复杂数据类型

王肖燕

内容介绍

一．组合数据类型

- ✓ 字符串(str)、元组(tuple)、列表(list)、集合(set)、字典(dict)

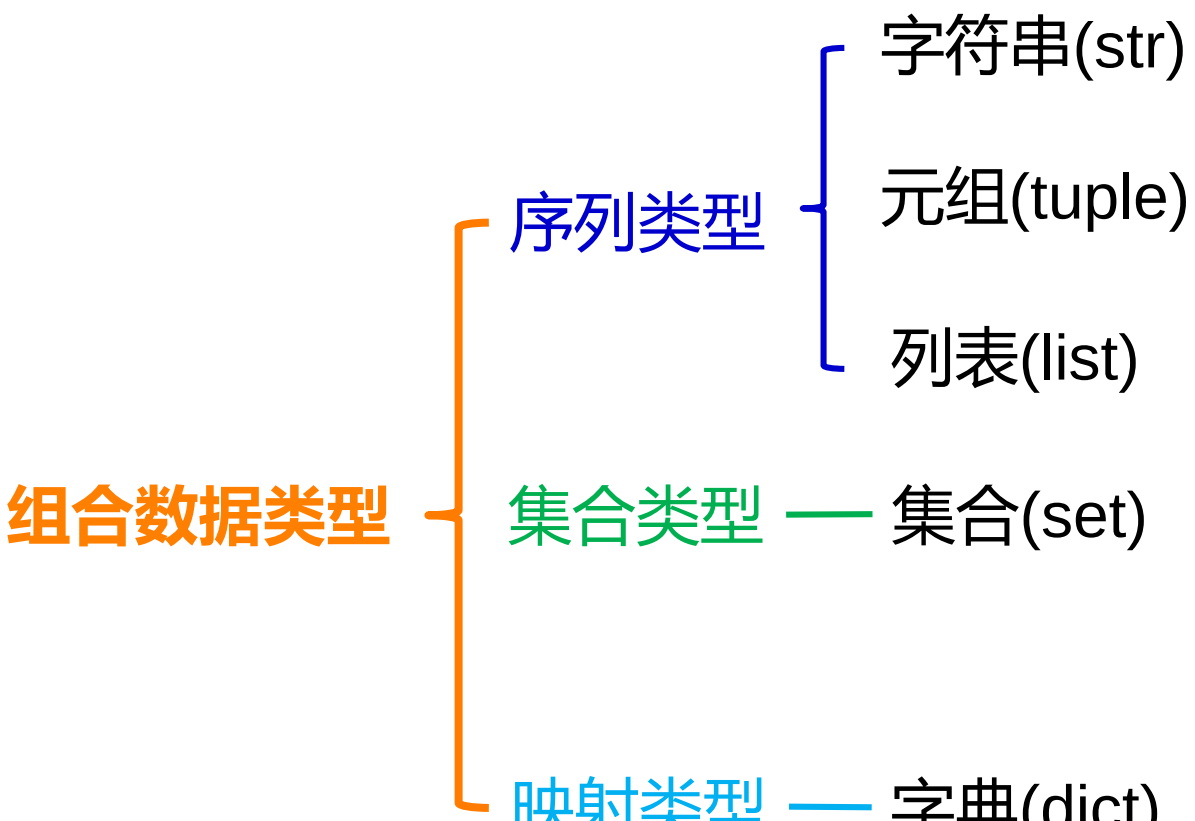
二．for循环结构

- ✓ for语句、else扩展模式

三．列表推导式

四．应用举例

强大的组合数据类型



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

集合与数学中的概念一致，即包含0个或多个数据项的无序集合。元素不可重复，元素类型只能是固定数据类型

映射类型是“键-值”数据项的组合，每个元素是一个键值对，即(key, value)，元素之间是无序的

字符串

- 使用双引号或单引号引起来的东西
- 是单个字符的有序组合——序列类型
- 由于字符串类型十分常用且单一字符串只表达一个含义，也被看作是基本数据类型
- 字符串的索引方法：
  - 正向递增序号：s[0], s[1], s[2] …….
  - 反射递减序号：s[-1], s[-2], s[-3] ……

