第3章

数据结构与函数设计

本章学习目标：

- 熟练掌握序列的基本概念
- 熟练掌握列表、元组、字典、字符串的概念和各种用法
- 熟练掌握各种序列类型之间的转化
- 了解集合的基本概念和用法
- 熟练掌握自定义函数的设计和使用
- 深入了解各类参数以及传递过程

本章主要介绍两方面内容，一是常用的数据结构，二是函数设计。在数据结构方面，先介绍序列的基本概念，然后介绍各种序列类型，包括列表、元组、字符串和字典，最后讲解集合的概念和用法；在函数设计方面，先介绍函数的定义，接着对函数的返回值和形参、实参、默认参数、关键参数、可变长度参数、序列参数等各类参数进行介绍，由此完成对函数的比较细致、全面的讲解。

3.1 序列

在 Python 中，最基本的数据结构是序列（sequence）。序列中的每个元素被分配一个序号，即元素的位置，也称为索引。第 1 个索引是 0，第 2 个是 1，以此类推。序列

中的最后一个元素标记为－1，倒数第 2 个元素为－2，以此类推。

Python 中包含 6 种内建的序列，即列表、元组、字符串、Unicode 字符串、buffer 对象和 xrange 对象。本章重点讨论列表和元组，列表和元组的主要区别在于列表可以修改，而元组不能。

所有序列类型都可以进行某些特定的操作，这些操作包括索引(indexing)、分片(sliceing)、加(adding)、乘(multiplying)以及检查某个元素是否属于序列的成员(成员资格)。除此之外，Python 还有计算序列长度、找出最大元素和最小元素的内建函数。

3.1.1 列表


视频讲解

列表由一系列按特定顺序排列的元素组成。用户可以创建包含字母表中所有字母、数字 0～9 或所有家庭成员姓名的列表；也可以将任何内容加入到列表中，其中的元素之间可以没有任何关系。鉴于列表通常包含多个元素，给列表指定一个表示复数的名称(例如 letters、digits 或 names)是个不错的主意。在 Python 中，用方括号([])来表示列表，并用逗号来分隔其中的元素。下面是一个简单的列表示例，这个列表包含几种自行车。

【例 3-1】 列表实例。

```
bicycles.py
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)
```

如果让 Python 将列表打印出来，Python 将打印列表的内部表示，包括方括号，即['trek', 'cannondale', 'redline', 'specialized']，但这不是要让用户看到的输出。下面来学习如何访问列表元素。

列表是有序集合，因此要访问列表中的任何元素，只需将该元素的位置或索引告诉 Python 即可。如果要访问列表元素，可以先指出列表的名称，再指出元素的索引，并将其放在方括号内。例如，代码“bicycles＝['trek', 'cannondale','redline', 'specialized']print(bicycles[0])”从列表 bicycles 中提取第一款自行车。当用户请求获取列表元素时，Python 只返回该元素(即 trek)，而不包括方括号和引号，这正是想要看到的整洁、干净的输出。用户还可以对任何列表元素调用第 2 章介绍的字符串

方法，例如可以使用 title()方法让元素'trek'的格式更整洁，即“ bicycles＝['trek', 'cannondale', 'redline', 'specialized'] print(bicycles[0]. title())”，这个示例的输出与前一个示例相同，只是首字母 T 是大写的。

例 3-2 为 Python 的列表操作。

【例 3-2】 列表操作实例。

```
sample_list = ['a','b',0,1,3]
```

得到列表中的某一个值：

```
value_start = sample_list[0]
end_value = sample_list[-1]
```

删除列表的第 1 个值：

```
del sample_list[0]
```

在列表中插入一个值：

```
sample_list.insert(0,'sample value')
```

得到列表的长度：

```
list_length = len(sample_list)
```

列表遍历：

```
for element in sample_list:print(element)
```

下面是 Python 列表的高级操作和技巧。

产生一个数值递增列表：

```
num_inc_list = range(30)              #返回列表[0,1,2,...,29]
```

用某个固定值初始化列表：

```
initial_value = 0
list_length = 5
```

```
sample_list = [initial_value for i in range(10)]
sample_list = [initial_value] * list_length
# sample_list == [0,0,0,0,0]
```

