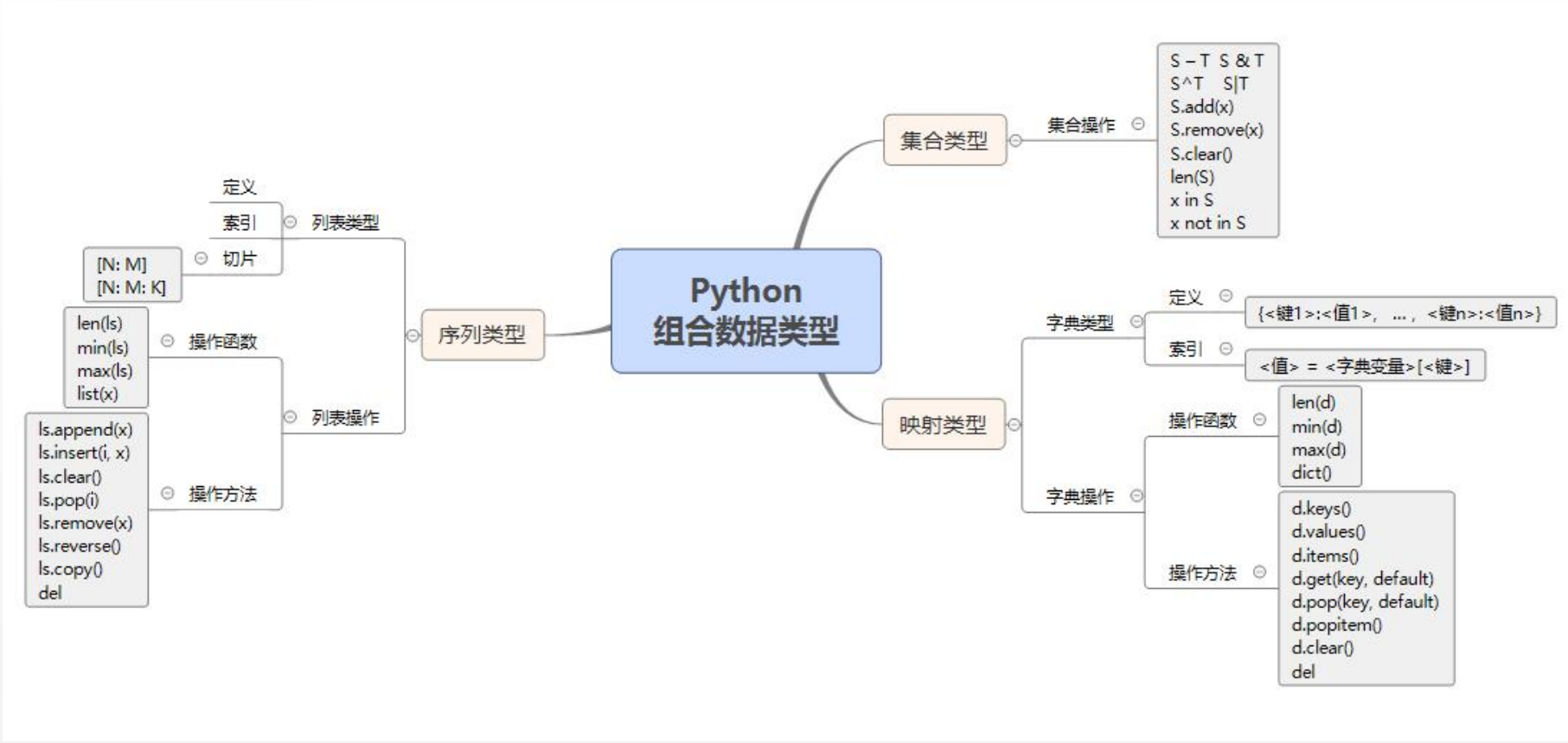
全国计算机等级考试二级教程
Python语言程序设计
(2018年版)

【第6章】
组合数据类型

考纲考点

- 组合数据类型的基本概念
- 列表类型：定义、索引、切片
- 列表类型的操作：列表的操作函数、列表的操作方法
- 字典类型：定义、索引
- 字典类型的操作：字典的操作函数、字典的操作方法

