扫一扫

视频讲解

第5章 组合数据类型

扫一扫

知识图谱

Python 语言提供了常用的组合数据类型，包括列表、元组、字符串、字节序列、字典和集合等。使用这些组合类型可以实现复杂数据的处理。

5.1 Python 序列数据概述

5.1.1 数组

数组是一种数据结构，用于存储和处理大量的数据。将所有的数据存储在一个或多个数组中，然后通过索引下标访问并处理数组的元素，可以实现复杂数据处理的任务。

Python 语言没有提供直接创建数组的功能，但可以使用其内置的序列数据类型（例如列表）实现数组的功能。

5.1.2 序列数据类型

序列（sequence）数据类型是 Python 的基础数据结构，是一组有顺序的元素的集合。序列数据可以包含一个或多个元素（也称为对象，元素或对象还可以是其他序列数据），也可以是一个没有任何元素的空序列。

Python 内置的序列数据类型包括列表（list）、元组（tuple）、字符串（str）和字节序列（bytes 和 bytearray）。

列表也称为表，用于存储其值可变的表。例如：

```
>>> s2 = [1,2,3]
>>> s2[2] = 4
>>> s2                                    #输出:[1, 2, 4]
```

元组也称为定值表，用于存储值固定不变的值表。例如：

```
>>> s1 = (1,2,3)
>>> s1                                    #输出:(1, 2, 3)
>>> s1[2]                                 #输出:3
```

字符串是包括若干字符的序列数据，支持序列数据的基本操作。例如：

```
>>> s3 = "abc"
>>> s3 = "Hello, world!"
>>> s3[:5]                                #字符串前 5 个字符.输出:'Hello'
```

字节序列是包括若干字节的序列数据。Python 抓取网页时返回的页面通常为 utf-8 编

码的字节序列数据。字节序列和字符串直接可以相互转换。例如：

```
>>> s1 = b"abc"
>>> s1.decode("utf-8")                    #输出:'abc'
>>> s2 = "百度"
>>> s2.encode("utf-8")                    #输出:b'\xe7\x99\xbe\xe5\xba\xa6'
```

5.2 序列数据的基本操作

5.2.1 序列的长度、最大值、最小值、求和

通过内置函数 len()、max()、min()可以获取序列的长度、序列中元素的最大值、序列中元素的最小值。通过内置函数 sum()可以获取列表或元组中各元素之和；如果有非数值元素，则导致 TypeError；对于字符串(str)和字节数据(bytes)，也将导致 TypeError。

【例 5.1】 序列数据的求和示例。

```
>>> t1 = (1,2,3,4)
>>> sum(t1)                  #输出:10
>>> t2 = (1,'a',2)
>>> sum(t2)                  #TypeError: unsupported operand type(s) for + : 'int' and 'str'
>>> s = '1234'
>>> sum(s)                   #TypeError: unsupported operand type(s) for + : 'int' and 'str'
```

【例 5.2】 序列的长度、最大值、最小值操作示例。

```
>>> s = 'abcdefg'      >>> t = (10,2,3)      >>> lst = [1,2,9,5,4]      >>> b = b'ABCD'
>>> len(s)             >>> len(t)           >>> len(lst)              >>> len(b)
7                      3                      5                        4
>>> max(s)             >>> max(t)           >>> max(lst)              >>> max(b)
'g'                    10                     9                        68
>>> min(s)             >>> min(t)           >>> min(lst)              >>> min(b)
'a'                    2                      1                        65
>>> s2 = ''            >>> t2 = ()          >>> lst2 = [ ]            >>> b2 = b''
>>> len(s2)            >>> len(t2)          >>> len(lst2)             >>> len(b2)
0                      0                      0                        0
```

5.2.2 序列的索引访问操作

序列表示可以通过索引下标访问的可迭代对象。用户可以通过整数下标访问序列 s 的元素。

```
s[i]                                      #访问序列 s 在索引 i 处的元素
```

索引下标从 0 开始，第 1 个元素为 s[0]，第 2 个元素为 s[1]，以此类推，最后一个元素为 s[len(s) − 1]。索引下标也可以从最后一个元素开始，从−1 开始，即最后一个元素为 s[−1]，第 1 个元素为 s[−len(s)]。

如果索引下标越界，则导致 IndexError；如果索引下标不是整数，则导致 TypeError。例如：

```
>>> s = 'abc'
>>> s[0]                                  #输出:'a'
>>> s[-1]                                 #输出:'c'
>>> s[3]                                  #IndexError: string index out of range
>>> s['a']                                #TypeError: string indices must be integers
```

序列 s 的索引下标示意图如图 5-1 所示。

图 5-1　序列 s 的索引下标示意图

【例 5.3】　序列的索引访问示例。

```
>>> s = 'abcdef'
>>> s[0]
'a'
>>> s[2]
'c'
>>> s[-1]
'f'
>>> s[-3]
'd'
```

```
>>> t = ('a','e','i','o','u')
>>> t[0]
'a'
>>> t[1]
'e'
>>> t[-1]
'u'
>>> t[-5]
'a'
```

```
>>> lst = [1,2,3,4,5]
>>> lst[0]
1
>>> lst
[1, 2, 3, 4, 5]
>>> lst[2] = 'a'
>>> lst[-2] = 'b'
>>> lst
[1, 2, 'a', 'b', 5]
```

```
>>> b = b'ABCDEF'
>>> b[0]
65
>>> b[1]
66
>>> b[-1]
70
>>> b[-2]
69
```

5.2.3　序列的切片操作

通过切片(slice)操作可以截取序列 s 的一部分。切片操作的基本形式如下。

```
s[i:j]  或者  s[i:j:k]
```

其中,i 为序列开始下标(包含 s[i]);j 为序列结束下标(不包含 s[j]);k 为步长。如果省略 i,则从下标 0 开始;如果省略 j,则直到序列结束为止;如果省略 k,则步长为 1。

注意:下标也可以为负数。如果截取范围内没有数据,则返回空元组;如果超过下标范围,不报错。

【例 5.4】　序列的切片操作示例。

```
>>> s = 'abcdef'
>>> s[1:3]
'bc'
>>> s[3:10]
'def'
>>> s[8:2]
''
>>> s[:]
'abcdef'
>>> s[:2]
'ab'
>>> s[::2]
'ace'
>>> s[::-1]
'fedcba'
```

```
>>> t = ('a','e','i','o','u')
>>> t[-2:-1]
('o',)
>>> t[-2:]
('o', 'u')
>>> t[-99:-5]
()
>>> t[-99:-3]
('a', 'e')
>>> t[::]
('a', 'e', 'i', 'o', 'u')
>>> t[1:-1]
('e', 'i', 'o')
>>> t[1::2]
('e', 'o')
```

```
>>> lst = [1,2,3,4,5]
>>> lst[:2]
[1, 2]
>>> lst[:1] = [ ]
>>> lst
[2, 3, 4, 5]
>>> lst[:2]
[2, 3]
>>> lst[:2] = 'a'
>>> lst[1:] = 'b'
>>> lst
['a', 'b']
>>> del lst[:1]
>>> lst
['b']
```

```
>>> b = b'ABCDEF'
>>> b[2:2]
b''
>>> b[0:1]
b'A'
>>> b[1:2]
b'B'
>>> b[2:2]
b''
>>> b[-1:]
b'F'
>>> b[-2:-1]
b'E'
>>> b[0:len(b)]
b'ABCDEF'
```

5.2.4　序列的连接和重复操作

通过连接操作符＋可以连接两个序列(s1 和 s2),形成一个新的序列对象;通过重复操

作符 * ,可以重复一个序列 n 次(n 为正整数)。序列连接和重复操作的基本形式如下。

```
s1 + s2  或者  s * n  或者  n * s
```

连接操作符+和重复操作符 * 也支持复合赋值运算,即+=和 * =。

【例 5.5】 序列的连接和重复操作示例。

```
>>> s1 = 'abc'          >>> t1 = (1,2)          >>> lst1 = [1,2]          >>> b1 = b'ABC'
>>> s2 = 'xyz'          >>> t2 = ('a','b')      >>> lst2 = ['a','b']      >>> b2 = b'XYZ'
>>> s1 + s2             >>> t1 + t2             >>> lst1 + lst2          >>> b1 + b2
'abcxyz'                (1, 2, 'a', 'b')       [1, 2, 'a', 'b']        b'ABCXYZ'
>>> s1 * 3              >>> t1 * 2              >>> 2 * lst2             >>> b1 * 3
'abcabcabc'             (1, 2, 1, 2)           ['a', 'b', 'a', 'b']    b'ABCABCABC'
>>> s1 += s2            >>> t1 += t2            >>> lst1 += lst2         >>> b1 += b2
>>> s1                  >>> t1                  >>> lst1                 >>> b1
'abcxyz'                (1, 2, 'a', 'b')       [1, 2, 'a', 'b']        b'ABCXYZ'
>>> s2 * = 2            >>> t2 * = 2            >>> lst2 * = 2           >>> b2 * = 2
>>> s2                  >>> t2                  >>> lst2                 >>> b2
'xyzxyz'                ('a', 'b', 'a', 'b')   ['a', 'b', 'a', 'b']    b'XYZXYZ'
```

5.2.5 序列的成员关系操作

用户可以通过下列方式之一判断一个元素 x 是否存在于序列 s 中。

- x in s:如果为 True,则表示存在。
- x not in s:如果为 True,则表示不存在。
- s.count(x[, start[, end]]):返回 x 在 s(指定范围[start,end))中出现的次数。
- s.index(x[, start[, end]]):返回 x 在 s(指定范围[start,end))中第一次出现的下标。

指定范围[start, end)从下标 start(包括,默认为 0)开始,到下标 end 结束(不包括,默认为 len(s))。

对于 s.index(value, [start, [stop]])方法,如果找不到时,则导致 ValueError。例如:

```
>>>'To be or not to be,this is a question'.index('123') #ValueError: substring not found
```

【例 5.6】 序列中元素的存在性判断示例。

```
>>> s = 'Good, better,    >>> t = ('r', 'g', 'b')   >>> lst = [1,2,3,2,1]    >>> b = b'Oh, Jesus!'
best!'                    >>> 'r' in t              >>> 1 in lst             >>> b'O' in b
>>> 'o' in s              True                      True                    True
True                      >>> 'y' not in t          >>> 2 not in lst         >>> b'o' not in b
>>> 'g' not in s          True                      False                   True
True                      >>> t.count('r')          >>> lst.count(1)        >>> b.count(b's')
>>> s.count('e')          1                         2                       2
3                         >>> t.index('g')          >>> lst.index(3)        >>> b.index(b's')
>>> s.index('e', 10)      1                         2                       6
10
```

5.2.6 序列的比较运算操作

两个序列支持比较运算符(<、<=、==、!=、>=、>),字符串比较运算按顺序逐个元素进行比较。

【例 5.7】 序列的比较运算示例。

```
>>> s1 = 'abc'          >>> t1 = (1,2)          >>> s1 = ['a','b']        >>> b1 = b'abc'
>>> s2 = 'abc'          >>> t2 = (1,2)          >>> s2 = ['a','b']        >>> b2 = b'abc'
>>> s3 = 'abcd'         >>> t3 = (1,2,3)        >>> s3 = ['a','b','c']   >>> b3 = b'abcd'
>>> s4 = 'cba'          >>> t4 = (2,1)          >>> s4 = ['c','b','a']   >>> b4 = b'ABCD'
>>> s1 > s4             >>> t1 < t4            >>> s1 < s2              >>> b1 < b2
False                   True                   False                    False
>>> s2 <= s3            >>> t1 <= t2           >>> s1 <= s2             >>> b1 <= b2
True                    True                   True                     True
>>> s1 == s2            >>> t1 == t3           >>> s1 == s2             >>> b1 == b2
True                    False                  True                     True
>>> s1 != s3            >>> t1 != t2           >>> s1!= s3              >>> b1 >= b3
True                    False                  True                     False
>>> 'a' > 'A'           >>> t1 >= t3           >>> s1 >= s3             >>> b3!= b4
True                    False                  False                    True
>>> 'a' >= ''           >>> t4 > t3            >>> s4 > s3              >>> b4 > b3
True                    True                   True                     False
```

5.2.7 序列的排序操作

通过内置函数 sorted()可以返回序列的排序列表。通过类 reversed 的构造函数，可以返回序列的反序迭代器。内置函数 sorted()形式如下。

```
sorted(iterable, key = None, reverse = False)          # 返回序列的排序列表
```

其中，key 是用于计算比较键值的函数(带一个参数)，例如，key＝str. lower。如果 reverse＝True，则反向排序。

【例 5.8】 序列的排序操作示例。

```
>>> s1 = 'axd'                >>> sorted(s2)                  >>> s3 = 'abAC'
>>> sorted(s1)                [1, 2, 4]                     >>> sorted(s3, key = str.lower)
['a', 'd', 'x']               >>> sorted(s2,reverse = True)   ['a', 'A', 'b', 'C']
>>> list(reversed(s1))        [4, 2, 1]                     >>> list(reversed(s3))
['d', 'x', 'a']               >>> list(reversed(s2))          ['C', 'A', 'b', 'a']
>>> s2 = (1,4,2)              [2, 4, 1]
```