list 的方法：

```
L.append(var)       # 追加元素
L.insert(index,var) # 在指定位置插入元素
L.pop(var)          # 返回最后一个元素,并从 list 中将其删除
L.remove(var)       # 删除第一次出现的该元素
L.count(var)        # 该元素在列表中出现的个数
L.index(var)        # 该元素的位置,无则抛出异常
L.extend(list)      # 追加 list,即合并 list 到 L 上
L.sort()            # 排序
L.reverse()         # 倒序
```

list 操作符：、＋、＊，关键字 del：

```
a[1:]               # 片段操作符,用于子 list 的提取
[1,2] + [3,4]       # 为[1,2,3,4],同 extend()
[2] * 4             # 为[2,2,2,2]
del L[1]            # 删除指定下标的元素
del L[1:3]          # 删除指定下标范围的元素
```

list 的复制：

```
L1 = L              # L1 为 L 的别名,用 C 语言来说就是指针地址相同,对 L1 操作即对 L 操作
                    # 函数参数就是这样传递的
L1 = L[:]           # L1 为 L 的克隆,即另一个副本
```

3.1.2 元组

创建元组(即常量数组)：

```
tuple = ('a', 'b', 'c', 'd', 'e')
```

元组可以用 list 的[]操作符提取元素，但不能直接修改元素。

元组的操作：索引、切片、连接、重复。

【例 3-3】　元组操作实例。

```
t = ("fentiao", 5, "male")
#正向索引
print(t[0])
#反向索引
print(t[-1])
#元组嵌套时元素的访问
t1 = ("fentiao", 5, "male", ("play1", "play2", "play3") )
print(t1[3][1])
#切片
print(t[:2])
#逆转元组元素
print(t[::-1])
#连接
print(t + t1)
#重复
t * 3
```

3.1.3　字符串

字符串是 Python 语言中的一种数据类型。字符串由任意字符构成，一个字符可能是一个字母、数值、符号或者标点符号。字符串是用来记录文本信息的，它们在 Python 中作为序列，也就是说一个包含其他对象的有序集合。序列中的元素包含了一个从左到右的顺序，序列中的元素根据它们的相对位置进行存储和读取。从严格意义上说，字符串是单个字符的字符串的序列。

例如：

```
str = "Hello My friend"
```

字符串是一个整体，如果用户想直接修改字符串的某一部分，则是不可能的，但能够读出字符串的某一部分。

子字符串的提取：

```
str[:6]
```

字符串包含判断操作符：in、not in

```
"He" in str
"she" not in str
```

string 模块提供的方法：

```
S.find(substring, [start [,end]])          #可指定范围查找子串,返回索引值,否则返回-1
S.rfind(substring,[start [,end]])          #反向查找
S.index(substring,[start [,end]])          #同 find(),只是找不到产生 ValueError 异常
S.rindex(substring,[start [,end]])         #同上,反向查找
S.count(substring,[start [,end]])          #返回找到子串的个数
S.lowercase()                              #首字母小写
S.capitalize()                             #首字母大写
S.lower()                                  #转小写
S.upper()                                  #转大写
S.swapcase()                               #大小写互换
S.split(str, ' ')                          #将 string 转 list,以空格切分
S.join(list, ' ')                          #将 list 转 string,以空格连接
```

处理字符串的内置函数：

```
len(str)                                   #串长度
cmp("my friend", str)                      #字符串比较,第一个大,返回 1
max('abcxyz')                              #寻找字符串中最大的字符
min('abcxyz')                              #寻找字符串中最小的字符
```

string 的转换：

```
float(str)                                 #变成浮点数,float("1e-1")的结果为 0.1
int(str)                                   #变成整型,int("12") 的结果为 12
int(str,base)                              #变成 base 进制整型数,int("11",2)的结果为 2
long(str)                                  #变成长整型
long(str,base)                             #变成 base 进制长整型
```