字符串的操作

操作	含义
+	连接
*	重复
<string>[ ]	索引
<string>[:]	剪切
len(<string>)	长度
<string>.upper()	字符串中字母大写
<string>.lower()	字符串中字母小写
<string>.strip()	去两边空格及去指定字符
<string>.split()	按指定字符分割字符串为数组
<string>.join()	连接两个字符串序列
<string>.find()	搜索指定字符串
<string>.replace()	字符串替换
for <var> in <string>	字符串迭代

```
>>> s1="abcde"
>>> s2="fghij"
>>> s1+s2
'abcdefghij'
>>> s1*3
'abcdeabcdeabcde'
>>> s1[0]
'a'
>>> s1[0:3]
'abc'
>>> len(s1)
5
>>> s1.upper()
'ABCDE'
>>> "  abc   ".strip()
'abc'
>>> "abacda".strip('a')
'bacd'
>>> "a b c d".split()
['a', 'b', 'c', 'd']
>>> "a,b,c,d".split(',')
['a', 'b', 'c', 'd']
```

```
>>> '_'.join('Python')
'P_y_t_h_o_n'
>>> ' '.join(['Very','Good'])
'Very Good'
>>> s1.find("d")
3
>>> s1.replace("c","666")
'ab666de'
>>> s1
'abcde'
```

元组

- **形式**：采用**逗号**和**圆括号（可选）**来表示
  - >>> t1=(1,2,3,4,5) 或 t1=1,2,3,4,5
  - >>> t2=('boy','girl','apple')
  - >>> t3=(1,2,3,4,'abc')
- **特性**：**不可变**序列类型
  - 一旦生成，不可替换和删除
  - 只能引用，可使用的索引体系：
    - 正向递增序号：t1(0), t1(1), t1(2), t1(3), t1(4)
    - 反射递减序号：t1(-1), t1(-2), t1(-3), t1(-4), t1(-5)
  - 使用tuple()可以把字符串转换成元组

元组的标志性符号

- 小括号？NO
- 逗号(,)才是关键，小括号只起到补充的作用
  - 创建空元组
    - >>> t = ()
  - 创建的元组只有一个元素，在它后边加上逗号
    - >>> t = (1,)
    - >>> tt = 1,
  - 逗号作用的体现，下述运算有什么不同?
    - >>> 8*(8)
    - >>>8*(8,)

列表

- **形式**：采用**逗号**和**方括号**来表示
  - >>> L1=[1, 2, 3, 4, 5]
  - >>> L2=[‘boy’, ’girl’, ’apple’]
  - >>> L3=[1, 2, 3, 4, ’abc’]
  - >>> L4=[(1,2,3), (4,5,6), 1, 2, 3, ”abc”]
- **特性**：**可变**序列类型
  - 替换、删除和增加
  - 索引体系：正向递增序号或反射递减序号
  - 可通过list()函数将元组或字符串转化成列表

序列类型的通用操作符和函数

操作符	描述
x in s	如果x是s中的元素，返回True，否则返回False
x not in s	如果x不是s中的元素，返回True，否则返回False
s+t	连接s和t
s*n或n*s	将序列s重复n次
s[i]	索引，返回序列的第i个元素
s[i:j]	分片，返回s的子序列，包含s中第i到j-1的元素
s[i:j:k]	返回s的子序列，包含s中第i到j-1的元素以k为步数
len(s)	序列s的长度（元素个数）
min(s)	序列s中最小的元素
max(s)	序列s中最大的元素
s.index(x[,i][,j])	序列s中从i开始到j位置中第一次出现元素x的位置
s.count(x)	序列s中出现x的总次数

操作和函数举例

```
>>> l1=[1,2,3,4,5,1,1,2]
>>> l2=['a','b','c','d','e','d',10]
>>> l1[1:5:2]
[2, 4]
>>> max(l1)
5
>>> min(l2)
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    min(l2)
TypeError: unorderable types: int() < str()
```

```
>>> l2.index('d')
3
>>> l2.index('d',4,6)
5
>>> l1.count(1)
3
>>> l2.count('d')
2
```