知识导图


Python
组合数据类型
序列类型
列表类型
定义
索引
切片
[N: M]
[N: M: K]
列表操作
操作函数
len(ls)
min(ls)
max(ls)
list(x)
操作方法
ls.append(x)
ls.insert(i, x)
ls.clear()
ls.pop(i)
ls.remove(x)
ls.reverse()
ls.copy()
del
集合类型
集合操作
S - T S & T
S^T S|T
S.add(x)
S.remove(x)
S.clear()
len(S)
x in S
x not in S
映射类型
字典类型
定义
{<键1>:<值1>, ... , <键n>:<值n>}
索引
<值> = <字典变量>[<键>]
字典操作
操作函数
len(d)
min(d)
max(d)
dict()
操作方法
d.keys()
d.values()
d.items()
d.get(key, default)
d.pop(key, default)
d.popitem()
d.clear()
del

组合数据类型的基本概念

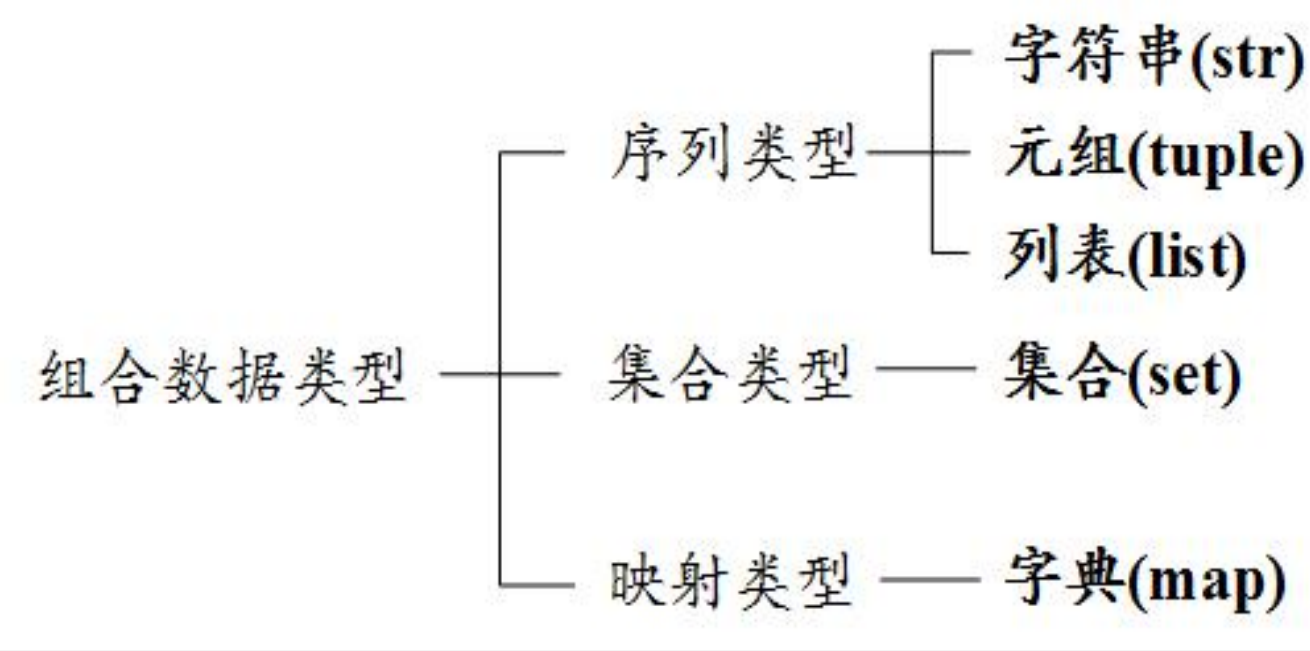
组合数据类型

- Python语言中最常用的组合数据类型有3大类，分别是**集合类型、序列类型和映射类型**。
- 集合类型是一个具体的数据类型名称，而序列类型和映射类型是一类数据类型的总称。

组合数据类型

- 集合类型是一个元素集合，元素之间无序，相同元素在集合中唯一存在。
- 序列类型是一个元素向量，元素之间存在先后关系，通过序号访问，元素之间不排他。序列类型的典型代表是字符串类型和列表类型。
- 映射类型是“键-值”数据项的组合，每个元素是一个键值对，表示为(key, value)。映射类型的典型代表是字典类型。

