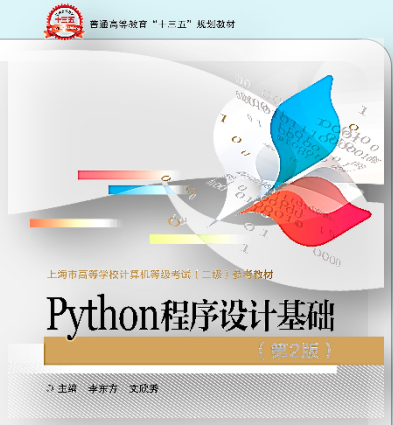
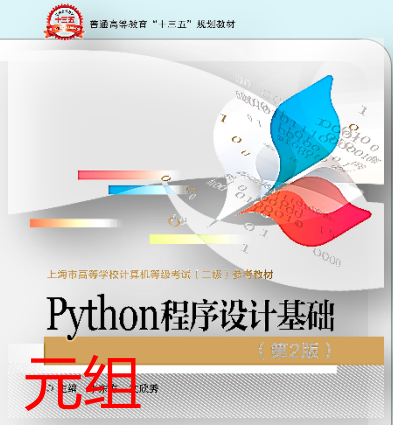
4.1.3 元组



- 元组（tuple）是Python中另一种内置的存储有序数据的结构

  元组是不可改变的



4.1.3 元组

- 创建元组

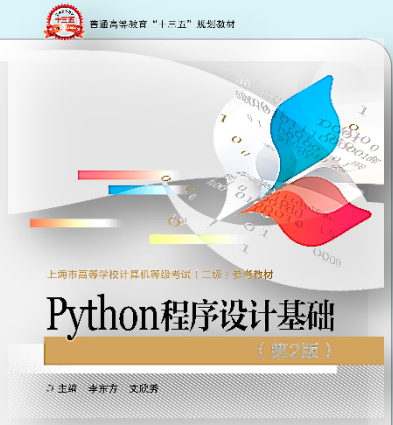
  如果用逗号分隔了一些值，那么将自动创建元组，元组通常用圆括号()括起来。换句话说，任意类型的对象如果以逗号隔开，则默认为元组。

```
>>> 1,2,3
(1, 2, 3)
>>> t="a", "b", "c", "d" #元组打包（tuple packing）
>>>t
('a', 'b', 'c', 'd')
```

注意，创建仅包括一个值的元组必须加个半角逗号

```
>>> 42,
(42,)
```

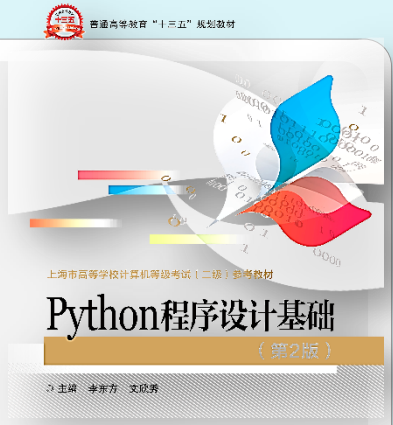
4.1.3 元组



- 【例4-9】 元组创建举例。

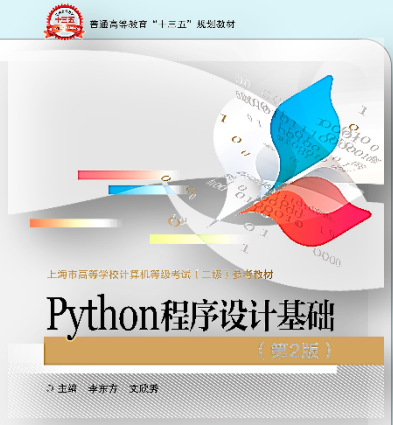
```
>>> tup1=(1,2,3)
(1, 2, 3)
>>>tup2= ('physics', 'chemistry', 1997, 2000)
('physics', 'chemistry', 1997, 2000)
>>> t = (12345, 54321, 'hello!')
>>> u = t, (1, 2, 3, 4, 5) # 元组允许嵌套
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> u[1][2]  #访问二维元组的元素
3
```

4.1.3 元组



- 【例4-9】 元组创建举例(续)

```
>>>v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
>>> v[1][1]
2
>>> v[1][1]=9
>>> v
([1, 2, 3], [3, 9, 1])
>>> tu = [(1, 2, 3), (3, 2, 1)]
>>> tu[1][1]=9
Traceback (most recent call last):
File "<pyshell#15>", line 1, in <module>
      tu[1][1]=9
TypeError: 'tuple' object does not support item assignment
```

