            lst.append(i)  
    if n == sum(lst):  
        print(n,end=' ')
```



```
lst = [ i for i in range(1,n) if n%i==0 ]
```


例：输入n，输出Fibonacci数列的前n项

```
n = int(input())
fibonacci = [1, 1]
if 1 <= n <= 2:
    print(fibonacci[:n])
elif n > 2:
    for i in range(2, n):
        f = fibonacci[i-1] + fibonacci[i-2]
        fibonacci.append(f)
    print(fibonacci)
```

修改列表元素

```
>>> t = [1, 2, 3, 'abc', ['def']]
>>> t[4] = 'XXX'
>>> t
[1, 2, 3, 'abc', 'XXX']
```

修改某一个元素

列表名[索引] = 新值

注意序号索引方向

```
>>> t
[2, 3, 4, 'abc', ['cat', 'dog', 'duck']]
>>> t[0], t[1] = t[1], t[0]
>>> t
[3, 2, 4, 'abc', ['cat', 'dog', 'duck']]
```

通过解包赋值，交换两个元素值

```
>>> t[0:3] = [2,3,4]
```

修改t[0]t[1]t[2]元素的值

```
>>> t
```

```
[2, 3, 4, 'abc', 'XXX']
```

```
>>> t[4] = ['cat','dog','duck']
```

```
>>> t
```

```
[2, 3, 4, 'abc', ['cat', 'dog', 'duck']]
```

```
>>>t= [1,2,3,4]
```

```
>>> t[1:3] = ['a','b','c']
```

```
>>> t
```

```
[1, 'a', 'b', 'c', 4]
```

修改t[1]t[2]元素的值，同时增加t[3]元素

```
>>> t=[1, 'a', 'b', 'c', 4]
```

```
>>> t[3:3] = [666]
```

```
>>>t
```

```
[1, 'a', 'b', 666, 'c', 4]
```

插入t[3]元素

修改一部分元素的值：使用切片
切片赋值：赋值号右边必须是序列类型，若给出的数据超出了切片范围，则其余元素插入到列表中。

删除列表元素:del pop

1. 按索引删除元素——del命令

删除一个或某个范围内的元素

删除对数据对象的引用: del t

del是删除引用(变量)而不是删除对象(数据),
对象由自动垃圾回收机制 (GC) 删除

```
>>> t = [1, 2, 3, 4, "cat"]
>>> del t[4]
>>> t
[1, 2, 3, 4]
>>> del t[1:3]
>>> t
[1, 4]
```

```
>>> t
[2, 3, 4]
>>> del t
>>> t
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
NameError: name 't' is not defined
```

2. 按索引删除元素——pop方法

会将删除元素返回; pop()默认删除的是最后一个元素

```
>>> t = [1, 2, 3, 4, "cat"]
>>> t.pop()
'cat'
>>> t
[1, 2, 3, 4]
```

```
>>> t
[1, 2, 3, 4]
>>> t.pop(0)
1
>>> t
[2, 3, 4]
```

删除列表元素:remove

3. 按值删除元素——remove方法

删除单个元素，若有多个符合条件的元素，则删除**首个符合条件的元素**（只删除一个），**若找不到，会出现错误**

```
>>> t = [1, 2, 3, 2, 4, 5, 2]
>>> t.remove(2)
>>> t
[1, 3, 2, 4, 5, 2]
```

删除列表元素:remove

例：删除列表中的多个相同元素，如删除列表中所有的2

```
t = [1, 2, 3, 2, 4, 5, 2]
```

```
n = 2
```

```
while n in t:  
    t.remove(n)
```

```
print(t)
```

使用while循环

```
t = [1, 2, 3, 2, 4, 5, 2]
```

```
n = 2
```

```
newt = []  
for i in t:  
    if i != n:  
        newt.append(i)
```

```
print(newt)
```

利用列表生成式

```
newt = [i for i in t if i != n]
```

```
t = [1,2,3,2,4,5,2]
```

```
n = 2
```

```
for i in t:
```

```
    if i==n:
```

```
        t.remove(n)
```

```
print(t)
```



删除列表所有元素:del , clear()

清空列表中的所有元素，列表本身还存在：

```
>>> t = [1, 2, 3, 2, 4, 5, 2]
>>> t.clear()
>>> t
[]
```

del t

del后直接跟列表名，则将彻底删除该列表对象！

```
>>> t = [1, 2, 3, 2, 4, 5, 2]
>>> del t[:]
>>> t
```

[] 经过删除“所有元素”的del操作后，列表中不包含任何元素，但是仍保留其列表的本质。