字符串的格式化(注意其转义字符,大多如 C 语言)：

```
str_format % (参数列表)      #参数列表是以 tuple 的形式定义的,即不可以在运行中改变
```

【例 3-4】 字符串操作实例。

```
>>> print("%s's height is %dcm" % ("My brother", 180))
My brother's height is 180cm
```

3.1.4　列表与元组之间的转换

使用 list()和 tuple()函数进行元组和列表的相互转换。

```
tuple(ls)
list(ls)
```

3.2　字典

视频讲解

字典是一种通过名字或者关键字引用的数据结构，其键可以是数字、字符串、元组，这种结构类型也称为映射。字典类型是 Python 中唯一内建的映射类型。

3.2.1　创建字典

（1）直接创建字典：

```
d = {'one':1,'two':2,'three':3}
```

（2）通过 dict()创建字典：

```
#_*_ coding:utf-8 _*_
items = [('one',1),('two',2),('three',3),('four',4)]
d = dict(items)
print(d)
```

（3）通过关键字创建字典：

```
#_*_ coding:utf-8 _*_
d = dict(one = 1,two = 2,three = 3)
print(d)
print(d['one'])
print(d['three'])
```

（4）字典的格式化字符串：

```
#_*_ coding:utf-8 _*_
d = {'one':1,'two':2,'three':3,'four':4}
```

```
print(d)
print("three is %(three)s." % d)
```

3.2.2 字典的方法

每一个元素是 pair,包含 key、value 两部分。key 是 integer 或 string 类型,value 是任意类型。键是唯一的,字典只认最后一个赋的键值。

dictionary 的方法:

```
D.get(key, 0)          # 同 dict[key],如果指定键的值不存在,返回默认值
D.has_key(key)         # 有该键返回 True,否则返回 False
D.keys()               # 返回字典键的列表
D.values()
D.items()
D.update(dict2)        # 增加合并字典
D.popitem()            # 得到一个 pair,并从字典中删除它,若已空则抛出异常
D.clear()              # 清空字典,同 del dict
D.copy()               # 复制字典
D.cmp(dict1,dict2)     # 比较字典(优先级为元素个数、键值大小)
                       # 第一个大返回 1,小返回-1,一样返回 0
```

dictionary 的复制:

```
dict1 = dict          # 别名
dict2 = dict.copy()   # 克隆,即另一个副本
```

3.2.3 列表、元组与字典之间的转换

这三者之间的转换并不复杂,但字典的转换由于有 key 的关系,所以其他二者不能转换为字典。

(1) 对元组进行转换:

```
>>> fruits = ('apple','banana','orange')          # 元组转换为列表
>>> list(fruit)
```

(2) 对列表的转换:

```
>>> fruit_list = ['apple','banana','orange']      # 列表转换为元组
>>> tuple(fruit_list)
```

（3）对字典的转换：用户可以使用 tuple()和 list()函数将字典转换为元组和列表，但要注意这里转换后和转换前的元素顺序是不同的，因为字典类似于散列，列表类似于链表，元组类似于列表，只是元素无法改变，所以要把散列转换为链表而顺序不变是不可行的。此时可以借助于有序字典(OrderedDict)，有序字典是字典的子类，它可以记住元素添加的顺序，从而得到有序的字典。对于有序字典这里不深入探讨，只给出普通字典的例子做参考，代码如下：

【例 3-5】 字典转换例子。

```
>>> fruit_dict = {'apple':1, 'banana':2, 'orange':3}
>>> tuple(fruit_dict)                  # 将字典的 key 转换为元组
>>> tuple(fruit_dict.value())          # 将字典的 value 转换为元组
>>> list(fruit_dict)                   # 将字典的 key 转换为列表
>>> list(fruit_dict.value())           # 将字典的 value 转换为列表
```

