第5章 系列数据类型

系列数据类型（bytes、bytearray、list、str 和 tuple）是 Python 内置的组合数据类型，可以实现复杂数据的处理。

5.1 Python 系列数据概述

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，可实现复杂数据处理任务。

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系列数据类型（例如列表 list），实现数组的功能。

5.1.2 系列数据类型

系列数据类型是 Python 基础的数据结构，是一组有顺序的元素的集合。系列数据可以包含一个或多个元素（对象，元素也可以是其他系列数据），也可以是一个没有任何元素的空系列。

Python 内置的系列数据类型包括：元组（tuple）、列表（list）、字符串（str）和字节数据（bytes 和 bytearray）。

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
```

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]
```

字符串是包括若干字符的系列数据，支持系列数据的基本操作。例如：

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

字节系列数据是包括若干字节的序列。Python 抓取网页时返回的页面，通常为 UTF-8 编码的字节序列。字节系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 系列数据的基本操作

5.2.1 系列的长度、最大值、最小值、求和

通过内置函数 len()、max()、min()，可以获取系列的长度、系列中元素最大值、系列中元素最小值。内置函数 sum()可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】 系列数据的求和示例。

```
>>> 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】 系列的长度、最大值、最小值操作示例。

```
>>> s='abcdefg'
>>> len(s)
7
>>> max(s)
'g'
>>> min(s)
'a'
>>> s2=''
>>> len(s2)
0
```

```
>>> t=(10,2,3)
>>> len(t)
3
>>> max(t)
10
>>> min(t)
2
>>> t2=()
>>> len(t2)
0
```

```
>>> lst=[1,2,9,5,4]
>>> len(lst)
5
>>> max(lst)
9
>>> min(lst)
1
>>> lst2=[ ]
>>> len(lst2)
0
```

```
>>> b=b'ABCD'
>>> len(b)
4
>>> max(b)
68
>>> min(b)
65
>>> b2=b''
>>> len(b2)
0
```

5.2.2 系列的索引访问操作

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

```
s[i]          #访问系列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]，依此类推，最后一个元素为 s[len(s) – 1]。

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[3]        #IndexError: string index out of range
>>> s['a']      #TypeError: string indices must be integers
```

系列 s 的索引下标示意图如图 5-1 所示。

s[−5]	s[−4]	s[−3]	s[−2]	s[−1]
'bonus'	−228	'purple'	'100'	19.84
s[0]	s[1]	s[2]	s[3]	s[4]

图 5-1　系列 s 的索引下标示意图

【例 5.3】 系列的索引访问示例。

>>> s='abcdef'	>>> t=('a','e','i','o','u')	>>> lst=[1,2,3,4,5]	>>> b=b'ABCDEF'
>>> s[0]	>>> t[0]	>>> lst[0]	>>> b[0]
'a'	'a'	1	65
>>> s[2]	>>> t[1]	>>> lst	>>> b[1]
'c'	'e'	[1, 2, 3, 4, 5]	66
>>> s[-1]	>>> t[-1]	>>> lst[2]='a'	>>> b[-1]
'f'	'u'	>>> lst[-2]='b'	70
>>> s[-3]	>>> t[-5]	>>> lst	>>> b[-2]
'd'	'a'	[1, 2, 'a', 'b', 5]	69

5.2.3　系列的切片操作

通过切片操作，可以截取系列 s 的一部分。切片操作的基本形式为：

```
s[i:j]  或者  s[i:j:k]
```

其中，i 为系列开始下标（包含 s[i]）；j 为系列结束下标（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，则直到系列结束为止；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】 系列的切片操作示例。

>>> s='abcdef'	>>> t=('a','e','i','o','u')	>>> lst=[1,2,3,4,5]	>>> b=b'ABCDEF'
>>> s[1:3]	>>> t[–2:–1]	>>> lst[:2]	>>> b[2:2]
'bc'	('o',)	[1, 2]	b''

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

