

容器的介绍

什么是容器?

常见的容器

字符串

定义

下标

切片(非常重要)

常用操作方法

判断类型

查找与替换

大小写转换

其他功能

总结

列表

引入

定义

下标与切片

遍历

常用操作方法

添加元素

查询元素

删除元素

修改与排序

总结

元组

概念与定义

下标与切片

循环遍历

常见操作

总结

字典

引入

定义

常见操作

添加元素

删除元素

修改元素

查询元素

字典的遍历

总结

集合

定义

常见操作

去重

其他操作

总结

公共操作

公共运算符

公共方法

推导式

什么是推导式

常见的推导式类型

列表推导式

字典推导式

集合推导式

容器的介绍

学习目标:

- 1 知道容器的作用
- 2 知道Python中有哪些常见的容器

什么是容器？



容器：容器是一种把多个元素组织在一起的数据结构，容器中的元素可以逐个地迭代获取，可以用各种内置方法对容器 中的数据进行增删改查等操作。

简而言之，容器就是存储多个数据的东西，同时Python为了方便我们对容器中的数据进行增加删除修改查询专门提供了相应的方法，便于我们操作。

思考：在之前的学习，我们知道变量也是可以存储数据的，那么为什么还要使用容器类型来存储数据呢？

解释：变量只能存储简单的数据，比如" name = zs ", " age=10 ", 这里我们发现变量只能存储单个数据，而且只能存储单一的数据类型，那么假如我们需要存储多个类型不一的数据该怎么办呢？比如存储张三同学的个人信息，包括身高、体重、姓名、籍贯、数学成绩等等，那么使用多个变量来存储将变得十分不方便，这个时候就需要容器类型了。

常见的容器

Python中的容器类型有很多，以下是一些常见的容器类型

- 1 序列类型(sequence)：序列是一种数据结构，是有序的容器类型
 - 字符串(str)
 - 列表(list)

- 元组(tuple)

2 映射类型：通过名称访问其各个值的数据结构，称之为mapping，字典(dict)是Python中唯一的内置映射类型。

- 字典(dict)

3 其他类型

- 集合(set)

字符串

学习目标：

- 1** 知道什么是字符串
- 2** 知道什么是下标
- 3** 掌握切片的使用
- 4** 掌握查找方法find(), index()
- 5** 掌握修改方法 replace(), split()

定义

定义：一个字符串是由若干字符组成的一个有序序列，它属于容器类型的一种，属于不可变类型。

比如常见的字符串：

- "abc"
- "你好"
- '1234567'
- 'hello world cskaoyan '

定义格式：

- 1** 一对引号，单引号双引号都可以，它们没有区别
- 2** 三引号，可以保留字符串的格式
- 3** 在字符串中，使用\表示转义

```

1  # 定义字符串
2  s1 = 'abc'                # 使用单引号
3  s2 = '12345'
4  s3 = "你好,王道考研"    # 使用双引号
5  s4 = "god bless you"
6  s5 = """                 # 使用三引号
7  我是安卓人
8      你是苹果人
9      他是鸿蒙人
10     我们都是地球人
11     """
12
13  s6 = 'I\'m a singer'    # 转义字符

```

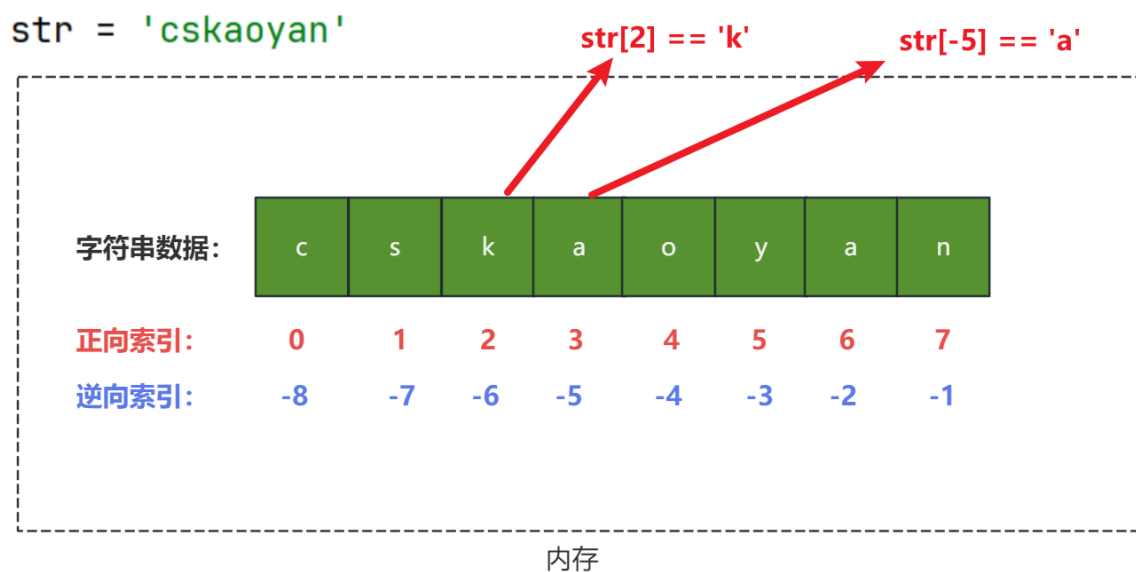
注意:

- 1 字符串有序
- 2 字符串不可变
- 3 字符串是一种容器类型, 可以使用容器类型的公共操作

下标

在计算机底层, Python中的字符串是一段连续的内存地址。

下标又叫做索引, 也可以理解为编号, 用来表示某个字符在整个字符串中的位置。



重点说明:

- 1 在Python中, 分为正向索引和逆向索引

- 2 从左往右(从前往后)编号, 且 编号从 0 开始
- 3 从右往左(从后往前)编号, 且 编号从 -1 开始
- 4 可以通过 `字符串变量名[索引]` 的形式, 从字符串中获取到指定的内容
- 5 可以通过 `len(字符串变量名)` 的形式, 获取字符串的长度, 也就是字符串中字符的个数
- 6 序列类型 (字符串、列表、元组) 也同样支持下标操作

切片(非常重要)

什么是字符串切片?

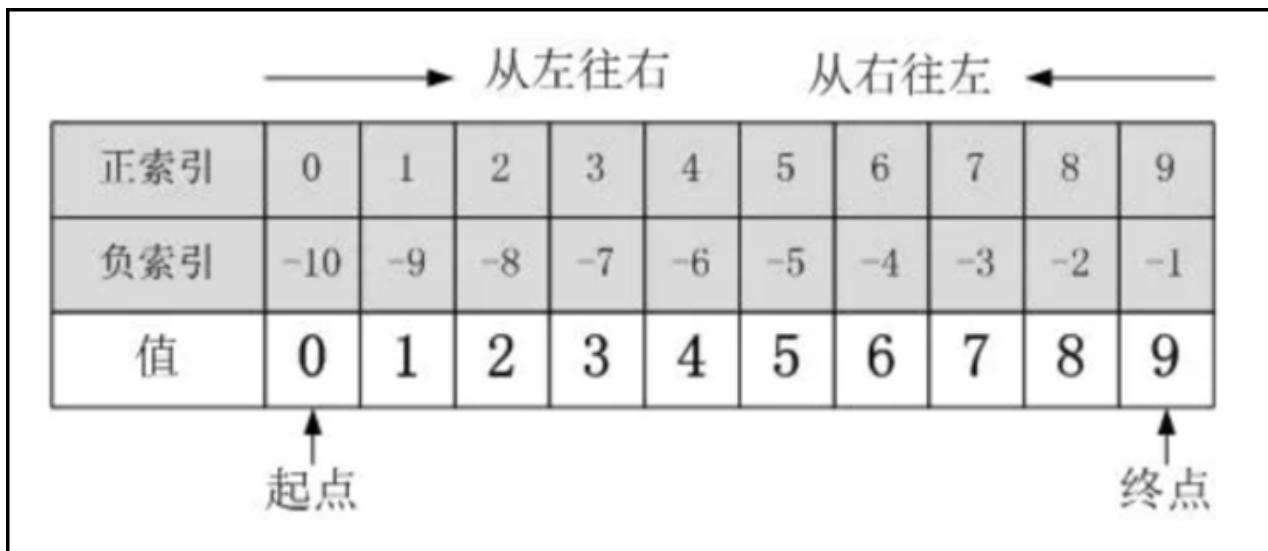
切片是指对操作的对象截取其中一部分的操作。比如想要取字符串"20252025@qq.com"中的QQ号的时候就可以使用切片。