5.2.8 内置函数 all()和 any()

通过内置函数 all()和 any()可以判断序列的元素是否全部和部分为 True。函数形式如下。

- `all(iterable)`　　# 如果序列的所有值都为 True，返回 True；否则，返回 False
- `any(iterable)`　　# 如果序列的任意值为 True，返回 True；否则，返回 False

例如：

```
>>> any((1, 2, 0))    >>> all([1, 2, 0])
True                   False
```

5.2.9 序列的拆分操作

1. 变量个数和序列长度相等

使用赋值语句可以将序列值进行拆分(也称为解包)，然后赋值给多个变量，形式如下。

```
变量 1, 变量 2, …, 变量 n = 序列或可迭代对象
```

如果变量个数和序列的元素个数不一致时，将导致 ValueError。例如：

```
>>> a, b = (1, 2)
>>> a, b                          #输出:(1, 2)
>>> a, b, c = (1, 2)              #ValueError: not enough values to unpack (expected 3, got 2)
>>> data = (1001, '张三', (80, 79, 92))
>>> sid, name, scores = data
>>> scores                        #输出:(80, 79, 92)
>>> sid, name, (chinese, math, english) = data
>>> math                          #输出:79
```

2. 变量个数和序列长度不等

如果序列长度未知，可以使用“*元组变量”的形式，将多个值作为元组赋值给元组变量。一个赋值语句中，“*元组变量”只允许出现一次，否则导致 SyntaxError。例如：

```
>>> first, *middles, last = range(10)
>>> middles                              #输出:[1, 2, 3, 4, 5, 6, 7, 8]
>>> first, second, third, *lasts = range(10)
>>> lasts                                #输出:[3, 4, 5, 6, 7, 8, 9]
>>> *firsts, last3, last2, last1 = range(10)
>>> firsts                               #输出:[0, 1, 2, 3, 4, 5, 6]
>>> first, *middles, last = sorted([70, 85, 89, 88, 86, 95, 89])     #去掉最高分和最低分
>>> sum(middles)/len(middles)            #计算去掉最高分和最低分后的平均值。输出:87.4
```

3. 使用临时变量“_”

如果只需要部分数据，序列的其他位置可以使用临时变量“_”。例如：

```
>>> _, b, _ = (1, 2, 3)
>>> b                                    #输出:2
>>> record = ('Zhangsan', 'szhang@abc.com', '021-62232333', '13912349876')
>>> name, _, *phones = record
>>> phones                               #输出:['021-62232333', '13912349876']
```

5.3 列表

列表(list)是一组有序项目的数据结构。在创建一个列表后，用户可以访问、修改、添加或删除列表中的项目，即列表是可变的数据类型。在 Python 中没有数组，可以使用列表代替数组。

5.3.1 使用列表字面量创建列表实例对象

使用列表字面量可以创建列表实例对象。列表字面量列表采用方括号中用逗号分隔的项目定义。其基本形式如下。

```
[x1 [, x2, …, xn]]
```

【例 5.9】 使用列表字面量创建列表实例对象示例。

```
>>> a1 = [ ]
>>> a2 = [1]
>>> a3 = ["a","b","c"]
>>> print(a1, a2, a3)                    #输出:[ ] [1] ['a', 'b', 'c']
```

5.3.2 使用 list 对象创建列表实例对象

用户也可以使用 list 对象来创建列表实例对象。其基本形式如下。

- list()　　#创建一个空列表
- list(iterable)　　#创建一个列表，包含的项目为可枚举对象 iterable 中的元素

【例 5.10】 使用 list 对象创建列表实例对象示例。

```
>>> a1 = list()
>>> a2 = list("abc")
>>> a3 = list(range(3))
>>> print(a1, a2, a3)      #输出:[ ] ['a', 'b', 'c'] [0, 1, 2]
```

5.3.3 列表解析表达式

使用列表解析可以简单、高效地处理一个可迭代对象，并生成结果列表。列表解析表达式的形式如下。

- [expr for i_1 in 序列 1... for i_N in 序列 N]　　#迭代序列里所有内容，并计算生成列表
- [expr for i_1 in 序列 1... for i_N in 序列 N if cond_expr]　　#按条件迭代，并计算生成列表

表达式 expr 使用每次迭代的内容 $i_1 \cdots i_N$ 计算生成一个列表。如果指定了条件表达式 cond_expr，则只有满足条件的元素参与迭代。

【例 5.11】 列表解析表达式示例。

```
>>> [i ** 2 for i in range(10)]                                    #平方值
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> [(i,i ** 2) for i in range(10)]                                #序号, 平方值
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64), (9, 81)]
>>> [i for i in range(10) if i % 2 == 0]                           #取偶数
[0, 2, 4, 6, 8]
>>> [(x, y, x * y) for x in range(1, 4) for y in range(1, 4) if x >= y]  #二重循环
[(1, 1, 1), (2, 1, 2), (2, 2, 4), (3, 1, 3), (3, 2, 6), (3, 3, 9)]
```

5.3.4 列表的序列操作

列表支持序列的基本操作，包括索引访问、切片操作、连接操作、重复操作、成员关系操作、比较运算操作，以及求列表长度、最大值、最小值等内置函数。

列表是可变对象，故可以改变列表对象中元素的值，也可以通过 del 删除某元素。

```
a[下标] = x                    #设置列表元素,x 为任意对象
del a[下标]                    #删除列表元素
```

列表是可变对象，故用户可以改变其切片的值，也可以通过 del 删除切片。

```
a[i:j] = x                     #设置列表内容,x 为任意对象,也可以是元组、列表
del a[i:j]                     #删除列表的一系列元素,等同于 a[i:j] = [ ]
a[i:j] = [ ]                   #删除列表的一系列元素
```

【例 5.12】 列表的序列操作示例。

```
>>> a = [1,2,3,4,5,6]
>>> a[1] = 'a'
>>> a
[1, 'a', 3, 4, 5, 6]
>>> a[2] = [ ]
>>> a
[1, 'a', [ ], 4, 5, 6]
>>> del a[3]
>>> a
[1, 'a', [ ], 5, 6]
>>> a[:2]
[1, 'a']
>>> a[2:3] = [ ]
>>> a
[1, 'a', 5, 6]
>>> a[:1] = [ ]
>>> a
['a', 5, 6]
>>> a[:2] = 'b'
>>> a
['b', 6]
>>> del a[:1]
>>> a
[6]
```

5.3.5 列表对象的方法

列表是可变对象，其包含的主要方法如表 5-1 所示。假设表中的示例基于 a=[1,3,2]。

表 5-1 列表对象的主要方法

方 法	说 明	示 例	说 明
a.append(x)	把对象 x 追加到列表 a 尾部	a.append('a') a.append([1,2])	a=[1, 3, 2, 'a'] a=[1, 3, 2, 'a', [1, 2]]
a.clear()	删除所有元素。相当于 del s[:]	a.clear()	a=[]
a.copy()	拷贝列表	a1 = a.copy() id(a),id(a1)	a1=a=[1, 3, 2] a 和 a1 的 ID 不同
a.count(x)	返回对象 x 在列表中出现的次数	a.count(1) a.count(9)	1 0
a.extend(t)	把序列 t 附加到列表 a 的尾部	a.extend([4]) a.extend('ab')	a=[1, 3, 2, 4] a=[1, 3, 2, 4, 'a', 'b']
a.index(x)	返回对象 x 在列表 a 中的最低索引。如果该对象在列表中不存在，则报错	a.index(3) a.index(9)	1 ValueError: 9 is not in list
a.insert(i, x)	在下标 i 位置插入对象 x	a.insert(1,4) a.insert(8,5)	a=[1, 4, 3, 2] a=[1, 4, 3, 2, 5]
a.pop([i])	返回并移除下标 i 位置对象，省略 i 时为最后对象。若超出下标，将导致 IndexError	a.pop() a.pop(0)	2。a=[1, 3] 1。a=[3]
a.remove(x)	移除列表中第一个出现的 x。若对象不存在，将导致 ValueError	a.remove(1) a.remove('a')	a=[3, 2] ValueError: x not in list
a.reverse()	列表反转	a.reverse()	a=[2, 3, 1]
a.sort()	列表排序	a.sort()	a=[1, 2, 3]

5.3.6 列表的典型示例代码

列表的典型示例如表 5-2 所示。假设表中的示例基于 a1=[1,2,3]、a2=[5, 6]、a3=[0, 1, 2, 3, 4, 2]、a4=[1, 8, 5]以及 a5=['W', 'e', 'l', 'c', 'o', 'm', 'e']。

表 5-2 列表的典型示例

功 能 示 例	实 现 代 码	结 果 说 明
使用字面量创建列表	a1 =[1,2,3]	a1=[1, 2, 3]
使用 list 对象创建列表	aList1 = list('abc') aList2 = tuple(range(3))	aList1=['a', 'b', 'c'] aList2=[0, 1, 2]
使用列表解析表达式创建列表	aa = [i for i in range(6) if i%2 != 0]	aa=[1, 3, 5]
拷贝列表(copy()方法)	bb = a1.copy()	bb=a1=[1,2,3]
使用索引访问列表中的元素	a1[1] a1[-1]	2 3
使用 for 循环访问列表中的元素	for i in a1: print(i, end=' ')	1 2 3

续表

功能示例	实现代码	结果说明
列表的切片操作	a1[1：3] a1[－2：]	[2，3] [2，3]
列表的连接操作(＋运算符)	a1 ＋ a2	[1,2,3,5,6]
列表的重复操作(＊运算符)	a2 ＊ 2	[5，6，5，6]
列表的成员关系操作(in、not in运算符)	2 in a1 3 not in a1	True False
列表的比较运算操作(＝＝、!＝、>、>＝、<、<＝)	a1 > a2	False
返回列表中元素的个数(len()函数)	len(a1)	3
返回元素在列表中出现的次数	a3.count(2)	2
返回元素在列表中的最低索引	a1.index(3)	2
判断列表的元素是否全部或部分为True(all()和 any()函数)	all(a3) any(a3)	False True
将元素添加到列表末尾(append()方法)	a1.append(10)	a1＝[1，2，3，4，10]
将一个列表附加到另一个列表末尾(extend()方法)	a1.extend(a2)	a1＝ [1，2，3，5，6]
在列表给定索引处插入元素(insert()方法)	a1.insert(0，10)	a1＝[10，1，2，3]
删除列表中的指定值(remove()方法)	a3.remove(2)	a3＝[0，1，3，4，2]
弹出列表元素(pop()方法)	a1.pop(1)	输出2。a1＝[1，3]
删除列表给定索引处的元素(del语句)	del(a1[1])	a1＝[1，3]
删除列表切片的元素(del语句)	del(a1[：2])	a1＝[3]
列表的反转	a1.reverse()	a1＝[3，2，1]
列表中元素的最大值和最小值(max()和min()函数)	max(a1) min(a1)	3 1
列表中所有元素的总和	sum(a1)	6
列表的排序(sort()方法，直接改变原列表的元素顺序)	a4.sort()	a4＝ [1，5，8]
列表的排序(sorted()函数，按升序或降序返回新的列表)	sorted(a4) sorted(a4，reverse ＝ True)	输出[1，5，8]。a4＝[1，5，8] 输出[8，5，1]。a4＝[1，5，8]
连接列表的元素以创建字符串	char ＝ '' char.join(a5)	'Welcome'
清空列表内容	a1.clear()	a1＝[]

5.4 元组

元组(tuple)是一组有序序列，包含零个或多个对象引用。元组和列表十分类似，但元组是不可变的对象，即不能修改、添加或删除元组中的项目(但可以访问元组中的项目)。

5.4.1 使用元组字面量创建元组实例对象

使用元组字面量可以创建元组实例对象。元组字面量采用圆括号中用逗号分隔的项目定义。圆括号可以省略。元组的基本形式如下。

```
x1, [x2, …, xn] 或者 (x1, [x2, …, xn])
```

其中，x1，x2，…，xn 为任意对象。

【例 5.13】 使用元组字面量创建元组实例对象示例。

```
>>> t1 = 1,2,3
>>> t2 = ()
>>> t3 = 1
>>> t4 = (1)
>>> t5 = 'a','b','c'
>>> t6 = 2.0,
>>> print(t1)
(1, 2, 3)
>>> print(t2)
()
>>> print(t3)
1
>>> print(t4)
1
>>> print(t5)
('a', 'b', 'c')
>>> print(t6)
(2.0,)
```

注意：如果元组中只有一个项目时，后面的逗号不能省略。这是因为 Python 解释器把(x1)解释为 x1，例如(1)解释为整数 1，(1,)解释为元组。

5.4.2 使用 tuple 对象创建元组实例对象

用户也可以使用 tuple 对象来创建元组，其基本形式如下。

- `tuple()` ＃创建一个空元组
- `tuple(iterable)` ＃创建一个元组，包含的项目为可枚举对象 iterable 中的元素

【例 5.14】 使用 tuple 对象创建元组实例对象示例。

```
>>> t1 = tuple()
>>> t2 = tuple("abc")
>>> t3 = tuple([1,2,3])
>>> t4 = tuple(range(3))
>>> print(t1,t2,t3,t4)          ＃输出:() ('a', 'b', 'c') (1, 2, 3) (0, 1, 2)
```