```
>>> t
[2, 3, 4]
>>> del t
>>> t
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
NameError: name 't' is not defined
```

删除列表(元素)小结

- 已知待删除元素的索引时，可使用del命令和pop方法；
- pop方法对于删除列表最末尾元素最为简单方便；
- 明确知道待删除元素值时，用remove方法更为简单。
- 与del命令和remove方法不同，pop方法在删除元素的同时会“弹出”这个被删除的元素，如果需要可以用一个变量“接住”它，以便进行进一步的后期操作。
- 清除所有元素：clear方法和del 列表名[:]
- 彻底删除列表：del 列表名

查找列表元素:index (用途比较多)

index方法

列表.index(元素)

用来在列表中查找指定的元素，如果找到返回第一个元素对应的索引；

如果找不到，会直接报错。【注意列表中无find方法】

```
>>> t = [1, 2, 3, 'cat', 'dog']
>>> t.index('dog')
4
```

```
>>> t = [1, 2, 3, 'cat', 'dog']
>>> t.index(5)
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
ValueError: 5 is not in list
```

count方法

列表.count(元素)

用来统计并返回列表中指定元素的个数

[例] 公交 649 路木华园到丁字沽方向车站如下:
stations = ["木华园站", "河工大体育中心站", "金声园站", "河工大教师公寓站", "学院楼站", "兴工广场站", "安津医院站", "安津医院站", "北辰工业园站", "刘园地铁站", "瑞景新苑地铁站", "佳宁里站", "佳园里地铁站", "佳园南里站", "绥中南楼站", "本溪路站", "丁字沽十三段站", "丁字沽八段站", "丁字沽四段站", "丁字沽站", "丁字沽公交站"]

请输入当前上车站点和要下车的站点名称，计算经过的站点个数并输出，如果不经该站则输出“请到对面车站乘车”

例如：

输入： "兴工广场站", "刘园地铁站"

输出： 4

```
stations = ["木华园站", "河工大体育中心站", "金声园站", "河工大教师公寓站", "学院楼站", "兴工广场站", "安津医院站", "安津医院站", "北辰工业园站", "刘园地铁站", "瑞景新苑地铁站", "佳宁里站", "佳园里地铁站", "佳园南里站", "绥中南楼站", "本溪路站", "丁字沽十三段站", "丁字沽八段站", "丁字沽四段站", "丁字沽站", "丁字沽公交站"]
```

```
start, stop = eval(input())
```

```
startindex = stations.index(start)
```

```
stopindex = stations.index(stop)
```

```
if stopindex >= startindex:
```

```
    print(stopindex - startindex)
```

```
else:
```

```
    print("请到对面车站乘车")
```

列表排序

sort方法 sorted命令

sort方法排序: 原地排序, 修改了原来的列表

```
>>> ls=[6,5,4,3,2,1]
>>> ls.sort(reverse=True)
>>> ls
[6, 5, 4, 3, 2, 1]
```

reverse参数为True时对列表元素按降序排序

```
>>> ls=[(6,3),(2,5),(4,1),(7,6)]
>>> ls.sort()
>>> ls
[(2, 5), (4, 1), (6, 3), (7, 6)]
```

嵌套结构的列表默认元素的第一个子元素
为关键字按升序排序

```
>>> ls=[6,5,4,3,2,1]
>>> ls.sort()
>>> ls
[1, 2, 3, 4, 5, 6]
```

缺省参数的sort方法默认对列表元素按升序排序

列表排序

sort方法 sorted命令

```
>>> ls=[6,5,4,3,2,1]
>>> sorted(ls)
[1, 2, 3, 4, 5, 6]
>>> ls
[6, 5, 4, 3, 2, 1]
```

```
>>> ls=[(6,3),(2,5),(4,1),(7,6)]
>>> sorted(ls,reverse=True)
[(7, 6), (6, 3), (4, 1), (2, 5)]
>>> ls
[(6, 3), (2, 5), (4, 1), (7, 6)]
```

sorted命令的使用和参数含义和sort方法一致。

但sorted命令生成新的有序列表，不改变原来的列表。

列表排序

sort方法 sorted命令

```
countries=["New Zealand","Australia","China","United  
Kingdom","Togo","Turkey","Sao Tome and Principle","Samoa"]
```

```
countries.sort(key=len, reverse = True)
```

按字符串长度， 逆序排序

```
['Sao Tome and Principle', 'United Kingdom', 'New Zealand', 'Australia', 'Turkey',  
'China', 'Samoa', 'Togo']
```

例：计算中位数

先对系列数据进行排序，然后，中位数的计算方式分为如下两种：

1. 若列表中元素的个数为奇数，则中位数为**排序后**列表中中间位置的那个数。
2. 若列表中元素的个数为偶数，则中位数为**排序后**列表中中间位置两个数的平均值。