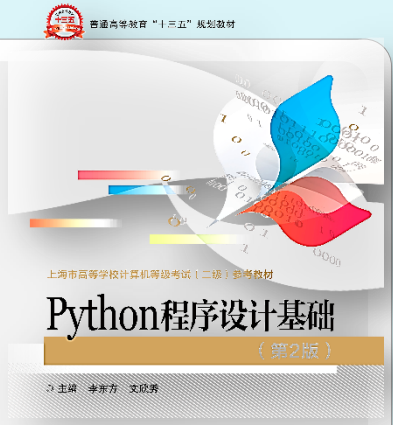


4.1.3 元组

- 元组的基本操作

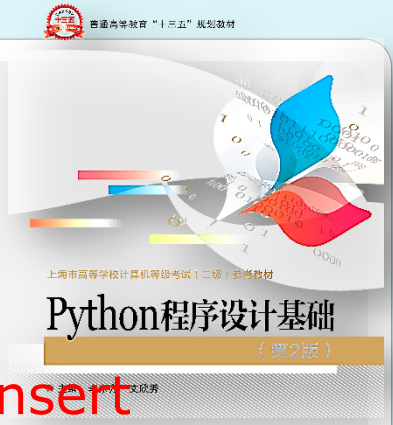
操　　作	含　　义
<tup>[i]	索引（求<tup>中位置索引为 i 的元素）
<tup>[i:j]	切片（求<tup>的位置索引为 i~j-1 的子元组）
<tup1>+<tup2>	将<tup1>和<tup2>连接
<tup>*<int-expr>或<int-expr>*<tup>	将<tup>复制<int-expr>次
len(<seq>)	求<tup>长度
for<var>in<tup>:	对<tup>中元素循环
<expr>in<tup>	查找<tup>中是否存在<expr>，返回值为布尔类型
del <tup>	删除元组
max(<tup>)	返回元组中最大值
min(<tup>)	返回元组中最小值

4.1.3 元组



- 元组的基本操作举例

```
>>>tup1 = ('physics', 'chemistry', 1997, 2000)
>>>tup2 = (1, 2, 3, 4, 5, 6, 7 )
>>> print ("tup1[0]: ", tup1[0])
tup1[0]:  physics
>>>print ("tup2[1:5]: ", tup2[1:5])
tup2[1:5]:  (2, 3, 4, 5)
>>> tup1 = (12, 34.56)
>>> tup2 = ('abc', 'xyz')
>>> print(tup1+tup2)
(12, 34.56, 'abc', 'xyz')
>>> del tup1
```



4.1.3 元组

由于元组是不可修改的，因此：

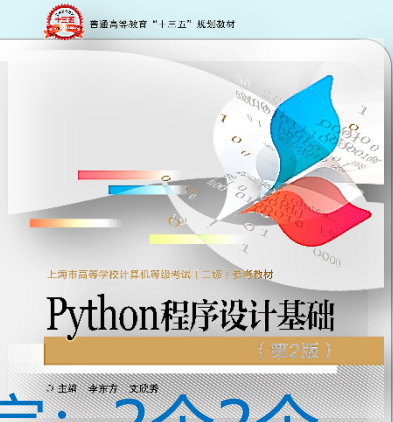
不能向元组增加元素，元组**没有**append、extend或insert方法。

不能从元组删除元素，元组**没有**remove、pop或clear方法。

元组**没有**sort、reverse方法。

```
>>>t=(2,23,41,3,7,1,10,48,5)
>>> 2 in t
True
>>> 4 in t
False
>>> max(t)
48
>>> min(t)
1
```

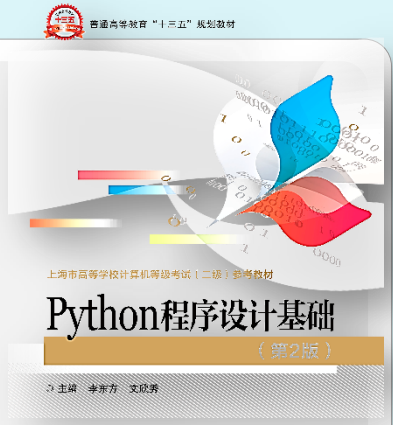
4.1.3 元组



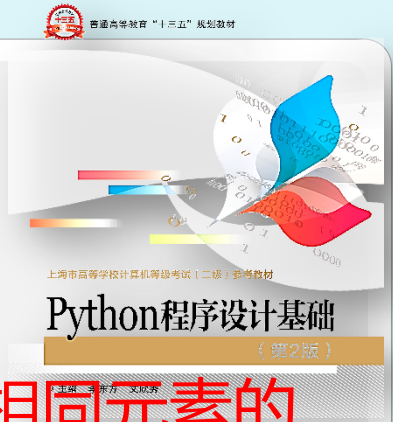
- 【例4-10】有一筐鸡蛋,1个1个拿，正好拿完；2个2个拿，还剩1个；3个3个拿，正好拿完；4个4个拿，还剩1个；5个5个拿，还差1个；6个6个拿，还剩3个；7个7个拿，正好拿完；8个8个拿，还剩1个；9个9个拿，正好拿完。问筐里最少有多少鸡蛋?

```
i=0
while True:
    if (i%2 == 1 and i%3 == 0 and i%4 == 1 and
    i%5 == 4 and i%6 == 3 and i%7 == 0 and
    i%8 == 1 and i%9 == 0):
        print ("i= ",i)
        break
    i+=1
```

4.1.3 元组




```
i=0
remain=(0,1,0,1,4,3,0,1,0)
while True:
    flag=1
    for j in range(1,10):
        if i%j!=remain[j-1]:
            flag=0
            break
    if flag==1:
        print ('i= ',i)
        break
    i+=1
```