字符串、列表、元组都支持切片操作。

字符串切片的语法: `字符串[起始位置: 结束位置: 步长]`

注意: 选取的区间从"起始位置"位开始, 到"结束位置"的前一位结束 (不包含结束位本身), 步长表示选取间隔的长度。

简而言之, 就是**左闭右开**。



字符串切片的使用案例: 比如对于字符串 "cskaoyan2025"

```
1 name = "cskaoyan2025"
2
3
4      c   s   k   a   o   y   a   n   2   0   2   5
5
6 正向索引    0   1   2   3   4   5   6   7   8   9   10  11
7 逆向索引   -12 -11 -10 -9  -8  -7  -6  -5  -4  -3  -2  -1
```

```

8
9 """
10
11 print(name[0:3:1])      # "csk"
12 print(name[0:3])        # "csk"      步长默认为1
13 print(name[0:3:2])      # "ck"
14 print(name[:5])         # "cskao"    不写起始位置，默认从0开始
15 print(name[5:])         # "yan2025"  不写结束位置，默认到最后
16
17 print(name[-4:])        # "2025"    也可以使用逆向索引的下标
18 print(name[-12:-4])     # "cskaoyan"
19 print(name[-4:-12:-1])  # "2nayorks"  如果要逆序，那么步长为负

```

注意事项：

- 1 指定的区间属于**左闭右开**，**[开始索引，结束索引)**，从起始位开始，到结束位的前一位结束
- 2 默认从头开始，开始索引数字可以省略，冒号不能省略
- 3 默认到末尾结束，结束索引数字可以省略，冒号不能省略
- 4 步长默认为1，如果连续切片，数字和冒号都可以省略

练一练

```

1 s = "hello world"
2
3 print(s[4])
4
5 print(s)
6
7 print(s[:])
8
9 print(s[1:])
10
11 print(s[:5])
12
13 print(s[:-1])
14
15 print(s[-4:-1])
16
17 print(s[-1:-4])
18
19 print(s[1:5:2])
20
21 print(s[::-1])

```

字符串切片常见使用场景：

- 1 字符串反转

常用操作方法

字符串提供了很多的方法, 掌握常用的方法即可。

capitalize()	endswith()	index()	isdigit()	isspace()	lower()	removesuffix()	rpartition()	startswith()	upper()
casefold()	expandtabs()	isalnum()	isidentifier()	istitle()	lstrip()	replace()	rsplit()	strip()	zfill()
center()	find()	isalpha()	islower()	isupper()	maketrans()	rfind()	rstrip()	swapcase()	
count()	format()	isascii()	isnumeric()	join()	partition()	rindex()	split()	title()	
encode()	format map()	isdecimal()	isprintable()	ljust()	removeprefix()	rjust()	splitlines()	translate()	

对于以上的方法, 按照功能对其分类如下

判断类型

```
1 # 如果 string 中只包含空格, 则返回True
2 def isspace()
3
4 # 如果 string 至少有一个字符并且所有字符都是字母或数字则返回 True
5 def isalnum()
6
7 # 如果 string 至少有一个字符并且所有字符都是字母则返回 True
8 def isalpha()
9
10 # 如果 string 只包含数字则返回 True, 包含Unicode数字, , 全角数字(双字节)
11 def isdecimal()
12
13 # 如果 string 只包含数字则返回 True, 包含Unicode数字, byte数字(单字节),
    全角数字(双字节)
14 def isdigit()
15
16 # 如果 string 只包含数字则返回 True, 包含Unicode 数字, 全角数字(双字节),
    汉字数字
17 def isnumeric()
18
19 # 如果 string 是标题化的(每个单词的首字母大写)则返回 True
20 def istitle()
21
22 # 如果 string 中包含至少一个区分大小写的字符, 并且所有这些(区分大小写的)字符都
    是小写, 则返回 True
23 def islower()
24
25 # 如果 string 中包含至少一个区分大小写的字符, 并且所有这些(区分大小写的)字符都
    是大写, 则返回 True
26 def isupper()
```

查找与替换

```
1 # 检查字符串是否是以 str 开头，是则返回 True
2 def startswith(str)
3
4 # 检查字符串是否是以 str 结束，是则返回 True
5 def endswith(str)
6
7 # 检测 str 是否包含在 string 中，如果start 和 end 指定范围，则检查是否包含
  在指定范围内，如果是返回开始的索引值，否则返回-1
8 # 与index区别,只能用位置参数，不能用keyword
9 def find(str,start=0,end=len(str))
10
11 # 类似于 find(), 不过是从右边开始查找
12 def rfind(str,start=0,end=len(str))
13
14 # 跟 find() 方法类似，不过如果 str 不在 string 会报错
15 def index(str,start=0,end=len(str))
16
17 # 类似于 index(), 不过是从右边开始
18 def rindex(str,start=0,end=len(str))
19
20 # 把 string 中的 old_str 替换成新_str，如果 num 指定，则替换不超过 num
  次
21 def replace(old_str, new_str, num=string.count(old))
```

常用方法：

1 find()

2 replace()

练一练：

查找大串中子串出现的次数（count）。

大串：dbjakbdj123bkabdjka12bjkabdj123bjadbja123bfegbrk123;jfpej12

子串：123

大小写转换

```
1 # 把字符串的第一个字符大写
2 def capitalize()
3
4 # 把字符串的每个单词首字母大写
5 def title()
6
```

```

7 # 转换 string 中所有大写字符为小写
8 def lower()
9
10 # 转换 string 中的小写字母为大写
11 def upper()
12
13 # 翻转 string 中的大小写
14 def swapcase()

```

其他功能

```

1 # 返回一个原字符串左对齐, 并使用空格填充至长度 width 的新字符串
2 def ljust(width)
3
4 # 返回一个原字符串右对齐, 并使用空格填充至长度 width 的新字符串
5 def rjust(width)
6
7 # 返回一个原字符串居中, 并使用空格填充至长度 width 的新字符串
8 def center(width)
9
10 # 截掉 string 左边(开始)的空白字符, 可以去除字符 char
11 def lstrip(char)
12
13 # 截掉 string 右边(末尾)的空白字符, 可以去除字符 char
14 defrstrip(char)
15
16 # 截掉 string 左右两边的空白字符, 可以去除字符 char
17 def strip(char)
18
19 # 把字符串 string 分成一个 3 元素的元组 (str 前面, str, str 后面)
20 def partition(str)
21
22 # 类似于 partition() 方法, 不过是从右边开始查找
23 def rpartition(str)
24
25 # 以 str 为分隔符拆分 string, 如果 num 有指定值, 则仅分隔 num + 1 个子字符串, str 默认包含空格
26 def split(str="", num)
27
28 # splitlines 只是换行, 每行字符串的内容不做修改
29 def splitlines()
30
31 # 用字符串将可迭代对象中的元素连接起来, seq 为可迭代的对象, 比如
    ', '.join(['a', 'b', 'c']) -> 'a,b,c'
32 def join(seq)

```

总结

1 字符串的定义

- 字符串有序
- **字符串不可变**
- 字符串也是一种字符的容器

2 字符串的下标

- 从左往右，下标从0开始
- 从右往左，下标从-1开始

3 字符串的切片

- 语法：**字符串[起始位置:结束位置:步长]**
- 左闭右开
- 起始位置默认从头开始，结束位置默认到末尾，步长默认为1

4 字符串常见方法

- find()
- index()
- replace()
- split()

练一练：

- 1 输入一个字符串，打印所有偶数位上的字符(下标是0，2，4，6...位上的字符)
- 2 给定一个文件名，判断其尾部是否以".png"结尾

列表

学习目标：

- 1 掌握如何定义一个列表
- 2 掌握列表的下标与切片
- 3 掌握列表的遍历
- 4 掌握列表的常见增删改查操作

引入

假设一个班上有80个人，现在需要统计，某一门课程的平均成绩，是否需要定义80个变量？

很明显：

- 1 如果程序需要多少数据，就定义多少个变量，过于麻烦了，这种代码写出来也不美观。
- 2 当然，更重要的是多个变量单独定义且单独存在，之间没有任何关联，很不方便管理和维护。

这样的做法，就好比，去超市购物，把东西一个一个单独带回家。

很显然，购物需要**"袋子"**装东西，Python变量也需要一个**容器**来存放数据。

Python中用于存放数据的容器有很多，但最常见的容器是列表（List）。定义一个列表，就可以存储很多数据。

思考：

列表中的数据是胡乱摆放在一起吗？

定义

基本语法： `列表名称 = [数据1, 数据2, 数据3, 数据4, ...]`

```
1 # 列表定义举例：
2 name_list = ['tom', 'jerry', 'bob', 'alice']
3 score_list = [6, 7, 6.5, 8.5]
```

