第四章 组合数据类型

4.1 列表

4.1.1 列表的表达

- 序列类型：内部元素有位置关系，能通过位置序号访问其中元素
- 列表是一个可以使用多种类型元素，支持元素的增、删、查、改操作的序列类型

In [2]:

```
ls = ["Python", 1989, True, {"version": 3.7}]
ls
```

Out[2]:

```
['Python', 1989, True, {'version': 3.7}]
```

- 另一种产生方式：list(可迭代对象)
- 可迭代对象包括：字符串、元组、集合、range()等

字符串转列表

In [2]:

```
list("人工智能是未来的趋势")
```

Out[2]:

```
['人', '工', '智', '能', '是', '未', '来', '的', '趋', '势']
```

元组转列表

In [3]:

```
list(("我", "们", "很", "像"))
```

Out[3]:

```
['我', '们', '很', '像']
```

集合转列表

In [4]:

```
list({"李雷", "韩梅梅", "Jim", "Green"})
```

Out[4]:

['Green', 'Jim', '李雷', '韩梅梅']

特殊的range()

In [5]:

```
for i in [0, 1, 2, 3, 4, 5]:
    print(i)
```

0
1
2
3
4
5

In [6]:

```
for i in range(6):
    print(i)
```

0
1
2
3
4
5

- range(起始数字,中止数字,数字间隔)

如果起始数字缺省，默认为0

必须包含中止数字

数字间隔缺省，默认为1

In [7]:

```
for i in range(1, 11, 2):
    print(i)
```

1
3
5
7
9

- range()转列表

In [8]:

```
list(range(1, 11, 2))
```

Out[8]:

[1, 3, 5, 7, 9]

4.1.2 列表的性质

- 列表的长度——len(列表)

In [11]:

```
ls = [1, 2, 3, 4, 5]
len(ls)
```

Out[11]:

5

- 列表的索引——**与同为序列类型的字符串完全相同**

变量名[位置编号]

正向索引从0开始
反向索引从-1开始

In [13]:

```
cars = ["BYD", "BMW", "AUDI", "TOYOTA"]
```

In [14]:

```
print(cars[0])
print(cars[-1])
```

BYD
TOYOTA

- 列表的切片

变量名[开始位置：结束位置：切片间隔]

In [25]:

```
cars = ["BYD", "BMW", "AUDI", "TOYOTA"]
```

- 正向切片

In [15]:

```
print(cars[:3])      # 前三个元素，开始位置缺省，默认为0；切片间隔缺省，默认为1
```

['BYD', 'BMW', 'AUDI']

In [17]:

```
print(cars[1:4:2])  # 第二个到第四个元素 前后索引差为2
```

['BMW', 'TOYOTA']

In [18]:

```
print(cars[:])      # 获取整个列表，结束位置缺省，默认取值到最后
```

['BYD', 'BMW', 'AUDI', 'TOYOTA']

In [19]:

```
print(cars[-4:-2])  # 获取前两个元素
```

['BYD', 'BMW']

- 反向切片

In [28]:

```
cars = ["BYD", "BMW", "AUDI", "TOYOTA"]
```

In [20]:

```
print(cars[:-4:-1])       # 开始位置缺省，默认为-1
print(cars[::-1])         # 获得反向列表
```

['TOYOTA', 'AUDI', 'BMW']
['TOYOTA', 'AUDI', 'BMW', 'BYD']

4.1.3 列表的操作符

- 用 **list1+lis2** 的形式实现列表的拼接

In [30]:

```
a = [1, 2]
b = [3, 4]
a+b             # 该用法用的不多
```

Out[30]:

[1, 2, 3, 4]

- 用 n*list 或 list*n 实现列表的成倍复制

初始化列表的一种方式

In [31]:

```
[0]*10
```

Out[31]:

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

4.1.4 列表的操作方法

1、增加元素

- 在末尾增加元素——列表.append(待增元素)

In [22]:

```
languages = ["Python", "C++", "R"]
```

In [23]:

```
languages.append("Java")
languages
```

Out[23]:

```
['Python', 'C++', 'R', 'Java']
```

- 在任意位置插入元素——列表.insert(位置编号，待增元素)
  在位置编号相应元素前插入待增元素

In [24]:

```
languages.insert(1, "C")
languages
```

Out[24]:

```
['Python', 'C', 'C++', 'R', 'Java']
```

- 在末尾整体并入另一列表——列表1.extend(列表2)

append 将列表2整体作为一个元素添加到列表1中

In [25]:

```
languages.append(["Ruby", "PHP"])
languages
```

Out[25]:

['Python', 'C', 'C++', 'R', 'Java', ['Ruby', 'PHP']]

extend 将待列表2内的元素逐个添加到列表1中

In [26]:

```
languages = ['Python', 'C', 'C++', 'R', 'Java']
languages.extend(["Ruby", "PHP"])
languages
```

Out[26]:

['Python', 'C', 'C++', 'R', 'Java', 'Ruby', 'PHP']

2、删除元素

- 删除列表i位置的元素 列表.pop(位置)

In [27]:

```
languages = ['Python', 'C', 'C++', 'R', 'Java']
languages.pop(1)
languages
```

Out[27]:

['Python', 'C++', 'R', 'Java']

- 不写位置信息，默认删除最后一个元素

In [28]:

```
languages.pop()
languages
```

Out[28]:

['Python', 'C++', 'R']

- 删除列表中的第一次出现的待删元素 列表.remove(待删元素)

In [29]:

```
languages = ['Python', 'C', 'R', 'C', 'Java']
languages.remove("C")
languages
```

Out[29]:

['Python', 'R', 'C', 'Java']

In [32]:

```
languages = ['Python', 'C', 'R', 'C', 'Java']
while "C" in languages:
    languages.remove("C")
languages
```

Out[32]:

['Python', 'R', 'Java']

3、查找元素

- 列表中第一次出现待查元素的位置 列表.index(待查元素)

In [35]:

```
languages = ['Python', 'C', 'R','Java']
idx = languages.index("R")
idx
```

Out[35]:

2

4、修改元素

- 通过"先索引后赋值"的方式，对元素进行修改 列表名[位置]=新值

In [36]:

```
languages = ['Python', 'C', 'R','Java']
languages[1] = "C++"
languages
```

Out[36]:

['Python', 'C++', 'R', 'Java']

5、列表的复制

- 错误的方式

In [41]:

```
languages = ['Python', 'C', 'R','Java']
languages_2 = languages
print(languages_2)
```

```
['Python', 'C', 'R', 'Java']
```

In [42]:

```
languages.pop()
print(languages)
print(languages_2)
```

```
['Python', 'C', 'R']
['Python', 'C', 'R']
```

- 正确的方式——浅拷贝

- 方法1：列表.copy()

In [43]:

```
languages = ['Python', 'C', 'R','Java']
languages_2 = languages.copy()
languages.pop()
print(languages)
print(languages_2)
```

```
['Python', 'C', 'R']
['Python', 'C', 'R', 'Java']
```

- 方法2：列表[：]

In [44]:

```
languages = ['Python', 'C', 'R','Java']
languages_3 = languages[:]
languages.pop()
print(languages)
print(languages_3)
```

```
['Python', 'C', 'R']
['Python', 'C', 'R', 'Java']
```

6、列表的排序

- 使用列表.sort()对列表进行永久排序
- 直接在列表上进行操作，无返回值

In [45]:

```
ls = [2, 5, 2, 8, 19, 3, 7]
ls.sort()
ls
```

Out[45]:

```
[2, 2, 3, 5, 7, 8, 19]
```

- 递减排列

In [46]:

```
ls.sort(reverse = True)
ls
```

Out[46]:

```
[19, 8, 7, 5, 3, 2, 2]
```

- 使用sorted(列表)对列表进行临时排序
- 原列表保持不变，返回排序后的列表

In [48]:

```
ls = [2, 5, 2, 8, 19, 3, 7]
ls_2 = sorted(ls)
print(ls)
print(ls_2)
```

```
[2, 5, 2, 8, 19, 3, 7]
[19, 8, 7, 5, 3, 2, 2]
```

In [51]:

```
sorted(ls, reverse = True)
```

Out[51]:

```
[19, 8, 7, 5, 3, 2, 2]
```

7、列表的翻转

- 使用列表.reverse()对列表进行永久翻转
- 直接在列表上进行操作，无返回值

In [53]:

```
ls = [1, 2, 3, 4, 5]
print(ls[::-1])
ls
```

```
[5, 4, 3, 2, 1]
```

Out[53]:

```
[1, 2, 3, 4, 5]
```

In [54]:

```
ls.reverse()
ls
```

Out[54]:

```
[5, 4, 3, 2, 1]
```

8、使用for循环对列表进行遍历

In [55]:

```
ls = [1, 2, 3, 4, 5]
for i in ls:
    print(i)
```

```
1
2
3
4
5
```