```
>>> 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]=[]
>>> lst
[2, 3, 4, 5]
>>> lst[:2]
[2, 3]
>>> lst[:2]='a'
>>> lst[1:]='b'
>>> lst
['a', 'b']
>>> del lst[:1]
>>> lst
['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 系列的连接和重复操作

通过连接操作符+，可以连接两个系列（s1 和 s2），形成一个新的系列对象；通过重复操作符*，可以重复一个系列 n 次（n 为正整数）。系列连接和重复操作的基本形式为：

```
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】 系列的连接和重复操作示例。

```
>>> s1='abc'
>>> s2='xyz'
>>> s1+s2
'abcxyz'
>>> s1*3
'abcabcabc'
>>> s1 += s2
>>> s1
'abcxyz'
>>> s2 *= 2
>>> s2
'xyzxyz'
```

```
>>> t1=(1,2)
>>> t2=('a','b')
>>> t1+t2
(1, 2, 'a', 'b')
>>> t1*2
(1, 2, 1, 2)
>>> t1 += t2
>>> t1
(1, 2, 'a', 'b')
>>> t2 *= 2
>>> t2
('a', 'b', 'a', 'b')
```

```
>>> lst1=[1,2]
>>> lst2=['a','b']
>>> lst1+ lst2
[1, 2, 'a', 'b']
>>> 2 * lst2
['a', 'b', 'a', 'b']
>>> lst1 += lst2
>>> lst1
[1, 2, 'a', 'b']
>>> lst2 *=2
>>> lst2
['a', 'b', 'a', 'b']
```

```
>>> b1=b'ABC'
>>> b2=b'XYZ'
>>> b1+b2
b'ABCXYZ'
>>> b1*3
b'ABCABCABC'
>>> b1+=b2
>>> b1
b'ABCXYZ'
>>> b2*=2
>>> b2
b'XYZXYZ'
```

5.2.5 系列的成员关系操作

可以通过下列方式之一判断一个元素 x 是否存在于系列 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)              #返回x在s（指定范围[start,end)）中出现的次数
```

```
s.index(x[, i[, j]])     #返回x在s（指定范围[i, j)）中第一次出现的下标
```

其中，指定范围[i, j)，从下标 i（包括，默认为 0）开始，到下标 j 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】 系列中元素的存在性判断示例。

```
>>> s='Good, better, best!'
>>> 'o' in s
True
>>> 'g' not in s
True
>>> s.count('e')
3
>>> s.index('e', 10)
10
```

```
>>> t=('r', 'g', 'b')
>>> 'r' in t
True
>>> 'y' not in t
True
>>> t.count('r')
1
>>> t.index('g')
1
```

```
>>> lst=[1,2,3,2,1]
>>> 1 in lst
True
>>> 2 not in lst
False
>>> lst.count(1)
2
>>> lst.index(3)
2
```

```
>>> b=b'Oh, Jesus!'
>>> b'O' in b
True
>>> b'o' not in b
True
>>> b.count(b's')
2
>>> b.index(b's')
6
```

5.2.6 系列的比较运算操作

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

【例 5.7】 系列的比较运算示例。

```
>>> s1='abc'
>>> s2='abc'
>>> s3='abcd'
>>> s4='cba'
>>> s1 > s4
False
>>> s2 <= s3
True
>>> s1 == s2
True
>>> s1 != s3
True
>>> 'a' > 'A'
True
>>> 'a' >= ''
```

```
>>> t1=(1,2)
>>> t2=(1,2)
>>> t3=(1,2,3)
>>> t4=(2,1)
>>> t1<t4
True
>>> t1 <= t2
True
>>> t1 == t3
False
>>> t1 != t2
False
>>> t1 >= t3
False
>>> t4 > t3
```

```
>>> s1=['a','b']
>>> s2=['a','b']
>>> s3=['a','b','c']
>>> s4=['c','b','a']
>>> s1<s2
False
>>> s1<=s2
True
>>> s1==s2
True
>>> s1!=s3
True
>>> s1>=s3
False
>>> s4>s3
```

```
>>> b1=b'abc'
>>> b2=b'abc'
>>> b3=b'abcd'
>>> b4=b'ABCD'
>>> b1<b2
False
>>> b1<=b2
True
>>> b1==b2
True
>>> b1>=b3
False
>>> b3!=b4
True
>>> b4>b3
```