组合数据类型



集合类型概述

- Python语言中的集合类型与数学中的集合概念一致，即包含0个或多个数据项的无序组合。
- 集合是无序组合，用大括号（{}）表示，它没有索引和位置的概念，集合中元素可以动态增加或删除。

集合类型概述

- 集合中元素**不可重复**，元素类型只能是固定数据类型，例如：整数、浮点数、字符串、元组等，列表、字典和集合类型本身都是可变数据类型，不能作为集合的元素出现。

```
>>>S = {1010, "1010", 78.9}
>>>type(S)
<class 'set'>
>>>len(S)
3
>>>print(S)
{78.9, 1010, '1010'}
```

集合类型概述

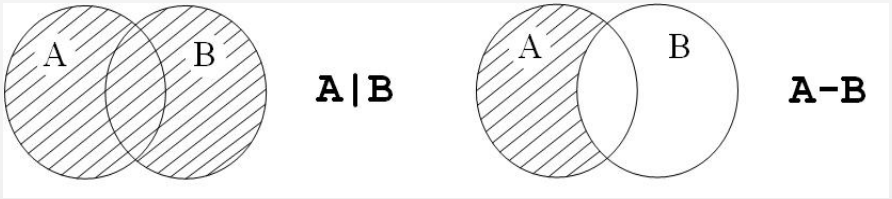
- 需要注意，由于集合元素**是无序的**，集合的打印效果与定义顺序可以不一致。由于集合元素独一无二，使用集合类型能够过滤掉重复元素。

```
>>>T = {1010, "1010", 12.3, 1010, 1010}
>>>print(T)
{1010, '1010', 12.3}
```

集合类型概述

- 集合类型有4个操作符，交集（&）、并集（|）、差集（-）、补集（^），操作逻辑与数学定义相同。

操作符的运算	描述
S – T	返回一个新集合，包括在集合S中但不在集合T中的元素
S & T	返回一个新集合，包括同时在集合S和T中的元素
S^T	返回一个新集合，包括集合S和T中非共同元素
S|T	返回一个新集合，包括集合S和T中所有元素



集合类型概述

```
>>>S = {1010, "1010", 78.9}
>>>T = {1010, "1010", 12.3, 1010, 1010}
>>>S - T
{78.9}
>>>T - S
{12.3}
>>>S & T
{1010, '1010'}
>>>T & S
{1010, '1010'}
>>>S ^ T
{78.9, 12.3}
>>>T ^ S
{78.9, 12.3}
>>>S | T
{78.9, 1010, 12.3, '1010'}
>>>T | S
{1010, 12.3, 78.9, '1010'}
```

集合类型概述

- 集合类型有一些常用的操作函数或方法

函数或方法	描述
S.add(x)	如果数据项x不在集合S中，将x增加到s
S.remove(x)	如果x在集合S中，移除该元素；不在产生KeyError异常
S.clear()	移除S中所有数据项
len(S)	返回集合S元素个数
x in S	如果x是S的元素，返回True，否则返回False
x not in S	如果x不是S的元素，返回True，否则返回False

集合类型概述

- 集合类型主要用于元素去重，适合于任何组合数据类型。

```
>>>S = set('知之为知之不知为不知')
>>>S
{'不', '为', '之', '知'}
>>>for i in S:
    print(i, end="")
不为之知
```

序列类型概述

- 序列类型是一维元素向量，元素之间存在先后关系，通过序号访问。
- 由于元素之间存在顺序关系，所以序列中可以存在相同数值但位置不同的元素。Python语言中有很多数据类型都是序列类型，其中比较重要的是：字符串类型和列表类型，此外还包括元组类型。