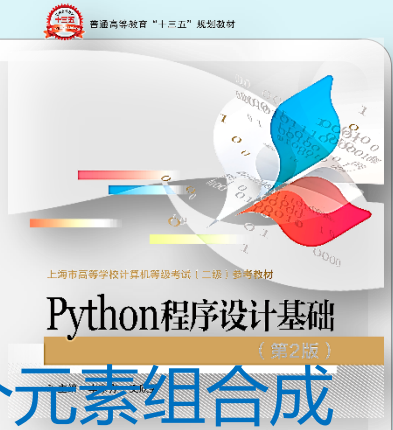
4.1.3 元组

- 元组与列表的相互转换

  tuple()函数接收一个列表，可返回一个包含相同元素的元组。*相当于冻结列表*

  list() 函数接收一个元组并返回一个列表。*相当于解冻元组*

```
>>>list1=[1,2,3]
>>>tup1=tuple(list1)
>>>tup1
(1,2,3)
>>>list(tup1)
[1,2,3]
```



4.1.3 元组

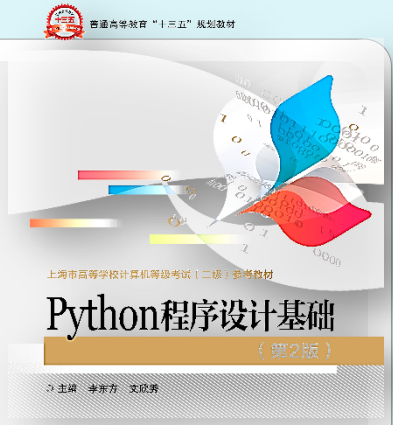
列表解析语句实现元组与列表的相互转换

【例4-11】 分别从两个列表中取不相等的两个元素组合成元组类型元素的新列表。

```
>>>[(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

```
combs = []
for x in [1,2,3]:
    for y in [3,1,4]:
        if x != y:
            combs.append((x, y))
print (combs)
```

```
[(x,y,x*y) for x in range(1,10) for y in range(1,10)]
```



4.1.3 元组

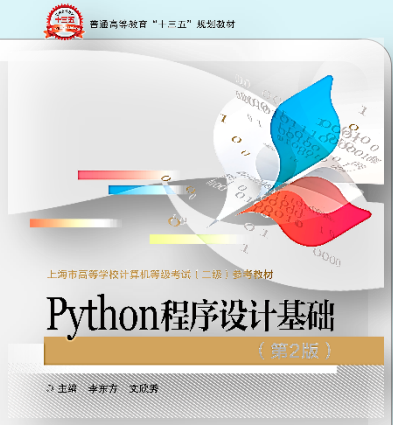
- 元组解包

t="a", "b", "c", "d"被称为元组打包

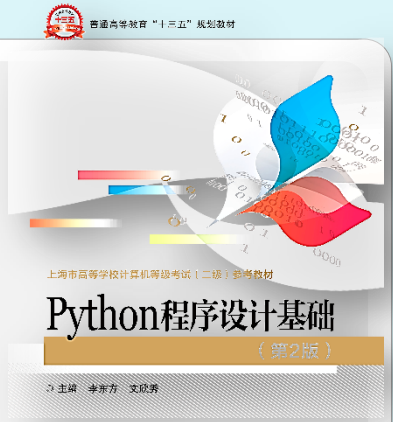
元组解包（unpacking），即将等号右侧元组中的元素按顺序依次赋给等号左边的变量。

```
>>>t=(1,2,3)
>>> a,b,c=t
>>> a
1
>>> b
2
>>> c
3
```

4.1.3 元组



在实际应用时，使用元组比使用列表的优势在于：首先，元组比列表的运算速度快。如果定义了一个常量集对象，并且需要在程序中不断地遍历它，则建议使用元组而不是列表。其次，使用元组相当于为不需要修改的数据进行了“写保护”，使得数据更安全。



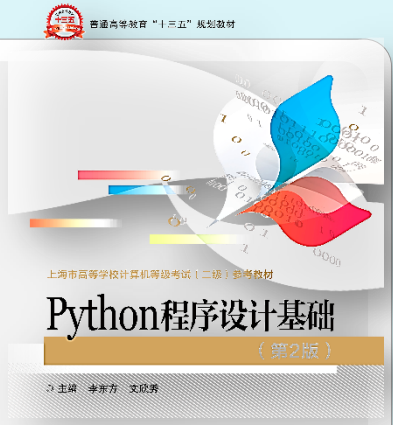
4.2 映射型组合数据——字典

- 创建字典

字典包含了一个索引的集合，称为键（key）和值（value）的集合。一个键对应一个值。这种一一对应的关联称为键值对（key-value pair），或称为项（item）。简单地说，字典就是用花括号包裹的键值对的集合。每个键值对用冒号“:”分隔，每对之间用逗号“,”分隔

d = {key1 : value1, key2 : value2 }

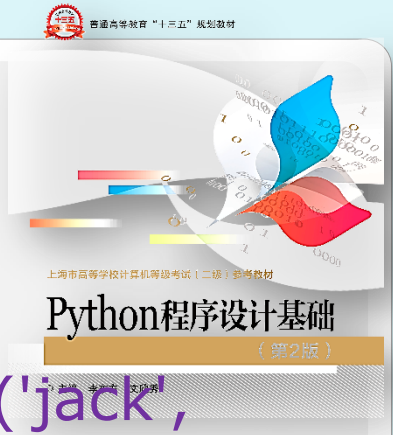
4.2 映射型组合数据——字典



- 创建字典举例

