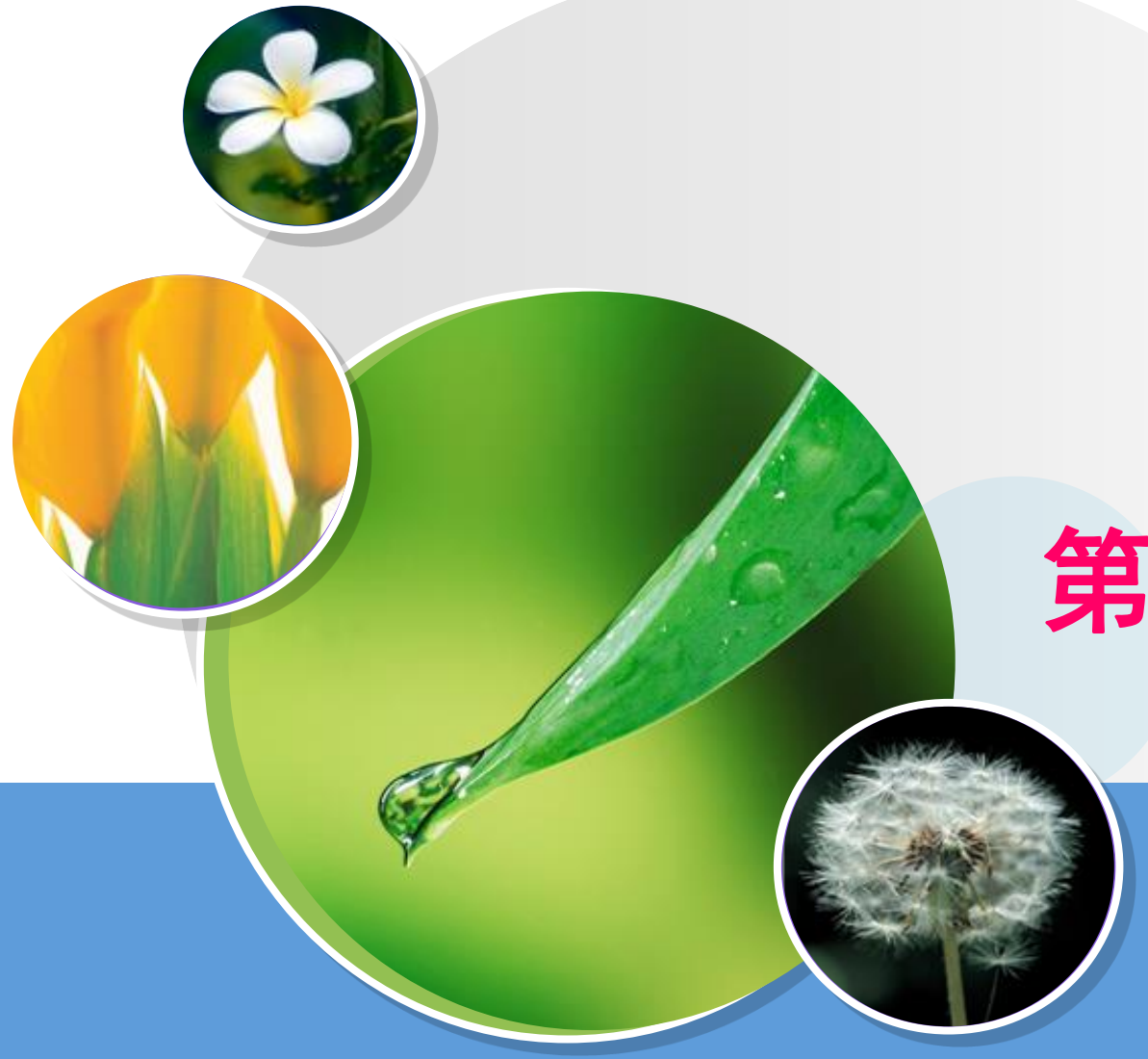

北京航空航天大學
BEIHANG UNIVERSITY

大学计算机基础

（理科类）

第6讲 Python的基本语法（二）

北京航空航天大学

第5讲回顾

1、比较y=x[:]、 copy模块的deepcopy函数、copy模块的copy函数的异同

- **deepcopy函数**：**深拷贝（deep copy，深复制）**，将被复制对象完全再复制一遍，作为独立的新个体单独存在。修改原有被复制对象不会对副本产生影响；对副本的修改也不会影响原对象
- **copy函数**：**浅拷贝（shallow copy）**，仅复制对象本身，而不对其中的子对象进行复制。如果对**原子对象**进行修改，则副本中的**子对象**也会随之修改；反之亦然。**y= x[:]**也是**浅拷贝**
  - ✓ 对于**简单**的对象，copy 与 deepcopy **没有区别**，修改原对象不会对副本产生影响
  - ✓ 对于**复杂**对象（序列里的嵌套序列，字典里的嵌套序列），二者则**不同**

- **深层复制 deepcopy()：复制了对象和对象的所有子对象**
- **浅层复制 copy()：复制父对象，子对象仍然使用引用的方式**

深拷贝与浅拷贝的比较

```
>>> import copy
>>> a=[1,2,3]

>>> b=copy.copy(a)
>>> c=copy.deepcopy(a)
>>> b==c
True
>>> a[2]=9
>>> b
[1, 2, 3]
>>> c
[1, 2, 3]
>>> a
[1, 2, 9]
```

对于**简单**的对象，copy 与 deepcopy **没有区别**，修改原对象不会对副本产生影响

```
>>> a=[1,2,['asd',6]]
>>> b=copy.copy(a)
>>> c=copy.deepcopy(a)
>>> d=a[:]
>>> a[2][0]='1'
>>> a
[1, 2, ['1', 6]]
>>> b
[1, 2, ['1', 6]]
>>> c
[1, 2, ['asd', 6]]
>>> d
[1, 2, ['1', 6]]
>>>
```