序列类型概述

- 字符串类型可以看成是单一字符的有序组合，属于序列类型。列表则是一个可以使用多种类型元素的序列类型。序列类型使用相同的索引体系，即正向递增序号和反向递减序号。

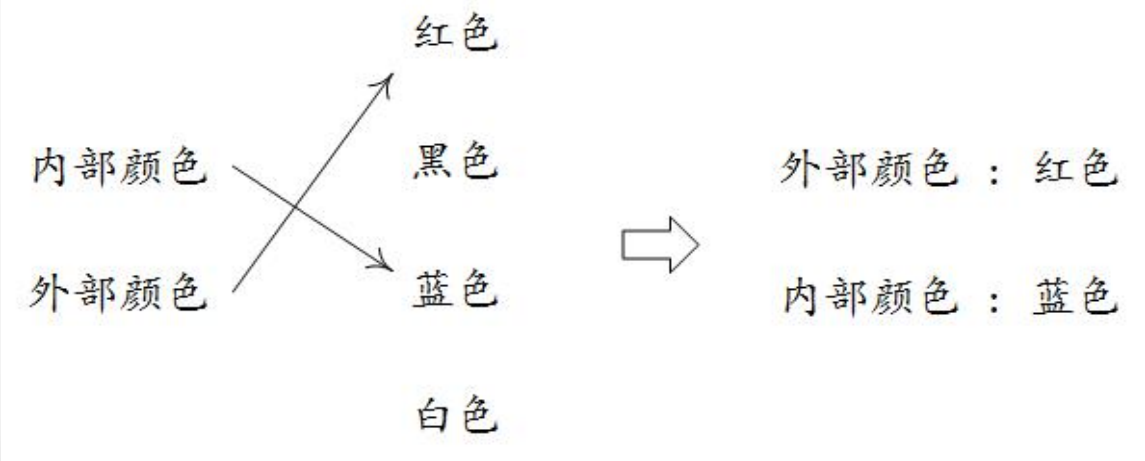
序列类型概述

- 序列类型有一些通用的操作符和函数

操作符	描述
x in s	如果x是s的元素，返回True，否则返回False
x not in s	如果x不是s的元素，返回True，否则返回False
s + t	连接s和t
s * n 或 n * s	将序列s复制n次
s[i]	索引，返回序列的第i个元素
s[i: j]	切片，返回包含序列s第i到j个元素的子序列（不包含第j个元素）
s[i: j: k]	步骤切片，返回包含序列s第i到j个元素以j为步数的子序列
len(s)	序列s的元素个数（长度）
min(s)	序列s中的最小元素
max(s)	序列s中的最大元素
s.index(x)	序列s中第一次出现元素x的位置
s.count(x)	序列s中出现x的总次数

映射类型概述

- 映射类型是“**键-值**”数据项的组合，每个元素是一个键值对，即元素是(key, value)，元素之间是无序的。键值对是一种二元关系，源于属性和值的映射关系



映射类型概述

- 映射类型是序列类型的一种扩展。在序列类型中，采用从0开始的正向递增序号进行具体元素值的索引。而映射类型则由用户来定义序号，即键，用其去索引具体的值。
- 键（key）表示一个属性，也可以理解为一个类别或项目，值（value）是属性的内容，键值对刻画了一个属性和它的值。键值对将映射关系结构化，用于存储和表达。

列表类型

列表的定义

- 列表是包含0个或多个元组组成的有序序列，属于序列类型。列表可以元素进行增加、删除、替换、查找等操作。列表没有长度限制，元素类型可以不同，不需要预定义长度。
- 列表类型用中括号（**[]**）表示，也可以通过**list(x)**函数将集合或字符串类型转换成列表类型。

列表的定义

```
>>>ls = [1010, "1010", [1010, "1010"], 1010]
>>>ls
[1010, '1010', [1010, '1010'], 1010]
>>>list('列表可以由字符串生成')
['列', '表', '可', '以', '由', '字', '符', '串', '生', '成']
>>>list()
[]
```

- 列表属于序列类型，所以列表类型支持序列类型对应的操作

列表的索引

- 索引是列表的基本操作，用于获得列表的一个元素。使用中括号作为索引操作符。

```
>>>ls = [1010, "1010", [1010, "1010"], 1010]
>>>ls[3]
1010
>>>ls[-2]
[1010, '1010']
>>>ls[5]
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    ls[5]
IndexError: list index out of range
```

列表的索引

- 可以使用遍历循环对列表类型的元素进行遍历操作，基本使用方式如下：

for <循环变量> in <列表变量>:

<语句块>

```
>>>ls = [1010, "1010", [1010, "1010"], 1010]
>>>for i in ls:
        print(i*2)
2020
10101010
[1010, '1010', 1010, '1010']
2020
```

列表的切片

- 切片是列表的基本操作，用于获得列表的一个片段，即获得一个或多个元素。切片后的结果也是列表类型。切片有两种使用方式：

<列表或列表变量>[N: M]

或

<列表或列表变量>[N: M: K]

列表的切片

- 切片获取列表类型从N到M（不包含M）的元素组成新的列表。当K存在时，切片获取列表类型从N到M（不包含M）以K为步长所对应元素组成的列表。

```
>>>ls = [1010, "1010", [1010, "1010"], 1010]
>>>ls[1:4]
['1010', [1010, '1010'], 1010]
>>>ls[-1:-3]
[]
>>>ls[-3:-1]
['1010', [1010, '1010']]
>>>ls[0:4:2]
[1010, [1010, '1010']]
```

列表类型的操作

列表的操作函数

- 列表类型继承序列类型特点，有一些通用的操作函数

操作函数	描述
len(ls)	列表ls的元素个数（长度）
min(ls)	列表ls中的最小元素
max(ls)	列表ls中的最大元素
list(x)	将x转变成列表类型

```
>>>ls = [1010, "1010", [1010, "1010"], 1010]
>>>len(ls)
4
>>>lt = ["Python", ["1010", 1010, [1010, "Python"]]]
>>>len(lt)
2
```

列表的操作函数

- min(ls)和max(ls)分别返回一个列表的最小或最大元素，使用这两个函数的前提是列表中各元素类型可以进行比较。

```
>>>ls = [1010, 10.10, 0x1010]
>>>min(ls)
10.1
>>>lt = ["1010", "10.10", "Python"]
>>>max(lt)
'Python'
>>>ls = ls + lt
>>>print(ls)
[1010, 10.1, 4112, '1010', '10.10', 'Python']
>>>min(ls)
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    min(ls)
TypeError: '<' not supported between instances of 'str' and 'float'
```

列表的操作函数

- list(x)将变量x转变成列表类型，其中x可以是字符串类型，也可以是字典类型。

```
>>>list("Python")
['P', 'y', 't', 'h', 'o', 'n']
>>>list({"小明", "小红", "小白", "小新"})
['小红', '小明', '小新', '小白']
>>>list({"201801":"小明", "201802":"小红", "201803":"小白"})
['201801', '201802', '201803']
```

列表的操作方法

- 列表类型存在一些操作方法，使用语法形式是：

<列表变量>.<方法名称>(<方法参数>)

方法	描述
ls.append(x)	在列表ls最后增加一个元素x
ls.insert(i, x)	在列表ls第i位置增加元素x
ls.clear()	删除ls中所有元素
ls.pop(i)	将列表ls中第i项元素取出并删除该元素
ls.remove(x)	将列表中出现的第一个元素x删除
ls.reverse()	列表ls中元素反转
ls.copy()	生成一个新列表，复制ls中所有元素

列表的操作方法

- ls.append(x)在列表ls最后增加一个元素x。

```
>>>lt = ["1010", "10.10", "Python"]
>>>lt.append(1010)
>>>print(lt)
['1010', '10.10', 'Python', 1010]
>>>lt.append([1010, 0x1010])
>>>print(lt)
['1010', '10.10', 'Python', 1010, [1010, 4112]]
```

列表的操作方法

- ls.append(x)仅用于在列表中增加一个元素，如果希望增加多个元素，可以使用**加号**，将两个列表合并。

```
>>>lt = ["1010", "10.10", "Python"]
>>>ls = [1010, [1010, 0x1010]]
>>>ls += lt
>>>print(lt)
['1010', '10.10', 'Python', 1010, [1010, 4112]]
```

列表的操作方法

- ls.insert(i, x)在列表ls中序号i位置上增加元素x，序号i之后的元素序号依次增加。

```
>>>lt = ["1010", "10.10", "Python"]
>>>lt.insert(1, 1010)
>>>print(lt)
['1010', 1010, '10.10', 'Python']
```

- ls.clear()将列表ls的所有元素删除，清空列表。

```
>>>lt = ["1010", "10.10", "Python"]
>>>lt.clear()
>>>print(lt)
[]
```