5.4.3 元组的序列操作

元组支持序列的基本操作，包括索引访问、切片操作、连接操作、重复操作、成员关系操作、比较运算操作，以及求元组长度、最大值、最小值等内置函数。

【例 5.15】 元组的序列操作示例。

```
>>> t1 = (1,2,3,4,5,6,7,8,9,10)
>>> len(t1)                     ＃输出:10
>>> max(t1)                     ＃输出:10
>>> sum(t1)                     ＃输出:55
```

5.4.4 元组对象的方法

元组是不可变对象，其包含的主要方法如表 5-3 所示。假设该表中的示例基于 t＝(1，2，2，3，3，3)。

表 5-3 tuple 对象的主要方法

方 法	说 明	示 例	结 果
t.count(x)	计算指定元素在元组中的重复次数	t.count(2) t.count(4)	2 0
t.index(x)	在元组中查找元素的最低索引值。如果不存在，则会产生错误 ValueError	t.index(3) t.index(4)	3 报错

5.4.5 元组的典型示例代码

元组的典型示例如表 5-4 所示。假设该表中的示例基于 t1=(1,2,3)、t2=(5,6)、t3=(0,1,2,3,4,2)、t4=(1,8,5)以及 t5=('W', 'e', 'l', 'c', 'o', 'm', 'e')。

表 5-4 元组的典型示例

功能示例	实现代码	结果说明
使用字面量创建元组	tp1=1,2,3 tp2=(1,2,3) tp3=(1,)	tp1=(1, 2, 3) tp2=(1, 2, 3) tp3=(1,)
使用 tuple 对象创建元组	tp1=tuple('abc') tp2=tuple([1, 2, 3]) tp3=tuple(range(3))	tp1=('a', 'b', 'c') tp2=(1, 2, 3) tp3=(0, 1, 2)
使用索引访问元组的元素	t1[1] t1[-1]	2 4
使用 for 循环访问元组的元素	for i in t2: print(i, end=' ')	5 6
元组的切片操作	t1[1: 3] t1[-2:]	(2, 3) (3, 4)
元组的连接操作(+运算符)	t1 + t2	(1,2,3,5,6)
元组的重复操作(*运算符)	t2 * 2	(5,6,5,6)
元组的成员关系操作(in、not in 运算符)	2 in t1 3 not int1	True False
元组的比较运算操作(==、!=、>、>=、<、<=)	t1 > t2	False
返回元组中元素的个数(len()函数)	len(t1)	3
返回指定元素在元组中出现的次数(count()方法)	t3.count(2)	2
返回指定元素在字符串中的最低索引(index()方法)	t1.index(3)	2
判断元组的元素是否全部或部分为 True (all()和 any()函数)	all(t3) any(t3)	False True
元组中元素的最大值和最小值(max()和 min()函数)	max(t1) min(t1)	3 1
元组中所有元素的总和	sum(t1)	6
元组的排序(sorted()函数,按升序或降序返回新的列表)	sorted(t4) sorted(t4, reverse = True)	输出(1, 5, 8)。t4=(1, 8, 5) 输出(8, 5, 1)。t4=(1, 8, 5)
连接元组的元素以创建字符串	char = '' char.join(t5)	'Welcome'

5.5 字符串

字符串(str)实现为有序的字符集合,即字符序列。在 Python 中没有独立的字符数据类型,字符即长度为 1 的字符串。

Python 内置数据类型 str 用于字符串处理。字符串对象的值为字符序列。字符串对象(字符串)是不可变对象。

更复杂的字符串和文本处理请参见第 15 章。

5.5.1 使用字符串字面量创建字符串对象

使用单引号或双引号括起来的内容是字符串字面量，Python 解释器自动创建 str 型对象实例。Python 字符串字面量可以使用以下四种方式定义。

(1) 单引号(' ')。包含在单引号中的字符串，其中可以包含双引号。

(2) 双引号(" ")。包含在双引号中的字符串，其中可以包含单引号。

(3) 三单引号(''' ''')。包含在三单引号中的字符串，可以跨行。

(4) 三双引号(""" """)。包含在三双引号中的字符串，可以跨行。

【例 5.16】 字符串字面量示例。

```
>>> 'Hello'                          #输出:'Hello'
>>> "不积跬步,无以至千里"             #输出:'不积跬步,无以至千里'
>>> type("Python")                   #输出:<class 'str'>
```

注意：两个紧邻的字符串，如果中间只有空格分隔，则自动拼接为一个字符串。例如：

```
>>> '不积小流' '无以成江河'          #输出:'不积小流无以成江河'
```

5.5.2 字符串编码

Python 3 中的字符默认为 16 位 Unicode 编码，ASCII 码是 Unicode 编码的子集。例如，字符'A'的 ASCII 码为 65，对应的八进制为 101，对应的十六进制为 41。

使用 u''或 U''的字符串称为 Unicode 字符串。在 Python 3 中默认为 Unicode 字符串。

```
>>> u'abc'                           #输出:'abc'
```

使用内置函数 ord()可以把字符转换为对应的 Unicode 码；使用内置函数 chr()可以把十进制数转换为对应的字符。例如：

```
>>> ord('A')                         #输出:65
>>> chr(66)                          #输出:'B'
>>> ord('张')                        #输出:24352
>>> chr(24352)                       #输出:'张'
```

5.5.3 转义字符

特殊符号(不可打印字符)可以使用转义序列表示。转义序列以反斜杠开始，紧跟一个字母，例如“\n”(新行)和“\t”(制表符)。如果希望字符串中包含反斜杠，则它前面必须还有另一个反斜杠。Python 转义字符如表 5-5 所示。

表 5-5 特殊符号的转义序列

转义序列	字符	转义序列	字符
\'	单引号	\n	换行(LF)
\"	双引号	\r	回车(CR)
\\	反斜杠	\t	水平制表符(HT)
\a	响铃(BEL)	\v	垂直制表符(VT)
\b	退格(BS)	\ooo	八进制 Unicode 码对应的字符
\f	换页(FF)	\xhh	十六进制 Unicode 码对应的字符

【例 5.17】 转义字符串示例。

```
>>> s = '学号\t 姓名\t 性别\\t 地址'
>>> s                                  #输出:'学号\t 姓名\t 性别\\t 地址'
>>> print(s)                           #输出:学号      姓名      性别\t 地址
```

使用转义字符后跟 Unicode 编码也可以表示字符。例如：

```
>>> '\101'                             #输出:'A'
>>> '\x41'                             #输出:'A'
```

使用 r''或 R''的字符串称为原始字符串，其中包含的任何字符都不进行转义。

```
>>> s = r'换\t 行\t 符\n'
>>> s                                  #输出:'换\\t 行\\t 符\\n'
>>> print(s)                           #输出:换\t 行\t 符\n
```

5.5.4 使用内置函数 str()创建字符串对象

str 是 Python 的内置数据类型，创建字符串类型对象实例的基本形式如下。

```
str(object = '')                                    #创建字符串对象,默认为空字符串
```

通过创建 str 对象可以把任意对象转换为字符串对象，返回 object.__str__()，如果对象没有定义__str__()，则返回 repr(object)。

实际上，在使用 print(123)输出数值时将自动调用 str(123)，把 123 转换为字符串，然后输出。

Python 还提供了另一个内置函数 repr()，该函数返回一个对象的更精确的字符串表示形式。在 Python 解释器中，直接输出的对象 obj 就是 repr(obj)。

在大多数情况下，内置函数 repr()和 str()的结果一致。

【例 5.18】 使用 str()创建字符串对象的应用示例(strToint.py)：将给定整数的各个位数上的数字累加。

```
intN = 123456                       #给定一个整数
total = 0                           #累加和 total 赋初值
for s in str(intN):                 #将整数转换为字符串,利用 for 语句迭代字符串序列
    total += int(s)                 #将字符转换为整数,实现各个位数上的数字累加
print(total)                        #输出给定整数的各个位数上的数字累加和
```

程序运行结果如下所示。

```
21
```

5.5.5 字符串的序列操作

字符串支持序列的基本操作，包括索引访问、切片操作、连接操作、重复操作、成员关系操作、比较运算操作，以及求字符串长度、最大值、最小值等内置函数。

通过 len(s)函数可以获取字符串 s 的长度，如果其长度为 0，则为空字符串。

【例 5.19】 字符串的序列操作示例。

```
>>> s1 = 'abcxyz'          >>> s1[3:]          >>> s1 > s2          'TigerTigerTiger'
>>> len(s1)                'xyz'             True               >>> max(s1)
6                          >>> s2 = 'Tiger'     >>> 3 * s2          'z'
```

5.5.6 字符串对象的方法

使用字符串对象提供的方法可以实现常用的字符串处理功能。字符串对象是不可变对象，故调用方法返回的字符串是新创建的对象。除了直接调用字符串对象的方法外，还可以使用 str 类方法来处理字符串。

【例 5.20】 字符串对象方法的调用示例。

```
>>> s = 'abc'
>>> s.upper()                    # 字符串对象 s 的方法.输出:'ABC'
>>> str.upper(s)                 # str 类方法,字符串 s 作为参数.输出:'ABC'
```

字符串对象所包含的主要方法如表 5-6 所示。假设表 5-6(1)和表 5-6(2)中的示例基于以下的字符串实例对象。

```
>>> s1 = 'yellow ribbon'
>>> s2 = 'Pascal Case'
>>> s3 = '123'
>>> s4 = 'iPhone13'
```

表 5-6(1) 字符串对象的主要方法(字符串类型判断)

方 法	说 明	示 例	结 果
s.isdigit()	是否全为数字(0～9)	s1.isdigit()	False
s.isalpha()	是否全为字母	s2.isupper()	False
s.isdecimal()	是否只包含十进制数字字符	s3.isdecimal()	True
s.isalnum()	是否全为字母或者数字	s4.isalnum()	True
s.islower()	是否全为小写	s1.islower()	True
s.isupper()	是否全为大写	s4.isupper()	False
s.isspace()	是否只包含空白字	s2.isspace()	False
s.istitle()	是否为标题,即各单词首字母大写	s2.istitle()	True
s.isnumeric()	是否只包含数字字符	s3.isnumeric()	True
s.isidentifier()	是否是合法标识	s3.isidentifier()	False
s.isprintable()	是否只包含可打印字	s4.isprintable()	True

表 5-6(2) 字符串对象的主要方法(字符串大小写转换)

方 法	说 明	示 例	结 果
s.capitalize()	转换为首字母大写,其余小写	s1.capitalize()	'Yellow ribbon'
s.lower()	转换为小写	s2.lower()	'pascal case'
s.upper()	转换为大写	s2.upper()	'PASCAL CASE'
s.swapcase()	大小写互换	s2.swapcase()	'pASCAL cASE'
s.title()	转换为各单词首字母大写	s1.title()	'Yellow Ribbon'
s.casefold()	转换为大小写无关的格式化字符串	s2.casefold()	'pascal case'

假设表 5-6(3)中的示例基于以下的字符串实例对象(其中，符号“␣”表示空格)。

```
>>> s1 = '␣␣123␣'
>>> s2 = '123'
>>> s3 = 'a\tb'
```

表 5-6(3)　字符串对象的主要方法(字符串的填充、空白和对齐)

方　法	说　明	示　例	结　果
s.strip([chars])	去两边空格,也可指定要去除的字符列表	s1.strip()	'123'
s.lstrip([chars])	去左边空格,也可指定要去除的字符列表	s1.lstrip()	'123␣'
s.rstrip([chars])	去右边空格,也可指定要去除的字符列表	s1.rstrip()	'␣␣123'
s.zfill(width)	左填充,使用 0 填充到 width 长度	s2.zfill(5)	'00123'
s.center(width[, fillchar])	两边填充,使用填充字符 fillchar(默认空格)填充到 width 长度	s2.center(7,'#')	'##123##'
s.ljust(width[, fillchar])	字符串左对齐调整,右侧使用填充字符 fillchar(默认空格)填充到 width 长度	s2.ljust(7,'#')	'123####'
s.rjust(width[, fillchar])	字符串右对齐调整,左侧使用填充字符 fillchar(默认空格)填充到 width 长度	s2.rjust(7)	'␣␣␣␣123'
s.expandtabs([tabsize])	跟踪当前光标位置,并将其找到的每个制表符字符替换为当前光标位置到下一个制表位的空格数。tabsize 默认为 8	s3.expandtabs(4)	'a␣␣␣b'

假设表 5-6(4)中的示例基于以下的字符串实例对象。

```
>>> s1 = "abABabCDABCD"
```

表 5-6(4)　字符串对象的主要方法(字符串测试、查找和替换)