对于**复杂**的对象，修改**原子对象**，copy 的**副本**随之**修改**

对于复杂的对象，修改**原子对象**，deepcopy 的**副本不受影响**

copy-deepcopy比较.py

第5讲回顾（续1）

2、 extend(L)方法的应用

- list1 =[1, 2, 3, 'add1', 'add2']，现欲生成列表 [1, 2, 3, 'add1', 'add2', 'app1', 'app2']
- list1.extend('app1', 'app2')对吗？为什么？
- 应该怎样写？

【讨论】如何访问二维列表？

- 通过键盘输入一个矩阵，每次输入矩阵的一行，各元素之间用一个空格分隔
- 访问该矩阵，提取主对角线的各元素，存储在一个列表中，并输出该列表

程序框架

```
#访问二维列表.py

n=int(input())                          #整数，代表矩阵的阶数
lis=[]                                  #n阶矩阵
for i in range(n):
    line=……                             #输入一行
    ……                                  #将每一行添加到列表中

x=[]                                    #存储主对角线的各元素
for i in range(n):                      #遍历每一行
    for j in range(n):                  #遍历每一列
        ……
            ……

print("主对角线上元素=",x)
```

目　录

北京航空航天大學
BEIHANG UNIVERSITY

5.4 字符串

北京航空航天大学

字符串

- **字符串：**用双引号" "或单引号' '包裹起来的一系列字符
  - 例： " world " ，或' world’
  - '123'代表字符串，而并非数字123

- **字符串是字符序列，可通过位置索引访问每个字符**

  <字符串>[<索引>]

  - 可以**使用索引从字符串中提取单个字符**
    - ✓ s='xyz‘
    - ✓ s[0]获取首字符‘x'
  - 或**使用切片提取任意长度的子串**
    - ✓ s[1:3] 获取子串 'yz'
  - 可以**间隔一定的步长，提取子串**

```
>>> s='123456789'
>>> s[1:8:2]
'2468'
```

字符串操作

- 序列的**通用操作**适用于字符串：**索引、分片、连接、乘法、成员资格检查**操作
- **字符串方法**：find、join、lower、upper、replace、split、strip、translate……

字符串与列表的区别

- Python字符串是**序列**，同样可通过索引引用
- 列表与字符串的**区别**
  - 字符串可以理解成是一个特殊的列表；
  - 列表的成员（元素）可以是任何数据类型，而字符串中只能是字符；
  - **列表**的成员（元素）**可修改**，而**字符串不能修改——对字符串的操作不会改变原字符串，而是生成一个新的字符串**。

字符串的遍历

字符串的遍历

要掌握！

- **方法**：使用**for**语句遍历。简洁！建议采用

```
#string_traversal_1.py
word='hardwork'
print('word中的所有字母是：')
for each_item in word:
    print(each_item)
```

循环变量为字符串中每个字符

```
>>>
word中的所有字母是：
h
a
r
d
w
o
r
k
```

字符串的应用示例：【例5.10】

【例5.10】计算任意一个单词所对应的总分数。

- 定义如下的规则
  - 英文字母a/A对应1分、b/B对应2分、c/C对应3分……
  - 每个英文单词的得分即为各个字母分数的和
  - 试计算任意一个单词所对应的总分数
  - 将结果显示在屏幕上

补充知识： ASCII码

- **西文字符的编码**
  - 在计算机内部，西文字符的编码采用**ASCII码**（American Standard Code for Information Interchange，美国信息交换标准交换代码）
  - 国际通用的是**7位**ASCII码（**基础ASCII码**），用7位二进制数表示一个字符的编码
  - 基础ASCII码可以表示**128**个不同字符
    - ✓ 普通字符**94**个： 26个大写英文字母、26个小写英文字母、10个阿拉伯数字、通用运算符号（+、-、×、÷等）及标点符号等共32个
    - ✓ 控制字符或通信专用字符**34**个
    - ✓ 基础ASCII码每个字符都对应一个数值，称为该字符的**ASCII码值**。其排列次序为$b_6b_5b_4b_3b_2b_1b_0$，编码为000 0000~111 1111
- ASCII码的编排有一定规律，字母A～Z、a～z都是**按顺序**编排的。而且小写字母比大写字母的码值**大32**

计算机内部用一个字节存放一个7位ACII码，最高位置0

基础ASCII码编码表

$b_6b_5b_4$ $b_3b_2b_1b_0$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	-	o	DEL

“A”的ASCII码值是65

“B”的ASCII码值是66

“a”的ASCII码值是97

“b”的ASCII码值是98

【例5.10】设计思路

- 为便于处理，先使用字符串的**lower方法**将输入的单词word转换为小写字符串

  new_word=word.lower()

- **如何迅速获得某个字母对应的分数?**
  - 内置函数**ord()**：获取某个字符的ASCII码
  - **基础ASCII码**中字母A～Z、a～z都是**按顺序**编排的，“a”的ASCII码值是97，“b”的ASCII码值是98
  - 字母e的分数= **ord('e') - ord('a') + 1** = 5
- **如何计算某个单词所对应的总分数?**
  - **for语句**遍历单词的每个字母
  - 将所有字母分数累加

【例5.10】程序

```
#（1）输入单词
word=input('请输入你要计算分数的单词：')
new_word=word.lower()

#（2）计算任意长度的字符串对应的总分数

score = 0
for every_item in new_word:
    ascii_value = ord(every_item)
    if ascii_value >= ord('a') and ascii_value <= ord('z'):
        score += ascii_value - ord('a') + 1
    else:
        print('请输入字符！')

#（3）输出结果
print(word,'的分数是：', score)
```