```
>>>dict1 = {'jack': 4098, 'sape': 4139}
>>> dict2 = {(1,2):['a','b'],(3,4):['c','d'],(5,6):['e','f']}
>>> dict2
{(1, 2): ['a', 'b'], (5, 6): ['e', 'f'], (3, 4): ['c', 'd']}
>>> type(dict2)
<class 'dict'>
>>>dict1 = {} #创建空字典
```

注意，**键必须是唯一的，必须是不可变数据类型的**，例如，字符串、数字或元组。值可以是任何数据类型。



4.2 映射型组合数据——字典

- dict()将列表或元组转换为字典

```
>>>dict1 = ([('sape',4139), ('guido',4127),('jack', 4098)])
>>>dict1
{'jack': 4098, 'sape':4139, 'guido':4127}
```

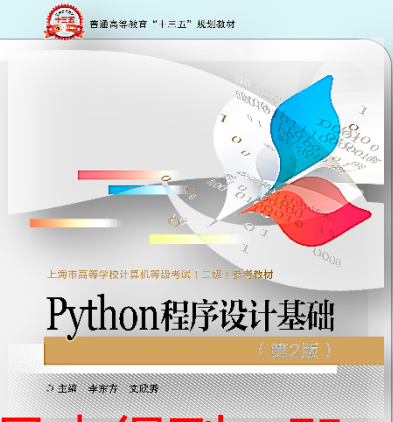
字典的数据结构是散列表，字典项是无序的，键值对的顺序可以随机变化，所以每次遍历输出的顺序可能不同。

- 通过关键字的形式创建字典

  键只能为字符串

```
>>> dict(name='allen',age='40')
{'name':'allen', 'age':'40'}
```

4.2 映射型组合数据——字典



- 访问字典中的值

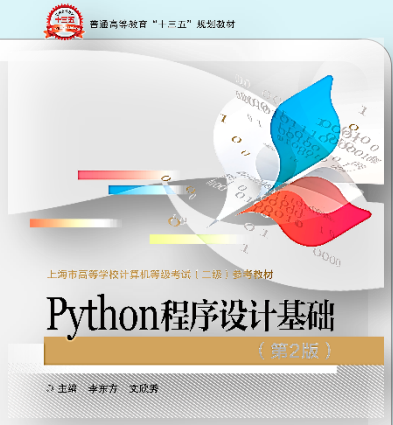
要得到字典中某个元素的值，可用字典键加上方括号来得到，即dict[key]形式返回键key对应的值value，如果key不在字典中，则会引发KeyError。

```
>>>dict={'name': 'earth', 'port': 80}
>>> dict
{'name': 'earth', 'port': 80}
>>> dict['port']
80
>>> dict['a']
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    dict['a']
KeyError: 'a'
```

若要检查字典dict中是否含有键key，可以使用in。

```
>>> d = {'name': 'alice'}
>>> 'name' in d
True
```

4.2 映射型组合数据——字典



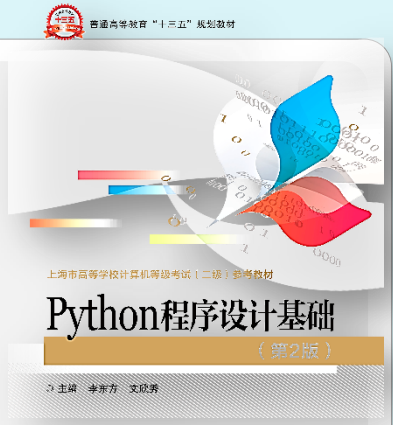
- 更新字典

可添加、删除或更新字典中的一个键值对

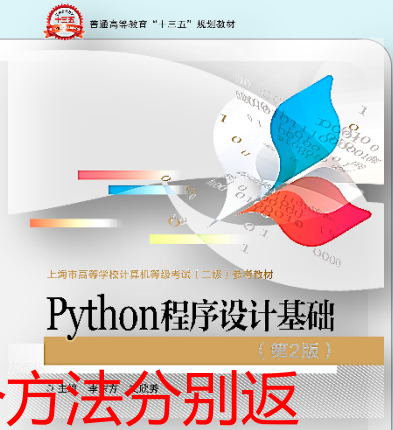
```
>>>adict={'name': 'earth', 'port': 80}
>>> adict['age']=18                    #添加新键值对
('age':18)
>>> adict
{'name': 'earth', 'port': 80, 'age': 18}
>>> adict['name']='moon'          #更新键'name'的值
>>> adict
{'name': 'moon', 'port': 80, 'age': 18}
>>> del adict['port']                  #删除键值对('port': 80)
>>> adict
{'name': 'moon', 'age': 18}
```

4.2 映射型组合数据——字典

- 字典的操作



字典对象的方法	含　义
dict.keys()	返回包含字典所有 key 的列表
dict.values()	返回包含字典所有 value 的列表
dict.items()	返回包含所有(键,值)项的列表
dict.clear()	删除字典中的所有项或元素，无返回值
dict.copy()	返回字典浅复制副本
dict.get(key,default=None)	返回字典中 key 对应的值，若 key 不存，则返回 default 的值（default 默认为 None）
dict.pop(key[,defalut])	若字典中存在 key，则删除并返回 key 对应的 value；如果 key 不存在，且没有给出 default 值，则引发 KeyError 异常
dict.setdefault(key,default=None)	若字典中不存在 key，则由 dict[key]=default 为其赋值
dict.update(adict)	将字典 adict 中键值对添加到 dict 中



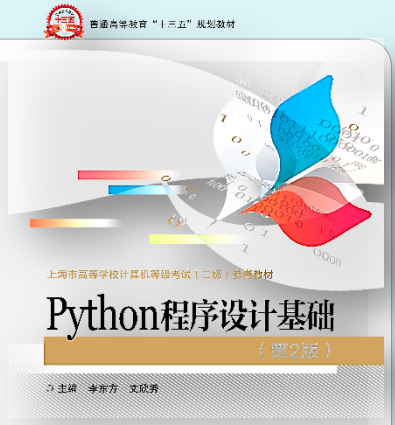
4.2 映射型组合数据——字典

- 返回字典所有的键、值和项

dict.keys()、dict.values()、dict.items()这三个方法分别返回包含原字典中每项的键、值和项(键,值)的列表

