第3章 复合数据类型

本章学习目标

(1) 掌握序列数据类型的通用方法；

(2) 掌握列表、元组和字典的特点和常用方法；

(3) 掌握命令行解析 argparse 模块的使用，以及图片和 PDF 文件元数据的提取。

本章内容概要

前面章节介绍了 Python 的基本数据类型，但是在实际编程中，不但要处理单个数据，还要处理多个数据。Python 中的列表、元组和字典能够实现对多数据的存储和处理。由于这三种常用的数据结构各有特色，可应用于不同的场景中，因此，本章对这几种数据结构单独进行介绍。

本章首先介绍序列数据类型的通用方法；接着介绍列表、元组和字典数据结构的构建以及它们的常用方法；最后简要介绍了双端队列和堆这两种数据结构。

本章的安全专题介绍使用 argparse 模块编写用户友好的命令行接口，并介绍图片的 GPS 信息获取和存储，以及 PDF 文件元数据的提取。


观看视频

3.1 序列数据

3.1.1 序列简介

序列，序是有序，列是一系列，序列是指一种包含多项数据的数据结构，这些数据项按照某种顺序存储，每个数据项在这个结构中的位置是特定的。

Python 中常见的序列数据类型包括字符串、列表和元组等。序列分为可变序列和不可变序列。不可变序列是指数据结构一旦建立，就不能修改其中的元素，字符串和元组属于不可变序列。可变序列是指序列中的元素可以修改，列表是可变序列。

序列数据中的数据项可以通过索引进行访问。索引既可以正向也可以反向。正向递增序号，从 0 开始；反向递减序号，从－1 开始。以字符串"python"为例，其索引如表 3-1 所示。

表 3-1 字符串"python"索引示意表

反向索引	－6	－5	－4	－3	－2	－1
列表元素	p	y	t	h	o	n
正向索引	0	1	2	3	4	5

```
>>> mystring = "python"
>>> mystring[-2]
```

```
'o'
>>> mystring[2]
't'
```

3.1.2　创建列表和元组

列表(list)是一个用中括号括起来的对象序列，元组(tuple)是一个用小括号括起来的对象序列。其中的元素可以是任意类型(不要求保持一致)，如数值、字符串、列表和元组等，元素与元素之间用逗号分隔。

Python 可以使用定义直接创建列表和元组，也提供了将其他序列转换为列表和元组的方法，具体请参考 3.3.1 节和 3.4.1 节。下面介绍使用定义直接创建列表和元组。

利用[item1,item2,…]的方式创建列表，利用(item1,item2,…)的方式创建元组。例如：

```
>>> list1 = [1,3,5,7,100]
>>> list1
[1, 3, 5, 7, 100]
>>> tuple1= (2,4,6,8)
>>> tuple1
(2,4,6,8)
```

列表是可变序列，即列表中的元素可以改变。例如，修改 list1[-3]对象的内容为 9。

```
>>> list1[-3] = 9
>>> list1
[1, 3, 9, 7, 100]
```

元组和字符串是不可变序列，即不能修改元组和字符串中某个索引对应的元素，如果修改，则会抛出异常。例如，修改元组 tuple1[-3]的内容为 5，则抛出 TypeError 异常。类似地，修改字符串 mystring[2]的内容，也抛出 TypeError 异常。

```
>>> tuple1 = (2,4,6,8)
>>> tuple1[-3] = 5
TypeError: 'tuple' object does not support item assignment
>>> mystring = "python"
>>> id(mystring)
2238164675312
>>> mystring[2] ="s"
TypeError: 'str' object does not support item assignment
```

如果需要修改字符串内容，可以直接对变量重新赋值。例如，给字符串变量 mystring 重新赋值为"pyshon"。通过 id()函数查看 mystring 字符串变量的内存地址，可以看出修改前后的内存地址是相同的。

```
>>> mystring = "pyshon"
>>> mystring
"pyshon"
>>> id(mystring)
2238164675312
```

列表和元组中的对象可以是任何类型，如数值、字符串、列表或元组等。一个列表和元组中可以包含不同的数据类型。例如，列表 list2 包含整数、浮点数、字符串、元组和列表等

多种数据类型。

```
>>> list2 = [512,3.9,'p',(3,9),[3,9]]
>>> list2
[512, 3.9, 'p', (3, 9), [3, 9]]
>>> tuple2 = (512,3.9,'p',(3,9),[3,9])
>>> tuple2
(512, 3.9, 'p', (3, 9), [3, 9])
```