True	True	True	False

5.2.7 系列的排序操作

通过内置函数 sorted()，可以返回系列的排序列表。通过类 reversed 构造函数，可以返回系列的反序的迭代器。内置函数 sorted()形式如下：

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

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】 系列的排序操作示例。

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

5.2.8 内置函数 all()和 any()

通过内置函数 all()和 any()，可以判断系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))
True
>>> all([1, 2, 0])
False
```

5.2.9 系列拆封

1．变量个数和系列长度相等

使用赋值语句，可以将系列值拆封，然后赋值给多个变量：

变量1，变量2，…，变量n = 系列或可迭代对象

变量个数和系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: need more than 2 values to unpack
>>> 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．变量个数和系列长度不等

如果系列长度未知，可使用***元组变量**，将多个值作为元组赋值给元组变量。一个赋值语句中，***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，系列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 元　　组

元组是一组有序系列，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。

5.3.1 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。其基本形式如下：

x1，[x2，…，xn]　或者　(x1，[x2，…，xn])

其中，x1、x2、…、xn 为任意对象。

【例 5.9】 使用元组字面量创建元组实例对象示例。

```
>>> t1=1,2,3
>>> t2=()
>>> t3=1,
>>> i1=(1)
>>> t4='a','b','c'
>>> t5=2.0,
>>>print(t1,t2,t3,t4,t5,i1)
(1, 2, 3) () (1,) ('a', 'b', 'c') (2.0,) 1
```

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。

5.3.2 使用 tuple 对象创建元组实例对象

也可以通过创建 tuple 对象来创建元组。其基本形式为：

```
tuple()                 #创建一个空列表
tuple(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 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)
() ('a', 'b', 'c') (1, 2, 3) (0, 1, 2)
```

5.3.3 元组的系列操作

元组支持系列的基本操作，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。

【例 5.11】 元组的系列操作示例。

```
>>> 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 列　　表

列表是一组有序项目的数据结构。创建一个列表后，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。Python 没有数组，可以使用列表代替。

5.4.1 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。其基本形式为：

```
[x1, [x2, …, xn]]
```

【例 5.12】 使用列表字面量创建列表实例对象示例。

```
>>> l1=[ ]
>>> l2=[1]
>>> l3=["a","b","c"]
>>> print(l1,l2,l3)          #输出：[] [1] ['a', 'b', 'c']
```

5.4.2 使用 list 对象创建元组实例对象

也可以通过创建 list 对象来创建列表。其基本形式为：

```
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 5.13】 使用 list 对象创建列表实例对象示例。

```
>>> l1=list()
>>> l2=list("abc")
>>> l3=list(range(3))
>>> print(l1,l2,l3)        #输出: [] ['a', 'b', 'c'] [0, 1, 2]
```

5.4.3 列表的系列操作

列表支持系列的基本操作，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。

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。

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

列表是可变对象，故可以改变其切片的值，也可以通过 del 删除切片。