列表的操作方法

- ls.pop(i)将返回列表ls中第i位元素，并将该元素从列表中删除。

```
>>>lt = ["1010", "10.10", "Python"]
>>>print(lt.pop(1))
10.10
>>>print(lt)
["1010", "Python"]
```

- ls.remove(x)将删除列表ls中第一个出现的x元素。

```
>>>lt = ["1010", "10.10", "Python"]
>>>lt.remove("10.10")
>>>print(lt)
["1010", "Python"]
```

列表的操作方法

- 除了上述方法，还可以使用Python保留字del对列表元素或片段进行删除，使用方法如下：

del <列表变量>[<索引序号>] 或

del <列表变量>[<索引起始>:<索引结束>]

```
>>>lt = ["1010", "10.10", "Python"]
>>>del lt[1]
>>>print(lt)
["1010", "Python"]
>>>lt = ["1010", "10.10", "Python"]
>>>del lt[1:]
>>>print(lt)
["1010"]
```

列表的操作方法

- ls.reverse()将列表ls中元素进行逆序反转。

```
>>>lt = ["1010", "10.10", "Python"]
>>>print(lt.reverse())
['Python', '10.10', '1010']
```

- ls.copy() 复制ls中所有元素生成一个新列表。

```
>>>lt = ["1010", "10.10", "Python"]
>>>ls = lt.copy()
>>>lt.clear()  # 清空lt
>>>print(ls)
["1010", "10.10", "Python"]
```

- 由上例看出，一个列表lt使用.copy()方法复制后赋值给变量ls，将lt元素清空不影响新生成的变量ls。

列表的操作方法

- 需要注意，对于基本的数据类型，如整数或字符串，可以通过等号实现元素赋值。但对于列表类型，使用等号无法实现真正的赋值。其中，ls = lt语句并不是拷贝lt中元素给变量ls，而是新关联了一个引用，即ls和lt所指向的是同一套内容。

```
>>>lt = ["1010", "10.10", "Python"]
>>>ls = lt      # 仅使用等号
>>>lt.clear()
>>>print(ls)
[]
```

列表的操作方法

- 使用索引配合等号（=）可以对列表元素进行修改。

```
>>>lt = ["1010", "10.10", "Python"]
>>>lt[1] = 1010
>>>print(lt)
["1010", 1010, "Python"]
```

- 列表是一个十分灵活的数据结构，它具有处理任意长度、混合类型的能力，并提供了丰富的基础操作符和方法。当程序需要使用组合数据类型管理批量数据时，请尽量使用列表类型。

字典类型

字典的定义

- “键值对”是组织数据的一种重要方式，广泛应用在Web系统中。键值对的基本思想是将“值”信息关联一个“键”信息，进而通过键信息查找对应值信息，这个过程叫映射。Python语言中通过字典类型实现映射。

字典的定义

- Python语言中的字典使用大括号{}建立，每个元素是一个键值对，使用方式如下：

  {<键1>:<值1>, <键2>:<值2>, … , <键n>:<值n>}

- 其中，键和值通过冒号连接，不同键值对通过逗号隔开。字典类型也具有和集合类似的性质，即键值对之间没有顺序且不能重复。

字典的定义

- 变量d可以看作是“学号”与“姓名”的映射关系。需要注意，字典各个元素并没有顺序之分。

```
>>>d = {"201801":"小明", "201802":"小红", "201803":"小白"}
>>>print(d)
{'201801': '小明', '201802': '小红', '201803': '小白'}
```

字典的索引

- 列表类型采用元素顺序的位置进行索引。由于字典元素“键值对”中键是值的索引，因此，可以直接利用键值对关系索引元素。
- 字典中键值对的索引模式如下，采用中括号格式：

<值> = <字典变量>[<键>]

```
>>>d = {"201801":"小明", "201802":"小红", "201803":"小白"}
>>>print(d["201802"])
小红
```

字典的索引

- 利用索引和赋值（=）配合，可以对字典中每个元素进行修改。

```
>>>d["201802"] = '新小红'
>>>print(d)
{'201801': '小明', '201803': '小白', '201802': '新小红'}
```