3.2 列表和元组通用的方法

列表和元组通用的方法和运算如表 3-2 所示。

表 3-2 列表和元组通用的方法和运算

（说明：表 3-2 中的 list 都可以替换为 tuple）

用 法	简 要 描 述
list[index]	返回列表中索引为 index 的元素
list.index(item)	返回列表中 item 元素的索引
list.count(item)	返回列表中 item 元素的个数
max(list)	返回列表中的最大项
min(list)	返回列表中的最小项
len(list)	返回列表的长度
lista+listb	返回列表 lista 和列表 listb 的拼接，lista 和 listb 并不改变
lista * n	n 个列表 lista 副本的拼接，lista 不发生改变
item in list	如果 item 是 list 中的元素在列表中，返回 True，否则返回 False


观看视频

3.2.1 通过索引访问元素

列表和元组都可以通过索引访问其中的元素，索引可以是正数也可以是负数。例如：

```
>>> list2 = [512,'p',(3,9),[3,9],3.9,"PY"]
>>> list2[3]
[3, 9]
>>> list2[-3]
[3, 9]
>>> tuple2 = (512,3.9,'p',(3,9),[3,9])
>>> tuple2[-2]
(3,9)
```

由于列表是可变序列，因此可以修改列表使其元素的值发生变化。例如，修改 list2[1] 的值为[3,6]。

```
>>> list2[1] = [3,6]                    #修改列表中的对象元素
>>> list2
[512,[3,6],(3,9),[3,9],3.9,"PY"]
```

3.2.2 slice 切片

切片（分片）是 Python 中非常灵活的方法，它是通过索引来获取序列中的某一段数据元素。

slice 语法格式：[start：end：step]

参数说明如下。

- start：切片开始的索引(默认从 0 开始)。
- end：切片结束的索引(不包含 end)(默认为序列长度)。
- step：步长,默认为 1。步长为负数,表示从右向左切片。

```
>>> list3 = [1,2,3,4,5,6,7,8,9]
>>> list3[:3]              #默认从索引 0 开始,到索引 3(不包括 3),取索引为 0、1、2 的数据对象
[1, 2, 3]
>>> list3[:6:2]            #默认从 0 开始,取索引为 0、2、4 的数据(因为步长为 2)
[1, 3, 5]
>>> list3[2:6:2]           #从索引为 2 开始,到索引为 6 结束(不包括 6),取索引为 2、4 的数据
[3, 5]
>>> list3[6:2:-2]          #从索引为 6 开始,到索引为 2 结束(不包括 2),逆序取索引为 6、4 的数据
[7, 5]
>>> list3[:]               #当开始和结束索引都缺失时,取全部的数据
[1,2,3,4,5,6,7,8,9]
>>> tuple3 = ('p','y','t','h','o','n')
>>> tuple3[1:5:3]
('y', 'o')
>>> tuple3[4:2:-1]
('o', 'h')
```

示例：利用切片判定一个字符串是否是回文。

例如,pstr1="abcddcba" 是回文,而 pstr2="abcdba" 不是回文。

```
1  #palindrom.py
2  print('请输入字符串:')
3  pstr = input()
4      if(pstr==pstr[len(pstr)::-1]):
5          print('pstr is a palindrom')
6      else:
7          print('pstr is not a palindrom')
8  print(pstr[len(pstr)::-1])
```

运行结果如下：

```
请输入字符串:
abcdcba
pstr is a palindrom
请输入字符串:
abcdcba
pstr is not a palindrom
```

3.2.3　查找与计数


观看视频

s.index(x)：返回元素 x 在列表 s 或元组 s 中第一次出现的位置,如果不存在,则抛出 ValueError 异常。

s.count(x)：返回 x 在列表 s 或元组 s 中出现的次数,如果没有则返回 0。

```
>>>> list4 = [1,2,2,3,4,2,5,6]
>>>> list4.index(6)          #返回数值 6 第一次出现的索引值
```

```
7
>>> list4.index(9)              #因为数值 9 并没有在列表中,所以抛出数据异常
ValueError: 9 is not in list
>>>> list4.count(9)
0
>>>> list4.count(2)
3
```

3.2.4 最大值、最小值和长度

使用 Python 内置的全局函数 max()、min()和 len()分别返回列表或元组中最大的元素、最小的元素和长度。注意,列表或元组的元素类型相同时才能使用以上内置函数,如果类型不同,则会抛出异常。

```
>>> tuple3 = (1,2,2,3,4,2,5,6)
>>> max(tuple3)              #返回 tuple3 中最大的元素
6
>>> min(tuple3)
1
>>> len(tuple3)              #返回 tuple3 中数据对象的个数
8
>>> tuple4 = ('1',2,2,3,4,2,5,6)
>>> max(tuple4)
TypeError: '>' not supported between instances of 'int' and 'str'
```

3.2.5 加法、乘法和成员运算

list1+list2：返回一个新列表或元组,新列表中的元素是两个列表或元组所包含的元素的总和。

list * n：返回一个新列表或元组,新列表或元组中的元素是原列表或元组元素重复 n 次。

x in list：判定列表或元组中是否存在 x 元素,返回 True 或 False。

```
>>> list5 = [1,3,5,7]
>>> list6 = [2,4,6]
>>> list5 + list6            #返回两个列表拼接的结果.注意,list5 和 list6 都没有发生变化
[1,3,5,7,2,4,6]
>>> list5 * 3
[1, 3, 5, 7, 1, 3, 5, 7, 1, 3, 5, 7]
>>> 5 in list6
False
```