列表类型特有的函数和方法

函数或方法	描述
ls[i]=x	替换列表ls第i个数据项为x
ls[i:j]=lt	用列表lt替换列表ls中第i到第j-1项数据
ls[i:j:k]=lt	用列表lt替换列表ls中第i到第j-1项数据以k为步长
del ls[i:j]	删除列表ls中第i到第j-1项数据
del ls[i:j:k]	删除列表ls中第i到第j-1项数据以k为步长
ls+=lt或ls.extend(lt)	将lt中的元素增加到列表ls中
ls*=n	更新列表ls，其元素重复n次
ls.append(x)	在列表ls最后增加一个元素x
ls.clear()	删除ls中所有的元素
ls.copy()	生成一个新列表，复制ls中所有的元素——浅复制
ls.insert(i,x)	在列表ls第i个位置增加元素x
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ls中出现的第一个元素x删除
ls.reverse()	将列表ls中的元素反转

切片替换操作

```
>>> list1=[1,2,3,4,5]
>>> list1[1]=6
>>> list1
[1, 6, 3, 4, 5]
>>> lt=['a','b']
>>> list1[1:4:2]=lt
>>> list1
[1, 'a', 3, 'b', 5]
```

```
>>> lt=['a','b','c','d']
>>> list1=[1,2,3,4,5]
>>> list1[1:4:2]=lt
Traceback (most recent call last):
  File "<pyshell#51>", line 1, in <module>
    list1[1:4:2]=lt
ValueError: attempt to assign sequence of size 4 to
```

列表复制操作

```
>>> ls=[1,2,3,4,5]
>>> lt=ls
>>> lt
[1, 2, 3, 4, 5]
>>> lt[1]='a'
>>> ls
[1, 'a', 3, 4, 5]
```

```
>>> ls=[1,2,3,4,5]
>>> lt=ls
>>> id(ls)
58496008
>>> id(lt)
58496008
```

```
>>> ls=[1,2,3,4,5]
>>> lt=ls.copy()
>>> lt
[1, 2, 3, 4, 5]
>>> lt[1]='a'
>>> lt
[1, 'a', 3, 4, 5]
>>> ls
[1, 2, 3, 4, 5]
>>> id(ls)
58495432
>>> id(lt)
58433352
```

看清区别！

集合

- **形式**：采用**逗号**和**大括号**来表示
  - >>> S1={1, 2, 3, 4, 5}
  - >>> S2={‘boy’, ’girl’, ’apple’}
  - >>> S3={1, 2, 3, 4, ’abc’}
  - >>> S4={(1,2,3), (4,5,6), 1, 2, 3, ”abc”}
- **特性**：**无序，可变**集合类型
  - 集合中只**能**包含数字、字符串、元组等不可变类型的数据，**不能**包含列表、字典、集合等可变类型的数据
  - 替换、删除和增加
  - 可通过set()函数将元组、字符串或列表转化成集合

集合类型的操作符

操作符	描述
S-T或S.difference(T)	
S-=T或S.difference_update(T)	
S&T或S.intersection(T)	
S&=T或 S.intersection_update(T)	
S^T或 s.symmetric_difference(T)	
S^=T 或 S.symmetric_difference_update()	
S|T或S.union(T)	
S|=T或S.update(T)	
S<=T或S.issubset(T)	
S>=T或S.issuperset(T)	

集合类型的操作符

表 6.2　集合类型的操作符（共 10 个）

操　作　符	描　　述
S－T 或 S.difference(T)	返回一个新集合，包括在集合 S 中但不在集合 T 中的元
S-=T 或 S.difference_update(T)	更新集合 S，包括在集合 S 中但不在集合 T 中的元素
S & T 或 S.intersection(T)	返回一个新集合，包括同时在集合 S 和 T 中的元素
S&=T 或 S.intersection_update(T)	更新集合 S，包括同时在集合 S 和 T 中的元素
S^T 或 s.symmetric_difference(T)	返回一个新集合，包括集合 S 和 T 中的元素，但不包括同时在其中的元素
S=^T 或 s.symmetric_difference_update(T)	更新集合 S，包括集合 S 和 T 中的元素，但不包括同时在其中的元素
S|T 或 S.union(T)	返回一个新集合，包括集合 S 和 T 中的所有元素
S=|T 或 S.update(T)	更新集合 S，包括集合 S 和 T 中的所有元素
S<=T 或 S.issubset(T)	如果 S 与 T 相同或 S 是 T 的子集，返回 True，否则返回 False，可以用 S<T 判断 S 是否是 T 的真子集
S>=T 或 S.issuperset(T)	如果 S 与 T 相同或 S 是 T 的超集，返回 True，否则返回 False，可以用 S>T 判断 S 是否是 T 的真超集