```
>>>d={'name':'alice','age':19,'sex':'F'}
>>> d.keys()
dict_keys(['age', 'sex', 'name'])
>>> d.items()
dict_items([('age', 19), ('sex', 'F'), ('name', 'alice')])
>>> d.values()
dict_values([19, 'F', 'alice'])
>>> for key in d.keys():    #遍历字典
        print ('key=%s, value=%s.' % (key,d[key]))
key=age, value=19.
key=sex, value=F.
key=name, value=alice.
```

4.2 映射型组合数据——字典



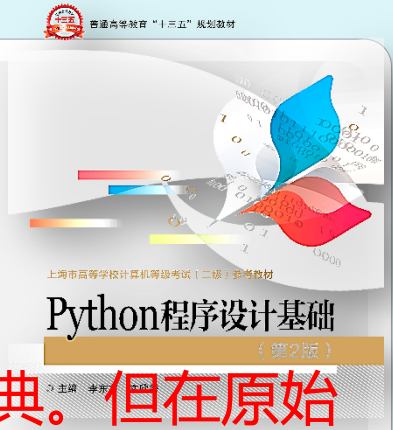
- 字典清空

用dict.clear()可清空原始字典中所有的元素。

对于两个相关联的字典对象x,y：若将x赋值为空字典，将不对y产生影响；而用clear方法清空x，也将清空字典y中所有元素。

```
>>> x={}
>>> y=x
>>> x['key']='value'
>>> y
{'key': 'value'}
>>> x={}
>>> y
{'key': 'value'}
```

```
>>> x = {}
>>> y = x
>>> x['key'] = 'value'
>>> y
{'key':'value'}
>>> x.clear()
>>> y
{}
```

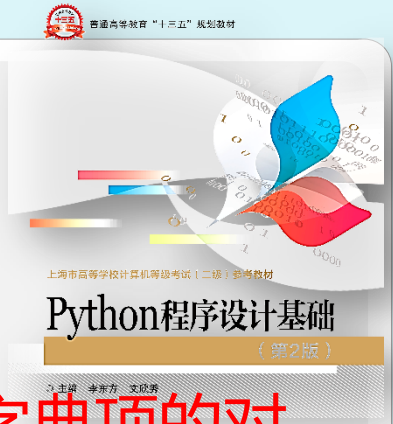


4.2 映射型组合数据——字典

- 字典的浅复制

dict.copy()方法返回一个具有相同键值对的新字典。但在原始字典中，如果修改某个值，副本字典也会相应改变，故称为浅复制（***shallow copy***，*仅复制字典对象直接包含的引用，不复制嵌套的对象*）。如果要避免该问题，可使用深复制（***deep copy***，*不仅复制字典对象，还要复制这个字典对象所引用的对象*）方法**deepcopy(dict)**。

```
x = {'a':[1],'b':[2,3,4]}
y = x.copy()
import copy
z = copy.deepcopy(x)
x['a'].append(5)
print (y)
print (z)
```

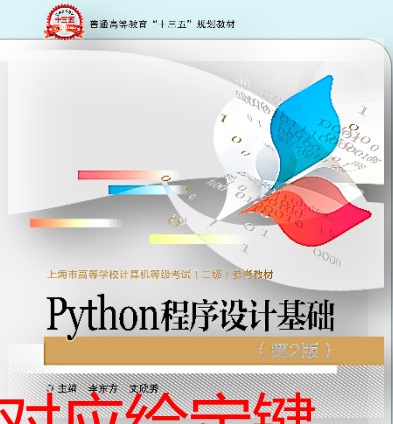


4.2 映射型组合数据——字典

- 以键查值

dict.get(key[, default=None])方法可访问字典项的对应值。若使用get访问一个不存在的key，会得到None值。还可以自定义默认值，替换None值。

```
>>> d = {}
>>> print (d.get('name'))
None
>>> d.get("name",'N/A')
'N/A'
>>> d[''name''] = 'Eric'
>>> d.get('name')
'Eric'
```



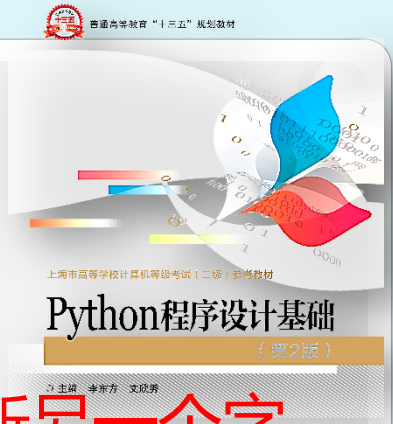
4.2 映射型组合数据——字典

- 移除键值对

dict.pop(key[,defalut])方法用来获得并返回对应给定键的值，然后将这个键值对从字典中移除。