3.2.6 序列封包和序列解包

序列封包(Sequence Packing)：把多个值赋给一个变量时,Python 将这些值封装成元组。

序列解包(Sequence Unpacking)：将序列(列表或元组)直接赋值给多个变量。序列中的元素依次赋值给每个变量。

```
>>> vals = 3,6,9
```

```
>> vals
(3, 6, 9)
>> type(vals)
tuple
>>> d,e,f = vals
>>> f
9
>>> list1 = [1,2,3]
>>> a,b,c = list1
>>> b
2
```

同时使用序列封包和解包，可以实现赋值运算符将多个值赋值给多个变量。例如：

```
>>> d,e,f = 2,4,6
```

上述操作等价于下面的操作步骤，先封包、后解包的过程。

```
>>> temp = 2,4,6          #封包
>>> d,e,f = temp          #解包
```

3.3　列表

除了以上操作，列表还有自身的一些特殊操作，如表 3-3 所示。

表 3-3　部分列表方法简要描述

用　　法	简 要 说 明
list.append(item)	把元素 item 添加到列表尾部
list.insert(index,item)	在列表中索引 index 之前插入 item
lst1.extend(lst2)	将列表 lst2 中元素逐个追加到 lst1 列表的尾部
list.remove(item)	移除列表中第一个出现的元素 item
list.reverse()	把列表中的元素逆序排序
list.sort()	把列表排序
list.pop()	移除列表中的最后一项
list.pop(index)	移除列表中索引 index 的项
list.copy()	复制列表

3.3.1　创建列表


观看视频

除了 3.1.2 节介绍的列表创建方法外，还可以通过内置函数 list()创建列表。例如：

```
>>> list1 = list(range(1,10,2))
>>> list1
[1, 3, 5, 7, 9]
>>> mystring ="my heart will go on"
>>> stringlist = list(mystring.split(" "))
>>> stringlist
['my', 'heart', 'will', 'go', 'on']
```

3.3.2　增加元素

列表增加元素的方式有：在末尾追加一个元素、在列表的指定位置增加一个元素、将另

一个列表追加到列表上。具体方法如下。

（1）append(item)：在列表末尾追加一个元素。注意，该方法可以接收单个值，也可以接收元组或列表，但是此时，元组或列表是作为一个整体追加到列表中，这样形成嵌套列表。

（2）insert(index,item)：在列表中索引 index 之前插入 item。

（3）extend(lista)：将一个列表 lista 中元素逐个追加到原列表的末尾。

```
>>> list2 = [1,2,3]
>>> list2.append([4,5])
>>> list2
[1, 2, 3, [4, 5]]
>>> list2.extend([4,5])
>>> list2
[1, 2, 3, [4, 5], 4, 5]
>>> list2.insert(5,9)
>>> list2
[1, 2, 3, [4, 5], 4, 9, 5]
```

3.3.3 删除元素

删除元素的方式有：按照索引删除元素、按照值删除元素、清空列表等方式。具体方法如下。

（1）del：根据索引删除列表中的一个元素或一段区间中的元素。

（2）remove(item)：根据元素本身删除列表中的某个元素，如果存在多个，则只删除第一个；如果不存在，则抛出 ValueError 异常。

（3）clear()：清空列表中所有的元素。

```
>>> list3 = [1,2,9,4,5,2,4,9]
>>> list3.remove(9)
>>> list3
[1, 2, 4, 5, 2, 4, 9]
>>> list3.remove(6)
ValueError: list.remove(x): x not in list
>>> del list3[1:3]
>>> list3
[1, 5, 2, 4, 9]
>>> del list3[2]
>>> list3
[1, 5, 4, 9]
>>> list3.clear()
>>> list3
[ ]
```

3.3.4 逆序和排序

（1）reverse()：将列表中的元素逆序。

（2）sort()：将列表中的元素排序（默认从小到大排序，如果需要从大到小排序，则需要添加参数 reverse = True）。

```
>>> list4 = [1, 3, 5, 2, 4, 6]
>>> list4.reverse()
```

```
>>> list4
[6, 4, 2, 5, 3, 1]
>>> list4.sort()
>>> list4
[1, 2, 3, 4, 5, 6]
>>> list4.sort(reverse = True)
>>> list4
[6, 5, 4, 3, 2, 1]
>>> list5 = [1, 3, 5, 2, 4, 6]
>>> sorted(list5,reverse=True)
[6, 5, 4, 3, 2, 1]
>>> list5
[1, 3, 5, 2, 4, 6]
```