```
s[i:j] =x        #设置列表内容，x为任意对象，也可以是元组、列表
del s[i:j]       #移去列表一系列元素，等同于s[i:j] =[ ]
s[i:j] =[]       #移去列表一系列元素
```

【例 5.14】 列表的系列操作示例。

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

5.4.4 list 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 s=[1,3,2]。

表 5-1 列表对象的主要方法

方　法	说　明	示　例
s.append(x)	把对象 x 追加到列表 s 尾部	s.append('a')　#s= [1, 3, 2, 'a'] s.append([1,2])　#s= [1, 3, 2, 'a', [1, 2]]
s.clear()	删除所有元素。相当于 del s[:]	s.clear()　#s= []
s.copy()	复制列表	s1=s.copy()　#s1= s=[1,3,2] id(s),id(s1)　#(43280824, 42613096)

续表

方　法	说　明	示　例
s.extend(t)	把系列 t 附加到 s 尾部	s.extend([4])　#s= [1, 3, 2, 4] s.extend('ab')　#s= [1, 3, 2, 4, 'a', 'b']
s.insert(i, x)	在下标 i 位置插入对象 x	s.insert(1,4)　#s= [1, 4, 3, 2] s.insert(8,5)　#s= [1, 4, 3, 2, 5]
s.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	s.pop()　　#输出 2。s= [1, 3] s.pop(0)　　#输出 1。s=[3]
s.remove(x)	移除列表中第一个出现的 x。若对象不存在，将导致 ValueError	s.remove(1)　#s= [3, 2] s.remove('a') #ValueError: list.remove(x): x not in list
s.reverse()	列表反转	s.reverse()　#s=[2, 3, 1]
s.sort()	列表排序	s.sort()　　#s=[1, 2, 3]

5.4.5　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 5.15】 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)]
```

5.5　字　符　串

字符串实现为有序的字符集合，即字符系列。str 对象是不可变对象。

5.5.1　字符串的系列操作

字符串支持系列的基本操作，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。

通过 len(s)，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 5.16】 字符串的系列操作示例。

```
>>> s1='abcxyz'
>>> len(s1)
6
>>> s1[3:]
'xyz'
>>> s2='123'
>>> s1>s2
True
>>> 3*s2
'123123123'
>>> max(s1)
'z'
```

5.5.2 字符串编码

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
```

对应地，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 5.17】 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
```

5.5.3 字符串格式化

1．%元算符形式

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': 2}
'Python has 002 quote types.'
>>> '%0*.*f' % (10, 5, 88)  #输出：'0088.00000'
```

格式字符串的标志符（flags）如下。

（1）'0'：数值类型格式化结果左边用零填充。

（2）'-'：结果左对齐。

（3）' '：对于正值，结果中将包括一个前导空格。

（4）'+'：数值结果总是包括一个符号('+'或'－')。

（5）'#'：使用另一种转换方式。

格式化类型字符（type）如下。

（1）%d 或%i：有符号整数（十进制）。

（2）%o：有符号整数（八进制）。

（3）%u：同%d，已过时。

（4）%x：有符号整数（十六进制，小写字符），标志符为'#'时，输出前缀'0x'。

（5）%X：有符号整数（十六进制，大写字符），标志符为'#'时，输出前缀'0X'。

（6）%e：浮点数字（科学记数法，小写 e），标志符为'#'时，总是带小数点。

（7）%E：浮点数字（科学记数法，大写 E），标志符为'#'时，总是带小数点。

（8）%f 或%F：浮点数字（用小数点符号），标志符为'#'时，总是带小数点。

（9）%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。

（10）%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。

（11）%c：字符及其 ASCII 码。

（12）%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。

（13）%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。

（14）%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。

（15）%%：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。

（1）b：二进制数。

（2）c：字符，整数转换为对应的 Unicode。

（3）d：十进制数。

（4）o：八进制数。

（5）x：十六进制数，小写字符，标志符为'#'时，输出前缀'0x'。

（6）X：十六进制数，大写字符，标志符为'#'时，输出前缀'0X'。

（7）e：浮点数字(科学记数法，小写 e)，标志符为'#'时，总是带小数点。

（8）E：浮点数字(科学记数法，大写 E)，标志符为'#'时，总是带小数点。

（9）f 或 F：浮点数字(用小数点符号)，标志符为'#'时，总是带小数点。

（10）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。

（11）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。

（12）n：数值，使用本地千位分隔符。

（13）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。

（14）%：百分比。

例如：

```
>>> format(81.2, "0.5f")    #输出：'81.20000'
>>> format(81.2, "%")       #输出：'8120.000000%'
```

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'
```

5.6 字 节 系 列

字节系列是由 8 位字节数据组成的系列数据类型，即 0≤x<256 的整数系列。Python 内置的字节系列数据类型包括：bytes（不可变对象）、bytearray（可变对象）和 memoryview。