使用字符串的**lower方法**将其转换为小写字符串

单词总分数，初始值设置为0

使用**for语句**遍历单词中每个字母

利用内置函数**ord()函数**获取其ASCII码值

计算该字母的分值，并累加

例5.10-word_score.py

```
>>>
========== RESTART: E:/
请输入你要计算分数的单词：Hardwork
Hardwork 的分数是： 98
>>>
========== RESTART: E:/
请输入你要计算分数的单词：word
word 的分数是： 60
>>>
```

字符串方法

要掌握！请认真学习

字符串方法

- 同其他内建数据类型一样，字符串也有许多的方法，常用的如**find、join、lower、upper、replace、split、strip、translate**等
- lower、upper、replace、strip、translate方法都返回一个字符串的**副本**，即**原字符串不会被改变**

常用字符串方法

方法名称	含义	示例	说明

join	用某个**连接符**将**字符串列表**中各元素连接起来，产生一个**新的字符串。**
	<'连接符'>.join(<字符串列表>)
	将列表转换为字符串
		>>>seq=['1', '2', '3', '4', '5']
		>>> '-'.join(seq)
		'1-2-3-4-5'
		>>> ''.join(seq)　#连接符为空
		'12345'
			它是split方法的逆方法。
			如果连接符**为空**，则是直接把**各元素紧挨着**写在一起

lower	返回一个字符串的**副本**，全部为小写字母。
	<字符串>.lower()
		>>> name='Peter'
		>>> new_name=name.lower()
		>>> new_name
		'peter'
			原字符串不会被改变

upper	返回一个字符串的**副本**，全部为大写字母。
	<字符串>.upper()
		>>> name='Peter'
		>>> new_name=name.upper()
		>>> new_name
		‘PETER'
			原字符串不会被改变

常用字符串方法

方法名称	含义	示例	说明

replace	将某字符串中所有与某指定子字符串**old**相匹配的项用希望被替换的子字符串new替换掉，得到一个新的字符串，并返回该字符串
	<字符串>.replace(old, new[, max])
	其中old指示的匹配项被替换为new，可选择最多替换max个
		>>> words='Peter loves China. Peter is a teacher. '
		>>>new_words=**words.replace('Peter', 'Bob')**
		>>> new_words
		'Bob loves China. Bob is a teacher. '
			原字符串不会被改变。
			可实现文字处理中的“**查找并替换**”功能。

split
将字符串转换为列表
	利用某个分隔符（如“+”)将一个**字符串**分割为序列，返回字符串中所有单词的**列表**。
	<字符串>.split([sep[, maxsplit]])
	其中sep作为分隔符（如果不特别指定则以空格切分单词）；可使用maxsplit指定最大切分数。
		>>> words='Peter loves China. '
		>>>
		words_list=words.split()
		>>> words_list
		['Peter', 'loves', 'China.']
		>>>**input().split()**
			它是join方法的逆方法。
			如果不提供任何分隔符，则Python自动把所有空格作为分隔符（如空格、制表符、换行符）。
			用途：**切分单词。将一行输入的多个数据存入列表**

字符串方法的应用示例：【例5.11】

【例5.11】切分单词，提取出一篇文章中的所有单词，放入一个列表中。注意去除标点符号。

- **设计思路**
  - 文章采用**三引号**括起来，赋给一个变量article
  - 采用**for循环**，遍历一个列表（存放要去除的标点），使用字符串的**replace**方法，将article中所有的**标点**字符替换为**空格**
  - 再对去掉标点后的字符串使用**split**方法，以**空格**分隔，将单词存入列表

【例5. 11】程序

例5.11-string_split.py

```
#（1）将要切分单词的字符串赋值给变量article
article='''All of us have read thrilling stories in which the hero had only
a limited and specified time to live. Sometimes it was as long as a year,
sometimes as short as 24 hours. But always we were interested in discovering
just how the doomed hero chose to spend his last days or his last hours. '''

#（2）使用replace方法，将article中所有的标点字符替换为空格
for i in [', ', '. ']:                          #遍历要去除的标点
    new_article=article.replace(i, ' ') #将article中的','或 '.'替换为空格
    article=new_article #将去掉一个标点的新字符串又赋给article，以便再去除其他标点
```

【例5. 11】程序（续）

```
#（3）对去掉标点后的字符串使用split方法，将字符串转换为列表
words_list=new_article.split()
print (‘new_article:\n’,new_article)        #其中“\n”表示在此处换行
print ('_____')
print ('words_list:\n',words_list)
```

【例5. 11】程序运行结果

去除标点后

```
>>>
new_article:
 All of us have read thrilling stories in which the hero had only
a limited and specified time to live  Sometimes it was as long as a year
sometimes as short as 24 hours  But always we were interested in discovering
just how the doomed hero chose to spend his last days or his last hours

_____________________________________________________________________
words_list:
 ['All', 'of', 'us', 'have', 'read', 'thrilling', 'stories', 'in', 'which', 'the', 'hero'
, 'had', 'only', 'a', 'limited', 'and', 'specified', 'time', 'to', 'live', 'Sometimes', '
it', 'was', 'as', 'long', 'as', 'a', 'year', 'sometimes', 'as', 'short', 'as', '24', 'hou
rs', 'But', 'always', 'we', 'were', 'interested', 'in', 'discovering', 'just', 'how', 'th
e', 'doomed', 'hero', 'chose', 'to', 'spend', 'his', 'last', 'days', 'or', 'his', 'last',
'hours']
>>>
```

- ‘live’、’hours’后面的句号均被去除了！

字符串与列表的相互转换

- **字符串转列表** -- 字符串方法：**split()**

```
>>> sen = 'Good morning everyone'
>>> sen.split()
['Good', 'morning', 'everyone']
>>> sen = 'A-B-C-D'
>>> sen.split('-')
['A', 'B', 'C', 'D']
```