字典的索引

- 使用大括号可以创建字典。通过索引和赋值配合，可以向字典中增加元素。

```
>>>t = {}
>>>t["201804"] = "小新"
>>>print(d)
{'201804': '小新'}
```

- 字典是存储可变数量键值对的数据结构，键和值可以是任意数据类型，通过键索引值，并可以通过键修改值。

字典类型的操作

字典的操作函数

- 字典类型有一些通用的操作函数

操作函数	描述
len(d)	字典d的元素个数（长度）
min(d)	字典d中键的最小值
max(d)	字典d中键的最大值
dict()	生成一个空字典

字典的操作函数

- len(d)给出字典d的元素个数，也称为长度。

```
>>>d = {"201801":"小明", "201802":"小红", "201803":"小白"}
>>>len(d)
3
```

- min(d)和max(d)分别返回字典d中最小或最大索引值。

```
>>>d = {"201801":"小明", "201802":"小红", "201803":"小白"}
>>>min(d)
'201801'
>>>max(d)
'201803'
```

字典的操作函数

- dict()函数用于生成一个空字典，作用和{}一致。

```
>>>d = dict()
>>>print(d)
{}
```

字典的操作方法

- 字典类型存在一些操作方法，使用语法形式是：

<字典变量>.<方法名称>(<方法参数>)

操作方法	描述
d.keys()	返回所有的键信息
d.values()	返回所有的值信息
d.items()	返回所有的键值对
d.get(key, default)	键存在则返回相应值，否则返回默认值
d.pop(key, default)	键存在则返回相应值，同时删除键值对，否则返回默认值
d.popitem()	随机从字典中取出一个键值对，以元组(key, value)形式返回
d.clear()	删除所有的键值对

字典的操作方法

- d.keys()返回字典中的所有键信息，返回结果是Python的一种内部数据类型dict_keys，专用于表示字典的键。如果希望更好的使用返回结果，可以将其转换为列表类型。

```
>>>d = {"201801":"小明", "201802":"小红", "201803":"小白"}
>>>d.keys()
dict_keys(['201801', '201802', '201803'])
>>>type(d.keys())
<class 'dict_keys'>
>>>list(d.keys())
['201801', '201802', '201803']
```

字典的操作方法

- d.values()返回字典中的所有值信息，返回结果是Python的一种内部数据类型dict_values。如果希望更好的使用返回结果，可以将其转换为列表类型。

```
>>>d = {"201801":"小明", "201802":"小红", "201803":"小白"}
>>>d.values()
dict_values(['小明', '小红', '小白'])
>>>type(d.values())
<class 'dict_values'>
>>>list(d.values())
['小明', '小红', '小白']
```

字典的操作方法

- d.items()返回字典中的所有键值对信息，返回结果是Python的一种内部数据类型dict_items。

```
>>>d = {"201801":"小明", "201802":"小红", "201803":"小白"}
>>>d.items()
dict_items([('201801', '小明'), ('201802', '小红'),
('201803', '小白')])
>>>type(d.items())
<class 'dict_items'>
>>>list(d.items())
[('201801', '小明'), ('201802', '小红'), ('201803', '小白')]
```

字典的操作方法

- d.get(key, default)根据键信息查找并返回值信息，如果key存在则返回相应值，否则返回默认值，第二个元素default可以省略，如果省略则默认值为空。

```
>>>d = {"201801":"小明", "201802":"小红", "201803":"小白"}
>>>d.get('201802')
'小红'
>>>d.get('201804')
>>>d.get('201804', '不存在')
'不存在'
```

字典的操作方法

- d.pop(key, default)根据键信息查找并取出值信息，如果key存在则返回相应值，否则返回默认值，第二个元素default可以省略，如果省略则默认值为空。相比d.get()方法，d.pop()在取出相应值后，将从字典中删除对应的键值对。

```
>>>d = {"201801":"小明", "201802":"小红", "201803":"小白"}
>>>d.pop('201802')
'小红'
>>>print(d)
{'201801': '小明', '201803': '小白'}
>>>d.pop('201804', '不存在')
'不存在'
```