从上面的代码可以看出，使用列表的 sort()方法，能够实现对列表的排序，原有列表结构发生变化；通过内置函数 sorted()排序，并没有对原有列表进行修改。下面对列表的 sort()方法和内置函数 sorted()进行简要总结。

区别 1：list.sort()是为列表定义的；内置函数 sorted()可以接受任何可迭代对象。

区别 2：list.sort()方法可以直接修改原有列表；内置函数 sorted()会从一个可迭代对象构建一个新的排序列表，原来列表不改变。

区别 3：相较于 list.sort()方法，使用内置函数 sorted()的时间效率高，但同时牺牲了空间效率。

相同点：它们实现的排序都是稳定排序[①]。

3.3.5　弹出元素

(1) pop()：将列表视为栈，实现出栈操作，即弹出列表中的最后一个元素(入栈用 append 方法)，本方法将列表弹出的元素返回。

(2) pop(0)：将列表作为队列，实现出队操作，即弹出列表中的第一个元素(入队用 append 方法)并返回。

(3) pop(i)：弹出索引为 i 的元素并返回。

```
>>> list6 = [1, 3, 5, 2, 4, 6]
>>> list6.pop()
>>> list6
[1, 3, 5, 2, 4]
>>> list6.pop(2)
>> list6
[1, 3, 2, 4]
>>> list6.pop()
4
```

3.3.6　浅拷贝和深拷贝


观看视频

list.copy()方法是一种浅拷贝方法(Shallow Copy)，在 copy 模块中的 copy.copy()也

① 稳定排序：排序前后两个相等的数相对位置不变，则是稳定排序。非稳定排序：排序前后两个相等的数相对位置发生了变化，则是不稳定排序。

是一种浅拷贝方法。如果要实现深拷贝(Deep Copy)，则可以使用 copy.deepcopy()。例如，在下面示例中，list1 列表中只有不可变序列数据项，这里是整数类型。list2 列表是 list1 的一份浅拷贝。当修改 list2 中的元素时，并没有影响到 list1 中的元素。


观看视频

```
>>> list1 = [1,2,3]
>>> list2 = list1.copy()
>>> list2[1] = 5
>>> print(list1)
[1, 2, 3]
>>> print(list2)
[1, 5, 3]
```

但是当列表元素有可变序列对象时，例如，在下面的示例中，list1 列表中有列表元素[7,8,9]。list2 列表是 list1 的一份浅拷贝。当修改 list2 中的可变序列元素时，list1 中的可变序列元素也同时“被修改”了。通过查看 list1[3][0]和 list2[3][0]的内存地址，可以看出它们实际在相同的内存位置。因此修改其中的一个，另外一个同样被修改。但是对于不可变序列元素，它们在内存中的位置是不同的，例如，id(list1[1])和 id(list2[1])是不同的内存位置，因此其中一个的修改不会影响到另一个。

```
>>> import copy
>>> list1 = [1,2,3,[7,8,9]]
>>> list2 = copy.copy(list1)
>>> list2[3][0]= 5
>>> list2[1] = 6
>>> print(list1)
[1, 2, 3, [5, 8, 9]]
>>> print(list2)
[1, 6, 3, [5, 8, 9]]
>>> print(id(list1[3][0]))
140718918981408
>>> print(id(list2[3][0]))
140718918981408
>>> print(id(list1[1]))
140718918981312
>>> print(id(list2[1]))
140718918981440
```

如果需要一份独立的拷贝，则需要使用深拷贝方法 copy.deepcopy()。例如，下面示例中，list2 列表是 list1 的一份深拷贝，当修改其中的可变序列元素[7,8,9]时，不会影响到另一个列表。通过查看 id(list1[3][0])和 id(list2[3][0])的内存地址，可以看出它们在不同的内存位置。

```
>>> import copy
>>> list1 = [1,2,3,[7,8,9]]
>>> list2 = copy.deepcopy(list1)
>>> list2[3][0]= 5
>>> list2[1] = 6
>>> list1[1] = 11
>>> print(list1)
[1, 11, 3, [7, 8, 9]]
>>> print(list2)
[1, 6, 3, [5, 8, 9]]
>>> print(id(list1[3][0]))
```

```
140718918981472
>>> print(id(list2[3][0]))
140718918981408
```

3.4　元组

相对于列表的可变，元组是一种不可变序列类型，即一旦创建不可修改。不可修改指的是不可改变元素的值，也不可增加或删除元素。除了不可修改外，可以进行其他操作，如3.2节介绍的列表和元组通用的方法。


观看视频

3.4.1　创建元组

除了使用(item1,item2,…)创建元组外，还可以使用内置函数 tuple()创建元组。

```
>>> tuple1 = (1,3,5)
>>> list1 = [2,4,6]
>>> tuple2 = tuple(list1)    #将列表转换为元组
>>> tuple2
(2,4,6)
```