- **列表转字符串** -- 字符串方法：**join()**

```
>>> seq = ['A','B','C','D']
>>> ''.join(seq)
'ABCD'
>>> '-'.join(seq)
'A-B-C-D'
```

- join()方法与split()方法互为**逆操作**
- join方法可以用来为列表中的每个元素添加一个**新**元素（即连接符），并将其连接成一个字符串

使用list函数将字符串转换为列表

- **list函数：将一个序列转换为列表**

格式： list(<序列>)

```
>>>string1= list('teachers')
>>>string1
['t', 'e', 'a', 'c', 'h', 'e', 'r', 's']
```

元组

```
>>>string2= list((10,20,30))
>>>string2
[10, 20, 30]
```

- **list函数适用于所有类型的序列（如元组），而不只是字符串！**

北京航空航天大學
BEIHANG UNIVERSITY

5.5 字典

北京航空航天大学

映射类型——字典

- **字典**是另一个非常有用的Python内建数据类型，Python中**唯一内建的映射类型**
  - 字典是用于存储**值**与**名字**相关联的某一类特殊数据的数据结构
    - ✓ 例如**通讯簿**，一个人的工作单位、电话号码、地址等信息都与其**姓名**相关联

    例：people={‘张三’：‘82337600’，‘李四’：‘82337601’，‘王五’：‘82337602’}，人名是键，电话号码是值

  - 字典中的值没有特定的顺序，打印出来是**随机**的
  - **字典由若干“键/值”对构成**

键值对

- **键值对**由键与值组成

- （1）键必须是**唯一**的，同一个字典中**各个键**，必须**各不相同**
- （2）键必须是任意**不可变**类型，如**数字、字符串或元组**
- （3）**一个键可以对应多个值，每个值可以是一个键值对**
  - 一个人名，可以有电话号码、工作单位、地址

```
people_new={‘张三’:{‘电话’:‘82337600’, ‘地址’:‘新主楼G105’},
            ‘李四’:{‘电话’:‘82337601’, ‘地址’:‘新主楼G106’},
            ‘王五':{'电话':'82337602', '地址':'新主楼G107'}}
```

另一种标准序列类型：元组

- 一个**元组**由数个**逗号分隔**的值组成，例如：

```
>>> t = 12345, 54321, 'hello'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello')
>>> u = t, (1, 2, 3)
>>> u
((12345, 54321, 'hello'), (1, 2, 3))
```

```
>>> x = 3
>>> y = 4
>>> x, y = y, x
>>> print (x, y)
4 3
>>> x, y = '3 4'.split()
>>> x
'3'
>>> y
'4'
```

- 元组在输出时总是带圆括号的，以便于正确表达嵌套结构；在输入时，**有或没有括号都可以，逗号分隔即可。**

如果元组是一个更大的表达式的一部分，则必须使用括号。

1、创建字典

创建字典

- **字典**由多个键和值构成的对组成，每个键值对称为**项**，键与值之间用冒号（：）隔开，项与项之间用逗号“，”隔开，整个字典由一对大括号括起来。空字典如{}所示
- **方法一（最直接）**：直接在{}中写出各项

```
>>> phonebook={'Alice':'02341', 'Peter':'09102', 'Bob':'03356'}
>>> phonebook                          #访问整个字典
{'Bob': '03356', 'Peter': '09102', 'Alice': '02341'}
>>> phonebook['Bob']                   #查找某个键对应的值
‘03356'
```

以0打头的数字必须表示成数字字符串

dict函数

- **方法二**：用dict函数，通过**其他字典**或者**（键，值）**这样的序列对（即**元组**）来创建字典
  - ✓ 先创建一个列表，其中每个元素是一个表示**（键，值）**的元组，其中键可以是数字、字符串或元组：
    列表名=[(键1, 值1)，(键2, 值2)，…]
  - ✓ 再用dict函数创建字典，将列表中的每个元组变成一个键值对：
    字典名=dict(列表名)

当键是字符串时，必须用单引号括起来，否则出错

dict函数（续）

```
>>> items=[('name','Alice'),('age',20),('sex', 'female')]   #创建列表，每个元素是一个表示（键，值）的元组
>>> d=dict(items)                                           #用dict函数创建字典，将列表items中的每个元组变成一个键值对
>>> d                                                       #访问整个字典
{'age': 20, 'name': 'Alice', 'sex': 'female'}
>>> d['name']                                               #查找某个键对应的值
'Alice'
```

当访问整个字典时，打印顺序是**随机**的

2、字典的基本操作

- **字典的基本操作与序列类似**，其中d为字典，k为键
  - ✓ **len(d)**：返回d中项的个数
  - ✓ **d[k]**：返回关联到键k上的值，**如果k是一个字符串，则必须用单引号括起来！**
  - ✓ **d[k]=v**：将值v关联到键k上，相当于**替换**某个已有的键对应的值，或者在字典中**添加**一个键/值对
  - ✓ **del d[k]**：**删除**键为k的**项**
  - ✓ **del d**：**删除字典**
  - ✓ **k in d**：检查d中是否含有键为k的项，并返回值“True”或“False”

必须先创建字典d

- **查找某个键对应的值时，采用<字典>[<键>]的形式**

循环遍历字典元素

- 采用for语句遍历字典的所有键，访问对应的值

```
#for_dic.py

phonebook={'Alice':'2341', 'Peter':'9102', 'Bob':'3356'} #字典
for key in phonebook:                          #遍历整个字典，key表示键
    print (key, 'corresponds to', phonebook[key])      #打印每个键对应的值
```

```
>>>
Bob corresponds to 3356
Peter corresponds to 9102
Alice corresponds to 2341
```