```
>>> d={'name':'alice','age':19,'sex':'F'}
>>> d.pop('name')
'alice'
>>> d
{'age': 19, 'sex': 'F'}
```

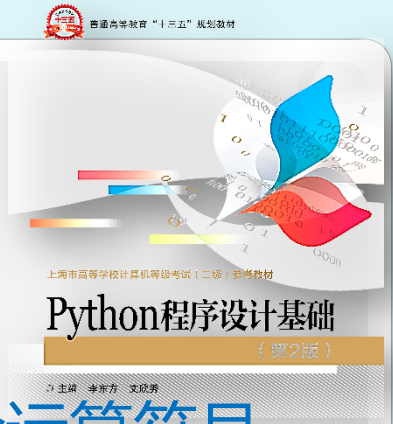
4.2 映射型组合数据——字典



- 字典更新

dict.update(adict)方法可以利用一个字典更新另一个字典。提供的字典中的所有键值对均会被添加到旧字典中，若有相同的键则会进行覆盖。

```
>>> d={'name':'alice','age':19,'sex':'F'}
>>> x={'name':'bob','phone':'12345678'}
>>> d.update(x)
>>> d
{'age': 19, 'phone': '12345678', 'sex': 'F', 'name': 'bob'}
```

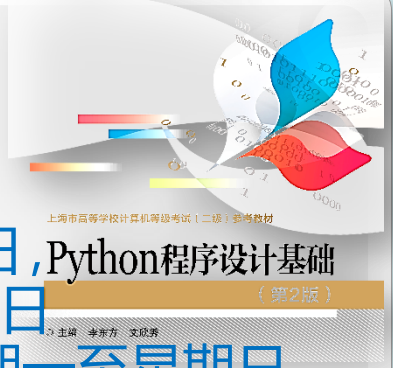


4.2 映射型组合数据——字典

- 【例4-13】 输入两个数字，并输入加减乘除运算符号，输出运算结果。若输入其他符号，则退出程序。

```
while True:
    a=float(input('请输入第一个数字'))
    b=float(input('请输入第二个数字'))
    t=input('请输入运算符号，其他符号为退出程序')
    tup=('+','-','*','/')
    if t not in tup:
        break
    dic={'+':a+b,'-':a-b,'*':a*b,'/':a/b}
    print('%s%s%s=%0.1f' %(a,t,b,dic.get(t)))
```

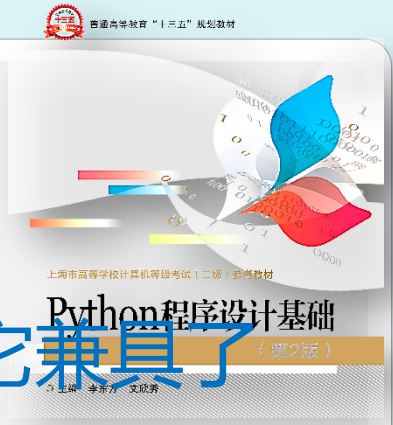
4.2 映射型组合数据——字典



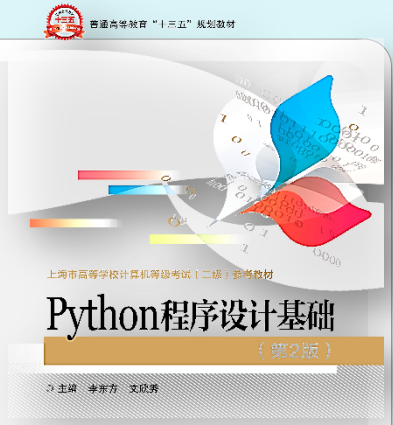
- 【例4-14】 引入内置模块calendar，输入年、月、日，根据weekday(year,month,day)的返回值，输出该日期是星期几。函数weekday()返回0～6分别对应星期一至星期日。

```
from calendar import *
y=input('请输入年')
m=input('请输入月')
d=input('请输入日')
dic={0:'星期一',1:'星期二',2:'星期三',3:'星期四',4:'星期五',\
5:'星期六',6:'星期日'}
if y.isdigit() and m.isdigit() \
   and d.isdigit() and 1<=int(m)<=12 \
   and 1<=int(d)<=31:
    w=weekday(int(y),int(m),int(d))
    print('您输入的%s年%s月%s日是%s' %(y,m,d,dic[w]))
else:
    print('输入日期有误')
```

4.3 集合型组合数据——集合



- 集合（set）是不重复元素的无序集，它兼具了列表和字典的一些性质。
- 集合有类似字典的特点：用花括号“{}”来定义，其元素是非序列类型的数据，也就是没有顺序，并且集合中的元素不可重复，也必须是不变对象，类似于字典中的键。集合的内部结构与字典很相似，区别是“只有键没有值”。
- 另一方面，集合也具有一些列表的特点：持有一系列元素，并且可原处修改。由于集合是无序的，不记录元素位置或者插入点，因此不支持索引、切片或其他类序列（sequence-like）的操作。

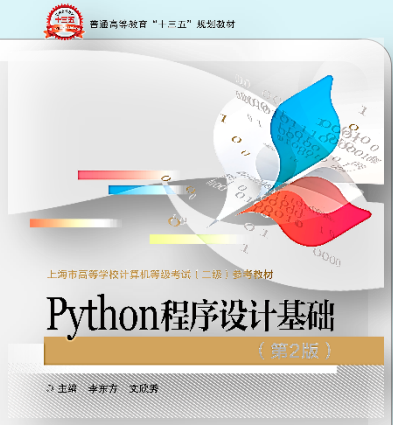


4.3 集合型组合数据——集合

- 集合的创建
  - 直接使用“{}”创建