读者需要注意，如果在初始化(创建)元组时只有一个元素，则必须在这个元素后面加一个逗号，否则将被视作变量。例如：

```
>>> tuple3=(1)
>>> type(tuple3)             #查看 tuple3 变量类型是整型，而不是元组类型
int
>>> tuple4=(2,)              #只有一个元素的元组，添加逗号
>>> type(tuple4)
tuple
```

3.4.2　列表和元组之间的转换

列表和元组两种类型之间可以相互转换。使用内置函数 list(元组)可以将元组转换为列表，使用内置函数 tuple(列表)可以将列表转换为元组。3.1.2节介绍元组是不可变序列，当需要修改元组中的数据时，可以先将其转换为列表，修改后再转换为元组。例如：

```
>>> tuple5 = ('p','y','t','h','o','n')
>>> list2 = list(tuple5)
>>> list2
['p', 'y', 't', 'h', 'o', 'n']
>>> list2[0] = 'P'
>>> tuple2 = tuple(list2)
>>> tuple2
('P', 'y', 't', 'h', 'o', 'n')
```

3.5　字典

如果在列表中存取了大量的数据，那么在列表中获取指定元素是一项很耗时的操作，这是因为列表是线性顺序存储的，如果查找之前已知索引，可以通过索引快速获取元素，如果

之前不知道索引，则只能通过遍历找到符合条件的元素。但是，如果数据具有唯一性，可以通过字典方式存储，可实现快速查找操作。例如，公民的身份证、学生的学号、职工的工号等。前面介绍的字符串、列表和元组都属于有序序列。本节介绍的字典属于无序序列。字典(dict)是用来表示键-值对的一种数据结构类型。部分字典方法的简要描述如表 3-4 所示。

表 3-4　部分字典方法的简要描述

用　　法	简 要 说 明
dict.pop(key)	从 dict 中移除键 key 对应的(键,值)对,并返回值
dict.popitem()	从 dict 中移除字典中的最后一对(键,值),并返回值
dict.get(key)	返回键 key 对应的值,即 dict[key]
dict.items()	返回字典 dict 中的(键,值)对
dict.keys()	返回字典 dict 的键
dict.values()	返回字典 dict 的值
d1.update(d2)	d2 中所有的(键,值)对加入 d1 中


观看视频

3.5.1　创建字典

(1) 通过 {key1：value1,key2：value2,key3：value3…}键值对创建字典。

键是唯一的，不允许重复，而值是可以重复的，因此键可以是 Python 中任意不可变数据，例如，整数、实数、复数或字符串、元组等可哈希数据，但不能使用列表、集合、字典或其他可变类型。字典通过键来计算值的位置，快速获取和它唯一对应的值。

```
>>> dict0 ={}
>>> dict1 = {'server': 'python.org', 'database': 'mysql'}
>>> dict2 = {"001":"张三","002":"李四","003":"王五"}
>>> dict3 = {"河北":"石家庄","广西":"南宁","广东":"广州"}
>>> dict4 = {['河北']:'石家庄'}        #列表由于可变,不能作为键
TypeError: unhashable type: 'list'
```

(2) 通过内置函数 dict()创建字典。

```
>>> dict4 = dict()                     #创建空字典
>>> dict4
{}
>>> dict4 = dict([('one', 1), ('two', 2), ('three', 3)])
>>> dict4
{'one': 1, 'two': 2, 'three': 3}
>>> dict5 = dict([('spring',1),('summer',2),('autumn',3),('winter',4)])
>>> dict5
{'spring': 1, 'summer': 2, 'autumn': 3, 'winter': 4}
```

对字典进行 sorted 排序时，是按照键进行排序的。例如：

```
>>> sorted(dict5)
['autumn', 'spring', 'summer', 'winter']
```

(3) 以关键参数的形式创建字典。

```
>>> dict6 = dict(name='zhang', age=28)
>>> dict6
{'name': 'zhang', 'age': 28}
```

(4) 通过 dict.fromkeys(seq[,value])函数创建一个新字典,以给定元素做字典的键,value 为字典所有键对应的初始值,不给定 value,则默认为 None。

```
>>> dict7 = dict.fromkeys(['name', 'sex','age'])
>>> dict7
{'name': None, 'sex': None, 'age': None}
```

3.5.2　访问元素

通过 dict[键]来访问字典中对应键的值,如果键值不存在,则会抛出 KeyError 异常。

```
>>> dict6['name']
'zhang'
>>> dict6['age']
28
>> dict6['sex']                          #dict6 字典中没有'sex'键
KeyError: 'sex'
>>> month = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5, 'Jun': 6, 'Jul': 7, 'Aug': 8, 'Sep':
9, 'Oct': 10,'Nov': 11, 'Dec': 12}
>>> month['Aug']
8
```