```
nlist = eval(input())
nlist= sorted(nlist)  #或nlist.sort()
l = len(nlist)
if l % 2 == 0:
    print( (nlist[l//2 - 1] + nlist[l//2]) / 2 )
else:
    print(nlist[l//2])
```

例：学校举办朗诵比赛，邀请了10位评委为每一名参赛选手的表现打分。假设列表`lst_score = [9, 10, 8, 9, 10, 7, 6, 8, 7, 8]`，存放了某一位参赛选手的所有评委评分。试编写程序，根据以下规则计算该参赛选手的最终得分：

- (1) 去掉一个最高分。
- (2) 去掉一个最低分。
- (3) 最终得分为剩下8个分数的平均值。

```
score = [9, 10, 8, 9, 10, 7, 6, 8, 7, 8]
score.sort()
score.pop()
score.pop(0)
print(sum(score)/len(score))
```

列表的复制

copy

- 列表的复制：

1. 使用列表的切片方法 `s = t[:]`
2. 使用列表的copy方法 `s = t. copy()`
3. 通过列表之间的赋值操作 `s=t`

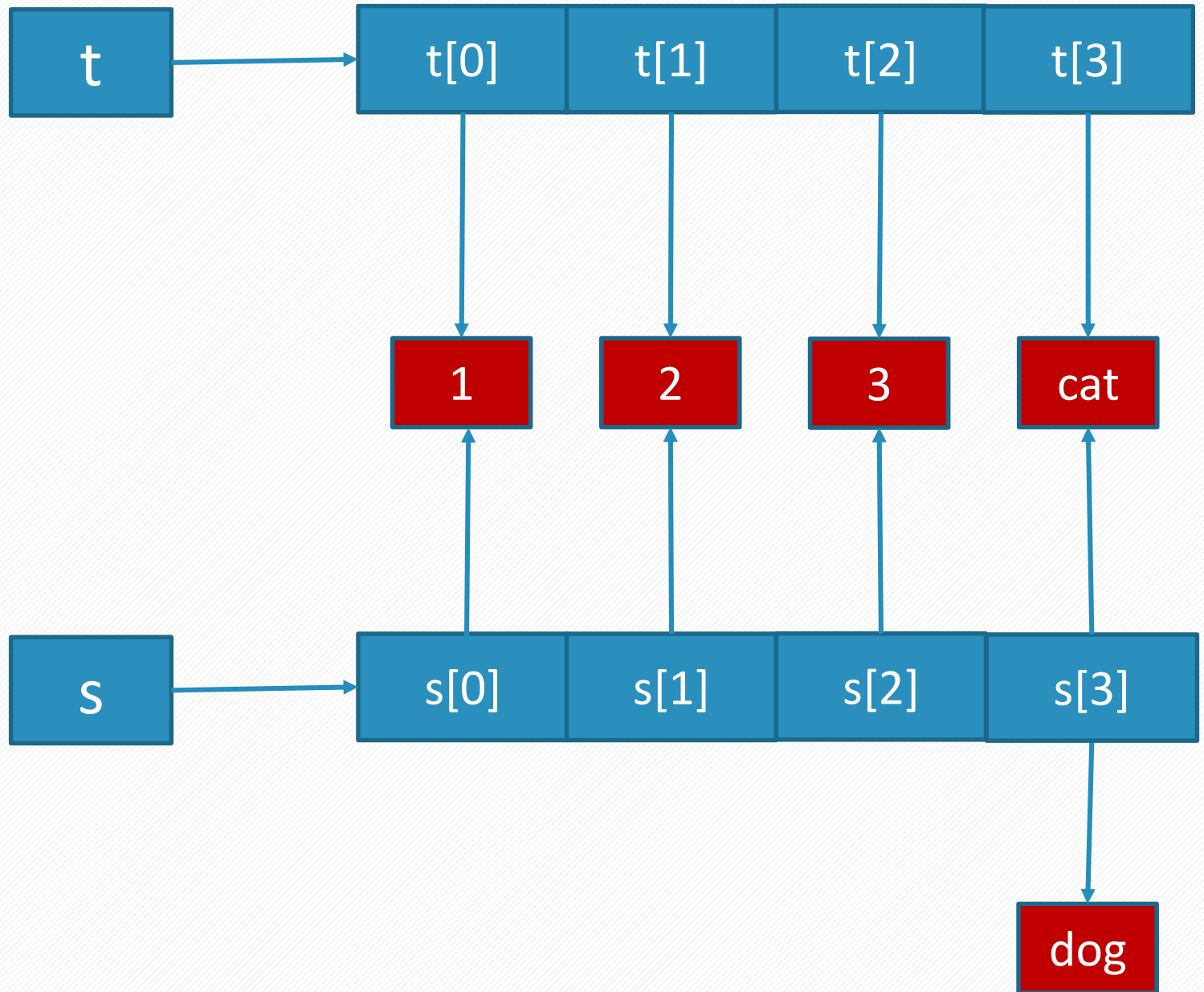
copy方法和赋值操作都能得到 “一样” 的列表，但是两者的实现机制有着本质的区别。

切片和copy方法的新表和原表各自为**独立**的，而**赋值操作**的新表是原列表的**别名**。

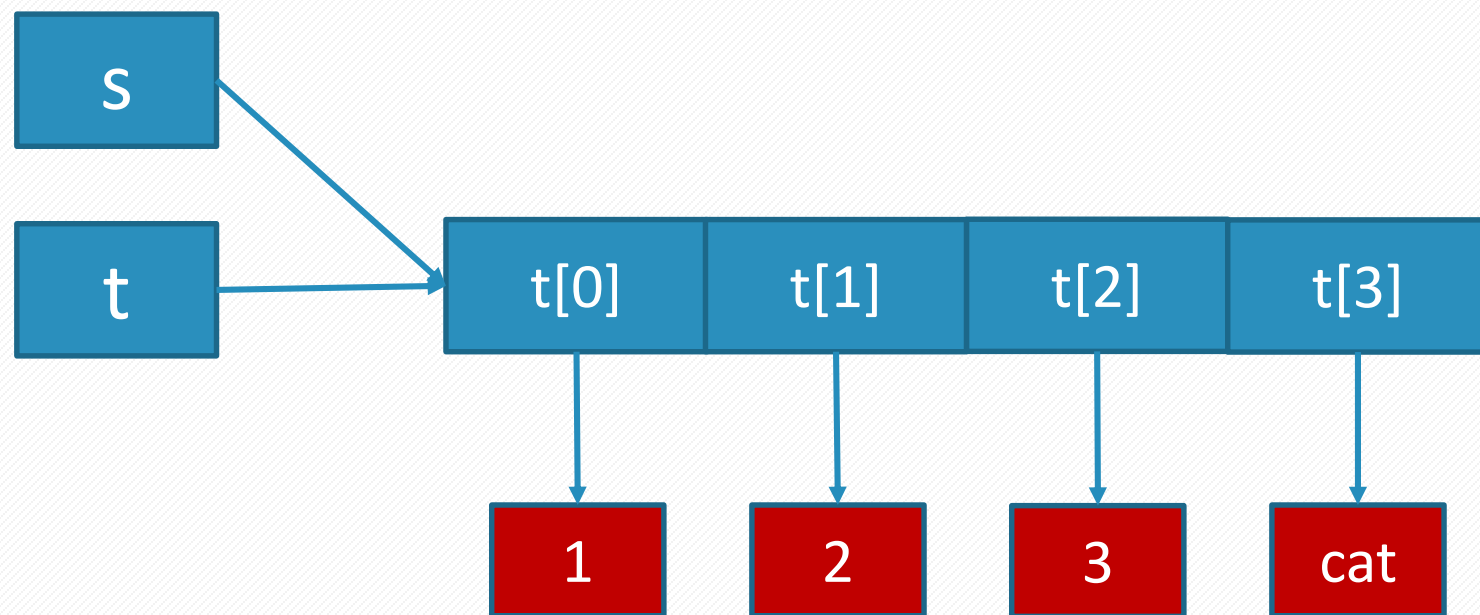