3.3　集合

3.3.1　集合的创建

在 Python 中集合由内置的 set 类型定义，如果要创建集合，需要将所有项(元素)放在花括号({})内，以逗号(,)分隔。

【例 3-6】 集合实例。

```
>>> s = {'P', 'y', 't', 'h', 'o', 'n'}
>>> type(s)
<class 'set'>
```

集合可以有任意数量的元素，它们可以是不同的类型(例如数字、元组、字符串等)，但是集合不能有可变元素(例如列表、集合或字典)：

```
>>> s = {1, 2, 3}                                    # 整型的集合
>>> s = {1.0, 'Python', (1, 2, 3)}                   # 混合类型的集合
>>> s = set(['P', 'y'])                              # 从列表创建
>>> s = {1, 2, [3, 4]}                               # 不能有可变元素
TypeError: unhashable type: 'list'
```

创建空集合比较特殊，在 Python 中空花括号({})用于创建空字典。如果要创建

一个没有任何元素的集合，使用 set()函数(不要包含任何参数)：

```
>>> d = {}                              #空字典
>>> type(d)
<class 'dict'>
>>> s = set()                           #空集合
>>> type(s)
<class 'set'>
```

回顾数学的相关知识，发现集合具有以下特性。

(1) 无序性：在一个集合中，每个元素的地位都是相同的，元素之间是无序的。在集合上可以定义序关系，在定义了序关系之后元素之间就可以按照序关系排序，但就集合本身的特性而言，元素之间没有必然的序。

(2) 互异性：在一个集合中，任何两个元素都认为是不相同的，即每个元素只能出现一次。有时需要对同一元素出现多次的情形进行刻画，此时可以使用多重集，其中的元素允许出现多次。

(3) 确定性：给定一个集合，任意一个元素，该元素或者属于或者不属于该集合，二者必是其一，不允许有模棱两可的情况出现。

当然，Python 中的集合也具有这些特性。例如：

```
#无序性
>>> s = set('Python')
>>> s
{'y', 'n', 'h', 'o', 'P', 't'}
>>> s[0]                                    #不支持索引
TypeError: 'set' object does not support indexing
#互异性
>>> s = set('Hello')
>>> s
{'e', 'H', 'l', 'o'}
#确定性
>>> 'l' in s
True
>>> 'P' not in s
True
```

注意：由于集合是无序的，所以索引没有任何意义。也就是说，无法使用索引或切片访问或更改集合元素。

3.3.2　集合的运算

集合之间也可以进行数学集合运算(例如并集、交集等),可用相应的操作符或方法来实现。考虑 A、B 两个集合,进行以下操作。

【例 3-7】 集合运算实例。

```
>>> A = set('abcd')
>>> B = set('cdef')
```

1. 子集

子集为某个集合中一部分的集合,故也称部分集合。

使用操作符"<"执行子集操作,同样,也可以使用 issubset()方法完成。例如:

```
>>> C = set('ab')
>>> C < A
True
>>> C < B
False
>>> C.issubset(A)
True
```

2. 并集

一组集合的并集是这些集合的所有元素构成的集合,而不包含其他元素。

使用操作符"|"执行并集操作,同样,也可以使用 union()方法完成。例如:

```
>>> A | B
{'e', 'f', 'd', 'c', 'b', 'a'}
>>> A.union(B)
{'e', 'f', 'd', 'c', 'b', 'a'}
```

3. 交集

两个集合 A 和 B 的交集是含有所有既属于 A 又属于 B 的元素且没有其他元素的集合。

使用操作符"&"执行交集操作,同样,也可以使用 intersection()方法完成。例如:

```
>>> A & B
{'d', 'c'}
>>> A.intersection(B)
{'d', 'c'}
```

4. 差集

A 与 B 的差集是所有属于 A 但不属于 B 的元素构成的集合。

使用操作符“-”执行差集操作，同样，也可以使用 difference()方法完成。例如：

```
>>> A - B
{'b', 'a'}
>>> A.difference(B)
{'b', 'a'}
```

