Python 语言程序设计

陈 峦 副教授

13880209111，chluan@uestc.edu.cn

研究院大楼316#

第七章 字典与集合

- **Python**中的列表和元组都属于序列类型。
- 序列的特点是数据元素之间保持先后的顺序关系，通过位置编号（索引）来访问序列的数据元素。
- 字典和集合的数据元素之间没有任何确定的顺序关系，属于无序的数据集合体，因此不能像序列那样通过位置索引来访问数据元素。

- 在Python中，字典是由“关键字:值”对组成的集合体。
- 集合是指由无序的、不重复的元素组成的集合体，类似于数学中的集合概念。
- 作为一种复合数据类型，字典和集合之间的主要区别在于它们的操作，字典主要关心其元素的检索、插入和删除，集合主要考虑集合之间的并、交和差操作。

7.1 字典概述

- 在Python中，字典（dictionary）是在大括号中放置一组逗号分隔的“关键字:值”对（key-value pair），它是无序的“关键字:值”对的集合体。
- 关键字就相当于索引，而它对应的“值”就是数据，数据是根据关键字来存储的，只要找到这个关键字就可以找到需要的值，这种对应关系是唯一的。

- 在同一个字典之内关键字必须是互不相同的，字典中一个关键字只能与一个值关联，对于同一个关键字，后添加的值会覆盖之前的值。

1. 字典的索引

- 字典是Python中唯一的映射类型，采用“关键字:值”对的形式存储数据。
- 序列是以连续的整数为索引；字典以关键字为索引，关键字可以是任意不可变类型，如整数、字符串。

- 如果元组中只包含字符串和数字，则元组也可以作为关键字；
- 如果元组直接或间接地包含了可变类型，就不能作为关键字。
- 不能用列表做关键字，因为列表可以修改。

- Python对关键字进行哈希函数运算，根据计算的结果决定值的存储地址，所以字典是无序存储的，且关键字必须是可哈希的。
- 可哈希表示关键字必须是不可变类型，否则会出现TypeError异常。
- 可以用hash()函数求得数据的哈希值，用来判断一个数据能否作为字典的关键字。

- 例：
- >>> hash((2,3))
- 1497441108
- >>> hash("uestc")
- -1265515743
- >>> hash([2,3]) #出现错误
- #TypeError: unhashable type: 'list'

元组是可哈希类型，可以作为字典的关键字，而列表是不可哈希类型，不能作为字典的关建字。

2.字典与序列的区别

- （1）存取和访问数据的方式不同。
- 字典中的元素是通过关键字来存取的，而序列是通过编号来存取的。
- 字典通过关键字将一系列值联系起来，这样就可以使用关键字从字典中取出一个元素。
- 如同列表和元组一样，可以使用索引操作从字典中获取内容，但字典的索引是关键字，而序列从起始元素开始按顺序编号进行索引。

- （2）列表、元组是有序的数据集合体，而字典是无序的数据集合体。
- 与列表、元组不同，保存在字典中的元素并没有特定的顺序。
- 实际上，Python将各项从左到右随机排序，以便快速查找。
- 关键字提供了字典中元素的象征性位置，而不代表物理存储顺序。

- （3）字典是可变类型，可以在原处增长或缩短，无须生成一份副本。
- （4）字典是异构的，可以包含任何类型的数据，如列表、元组或其他字典，支持任意层次的嵌套。

7.2 字典的操作

- 对于字典，无法实现有序分片和连接。
- 字典的主要操作是依据关键字来存储和提取值，也可以用del语句来删除“关键字:值”对。
- 如果用一个已经存在的关键字存储值，则以前为该关键字分配的值就会被覆盖。
- 试图从一个不存在的关键字中取值会导致错误。

7.2.1 字典的创建

- 字典就是用大括号括起来的“关键字:值”对的集合体，每一个“关键字:值”对也称为字典的元素或数据项。

1. 创建字典并赋值

- 创建字典并赋值的一般格式为：
- 字典名={[关键字1:值1[,关键字2:值2,……,关键字n:值n]])
- 其中，关键字与值之间用冒号“:”分隔，字典元素与元素之间用逗号“,”分隔，字典中的关键字必须是唯一的，而值可以不唯一。
- 当“关键字:值”对都省略时产生一个空字典。

- 例：
- >>> d1={}
- >>> d2={"x":2","y":3}
- >>> d1,d2
- ({}, {'x': 2, 'y': 3})
- >>> d3={"var":{"x":2,"y":3},"z":4}
- >>> d3
- {'var': {'x': 2, 'y': 3}, 'z': 4}

2. dict()函数

- 可以用dict()函数创建字典。
- （1）使用dict()函数创建一个空字典并给变量赋值。
- 例：
- >>> d1=dict()
- >>> d1
- {}