注意事项：

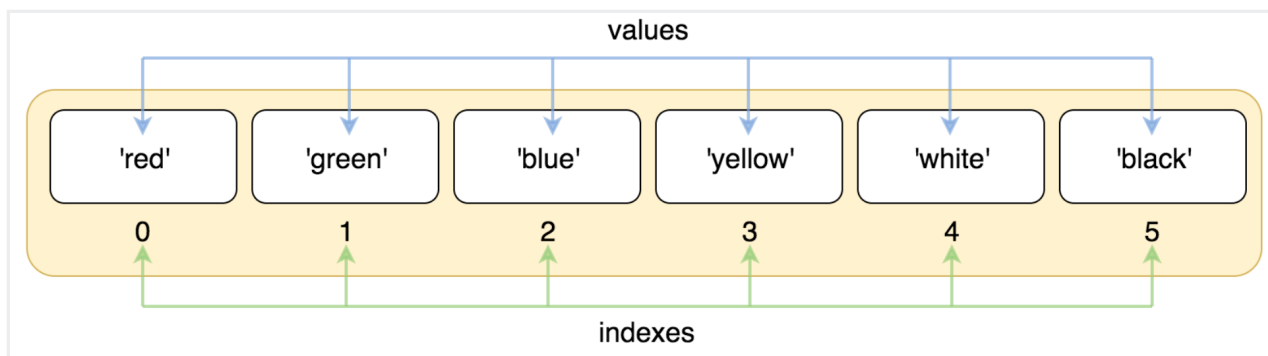
- 1 列表是Python中使用最频繁的容器类型，类似于其他语言中的数组
- 2 列表使用 `[]` 定义，数据之间使用 `,` 分隔
- 3 列表可以一次存储多个数据且可以为不同的数据类型

下标与切片

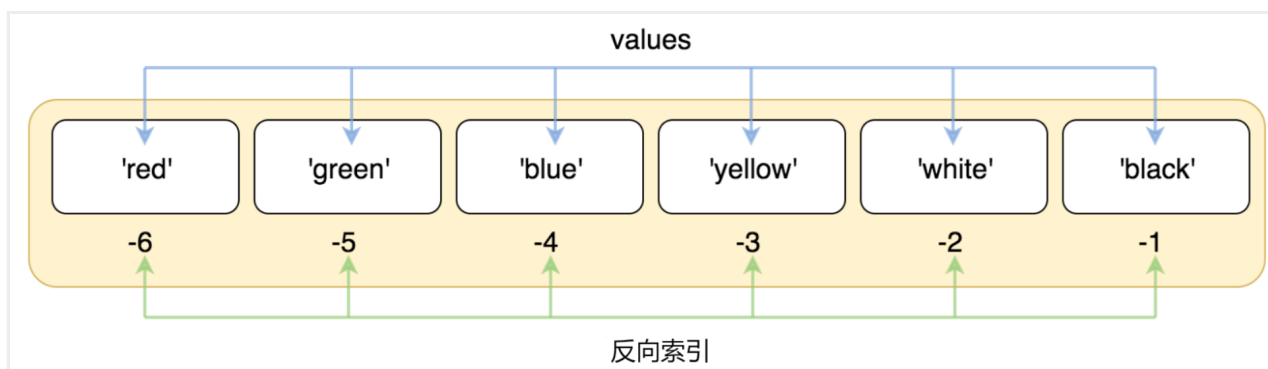
- 1 列表也有下标，和字符串类似
- 2 可以通过 `列表名[下标]` 访问列表中的值

3 列表也支持切片操作

与字符串的下标一样，列表的索引从0开始，然后第二个是1，以此类推。



索引也可以从尾部开始，最后一个元素的索引为 -1，往前一位为 -2，以此类推。



```
1 # 列表的下标
2 list1 = ['red', 'green', 'blue', 'yellow', 'white', 'black']
3
4 print( list1[0] )      # red
5 print( list1[1] )      # green
6 print( list1[2] )      # blue
7
8 print( list1[-1] )     # black
9 print( list1[-2] )     # white
10 print( list1[-3] )    # yellow
11
12 # 取值的时候，下标注意不要越界，否则会报错
13 # IndexError: list index out of range
14 print( list1[10] )
```

```
1 # 列表的切片
2 nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
3
4 print(nums[0:4])       # [10, 20, 30, 40]
5 print(nums[2:5])       # [30, 40, 50]
6 print(nums[1:5:2])     # [20, 40]
7 print(nums[-1:-3:-1])  # [90, 80]
8 print(nums[-1::-2])    # [90, 70, 50, 30, 10]
```

遍历

遍历概念：

- 1 **遍历** 就是 **从头到尾 依次** 从 **列表** 中获取数据，在 **循环体内部** 针对 **每一个元素**，执行相同的操作
- 2 在 Python 中为了提高列表的遍历效率，专门提供的 **迭代 iteration 遍历**
- 3 使用 `iter()` 就能够实现迭代遍历

```
1 name_list = ['zs', 'ls', 'ww', 'zl']
2
3 # 使用while循环进行遍历(较麻烦，以后不常用)
4 i = 0
5 length = len(name_list)
6 while i < length:
7     temp_name = name_list[i]
8     print(temp_name)
9     i += 1
10
11
12 # 使用for循环进行迭代遍历(更简单，推荐使用)
13 for name in name_list:
14     print(name)
```

常用操作方法

列表的作用是一次性存储多个数据，程序员可以对这些数据进行的操作有：增、删、改、查。

序号	方法	描述
1	<code>list.append(obj)</code>	在列表末尾添加新的对象
2	<code>list.count(obj)</code>	统计某个元素在列表中出现的次数
3	<code>list.extend(seq)</code>	在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）
4	<code>list.index(obj)</code>	从列表中找出某个值第一个匹配项的索引位置
5	<code>list.insert(index, obj)</code>	将对象插入列表

6	<code>list.pop([index=-1])</code>	移除列表中的一个元素（默认最后一个元素），并且返回该元素的值
7	<code>list.remove(obj)</code>	移除列表中某个值的第一个匹配项
8	<code>list.reverse()</code>	反向列表中元素
9	<code>list.sort(key=None, reverse=False)</code>	对原列表进行排序
10	<code>list.clear()</code>	清空列表
11	<code>list.copy()</code>	复制列表

添加元素

往列表中，添加元素，主要借助于下面的三个方法来实现。

编号	函数	描述
1	<code>append(obj)</code>	在列表末尾添加新的对象
2	<code>extend(seq)</code>	在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）
3	<code>insert(index,obj)</code>	指定位置新增数据

```

1  """
2  append() 方法：在列表末尾添加新的对象
3
4  注意：
5      1. 会追加列表末尾
6      2. 可以重复添加，也就是列表可以存储重复的数据
7      3. 列表追加数据的时候，直接在原列表里面追加了指定数据，即修改了原列表，故列表为可变类型数据。
8  """
9  name_list = ['Tom', 'Lily', 'Rose']
10 name_list.append('Jennifer')
11 print(name_list)                                # ['Tom', 'Lily',
'Rose', 'Jennifer']
12
13
14 #####
#####
15
16 """
17 extend() 方法：在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）

```

```

18
19 注意：
20     1. 会追加列表末尾
21     2. 括号中的参数必须是一个序列，是一个可迭代对象
22     3. 如果参数是字符串，不会报错，但是会把这个字符串的每一个字符独立的添加到列
    表中来
23     """
24     city_list = ["北京", "上海", "广州", "深圳"]
25     new_city_list = ["成都", "杭州", "重庆", "武汉"]
26
27     city_list.extend(new_city_list)
28     print(city_list)
29
30     #####
    #####
31
32     """
33     insert() 方法：指定位置新增数据， insert(下标, 新数据)
34
35 注意：
36     1. 在当前指定下标的元素 【前面】，插入新元素，不论下标正负
37     2. 下标可以超过当前列表的范围，如果超过了范围，那么添加到开头或末尾
38
39     """
40     cls_list = ['zs', 'ls', 'ww', 'zl']
41     cls_list.insert(1, '李飞') # ['zs', '李飞', 'ls', 'ww',
    'zl']
42     print(cls_list)
43
44     cls_list.insert(-2, "常乃超") # ['zs', '李飞', 'ls', '常乃
    超', 'ww', 'zl']
45     print(cls_list)
46
47     cls_list.insert(100, '嘉豪') # ['zs', '李飞', 'ls', '常乃
    超', 'ww', 'zl', '嘉豪']
48     print(cls_list)