5.6.1 bytes 常量

使用字母 b 加单引号或双引号括起来的内容，是 bytes 常量。Python 解释器自动创建 bytes 型对象实例。bytes 常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，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 5.18】 bytes 常量示例。

```
>>> 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为：

```
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，复制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，则导致 ValueError。

【例 5.19】 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为：

```
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，复制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 5.20】 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 的系列操作

bytes 和 bytearray 支持系列的基本操作，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，以及求系列长度、最大值、最小值等。

bytes 和 bytearray 一般基于 ASCII 字符串，故 bytes 和 bytearray 基本上支持 str 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，否则导致 TypeError。

【例 5.21】 字节的系列操作示例。

```
>>> b1=b"abc"
>>> b1.replace(b'a',b'f')   #输出：b'fbc'
>>> b1.replace('b','g')
        #TypeError: expected bytes, bytearray or buffer compatible object
```

5.6.5 字节编码和解码

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。

【例 5.22】 字节编码和解码示例。

```
>>> s='好好学习'
>>> b=s.encode()
>>> b            #输出：b'\xe5\xa4\xa9\xe4\xbd\x93\xe5\x90\x91\xe4\xb8\x8a'
>>> b.decode() #输出：'好好学习'
```

复 习 题

一、单选题

1．Python 语句 print(type([1, 2, 3, 4]))的输出结果是_____。

A．<class 'tuple'>　B．<class 'dict'>　C．<class 'set'>　D．<class 'list'>

2．Python 语句 print(type((1, 2, 3, 4)))的结果是_____。

A．<class 'tuple'>　B．<class 'dict'>　C．<class 'set'>　D．<class 'list'>

3．Python 语句 print(type({1, 2, 3, 4}))的输出结果是_____。

A．<class 'tuple'>　B．<class 'dict'>　C．<class 'set'>　D．<class 'list'>

4．Python 语句 a = [1,2,3,None,(),[],];print(len(a))的输出结果是_____。

A．4　　B．5　　C．6　　D．7

5．Python 语句 nums = set([1,2,2,3,3,3,4]);print(len(nums))的输出结果是_____。

A．1　　B．2　　C．4　　D．7

6．Python 语句 s='hello';print(s[1:3])的运行结果是_____。

A．hel　　B．he　　C．ell　　D．el

7．Python 语句 s1=[4,5,6];s2=s1;s1[1]=0;print(s2)的运行结果是_____。

A．[4,5,6]　　B．[0,5,6]　　C．[4,0,6]　　D．以上都不对

8．Python 语句 d={1:'a',2:'b',3:'c'}; print(len(d))的运行结果是_____。

A．0　　B．1　　C．3　　D．6

9．Python 语句 a = [1,2,3,None,(),[],]; print(len(a))的运行结果是_____。

A．语法错　　B．4　　C．5　　D．6

10．Python 语句 print('\x48\x41!')的运行结果是_____。

A．'\x48\x41!'　　B．4841!　　C．4841　　D．HA!

11．Python 语句 s={'a',1,'b',2};print(s['b'])的运行结果是_____。

A．语法错　　B．'b'　　C．1　　D．2

12．Python 语句 print(r"\nGood")的运行结果是_____。

A．新行和字符串 Good　　B．r"\nGood"

C．\nGood　　D．字符 r、新行和字符串 Good

二、填空题

1．Python 语句 fruits=['apple','banana','pear'];print(fruits[-1][-1])的结果是_____。

2．Python 语句 fruits=['apple','banana','pear'];print(fruits.index('apple'))的结果是_____。

3．Python 语句 fruits=['apple','banana','pear'];print('Apple' in fruits)的结果是_____。

4．Python 语句 print(sum(range(10)))的结果是_____。

5．Python 语句 print('%d%%%d' %(3/2, 3%2))的结果是_____。

6．Python 语句 s = [1, 2, 3, 4];s.append([5,6]);print(len(s))的运行结果是_____。

7．Python 语句 s1=[1,2,3,4];s2=[5,6,7];print(len(s1+s2))的运行结果是_____。

8．Python 语句 print(tuple(range(2)), list(range(2)))的运行结果是_____。

9．Python 语句 print(tuple([1,2,3]), list([1,2,3]))的运行结果是_____。

10．Python 列表解析表达式[i for i in range(5) if i%2!=0]和[i**2 for i in range(3)]的值分别为_____。

11．Python 语句 first, *middles, last = range(6)执行后，middles 的值为_____；first, second, third, *lasts = range(6)执行后，lasts 的值为_____；*firsts, last3, last2, last1 = range(6)执行后，firsts 的值为_____；first, *middles, last = sorted([86, 85, 99, 88, 60, 95, 96])执行后，sum(middles)/len(middles)的值为_____。

12．在 Python 中，设有 s=('a','b','c','d','e')，则 s[2]值为_____；s[2:4]值为_____；s[:3]值为_____；s[3:]值为_____；s[1::2]值为_____；s[-2]值为_____；s[::-1]值为_____；s[-2:-1]值为_____；s[-2:]值为_____；s[-99:-5]值为_____；s[-99:-3]值为_____；s[::]值为_____；s[1:-1]值为_____。

13．在 Python 中，设有 s=[1,2,3,4,5,6]，则 max(s)值为_____；min(s)值为_____；语句序列“s[:1]=[];s[:2]='a';s[2:]='b';s[2:3]=['x', 'y'];del s[:1]”执行后，s 值为_____。

14．在 Python 中，设有 s=['a', 'b']，则语句序列“s.append([1,2]); s.extend('34'); s.extend([5,6]); s.insert(1,7); s.insert(10,8); s.pop(); s.remove('b'); s[3:]=[]; s.reverse()”执行后，s 值为_____。

三、思考题

1．Python 中如何实现 tuple 和 list 的转换？

2．阅读下面的 Python 语句，请问输出结果是什么？