- （2）使用列表或元组作为dict()函数参数。
- 例：
- >>> d1=dict((["x",2],["y",3]))
- >>> d1
- {'x': 2, 'y': 3}
- >>> d2=dict([["x",1],["y",2]])
- >>> d2
- {'x': 1, 'y': 2}

- （3）将数据按“关键字=值”形式作为参数传递给dict()函数。
- 例：
- >>> d=dict(x=1,y=2)
- >>> d
- {'x': 1, 'y': 2}

- 例：
- >>> x=["aa","bb","cc"]
- >>> y=[1,2,3]
- >>> s=dict(zip(x,y))
- >>> s
- {'aa': 1, 'bb': 2, 'cc': 3}

- 例：
- >>> x=("aa","bb","cc")
- >>> y=(1,2,3)
- >>> s=dict(zip(x,y))
- >>> s
- {'aa': 1, 'bb': 2, 'cc': 3}

- 例：
- >>> x=[1,3,5,7]
- >>> s=dict(enumerate(x))
- >>> s
- {0: 1, 1: 3, 2: 5, 3: 7}
- >>> x=(1,3,5,7)
- >>> s=dict(enumerate(x))
- >>> s
- {0: 1, 1: 3, 2: 5, 3: 7}

7.2.2 字典的常用操作

1. 字典的访问

- Python通过关键字来访问字典的元素，一般格式为：
- 字典名[关键字]
- 如果关键字不在字典中，会引发一个KeyError错误。

- （1）以关键字进行索引计算。
- 例：
- >>> d={"x":2,"y":3}
- >>> d["x"]
- 2
- >>> d["z"]　#出现错误
- #KeyError: 'z'
- >>> d[x]　#出现错误
- #NameError: name 'x' is not defined

- （2）字典嵌套字典的关键字索引。
- 例：
- >>> d={"var":{"x":2,"y":3},"z":4}
- >>> d["var"]["x"]
- 2
- >>> d["var"]["y"]
- 3

- （3）字典嵌套列表的关键字索引。
- >>> d={"x":[2,3,4],"y":5}
- >>> d["x"][0]
- 2
- >>> d["x"][2]
- 4

- （4）字典嵌套元组的关键字索引。
- 例：
- >>> d={"x":(2,3,4),"y":5}
- >>> d["x"][2]
- 4

2. 字典的更新

- 更新字典值的语句格式为：
- 字典名[关键字]=值
- 如果关键字已经存在，则修改关键字对应的元素的值；
- 如果关键字不存在，则在字典中增加一个新元素，即“关键字:值”对。

- 例：
- >>> d={"x":2,"y":3}
- >>> d["x"]=4
- >>> d["y"]=[5,6,7]
- >>> d
- {'x': 4, 'y': [5, 6, 7]}
- >>> d["z"]=8
- >>> d
- {'x': 4, 'y': [5, 6, 7], 'z': 8}

3. 字典元素的删除

- 删除字典元素使用以下函数或方法。
- del 字典名[关键字]：删除关键字所对应的元素。
- del 字典名：删除整个字典。
- 例：
- >>> d={"x":2,"y":3,"z":4}
- >>> del d["x"]
- >>> d
- {'y': 3, 'z': 4}

4. 检查字典关键字是否存在

- 通过以下运算符判断关键字是否存在于字典中。
- 关键字 in 字典：值为True，则表示关键字存在于字典中。
- 关键字 not in 字典：值为True，则表示关键字不存在于字典中。

- 例：
- >>> d={"x":2,"y":3,"z":4}
- >>> "y" in d
- True
- >>> "x" not in d
- False
- >>> list(d)
- ['x', 'y', 'z']

5. 字典的长度和运算

- len()函数可以获取字典所包含“关键字:值”对的数目，即字典长度。
- 虽然也支持max()、min()、sum()和sorted()函数，但只针对字典的关键字进行计算，很多情况下没有实际意义。

- 例：
- >>> d={"x":2,"y":3,"z":4}
- >>> len(d)
- 3
- >>> max(d)
- 'z'
- >>> min(d)
- 'x'
- >>> sorted(d)
- ['x', 'y', 'z']

- 字典不支持连接(+)和重复操作符(*)，关系运算中只有“==”和“!=”有意义。
- 例：
- >>> d={"a":2,"b":3}
- >>> t={"A":2,"B":3}
- >>> d==t
- False
- >>> s=d+t　　#出现错误
- #TypeError: unsupported operand type(s) for +: 'dict' and 'dict'

7.2.3 字典的常用方法

- Python字典和集合实际上也是对象，Python提供了很多有用的方法。

1. fromkeys()方法

- **d.fromkeys(序列[，值]):创建并返回一个新字典，以序列中的元素做该字典的关键字，指定的值做该字典中所有关键字对应的初始值（默认为None）。**

- 例：
- >>> d={}.fromkeys(("x","y"),2)
- >>> d
- {'x': 2, 'y': 2}
- >>> t={}.fromkeys(["x","y"])
- >>> t
- {'x': None, 'y': None}
- >>> t["x"]=3
- >>> t
- {'x': 3, 'y': None}