方　法	说　明	示　例	结　果
str.startswith(prefix[, start[, end]])	是否以 prefix 开头	s1.startswith("AB") s1.startswith("AB",2)	False True
str.endswith(suffix[, start[, end]])	是否以 suffix 结尾	s1.endswith("CD")	True
str.count(sub[, start[, end]])	返回指定字符串出现的次数	s1.count("ab")	2
str.index(sub[, start[, end]])	搜索指定字符串,返回下标,没有则导致 ValueError	s1.index("AB")	2
str.rindex(sub[, start[, end]])	从右边开始搜索指定字符串,返回下标,没有则导致 ValueError	s1.rindex("AB")	8
str.find(sub[, start[, end]])	搜索指定字符串,返回下标,没有则返回－1	s1.find("CD")	6
str.rfind(sub[, start[, end]])	从右边开始搜索指定字符串,返回下标,没有则返回－1	s1.rfind("CD")	10
str.replace(old, new[, count])	替换 old 为 new,可选 count 为替换次数	s1.replace("ab","x")	'xABxCDABCD'

说明：可选指定范围[start, end)为从下标 start(包括 start,默认为 0)开始到下标 end 结束(不包括 end,默认为 len(s))。

假设表 5-6(5)中的示例基于以下的字符串实例对象。

```
>>> s1 = 'one,two,three'
>>> s2 = 'abc\n123\nxyz'
```

表 5-6(5) 字符串对象的主要方法(字符串拆分和组合)

方 法	说 明	示 例	结 果
s.split(sep=None, maxsplit=-1)	按指定字符(默认为空格)分割字符串,返回列表。maxsplit 为最大分割次数,默认-1,无限制	s1.split(',')	['one', 'two', 'three']
s.rsplit(sep=None, maxsplit=-1)	从右侧按指定字符分割字符串,返回列表	s1.rsplit(',', 1)	['one,two', 'three']
s.partition(sep)	根据分隔符 sep 分割字符串为两部分,返回元组(left, sep, right)	s1.partition(',')	('one', ',', 'two,three')
s.rpartition(sep)	根据分隔符 sep 从右侧分割字符串为两部分,返回元(left, sep, right)	s1.rpartition(',')	('one,two', ',', 'three')
s.splitlines([keepends])	按行分割字符串,返回列表	s2.splitlines() s2.splitlines(True)	['abc', '123', 'xyz'] ['abc\n', '123\n', 'xyz']
s.join(iterable)	组合 iterable 中的各元素成字符串,若包含非字符串元素,则导致 TypeError	s3=['a', 'b', 'c'] s4=': ' s4.join(s3)	'a: b: c'

假设表 5-6(6)中的示例基于以下的字符串实例对象。

```
>>> s1 = 'one,two,three'
>>> s2 = 'abc\n123\nxyz'
```

表 5-6(6) 字符串对象的主要方法(字符串翻译和转换)

方 法	说 明	示 例

str.maketrans(x[, y[, z]])	静态方法。创建用于 translate 方法的转换表	>>> table1=str.maketrans('1234567','一二三四五六日')
		>>> s1='1 3 4 9'
		>>> s1.translate(table1)
		'一 三 四 9'
		>>> weeks={'1': 'M 一','2': 'T 二','3': 'W 三','4': 'T 四','5': 'F 五','6': 'S 六','7': 'S 日'}
		>>> table2=str.maketrans(weeks)
		>>> s1.translate(table2)
		'M 一 W 三 T 四 9'

s.translate(map)	根据 map 转换	

5.5.7 字符串的格式化

通过字符串格式化可以输出特定格式的字符串。Python 中字符串格式化包括以下几种方式。

- 字符串.format(值 1, 值 2, …)
- str.format(格式字符串 1,值 1, 值 2, …)
- format(值, 格式字符串)
- 格式字符串 % (值 1, 值 2, …)　　#兼容 Python 2 的格式,不建议使用

例如：

```
>>> "学生人数{0},平均成绩{1}".format(15, 81.2)
'学生人数 15,平均成绩 81.2'
>>> str.format("学生人数{0},平均成绩{1:2.2f}", 15, 81.2)
'学生人数 15,平均成绩 81.20'
>>> format(81.2, "0.5f")                    #输出:'81.20000'
>>> "学生人数 %4d,平均成绩 %2.1f" % (15, 81)
'学生人数 15,平均成绩 81.0'
```

1. %运算符形式

Python 支持类似于 C 语言的 printf 格式化输出。采用如下形式。

```
格式字符串 % (值 1, 值 2, …)　　#兼容 Python 2 的格式,不建议使用
```

格式化字符串与 C 语言的 printf 格式化字符串基本相同。格式字符串由固定文本和格式说明符混合组成。格式说明符的语法如下。

```
%[(key)][flags][width][.precision][Length]type
```

其中,key(可选)为映射键(适用于映射的格式化,例如'%(lang)s');flags(可选)为修改输出格式的字符集;width(可选)为最小宽度,如果为 *,则使用下一个参数值;precision(可选)为精度,如果为 *,则使用下一个参数值;Length 为修饰符(h、l 或 L,可选),Python 忽略该字符;type 为格式化类型字符。例如：

```
>>> '结果:%f' % 88                         #输出:'结果:88.000000'
>>> '姓名:%s, 年龄:%d, 体重:%3.2f' % ('张三', 20, 53)
'姓名:张三, 年龄:20, 体重:53.00'
>>> '%(lang)s has %(num)03d quote types.' % {'lang':'Python', 'num': 4}
'Python has 004 quote types.'
>>> '%0*.*f' % (10, 5, 88)                  #输出:'0088.00000'
```

格式字符串的标志符(flags)如下。

- '0':数值类型格式化结果左边用 0 填充。
- '-':结果左对齐。
- ' ':对于正值,结果中将包括一个前导空格。
- '+':数值结果总是包括一个符号('+'或'-')。
- '#':使用另一种转换方式。

格式化类型字符(type)如下。

- %d 或%i:有符号整数(十进制)。
- %o:有符号整数(八进制)。
- %u:同%d,已过时。
- %x:有符号整数(十六进制,小写字符),标志符为'#'时,输出前缀'0x'。
- %X:有符号整数(十六进制,大写字符),标志符为'#'时,输出前缀'0X'。
- %e:浮点数字(科学记数法,小写 e),标志符为'#'时,总是带小数点。
- %E:浮点数字(科学记数法,大写 E),标志符为'#'时,总是带小数点。
- %f 或%F:浮点数字(用小数点符号),标志符为'#'时,总是带小数点。

- %g：浮点数字(根据值的大小采用%e或%f)，标志符为'#'时，总是带小数点，保留后面0。
- %G：浮点数字(根据值的大小采用%E或%F)，标志符为'#'时，总是带小数点，保留后面0。
- %c：字符及其ASCII码。
- %r：字符串，使用转换函数repr()，标志符为'#'且指定precision时，截取precision个字符。
- %s：字符串，使用转换函数str()，标志符为'#'且指定precision时，截取precision个字符。
- %a：字符串，使用转换函数ascii()，标志符为'#'且指定precision时，截取precision个字符。
- %%：百分号标记。

2. format内置函数

format内置函数的基本形式如下。

```
• format(value)                          #等同于str(value)
• format(value, format_spec)             #等同于type(value).__format__(format_spec)
```

格式化说明符(format_spec)的基本格式如下。

```
[[fill]align][sign][#][0][width][,][.precision][type]
```

其中，fill(可选)为填充字符，可以为除{}外的任何字符；align为对齐方式，包括"<"(左对齐)、">"(右对齐)、"="(填充位于符号和数字之间，例如'+000000120')、"^"(居中对齐)；sign(可选)为符号字符，包括"+"(正数)、"-"(负数)、" "(正数带空格，负数带-)；'#'(可选)使用另一种转换方式；'0'(可选)数值类型格式化结果左边用零填；width(可选)是最小宽度；precision(可选)是精度；type是格式化类型字符。

格式化类型字符(type)如下。

- b：二进制数。
- c：字符，整数转换为对应的unicode。
- d：十进制数。
- o：八进制数。
- x：十六进制数，小写字符，标志符为'#'时，输出前缀'0x'。
- X：十六进制数，大写字符，标志符为'#'时，输出前缀'0X'。
- e：浮点数字(科学记数法，小写e)，标志符为'#'时，总是带小数点。
- E：浮点数字(科学记数法，大写E)，标志符为'#'时，总是带小数点。
- f或F：浮点数字(用小数点符号)，标志符为'#'时，总是带小数点。
- g：浮点数字(根据值的大小采用e或f)，标志符为'#'时，总是带小数点，保留后面0。
- G：浮点数字(根据值的大小采用E或F)，标志符为'#'时，总是带小数点，保留后面0。
- n：数值，使用本地千位分隔符。
- s：字符串，使用转换函数str()，标志符为'#'且指定precision时，截取precision个字符。
- %：百分比。

• _：十进制千分位分隔符或者二进制 4 位分隔符(Python 3.6 及以上版本的新增功能)。例如：

```
>>> format(81.2, "0.5f")                    #输出:'81.20000'
>>> format(0.812, "%")                      #输出:'81.200000%'
>>> format(1000000, "_")                    #输出:1_000_000'
>>> format(1024, "_b")                      #输出:'100_0000_0000'
```

3. 字符串的 format 方法

字符串 format 方法的基本形式如下。

- **str.format(格式字符串,值 1, 值 2, …)　　#类方法**
- **格式字符串.format(值 1, 值 2, …)　　#对象方法**
- **格式字符串.format_map(mapping)**

格式字符串由固定文本和格式说明符混合组成。格式说明符的语法如下：

{[索引和键]:format_spec}

其中,可选的索引对应于要格式化参数值的位置,可选的键对应于要格式化的映射的键；格式化说明符(format_spec)同 format 内置函数。例如：

```
>>> "int: {0:d}; hex: {0:x}; oct: {0:o}; bin: {0:b}".format(100)
'int: 100; hex: 64; oct: 144; bin: 1100100'
>>> "int: {0:d}; hex: {0:#x}; oct: {0:#o}; bin: {0:#b}".format(100)
'int: 100; hex: 0x64; oct: 0o144; bin: 0b1100100'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c')    #输出:'c, b, a'
>>> str.format_map('{name:s},{age:d},{weight:3.2f}', {'name':'Mary', 'age':20, 'weight':49})
'Mary,20,49.00'
```

【例 5.21】 字符串示例(string.py)：格式化输出字符串堆积的三角形。其中,str.center()方法用于字符串两边填充；str.rjust(width[, fillchar])方法用于字符串右填充。

```
print("1".center(20))                    #1 行 20 个字符,居中对齐
print(format("121", "^20"))              #1 行 20 个字符,居中对齐
print(format("12321", "^20"))            #1 行 20 个字符,居中对齐
print("1".rjust(10," "))                 #1 行 10 个字符,右对齐,加空格
print(format("121", ">10"))              #1 行 10 个字符,右对齐,加空格
print(format("12321", ">10"))            #1 行 10 个字符,右对齐,加空格
print("1".rjust(20,"*"))                 #1 行 20 个字符,右对齐,加 * 号
print(format("121", "*>20"))             #1 行 20 个字符,右对齐,加 * 号
print(format("12321", "*>20"))           #1 行 20 个字符,右对齐,加 * 号
```

程序运行结果如图 5-2 所示。

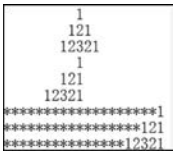


图 5-2　格式化输出字符串堆积的三角形

5.5.8　f 字符串(字符串插值)

自 Python 3.6 版本开始,增加了对格式化字符串变量的支持,以 f 开始的字符串中可以包含嵌入在花括号“{}”中的变量,称之为字符串变量替换(字符串插值)。例如：

```
>>> name = "Jerry"
>>> f"He said his name is {name}."              #输出:'He said his name is Jerry.'
>>> score, width, precision = 12.34567, 10, 2
>>> f"result: {score:{width}.{precision}f}"     #输出:'result:  12.35'
```

5.5.9 字符串的典型示例代码

字符串的典型示例如表 5-7 所示。假设该表中的示例基于 s1='abcdefg'、s2='The quick brown fox'以及 t1=('P', 'y', 't', 'h', 'o', 'n')。

表 5-7 字符串的典型示例

功能示例	实现代码	结果说明
使用字面量创建字符串对象	aStr = "Hello"	aStr ='Hello'
使用转义字符	print('abc\tXYZ')	abc XYZ
使用字符串编码(ord()和 chr()函数)	ord('A') chr(97)	65 'a'
使用 str 对象创建字符串对象	s = str(123)	s='123'
字符串格式化(字符串对象的 format()方法)	'The price is: {0: 8.2f}'.format(9.9)	'The price is: 9.90'
字符串格式化(str 的静态方法 format())	str.format('The price is: {0: 8.2f}', 9.9)	'The price is: 9.90'
字符串格式化(format()函数)	format(9.9, '8.2f')	' 9.90'
字符串格式化(%运算符)	'The price is: %8.2f' % 9.9	'The price is: 9.90'
字符串格式化(f 字符串)	price = 9.9 f'The price is: {price: 8.2}'	'The price is: 9.9'
使用索引访问字符串的元素	s1[1] s1[-1]	'b' 'g'
使用 for 循环访问字符串的元素	for i in s1: print(i, end='')	abcdefg
字符串的切片操作	s1[1: 3] s1[: 2]	'bc' 'ab'
字符串的连接操作(+运算符)	'abc' + '123'	'abc123'
字符串的重复操作(*运算符)	'Ha' * 3	'HaHaHa'
字符串的成员关系操作(in、notin 运算符)	'a' in s1 'A' not in s1	True True
字符串的比较运算操作(==、!=、>、>=、<、<=)	'abc'> 'ABC'	True
返回字符串中元素的个数(len()函数)	len(s1)	7
返回指定元素在字符串中出现的次数(count()方法)	s2.count('o')	2
返回指定元素在字符串中的最低索引(index()、rindex()、find()、rfind()方法)	s2.index('o') s2.rfind('o')	12 17
字符串的类型判断(isalnum()、isupper()等方法)	'123'.isdigit()	True
字符串大小写转换(lower()、title()等方法)	'blue sky'.title()	'Blue Sky'
判断字符串是否以指定前缀开始(startswith()方法)	s2.startswith('Th')	True
判断字符串是否以指定后缀结束(endswith()方法)	s2.endswith('fox')	True
字符串拆分(split()、rsplit()、partition()、rpartition()、splitlines()方法)	'Hello, world'.split() 'Hello, world'.split(',')	['Hello,', 'world'] ['Hello', ' world']
连接可迭代对象的元素以创建字符串(join()方法)	char = '' char.join(t1)	'Python'