5. 对称差

两个集合的对称差是只属于其中一个集合，而不属于另一个集合的元素组成的集合。

使用操作符“^”执行对称差操作，同样，也可以使用 symmetric_difference()方法完成。例如：

```
>>> A ^ B
{'b', 'e', 'f', 'a'}
>>> A.symmetric_difference(B)
{'b', 'e', 'f', 'a'}
```

6. 更改集合

虽然集合中不能有可变元素，但集合本身是可变的。也就是说，可以添加或删除其中的元素。

用户可以使用 add()方法添加单个元素，使用 update()方法添加多个元素，update()可以使用元组、列表、字符串或其他集合作为参数。例如：

```
>>> s = {'P', 'y'}
>>> s.add('t')                              # 添加一个元素
>>> s
{'P', 'y', 't'}
>>> s.update(['h', 'o', 'n'])               # 添加多个元素
>>> s
{'y', 'o', 'n', 't', 'P', 'h'}
```

```
>>> s.update(['H', 'e'], {'l', 'l', 'o'})        #添加列表和集合
>>> s
{'H', 'y', 'e', 'o', 'n', 't', 'l', 'P', 'h'}
```

在所有情况下元素都不会重复。

7. 从集合中删除元素

用户可以使用 discard()和 remove()方法删除集合中特定的元素。

两者之间唯一的区别在于如果集合中不存在指定的元素,使用 discard()结果保持不变,但在这种情况下 remove()会引发 KeyError。例如:

```
>>> s = {'P', 'y', 't', 'h', 'o', 'n'}
>>> s.discard('t')                              #去掉一个存在的元素
>>> s
{'y', 'o', 'n', 'P', 'h'}
>>> s.remove('h')                               #删除一个存在的元素
>>> s
{'y', 'o', 'n', 'P'}
>>> s.discard('w')                              #去掉一个不存在的元素(正常)
>>> s
{'y', 'o', 'n', 'P'}
>>> s.remove('w')                               #删除一个不存在的元素(引发错误)
KeyError: 'w'
```

类似地,用户可以使用 pop()方法删除和返回一个项目,还可以使用 clear()方法删除集合中的所有元素。例如:

```
>>> s = set('Python')
>>>
>>> s.pop()                                     #随机返回一个元素
'y'
>>>
>>> s.clear()                                   #清空集合
```

注意:集合是无序的,所以无法确定哪个元素将被 pop,完全随机。

3.3.3　集合的方法

使用 dir()来查看方法列表:

```
>>> dir(set)
['__and__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__'
, '__ge__', '__getattribute__', '__gt__', '__hash__', '__iand__', '__init__', '__ior__', '__isub__',
'__iter__', '__ixor__', '__le__', '__len__', '__lt__', '__ne__','__new__', '__or__', '__rand__', '_
_reduce__', '__reduce_ex__', '__repr__', '__ror__','__rsub__', '__rxor__', '__setattr__', '_
_sizeof__', '__str__', '__sub__', '__subclasshook__', '__xor__', 'add', 'clear', 'copy', '
difference', 'difference_update', 'discard', 'intersection', 'intersection_update', '
isdisjoint', 'issubset', 'issuperset', 'pop', 'remove', 'symmetric_difference', 'symmetric_
difference_update', 'union', 'update']
```

可以看到有表 3.1 所示的方法可用。

表 3.1　集合的方法

方　法	描　述
add()	将元素添加到集合中
clear()	删除集合中的所有元素
copy()	返回集合的浅复制
difference()	将两个或多个集合的差集作为一个新集合返回
difference_update()	从一个集合中删除另一个集合的所有元素
discard()	删除集合中的一个元素(如果元素不存在,则不执行任何操作)
intersection()	将两个集合的交集作为一个新集合返回
intersection_update()	用自己和另一个的交集来更新这个集合
isdisjoint()	如果两个集合有一个空交集,返回 True
issubset()	如果另一个集合包含这个集合,返回 True
issuperset()	如果这个集合包含另一个集合,返回 True
pop()	删除并返回任意的集合元素(如果集合为空,会引发 KeyError)
remove()	删除集合中的一个元素(如果元素不存在,会引发 KeyError)
symmetric_difference()	将两个集合的对称差作为一个新集合返回
symmetric_difference_update()	用自己和另一个的对称差来更新这个集合
union()	将集合的并集作为一个新集合返回
update()	用自己和另一个的并集来更新这个集合