2. keys()、values()、items()方法

- **d.keys()**：返回一个包含字典所有关键字的列表。
- **d.values()**：返回一个包含字典所有值的列表。
- **d.items()**：返回一个包含所有(关键字,值)元组的列表。

- 例：
- >>> d={"x":2,"y":3,"z":4}
- >>> d.keys()
- dict_keys(['x', 'y', 'z'])
- >>> d.values()
- dict_values([2, 3, 4])
- >>> d.items()
- dict_items([('x', 2), ('y', 3), ('z', 4)])

3. 字典复制与删除的方法

- **d.copy()：返回字典d的副本。**
- **d.clear()：删除字典d中的全部元素，d变成一个空字典。**
- **d.pop(key)：是从字典d中删除关键字key并返回删除的值。**

- 例：
- >>> d={"x":2,"y":3,"z":4}
- >>> d.pop("z")
- 4
- >>> d
- {'x': 2, 'y': 3}
- >>> d.clear()
- >>> d
- {}

- 例：
- >>> d={"x":2,"y":3,"z":4}
- >>> s=d
- >>> t=d.copy()
- >>> d.clear()
- >>> s
- {}
- >>> t
- {'x': 2, 'y': 3, 'z': 4}

- d.popitem()：删除字典的“关键字:值”对，并返回关键字和值构成的元组。
- 例：
- >>> d={"x":2,"y":3,"z":4}
- >>> d.popitem()
- ('z', 4)
- >>> d
- {'x': 2, 'y': 3}

4. get()方法和pop()方法

- d.get(key[,value])：如果字典d中存在关键词key，则返回关键字对应的值；若key在字典中不存在，则返回value的值，value默认为None。该方法不改变原对象的数据。

- 例：
- >>> d={"x":2,"y":3,"z":4}
- >>> d.get("x")
- 2
- >>> d.get("m")
- （无输出）
- >>> d.get("n","notexists")
- 'notexists'
- >>> d
- {'x': 2, 'y': 3, 'z': 4}

- **d.pop(key[,value])：和get方法相似。如果字典d中存在关键词key，删除并返回关键词对应的值；若key在字典中不存在，则返回value的值，value默认为None。**

- 例：
- >>> d={"x":2,"y":3,"z":4}
- >>> d.pop("y","notexists")
- 3
- >>> d
- {'x': 2, 'z': 4}
- >>> d.pop("m","notexists")
- 'notexists'

5. setdefault()方法和update()方法

- d.setdefault(key,[value])：如果字典d中key存在，则返回其值；若key不存在，则给字典添加key:value对，value默认为None。

- >>> d={"x":2,"y":3,"z":4}
- >>> d.setdefault("x",5)
- 2
- >>> d
- {'x': 2, 'y': 3, 'z': 4}
- >>> d.setdefault("m",5)
- 5
- >>> d
- {'x': 2, 'y': 3, 'z': 4, 'm': 5}

- **d2.update(d1)：将字典d1的“关键字:值”对添加到字典d2中。d1合并到d2，d1没有变化，d2变化了。**

- 例：
- >>> d={"x":2,"y":3,"z":4}
- >>> t={"m":5,"n":6}
- >>> d.update(t)
- >>> d
- {'x': 2, 'y': 3, 'z': 4, 'm': 5, 'n': 6}
- >>> t
- {'m': 5, 'n': 6}

7.2.4 字典的遍历

- 结合for循环语句，字典的遍历是很方便的，有多种方式。
- 注意：访问一个不存在的关键字时，会发生KeyError异常，访问前可使用in或not in判断一下字典关键字是否存在。

1. 遍历字典的关键字

- d.keys()：返回一个包含字典所有关键字的列表，所以对字典关键字的遍历转换为对列表的遍历。
- 例：
- >>> d={"x":2,"y":3,"z":4}
- >>> for k in d.keys():print(k,d[k])
- x 2
- y 3
- z 4

2. 遍历字典的值

- d.values()：返回一个包含字典所有值的列表，所以对字典值的遍历转换为对列表的遍历。
- 例：
- >>> d={"x":2,"y":3,"z":4}
- >>> for i in d.values():print(i)
- 2
- 3
- 4

3. 遍历字典的元素

- d.items()：返回一个包含所有(关键字,值)元组的列表，所以对字典元素的遍历转换为对列表的遍历。
- 例：
- >>> d={"x":2,"y":3,"z":4}
- >>> for i in d.items():print(i)
- ('x', 2)
- ('y', 3)
- ('z', 4)