3.5.3　增加、修改元素


观看视频

d[key] = value：将 key 对应的值修改为 value,如果 key 不存在则增加新的键值对。

```
>>> dict8 ={"001":["张三",90],"002":["李四",85],"003":["王五",76]}
>>> dict8["002"] = ["赵六",80]       #修改值
>>> dict8["005"] = ["田七",79]       #增加键值对
>>> dict8
{'001': ['张三', 90], '002': ['赵六', 80], '003': ['王五', 76], '005': ['田七', 79]}
```

3.5.4　删除元素

del dict[key]：通过键来实现删除元素,如果键值不存在,则会抛出 KeyError 异常。

popitem()：返回并删除字典中的最后一对键和值。

pop(key)：key 存在就移除,并返回它的 value 值,如果字典已经为空,则抛出 KeyError 异常。

```
>>> dict6 = dict(name='Dong', sex='M',age=39)
>>> del dict6['age']
>>> dict6
{'name': 'Dong', 'sex': 'M'}
>>> del dict6['info']                    #字典 dict6 没有 'info'键
KeyError: 'info'
>>> dict6.popitem()
('sex', 'M')
>>> dict6
{'name': 'Dong'}
>>> dict6.pop('name')
'Dong'
>>> dict6
{}
>>> days = {'Mo':'Monday','Tu':'Tuesday','We':'Wednesday'}
>>> days.pop('We')
```

```
'Wednesday'
>>> days
{'Mo': 'Monday', 'Tu': 'Tuesday'}
>>> days.popitem()
('Tu', 'Tuesday')
>>> days
{'Mo': 'Monday'}
>>> days.pop('Tu')                    #字典 days 没有'Tu'键
KeyError: 'Tu'
```

3.5.5 get()方法和 items()方法

get()方法：返回指定键对应的值，该键不存在时返回 None。

items()方法：返回字典的键、值对，用元组进行封装。

```
>>> dict9 ={"河北":"石家庄","广西":"南宁","广东":"广州"}
>>> dict9.get("河北")
石家庄
>>> print(dict9.get('贵州'))
None
>>> dict9.items()
dict_items([('河北', '石家庄'), ('广西', '南宁'), ('广东', '广州')])
```

3.5.6 keys()方法和 values()方法

keys()：返回字典的键。

values()：返回字典的值。

```
>>> dict9.keys()
dict_keys(['河北', '广西', '广东'])
>>> dict9.values()
dict_values(['石家庄', '南宁', '广州'])
```

3.5.7 字典长度和字典检索

len(dict)：同列表和元组相同，返回字典中键的数量。

key in dict：测试某个特定的键是否在字典中，如果存在则返回 True，否则返回 False。

```
>>> len(dict9)
3
>>> '河北' in dict9
True
>>> '河南' in dict9
False
```

3.5.8 update()方法

d1.update(d2)：d2 中所有的键和值加入 d1 中，如果存在相同的键，则覆盖 d1 中的键和值。

```
>>> days = {'Mo':'Monday','Tu':'Tuesday','We':'Wednesday'}
>>> favorites = {'Sa':'Saturday','Su':'Sunday'}
>>> days.update(favorites)
>>> days
```

```
{'Mo': 'Monday', 'Tu': 'Tuesday', 'We': 'Wednesday', 'Sa':'Saturday','Su':'Sunday'}
```

3.6　其他数据结构


观看视频

3.6.1　双端队列

双端队列是指首尾都能进出元素的线性数据结构。因此可以当作栈(后进先出)使用，也可以当作一般的队列(先进先出)使用。collections 模块中的 deque 类模拟了双端队列的相关操作。

```
>>> from collections import deque
>>> dir(deque)
```

```
1  #stack.py
2  from collections import deque
3  stack = deque(("Alice","Bob"))
4  stack.append("Tom")
5  stack.pop()
6  print(stack)
```

运行结果如下：

```
deque(['Alice', 'Bob'])
```

```
1  #queue.py
2  from collections import deque
3  q = deque(("Alice","Bob"))
4  q.append("Tom")
5  q.popleft()
6  print(q)
```

运行结果如下：

```
deque(['Bob', 'Tom'])
```


观看视频

3.6.2　堆(优先队列)

堆是一种特殊的二叉树数据结构，它是优先队列的一种，父节点的值会大于或小于所有子节点。小根堆(Min-Heap)是其中的每一个节点都小于或等于其两个子节点的一棵二叉树。大根堆(Max-Heap)是其中的每一个节点都大于或等于其两个子节点的一棵二叉树。将最大的节点放到最靠近根节点的位置。模块 heapq 实现了堆的相关操作。

```
>>> import heapq
>>> dir( heapq)
```

```
1  #heap.py
2  from heapq import *
3  num = list(range(1,10))
```

```
4  heapify(num)            #初始化一个堆,默认是小根堆
5  num.insert(3,15)
6  heapify(num)
7  print(num)
8  heappush(num,2.8)       #将元素压入堆中
9  print(num)
10 print(heappop(num))     #从堆中弹出堆顶元素
```