```
>>> s3={1,2,3,4,5}
>>> s3
{1, 2, 3, 4, 5}
>>> s4=set() #注意创建空集合要用set()而非{}
>>> s4
set()
>>> type(s4)
<class 'set'>
>>> s5={}
>>> type(s5)
<class 'dict'>
```

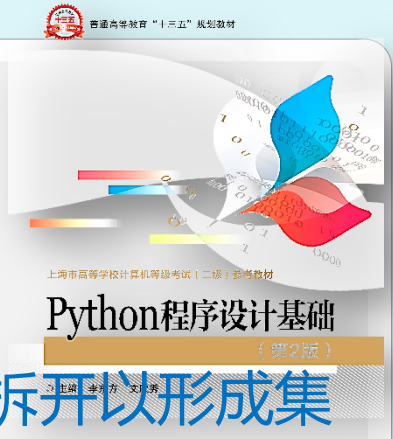
4.3 集合型组合数据——集合



- 集合的创建

```
>>> s5={'Python',(1,2,3)}
>>> s5
{'Python', (1, 2, 3)}
>>> s6={"Python",[1,2,3]}
Traceback (most recent call last):
  File "<pyshell#69>", line 1, in <module>
    s6={"Python",[1,2,3]}
TypeError: unhashable type: 'list'
>>> s7={"Python",{'name':'alice'}}
Traceback (most recent call last):
  File "<pyshell#70>", line 1, in <module>
    s7={"Python",{'name':'alice'}}
TypeError: unhashable type: 'dict'
```

从上面例子可以看出，通过“{}”无法创建含有列表或字典元素的集合。



4.3 集合型组合数据——集合

- 集合的创建
  - 由字符串创建。用函数set(str)将str中的字符拆开以形成集合。

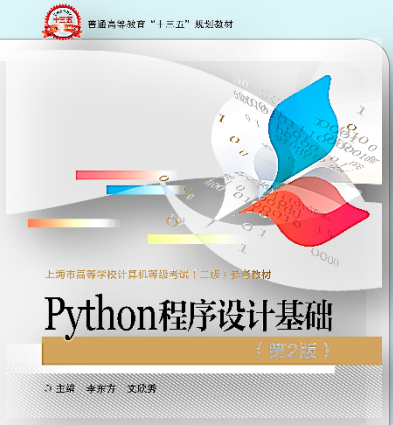
```
>>> s1=set('helloPython')
>>> s1
{'o', 'l', 't', 'y', 'h', 'e', 'p', 'n'}
```

  - 由列表或元组创建。用函数set(seq)创建集合，参数可以是列表或元组。

```
>>> s2=set([1,'name',2,'age','hobby'])
>>> s2
{'age', 1, 2, 'hobby', 'name'}
>>> s2=set((1,2,3))
>>> s2
{1, 2, 3}
```

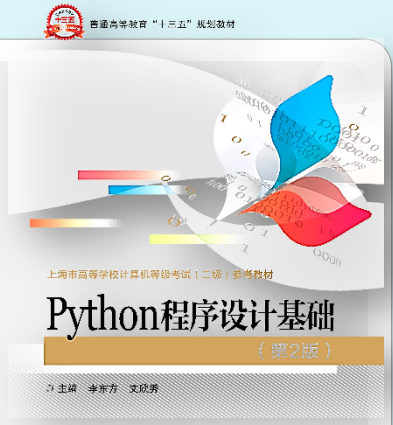
4.3 集合型组合数据——集合

- 集合的修改



集合对象的方法	含　义
set.add(x)	向集合中添加元素 x
set.update(a_set)	使用集合 a_set 更新原集合
set.pop()	删除并返回集合中的任意元素
set.remove(x)	删除集合中的元素 x，如果 x 不存在则报错
set.discard(x)	删除集合中的元素 x，如果 x 不存在则什么也不做
set.clear()	清空集合中的所有元素

4.3 集合型组合数据——集合



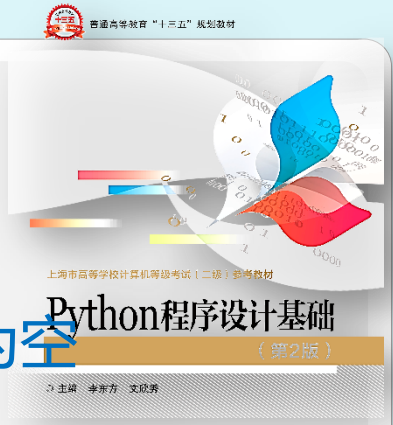
- set.add(x) 向集合set中添加元素x

```
>>> a_set={1,2}
>>> a_set.add("Python")
>>> a_set
{'Python', 1, 2}
```

- set.update(set1) 用集合set1更新set