其中一些方法在上述示例中已经用过,如果有些方法用户不会用,可使用 help() 函数查看其用途及详细说明。

1. 集合与内置函数

表 3.2 中的内置函数通常作用于集合执行不同的任务。

表 3.2　集合的内置函数

函　　数	描　　述
all()	如果集合中的所有元素都是 True(或者集合为空),则返回 True
any()	如果集合中的所有元素都是 True,则返回 True; 如果集合为空,则返回 False
enumerate()	返回一个枚举对象,其中包含了集合中所有元素的索引和值(配对)
len()	返回集合的长度(元素个数)
max()	返回集合中的最大项
min()	返回集合中的最小项
sorted()	从集合中的元素返回新的排序列表(不排序集合本身)
sum()	返回集合的所有元素之和

2. 不可变集合

frozenset 是一个具有集合特征的新类,但是一旦分配,它里面的元素就不能更改。这一点和元组非常类似: 元组是不可变的列表,frozenset 是不可变的集合。集合是 unhashable 的,因此不能用作字典的 key; 而 frozenset 是 hashable 的,可以用作字典的 key。用户可以使用 frozenset()函数创建 frozenset,例如:

```
>>> s = frozenset('Python')
>>> type(s)
<class 'frozenset'>
```

frozenset 也提供了一些方法,和 set 中的类似,同样使用 dir()查看:

```
>>> dir(frozenset)
['__and__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__
lt__', '__ne__', '__new__', '__or__', '__rand__', '__reduce__', '__reduce_ex__', '__repr__', '__ror__'
, '__rsub__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__xor
__', 'copy', 'difference', 'intersection', 'isdisjoint', 'issubset', 'issuperset', 'symmetric_
difference', 'union']
```

由于 frozenset 是不可变的,所以没有添加或删除元素的方法。

3.4　函数的定义

函数是一段按逻辑组织好的、可重复使用来实现单一或者相关联功能的代码,使用函数能有效地提高应用的模块性和代码的重复利用率。在 Python 中已提供了许多内建函数,例如 print()。用户也可以自己创建函数来满足特定的需求,这种函数

被称为用户自定义函数。本节的主要内容就是用户自定义函数的设计和实现。

在 Python 中用户可以根据自己的需求自定义函数，但是在定义函数的过程中必须遵循以下几条简单的规则：

（1）函数代码块以 def 关键字开头，后接函数标识符名称和圆括号“()”。

（2）任何传入参数和自变量必须放在圆括号之内，圆括号之间用于定义参数。

（3）函数的第 1 行语句可以选择性地使用文档字符串(用于存放函数说明)。

（4）函数内容以冒号起始，并且缩进。

（5）“return［表达式］”结束函数，选择性地返回一个值给调用方，不带表达式的 return 相当于返回 None。

下面是定义函数的一般语法：

```
>>> def functionname(parameters):
        """函数_文档字符串"""
            function_suite
            return [expression]
```

在默认情况下，参数值和参数名称都是按函数声明中参数定义时的顺序匹配起来的。下面是自定义函数的一个简单实例，用来打印简单的问候语。

【例 3-8】 自定义函数实例。

```
>>> def greeting():
 """显示简单的问候语"""
         print("Hello!Python world")
>>> greeting()
Hello!Python world
```

这个实例演示了最简单的自定义函数结构。第 1 行代码使用关键字 def 来告诉 Python 接下来要定义一个函数。这是函数定义向 Python 指出了函数名，还可能在括号内指出函数为完成其任务需要哪些参数。在这里函数名为 greeting()，它不需要任何参数就能完成其工作，因此括号内是空的(即便如此，括号也必不可少)。最后定义以冒号结尾。