7.3 集合的操作

- 在Python中，集合（ set ）是一个无序排列的、不重复的数据集合体，类似于数学中的集合概念，可对其进行并、交、差等运算。

- 集合和字典都属于无序集合体，有许多操作是一致的。
- 例：判断集合元素是否在集合中存在(x in set，x not in set)，求集合的长度len()、最大值max()、最小值min()、数值元素之和sum()，集合的遍历for x in set。
- 作为一个无序的集合体，集合不记录元素位置或插入点，因此不支持索引、分片等操作。

7.3.1 集合的创建

- 在Python中，创建集合有两种方式：
- 一种是用一对大括号将多个用逗号分隔的数据括起来；
- 另一种使用set()函数，该函数可以将字符串、列表、元组等类型的数据转换成集合类型的数据。

- 例：
- >>> s1={1,2,3,4,5}
- >>> s1
- {1, 2, 3, 4, 5}
- >>> s2=set("uestc")
- >>> s2
- {'c', 'u', 'e', 't', 's'}

- 在Python中，用大括号将集合元素括起来，这与字典的创建类似，但{}表示空字典，空集合用set()表示。

- 例：
- >>> s={}
- >>> type(s)
- <class 'dict'>
- >>> t=set()
- >>> t
- set()
- >>> type(t)
- <class 'set'>

- 注意：
- 集合中不能有相同元素，如果在创建集合时有重复元素，Python会自动删除重复的元素。
- 例：
- >>> s={1,2,2,2,3,3,4,4,4,5}
- >>> s
- {1, 2, 3, 4, 5}

- 集合的自动删除重复元素这个特性非常有用，例如，要删除列表中大量的重复元素，可以先用set()函数将列表转换成集合，再用list()函数将集合转换成列表，操作效率非常高。

- 例：
- >>> s=[1,2,3,4,2,2,3,4,3,2,3,4]
- >>> t=set(s)
- >>> t
- {1, 2, 3, 4}
- >>> s=list(t)
- >>> s
- [1, 2, 3, 4]

- Python集合包含两种类型：可变集合（set）和不可变集合（frozenset）。
- （注：前面讲述的就是创建可变集合的方法）
- 可变集合可以添加和删除集合元素，但集合中的元素必须是不可修改的，因此集合的元素不能是列表或字典，只能是数值、字符串或元组。
- 集合的元素不能是可变集合，因为可变集合是可以修改的，不能做其他集合的元素，也不能作为字典的关键字。

- 例：
- >>> s={1,2,(3,4),5}
- >>> s
- {1, 2, (3, 4), 5}
- >>> s={1,2,[3,4],5} #出现错误
- #TypeError: unhashable type: 'list'
- >>> s={1,2,{"x":3},4}　　#出现错误
- #TypeError: unhashable type: 'dict'

- Python提供frozenset()函数来创建不可变集合，不可变集合是不能修改的，因此能作为其他集合的元素，也能作为字典的关键字。

- 例：
- >>> s=frozenset({1,2,3})
- >>> type(s)
- <class 'frozenset'>
- >>> s
- frozenset({1, 2, 3})
- >>> t={4,5,s,6}
- >>> t
- {frozenset({1, 2, 3}), 4, 5, 6}

7.3.2 集合的常用运算

1. 传统的集合运算

- s1|s2|…|sn：计算s1，s2，……，sn的并集。
- s1&s2&……&sn：计算s1，s2，……，sn的交集。
- s1-s2-…-sn：计算s1，s2，……，sn的差集。
- s1^s2：计算s1，s2的对称差集，即求s1和s2中相异元素。
- 优先级从高到低依次为：-、&、^、|。

- 例：
- >>> s={1,2,3}|{"a","b"}|{4,5}
- >>> s
- {1, 2, 3, 'a', 4, 5, 'b'}
- >>> s={1,2,3,4}&{1,3,4,5}&{2,4,6}
- >>> s
- {4}
- >>> s={1,2,3,4,5}-{1,3}-{2,4}
- >>> s
- {5}

- 例：
- >>> s={1,2,3,4}-{2,4,6,8}
- >>> s
- {1, 3}
- >>> s={1,2,3,4}^{2,4,6,8}
- >>> s
- {1, 3, 6, 8}
- >>> t={1,2,3,4}&{2,4,6,8}
- >>> t
- {2, 4}

- 例：
- >>> s={1,2,3,4}
- >>> t={1,3,5,7}
- >>> x=s-t|s&t
- >>> x
- {1, 2, 3, 4}
- >>> x=s^t|s&t
- >>> x
- {1, 2, 3, 4, 5, 7}

- 例：
- >>> s={1,2,3,4}
- >>> t={1,3,5,7}
- >>> x=s^t-s
- >>> x
- {1, 2, 3, 4, 5, 7}
- >>> x=(s^t)-s
- >>> x
- {5, 7}

- 例：
- >>> s={1,2,3,4}
- >>> t={1,3,5,7}
- >>> x=s|t-s
- >>> x
- {1, 2, 3, 4, 5, 7}
- >>> x=(s|t)-s
- >>> x
- {5, 7}