```
n = int(input("请输入图形的行数： "))
for i in range(n,0,-1):
       print(" ".rjust(20-i),end='')
       for j in range(2 * i-1):print("*",end='')
       print("\n")
for i in range( 1, n):
   print(" ".rjust(19 - i),end='')
```

```
    for j in range(2 * i+1):print("*",end='')
    print("\n")
```

3．阅读下面的 Python 语句，请问输出结果是什么？

```
n = int(input("请输入上(或下)三角的行数： "))
for i in range(0,n):
       print(" ".rjust(19-i),end='')
       for j in range(2 * i+1):print("*",end='')
       print("\n")
for i in range(n-1, 0,-1):
    print(" ".rjust(20 - i),end='')
    for j in range(2 * i-1):print("*",end='')
    print("\n")
```

4．阅读下面的 Python 语句，请问输出结果是什么？

```
daysOfWeek = ['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday',
'Sunday']
months = ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct',
'Nov','Dec']
print("DAYS: %s, MONTHS %s" % (daysOfWeek, months))
```

5．阅读下面的 Python 语句，请问输出结果是什么？

```
names1 = ['Amy', 'Bob', 'Charlie', 'Daling']
names2 = names1; names3 = names1[:]
names2[0] = 'Alice';names3[1] = 'Ben'
sum = 0
for ls in (names1, names2, names3):
    if ls[0] == 'Alice': sum += 1
    if ls[1] == 'Ben': sum += 2
print(sum)
```

上 机 实 践

1．统计输入的字符串中单词的个数，单词之间用空格分隔。运行效果如图 5-2 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 5-2　统计单词运行效果

2．编写程序，实现删除一个 list 里面的重复元素。

提示：

可以利用 s.append(x)方法把对象 x 追加到列表 s 尾部。

3．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：

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。

4．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。

提示：

可以利用“if (s[i] % 2) ==0:…”的语句形式判断列表中第 i 个元素是否偶数。

5．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，运行效果如图 5-3 所示。

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

图 5-3　ASCII 码列表运行效果

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。