```

查询元素

查询元素，主要借助于下面的几个方法来实现。

编号	函数	描述
1	index()	从列表中找到某个值第一个匹配项的索引位置
2	count()	统计某个元素在列表中出现的次数

3	in	判断指定数据是否在某个列表序列，如果在返回True，否则返回False
4	not in	判断指定数据是否不在某个列表序列，如果不在返回True，否则返回False

```

1  """
2  index(元素值, 起始位置, 结束位置)：从列表中找出某个值第一个匹配项的索引位置
3
4  注意事项：
5      1. 元素必须在列表中，否则会报错
6      2. 起始位置和结束位置这两个参数可以省略，默认查询整个列表
7  """
8  my_list = ['小李', '小明', '嘉豪', '小张', '嘉豪', '老王', '老李', '楚云飞', '嘉豪']
9  print(my_list.index('嘉豪'))
10 print(my_list.index("嘉豪", 3))
11 print(my_list.index("嘉豪", 3, 10))
12
13 #####
14 """
15 count()：统计某个元素在列表中出现的次数, 如果没有，返回0
16 """
17
18 my_list = ['小李', '小明', '嘉豪', '小张', '嘉豪', '老王', '老李', '楚云飞', '嘉豪']
19 print(my_list.count("嘉豪"))          # 3
20 print(my_list.count("风华"))          # 0
21
22 #####
23 """
24 in:      判断指定数据在某个列表序列，如果在返回True，否则返回False
25 not in:  判断指定数据不在某个列表序列，如果不在返回True，否则返回False
26
27 """
28 my_list = ['小李', '小明', '嘉豪', '小张', '嘉豪', '老王', '老李', '楚云飞', '嘉豪']
29 print('小李' in my_list)              # True
30 print('小李' not in my_list)          # False
31

```

删除元素

删除元素，主要借助于以下的几种方式来实现：

编号	函数	描述
1	del 列表[索引]	删除列表中指定位置的某个元素
2	pop()	移除列表中的一个元素（默认最后一个元素），并且返回该元素的值
3	remove()	移除列表中某个数据的第一个匹配项。
4	clear()	清空列表，删除列表中的所有元素，返回空列表。

```
1  """
2  del: 这是一个关键字，可以帮助我们删除容器中的某个元素
3
4  """
5  hero_list = ['吴用', '宋江', '鲁智深', '林冲', '花荣', '晁盖']
6  del hero_list[1]
7  print(hero_list)                # ['吴用', '鲁
   智深', '林冲', '花荣', '晁盖']
8
9
10 #####
11 #####
12 """
13 pop(): 移除列表中的一个元素（默认最后一个元素），并且返回该元素的值
14         也可以指定元素的下标
15
16 """
17 hero_list = ['吴用', '宋江', '鲁智深', '林冲', '花荣', '晁盖']
18 print(hero_list.pop())           # 晁盖
19 print(hero_list.pop(0))          # 吴用
20 #####
21 #####
22 """
23 remove(obj): 移除列表中某个数据的第一个匹配项
24
25 """
26 my_list = ['小李', '小明', '嘉豪', '小张', '嘉豪', '老王', '老李', '楚云飞', '嘉
   豪']
27 my_list.remove("嘉豪")
```

```

27 print(my_list)                # ['小李', '小明', '小张', '嘉豪', '老
   王', '老李', '楚云飞', '嘉豪']
28
29
30 #####
   #####
31 """
32 clear(): 清空列表，删除列表中的所有元素，返回空列表。
33 """
34
35 hero_list = ['吴用', '宋江', '鲁智深', '林冲', '花荣', '晁盖']
36 hero_list.clear()
37 print(hero_list)              # []
38

```

修改与排序

修改列表，主要有以下几种方式：

编号	函数	描述
1	列表[索引] = 修改后的值	直接通过下标修改
2	reverse()	将数据序列进行倒序排列
3	list.sort(key=None, reverse=False)	对列表序列进行排序

```

1  """
2  列表[索引] = 修改后的值： 直接通过下标修改
3
4  """
5  hero_list = ['吴用', '宋江', '鲁智深', '林冲', '花荣', '晁盖']
6  hero_list[0] = "智多星"
7  print(hero_list)              # ['智多星', '宋江', '花和尚', '林冲', '花
   荣', '晁盖']
8
9  #####
   #####
10
11 """
12 reverse(): 反转，将数据序列进行倒序排列
13
14 """
15 hero_list = ['吴用', '宋江', '鲁智深', '林冲', '花荣', '晁盖']
16 hero_list.reverse()

```

```

17 print(hero_list)                # ['晁盖', '花荣', '林冲', '鲁智深', '宋
   江', '吴用']

18

19 #####
   #####

20 """
21 list.sort( key=None, reverse=False): 对列表序列进行排序
22
23 参数解释:
24     1. key: 如果是一个对象列表, 那么指定对象中的某一个属性进行排序, 当前可以省
   略
25     2. reverse: 默认为False, 升序排序; True: 降序排序
26 """
27 score_list = [80,90,82,70,55,98,63]
28 score_list.sort()                # [55, 63, 70, 80, 82, 90,
   98]
29 print(score_list)
30
31 score_list.sort(reverse=True)    # [98, 90, 82, 80, 70, 63,
   55]
32 print(score_list)

```

总结

1 列表的定义: `name_list = [元素1, 元素2, 元素3 ...]`

- 列表有序
- **列表可变**
- 列表中的元素类型可以不一致

2 列表的下标与切片

3 列表的遍历: `for element in list:`

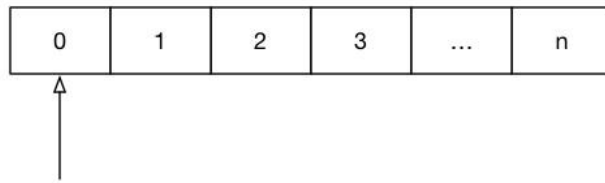
- 使用for进行迭代遍历

4 列表的增删改查方法

- 增加: `append()`、`extend()`、`insert()`
- 删除: `del`、`remove()`、`pop()`、`clear()`
- 查询: `index()`、`count()`、`in`、`not in`
- 修改: `列表[索引] = 修改后的值`、`reverse()`、`sort()`

列表的索引值是从 0 开始的

`len(列表)` 获取列表的长度 $n + 1$
`列表.count(数据)` 数据在列表中出现的次数



`列表.sort()` 升序排序
`列表.sort(reverse=True)` 降序排序
`列表.reverse()` 反转/逆序

`列表[索引]` 从列表中取值
`列表.index(数据)` 获得数据第一次出现的索引

`del 列表[索引]` 删除指定索引的数据
`列表.remove[数据]` 删除第一个出现的指定数据
`列表.pop` 删除末尾数据
`列表.pop(索引)` 删除指定索引的数据

`列表.insert(索引, 数据)` 在指定位置插入数据
`列表.append(数据)` 在末尾追加数据
`列表.extend(列表2)` 将列表 2 的数据追加到列表 1

元组

学习目标：

- 1 掌握如何定义一个元组
- 2 掌握元组的常用操作

概念与定义

定义Tuple（元组）与列表类似，不同之处在于元组内的**元素不能修改**。

- **元组**表示多个元素组成的序列
- 用于存储**一串信息**，**数据**之间使用,分隔
- 元组用()定义
- 元组的**索引**从0开始

基本语法： `元组名 = (元素1, 元素2, 元素3, 元素4 ...)`

```

1  # 元组的定义:
2
3  # 1. 普通的元组 (包含多个元素)
4  t1 = (10,20,30)
5  t2 = ('zs','ls','ww')
6
7  # 2. 特殊的元组 (只含有一个元素)
8  t3 = (10,)
9
10
11 # !!! 错误的写法, 该t4的类型就是10的类型, 是int类型, 不能省略 ,
12 t4 = (10)

```

注意事项:

- 1 元组同样有序
- 2 元组不可修改, 而列表可以修改
- 3 元组使用 `()` 定义, 数据之间使用 `,` 分隔

下标与切片

因为元组也是一个序列, 所以我们可以访问元组中的指定位置的元素, 也可以截取索引中的一段元素。

也就是元组内的元素也有下标, 也同样支持通过下标取值与切片操作, 和列表几乎一样, 没什么区别。

练一练

```

1  t1 = (10,20,30,40,50,60,70,80,90)
2
3  # 1. 通过下标获取对应位置的值
4  print(t1[3])           # 40
5  print(t1[5])           # 60
6
7  # 2. 切片操作
8  print(t1[2:5:1])        # (30,40,50)
9  print(t1[4:])           # (50,60,70,80,90)
10 print(t1[:4])           # (10,20,30,40)
11 print(t1[2::3])         # (30,60,90)
12 print(t1[::-2])         # (90,70,50,30,10)
13 print(t1[-3:])          # (70,80,90)

```

循环遍历

元组的循环遍历和列表类似，也支持while循环和for迭代两种方式。

```
1  # 定义一个元组
2  t1 = ('李云龙', '楚云飞', '谢宝庆', '蟹部落')
3
4  # while循环遍历（一般不用）
5  index = 0
6  while index < len(t1):
7      print(t1[index])
8      index += 1
9
10
11 # for循环迭代遍历（更简单，推荐使用）
12 for name in t1:
13     print(name)
```