- 例：
- >>> s={1,2,3,4}
- >>> t={1,3,5,7}
- >>> x=s&t-s
- >>> x
- set()
- >>> x=s&(t-s)
- >>> x
- set()

运　算　符	功 能 说 明
+	算术加法，列表、元组、字符串合并与连接，正号
-	算术减法，集合的差集，相反数
*	算术乘法，序列重复
/	真除法
//	求整商，如果操作数中有实数，结果为实数形式的整数
%	求余数，字符串格式化
**	幂运算
<、<=、>、>=、==、!=	(值)大小比较，集合的包含关系比较
or	逻辑或
and	逻辑与
not	逻辑非
in	成员测试
is	对象同一性测试，即测试是否为同一个对象或内存地址是否相同
|、^、&、<<、>>、~	位或、位异或、位与、左移位、右移位、位求反
&、|、^	集合交集、并集、对称差集
@	矩阵相乘运算符

2. 集合的比较

- **s1==s2 ：** 如果**s1**和**s2**具有相同的元素，则返回**True**，否则返回**False**。
- 判断两个集合是否相等，只需判断其中的元素是否一致，而与顺序无关（集合是无序的）。
- **s1!=s2：** 如果**s1**和**s2**具有不同的元素，则返回**True**，否则返回**False**。

- 例：
- >>> s={1,2,3,4}
- >>> t={4,3,2,1}
- >>> s==t
- True
- >>> s!=t
- False
- >>> s==((s^t|s&t)-(t-s))
- True

- **s1>s2：真包含。如果s1不等于s2，且s2中所有的元素都是s1的元素（s1是s2的纯超集），则返回True，否则返回False。**
- **s1>=s2：包含。如果s2中所有的元素都是s1的元素（s1是s2的超集），则返回True，否则返回False。**

- 例：
- >>> s={1,2,3}
- >>> t={1,2,3,4}
- >>> t>=s
- True
- >>> t>s
- True
- >>> s>s
- False
- >>> s>=s
- True

- **s1<2：真包含于。如果s1不等于s2，且s1中所有的元素都是s2的元素（s1是s2的纯子集），则返回True，否则返回False。**
- **s1<=s2：包含于。如果s1中所有的元素都是s2的元素（s1是s2的子集），则返回True，否则返回False。**

- 例：
- >>> s=(1,2,3)
- >>> s={1,2,3}
- >>> t={1,2,3,4}
- >>> s<t
- True
- >>> s<=t
- True
- >>> s<s
- False
- >>> s<=s
- True

3. 集合元素的并入

- **s1|=s2：将s2的元素并入s1中。即s1=s1|s2。**

- 例：
- >>> s={1,2,3}
- >>> t={1,3,5,7}
- >>> s|=t
- >>> s
- {1, 2, 3, 5, 7}
- >>> s-=t
- >>> s
- {2}
- >>> s=s|t
- >>> s
- {1, 2, 3, 5, 7}

- 例：
- >>> s=frozenset({1,2,3})
- >>> t={1,3,5,7}
- >>> x={1,3,5,7}
- >>> s|=t
- >>> s
- frozenset({1, 2, 3, 5, 7})
- >>> x|=s
- >>> x
- {1, 2, 3, 5, 7}

4. 集合的遍历

- 集合与for循环语句配合使用，可实现对集合各个元素的遍历。
- 例：
- >>> s={1,2,3,4}
- >>> for i in s:print(i,end=" ")
- 1 2 3 4

- 例：
- m={1,2,3,4}
- n=set()
- s=0
- for i in m:
- 　　s+=i
- 　　n|={10-i}
- print(s)
- for j in n:print(j,end=" ")

```
10
8 9 6 7
```

7.3.3 集合的常用方法

- Python以面向对象方式为集合类型提供了很多方法，有些适用于可变集合类型和不可变集合类型，有些只适用于可变集合类型。

1. 适用于可变集合和不可变集合的方法

s1.issubset(s2)：如果集合s1是s2的子集，则返回True，否则返回False。

s1.issuperset(s2)：如果集合s1是s2的超集，则返回True，否则返回False。

s1.isdisjoint(s2)：如果集合s1和s2没有共同元素，则返回True，否则返回False。

- 例：
- >>> s={1,2,3,4}
- >>> t={1,3}
- >>> t.issubset(s)
- True
- >>> s.issubset(s)
- True
- >>> set().issubset(s)
- True
- >>> {}.issubset(s)　#出现错误
- #AttributeError: 'dict' object has no attribute 'issubset'

- 例：
- >>> s={1,2,3,4}
- >>> t={1,3}
- >>> s.issuperset(t)
- True
- >>> s.issuperset(s)
- True
- >>> s.issuperset(set())
- True
- >>> s.issuperset(frozenset({1,3}))
- True