运行结果如下：

```
[1, 2, 3, 7, 4, 5, 6, 15, 8, 9]
[1, 2, 3, 7, 2.8, 5, 6, 15, 8, 9, 4]
1
```

3.7 安全专题


观看视频

3.7.1 命令行参数解析模块 argparse

使用和自己动手编写安全工具，在安全攻防中是一种重要的能力。像类似 nmap 这样的安全工具都提供了友好的命令行接口，因此在用户使用 Python 编写安全工具时，最好能够支持命令行的使用。本节介绍几个命令行解析模块。Python 的内置模块 sys.argv、optparse 和 argparse 提供相关的命令行参数解析。

sys.argv 模块中，sys.argv[]用来获取命令行输入的参数（参数和参数之间空格分隔）。其中，sys.argv[0]表示脚本本身，从参数 1 开始，表示获取的参数。例如，在下面的 printArgvs.py 示例程序中，第 4 行代码将参数保存到变量 args 中，第 6～8 行代码分别输出前三个参数，第 9 行代码弹出最后一个参数 green，第 10 行代码弹出第一个参数，即脚本本身。运行结果如图 3-1 所示。

```
1  #printArgvs.py
2  import sys
3
4  args = sys.argv            #返回参数列表
5  print(args)                #输出参数列表
6  print('Script:', args[0])
7  print('First arg is :', args[1])
8  print('Second arg is :', args[2])
9  print(args.pop())          #弹出最后一个参数
10 print(args.pop(0))         #弹出第一个参数
11 print(args)                #输出当前参数列表
```

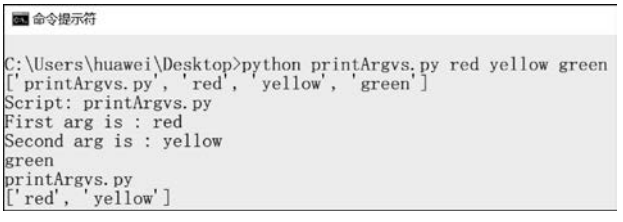

图 3-1　printArgvs.py 运行结果

getopt是C语言风格的命令行选项解析器。不熟悉C语言的getopt()函数或希望写更少代码并获得更完善帮助和错误消息的用户应考虑改用argparse模块。optparse模块相比原有getopt模块更为方便、灵活并具有强大的命令行选项解析库，但是从3.2版本开始已经废弃，代替它的是argparse模块。

通过使用argparse模块能够轻松地编写命令行接口应用。通过程序定义参数，argparse模块调用底层的sys.argv解析参数。模块还会自动生成帮助和使用手册，并在用户给程序传入无效参数时报告错误提示信息。在下面的argparseHash.py示例程序中给出了使用该模块的基本步骤。

第一步：创建一个解析器。通过argparse.ArgumentParser()创建一个ArgumentParser对象，该对象中包含将命令行解析成Python数据类型所需的全部信息。

第二步：添加参数。通过调用add_argument()方法给ArgumentParser对象添加程序的参数信息。例如，在下面的argparseHash.py示例程序中，第15、18行代码添加了两个参数信息，一个是要进行哈希的数据，另一个是采用的哈希算法。

第三步：解析参数。通过parse_args()方法解析参数，该方法一般不带参数，ArgumentParser对象将自动从sys.argv中确定命令行参数，例如，下面的argparseHash.py示例程序中第23行代码。接着把每个参数转换为适当的类型进而调用相应的操作。argparseHash.py的运行结果如图3-2所示。

```
1  #argparseHash.py
2
3  import argparse
4  import hashlib
5
6  #给出用户可选择的哈希算法列表
7  ##HASH_LIBS = ['md5', 'sha1', 'sha256', 'sha512']
8  #列出 hashlib 库中用户可选择的哈希算法列表
9  HASH_LIBS = hashlib.algorithms_available
10
11 #创建 ArgumentParser 对象
12 parser = argparse.ArgumentParser(description = '哈希计算程序')
13
14 #添加参数
15 parser.add_argument('data', help='请输入要哈希的数据')
16
17 #添加参数
18 parser.add_argument('--hash_name',
19                     help='请输入哈希函数的名称',choices=HASH_LIBS,
20                     default = 'SHA256')
21
22 #解析参数
23 args = parser.parse_args()
24
25 hs = args.hash_name
26 h = hashlib.new(hs)
```

```
27  h.update(args.data.encode('UTF-8'))
28  print(h.hexdigest().upper())
```