使用print函数打印时，键和值都不显示引号

3、字典方法

- clear、copy、deepcopy、get、keys、pop、popitem、update、values等

① clear方法

- clear方法清除字典中的所有的项，原地操作，返回None
- clear方法的调用：<待清除的字典名>.clear()

```
>>> d={'name':'Alice','age':23}
>>> d.clear()
>>> d
{}
```

② 字典的get方法

- 字典的**get()方法**可以根据键返回值，如果字典中不存在输入的键，返回**None**　　**调用**：**<字典名>.get(<键>[, 'Not found'])**
  - 当get()方法有两个参数时，第一个为**键**，第二个为默认值**'Not found'**。如果字典中不存在输入的键，则返回该默认值。

```
>>> tel = {'gree': 4127, 'jack': 4098, 'shy': 4139}
>>> tel.get('gree')
4127
>>> tel.get('jack', 'Not found')
4098
>>> tel.get('rose', 'Not found')
'Not found'
```

rose不存在

③ 字典的pop方法

③ pop方法

- **pop方法获得对应于给定键的值，然后将这个键值对从字典中移除**
- pop方法的调用：**<被访问的字典名>.pop(<要移除键值对的键>)**

```
>>> d={'name':'Alice','age':23}
>>> d.pop('name')
'Alice'
>>> d
{'age': 23}
```

思考：列表的pop方法是如何调用的?

<被访问的列表名>.pop(<要移除元素的索引>)

④ keys方法

- keys()方法以列表的形式返回字典中所有的键，顺序不定
- keys方法的调用：<字典名>.keys()

```
>>> tel = {'gree': 4127, 'jack': 4098, 'shy': 4139}
>>> tel.keys()
dict_keys(['jack', 'shy', 'gree'])
```

- 如果需要它有序，则要将其转换成列表，再使用sort()方法

```
>>> name=tel.keys()
>>> lis=list(name)
>>> lis
['gree', 'shy', 'jack']
```

```
>>> lis.sort()
>>> lis
['gree', 'jack', 'shy']
```

⑤ values方法

- values方法以列表的形式返回字典中所有的值
- values方法的调用：<字典名>.values()
- 返回值的列表中可以包含重复的元素

```
>>> dic = {'gree': 95, 'jack': 88, 'shy': 73,
'alice':95}
>>> score=dic.values()                    #获取所有的值
>>> score
dict_values([95, 95, 73, 88])

>>> lis_score=list(score)                 #转换为列表
>>> lis_score
[95, 95, 73, 88]
```

【例5.12】字典应用：超市收银系统

【例5.12】超市收银系统

- 一般超市或商店的收银台在对商品扫码之后会得到一串数字（商品代码），每种商品有唯一的商品代码。假设已知所有商品的**名称**、**商品代码**和**单价**
- 编写程序
  - ✓ 输入扫码得到的**商品代码**（假设固定为**6位**数字）
  - ✓ 统计购买的**商品明细（名称、数量、单价、总价）**，存入列表，并打印
  - ✓ 计算购买商品的**总金额**，打印

【例5.12】字典应用：超市收银系统

- **商品目录**

名称	代码	单价
chips	932071	3.50
chocolate	114049	8.00
soap	304985	2.90
Cupcake	955962	4.90
cookie	313903	6.50
milk	243813	3.20
toothbrush	961995	4.80
toothpaste	933328	9.30
teapot	914500	29.80

- **输入**
  - ✓ 每行为一个6位数字，代表所购买的**商品代码**。当所有购买商品输入完毕时，以输入**0**为**结束输入**标志。
- **输出**
  - ✓ **商品明细**（名称、数量、单价、总价）
  - ✓ 购买商品的**总金额**

【例5.12 】设计思路

设计思路

- （1）创建两个**字典names、prices**，分别存储商品**名称**和**单价**，它们的**键**都是**商品代码**

names = {932071: 'chips', 114049: 'chocolate‘,……}

prices = {932071: 3.50, 114049: 8.00,……}

- （2）创建一个**字典numbers**，记录购买的各种商品的**数量**
  - ✓ 采用**for语句**初始化，使各键值对的**初值为0**

numbers = {i:0 for i in names.keys()}

```
numbers={}
for i in names.keys():
    numbers[i] =0
```

i表示**键**，为商品代码；**0**表示各键值对的**初值**为0；for语句表示遍历字典names的键

numbers= {933328: 0, 114049: 0, ……}

【例5.12 】设计思路（续1）

- （3）为实现**不断输入**所购买商品的商品代码，采用**while True**语句，当输入为**0**时使用**break**终止循环

```
while True:
    s = int(input())          #输入购买的商品代码
    numbers[s]+=1             #该商品的数量加1
    if s == 0:
        break
```

字典numbers的键s对应的值加1

【例5.12 】设计思路（续2）

- （4）采用**for**语句统计购买的商品明细，存入列表lis

```
lis = [(names[i],numbers[i],prices[i],numbers[i]*prices[i]) for i in names.keys() if numbers[i]>0 ]
```

  - ✓ 列表中每个元素是一个**元组**，包括4个子元素：名称、数量、单价、总价
  - ✓ if numbers[i]>0——只统计数量**大于0**的商品
  - ✓ 总价=数量*单价=numbers[i]*prices[i]