- 例：
- >>> s={1,2,3,4}
- >>> t={1,3}
- >>> s.isdisjoint(t)
- False
- >>> s.isdisjoint(frozenset({5,6,7}))
- True
- >>> set().isdisjoint(t)
- True

- **s1.union(s2,…,sn)：返回s1,s2,……,sn的并集：s1∪s2∪…∪sn。**
- **s1.intersection(s2,…,sn)：返回s1,s2,……,sn的交集：s1∩s2∩…∩sn。**
- **s1.difference(s2,…,sn)：返回s1,s2,……,sn的差集：s1-s2-…-sn。**
- **s1.symmetric_difference(s2)：返回s1和s2的对称差值：s1^s2。**

- 例：
- >>> {1,2}.union({3,4},{"aa","bb"})
- {1, 2, 3, 4, 'bb', 'aa'}
- >>> {1,2,3,4}.intersection({2,4},{3,4,5})
- {4}
- >>> {1,2,3,4}.difference({2,4},{3,4,5})
- {1}
- >>> {1,2,3,4}.symmetric_difference({1,3,5})
- {2, 4, 5}

- s.copy()：复制集合s。
- 例：
- >>> s={1,2,3}
- >>> s.copy()
- {1, 2, 3}
- >>> t=frozenset({1,2,3})
- >>> t.copy()
- frozenset({1, 2, 3})

2. 适用于可变集合的方法

- **s.add(x)**：在集合s中添加对象x。
- **s.update(s1,s2,…,sn)**：用集合s1,s2,……,sn中的成员修改集合s，s=s∪s1∪s2∪……∪sn。

- 例：
- >>> s={1,2,3}
- >>> s.add("aa")
- >>> s
- {1, 2, 3, 'aa'}
- >>> s.add(4)
- >>> s
- {1, 2, 3, 4, 'aa'}
- >>> s.add({5,6})　#出现错误
- # TypeError: unhashable type: 'set'

- 例：
- >>> s={1,2,3}
- >>> s.update({4},{5,6})
- >>> s
- {1, 2, 3, 4, 5, 6}
- >>> s.update(7)　　#出现错误
- # TypeError: 'int' object is not iterable

- **s.intersection_update(s1,s2,…,sn)**：集合s中的成员是共同属于s1,s2,……,sn的元素，s=s∩s1∩s2∩…∩sn。
- **s.difference_update(s1,s2,…,sn)**：集合s中的成员是属于s但不包含在s1,,s2,……,sn的元素，s=s-s1-s2-…-sn。
- **s.symmetric_difference_update(s1)**：集合s中的成员更新为那些包含在s或s1中，但不是s和s1共有的元素，s=s^s1。

- 例：
- >>> s={1,2,3}
- >>> s.intersection_update({1,3,5},{2,3,4})
- >>> s
- {3}
- >>> t={1,2,3,4}
- >>> t.difference_update({1,3,5},{4,5,6})
- >>> t
- {2}

- 例：
- >>> s={1,2,3,4}
- >>> s.symmetric_difference_update({1,3,5,7})
- >>> s
- {2, 4, 5, 7}

- s.remove(x)：从集合s中删除x，若x不存在，则引发KeyError错误。
- s.discard(x)：如果x是s的成员，则删除x。x不存在，也不出现错误。
- s.pop()：删除集合s中任意一个对象，并返回它。
- s.clear()：删除集合s中所有元素。

- 例：
- >>> s={1,2,3,4}
- >>> s.remove(3)
- >>> s
- {1, 2, 4}
- >>> s.remove(5)　#出现错误
- # KeyError: 5
- >>> s.discard(5)
- >>> s
- {1, 2, 4}

- 例：
- >>> s={1,2,3,4}
- >>> s.pop()
- 1
- >>> s
- {2, 3, 4}
- >>> t=s.pop() 2
- >>> t
- 2
- >>> s
- {3, 4}

- 例：
- >>> s={1,2,3}
- >>> s.clear()
- >>> s
- set()

7.4 字典与集合的应用

- 例：从键盘输入10个整数存入序列s中，其中凡相同的数在s中只存入第一次出现的数，其余的都被剔除。
- s=set()
- for i in range(10):
- 　　x=int(input("x="))
- 　　s.add(x)
- print("s=",s)

- 例：输入年、月、日，判断这一天是这一年的第几天。

```
year=int(input('请输入年份:'))
month=input('请输入月份:')
day=int(input('请输入日期:'))
dic={'1':31,'2':28,'3':31,'4':30,'5':31,'6':30,'7':31,\
    '8':31,'9':30,'10':31,'11':30,'12':31}
days=0
if ((year%4==0) and (year%100!=0)) or (year%400==0):
    dic['2']=29       #如果是闰年，则2月份是29天
if int(month)>1:
    for obj in dic:
        if month==obj:
            for i in range(1,int(obj)):
                days+=dic[str(i)]
    days+=day
else:
    days=day
print('{}年{}月{}日是该年的第{}天'.format(year,month,day,days))
```