常见操作

因为元组是不可变的，所以元组支持的操作不多。

编号	函数	描述
1	index()	查找某个数据，如果数据存在返回对应的下标，否则报错，语法和列表、字符串的index方法相同
2	count()	统计某个数据在当前元组出现的次数
3	len()	统计元组中数据的个数
4	元组[索引]	根据索引下标查找元素

```
1  # 1. 定义一个元组
2  tup = ('aa', 'bb', 'cc', 'dd', 'ee', 'bb', 'cc')
3
4  # 2. 查找某个元素 index()
5  print(tup.index('bb'))                                # 1
6  print(tup.index('bb', 3, 6))                          # 5
7
8  # 3. 统计某个元素出现的次数 count()
9  print(tup.count('aa'))                                # 1
10 print(tup.count('bb'))                                # 2
11
12 # 4. 统计元组中元素的总个数 len()
13 print(len(tup))                                       # 7
```

```

14
15 # 5. 根据下标获取元素的值
16 print(tup[3])                                # dd
17
18
19 # 补充： 也可以把元组转为列表，或者是把列表转为元组
20 name_list = list(tup);                        # 把元组转为列表
21 tup2 = tuple(name_list);                      # 把列表转为元组
22
23
24 # 补充： 元组是不可变序列，不能通过下标修改元组内部的元素
25 # TypeError: 'tuple' object does not support item assignment
26 tup[3] = 'DD'

```

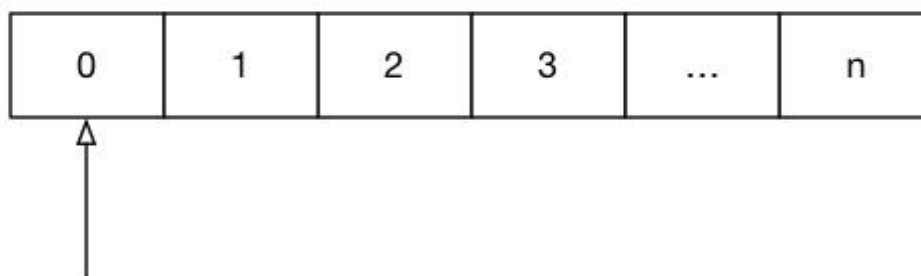
总结

元组与列表很相似，区别是元组是不可变序列，而列表是可变序列。

元组的索引值是从 **0** 开始的

len(元组) 获取元组的长度 $n + 1$

元组.count(数据) 数据在元组中出现的次数



元组[索引] 从列表中取值

元组.index(数据) 获得数据第一次出现的索引

1 元组的定义

- 多个元素: `tup = (元素1, 元素2, 元素3, ...)`
- 单个元素: `tup = (元素1,)`

2 元组的下标与切片

- 与列表一致

3 元组的常见操作

- `index()`
- `count()`
- `len()`
- 通过下标取值

4 元组的不可变

- **不能通过下标修改值**
- **可以修改元组引用指向的地址**

练一练:

- 1 幸运数字6: 输入任意数字, 如数字8, 生成nums列表, 元素值为1~8, 从中选取幸运数字(能够被6整除)移动到新列表lucky, 打印nums与lucky。
- 2 列表嵌套: 有3个教室[[],[],[]], 8名讲师['A','B','C','D','E','F','G','H'], 将8名讲师随机分配到3个教室中

字典

学习目标:

- 1 掌握字典的定义与基本使用
- 2 知道字典的应用场景

引入

序列类型的容器, 比如字符串、列表、元组等都是用来存储一系列数据的, 比如存储班级所有同学的名字、中国所有的城市等这种扁平的、单一的数据, 也可以理解为是excel数据表中的**单列数据**。

但是, 有的时候我们需要存储**双列数据**, 那么这个时候就需要使用**字典**

单列数据，比如城市，就可以存入一个序列容器

双列数据，比如需要存储每一个业务员对应的金额就需要使用字典容器

销售日期	买家城市	销售商品	销售数量	单位	销售单价	业务员	销售金额
202x/xx/xx	珠海	商品a	6	台	130000	稻小壳1	780000
202x/xx/xx	北京	商品a	5	台	138000	稻小壳2	690000
202x/xx/xx	北京	商品a	7	台	138000	稻小壳3	966000
202x/xx/xx	上海	商品a	7	台	138000	稻小壳4	966000
202x/xx/xx	苏州	商品a	8	台	130000	稻小壳5	1040000
202x/xx/xx	西安	商品a	4	台	130000	稻小壳6	520000
202x/xx/xx	深圳	商品a	6	台	138000	稻小壳7	828000
202x/xx/xx	珠海	商品a	8	台	130000	稻小壳8	1040000
202x/xx/xx	广州	商品a	3	台	138000	稻小壳9	414000
202x/xx/xx	哈尔滨	商品a	3	台	130000	稻小壳10	390000

字典存储的是键值对，一个键对应一个值。比如上图中，业务员的名字就是键，业务员的销售金额就是值。

再举个例子，比如生活中常用的公共电话和它的功能也是一一对应的。

键

电话号码	功能
110	警察局
120	医院
122	车辆救援
119	火警
12345	消费者权益保护

值

定义

字典定义的特点：

- 1 符号为大括号(花括号) `{ }`
- 2 数据以键值对的形式出现
- 3 各个键值对之间使用 `,` 间隔

基本语法：`字典名 = {键1:值1, 键2: 值2, 键3: 值3, ...}`

```

1  # 字典的定义举例
2
3  # 定义一个有键值对的字典
4  student = {"name": "bob", "age": 20, "height": 180}
5
6  # 定义一个空字典
7  dict1 = {}
8  dict2 = dict()

```

注意事项：

- 1 一般称冒号前面的为键(key)，简称k；冒号后面的为值(value)，简称v；key:value就是我们通常说的键值对
- 2 **键**key是索引，必须是**不可变类型**，往往是字符串
- 3 **值**可以取任何数据类型，但**键**只能使用**字符串**、**数字**或**元组**
- 4 在同一个字典中，键不能重复，值可以重复

常见操作

字典内置的函数如下：

序号	函数	描述
1	<code>dict.clear()</code>	删除字典内所有元素
2	<code>dict.copy()</code>	返回一个字典的浅复制
3	<code>dict.fromkeys()</code>	创建一个新字典，以序列seq中元素做字典的键，val为字典所有键对应的初始值
4	<code>dict.get(key, default=None)</code>	返回指定键的值，如果键不在字典中返回 default 设置的默认值
5	<code>key in dict</code>	如果键在字典dict里返回true，否则返回false
6	<code>dict.items()</code>	以列表返回一个视图对象，包含所有的键与值
7	<code>dict.keys()</code>	以列表返回一个视图对象，包含所有的键
8	<code>dict.setdefault(key, default=None)</code>	和get()类似，但如果键不存在于字典中，将会添加键并将值设为 default
9	<code>dict.update(dict2)</code>	把字典dict2的键/值对更新到dict里
10	<code>dict.values()</code>	以列表返回一个视图对象，包含所有的值
11	<code>dict.pop(key)</code>	删除字典 key（键）所对应的值，返回被删除的值。
12	<code>dict.popitem()</code>	随机字典中的一个键值对，并返回它的值

对于以上内置的函数，也基本可以按照增删改查来分类。

添加元素

基本语法: `字典名[key] = value`

- 如果key不存在, 那么添加键值对
- 如果key已经存在, 那么更新它的值

```
1  # 添加元素
2  s = {}
3  s['name'] = '张三'    # 添加键值对
4  s['age'] = 30         # 添加键值对
5  s['age'] = 50         # 修改键的值
6
7  print(s)              # {'name': '张三', 'age': 50}
```

删除元素

字典内元素的删除, 有如下几种方式:

编号	函数	解释
1	<code>del 字典名</code>	删除整个字典
2	<code>del 字典名[键]</code>	删除指定的键值对, 如果键不存在会报错
3	<code>dict.pop(key, default)</code>	删除指定的键值对, 并返回它的值
4	<code>dict.popitem()</code>	删除字典中最后添加的那个键值对, 并返回它
5	<code>dict.clear()</code>	清空字典, 删除字典中所有的键值对

```
1  """
2  1. 删除整个字典
3      del 字典名
4  """
5  s = {'name': '张三', 'age': 50}
6  del s
7  # NameError: name 's' is not defined
8  # 字典s不存在了
9  print(s)
10
11
12  """
13  2. 删除指定的键值对, 如果键不存在, 会报错
```

```

14     del 字典名[键]
15
16     """
17 s = {'name': '张三', 'age': 50}
18 del s['name']
19 del s['height']          # 删除不存在的键，会报错
20
21
22     """
23 3. 删除指定的键值对，并返回它的值
24     dict.pop(key, [default])
25         - default为可选参数，如果没有传default，那么键不存在的时候会报错
26         - 如果传了default，那么键不存在的时候不会报错
27     """
28
29 s = {'name': 'zs', "age": 20, "height": 180, "score": 10}
30
31 # 从字典中删除键值对，并返回 'zs'
32 name = s.pop('name')
33
34 # 从字典中删除不存在的键值对，会报错
35 weight = s.pop('weight')
36
37 # 从字典中删除不存在的键值对，键值对不存在，返回默认值
38 weight = s.pop('weight', 180)
39
40
41     """
42 4. dict.popitem()
43     删除字典中最后添加的那个键值对，并返回它
44     """
45 s = {'name': 'zs', "age": 20, "height": 180, "score": 10}
46
47 # score:10 是最后添加的，这里返回它
48 kv = s.popitem()
49
50
51     """
52 5. dict.clear()
53     清空字典，但是字典对象仍然存在
54     """
55 s = {'name': 'zs', "age": 20, "height": 180, "score": 10}
56 s.clear()
57 print(s)          # {}