```
>>> a_set={1,2,3}
>>> b_set={3,4,5}
>>> a_set.update(b_set)
>>> a_set
{1, 2, 3, 4, 5}
>>> b_set
{3, 4, 5}
```

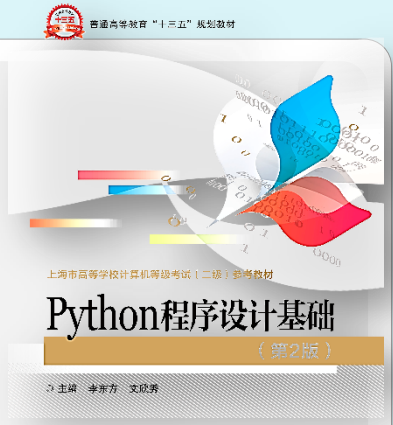
4.3 集合型组合数据——集合



- set.pop()
  从set中任意删除一个元素，不可指定，set不可为空

```
>>> a_set={1,2,3}
>>> a_set.pop()
1
>>> b_set=set()
>>> b_set.pop()
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    b_set.pop()
KeyError: 'pop from an empty set'
>>> a_set
{2, 3}
>>> a_set.pop(3)
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    a_set.pop(3)
TypeError: pop() takes no arguments (1 given)
```

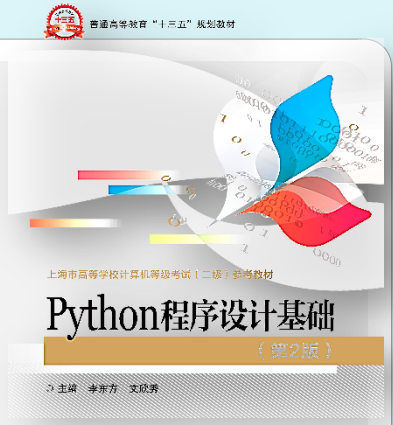
4.3 集合型组合数据——集合



- set.remove(x) 与set.discard(x)
  从set中删除元素x

```
>>> a_set=set("12345")
>>> a_set
{'3', '1', '5', '2', '4'}
>>> a_set.remove(4)
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    a_set.remove(4)
KeyError: 4
>>> a_set.remove('4')
>>> a_set.discard(5)
>>> a_set.discard("5")
>>> a_set
{'3', '1', '2'}
```

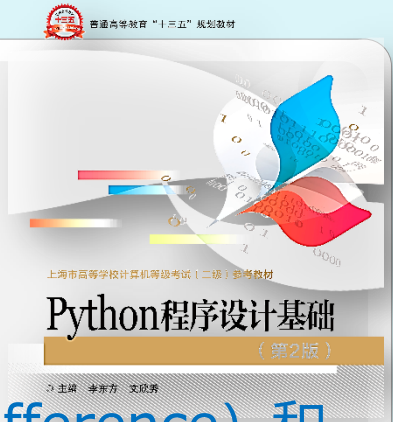
4.3 集合型组合数据——集合



- set.clear()
  删除set中所有元素

```
>>> a_set
{'3', '1', '2'}
>>> a_set.clear()
>>> a_set
set()
```

4.3 集合型组合数据——集合

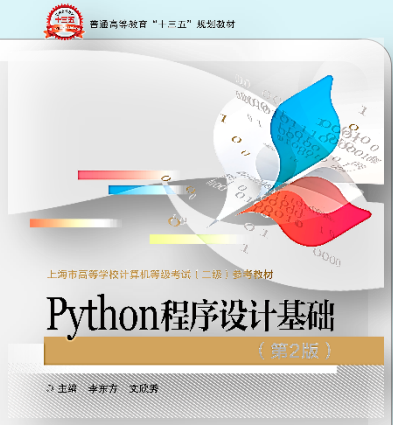


- 集合的数学运算

集合支持联合（Union）、交（Intersection）、差（Difference）和对称差集（Symmetric Difference）等数学运算

Python 符号	集合对象的方法	含　义
s1 & s2	s1.intersection(s2)	返回 s1 与 s2 的交集
s1 | s2	s1.union(s2)	返回 s1 与 s2 的并集
s1 - s2	s1.difference(s2)	返回 s1 与 s2 的差集
s1^s2	s1.symmetric_difference(s2)	返回 s1 与 s2 的对称差
x in s1		测试 x 是否是 s1 的成员
x not in s1		测试 x 是否不是 s1 的成员
s1<=s2	s1.issubset(s2)	测试是否 s1 是 s2 的子集
s1>=s2	s1.issuperset(s2)	测试是否 s1 是 s2 的超集
	s1.isdisjoint(s2)	测试 s1 和 s2 是否有交集
s1|= s2	s1.update(s2)	用 s2 更新 s1

4.3 集合型组合数据——集合



- 【例4-16】集合运算举例

```
>>> s1={'a','e','i','o','u'}
>>> s2={'a','b','c','d','e'}
>>> s1|s2
{'e', 'o', 'c', 'u', 'i', 'd', 'a', 'b'}
>>> s3=s1&s2
>>> s3
{'a', 'e'}
>>> s3.issubset(s1)        #s3<=s2
True
>>> s1.issuperset(s3)      #s1>=s3
True
>>> s1.difference(s2)
{'u', 'i', 'o'}
>>> s1-(s1&s2)     #s1-s2
{'u', 'i', 'o'}
>>> s1.symmetric_difference(s2)
{'i', 'o', 'd', 'c', 'u', 'b'}
>>> (s1|s2)-(s1&s2)     #s1^s2
{'o', 'c', 'u', 'i', 'd', 'b'}
```