上述操作符表达了集合类型的 4 种基本操作：交集（&）、并集（|）、差补集（^），操作逻辑与数学定义相同，如图 6.3 所示。

集合类型的操作方法

操作函数或方法	描述
S.add(x)	
S.clear()	
S.copy()	
S.pop()	
S.discard(x)	
S.remove(x)	
S.isdisjoint(T)	
len(S)	
x in S	
x not in S	

集合类型的操作方法

表 6.3　集合类型的操作函数或方法（共 10 个）

操作函数或方法	描　　述
S.add(x)	如果数据项 x 不在集合 S 中，将 x 增加到 s
S.clear()	移除 S 中的所有数据项
S.copy()	返回集合 S 的一个副本
S.pop()	随机返回集合 S 中的一个元素，如果 S 为空，产生 KeyError 异
S.discard(x)	如果 x 在集合 S 中，移除该元素；如果 x 不在集合 S 中，不
S.remove(x)	如果 x 在集合 S 中，移除该元素；不在则产生 KeyError 异常
S.isdisjoint(T)	如果集合 S 与 T 没有相同元素，返回 True
len(S)	返回集合 S 的元素个数
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

字典

- **形式**：采用**逗号**和**大括号**来表示
  - >>> d1={1:'a', 2:'b', 3:'c', 4:'d', 5:'e'}
  - >>> d2={'中国':'北京', '美国':'华盛顿', '法国':'巴黎'}
  - >>> d3={'170101':'张三', '170102':'李四', '170103':'王二'}
- **特性**：**无序，可变**集合类型
  - 包含若干"键:值"元素的无序可变序列
  - 每个元素包含用冒号分隔开的"键"和"值"两部分，表示一种映射或对应关系
  - "键"只能是不可变数据，整数、实数、字符串元组等
  - "键"不允许重复，"值"可以重复

字典类型的函数和方法

函数和方法	描述
dd.keys()	返回字典dd中所有的键信息
dd.values()	返回字典dd中所有的值信息
dd.items()	返回字典dd中所有的键值对
dd.get(key[, default])	键存在则返回相应值，否则返回默认值
dd.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
dd.popitem()	随机从dd中选择一个键值对，以元组(key, value)形式返回
dd.clear()	删除字典dd中所有的键值对
del dd[key]	删除字典中关键字为key的键值对
key in dd	Key是字典的键则返回True，否则返回False

字典类型的函数和方法

函数和方法	描述
dd.keys()	返回字典dd中所有的键信息
dd.values()	返回字典dd中所有的值信息
dd.items()	返回字典dd中所有的键值对
dd.get(key[, default])	键存在则返回相应值，否则返回默认值
dd.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
dd.popitem()	随机从dd中选择一个键值对，以元组(key, value)形式返回
dd.clear()	删除字典dd中所有的键值对
del dd[key]	删除字典中关键字为key的键值对
key in dd	Key是字典的键则返回True，否则返回False

for循环

- for <循环变量> in <遍历结构>:

  <语句块>

- for语句的循环次数
  - 根据遍历结构中的元素个数确定的
- 执行过程：从遍历结构中逐一提取元素，放在循环变量中，对于所提取的每个元素执行一次语句块
- 遍历结构：可以是组合数据类型

for循环遍历组合类型

字符串:

```
s="abcdef"
for e in s:
    print(e,end=",")
```

列表:

```
l1=[1,2,3,4,5]
sum=0
for e in l1:
    sum+=e
print(sum)
```

集合:

```
set1={1,2,3,4,5,1}
sum=0
for e in set1:
    sum+=e
print(sum)
```

元组:

```
t1=(1,2,3,'a',5)
for e in t1:
    print(e,end=",")
```

字典的四种遍历方法:

```
d={'a':"China",'b':"USA",'C':"France"}
for k in d.keys():
    print(k,end=",")
for v in d.values():
    print(v,end=",")
for kv in d.items():
    print(kv)
for k,v in d.items():
    print(k+':'+v)
```

else扩展

- for循环的else扩展模式

```
for <循环变量> in <遍历结构> :
    语句块
else:
    print("循环正常结束")
```