- （5）遍历列表lis，采用**sum**函数计算购买商品的**总金额**

total=sum(j[3] for j in lis)

j[3]为第j种商品的总价

【例5.12】完整程序

例5.12-超市收银系统py

```
# （1）创建两个字典，分别存储商品名称和单价
# 它们的键都是商品代码
names = {932071: 'chips', 114049: 'chocolate', 304985: 'soap', 955962: 'cupcake',
         313903: 'cookie', 243813: 'milk', 961995: 'toothbrush', 933328: 'toothpaste', 914500: 'teapot'}
prices = {932071: 3.50, 114049: 8.00, 304985: 2.90, 955962: 4.90, 313903: 6.50,
          243813: 3.20, 961995: 4.80, 933328: 9.30, 914500: 29.80}

# （2）创建字典，记录购买的各种商品的数量
# i表示键，为商品代码；0表示各键值对的初值为0；for语句表示遍历字典names的键
#numbers = {i:0 for i in names.keys()}          #为键值对赋初值的简便方法
#也可以像下面这样
numbers={}
for i in names.keys():
    numbers[i] =0

print('字典numbers初值=',numbers)

# （3）输入购买商品的代码，并计算同一种商品的数量。输入为0时终止
while True:
    s = int(input())          #输入购买的商品代码
    #print('s=',s)
    if s == 0:
        break
    numbers[s]+=1             #商品数量加1，即字典numbers的键s所对应的值加1
print('购买商品的数量numbers: \n',numbers)  #"\n"表示换行
print()
```

【例5.12】完整程序（续）

```
#  （4）统计购买的商品明细，存入列表lis
#列表中每个元素是一个元组，包括4个子元素：名称、数量、单价、总价。也可以用列表存储
'''
lis = [(names[i],numbers[i],prices[i],numbers[i]*prices[i]) for i in names.keys() if numbers[i]>0 ]
#只统计数量大于0的商品。总价=数量*单价=numbers[i]*prices[i]
这条语句跟下面分开书写的写法结果相同。但下面的写法更容易理解。
'''
lis=[]
for i in names.keys():      #遍历每个键
    if numbers[i]>0:
        lis.append((names[i],numbers[i],prices[i],numbers[i]*prices[i]))#将已购买商品的明细添加至lis末尾
print('购买的商品lis: \n',lis)

#  （5）计算购买商品的总金额
#total=sum(j[3] for j in lis)  #所购买所有商品的总金额。遍历列表lis，利用sum函数求每个元素的总价之和
#j[3]为第j种商品的总价。上面这样写也可以
total=0
for j in lis:
    total+=j[3]

print("total:%.2f" % total)
```

【例5.12】程序运行结果

```
932071
114049
932071
932071
114049
114049
932071
0
购买商品的数量numbers：
 {933328: 0, 114049: 3, 914500: 0, 243813: 0, 932071: 4, 304985: 0, 955962: 0,
961995: 0, 313903: 0}

购买的商品lis：
 [('chocolate', 3, 8.0, 24.0), ('chips', 4, 3.5, 14.0)]
total:38.00
```

字典类型小结

- 由大括号将元素括起，每个元素是“键:值”的形式
  - d[k]：返回d中键为k的值，若k不存在会提示出错；
  - d[k] = v：将值v与键k关联，若已有值则覆盖；
  - del d[k]：从d中将键为k的项移除；
  - d.clear()：清除d中所有的项，原地操作，返回{}；
  - d.get(k,v)：如果k在d中，返回d[k]，否则返回v；
  - d.pop(k)：从d中将键为k的项移除，并**返回**相应的**值**；
  - d.keys()：返回包含d中**所有键**的列表；
  - d.values()：返回d中**所有值**的列表；
  - k in d：若k在d中，则返回True；
  - len(d)：返回d中元素的数量；
  - for k in d：依据d中keys进行遍历，此过程中请勿删除字典元素

北京航空航天大學
BEIHANG UNIVERSITY

5.6 函数、模块及文件

北京航空航天大学

5.6.1 函数

- **函数**（function）**是由多条语句组成的能够实现特定功能的程序段，可以在程序中被多次调用**
- 需要时，传递不同参数的形式进行调用
- 使用**def（函数定义）语句**创建（或定义）函数
  - 函数的定义一般包括**函数名**、**参数**、**函数体**和**返回值**等4部分，其中**函数名和函数体必不可少**，参数和返回值根据需要定义

函数的定义

- **创建（或定义）函数**

```
def 函数名(参数):
    函数体
    return 返回值
```

- 参数称为**形式参数（形参）**。多个形式参数之间用**逗号**分开。**有时也可以没有参数**
- **return语句**把计算结果返回给调用函数的位置
- 函数可能显式**返回值**，**也可能不返回值**
  - ✓ 例如只是将计算结果打印输出

Python的两类函数

- **Python中有两类函数**

1、Python提供的标准函数，称为内建函数（Build-in Function），用户可以直接使用

- ✓ 如abs(x)、chr(i)、ord(i)、list(x)、tuple(x)、dict(x)、str(x)、print()、input()、range(a,b)

2、自定义函数。即用户自己创建的函数

```
#circle.py 定义函数，计算圆面积
def area(radius):
    s=math.pi*(radius**2)
    return s
```

调用函数

- **调用函数**：**在程序内或其他程序中使用已定义的函数实现特定的功能。**
  - 如果函数在定义时有**形参**，则在调用函数时必须给函数提供实际参数（**实参**）
  - 调用函数时，**实参被传递给形参**，然后执行函数体，直到遇到return语句或者执行完函数体中所有的语句
    - ✓ 若有return语句，将返回“return”之后的**表达式的值**
    - ✓ 若没有return语句，则返回**None**，执行点转移至调用点之后的代码