```
>>> t = [1, 2, 3, 'cat']
>>> s = t.copy()
>>> s
[1, 2, 3, 'cat']
>>> s[3] = 'dog'
>>> s
[1, 2, 3, 'dog']
>>> t
[1, 2, 3, 'cat']
```

```
>>> t = [1,2,3,"cat"]
>>> s = t[:]
>>> s
[1, 2, 3, 'cat']
```

两个列表有独立空间.



```
>>> t = [1, 2, 3, 'cat']
>>> s = t
>>> s
[1, 2, 3, 'cat']
>>> s[3] = 'dog'
>>> s
[1, 2, 3, 'dog']
>>> t
[1, 2, 3, 'dog']
```



赋值,两个列表共享空间.

id函数: id(变量) 得到变量的地址

is函数: 判读两个变量是否引用了

相同的对象: `s is t`

```
>>> t = [1, 2, 3, 'cat']
>>> s = t
>>> s
[1, 2, 3, 'cat']
>>> s[3] = 'dog'
>>> s
[1, 2, 3, 'dog']
>>> t
[1, 2, 3, 'dog']

>>> id(s)
64397680
>>> id(t)
64397680

>>> s == t
True
>>> s is t
True
```

例：设有列表lst_odd = [1, 3, 5, 7, 9]和列表lst_even = [2, 4, 6, 8, 10]。试编写程序，将两个列表合并成一个新的列表，并将新列表按照元素的大小降序排列。[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

```
lst_odd = [1, 3, 5, 7, 9]
lst_even = [2, 4, 6, 8, 10]
newlist = lst_odd + lst_even
newlist.sort(reverse = True)
print(newlist)
```

例：生成一个包含20个两位随机整数列表，将其前10个元素进行升序排序；将后10个元素进行降序排序。

```
import random
xlist = random.sample(range(10,100), 20)
xlist[:10] = sorted(xlist[:10])
xlist[10:] = sorted(xlist[10:],reverse = True)
print(xlist)
```

列表生成式List Comprehension

列表 = [循环变量相关表达式exp for 循环变量 in 可迭代对象]

工作过程：

- 遍历可迭代对象中的每个元素；
- 每次循环得到一个循环变量，然后通过exp得到一个新的值；
- 最后把所有通过exp得到的值以一个新列表的形式返回

```
mylist = []
```

```
for i in iterable:
```

```
    mylist.append(i)
```

等价于： mylist = [i for i in iterable]

mylist = [i for i in range(5)]

创建一个由0 ~ 4这5个数组成的数值列表

```
>>> mylist = [i for i in range(5)]
>>> mylist
[0, 1, 2, 3, 4]
```

mylist = [i*i for i in range(5)]

创建一个由0 ~ 4这5个数的平方值组成的数值列表

```
>>> mylist = [i*i for i in range(5)]
>>> mylist
[0, 1, 4, 9, 16]
```

t = [1,2,2,3,4]

mylist = [i for i in t if i != 2]

创建一个由t中不等于2的数据组成的数值列表

```
>>> mylist = [i for i in [1,2,2,3,4] if i != 2]
>>> mylist
[1, 3, 4]
```


过滤出一个指定的数字列表^t中值大于20的元素,
并创建一个列表

```
[x for x in mylist if x >20]
```

例：输入一行单词，以空格相隔，输出长度最长的单词。

```
sentence = input()
wordlist = sentence.split()
wordlen = [(len(item)) for item in wordlist] #产生一个单词长度的列表
maxlen = max(wordlen) #得到最长长度
l = [item for item in wordlist if len(item) == maxlen] #生成单词长度等于最长长度的单词列表
print(wordlen,l)
```

运行结果：

I think it is very easy

[1, 5 2, 2, 4, 4] ['think']

【例】已知有学生信息列表如下：

```
ls=[[ '张三', 18, '江苏南京'], ['李四', 19, '山东济南']]
```

请用列表生成式生成包含学生姓名的列表。

```
>>> ls=[[ '张三', 18, '江苏南京'], ['李四', 19, '山东济南']]
>>> names=[item[0] for item in ls]
>>> names
['张三', '李四']
...
```

如果需要生成包含学生姓名和籍贯的列表，列表生成式该如何写？

```
[item[0], item[2]] for item in ls]
```

列表生成式是允许嵌套的，其嵌套格式如下：

列表 = [循环变量相关表达式 for 循环变量A in 可迭代对象A for
循环遍历B in 可迭代对象B]

```
>>> ls2=[i*j for i in range(1,3) for j in range(1,5)]  
>>> ls2  
[1, 2, 3, 4, 2, 4, 6, 8]
```



```
>>> for i in range(1,3):  
    for j in range(1,5):  
        ls.append(i*j)  
  
>>> ls  
[1, 2, 3, 4, 2, 4, 6, 8]
```

例：计算两个集合的全排列，并将结果作为保存至一个新的列表中

```
fruits = ['香蕉', '苹果', '橙子' ]
```

```
drinks = ['可乐', '牛奶']
```

```
blst=[f,d] for f in fruits for d in drinks]
```

```
blst = []
```

```
for f in fruits:
```

```
    for d in drinks:
```

```
        blst.append([f, d])
```

【例】 请用列表生成式生成九九乘法表列表并输出。

```
1*1=1
2*1=2  2*2=4
3*1=3  3*2=6  3*3=9
4*1=4  4*2=8  4*3=12  4*4=16
5*1=5  5*2=10  5*3=15  5*4=20  5*5=25
6*1=6  6*2=12  6*3=18  6*4=24  6*5=30  6*6=36
7*1=7  7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49
8*1=8  8*2=16  8*3=24  8*4=32  8*5=40  8*6=48  8*7=56  8*8=64
9*1=9  9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81
```

【分析】

- I. 九九表中每一项可以写作 $i * j = r$ 。
- II. i 对应行号，看作外循环； j 对应列号，看作内循环。
- III. 乘法表共有9行，即 i 的取值范围为1~9；第 i 行共有 i 项乘法式子，即 j 的取值范围为1~ i 。