图 3-2　argparseHash.py 运行结果


观看视频

3.7.2　图片元数据解析模块 exifread

exifread 模块能够提取 JPG 文件的 exif 数据。Exif(Exchangeable image file,可交换图形文件)标准中包含了专门为数码相机的照片而定制的元数据,可以记录数码照片的拍摄参数、缩略图及其他属性信息,甚至全球定位信息。exifread 模块是 Python 的第三方库,具体链接为 https://pypi.org/project/ExifRead/。需要进行安装,命令为 pip install exifrread。

带有标签的照片如果放到博客、朋友圈或者 web 网站,可能会被恶意者利用。例如,下面的 jpgExif.py 示例程序中,通过提取元数据输出了图片的经纬度信息。

```
1  #jpgExif.py
2  from exifread import process_file
3  jpgFile = open("testjpg.jpg", 'rb')          # 打开图像文件
4  #process_file 查看官方文档说明,并解释
5  Tags = process_file(jpgFile)
6  print(type(Tags))                            #查看类型
7  print(Tags['Image DateTime'])                #查看图片创建时间
8  print(Tags['GPS GPSLatitude'])               #查看经度信息
9  print(Tags['GPS GPSLongitude'])              #查看纬度信息
```

运行结果如下:

```
<class 'dict'>
创建时间:2019:08:19 07:20:00
经度:[25, 18, 65619/1250]
纬度:[110, 24, 84607/2000]
```

3.7.3　PDF 文件元数据解析模块 PyPDF3

PyPDF3 是一款 PDF 文档管理工具,它可以提取文档中的内容信息,对文档按照页进行分割、合并、复制、加解密等操作。该模块是第三方模块,具体链接为 https://pypi.org/

project/PyPDF3/。需要进行安装，命令为 pip install PyPDF3。

PDF 文件中的元数据包括标题、作者姓名、创建日期、修改日期、主题、用于创建此 PDF 文件的应用程序、PDF 的文件大小、PDF 文件中的页数以及与该文件关联的所有标记等信息。使用 PyPDF3 模块可以读取 PDF 文件元数据。例如，在下面的 pdfExif.py 示例程序中，通过提取元数据输出了 PDF 文件的三种元数据信息，分别是作者、创建此 PDF 文件的应用程序和标题。假设计算机取证人员已获取多个 PDF 文件，需要检索某个特定的元数据，如作者，此时可以通过该工具实现。

```
#pdfExif.py
from PyPDF3 import PdfFileReader

# 获取 PdfFileReader 对象
pdfFileReader = PdfFileReader('meta-testpdf.pdf')
documentInfo = pdfFileReader.getDocumentInfo()
#以字典格式保存元数据信息
print('documentInfo = %s' % documentInfo)
#遍历字典,输出键值
for mentaItem in documentInfo:
    print(documentInfo[mentaItem])
```

运行结果如下：

```
documentInfo = {'/Producer': 'pypdf', '/Author': '张瑞霞', '/Title': 'Python 编程及网络安全实践'}
pypdf
张瑞霞
Python 编程及网络安全实践
```

习题

1. 编写程序，要求用户输入一个非空的整数列表，输出列表的第一个、最后一个元素，并输出列表的长度以及最大的元素。

2. 编写程序，要求用户输入一个单词列表，并依次执行以下操作。

（1）增加一个新的单词 NewWord。

（2）把列表容器中的单词正向排序并输出，要求不破坏原有列表。

（3）把列表容器中的单词反向排序并输出，要求使用列表的 sort 方法。

（4）输出 NewWord 的索引。

（5）将列表中的第一个单词移除并添加到列表的结尾。

（6）从列表容器中删除 NewWord。

（7）从列表容器中 pop 一个元素。

3. 编写程序，要求用户通过 random 模块随机产生长度分别为 3 和 5 的两个列表 list1 和 list2，并执行以下操作：

（1）合并两个列表为新的列表 list3。

（2）输出列表 list3 中每个元素出现的次数。

（3）将 list3 前面的四个元素定义为新的列表 list4。

4. 已知列表 grade 中的元素表示学生成绩的五个等级，即 A,B,C,D,E，要求输出一个新的列表 grade_count，其元素是五个等级分别出现的次数。例如，grade = ['A','C','E','A','B','C','D','B','C','D']，则 grade_count=[2,2,3,2,1]。

5. 已知列表 grade 中的元素表示学生成绩的五个等级，即 A,B,C,D,E，要求编写一系列语句，输出一个字典 grade_count，其元素的键和值分别是等级和等级出现的次数，并要求按照次数进行排序。

例如，grade = ['A','C','E','A','B','C','D','B','C','D']，则 grade_count=[('C',3),('A',2),('B',2),('D',2),('E',1)]。

6. 从网站或朋友圈中找一张带有经纬度位置信息的图片，要求使用 exifread 模块输出该图片的经、纬度信息以及图片的哈希值，并要求使用 argparse 模块定义用户的哈希类型参数选项。

7. 已知某 JPG 图片文件，请使用一行命令，输出该文件的十六进制摘要值。