5.5.10 字符串应用举例

【例 5.22】 字符串的应用示例 1(str_count.py)：输入任意字符串，统计其中元音字母('a'、'e'、'i'、'o'、'u'，不区分大小写)出现的次数和频率。

```
s1 = input('请输入字符串:')                    # 'The quick brown fox jumps over the lazy dog'
s2 = s1.upper()                                # 转换为大写
countall = len(s1)                             # 字符串长度
counta = s2.count('A')                         # 统计元音字母'a'(不区分大小写)出现的次数
counte = s2.count('E')                         # 统计元音字母'e'(不区分大小写)出现的次数
counti = s2.count('I')                         # 统计元音字母'i'(不区分大小写)出现的次数
counto = s2.count('O')                         # 统计元音字母'o'(不区分大小写)出现的次数
countu = s2.count('U')                         # 统计元音字母'u'(不区分大小写)出现的次数
print('所有字母的总数为:', countall)
print('元音字母出现的次数和频率分别为:')
print('A:{0}\t{1:2.2f} % '.format(counta, counta/countall * 100))
print('E:{0}\t{1:2.2f} % '.format(counte, counte/countall * 100))
print('I:{0}\t{1:2.2f} % '.format(counti, counti/countall * 100))
print('O:{0}\t{1:2.2f} % '.format(counto, counto/countall * 100))
print('U:{0}\t{1:2.2f} % '.format(countu, countu/countall * 100))
```

程序运行结果如图 5-3 所示。

【例 5.23】 字符串的使用示例 2(txt_count.py)：读取文本文件，统计其中的行数、字符数和单词个数。

```
file_name = "txt_count.py"                     # 文本文件名(即本例的源文件)
line_counts = 0                                # 行数
word_counts = 0                                # 单词个数
character_counts = 0                           # 字符数
with open(file_name, 'r', encoding = 'utf8') as f:    # 打开文件(读取模式,指定编码)
    for line in f:
        words = line.split()                   # 分离出单词
        line_counts += 1                       # 行数加 1
        word_counts += len(words)              # 单词个数加 1
        character_counts += len(line)          # 字符数加 1
print("行数:", line_counts)
print("单词个数:", word_counts)
print("字符个数:", character_counts)
```

程序运行结果如图 5-4 所示。

```
请输入字符串: The quick brown fox jumps over the lazy dog
所有字母的总数为:  43
元音字母出现的次数和频率分别为:
A: 1    2.33%
E: 3    6.98%
I: 1    2.33%
O: 4    9.30%
U: 2    4.65%
```

图 5-3 统计元音字母出现的次数和频率

```
行数:  13
单词个数:  47
字符个数:  470
```

图 5-4 统计文件中行数字符数和单词个数

5.6 字节序列

字节序列(bytes 和 bytearray)是由 8 位字节数据组成的序列数据类型，即 $0 \leqslant x < 256$ 的整数序列。Python 内置的字节序列数据类型包括 bytes(不可变对象)、bytearray(可变对象)

和 memoryview。

字节序列以二进制形式来存储数据，最终由程序解析并确定这些数据表示的内容(字符串、数字、图片、音频等)。例如，通过使用 urllib.request 模块读取的网页内容为字节序列，需要通过 decode 方法解码为字符串；反之亦然，字符串也可以转换成字节序列。

5.6.1 bytes 字面量

使用字母 b 加单引号或双引号括起来的内容是 bytes 字面量。Python 解释器自动创建 bytes 型对象实例。bytes 字面量与字符串定义方式类似。

(1) 单引号(b' ')。包含在单引号中的字符串，其中可以包含双引号。

(2) 双引号(b" ")。包含在双引号中的字符串，其中可以包含单引号。

(3) 三单引号(b''' ''')。包含在三单引号中的字符串，可以跨行。

(4) 三双引号(b""" """)。包含在三双引号中的字符串，可以跨行。

注意：在引号中只能包含 ASCII 码字符，否则导致 SyntaxError。例如：

```
>>> b'张'                #SyntaxError: bytes can only contain ASCII literal characters
```

【例 5.24】 bytes 字面量示例。

```
>>> b'abc'
b'abc'
>>> b'abc\'x\''
b"abc'x'"
>>> b'abc"x"'
b'abc"x"'
>>> x = b'c:\\Python33'
>>> x
b'c:\\Python33'
>>> b"xyz"
b'xyz'
>>> b"x\tyz"
b'x\tyz'
>>> print(b"x\tyz")
b'x\tyz'
>>> print(b"x'y'z")
b"x'y'z"
>>> print(b"x\ny")
b'x\ny'
>>> s1 = '''a
\tb
\tc
\td'''
>>> s1 = b'''a
\tb
\tc
\td'''
>>> s1
b'a\n\tb\n\tc\n\td'
>>> s2 = b"""
She said:
"Yes!"
"""
>>> s2
b'\nShe
said:\n"Yes!"\n'
>>> print(s2)
b'\nShe
said:\n"Yes!"\n'
```

5.6.2 创建 bytes 对象

创建 bytes 类型的对象实例的基本形式如下。

```
• bytes()                              #创建空 bytes 对象
• bytes(n)                             #创建长度为 n(整数)的 bytes 对象，各字节为 0
• bytes(iterable)                      #创建 bytes 对象，使用 iterable 中的字节整数
• bytes (object)                       #创建 bytes 对象，拷贝 object 字节数据
• bytes([source[, encoding[, errors]]])  #创建 bytes 对象
```

如果 iterable 中包含非 0≤x<256 的整数，将导致 ValueError。

【例 5.25】 创建 bytes 对象示例。

```
>>> bytes()
b''
>>> bytes(2)
b'\x00\x00'
>>> bytes((1,2,3))
b'\x01\x02\x03'
>>> bytes('abc','utf-8')
b'abc'
>>> bytes((123, 456))
Traceback (most recent call last):
  File "<pyshell  #95>", line 1, in <module>
    bytes((123, 456))
ValueError: bytes must be in range(0, 256)
```

5.6.3 创建 bytearray 对象

创建 bytearray 类型的对象实例的基本形式如下。

- bytearray()　　#创建空 bytearray 对象
- bytearray(n)　　#创建长度为 n(整数)的 bytearray 对象,各字节为 0
- bytearray(iterable)　　#创建 bytearray 对象,使用 iterable 中的字节整数
- bytearray(object)　　#创建 bytearray 对象,拷贝 object 字节数据
- bytearray([source[, encoding[, errors]]])　　#创建 bytearray 对象

如果 iterable 中包含非 0≤x<256 的整数,则导致 ValueError。

【例 5.26】 创建 bytearray 对象示例。

```
>>> bytearray()
bytearray(b'')
>>> bytearray(2)
bytearray(b'\x00\
x00')
>>> bytearray((1,2,3))
bytearray(b'\x01\x02\
x03')
>>> bytearray('abc','utf
-8')
bytearray(b'abc')
>>> bytearray((123,456))
Traceback (most recent call last):
  File "< pyshell    # 102 >", line 1, in
<module>
    bytearray((123,456))
ValueError: byte must be in range(0, 256)
```

5.6.4 bytes 和 bytearray 的序列操作

bytes 和 bytearray 支持序列的基本操作,包括索引访问、切片操作、连接操作、重复操作、成员关系操作、比较运算操作,以及求序列长度、最大值、最小值等内置函数。

bytes 和 bytearray 一般基于 ASCII 字符串,故 bytes 和 bytearray 基本上支持字符串对象的类似方法,但不支持 str.encode()(把字符串转换为 bytes 对象)、str.format()/str.format_map()(字符串格式化)、str.isidentifier()/str.isnumeric()/str.isdecimal()/str.isprintable()(这些判断无意义)。

注意:bytes 和 bytearray 的方法不接受字符串参数,只接受 bytes 和 bytearray 参数,否则将导致 TypeError。

【例 5.27】 字节的序列操作示例。

```
>>> b1 = b"abc"
>>> b1.replace(b'a',b'f')        #输出:b'fbc'
>>> b1.replace('b','g')          # TypeError: a bytes-like object is required, not 'str'
```

5.6.5 字节编码和字节解码

字符串可以通过 str.encode()方法编码为字节码,通过 bytes 和 bytearray 的 decode()方法解码为字符串。

默认情况下,Python 字符串采用 utf-8 编码。创建字符串时,也可以指定其编码方式:

```
str(object = b'', encoding = 'utf-8', errors = 'strict')      #按指定编码,根据字节码对象创
                                                              #建 str 对象
```

其中,object 为字节码对象(bytes 或 bytearray);encoding 为编码;errors 为错误控制。该构造函数的结果等同于 bytes 对象 b 的对象方法:

```
b.decode(encoding, errors)                                    #把字节码对象 b 解码为对应编码的字符串
```

与之相对应,用户也可以把字符串对象 s 编码为字节码对象:

```
s.encode(encoding = "utf-8", errors = "strict")    #把字符串对象 s 编码为字节码对象
```

【例 5.28】 字符串编码和解码示例。

```
>>> s1 = 'Sample!例子!'
>>> b1 = s1.encode(encoding = 'cp936')
>>> b1
b'Sample!\xc0\xfd\xd7\xd3\xa3\xa1'
>>> b1.decode(encoding = 'cp936')
'Sample!例子!'
>>> b1.decode()          #默认解码 utf-8,出错
Traceback (most recent call last):
  File "<pyshell   #56>", line 1, in <module>
    b1.decode()
UnicodeDecodeError: 'utf-8' codec can't decode byte
0xc0 in position 7: invalid start byte
```

【例 5.29】 字节编码和解码示例。

```
>>> s = '好好学习'
>>> b = s.encode()
>>> b                          #输出:b'\xe5\xa5\xbd\xe5\xa5\xbd\xe5\xad\xa6\xe4\xb9\xa0'
>>> b.decode()                 #输出:'好好学习'
```

5.6.6 字节序列的典型示例代码

字节序列的典型示例如表 5-8 所示。假设该表中的示例基于 s1='Hello'以及 b1=b'Hello'。

表 5-8 字节序列的典型示例

功能示例	实现代码	结果说明
使用字面量创建字节序列对象	b1=b"Hello"	b1=b'Hello'
使用 bytes 对象创建不可变字节序列对象	bytes((1,2,3))	b'\x01\x02\x03'
使用 bytearray 对象创建可变字节序列对象	bytearray((1,2,3))	bytearray(b'\x01\x02\x03')
把字符串对象编码为字节序列对象(encode 方法)	s1.encode()	b'Hello'
把字节序列对象解码为字符串对象(decode 方法)	b1.decode()	'Hello'

5.7 字典(映射)

字典(dict,或映射(map))是一组键/值对的数据结构。每个键对应于一个值。在字典中键不能重复。根据键可以查询到值。

5.7.1 对象的哈希(hash)值

字典是键和值的映射关系。字典的键必须是可 hash 的对象,即实现了__hash__()的对象。对象的 hash 值也可以使用内置函数 hash()获得。

【例 5.30】 对象的哈希(hash)值示例。

```
>>> hash(100) #结果:100
>>> hash(1.23) #结果:530343892119149569
```

不可变对象(bool、int、float、complex、str、tuple、frozenset 等)是可 hash 对象。可变对象通常是不可 hash 对象,因为可变对象的内容可以改变,因此无法通过 hash()函数获取唯一的 hash 值。

字典的键只能使用不可变对象,但字典的值可以使用不可变对象或可变对象。一般而言,应该使用简单的对象作为键。

5.7.2 字典的定义

字典通过花括号中用逗号分隔的项目(键/值。键/值对使用冒号分隔)定义。其基本形

式如下。

```
{键 1:值 1 [, 键 2:值 2, …, 键 n:值 n]}
```

键必须为可 hash 对象，因此不可变对象（bool、int、float、complex、str、tuple、frozenset等）可以作为键；值则可以为任意对象。字典中的键是唯一的，不能重复。