```

修改元素

字典的修改和添加元素是一样的。

基本语法: `字典名[key] = value`

- 如果key不存在, 那么添加键值对
- 如果key已经存在, 那么更新它的值

```
1 # 修改键值对的值
2 s = {}
3 s['age'] = 30      # 添加键值对
4 s['age'] = 50      # 修改键的值
5
6 print(s)           # {'name': '张三', 'age': 50}
```

查询元素

字典的查找有以下几种方式

编号	函数	描述
1	字典[key]	直接通过key查找。如果key不存在会报错
2	get(key, 默认值)	返回指定键的值, 如果键不在字典中返回 default 设置的默认值
3	keys()	以可遍历的形式返回字典中所有的键
4	values()	以可遍历的形式返回字典中的所有值
5	items()	以可遍历的形式返回可遍历的(键, 值)元组数组

```
1 s = {'name': 'zs', "age": 20, "height": 180, "score": 10}
2
3 ## 1. 直接通过key查找
4 print(s['name'])
5 # print(s['nickname'])      # 会报错
6
7
8 ## 2. 以get()通过key来查找
9 name = s.get("name")
10 nickname = s.get("nickname")      # nickname为None, 不会报错
11 nickname = s.get('nickname', '小明')      # nickname为'小明'
12
13 ## 3. 以可遍历的形式返回字典中所有的键
```

```

14 keys = s.keys()
15 print(type(keys))          # dict_keys
16 print(keys)
17
18 ## 4. 以可遍历的形式返回字典中的所有值
19 vals = s.values()
20 print(type(vals))          # dict_values
21 print(vals)
22
23 ## 5. 以可遍历的形式返回可遍历的(键, 值)元组数组
24 itms = s.items()
25 print(type(itms))          # dict_items
26 print(itms)

```

注意事项:

- 1 keys()、values()、items()返回的都是**可迭代**对象,支持for循环进行遍历
- 2 keys()、values()、items()返回的都是**视图**对象,会随着字典中原数据的变化而变化

字典的遍历

字典的遍历,有如下几种不同的形式,可以分为:

- 遍历字典的key
- 遍历字典的value
- 遍历字典的元素
- 遍历字典的键值对

```

1  # 遍历字典的key
2  dict1 = {'name': 'Tom', 'age': 20, 'gender': 'male'}
3  for key in dict1.keys():
4      print(key)
5
6  # 遍历字典的value
7  dict1 = {'name': 'Tom', 'age': 20, 'gender': 'male'}
8  for value in dict1.values():
9      print(value)
10
11 # 遍历字典的元素
12 dict1 = {'name': 'Tom', 'age': 20, 'gender': 'male'}
13 for item in dict1.items():
14     print(item)
15
16

```

```

17 # 遍历字典的键值对 (第一种写法)
18 dict1 = {'name': 'Tom', 'age': 20, 'gender': 'male'}
19 for key, value in dict1.items():
20     print(f'{key} = {value}')
21
22 # 遍历字典的键值对 (第二种写法)
23 for key in dict1.keys():
24     print(f"key:{key}, value:{dict1[key]}")

```

总结

字典是Python唯一的一种映射容器，也是除了列表之外最常用的容器。字典是用来存储键值对的容器，在这个容器中有多个键，每一个键对应一个值。

`len(字典)` 获取字典的 键值对数量

	key	value
→	name	小明
	age	18
	gender	True
	height	1.75

字典.keys() 所有 key 列表

字典.values() 所有 value 列表

字典.items() 所有 (key, value) 元组列表

字典[key] 可以从字典中取值，key 不存在会报错

字典.get(key) 可以从字典中取值，key 不存在不会报错

del 字典[key] 删除指定键值对，key 不存在会报错

字典.pop(key) 删除指定键值对，key 不存在会报错

字典.popitem() 随机删除一个键值对

字典.clear() 清空字典

字典[key] = value

如果 key 存在，修改数据

如果 key 不存，新建键值对

字典.setdefault(key, value)

如果 key 存在，不会修改数据

如果 key 不存在，新建键值对

字典.update(字典2) 将字典 2 的数据合并到字典 1

1 字典的定义：字典名 = {key1:value1, key2:value2, key3:value3, ...}

- 字典使用大括号 {} 包起来
- 键值对之间使用 , 间隔

2 字典的常见操作

- 添加元素：字典名[key] = value

- 删除元素
 - `del` 字典名
 - `del` 字典名[键]
 - 字典名.`pop`(键)
 - 字典名.`popitem`()
 - 字典名.`clear`()
- 修改元素：字典名[key] = value
- 查询元素
 - 字典[key]
 - `get`(key, 默认值)
 - `keys`()
 - `values`()
 - `items`()

3 字典的遍历

4 字典的特征

- **可变容器**：字典中的数据是可变的，和列表一样，是可变容器，而元组是不可变容器
- **键的唯一性**：字典中的键必须是唯一的，如果同一个键被赋值多次，那么后面的值会把前面的值覆盖
- **键的类型限制**：字典使用哈希表存储，字典的键必须是可哈希的，也就是必须是不可变类型，一般建议设置为字符串类型
- **有序性**：需要说明的是，在Python3.7之前，字典是无序的；而在3.7之后，字典使用数组保存插入顺序，所以当前我们认为字典有序

练一练：

- 1** 小明去超市购买水果，购买苹果花了32.8元，香蕉 22元， 葡萄 15.5元

请你存储以上信息并计算小明总共花了多少元。

- 2** 小明，小刚去超市里购买水果。

小明购买了苹果，草莓，香蕉，一共花了89块钱

小刚购买了葡萄，橘子，樱桃，一共花了87块钱

请从上面的描述中提取数据，存储到合适的容器中，可以根据姓名获取这个人购买的水果种类和总费用。

集合

学习目标：

- 掌握集合的定义
- 理解集合的两大特点：**无序**、**不可重复**
- 掌握集合的常见操作：去重、添加元素、删除元素

定义

集合（set）是一个无序的不重复元素序列。

集合中的元素不会重复，并且可以进行交集、并集、差集等常见的集合操作。

集合主要用来去重。

可以使用大括号 `{ }` 创建集合，元素之间用逗号 `,` 分隔， 或者也可以使用 `set()` 函数创建集合。

```
1  # 1. 创建一个空集合
2  my_set = set()
3
4  # !!! 创建空集合不能写一下的形式，因为这样写是用来创建空字典
5  myset = {}
6
7  # 2. 创建一个含有元素的集合
8  myset = {10, 20, 30, 40}
9  myset = {'a', 'b', 'c', 'd'}
10
11
12 # 3. 通过别的序列来创建集合
13 myset1 = set("wangdao")           # 通过字符串创建集合
14 myset2 = set([10, 20, 30, 40, 10, 20]) # 通过列表创建集合
15 myset3 = set((10, 20, 30, 40, 10, 20)) # 通过元组创建集合
```

常见操作

集合主要用来**去重**，其他的了解即可。

去重

当一个列表中的元素有重复值的时候，我们可以使用集合进行去重。

```
1 my_list = ['a', 'b', 1, 'a', 'b']
2 my_set = set(my_list)      # 列表转集合
3 print(my_set)              # {1, 'b', 'a'}
4
5 my_list = list(my_set)     # 集合转列表
6 print(my_list)             # [1, 'b', 'a']
```

其他操作

集合内置方法完整列表

编号	方法	描述
1	<code>add()</code>	为集合添加元素
2	<code>clear()</code>	移除集合中的所有元素
3	<code>copy()</code>	拷贝一个集合
4	<code>difference()</code>	返回多个集合的差集
5	<code>difference_update()</code>	移除集合中的元素，该元素在指定的集合也存在。
6	<code>discard()</code>	删除集合中指定的元素
7	<code>intersection()</code>	返回集合的交集
8	<code>intersection_update()</code>	返回集合的交集。
9	<code>isdisjoint()</code>	判断两个集合是否包含相同的元素，如果没有返回 <code>True</code> ，否则返回 <code>False</code> 。
10	<code>issuperset()</code>	判断该方法的参数集合是否为指定集合的子集
11	<code>pop()</code>	随机移除元素
12	<code>remove()</code>	移除指定元素
13	<code>symmetric_difference()</code>	返回两个集合中不重复的元素集合。
14	<code>symmetric_difference_update()</code>	移除当前集合中在另外一个指定集合相同的元素，并将另外一个指定集合中不同的元素插入到当前集合中。
15	<code>union()</code>	返回两个集合的并集
16	<code>update()</code>	给集合添加元素
17	<code>len()</code>	计算集合元素个数