4.2 元组

4.2.1 元组的表达

- 元组是一个可以使用多种类型元素，一旦定义，内部元素不支持增、删和修改操作的序列类型

通俗的讲，可以将元组视作“不可变的列表”

In [67]:

```
names = ("Peter", "Pual", "Mary")
```

4.2.2 元组的操作

- 不支持元素增加、元素删除、元素修改操作
- 其他操作与列表的操作完全一致

4.2.3 元组的常见用处

打包与解包

- 例1

In [56]:

```
def f1(x):                  # 返回x的平方和立方
    return x**2, x**3       # 实现打包返回

print(f1(3))
print(type(f1(3)))          # 元组类型
```

```
(9, 27)
<class 'tuple'>
```

In [57]:

```
a, b = f1(3)                # 实现解包赋值
print(a)
print(b)
```

```
9
27
```

- 例2

In [58]:

```
numbers = [201901, 201902, 201903]
name = ["小明", "小红", "小强"]
list(zip(numbers,name))
```

Out[58]:

```
[(201901, '小明'), (201902, '小红'), (201903, '小强')]
```

In [59]:

```
for number,name in zip(numbers,name):    # 每次取到一个元组，立刻进行解包赋值
    print(number, name)
```

```
201901 小明
201902 小红
201903 小强
```

4.3 字典

4.3.1 字典的表达

- 映射类型： 通过“键”-“值”的映射实现数据存储和查找
- 常规的字典是无序的

In [ ]:

```
students = {201901: '小明', 201902: '小红', 201903: '小强'}
students
```

字典键的要求

- 1、字典的键不能重复

In [61]:

```
students = {201901: '小明', 201901: '小红', 201903: '小强'}
students
```

Out[61]:

```
{201901: '小红', 201903: '小强'}
```

- 2、字典的键必须是不可变类型，如果键可变，就找不到对应存储的值了

- 不可变类型：数字、字符串、元组。 一旦确定，它自己就是它自己，变了就不是它了。
- 可变类型：列表、字典、集合。 一旦确定，还可以随意增删改。

In [67]:

```
d1 = {1: 3}
d2 = {"s": 3}
d3 = {(1,2,3): 3}
```

In [68]:

```
d = {[1, 2]: 3}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-68-bf7f06622b3f> in <module>
----> 1 d = {[1, 2]: 3}

TypeError: unhashable type: 'list'
```

In [69]:

```
d = {{1:2}: 3}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-69-188e5512b5fe> in <module>
----> 1 d = {{1:2}: 3}

TypeError: unhashable type: 'dict'
```

In [70]:

```
d = {{1, 2}: 3}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-70-c2dfafc1018a> in <module>
----> 1 d = {{1, 2}: 3}

TypeError: unhashable type: 'set'
```

4.3.2 字典的性质

- 字典的长度——键值对的个数

In [83]:

```
students = {201901: '小明', 201902: '小红', 201903: '小强'}
len(students)
```

Out[83]:

3

- 字典的索引

通过 字典[键] 的形式来获取对应的值

In [71]:

```
students = {201901: '小明', 201902: '小红', 201903: '小强'}
students[201902]
```

Out[71]:

'小红'

4.3.3 字典的操作方法

1、增加键值对

- 变量名[新键] = 新值

In [72]:

```
students = {201901: '小明', 201902: '小红', 201903: '小强'}
students[201904] = "小雪"
students
```

Out[72]:

{201901: '小明', 201902: '小红', 201903: '小强', 201904: '小雪'}

2、删除键值对

- 通过del 变量名[待删除键]

In [73]:

```
students = {201901: '小明', 201902: '小红', 201903: '小强'}
del students[201903]
students
```

Out[73]:

{201901: '小明', 201902: '小红'}

- 通过变量名.pop(待删除键)

In [74]:

```
students = {201901: '小明', 201902: '小红', 201903: '小强'}
value = students.pop(201903)    # 删除键值对，同时获得删除键值对的值
print(value)
print(students)
```

小强
{201901: '小明', 201902: '小红'}

- 变量名.popitem() 随机删除一个键值对，并以元组返回删除键值对

In [77]:

```
students = {201901: '小明', 201902: '小红', 201903: '小强'}
key, value = students.popitem()
print(key, value)
print(students)
```

201903 小强
{201901: '小明', 201902: '小红'}

3、修改值

- 通过先索引后赋值的方式对相应的值进行修改

In [78]:

```
students = {201901: '小明', 201902: '小红', 201903: '小强'}
students[201902] = "小雪"
students
```

Out[78]:

{201901: '小明', 201902: '小雪', 201903: '小强'}

4、d.get( )方法

d.get(key,default) 从字典d中获取键key对应的值，如果没有这个键，则返回default

- 小例子：统计"牛奶奶找刘奶奶买牛奶"中字符的出现频率

In [81]:

```
s = "牛奶奶找刘奶奶买牛奶"
d = {}
print(d)
for i in s:
    d[i] = d.get(i, 0)+1
    print(d)
# print(d)
```

```
{}
{'牛': 1}
{'牛': 1, '奶': 1}
{'牛': 1, '奶': 2}
{'牛': 1, '奶': 2, '找': 1}
{'牛': 1, '奶': 2, '找': 1, '刘': 1}
{'牛': 1, '奶': 3, '找': 1, '刘': 1}
{'牛': 1, '奶': 4, '找': 1, '刘': 1}
{'牛': 1, '奶': 4, '找': 1, '刘': 1, '买': 1}
{'牛': 2, '奶': 4, '找': 1, '刘': 1, '买': 1}
{'牛': 2, '奶': 5, '找': 1, '刘': 1, '买': 1}
```

5、d.keys( ) d.values( )方法

In [82]:

```
students = {201901: '小明', 201902: '小红', 201903: '小强'}
print(list(students.keys()))
print(list(students.values()))
```

```
[201901, 201902, 201903]
['小明', '小红', '小强']
```

6、d.items( )方法及字典的遍历

In [103]:

```
print(list(students.items()))
for k, v in students.items():
    print(k, v)
```

```
[(201901, '小明'), (201902, '小红'), (201903, '小强')]
201901 小明
201902 小红
201903 小强
```

4.4 集合

4.4.1 集合的表达

- 一系列互不相等元素的无序集合
- 元素必须是不可变类型：数字，字符串或元组，可视作字典的键
- 可以看做是没有值，或者值为None的字典

In [83]:

```
students = {"小明", "小红", "小强", "小明"}    #可用于去重
students
```

Out[83]:

```
{'小强', '小明', '小红'}
```

4.4.2 集合的运算

- 小例子 通过集合进行交集并集的运算

In [108]:

```
Chinese_A = {"刘德华", "张学友", "张曼玉", "钟楚红", "古天乐", "林青霞"}
Chinese_A
```

Out[108]:

```
{'刘德华', '古天乐', '张学友', '张曼玉', '林青霞', '钟楚红'}
```

In [116]:

```
Math_A = {"林青霞", "郭富城", "王祖贤", "刘德华", "张曼玉", "黎明"}
Math_A
```

Out[116]:

```
{'刘德华', '张曼玉', '林青霞', '王祖贤', '郭富城', '黎明'}
```

- 语文和数学两门均为A的学员

- S & T 返回一个新集合，包括同时在集合S和T中的元素

In [111]:

```
Chinese_A & Math_A
```

Out[111]:

{'刘德华', '张曼玉', '林青霞'}

- 语文或数学至少一门为A的学员
- S | T 返回一个新集合，包括集合S和T中的所有元素

In [112]:

```
Chinese_A | Math_A
```

Out[112]:

{'刘德华', '古天乐', '张学友', '张曼玉', '林青霞', '王祖贤', '郭富城', '钟楚红', '黎明'}

- 语文数学只有一门为A的学员
- S ^ T 返回一个新集合，包括集合S和T中的非共同元素

In [113]:

```
Chinese_A ^ Math_A
```

Out[113]:

{'古天乐', '张学友', '王祖贤', '郭富城', '钟楚红', '黎明'}

- 语文为A，数学不为A的学员
- S - T 返回一个新集合，包括在集合S但不在集合T中的元素

In [114]:

```
Chinese_A - Math_A
```

Out[114]:

{'古天乐', '张学友', '钟楚红'}

- 数学为A，语文不为A的学员

In [115]:

```
Math_A - Chinese_A
```

Out[115]:

{'王祖贤', '郭富城', '黎明'}

4.4.3 集合的操作方法

- 增加元素——S.add(x)

In [121]:

```
stars = {"刘德华", "张学友", "张曼玉"}
stars.add("王祖贤")
stars
```

Out[121]:

{'刘德华', '张学友', '张曼玉', '王祖贤'}

- 移除元素——S.remove(x)

In [122]:

```
stars.remove("王祖贤")
stars
```

Out[122]:

{'刘德华', '张学友', '张曼玉'}

- 集合的长度——len(S)

In [123]:

```
len(stars)
```

Out[123]:

3

- 集合的遍历——借助for循环

In [124]:

```
for star in stars:
    print(star)
```

张学友
张曼玉
刘德华