```
ls = [str(i) + '*' + str(j) + '=' + str(i*j) + ('\n' if i==j else '\t') \
      for i in range(1,10) for j in range(1,i+1)]
```

```
for item in ls:
```

```
    print(item,end="")
```

每行最后一项后加上\n，
其余各项后加上\t

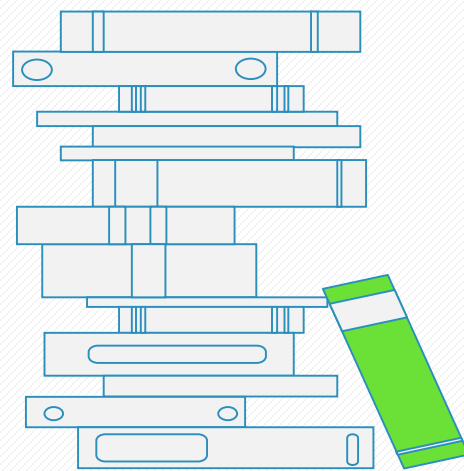


数值列表的简单统计计算

1. min函数——求数值列表中的最小值元素。
2. max函数——求数值列表中的最大值元素。
3. sum函数——求数值列表中元素之和。



元组与元组操作 tuple



元组与元组定义

元组 (tuple) 与列表类似，也是用来存放一组相关的数据。两者的不同之处主要有两点：

a) 元组使用圆括号 () ， 列表使用方括号 [] ；

b) 元组的元素不能修改。

可以将元组理解为不能修改的 “列表” 一低配版。

元组与元组定义

创建一个空元组:

```
tp = tuple()
```

```
tp = ()
```

```
tp = 1, 2, 3, 4 #默认为tuple
```

```
tp = (1, 2, 3, 4)
```

不使用(), 多个数据用 “, ” 隔开也可以定义元组。

元组与元组定义

```
>>> t= ("cat")
```

```
>>> type(t)
```

```
<class 'str'>
```

```
>>> t= ("cat",)
```

```
>>> type(t)
```

```
<class 'tuple'>
```

元组只有1个元素时，系统将其视作单个的字符串。

若只有1个元素，则需要在这个数据后面加逗号

元组与元组定义

```
>>> a = list(eval(input()))
```

```
1,2,3,4,5
```

```
>>> a
```

```
[1, 2, 3, 4, 5]
```

```
>>> t = eval(input())
```

```
1, 2, 3, 4, 5
```

```
>>> t
```

```
(1, 2, 3, 4, 5)
```

```
>>> type(t)
```

```
<class 'tuple'>
```

元组与元组定义

元组是不可变的：

既不能增加元素；
也不能修改元素。

均不支持这些方法：

append

extend

insert

pop

remove

sort

del

可以完成的操作：

切片：

```
>>> t = 1, 2, 3, 4
```

```
>>> t[1:]
```

+

```
(2, 3, 4)
```

*

```
>>> t = (1,2,3)
```

```
>>> s = (4,5,6)
```

```
>>> s + t
```

```
(4, 5, 6, 1, 2, 3)
```

```
>>> t * 2
```

```
(1, 2, 3, 1, 2, 3)
```

sorted

max

```
>>> sorted(t,reverse = True)
```

min

```
[3, 2, 1]
```

sum

```
>>> max(t)
```

```
3
```

```
>>> min(t)
```

```
1
```

```
>>> sum(t)
```

```
6
```

元组的操作

元组是“不能修改”的列表，因此列表中不涉及元素修改的操作都适用于元组。

操作	列表	元组
读元素	√	√
写元素	√	×
append方法	√	×
insert方法	√	×
pop方法	√	×
del命令	√	只支持删除整个元组
remove方法	√	×
len函数	√	√
in运算	√	√
not in运算	√	√
index方法	√	√
count方法	√	√

元组的操作

操作	列表	元组
遍历元素	√	√
sort方法	√	×
sorted函数	√	排序结果为列表
切片	√	√
+运算	√	√
*运算	√	√
extend方法	√	×
copy方法	√	×
赋值	√	√
max函数	适用于数值列表	适用于数值元组
min函数	适用于数值列表	适用于数值元组
sum函数	适用于数值列表	适用于数值元组

元组是不可变类型，但是可以利用元组生成新的元组

例： 利用列表生成式重新生成列表