- 添加元素
 - `add()`
 - `update()`
- 删除元素
 - `remove()`
 - `discard()`

- pop()
- 集合还支持特定的运算符来求交集、并集和差集
 - &: 交集
 - -: 差集
 - |: 并集

```

1  # 1. 定义一个集合
2  my_set = {'a', 'b', 1, 2}
3
4  # 2. 添加元素
5  my_set.add('张飞')           # {1, 2, 'a', 'b', '张飞'}
6  my_set.add('a')             # {1, 2, '张飞', 'a', 'b'}
7
8  my_set.update('李四')       # {'b', 2, 1, '四', '李', '张
   飞', 'a'}
9  my_set.update('A')         # {1, 2, 'A', 'b', '张飞',
   '李', '四', 'a'}
10 my_set.update(["李云龙", "楚云飞"]) # {'a', 1, 2, '楚云飞', 'b',
   '李', '四', '张飞', 'A', '李云龙'}
11
12 # 3. 删除元素
13 my_set.remove('A')          # {1, 2, '李', 'b', '李云龙',
   'a', '四', '楚云飞', '张飞'}
14 # my_set.remove("丁伟")    # 丁伟不存在于集合，会报错
15
16 my_set.discard('李云龙')    # {1, 2, '四', '张飞', '李',
   'a', '楚云飞', 'b'}
17 my_set.discard("丁伟")     # 丁伟不存在于集合，不会报错
18
19 random_element = my_set.pop()
20 print(random_element)      # 重复运行，每次不一样（pop是随机
   删除一个元素并返回它的值）
21
22
23 # 4. 求交集、并集和差集
24 set1 = {1, 2, 3, 4}
25 set2 = {3, 4, 5, 6}
26
27 # 并集
28 union_set = set1 | set2
29 print(union_set)           # 输出: {1, 2, 3, 4, 5, 6}
30
31 # 差集
32 diff_set = set1 - set2
33 print(diff_set)            # 输出: {1, 2}
34

```

```
35 # 交集
36 inter_set = set1 & set2
37 print(inter_set) # 输出: {3, 4}
```

总结

集合主要是用来去重。

- 集合的定义

- 创建空集合

```
my_set = set()
```

- 创建有元素的集合

```
my_set = {元素1, 元素2, 元素3 ...}
```

- 通过序列创建集合(这一步会去重)

```
my_set = set(序列名)
```

- 集合的常用操作

- 去重

- 添加元素

- add()
 - update()

- 删除元素

- remove()
 - discard()
 - pop()

- 求交集、并集和差集

公共操作

学习目标：

- 1 掌握数据容器的公共运算符
- 2 掌握数据容器的公共方法

对于Python中的容器类型，有很多公共的操作与方法，分为以下两类：

- 公共运算符
- 公共方法

公共运算符

运算符	描述	支持的容器类型
+	合并	字符串、列表、元组
*	复制	字符串、列表、元组
in	元素是否存在(字典是key)	字符串、列表、元组、字典、集合
not in	元素是否不存在 (字典是key)	字符串、列表、元组、字典、集合
切片运算符	[start: end: step] , 对序列进行截取	字符串、列表、元组

```
1  # 1. 合并运算符 +
2
3  # 对列表合并
4  list1 = ['zs', 'ls', 'ww']
5  list2 = [1, 2, 3, 4]
6
7  print(list1 + list2)      # 输出: ['zs', 'ls', 'ww', 1, 2, 3, 4]
8
9  # 对元组合并
10 tup1 = ('a', 'b', 'c')
11 tup2 = (1, 2, 3)
12 print(tup1 + tup2)       # 输出: ('a', 'b', 'c', 1, 2, 3)
13
14 # 对字符串合并
15 s1 = 'hello'
16 s2 = 'Kaoyan'
17 print(s1 + s2)           # 输出: helloKaoyan
18
19 #####
20 # 2. 复制运算符 *
21
22 # 对列表复制
```

```

23 list3 = [1, 2]
24 new_list2 = list3 * 3
25 print(new_list2)          # 输出: [1, 2, 1, 2, 1, 2]
26
27 # 对元组复制
28 tuple3 = (1, 2)
29 new_tuple2 = tuple3 * 3
30 print(new_tuple2)        # 输出: (1, 2, 1, 2, 1, 2)
31
32 # 对字符串复制
33 s3 = 'abc'
34 print(s3 * 3)            # 输出: 'abccabccabc'
35
36 #####
37 # 3. in 与 not in
38
39 # 判断列表
40 list4 = [10, 20, 30]
41 print(20 in list4)        # 输出: True
42 print(40 not in list4)    # 输出: True
43
44 # 判断元组
45 tuple4 = (10, 20, 30)
46 print(20 in tuple4)      # 输出: True
47 print(40 not in tuple4)   # 输出: True
48
49
50 # 判断字符串
51 s4 = 'abc你好'
52 print('a' in s4)          # 输出: True
53 print('宝' not in s4)     # 输出: True
54
55 # 判断集合
56 set3 = {1,2,3,4,'a','b','c'}
57 print(1 in set3)
58 print('A' not in set3)
59
60 # 判断字典
61 dict3 = {'name':'张三','age':20,'height':180}
62 print('name' in dict3)
63 print('20' in dict3)
64 print('gender' not in dict3)

```

对于比较运算符，其实部分容器也是支持的。

容器类型	<code>==</code> / <code>!=</code> (相等 / 不等)	<code>></code> / <code><</code> / <code>>=</code> / <code><=</code> (大于 / 小于 / 等)
列表 / 元组	比较 元素、顺序、数量	按 元素顺序逐个比较 ，短的前缀容器更小
集合	比较 元素 (与顺序无关)	判断 子集 和 超集 关系(用的很少)
字典	比较 键值对 (与顺序无关)	不支持 ，会抛出 <code>TypeError</code>

```

1  # --- 列表 (List) 比较 ---
2  list_a = [1, 2, 3]
3  list_b = [1, 2, 3]
4  list_c = [1, 2, 4]
5  list_d = [1, 2]
6  list_e = [1, 3, 2]
7
8  print(f"list_a == list_b: {list_a == list_b}")  # True (所有元素、顺序都相同)
9  print(f"list_a != list_c: {list_a != list_c}")  # True (元素不同)
10
11 # 大小比较
12 print(f"list_a > list_c: {list_a > list_c}")      # False (比较到第三个元素 3 > 4 为 False)
13 print(f"list_c > list_a: {list_c > list_a}")      # True (比较到第三个元素 4 > 3 为 True)
14 print(f"list_a > list_d: {list_a > list_d}")      # True (list_d是list_a的前缀，且更短)
15
16 # 元组 (Tuple) 比较，规则和列表一样
17 tuple_a = (1, 2, 3)
18 tuple_b = (1, 2, 3)
19 tuple_c = (1, 2, 4)
20
21 print(f"\ntuple_a == tuple_b: {tuple_a == tuple_b}")  # True
22 print(f"tuple_a < tuple_c: {tuple_a < tuple_c}")      # True (3 < 4)
23
24 # 混合类型比较会报错
25 # mixed_list_1 = [1, 2]
26 # mixed_list_2 = [1, 'a']
27 # print(mixed_list_1 > mixed_list_2) # TypeError: '>' not supported between instances of 'int' and 'str'