字典也可以通过创建 dict 对象来创建。其基本形式如下。

```
• dict()                    # 创建一个空字典
• dict( ** kwargs)          # 使用关键字参数创建一个新的字典.此方法最紧凑
• dict(mapping)             # 从一个字典对象创建一个新的字典
• dict(iterable)            # 使用序列创建一个新的字典
```

【例 5.31】 创建字典对象示例。

```
>>> {}                              >>> dict({1:'food', 2:'drink'})
{}                                  {1: 'food', 2: 'drink'}
>>> {'a':'apple','b':'boy'}         >>> dict([('id','1001'),('name','Jenny')])
{'a': 'apple', 'b': 'boy'}          {'id': '1001', 'name': 'Jenny'}
>>> dict()                          >>> dict(baidu = 'baidu.com', google = 'google.com')
{}                                  {'baidu': 'baidu.com', 'google': 'google.com'}
```

5.7.3 字典解析表达式

使用字典解析表达式可以简单、高效地处理一个可迭代对象，并生成结果字典。字典解析表达式的形式如下。

```
• {k: v for i1 in 序列 1 … for iN in 序列 N}                  # 迭代序列里所有内容,并计算生成字典
• {k: v for i1 in 序列 1 … for iN in 序列 N if cond_expr}     # 按条件迭代,并计算生成字典
```

表达式 k 和 v 使用每次迭代内容 $i_1 \cdots i_N$ 计算生成一个字典。如果指定了条件表达式 cond_expr，则只有满足条件的元素参与迭代。

【例 5.32】 字典解析表达式示例。

```
>>> {key:value for key in "ABC" for value in range(3)}
{'A': 2, 'B': 2, 'C': 2}
>>> d1 = {1:'food', 2:'drink', 3:'fruit'}
>>> {value:key for key,value in d1.items()}
{'food': 1, 'drink': 2, 'fruit': 3}
>>> {x:x * x for x in range(10) if x % 2 == 0}
{0: 0, 2: 4, 4: 16, 6: 36, 8: 64}
```

5.7.4 字典的访问操作

字典 d 可以通过键 key 来访问，其基本形式如下。

```
• d[key]                    # 返回键为 key 的 value; 如果 key 不存在,则导致 KeyError
• d[key] = value            # 设置 d[key]的值为 value; 如果 key 不存在,则添加键/值对
• del d[key]                # 删除字典元素; 如果 key 不存在,则导致 KeyError
```

【例 5.33】 字典的访问示例。

```
>>> d = {1:'food', 2:'drink'}          >>> del d[2]
>>> d                                  >>> d
{1: 'food', 2: 'drink'}                {1: 'food', 3: 'fruit'}
>>> d[1]                               >>> d[2]
```

```
'food'                                  Traceback (most recent call last):
>>> d[3] = 'fruit'                        File "<pyshell#100>", line 1, in <module>
>>> d                                       d[2]
{1: 'food', 2: 'drink', 3: 'fruit'}     KeyError: 2
```

5.7.5 字典的视图对象

字典 d 支持下列视图对象，通过它们可以动态访问字典的数据。

- **d.keys()** #返回字典 d 的键 key 的可迭代对象
- **d.values()** #返回字典 d 的值 value 的可迭代对象
- **d.items()** #返回字典 d 的(key, value)对的可迭代对象

【例 5.34】 字典的视图对象示例。

```
>>> d = {1:'food', 2:'drink',      >>> d.values()                  >>> d.items()
3:'fruit'}                         dict_values(['food', 'drink',  dict_items([(1, 'food'), (2,
>>> d.keys()                       'fruit'])                      'drink'), (3, 'fruit')])
dict_keys([1, 2, 3])               >>> for v in d.values():       >>> for item in d.items():
>>> for k in d.keys():                 print(v, end = " ")            print(item, end = ' ')
    print(k, end = " ")            food drink fruit               (1, 'food') (2, 'drink') (3,
1 2 3                                                             'fruit')
```

5.7.6 字典的遍历

字典 d 及其视图 d.items()、d.values()、d.keys()都是可迭代对象，可以使用 for 循环进行迭代。例如：

```
>>> d = {1:'food', 2:'drink', 3:'fruit'}
>>> for k in d:
    print("键 = {},值 = {};".format(k, d[k]), end = " ")
键 = 1,值 = food; 键 = 2,值 = drink; 键 = 3,值 = fruit;
>>> for k, v in d.items():
    print("键 = {},值 = {};".format(k, v), end = " ")
键 = 1,值 = food; 键 = 2,值 = drink; 键 = 3,值 = fruit;
>>> for (k, v) in d.items():
    print("键 = {},值 = {};".format(k, v), end = " ")
键 = 1,值 = food; 键 = 2,值 = drink; 键 = 3,值 = fruit;
```

5.7.7 判断字典键是否存在

用户可以通过下列方式之一判断键 key 是否存在于字典 d 中。

- **key in d** #如果为 True，则表示存在
- **key not in d** #如果为 True，则表示不存在

【例 5.35】 判断字典键是否存在示例。

```
>>> d = dict(a = 'apple',b = 'boy',c = 'cat',d = 'dog')    >>> 'a' in d
>>> d                                                      True
{'d': 'dog', 'a': 'apple', 'c': 'cat', 'b': 'boy'}        >>> 'e' not in d
                                                          True
```

5.7.8 字典对象的长度和比较

通过内置函数 len()可以获取字典的长度(元素个数)。虽然字典对象也支持内置函数

max()、min()、sum()，以计算字典 Key，但没有太大意义。另外，字典对象也支持比较运算符(<、<=、==、!=、>=、>)，但只有==、!=有意义。

【例 5.36】 字典对象的长度和比较示例。

```
>>> d1 = {1:'food', 2:'drink'}
>>> d2 = {1:'food', 2:'drink', 3:'fruit'}
>>> d3 = {1:'food', 2:'drink', 3:'fruit'}
>>> len(d1)
2
>>> d1 == d2
False
>>> d2 != d3
False
```

5.7.9 字典对象的方法

字典是可变对象，其包含的主要方法如表 5-9 所示。假设表中的示例基于 d={1: 'food', 2: 'drink', 3: 'fruit'}。

表 5-9 字典对象的主要方法

方 法	说 明	示 例	结 果
d.clear()	删除所有元素	d.clear()	d={}
d.copy()	浅拷贝字典	d1=d.copy() id(d), id(d1)	d1=d=[1, 3, 2] d 和 d1 的 ID 不同
d.get(k)	返回键 k 对应的值，如果 key 不存在，返回 None	d.get(1),d.get(5)	('food', None)
d.get(k, v)	返回键 k 对应的值，如果 key 不存在，返回 v	d.get(1,'无'),d.get(5,'无')	('food', '无')
d.pop(k)	如果键 k 存在，返回其值，并删除该项目；否则导致 KeyError	d.pop(1)	输出'food'。d={2: 'drink', 3: 'fruit'})
d.pop(k, v)	如果键 k 存在，返回其值，并删除该项目；否则返回 v	d.pop(5,'无')	输出'无'。d={1: 'food', 2: 'drink', 3: 'fruit'}
d.popitem()	删除并返回最后添加的“键-值”对，结果为元组(k, v)	d.popitem()	(3, 'fruit')
d.setdefault(k, v)	如果键 k 存在，返回其值；否则添加项目 k=v，v 默认为 None	d.setdefault(1) d.setdefault(4)	'food' d={1: 'food', 2: 'drink', 3: 'fruit', 4: None}
d.update([other])	使用字典或键值对，更新或添加项目到字典	d1={1: '食物',4: '书籍'} d.update(d1)	d={1: '食物', 2: 'drink', 3: 'fruit', 4: '书籍'}

5.7.10 字典的典型示例代码

字典的典型示例如表 5-10 所示。假设表中的示例基于 d1={'zhangsan': 92, 'lisi': 89}。

表 5-10 字典的典型示例

功 能 示 例	实 现 代 码
使用字面量创建字典对象	>>> d1 = {'zhangsan': 92, 'lisi': 89}
使用 dict 对象创建字典对象	>>> d2 = dict(id=1001, name='zhangsan', score=92) >>> d2 #输出：{'id': 1001, 'name': 'zhangsan', 'score': 92}

续表

功能示例	实现代码
使用字典解析表达式创建字典对象	>>> d3 ={x: x * x for x in range(5)} >>> d3 #输出：{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

字典拷贝(copy 方法)	>>> d2 = d1.copy()
	>>> d1 == d2 #输出：True
	>>> d1 is d2 #输出：False

访问字典键的值	>>> d1['zhangsan'] #输出：92
访问字典键的值(get 方法)	>>> d1.get('lisi') #输出：89 >>> d1.get('wangwu',0) #输出：0
访问字典键的值，如果不存在则添加(setdefault 方法)	>>> d1.setdefault('lisi', 87) #输出：89 >>> d1.setdefault('wangwu', 87) #输出：87
设置字典键的值	>>> d1['zhangsan'] = 95 >>> d1 #输出：{'zhangsan': 95, 'lisi': 89}
添加新的“键－值”对	>>> d1['wangwu'] = 82 >>> d1 #输出：{'zhangsan': 95, 'lisi': 89, 'wangwu': 82}
从字典中删除或移除元素(pop 方法)	>>> d1.pop('zhangsan') #输出：92 >>> d1 #输出：{'lisi': 89}

把一个字典的内容添加到另一个字典中(update 方法)	>>> d2 = {'wangwu': 87, 'zhaoliu': 72}
	>>> d1.update(d2); d1
	{'zhangsan': 92, 'lisi': 89, 'wangwu': 87, 'zhaoliu': 72}

删除字典键(del 语句)	>>> del d1['zhangsan'] >>> d1 #输出：{'lisi': 89}
返回字典的键 key 的列表(keys 方法)	>>> d1.keys() #输出：dict_keys(['zhangsan', 'lisi'])
返回字典的值 value 的列表(values 方法)	>>> d1.values() #输出：dict_values([92, 89])
返回字典 d 的(key, value)对的列表(items 方法)	>>> d1.items() dict_items([('zhangsan', 92), ('lisi', 89)])
使用 for 循环遍历字典	>>> for k in d1: print(f'{k}: {d1[k]}') zhangsan: 92 lisi: 89
字典的成员关系操作(in、not in 运算符)	>>> 'zhangsan' in d1 #输出：True >>> 'jack'not in d1 #输出：True
清空字典内容	>>> d1.clear() #结果 d1 为{}

5.8 集合

集合数据类型是没有顺序的简单对象的聚集，且集合中元素不重复。Python 集合数据类型包括可变集合对象(set)和不可变集合对象(frozenset)。

5.8.1 集合的定义

可变集合(set)通过花括号中用逗号分隔的项目定义。其基本形式如下：

$\{x_1[, x_2, \cdots, x_n]\}$

其中，x_1，x_2，…，x_n 为任意的可 hash 对象。集合中的元素不可重复，且无序，其存储依据对象的 hash 码。hash 码是根据对象的值计算出来的一个唯一值。一个对象如果定义了特殊方法__hash__()，则该对象为可 hash 对象。所有内置不可变对象(bool、int、float、complex、str、tuple、frozenset 等)都是可 hash 对象；所以内置可变对象(list、dict、set)都是非 hash 对象(因为可变对象的值可以变化，故无法计算一个唯一的 hash 值)。在集合中可以包含内置不可变对象，不能包含内置可变对象。

注意：{}表示空的 dict，因为 dict 也使用花括号定义。空集为 set()。

用户可以通过创建 set 对象来创建可变集合，通过创建 frozenset 对象来创建不可变集合。其基本形式如下。

- `set()` # 创建一个空的可变集合
- `set(iterable)` # 创建一个可变集合，包含的项目为可枚举对象 iterable 中的元素
- `frozenset()` # 创建一个空的不可变集合
- `frozenset(iterable)` # 创建一个不可变集合，包含的项目为可枚举对象 iterable 中的元素

【例 5.37】 创建集合对象示例。

```
>>> {1,2,1}
{1, 2}
>>> {1,'a',True}
{1, 'a'}
>>> {1.2, True}
{True, 1.2}
>>> set()
set()
>>> frozenset()
frozenset()
>>> set('Hello')
{'H', 'e', 'o', 'l'}
>>> {'a',[1,2]}
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    {'a',[1,2]}
TypeError: unhashable type: 'list'
```

5.8.2 集合解析表达式

使用集合解析表达式可以简单、高效地处理一个可迭代对象，并生成结果集合。集合解析表达式的形式如下。

- `{expr for i1 in 序列 1…for iN in 序列 N}` # 迭代序列里所有内容，并计算生成集合
- `{expr for i1 in 序列 1…for iN in 序列 N if cond_expr}` # 按条件迭代，并计算生成集合

表达式 expr 使用每次迭代内容 $i_1 \cdots i_N$ 计算生成一个集合。如果指定了条件表达式 cond_expr，则只有满足条件的元素参与迭代。

【例 5.38】 集合解析表达式示例。

```
>>> {i for i in range(5)}                    #输出:{0, 1, 2, 3, 4}
>>> {2 ** i for i in range(5)}               #输出:{1, 2, 4, 8, 16}
>>> {x ** 2 for x in [1, 1, 2]}              #输出:{1, 4}
```