自测题

- 一、选择题
- 1．Python语句print(type({1:1,2:2,3:3,4:4}))的输出结果是（　　）。B
- A．<class 'tuple'>　　B．<class 'dict'>
- C．<class 'set'>　　D．<class 'frozenset'>

自测题

- 一、选择题
- 1．Python语句print(type({1:1,2:2,3:3,4:4}))的输出结果是（　　）。
- A．<class 'tuple'>　　B．<class 'dict'>
- C．<class 'set'>　　D．<class 'frozenset'>

- 2．以下不能创建字典的语句是（　　）。
- A．dict1={}　　B．dict2={3:5}
- C．dict3=dict([2,5],[3,4])
- D．dict4=dict(([1,2],[3,4]))
- 3．对于字典D={'A':10,'B':20,'C':30,'D':40}，对第4个字典元素的访问形式是（　　）。D
- A．D[3]　　B．D[4]　　C．D[D]　　D．D['D']

- 2．以下不能创建字典的语句是（　　）。C
- A．dict1={}　　B．dict2={3:5}
- C．dict3=dict([2,5],[3,4])
- D．dict4=dict(([1,2],[3,4]))
- 3．对于字典D={'A':10,'B':20,'C':30,'D':40}，对第4个字典元素的访问形式是（　　）。D
- A．D[3]　　B．D[4]　　C．D[D]　　D．D['D']

- 4．对于字典D={'A':10,'B':20,'C':30,'D':40}，len(D)的是（　　）。
- A．4　　B．8　　C．10　　D．12
- 5．对于字典D={'A':10,'B':20,'C':30,'D':40}，sum(list(D.values()))的值是（　　）。B
- A．10　　B．100　　C．40　　D．200

- 4．对于字典D={'A':10,'B':20,'C':30,'D':40}，len(D)的是（　）。A
- A．4　　B．8　　C．10　　D．12
- 5．对于字典D={'A':10,'B':20,'C':30,'D':40}，sum(list(D.values()))的值是（　）。B
- A．10　　B．100　　C．40　　D．200

- 6．以下不能创建集合的语句是（　　）。
- A．s1=set()　　B．s2=set(“abcd”)
- C．s3={}　　D．s4=frozenset((3,2,1))
- 7．设a=set([1,2,2,3,3,3,4,4,4,4])，则a.remove(4)的值是（　　）。
- A．{1, 2, 3}　　B．{1, 2, 2, 3, 3, 3, 4, 4, 4}
- C．{1, 2, 2, 3, 3, 3}　　D．[1, 2, 2, 3, 3, 3, 4, 4, 4]

- 6．以下不能创建集合的语句是（　　）。C
- A．s1=set()　　B．s2=set(“abcd”)
- C．s3={}　　D．s4=frozenset((3,2,1))
- 7．设a=set([1,2,2,3,3,3,4,4,4,4])，则a.remove(4)的值是（　　）。A
- A．{1, 2, 3}　　B．{1, 2, 2, 3, 3, 3, 4, 4, 4}
- C．{1, 2, 2, 3, 3, 3}　　D．[1, 2, 2, 3, 3, 3, 4, 4, 4]

- 8．下列语句执行后的结果是（　　）。
- fruits={'apple':3,'banana':4,'pear':5}
- fruits['banana']=7
- print(sum(fruits.values()))
- A．7　　B．19　　C．12　　D．15

- 8．下列语句执行后的结果是（　　）。D
- fruits={'apple':3,'banana':4,'pear':5}
- fruits['banana']=7
- print(sum(fruits.values()))
- A．7　　B．19　　C．12　　D．15

- 9．下列语句执行后的结果是（　　）。
- d1={1:'food'}
- d2={1:'食品',2:'饮料'}
- d1.update(d2)
- print(d1[1])
- A．1　　B．2　　C．食品　　D．饮料

- 9．下列语句执行后的结果是（　　）。C
- d1={1:'food'}
- d2={1:'食品',2:'饮料'}
- d1.update(d2)
- print(d1[1])
- A．1　　B．2　　C．食品　　D．饮料

- 10．下列Python程序的运行结果是（　　）。
- s1=set([1,2,2,3,3,3,4])
- s2={1,2,5,6,4}
- print(s1&s2-s1.intersection(s2))
- A．{1, 2, 4}　　B．set()
- C．[1,2,2,3,3,3,4]　　D．{1,2,5,6,4}

- 10．下列Python程序的运行结果是（　　）。B
- s1=set([1,2,2,3,3,3,4])
- s2={1,2,5,6,4}
- print(s1&s2-s1.intersection(s2))
- A．{1, 2, 4}　　B．set()
- C．[1,2,2,3,3,3,4]　　D．{1,2,5,6,4}