调用函数的两种方式

- **方式一**：如果函数是在**同一**程序中定义和调用，则调用时格式：

  格式

  ```
  函数名(<实参>)
  ```

- **方式二**：如果函数是分别在**不同**的程序中定义和调用，则在需要调用函数的程序中先**导入定义函数的模块（程序）**；然后再调用该模块中的函数

- 调用时格式：

  格式

  定义函数的程序名称

  ```
  import 模块名
  模块名.函数名(<实参>)
  ```

【例5.13】函数在同一程序中定义和调用

【例5.13】定义一个函数，计算圆面积。然后在同一程序中调用该函数，计算任意半径的圆的面积。

例5.13 -circle_one.py

```
import math

#定义函数，计算圆面积
def area(radius):
    s=math.pi*(radius**2)
    return s

r=float(input('请输入半径：'))
my_s=area(r)
print ('圆面积=%0.1f' % my_s)
```

```
                    #圆面积
                    #调用本程序中的area函数求圆面积
                    #结果只显示一位小数
```

函数的定义

调用函数

```
>>>
请输入半径： 4
圆面积=50.3
```

5.6.2 模块

- **小型**程序一般存储在一个文件中
- 当程序规模**较大**，如何完成较复杂程序的协同？
  - 将不同功能的程序段存储于不同的文件中
  - 每个文件称为一个**模块**（module）
  - 在其他程序中导入该模块，完成相应功能

【例5.14】定义一个程序，计算圆面积和圆周长。然后在另一程序中通过import语句访问该模块，计算任意半径的圆的面积和周长。

【例5.14】：在一个程序中定义函数

```
#circle.py
#函数的定义示例。计算圆面积和周长

import math

#定义函数，计算圆面积
def area(radius):
    s=math.pi*(radius**2)      #圆面积
    return s

#定义函数，计算圆周长
def circumference(radius):
    l=2*math.pi*radius         #圆周长
    return l
```

函数的定义

函数的定义

【例5.14】在另一程序中调用函数

例5.14-circle_top.py

```
import circle                                  #导入circle模块

r=float(input('请输入半径：'))
my_s=circle.area(r)
print ('圆面积=%0.1f' % my_s)

my_l=circle.circumference(r) #调用circle模块中的circumference函数求圆周长
print ('圆周长=%0.1f' % my_l)
```

调用circle模块中的area函数

```
>>>
请输入半径：5
圆面积=78.5
圆周长=31.4
```

5.6.3 文件

- **Python常用的、进行输入和输出的方式**
  - 利用**input函数**通过键盘输入数字或字符串
  - 利用**print函数**在屏幕上显示输出结果
- **缺点**
  - input函数**输入大量数据很繁琐**，且**易出错**
  - print函数的输出结果只能显示在屏幕上，**一旦退出Python，结果就没有了**
- ***如何输入许多数据？如何将计算结果永久保存下来？***

文件

- 计算的输入数据和结果通常使用**文件**来存储
  - Python提供多种便利进行文件创建及访问
- **文件：计算机中由操作系统管理的具有名字的存储区域；文件是存储在辅助存储器上的数据序列**
- 每一个文件都有一个**唯一**的名字
- 文件可以包含**任意**数据类型，通过后缀来区分
  - 【**例**】Word文件.docx，Powerpoint文件.pptx，Excel文件.xlsx，图像文件.jpg（照片）、.gif（图片），音频文件.wav，视频文件.avi、.mpeg，Python文件.py
  - 最简单的就是**文本**文件（.txt）

文件句柄

- 每种操作系统有自己相应的文件系统以进行文件的创建和访问
  - Python通过**文件句柄**（file handle）访问磁盘文件，实现操作系统无关性
  - **nameHandle = open('kids.txt', 'w')**
    - ✓ 通知操作系统创建一个名为kids.txt的磁盘文件，并返回该文件的文件句柄
    - ✓ 参数'w'指明文件可写

- **文件句柄是一个标识符，用来唯一地标识文件，以便应用程序引用文件对象**

在Python中如何实现文件操作?

- **文件操作**：**读取，写入**
- 内置**open函数**创建一个文件对象，作为计算机中文件的一个**链接**；然后通过调用相关的**方法**实现对文件的读写

- **在Python中如何实现文件操作?**
  - **（1）打开文件**：将磁盘上文件与程序中的对象相关联，即通过**open函数**来创建文件对象；
  - **（2）读/写文件**：调用相关的**方法**（如读取文件、写入文件）操作文件对象
  - **（3）关闭文件**：调用**close()方法**。则结束磁盘文件与文件对象之间的对应关系

读取和写入文件之前，都必须先打开文件，否则出错！

常用的文件操作

操　作	功　能
f=open('e:\data.txt','w')	“写”的方式打开文件e:\data.txt，如文件不存在，则自动创建文件。
infile=open('e:\data.txt','r')	以“读”的方式打开文件e:\data.txt。 **“r”可以省略**
infile=open('e:\data.txt')	以“读”的方式打开文件e:\data.txt
string=infile.read()	把文件e:\data.txt的**整个内容**读进**字符串**string
string=infile.read([size])	读取上次读取之后的size个字节到字符串string；若size未定义，则读取文件**剩余的所有字符**

常用的文件操作（续）

操　作	功　能
string=infile.readline([size])	从文件中读取并返回**下一行**（包括行结束符）到一个**字符串**string；若size有定义则返回size个字符
slist=infile.readlines([size])	读取**整个文件**的内容到**字符串列表**slist，其中**每个元素**是文件的**一行**；若size有定义则返回size个字符
s=f.write(string)	把字符串string写入到文件，返回写入字符的长度
f.close()	手动关闭文件

- **思考：read、readline、readlines有何区别?**

（1）open函数的使用

- **open(name[,mode[,buffering]])**
  - ✓ **name**：要打开的磁盘文件的文件名（必须包括**文件名后缀**），字符串格式，是**必选**参数
  - ✓ **mode**（模式）和**buffering**（缓冲）是**可选**参数