5.8.3 判断集合元素是否存在

用户可以通过下列方式之一判断一个元素 x 是否在集合 s 中存在。

- x in s：如果为 True，则表示存在。
- x not in s：如果为 True，则表示不存在。

【例 5.39】 集合中元素的判断示例。

```
>>> s = set('Hello')
>>> s
{'H', 'e', 'o', 'l'}
>>> 'h' in s
False
>>> 'o' not in s
False
```

5.8.4 集合的运算：并集、交集、差集和对称差集

集合支持表 5-11 所示的集合运算。

表 5-11 集合运算

运 算 符	说 明
s1 | s2 | …	返回 s1、s2、…的并集：s1∪s2∪…
s1 & s2 & …	返回 s1、s2、…的交集：s1∩s2∩…
s1 - s2 - …	返回 s1、s2、…的差集，也记作 s1\s2\…
s1 ^ s2	返回 s1、s2 的对称差集：s1△s2

集合的对象方法如表 5-12 所示。

表 5-12 集合的对象方法

方 法	说 明
s1.isdisjoint(s2)	如果集合 s1 和 s2 没有共同元素，返回 True；否则返回 False
s1.issubset(s2)	如果集合 s1 是 s2 的子集，返回 True；否则返回 False
s1.issuperset(s2)	如果集合 s1 是 s2 的超集，返回 True；否则返回 False
s1.union(s2,…)	返回 s1、s2、…的并集：s1∪s2∪…
s1.intersection(s2,…)	返回 s1、s2、…的交集：s1∩s2∩…
s1.difference(s2,…)	返回 s1、s2、…的差集：s1-s2-…
s1.symmetric_difference(s2)	返回 s1 和 s2 的对称差集：s1△s2

【例 5.40】 集合的运算示例。

```
>>> s1 = {1,2,3}
>>> s2 = {2,3,4}
>>> s1 | s2
{1, 2, 3, 4}
>>> s1 & s2
{2, 3}
>>> s1 - s2
{1}
>>> s1 ^ s2
{1, 4}
>>> s1.union(s2)
{1, 2, 3, 4}
>>> s1.intersection(s2)
{2, 3}
>>> s1.difference(s2)
{1}
>>> s1.symmetric_difference(s2)
{1, 4}
```

5.8.5 集合的比较运算：相等、子集和超集

集合支持表 5-13 所示的比较运算。

表 5-13 集合的比较运算

运 算 符	说 明	运 算 符	说 明
s1 == s2	s1 和 s2 的元素相同	s1 <= s2	s1 是 s2 的子集
s1 != s2	s1 和 s2 的元素不完全相同	s1 >= s2	s1 是 s2 的超集
s1 < s2	s1 是 s2 的纯子集	s1 > s2	s1 是 s2 的纯超集

集合对象的比较方法包括 s1.isdisjoint(s2)、s1.issubset(s2)和 s1.issuperset(s2)。

【例 5.41】 集合比较运算示例。

```
>>> s1 = {1,2,3}
>>> s2 = {3,2,1}
>>> s3 = {1,2}
>>> s4 = {7,9}
>>> s1 == s2
True
>>> s1!= s4
True
>>> s3 <= s1
True
>>> s2 > s3
True
>>> s3 < s4
False
>>> s3 > s4
False
>>> s1 >= s2
True
>>> s1.isdisjoint(s2)
False
>>> s3.issubset(s1)
True
>>> s2.issuperset(s3)
True
```

5.8.6 集合的长度、最大值、最小值、元素和

通过内置函数 len()、max()、min()、sum()可以获取集合的长度、元素最大值、元素最小值、元素之和。如果元素有非整数，则求和将导致 TypeError。

【例 5.42】 集合的长度、最大值、最小值、元素和示例。

```
>>> s1 = {1,3,5,7,9}
>>> s2 = {'1','2','3'}
>>> len(s1)
5
>>> max(s1)
9
>>> min(s2)
'1'
>>> sum(s2)
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    sum(s2)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

5.8.7 可变集合的方法

set 集合是可变对象，其包含的主要方法如表 5-14 所示。假设表中的示例基于“s1＝{1,2,3}；s2＝{2,3,4}”。

表 5-14 可变集合对象的主要方法

方 法	说 明	示 例	结 果

s1.union(s2, …)
s1.update(s2, …)
s1 |＝s2 | …
	并集(返回新的集合)
	并集(更新原集合)
	s1＝ s1∪s2∪…
		>>> s1.union(s2)
		>>> s1
		>>> s1.update(s2); s1
			{1, 2, 3, 4}
			{1, 2, 3}
			{1, 2, 3, 4}

s1.intersection(s2, …)
s1.intersection_update(s2, …)
s1 &＝ s2 &…
	交集(返回新的集合)
	交集(更新原集合)
	s1＝ s1∩s2∩…
		>>> s1.intersection(s2)
		>>> s1
		>>> s1.intersection_update(s2); s1
			{2, 3}
			{1, 2, 3}
			{2, 3}

s1.difference(s2, …)
s1.difference_update(s2, …)
s1 －＝ s2 －…
	交集(返回新的集合)
	差集(更新原集合)
	s1＝ s1－s2－…
		>>> s1.difference(s2)
		>>> s1
		>>> s1.difference_update(s2);s1
			{1}
			{1, 2, 3}
			{1}

s1.symmetric_difference(s2)
s1.symmetric_difference_update(s2)
s1 ^＝ s2
	对称差集(返回新的集合)
	对称差集(更新原集合)
	s1＝ s1△s2
		>>> s1.symmetric_difference(s2)
		>>> s1
		>>> s1.symmetric_difference_update(s2)
		>>> s1
			{1, 4}
			{1, 2, 3}
			{1, 4}

s.add(x)	把对象 x 添加到集合 s	>>> s1.add('a'); s1	{1, 2, 3, 'a'}
s.remove(x)	从集合 s 中移除对象 x。若不存在，则导致 KeyError	>>> s1.remove(1); s1	{2, 3}
s.discard(x)	从集合 s 中移除对象 x。若不存在，不报错	>>> s1.discard(3); s1	{1, 2}
s.pop()	从集合 s 随机弹出一个元素，如果 s 为空，则导致 KeyError	>>> s1.pop() >>> s1	1 {2, 3}

s.copy()	拷贝集合	>>> s3 ＝ s1.copy()
		>>> s1 ＝＝ s3
		>>> s1 is s3
			True
			False

续表

方　法	说　明	示　例	结　果
s.clear()	清空集合 s	>>> s1.clear(); s1	set()
s1.isdisjoint(s2)	判断集合 s1 和 s2 是否不相交	>>> s1.isdisjoint(s2)	False

5.8.8 集合的典型示例代码

集合的典型示例如表 5-15 所示。假设表中的示例基于“s1＝{1,2,3}；s2＝{2,3,4}；s3＝{'a', 'e', 'i', 'o', 'u'}”。

表 5-15　集合的典型示例

功能示例	实现代码	结果说明
使用字面量创建集合对象	set1 = {1, 2}	set1={1, 2}
使用 set 对象创建集合对象	set2 =set('a book')	set2= {' ', 'o', 'a', 'b', 'k'}
使用集合解析表达式创建集合对象	{i for i in range(6) if i%2 == 0}	{0, 2, 4}
集合拷贝(copy()方法)	s4 = s1.copy() s4 == s1 s4 is s1	True False
添加元素到集合中(add()方法)	s3.add('y') s3.add('i')	s3={'u', 'o', 'a', 'i', 'e', 'y'} s3={'u', 'o', 'a', 'i', 'e', 'y '}
从集合中删除指定元素(remove()方法)	s3.remove('i') s3.remove('x')	s3={'a', 'e', 'o', 'u'} KeyError: 'x'
从集合中删除指定元素(discard()方法)	s3.discard('i') s3.discard('x')	s3 = {'a', 'e', 'o', 'u'} 不报错
从集合中随机删除并返回一个元素(pop()方法)	s3.pop()	输出'e'。s3 = {'a', 'u', 'i', 'o'}
集合的成员关系操作(in、not in 运算符)	1 in s1 1 not in s1	True False
判断两个集合是否不相交	s1.isdisjoint(s2)	False
并集(union()、update()方法,|运算符)	s1.union(s2) s1 | s2	{1, 2, 3, 4} {1, 2, 3, 4}
交集(intersection()、intersection_update()方法,& 运算符)	s1.intersection(s2) s1 & s2	{ 2, 3} { 2, 3}
差集(difference()、difference_update()方法、－运算符)	s1.difference(s2) s1 － s2	{1} {1}
对称差集(symmetric_difference()、symmetric_difference_update()方法、^运算符)	s1. symmetric _ difference (s2) s1 ^ s2	{1, 4} {1, 4}
清空集合内容	s1.clear()	s1= set()(空集合)

5.9 数据类型综合应用举例

5.9.1 字符串简单加密和解密

基于按位逻辑异或的简单加密算法的原理如下：给定明文字符(例如 A)和秘钥字符(例如 P),其对应的 ASCII 码的按位逻辑异或即是加密后的密文字符,密文字符的 ASCII

码与相同秘钥字符的 ASCII 码按位逻辑异或后的字符则为加密前的明文。例如：

```
>>> ord('A') ^ ord('P')     #输出:17
>>> chr(17 ^ ord('P'))      #输出:'A'
```

故基于按位逻辑异或的简单字符串加密算法和解密算法可以共用一个函数，其设计思路如下。

（1）给定字符串 text(例如 The quick brown fox jumps over the lazy dog)和 key(例如 Python_1)，使用 itertools.cycle(key)构造一个循环字符串迭代器 keys。

（2）循环处理 text 的每个字符，使用 keys 中对应字符进行按位逻辑异或运算，结果就是加密后的密文(如果解密，结果就是解密后的明文)。

【例 5.43】 列表和字符串的综合应用：字符串简单加密和解密(crypt.py)。

```
from itertools import cycle      #导入模块
def crypt(text, key):            #定义字符串加密和解密函数
    result = []                  #创建一个空列表
    keys = cycle(key)            #使用 itertools.cycle(key)构造一个循环字符串迭代器 keys
    for ch in text:      #循环处理 text 的每个字符,使用 keys 中对应字符按位逻辑异或运算
        result.append(chr(ord(ch)^ord(next(keys)))) #将处理后的元素追加到列表的尾部
    return ''.join(result)                          #将列表中的各个元素组合为字符串
if __name__ == '__main__':                          #如果独立运行时,则运行测试代码
    plain = 'The quick brown fox jumps over the lazy dog'     #明文字符串
    key = 'Python_1'                                          #秘钥字符串
    print('加密前明文:{}'.format(plain))
    encrypted = crypt(plain, key)                             #字符串加密
    print('加密后密文:{}'.format(encrypted))
    decrypted = crypt(encrypted, key)                         #字符串解密
    print('解密后明文:{}'.format(decrypted))
```

程序运行结果如图 5-5 所示。

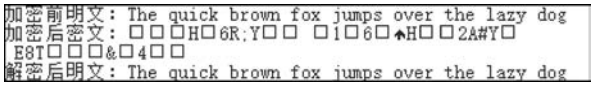


图 5-5 字符串简单加密和解密

5.9.2 去除列表中的重复项

用户可以通过构造一个集合来去除列表中的重复项，但结果不能保证原来的顺序。例如：

```
>>> a = [1, 8, 5, 1, 9, 2, 1, 10]
>>> list(set(a))          #输出:[1, 2, 5, 8, 9, 10]
```

通过定义一个生成器函数可以实现去除列表中重复元素同时保持原来顺序的功能。

【例 5.44】 列表和集合的综合应用：去除列表中的重复项生成器函数(deduplicate.py)。

```
def unique(items):                      #定义去除列表中重复项的生成器函数
    items_existed = set()               #创建一个空集合
    for item in items:                  #对于列表中的每一项
        if item not in items_existed:   #如果列表中的该项不在当前集合中
            yield item                  #生成器函数使用 yield 语句返回一个值,然后
                                        #保存当前函数的整个执行状态,等待下一次调用
```

```
            items_existed.add(item)    #将不重复元素 item 添加到集合中
if __name__ == "__main__":             #如果独立运行时,则运行测试代码
    a = [1, 8, 5, 1, 9, 2, 1, 10]      #存在重复项的列表
    a1 = unique(a)                     #去除列表中的重复项(保持原来的顺序)
    print(list(a1))                    #去除重复项后的列表内容
```

程序运行结果如图 5-6 所示。

```
[1, 8, 5, 9, 2, 10]
```

图 5-6 去除列表中的重复项

5.9.3 基于列表的简易花名册管理系统

通过列表可以很方便实现一个花名册管理系统,实现名字的显示、查询、增加、删除、修改等功能。

【例 5.45】 列表的综合应用:简易花名册管理系统(name_list.py)。