- else语句只在循环正常执行后才执行

列表推导式1

- 使用非常简洁的方式对列表或其他可迭代对象的元素进行遍历、过滤或再次计算，快速生成满足特定需求的新列表
- 代码简洁、可读性强
- 语法

```
[表达式  for e1 in 序列1 if c1
         for e2 in 序列2 if c2
           …
         for en in 序列n if cn]
```

- 逻辑上等价于一个循环语句

列表推导式2

- 嵌套列表的平铺
  - L1=[[1,2,3],[4,5,6],[7,8,9]]
  - [num for e in L1 for num in e]
- 过滤不符合条件的元素
  - L2=[1,-2,3,4,-5,6,0,8,-9,10]
  - [i for i in L2 if i>0]
- 同时遍历多个可迭代对象
  - [(x,y) for x in [1,2,3] for y in [3,1,4] if x!=y]

列表推导式3

- 使用函数或复杂表达式
  - 使用函数

```
def f(v):
    if v%2==0:
        v=v**2
    else:
        v=v+1
    return v
L1=[2,-2,3,4,-1]
r=[f(e) for e in L1 if e>0]
print(r)
```

  - 复杂表达式

```
L1=[2,-2,3,4,-1]
r=[e**2 if e%2==0 else e+1 for e in L1 if e>0]
print(r)
```

使用字典统计频次

```
 8 import string
 9 import random
10 x=string.ascii_letters+string.digits
11 ss=""
12 i=0
13 while i<1000:
14     ss=ss+random.choice(x)
15     i=i+1
16 d=dict()
17 for ch in ss:
18     d[ch]=d.get(ch,0)+1
19 for k,v in sorted(d.items()):
20     print(k,':',v)
```

Counter类

- 对于频次统计问题，使用collection模块的Counter类可以更加快捷地实现这个功能
- 提供更多的功能，如：查找出现次数最多的元素

```
38 from collections import Counter
39 fs=Counter(ss)
40 fs.items()
41 fs.most_common(1)
42 fs.most_common(3)
```

应用举例1

- 求元组中元素的平均值并打印，score = (70, 90, 78, 85, 97, 94, 65, 80)。

  ① 使用for循环

  ② 使用while循环

  ③ 使用函数sum

应用举例2

- scoreDict = {‘01’:70, ‘02’:90, ‘03’:78, ‘09’:85, ‘05’:97, ‘06’:94, ‘10’:65, ‘08’:80}
  - 计算字典中分数的平均值并打印
  - 统计哪些学号参加了考试，放入列表num中

应用举例3

① 列表['191023', '191024'] 、[90, 79]转换为字典{'191023': 90, '191024': 79}

② 数字加密，输入一个四位整数，加密规则为每位数字都加上5,然后用其除以10的余数代替该数字，再将第一位和第四位交换

③ 把列表[‘Brazil’, ‘Russia’, ‘India’, ‘China’]中的字符串使用逗号连接在一起

应用举例4

① 在字典中{"li":18, "wang":50, "zhang":20, "sun":22, "zhao":34, "qian":44}找到年龄最大的人，并输出其姓名

② 去除列表[1,2,3,4,2,2,3,4,5,3]中的重复的元素

应用举例5

- 找出列表[9,1,7,8,11,12,3,5,3,9,12,4,6,12]中的最大值，并输出所有最大值出现的位置。
  - 找到最大值，输出
  - 在列表中寻找所有的最大值，并输出其位置

应用举例6

① 判断今天是今年的第几天?

- ✓ import time
- ✓ date=time.localtime() #获取当前日期时间
- ✓ year,month,day=date[:3] #取年月日

② 输入任意日期，格式（2019/06/10），判断这个日期是这一年的第几天?

应用举例7

- **阿凡提与国王比赛下棋，国王说要是自己输了，阿凡提要什么都可以拿给他。阿凡提说那就要点米吧，要求如下：**
  - ① 棋盘一共64个小格子；
  - ② 在第一个格子里放1粒米，第二个格子里放2粒米，第三个格子里放4粒米，第四个格子里放8粒米；
  - ③ 以此类推，后面每个格子里的米都是前一个格子里的2倍，一直把64个格子都放满
  - ➢ 一共需要多少粒米?
  - ➢ 一斤按约26000粒计算，大概需要多少吨大米?

Thank You!