例：**fileHandle=open('newfile.txt', 'w')**

#只写模式，创建一个名为newfile.txt的磁盘文件，并返回该文件的**文件句柄**

文件模式参数mode

mode的值	描 述
'r'	**只读模式**（默认模式。**选择只读模式时‘r’可省略**）。读取文件内容。若文件不存在，则输出错误
'w'	**只写模式**。向文件写入内容。若打开的文件**已经存在**，则**原**有**文件**的内容会被**清空**；若文件不存在，则自动创建文件
'a'	**追加模式**。以写入模式打开，若打开的文件已经存在，则在文件末尾**追加**写入
'b'	**二进制模式**（可与其他模式结合使用，例如'rb'表示只读二进制文件，'wb'表示只写二进制文件）
'+'	**可读可写模式**（可与其他模式结合使用，例如'r+'表示读写）
‘t'	以**文本**模式打开（**默认**）

选用的模式与使用的方法要匹配

（2）基本文件方法：读取文件

- **fileHandle.read([size=-1])**，从文件读取**size**个字符；当未给定size或size为负值时，读取文件**剩余的所有字符**，并将文件内容以**字符串**返回
- **fileHandle.readline([size])**，从文件中读取并返回**下一行**（包括行结束符），返回**字符串**；若size有定义则返回**size**个字符
- **fileHandle.readlines([size])**，读取一个文件中的**所有行**，返回**列表**类型，其中每个元素是文件的一行；若size有定义则返回**size**个字符

- 当进行读取文件操作后，则该文件对象中只剩下未读取的内容
- 如果使用**fileHandle.readlines()**方法，则读取该文件中的**所有行**，操作结束后文件对象为**空**

（3）基本文件方法：写入文件

- **fileHandle.write(s)**，将字符串s写入文件**末端**，并**返回写入字符的长度**。
- **fileHandle.writelines(SL)**，SL为字符串的序列，将该序列的每个元素一次性写入文件末端
- **fileHandle.close()**，关闭文件

紧跟文件已有内容的最后一个字符写入

（1） 如果原文件中有内容，**只写模式**下write()方法会**清空原内容，写入新内容。**write()方法**有返回值，返回的是写入的字符总数**

（2）write()方法并不会在s后加上一个换行符。如果你希望加入的字符串末尾换行，则**需在字符串末尾加上换行符”\n“**

（3）**牢记使用close方法及时关闭文件**！避免在某些操作系统或设置中进行无用的修改；或避免因程序崩溃，导致写入缓存的数据并没有写入文件

参见read_write.py

【例5.15】文件读写示例：合并文件

- **思考**：**如何在一个已有文件后写入新的内容?**

- **在open函数中采用追加模式**
  - `f = open('hello.txt','a')`

【例5.15】合并文件。现有两个磁盘文件string1.txt和string2.txt文件，请读取两个文件的内容，合并，并写入一个新的磁盘文件中。

【例5.15】设计思路

设计思路

- **（1）打开文件**string1.txt 并**读取文件**：通过**open函数**打开string1.txt；调用**read()方法**从文件读取所有字符，以字符串返回
- **（2）打开文件**string2.txt 并**读取文件**
- **（3）合并字符串：**采用“+”将读取的两个字符串合并
- **（4）写入新文件**：通过**open函数**以只写模式创建一个新的文件对象；调用**write(s)方法**，将合并后字符串写入一个新文件merge_file.txt
- **（5）关闭文件**：调用**close()方法**

【例5. 15】程序

例5.15-file_merge.py

```
#（1）打开文件，读取文件string1.txt
f=open('string1.txt')                    #只读模式，创建文件对象
str1=f.read()                            #把文件string1.txt的所有内容读进字符串str1
print('string1.txt的内容为：\n',str1,sep='')  #sep=''表示打印的多个值之间不空格（默认是空一格）
print()
f.close()                                #关闭文件

#（2）打开文件，读取文件string2.txt
fp=open('string2.txt')
str2=fp.read()                           #把文件string2.txt的所有内容读进字符串
str2
print('string2.txt的内容为：\n',str2,sep='')
print()
fp.close()
```

读写文件之前一定要先打开文件！

必须关闭文件后再打开另一个文件

【例5. 15】程序（续）

```
#（3）将读取的两个字符串合并
str_merge=str1+ '\n' + str2          #第1个字符串的末尾换行，再连接第2个字符串
print('合并后的字符串为：\n',str_merge,sep='')

#（4）将合并后字符串写入新文件
f=open('merge_file.txt', 'w')        #只写模式，创建文件对象
f.write(str_merge)                   #向文件对象写入字符串

f.close()                            #关闭文件
```

【例5. 15】程序运行结果

```
string1.txt的内容为：
Hello,World!
我是中国人
我爱我的祖国

string2.txt的内容为：
I love BUAA.
我是北航人
空天报国，砥砺前行

合并后的字符串为：
Hello,World!
我是中国人
我爱我的祖国
I love BUAA.
我是北航人
空天报国，砥砺前行
```

```
f=open('string1.txt')
str1=f.read()
```

将string1.txt文件**所有内容**读进字符串

```
str_merge=str1+ '\n' + str2
f=open(‘merge_file.txt', 'w')
f.write(str_merge)
```

欲读取文件的存放路径

当要读取的文件放在不同的路径下时，open函数中的参数name可能不同（有时需要包含路径，有时则可以省略路径），视要读取的文件与源程序是否在同一路径下而定

最好使用正斜线，以免引起歧义

(1) 当把文件放在E盘根目录下时

f1=open('E:/my.txt') 或 f1=open('E:\my.txt')

(2) 当要读取的文件与源程序不在同一路径下时

f1=open('E:/course/Computer/2021/Python_example/chapter3/my.txt')

(3) 如果要读取的文件与源程序在同一路径下，则可以省略路径

f1=open('my.txt')

为避免找不到文件，最好将要读取的文件与源程序在同一路径下！