字典的操作方法

- d.popitem()随机从字典中取出一个键值对，以元组(key, value)形式返回。取出后从字典中删除这个键值对。

```
>>>d = {"201801":"小明", "201802":"小红", "201803":"小白"}
>>>print(d.popitem())
('201803', '小白')
>>>d
{'201801': '小明', '201802': '小红'}
```

- d.clear()删除字典中所有键值对。

```
>>>d = {"201801":"小明", "201802":"小红", "201803":"小白"}
>>>d.clear()
>>>print(d)
{}
```

字典的操作方法

- 此外，如果希望删除字典中某一个元素，可以使用Python保留字**del**。

```
>>>d = {"201801":"小明", "201802":"小红", "201803":"小白"}
>>>del d["201801"]
>>>print(d)
{'201802': '小红', '201803': '小白'}
```

- 字典类型也支持保留字in，用来判断一个键是否在字典中。如果在则返回True，否则返回False。

```
>>>d = {"201801":"小明", "201802":"小红", "201803":"小白"}
>>>"201801" in d
True
>>>"201804" in d
False
```

字典的操作方法

- 与其他组合类型一样，字典可以遍历循环对其元素进行遍历，基本语法结构如下：

for <变量名> in <字典名>

<语句块>

- for循环返回的变量名是字典的索引值。如果需要获得键对应的值，可以在语句块中通过get()方法获得。

```
>>>d = {"201801":"小明", "201802":"小红", "201803":"小白"}
>>>for k in d:
      print("字典的键和值分别是：{}和{}".format(k, d.get(k)))

字典的键和值分别是：201801和小明
字典的键和值分别是：201802和小红
字典的键和值分别是：201803和小白
```

实例解析：文本词频统计

文本词频统计

- 在很多情况下，会遇到这样的问题：对于一篇给定文章，希望统计其中多次出现的词语，进而概要分析文章的内容。这个问题的解决可用于对网络信息进行自动检索和归档。
- 在信息爆炸时代，这种归档或分类十分有必要。这就是“词频统计”问题。

文本词频统计

统计《哈姆雷特》英文词频

- 第一步：分解并提取英文文章的单词
- 第二步：对每个单词进行计数
- 第三走：对单词的统计值从高到低进行排序

文本词频统计

- 第一步：分解并提取英文文章的单词

通过txt.lower()函数将字母变成小写，排除原文大小写差异对词频统计的干扰。为统一分隔方式，可以将各种特殊字符和标点符号使用txt.replace()方法替换成空格，再提取单词。

文本词频统计

- 第二步：对每个单词进行计数

```
if word in counts:
        counts[word] = counts[word] + 1
else:
        counts[word] = 1
```

或者，这个处理逻辑可以更简洁的表示为如下代码：

```
counts[word] = counts.get(word,0) + 1
```

文本词频统计

- 第三步：对单词的统计值从高到低进行排序

由于字典类型没有顺序，需要将其转换为有顺序的列表类型，再使用sort()方法和lambda函数配合实现根据单词次数对元素进行排序。

```
items = list(counts.items())#将字典转换为记录列表
items.sort(key=lambda x:x[1], reverse=True)  #以第2列排序
```

文本词频统计

```
1   #CalHamlet.py
2   def getText():
3      txt = open("hamlet.txt", "r").read()
4      txt = txt.lower()
5      for ch in '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~':
6          txt = txt.replace(ch, " ")    #将文本中特殊字符替换为空格
7      return txt
8   hamletTxt = getText()
9   words  = hamletTxt.split()
10  counts = {}
11  for word in words:
12      counts[word] = counts.get(word,0) + 1
13  items = list(counts.items())
14  items.sort(key=lambda x:x[1], reverse=True)
15  for i in range(10):
16      word, count = items[i]
17      print ("{0:<10}{1:>5}".format(word, count))
```

```
>>>
the         1138
and          965
to           754
of           669
you          550
a            542
i            542
my           514
hamlet       462
in           436
```

本章小结

本章主要针对初学程序设计的读者，具体讲解了程序设计语言的基本概念，理解程序开发的IPO编写方法，配置Python开发环境的具体步骤，以及Python语言和Python程序特点等内容，进一步给出了5个简单Python实例代码，帮助读者测试Python开发环境，对该语言有一个直观认识。

Python大戏即将上演，一起来追剧吧。