- 二、填空题
- 1．在Python中，字典和集合都使用（　）作为定界符。字典的每个元素由两部分组成，即____和（　），其中（　）不允许重复。

- 二、填空题
- 1．在Python中，字典和集合都使用（　）作为定界符。字典的每个元素由两部分组成，即____和（　），其中（　）不允许重复。大括号，关键字，值，关键字

- 2．集合是一个无序、（　）的数据集，它包括和（　）两种类型，前者可以通过大括号或函数创建，后者需要通过（　）函数创建。

- 2．集合是一个无序、（　）的数据集，它包括和（　）两种类型，前者可以通过大括号或函数创建，后者需要通过（　）函数创建。
- 不重复，可变集合，不可变集合，set()，frozenset()

- 3．下列语句执行后，di['fruit'][1]的值是（　）。

  di={'fruit':['apple','banana','orange']}

- di['fruit'].append('watermelon')
- 4．语句print(len({}))的执行结果是（　）。

- 3．下列语句执行后，di['fruit'][1]的值是（　）。

  banana

- di={'fruit':['apple','banana','orange']}
- di['fruit'].append('watermelon')
- 4．语句print(len({}))的执行结果是（　）。0

- 5．设a=set([1,2,2,3,3,3,4,4,4,4])，则sum(a)的值是（　）。
- 6．{1,2,3,4} & {3,4,5}的值是（　），{1,2,3,4} | {3,4,5}的值是（　），{1,2,3,4} - {3,4,5}的值是（　）。

- 5．设a=set([1,2,2,3,3,3,4,4,4,4])，则sum(a)的值是（　）。10
- 6．{1,2,3,4} & {3,4,5}的值是（　），{1,2,3,4} | {3,4,5}的值是（　），{1,2,3,4} - {3,4,5}的值是（　）。{3, 4}，{1, 2, 3, 4, 5}，{1, 2}

- **7．设有s1={1,2,3}，s2={2,3,5}，则s1.update(s2)执行后，s1的值为（　），s1.intersection(s2)的执行结果为（　），s1.difference(s2)的执行结果为（　）。**

- **7．设有s1={1,2,3}，s2={2,3,5}，则s1.update(s2)执行后，s1的值为（　），s1.intersection(s2)的执行结果为（　），s1.difference(s2)的执行结果为（　）。**
- **{1, 2, 3, 5},{2, 3, 5},{1}**

- 8．下列程序的运行结果是（　）。
- d={1:'x',2:'y',3:'z'}
- del d[1]
- del d[2]
- d[1]='A'
- print(len(d))

- 8．下列程序的运行结果是（　）。2
- d={1:'x',2:'y',3:'z'}
- del d[1]
- del d[2]
- d[1]='A'
- print(len(d))

- 9．下面程序的运行结果是（　）。
- list1={}
- list1[1]=1
- list1['1']=3
- list1[1]+=2
- sum=0
- for k in list1:
- 　　sum+=list1[k]
- print(sum)

- 9．下面程序的运行结果是（　）。6
- list1={}
- list1[1]=1
- list1['1']=3
- list1[1]+=2
- sum=0
- for k in list1:
- 　　sum+=list1[k]
- print(sum)

- 10．下面程序的运行结果是（　）。
- s=set()
- for i in range(1,10):
- 　　s.add(i)
- print(len(s))

- 10．下面程序的运行结果是（　）。9
- s=set()
- for i in range(1,10):
- 　　s.add(i)
- print(len(s))

- 三、问答题
- 1．什么是空字典和空集合？如何创建？
- 2．设有列表a=['number','name','score']，b=['21001','denmer',90]，写一个语句将这两个列表的内容转换为字典，且以列表a中的元素为关键字，以列表b中的元素为值。

- 三、问答题
- 1．什么是空字典和空集合？如何创建？
- 2．设有列表a=['number','name','score']，b=['21001','denmer',90]，写一个语句将这两个列表的内容转换为字典，且以列表a中的元素为关键字，以列表b中的元素为值。
- dict(zip(a,b))

- 3．字典的遍历有哪些方法？
- 4．集合有哪两种类型？分别如何创建？
- 5．Python支持的集合运算有哪些？集合的比较运算有哪些？集合对象的方法有哪些？

- 6．分别写出下列两个程序的输出结果，输出结果为何不同？
- 程序一：
- d1={'a':1,'b':2}
- d2=d1
- d1['a']=6
- sum=d1['a']+d2['a']
- print(sum)

- 程序二：
- **d1={'a':1,'b':2}**
- **d2=dict(d1)**
- **d1['a']=6**
- **sum=d1['a']+d2['a']**
- **print(sum)**

- 程序二：
- d1={'a':1,'b':2}
- d2=dict(d1)
- d1['a']=6
- sum=d1['a']+d2['a']
- print(sum)
- 程序一结果为12，其中的语句d2=d1使得d1和d2指向（引用）相同的对象实例。程序二结果为7。其中的d2=dict(d1) 将产生一个新的实例赋给d2。