紧跟在“def greeting():”后面的所有缩进行构成了函数体。引号内的文本被称为注释，描述了该函数是用来做什么的。文档字符串用三引号括起，Python 使用它们来生成有关程序中函数的文档。

代码行 print("Hello! Python world")是函数体内唯一的一行代码，greeting()只做一项工作，即打印“Hello! Python world”。

如果要使用这个函数，可调用它，函数调用让 Python 执行函数的代码。如果要调用函数，可依次指定函数名以及用括号括起来的必要的参数，如第 4 行代码所示。由于这个函数不需要任何参数，因此在调用它时只需要输入 greeting()即可，和预期的一样，它打印“Hello! Python world”。

3.4.1　函数的调用


视频讲解

定义一个函数只是给了函数一个名称，以及指定了函数中应该包含哪些参数和代码块结构。当一个函数的基本结构完成以后，就可以在另一个函数里调用执行，当然也可以直接从 Python 提示符执行。下面通过实例来介绍如何调用自定义函数。

【例 3-9】 函数调用实例 1。

```
#定义函数
>>> def printme( strings ):
 """打印任何传入的字符串"""
      print(strings)
      return

#调用函数
>>> printme("我要调用用户自定义函数!")
>>> printme("再次调用同一函数")
```

以上实例的输出结果：

```
我要调用用户自定义函数!
再次调用同一函数
```

3.4.2　形参与实参

在前面定义 greeting()函数时并没有指定参数，现在假设给函数指定参数 username，调用这个函数并提供这种信息(人名)，它将打印相应的问候语。

这样，在 greeting()函数的定义中变量 username 就是一个形参(函数完成其工作所需的一项信息)。在函数的调用代码 greeting('Tom')中，值'Tom'是一个实参。实

参是调用函数时传递给函数的信息。在调用函数时将要让函数使用的信息放在括号内。在 greeting('Tom')中将实参'Tom'传递给了 greeting()函数,这个值被存储在形参 username 中。

3.4.3 函数的返回


视频讲解

在 Python 中函数的返回一般通过 return 语句实现,return 语句退出函数,并且选择性地向调用方返回一个表达式,不带参数值的 return 语句返回 None。在前面的几个例子中都没有示范如何返回数值,下面的实例将对此进行介绍。

【例 3-10】 函数调用实例 2。

```
#可写函数说明
>>> def sum(arg1, arg2):
    #返回两个参数的和
      total = arg1 + arg2
      print("函数内 : ", total)
      return total

#调用 sum()函数
>>> total = sum(10, 20)
```

以上实例的输出结果:

```
函数内 : 30
```

3.4.4 位置参数


视频讲解

在调用函数时,Python 必须将函数调用中的每个实参都关联到函数定义中的每个形参。因此,最简单的关联方式是基于实参的顺序,这种关联方式被称为位置参数。简单地说,就是在给函数传参数时按照顺序依次传值。下面通过实例进行介绍。

【例 3-11】 位置参数实例。

```
>>> def power(m, n):
        result = 1
        while n > 0:
            n = n-1
            result = result * m
        return result
```

调用函数并输出结果：

```
>>> print(power(4,3))
64
```

在 power(m,n)函数中有两个参数，即 m 和 n，这两个参数都是位置参数，在调用的时候传入的两个值按照顺序依次赋给 m 和 n。

3.4.5　默认参数与关键字参数


视频讲解

所谓默认参数，就是在写函数的时候直接给参数传默认的值，在调用的时候默认参数已经有值，这样就不用再传值了，最大的好处就是降低调用函数的难度。

修改例 3-11，见例 3-12。

【例 3-12】 默认参数实例。

```
>>> def power(m, n = 3):
        result = 1
            while n > 0:
            n = n-1
            result = result * m
        return result
```