```
def menu():                                          #显示菜单
    print("=" * 35)
    print("简易花名册程序")
    print("1.显示名字")
    print("2.添加名字")
    print("3.删除名字")
    print("4.修改名字")
    print("5.查询名字")
    print("6.退出系统")
    print("=" * 35)
names = []                                           # 创建存储花名册的列表对象
while True:                                          # 重复执行
    menu()                                           #显示菜单
    num = input("请输入选择的功能序号(1 到 6):")     #获取用户输入
    # 根据用户的选择,执行相应的功能
    if num == '1':                                   #打印花名册列表清单
        print(names)
    elif num == '2':                                 #添加名字
        name = input("请输入要添加的名字:")
        names.append(name)
        print(names)
    elif num == '3':                                 #删除名字
        name = input("请输入要删除的名字:")
        if name in names:
            names.remove(name)
        else:
            print("系统中不存在名字:{}".format(name))
        print(names)
    elif num == '4':                                 #修改名字
        name = input("请输入要修改的名字:")
        if name in names:
            index = names.index(name)
            new_name = input("请输入修改后的名字:")
            names[index] = new_name
        else:
            print("系统中不存在名字:{}".format(name))
        print(names)
    elif num == '5':                                 #查询名字
        name = input("请输入要查询的名字:")
        if name in names:
            print("系统中存在名字:{}".format(name))
```

```
        else:
            print("系统中不存在名字:{}".format(name))
    elif num == '6':                                    #退出系统
        break
    else:                                  #只能输入1～6之间整数所对应的功能序号
        print("选项错误,请重新选择!")
```

程序运行结果如图 5-7 所示。

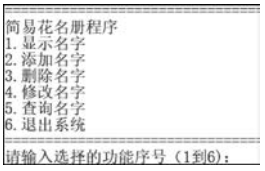


图 5-7　简易花名册管理系统

5.9.4　频数表和直方图

构建频数表的方法可以使用字典(键为项,值为计数)。通过遍历数据列表,在频数字典中递增对应项的值,可以很容易地实现频数表。

绘制直方图可以使用前面章节介绍的海龟对象。在前文的海龟绘图示例中,当创建一个新的海龟对象时,将自动创建一个海龟窗口。使用 turtle 模块中的 Screen()构造函数,还可以单独创建绘图窗口或者屏幕,然后添加海龟对象,以实现调用方法来自定义绘图窗口。

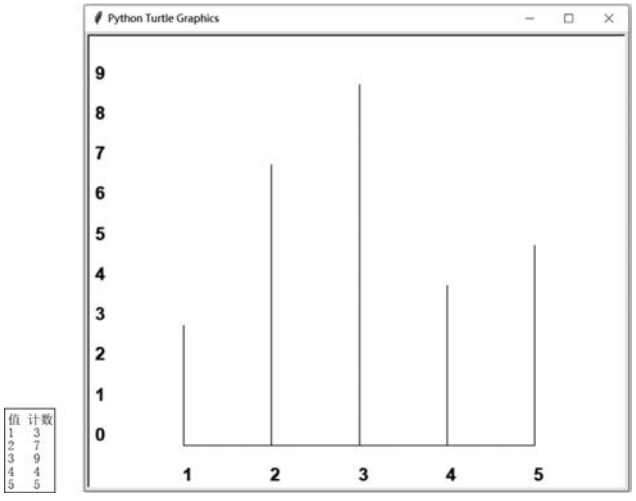
turtle 模块中的 Screen()构造函数及相关绘图函数的语法和调用形式如下。

- `wn = turtle.Screen()`　　#创建新的绘图窗口
- `wn.setworldcoordinates(xLL, yLL, xUR, yUR)`　　#调整窗口坐标,使窗口左下角的点为(xLL, yLL),窗口右上角的点为(xUR, yUR)
- `wn.exitonclick()`　　#当用户在窗口中的某个位置单击鼠标时,关闭窗口

【例 5.46】　字典和列表的综合应用：利用字典构建给定列表数据的频数表,并利用海龟绘制频数表所对应的直方图(freq.py)。程序运行结果如图 5-8 所示。

```
import turtle                                       #导入海龟模块
def freq_table(data_list):      #统计列表 data_list 中的值的个数,返回包含值计数的字典
    countDict = {}                                  #创建存储频数的字典 k:值,v:计数
    for item in data_list:                          #统计列表 data_list 中的值的个数
        countDict[item] = countDict.get(item,0) + 1
    return countDict
def draw_freq(freq_dict):                           #绘制值计数的字典 freq_dict 的直方图
    itemList = list(freq_dict.keys())               #获取键的列表
    maxItem = len(itemList) - 1                     #获取最大项目数
    itemList.sort()                                 #对键的列表排序
    countList = freq_dict.values()                  #获取计数的列表
    maxCount = max(countList)                       #获取最大的计数值
    #使用海龟对象,绘制直方图
    wn = turtle.Screen()                            #创建海龟绘图窗口
    t = turtle.Turtle()                             #创建海龟对象
    wn.setworldcoordinates(-1, -1, maxItem + 1, maxCount + 1)     #设置绘图窗口大小
    t.hideturtle()                                  #隐藏海龟
    t.up();t.goto(0, 0);t.down()                    #绘制基准线(X轴)
    t.goto(maxItem, 0); t.up()
    for i in range(0,maxCount+1):                   #绘制Y轴标签
        t.goto(-1, i)
        t.write(str(i), font=("Helvetica", 16, "bold"))
    for index in range(len(itemList)):
        t.goto(index, -1)                           #绘制标签
        t.write(str(itemList[index]),font=("Helvetica",16,"bold"))
        t.goto(index, 0)                            #绘制计数线条(高度)
```

```
            t.down()
            t.goto(index, freq_dict[itemList[index]])
            t.up()
        wn.exitonclick()                          #单击鼠标关闭绘图窗口
data = [3,1,2,1,3,1,2,2,3,5,3,5,4,5,3,4,5,2,3,3,2,2,3,4,2,5,4,3]
freq_dict = freq_table(data)                      #返回频数字典
#打印结果
itemList = list(freq_dict.keys())                 #获取键的列表
itemList.sort()                                   #对键的列表排序
print("值", "计数")
for item in itemList:                             #打印频数值及计数
    print(item, " ", freq_dict[item])
draw_freq(freq_dict)                              #绘制频数表所对应的直方图
```



值	计数
1	3
2	7
3	9
4	4
5	5

图 5-8　给定列表数据频数表的直方图

5.10　习题

扫描下方二维码，下载习题电子版。

扫一扫

习题

5.11　上机实践

1. 完成本章中的例 5.1～例 5.46，熟悉 Python 语言中序列数据类型(list、str、tuple、bytes、和 bytearray)以及字典和集合的运算和操作。

2. 编写程序，统计输入的字符串中英文字母、数字、空格和其他字符出现的次数。程序运行效果如图 5-9 所示。

```
请输入字符串：This is  a test. 123   45678!!! End?
所有字母的总数为:    37
英文字母出现的次数:  14
数字出现的次数:     8
空格出现的次数:     10
其他字符出现的次数:  5
```

图 5-9　统计字符的运行效果

提示：本实践题使用表 5-16 所示的字符串对象的方法、str 类方法确定字符/字符串是否为字母、数字、空格等。

表 5-16　本实践题所使用的字符串对象的方法/str 类方法

方　法	功　能
isalpha	判断字符/字符串是否全为字母
isdigit	判断字符/字符串是否全为数字(0～9)
isspace	判断字符/字符串是否只包含空白字

3. 编写程序，统计输入的字符串中单词的个数，单词之间用空格分隔。程序运行效果如图 5-10 所示。

4. 编写程序，实现删除一个列表中重复元素的功能。

提示：可以利用 s.append(x)方法把对象 x 追加到列表 s 尾部。

5. 编写程序，求列表 s=[9,7,8,3,2,1,55,6]中的元素个数、最大值、最小值、元素之和、平均值。请思考，有哪几种实现方法？

提示：用户可以分别利用 for 循环、while 循环、直接访问列表元素(for i in s…)、间接访问列表元素(for i in range(0,len(s))…)、正序访问(i=0；while i<len(s)…)、反序访问(i=len(s)-1；while i>=0…)以及 while True：…break 等各种方法。

6. 编写程序，将列表 s=[9,7,8,3,2,1,5,6]中的偶数变成它的平方，奇数保持不变。程序运行效果如图 5-11 所示。

```
请输入字符串: The quick brown fox jumps over the lazy dog.
其中的单词总数有:  9
```

图 5-10　统计单词个数的运行效果

```
变换前,s= [9, 7, 8, 3, 2, 1, 5, 6]
变换后,s= [9, 7, 64, 3, 4, 1, 5, 36]
```

图 5-11　奇数偶数运行效果

提示：用户可以利用“if (s[i] % 2) ==0：…”的语句形式判断列表中的第 i 个元素是否为偶数。

7. 编写程序，输入字符串，为其每个字符的 ASCII 码形成列表并输出。程序运行效果如图 5-12 所示。

提示：

(1) 使用 ord(s[i])将字符转换为对应的 Unicode 码。

(2) 利用 s.append(x)方法将对象 x 追加到列表 s 尾部。

8. 编写程序，创建由'Monday'～'Sunday'七个值组成的字典，输出键列表、值列表以及键值列表。程序运行效果参见图 5-13 所示。

```
请输入一个字符串:ABCDE123
[65, 66, 67, 68, 69, 49, 50, 51]
```

图 5-12　ASCII 码列表运行效果

```
1 2 3 4 5 6 7

Mon Tues Wed Thur Fri Sat Sun

(1, 'Mon') (2, 'Tues') (3, 'Wed') (4, 'Thur') (5, 'Fri') (6, 'Sat') (7, 'Sun')
```

图 5-13　字典运行效果

9. 假设有一个存放学生学号和语数英三门功课成绩的列表 studs 如下：

studs=[{'sid': '103', 'Chinese': 90, 'Math': 95, 'English': 92},{'sid': '101', 'Chinese': 80, 'Math': 85, 'English': 82}, {'sid': '102', 'Chinese': 70, 'Math': 75, 'English': 72}]。编写程序，将列表 studs 的数据内容提取出来，放到一个字典 scores 中，在屏幕上按学号从小到大的顺序显示输出所有学生的学号及语数英三门功课的成绩。程序运行效果如图 5-14 所示。

10. 编写程序，随机生成 10 个 0(含)～10(含)的整数，分别组成集合 A 和集合 B，输出

A 和 B 的内容、长度、最大值、最小值以及它们的并集、交集和差集。程序运行效果参见图 5-15 所示。

```
101:[80, 85, 82]
102:[70, 75, 72]
103:[90, 95, 92]
```

图 5-14　按学号顺序输出成绩

```
集合的内容、长度、最大值、最小值分别为：
{0, 8, 10, 5, 7} 5 10 0
{9, 2, 10, 5, 6} 5 10 2
A和B的并集、交集和差集分别为：
{0, 2, 5, 6, 7, 8, 9, 10} {10, 5} {0, 8, 7}
```

图 5-15　集合运行效果

11. 编写程序，判断一个字符串是否为回文字符串。回文字符串是指字符串的顺序和逆序内容完全相同。例如，字符串 abcba 就是一个回文字符串。程序运行效果如图 5-16 所示。要求使用至少“字符串逆序切片方法[::-1]”“for 循环遍历并逐字符比较”“将字符串转换为 list 后再利用列表的 reverse()函数处理”“使用字符串的 reversed()函数将字符串反序”以及“递归”等 5 种方法实现回文字符串的判断。为了简单起见，要求字符串中的字符连续出现，即当中无任何诸如空格、标点符号等分隔符，并且大小写一致。

```
输入字符串：abcba
方法一：abcba是回文字符串
方法二：abcba是回文字符串
方法三：abcba是回文字符串
方法四：abcba是回文字符串
方法五：abcba是回文字符串
```

图 5-16　判断回文字符串的运行效果

说明：

英语中有如下几个比较有趣的回文。

- Able was I ere I saw Elba。（在我看到厄尔巴岛之前，我曾所向无敌。——据称是拿破仑被流放到厄尔巴岛时说的一句话（虽然真实性值得怀疑）。这是英语中最著名的一个回文）。
- Madam, I' m Adam。（女士，我是 Adam）
- Was it a bar or a bat I saw?（我看到的是酒吧还是蝙蝠？）
- Was it a cat I saw?（我看到的是猫吗？）

汉语中也有很多经典的回文句，例如：

- 上海自来水来自海上。
- 黄山落叶松叶落山黄。
- 西湖回游鱼游回湖西。
- 清水池里池水清；静泉山上山泉静。

5.12　案例研究：猜单词游戏

本章案例研究通过一个简单的游戏案例帮助读者使用数据结构和算法实现基本的游戏人工智能，从而加深了解 Python 数据结构和基本算法流程。

“猜单词游戏”使用元组或者列表构建待猜测的英文单词库列表 WORDS，使用 random 模块的 choice()函数从单词的元组中随机抽取一个英文单词 word；然后把该英文单词的字母乱序排列（方法是每次随机抽取一个位置的字符放入乱序的 jumble 字符串中，并从原 word 中删除该字符）。

游戏一开始先显示乱序后的字符串 jumble，并提示用户输入猜测的结果，如果错误，提示继续输入，直至输入正确。猜对之后，可以询问是否继续游戏。游戏也可以通过按 Ctrl+C 组合键强制中断运行。

本章案例研究的解题思路和源代码等以电子版形式提供，具体请扫描右侧二维码。

扫一扫

案例研究