```
x = [ ('a',1), ('b', 2), ('c', 3) ]
```

```
y =[ (item[0], item[1]+1) for item in x]
```

```
print(y)
```

元组赋值：

```
a, b = (1, 2)           (a,b) = (b, a)
```

函数：

```
return 1,2====return (1,2)
```

例： 字符串s中存放了某个学生各科的期末考试成绩，编写程序，计算该学生所有科目的总分和平均分。

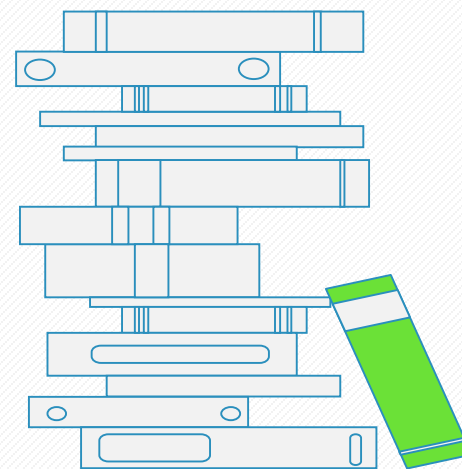
s="语文:80,数学:82,英语:90, 物理: 85,化学:88,美术:80"

```
s="语文:80,数学:82,英语:90, 物理: 85,化学:88,美术:80"  
slist = s.split(',')  
newlist = [item.split(":") for item in slist]  
scorelist = [int(item[1]) for item in newlist]  
print(sum(scorelist))
```

```
['语文:80', '数学:82', '英语:90', ' 物理: 85', '化学:88', '美术:80']  
[['语文', '80'], ['数学', '82'], ['英语', '90'], [' 物理', ' 85'], ['化学', '88'], ['美术', '80']]  
[80, 82, 90, 85, 88, 80]
```



序列转换函数





序列

- 本章介绍的列表、元组和前面学习的字符串都属于Python的一种基本数据类型——序列（sequence）。
- 序列的最大特点是元素的有序性，所以序列都是通过序号索引来访问元素的。
- 序列之间可以通过转换函数进行互相转换。

元组与列表之间的转换



```
>>> tupPlay1 = ("萧峰", "男", 98)
>>> tupPlay1
('萧峰', '男', 98)
>>> listPlay1 = list(tupPlay1)
>>> listPlay1
['萧峰', '男', 98]
```

```
>>> listScores = [98, 96, 95, 94, 92]
>>> listScores
[98, 96, 95, 94, 92]
>>> tupScores = tuple(listScores)
>>> tupScores
(98, 96, 95, 94, 92)
```

字符串转换成列表

1. 使用list函数

```
>>> names = "张三丰, 萧峰"
>>> guests = list(names)
>>> guests
['张', '三', '丰', ',', ' ', '萧', '峰']
```

list函数转换后字符串中的**单个字符**依次成为列表元素

2. 使用字符串的split方法

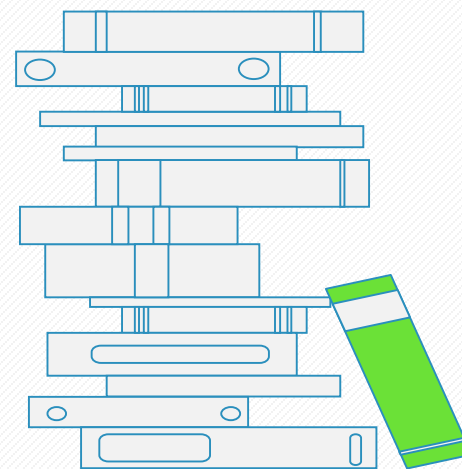
```
>>> names = "张三丰, 萧峰"
>>> guests = names.split(',')
>>> guests
['张三丰, 萧峰']
```

列表 = 字符串.split(分隔符)

分隔符如果缺省的话,
默认按照空格拆分字符串



综合应用（自学）



例1： 模拟掷两个骰子100 000次， 统计各点数出现的概率。

0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12

分析： 点数之和作为下标， 列表元素的值为出现的次数