调用函数并输出结果：

```
>>> print(power(4))
64
```

在修改后的函数中，对第 2 个形参 n 设置了一个默认值 3，在之后的函数调用中不提供该函数的实参，但函数仍旧能正常运行，因为在这个函数中 n 已经是一个默认参数。在设置默认参数时需要注意两点：一是必选参数在前，默认参数在后，否则 Python 解释器会报错；二是默认参数一定要指向不变对象。

关键字参数和函数调用关系紧密，在函数调用时可以通过使用关键字参数来确定传入的参数值。其最显著的特征就是使用关键字参数将允许函数调用时参数的顺序与声明时不一致，因为 Python 解释器能够用参数名匹配参数值。

下面通过实例来介绍关键字参数。

【例 3-13】 关键字参数实例 1。

```
#可写函数说明
>>> def printme(strings):
      """打印任何传入的字符串"""
          print(strings)
          return

#调用 printme()函数
>>> printme(strings = "My string")
```

以上实例的输出结果：

```
My string
```

下例能将关键字参数的顺序不重要展示得更清楚。

【例 3-14】 关键字参数实例 2。

```
#可写函数说明
>>>def printinfo(name, age):
     """打印任何传入的字符串"""
      print("Name: ", name)
      print("Age ", age)
      return

#调用 printinfo()函数
>>> printinfo( age = 40, name = "James")
```

以上实例的输出结果：

```
Name: James
Age 40
```

3.4.6 可变长度参数

在 Python 函数中还可以定义可变长度参数。顾名思义，可变长度参数就是所传入参数的个数是可变的，可以是一个、两个到任意个，也可以是 0 个。

和前面 3 种参数不同，可变长度参数在声明时不会全部命名。其基本语法如下：

```
>>> def functionname([formal_args,] * var_args_tuple):
    """函数_文档字符串"""
        function_suite
        return [expression]
```

注意，加了星号（*）的变量名会存放所有未命名的变量参数。

可变长度参数的实例如下：

【例 3-15】 可变长度参数实例。

```
#可写函数说明
>>> def printinfo(arg1, *vartuple):
    """打印任何传入的参数"""
        print("输出：")
        print(arg1)
        for var in vartuple:
            print(var)
            return

#调用 printinfo()函数
>>> printinfo(15)
>>> printinfo(75, 65, 55)
```

以上实例的输出结果：

```
15
75
65
55
```

本章小结

本章的主要内容分为两个部分。在第一部分介绍了 Python 中常见的数据结构，包括序列（例如列表、元组）、映射（例如字典）和集合等，主要包括序列的基本概念，序列的各种方法，字符串的两种重要使用方式（字符串格式化与字符串方法），利用字典格式化字符串以及字典的用法，集合的创建、运算和方法。

在第二部分介绍了自定义函数的基本语法、几种形式的函数参数、返回值。自定义函数由 def 开头，接下来确定函数名、形参的类型数量，还要加冒号，最后缩进写入函数体。形参是可选的，可有可无；同样，函数可以有返回值，也可以没有返回值，其主要通过 return 语句返回，也就是通过 return 语句将程序的控制权返回给函数的调用者。Python 的函数具有非常灵活的参数形态，既可以实现简单的调用，又可以传入非常复杂的参数。函数参数可以作为位置参数或者关键字参数进行传递，并且可

以使用默认参数传递，以及使用可变长度参数进行传递。

习题

1. 掌握并比较不同数据类型的异同。

2. 利用循环创建一个包含 100 个偶数的列表，并且计算该列表的和与平均值。请分别使用 while 和 for 循环实现。

3. 编写一个函数实现根据本金、年利率、存款年限计算得到本金和利息的功能，调用这个函数计算 10 000 元本金在银行以 5.5%的年利率存 5 年后获得的本金和利息。在该题中按复利计算利息。

4. 编写一个函数实现判断一个输入的数字是否为奇数的功能。

5. 编写一个名为 MakeTshirt()的函数，它接受一个尺码以及要印到 T 恤上的字样。这个函数应打印一个句子，概要地说明 T 恤的尺码和字样。请分别使用位置参数和关键字参数调用这个函数来制作一件 T 恤。