```
import random
f = [0]*13
for i in range(100000):
    f1 = random.randint(1,6)
    f2 = random.randint(1,6)
    f[f1+f2] += 1
for i in range(2, 13):
    print(i,f[i]/100000)
```


例2： 请用筛选法求出300以内的素数。

1	1	1	1	1	1	1	1	1	1	1
0	1	2	3	4	5	6	7	8	298	299

所有数标记为素数

0	0	1	1	1	1	1	1	1	1	1
0	1	2	3	4	5	6	7	8	298	299

将0和1剔除

0	0	1	1	0	1	0	1	0	0	1
0	1	2	3	4	5	6	7	8	298	299

将2的倍数剔除，
将其值设置为0，
表示为非素数

0	0	1	1	0	1	0	1	0	0	0	1
0	1	2	3	4	5	6	7	8	9	298	299

将3的倍数剔除

.....

将下一个**元素为1的下标对应**的倍数剔除

具体算法：

用i从2开始遍历整个列表;

如果第i个元素值为1,将后面所有能被i整除的下标对应的元素改写为0;

遍历结束后,保持为1的元素对应的**下标**即为要求的素数。

	2	3	4	5	6	7	8	9	10	Prime numbers
11	12	13	14	15	16	17	18	19	20	
21	22	23	24	25	26	27	28	29	30	
31	32	33	34	35	36	37	38	39	40	
41	42	43	44	45	46	47	48	49	50	
51	52	53	54	55	56	57	58	59	60	
61	62	63	64	65	66	67	68	69	70	
71	72	73	74	75	76	77	78	79	80	
81	82	83	84	85	86	87	88	89	90	
91	92	93	94	95	96	97	98	99	100	
101	102	103	104	105	106	107	108	109	110	
111	112	113	114	115	116	117	118	119	120	

#筛选法求素数。

```
1  primes = [1] * 300 # id: 0, 1, 2, 3 ~299
2  primes[0:2] = [0,0]
3  for i in range(2 , 300):
4      if primes[i]==1:
5          for j in range(i+1,300):
6              if primes[j] != 0 and j % i == 0:
7                  primes[j] = 0
8
9  print("300以内的素数包括: ")
10 for i in range(2 , 300):
11     if primes[i]:
12         print(i , end = ' ')
```

例3：折半查找(Binary search/二分查找)，折半查找条件是数据是有序的。
请用折半查找法在列表中查找指定的元素，找到输出其索引值，否则输出“没找到”。
(假设待查找元素 $x=4$)

确定low、high、mid的初值： $low = 0$ $high = \text{len}(ls)$ $mid = (low + high) // 2$

1	2	3	4	5	6	7	8	9	10
ls[0]	ls[1]	ls[2]	ls[3]	ls[4]	ls[5]	ls[6]	ls[7]	ls[8]	ls[9]
↑ low					↑ mid				↑ high

按如下规则重复：

如果 $ls[mid] < x$ ，确定 $(mid, high]$ 为下一查找区间，重新赋值 $low = mid + 1$ ，计算新mid，继续；

如果 $ls[mid] > x$ ，确定 $[low, mid)$ 为下一查找区间，重新赋值 $high = mid - 1$ ，计算新mid，继续；

直到 $ls[mid] == x$ ，找到位置为mid, break

1	2	3	4	5	6	7	8	9	10
ls[0]	ls[1]	ls[2]	ls[3]	ls[4]	ls[5]	ls[6]	ls[7]	ls[8]	ls[9]
			↑ low	↑ mid	↑ high				

#二分查找

```
1  ls = [34, 64, 67, 72, 73, 82, 83, 85, 87, 88, 90, 91, 96, 98]
2  x = int(input("请输入待查找的数:"))
3
4  low = 0
5  high = len(ls) - 1
6
7  while low <= high:
8      mid = (low + high) // 2
9      if ls[mid] < x:
10         low = mid + 1
11     elif ls[mid] > x:
12         high = mid - 1
13     else:
14         print("找到{},索引为{}".format(x,mid));
15         break
16 else:
17     print("没有找到{}".format(x))
```

例4： 对下面列表

names = "双儿 洪七公 赵敏 赵敏 双儿 双儿 洪七公 赵敏 郭靖 杨过 洪七公 赵敏 郭靖 洪七公 双儿 洪七公 赵敏 郭靖 杨过 洪七公 郭靖 郭靖 杨过 洪七公 郭靖 洪七公 郭靖 双儿 洪七公 赵敏"

统计出现最多的人名。

```
nlst = names.split()
```

```
namelst = [] #存放出现的不重复的人名
```

```
for i in nlst:
```

```
    if i not in namelst:
```

```
        namelst.append(i)
```



```
namelst=list(set(nlst))
```

```
cntlst = [nlst.count(i) for i in namelst]
```

```
m = max(cntlst)
```

```
print(namelst[cntlst.index(m)], m)
```

本周作业（第6次课）：

1. 课件内容和上课所讲内容的复盘——章节包括函数和列表
2. 实验书：例4-2，例4-4，例4-5；例5-5；例6-1，例6-2，例6-3

上机课实践：P96 4(3)、(4)、(6)；P108 4.(1)-(2)

3. 上机课实践：

lab.hebut.edu.cn 的python 练习系统的列表模块，函数模块补做含列表的内容

4. Python123.io中的作业

注：标有“上机课实践”尽量在上机课时间完成



The End