```

公共方法

编号	函数	字符串(str)	列表(list)	元组(tuple)	字典(dict)	集合(set)	描述与说明
0	del	✗	✓	✗	✓	✗	删除容器中的元素: 从列表、字典等可变容器中移除指定的元素或键值对。
1	len()	✓	✓	✓	✓	✓	获取长度: 返回容器中元素的数量。
2	max()	✓	✓	✓	✓	✓	获取最大值: 返回容器中“最大”的元素。
3	min()	✓	✓	✓	✓	✓	获取最小值: 返回容器中“最小”的元素。
4	sum()	✗	✓	✓	✗	✓	求和: 返回容器中所有元素的总和（元素必须是数字类型）。
5	sorted()	✓	✓	✓	✓	✓	排序: 返回一个 新的 已排序的列表，不改变原容器。
6	reversed()	✓	✓	✓	✓	✗	反转: 返回一个反转迭代器。
7	enumerate()	✓	✓	✓	✓	✓	枚举: 返回一个枚举对象，包含索引和值。

```
1  # 定义容器的变量
2  str1 = "abcd"
3  list1 = ['a', 'b', 'c', 'd']
4  tup1 = ('a', 'b', 'c', 'd')
5  dict1 = {'a':1, 'b':2, 'c':3, 'd':4}
6  set1 = {'a', 'b', 'c', 'd'}
7
8  # del
9  # del str1[0]           # 会报错，因为字符串不可变
10 # del list1[0]
11 # del tup1[0]          # 会报错，因为元组不可变
12 # del dict1['a']       # 删除键对应的键值对
13 # del set1[0]         # 会报错，因为集合无序，不能通过下标取值
14
15 # len()
16 print(len(str1))
17 print(len(list1))
18 print(len(tup1))
19 print(len(dict1))      # 获取键值对的个数
20 print(len(set1))
21
22 # max()
23 print(max(str1))
24 print(max(list1))
25 print(max(tup1))
26 print(max(dict1))      # 获取最大的键
27 print(max(set1))
28
29 # min()
30 print(min(str1))
31 print(min(list1))
32 print(min(tup1))
33 print(min(dict1))      # 获取最小的键
```

```

34 print(min(set1))
35
36 print('*'*30)
37 # sum() 元素必须是数字类型
38 # print(sum(str1)) # 不支持
39 # print(sum(list1))
40 # print(sum(tup1))
41 # print(sum(dict1)) # 不支持
42 # print(sum(set1))
43
44
45 # sorted()
46 print(sorted(str1,reverse=True))
47 print(sorted(list1,reverse=True))
48 print(sorted(tup1,reverse=True))
49 print(sorted(dict1,reverse=True)) # 返回key排序之后的列表
50 print(sorted(set1,reverse=True))
51
52 # reversed()
53 re_str1 = reversed(str1) # 返回的都是可迭代对象，打印
    的是对象的内存地址值
54 re_list1 = reversed(list1)
55 re_tup1 = reversed(tup1)
56 re_dict1 = reversed(dict1) # 字典是key的反转的可迭代对
    象
57 # print(reversed(set1)) # 无序，不可反转
58
59 # 迭代遍历
60 for m in re_dict1:
61     print(m)
62
63 # enumerate()
64 enum_str1 = enumerate(str1)
65 enum_list1 = enumerate(list1)
66 enum_tup1 = enumerate(tup1)
67 enum_dict1 = enumerate(dict1) # 键的下标，用的很少
68 enum_set1 = enumerate(set1) # 因为无序，所以每一个元素的下
    标是随机的
69
70 print(type(enum_str1))
71 print(type(enum_list1))
72 print(type(enum_tup1))
73 print(type(enum_dict1))
74 print(type(enum_set1))
75
76 # (0, 'a')
77 # (1, 'b')
78 # (2, 'c')

```

```
79 # (3, 'd')
80 for s in enum_str1:
81     print(s)
```

推导式

学习目标:

- 掌握列表、集合、字典推导式的使用

什么是推导式

推导式是 Python 中一种非常简洁、强大的语法，用于从一个可迭代对象（如列表、元组、字符串等）创建新的序列。

它的核心优势在于：

- **代码简洁**：将一个 for 循环和一个 `append()` 操作压缩到一行代码中。
- **可读性高**：对于熟悉它的人来说，推导式比传统的 for 循环更直观。
- **执行效率高**：通常比等价的 for 循环稍快。

基本语法： `[表达式 for 项目 in 可迭代对象 if 条件]`

其实最重要的就是代码简洁，接下来举个例子体会一下。

比如需要创建一个0-9的列表

```
1 # while的写法
2
3 # 1. 准备一个空列表
4 list1 = []
5
6 # 2. 书写循环，依次追加数字到空列表list1中
7 i = 0
8 while i < 10:
9     list1.append(i)
10    i += 1
11
12 #####
13
14 # for的写法
15 list1 = []
16 for i in range(10):
```

```

17     list1.append(i)
18
19     #####
20
21     # 列表推导式的写法
22     list1 = [i for i in range(10)]

```

常见的推导式类型

Python主要有以下3中推导式类型，他们的语法非常相似。

- 列表推导式
- 字典推导式
- 集合推导式

推导式类型	语法结构	描述
列表推导式	<code>[expr for item in iterable if condition]</code>	创建一个 新的列表 。
字典推导式	<code>{key_expr: value_expr for item in iterable if condition}</code>	创建一个 新的字典 。
集合推导式	<code>{expr for item in iterable if condition}</code>	创建一个 新的集合 （自动去重）。

列表推导式

作用：根据一个旧列表，快速生成一个满足特定条件的新列表。

语法结构： `[expr for item in iterable if condition]`

示例 1：创建一个包含 0 到 9 所有偶数的列表

```

1  # 传统 for 循环：
2  even_numbers = []
3  for num in range(10):
4      if num % 2 == 0:
5          even_numbers.append(num)
6  print(even_numbers) # [0, 2, 4, 6, 8]
7
8  # 列表推导式
9  even_list = [num for num in range(10) if num % 2 == 0]
10 # 解读：for num in range(10) 遍历 0-9, if num % 2 == 0 筛选出偶数, num
    将筛选出的数作为新列表的元素。

```

示例2：给定字符串 `text = "Hello World Cskaoyan"`，创建一个列表，包含该字符串中每个单词的长度

```
1 text = "Hello World Cskaoyan"
2
3 # 传统 for 循环:
4 len_list = []
5 words = text.split()
6 for word in words:
7     len_list.append(len(word))
8 print(len_list)
9
10 # 列表推导式
11 len_list2 = [len(word) for word in text.split()]
```

示例3：给定字典 `stu_scores = {'Alice': 95, 'Bob': 88, 'Charlie': 92}`，创建一个列表，只包含所有学生的名字（即字典的键）

```
1 # for循环
2 name_list = []
3 for key in stu_scores.keys():
4     name_list.append(key)
5 print(name_list)
6
7 # 列表推导式
8 name_list2 = [name for name in stu_scores.keys()]
9 print(name_list2)
```

字典推导式

作用：根据一个可迭代对象，快速生成一个新的字典。

语法结构： `{key_expr: value_expr for item in iterable if condition}`

示例1：将一个列表转换为字典，键为列表元素，值为元素的长度

```
1 fruits = ['apple', 'banana', 'cat']
2
3 # for循环
4 f_dict1 = {}
5 for element in fruits:
6     f_dict1[element] = len(element)
7
8 # 字典推导式
9 f_dict2 = {e:len(e) for e in fruits}
10 # 解读: for e in fruits 遍历列表, e 作为键 (key_expr), len(e) 作为值 (value_expr)。
```

示例2: 创建一个字典，键为从 1 到 5 的整数，值为该整数的平方

```
1 # 传统的写法
2 dict3 = {}
3 for num in range(1, 6):
4     dict3[num] = num * num
5 print(dict3)
6
7 # 字典推导式
8 dict4 = {num: num * num for num in range(1, 6)}
```

示例3: 给定字典

`student_scores = {'Alice': 95, 'Bob': 88, 'Charlie': 92, 'David': 70}`，创建一个新字典，只包含分数大于等于 90 的学生。

```
1 student_scores = {'Alice': 95, 'Bob': 88, 'Charlie': 92, 'David': 70}
2
3 # 传统的写法
4 score1 = {}
5 for key in student_scores.keys():
6     if student_scores[key] >= 90:
7         score1[key] = student_scores[key]
8 print(score1)
9
10 # 字典推导式:
11 score2 = {name: student_scores[name] for name in student_scores.keys() if student_scores[name] >= 90}
```

集合推导式

作用：根据一个可迭代对象，快速生成一个新的集合（会自动去除重复元素）。

语法结构：`{expr for item in iterable if condition}`

注意：语法与列表推导式类似，但使用大括号 {}

示例1：获取一个列表中所有元素的平方，并去除重复值

```

1 numbers = [1, 2, 2, 3, 4, 4]
2
3 # for循环
4 set1 = set()
5 for i in numbers:
6     set1.add(i * i)
7 print(set1)
8
9 # 集合推导式
10 set2 = {num * num for num in numbers}
11 print(set2)

```

示例2：创建一个集合，包含指定句子中所有不重复的单词

```

1 sentence = "the quick brown fox jumps over the lazy dog"
2 # for循环
3 set3 = set()
4 for word in sentence.split():
5     set3.add(word)
6 print(set3)
7
8 # 集合推导式
9 set4 = {word for word in sentence.split()}

```

示例3：给定字典 `product_prices = {'apple': 1.5, 'banana': 0.8, 'orange': 1.2, 'grape': 1.5}`，创建一个集合，包含所有不重复的价格

```

1 product_prices = {'apple': 1.5, 'banana': 0.8, 'orange': 1.2,
2 'grape': 1.5}
3 # for循环
4 set5 = set()
5 for k,v in product_prices.items():
6     set5.add(v)
7 print(set5)
8
9 # 推导式
10 set6 = {v for k,v in product_prices.items()}
11 print(set6)

```

练一练：

- 1 给定元组 `my_tuple = ('apple', 'banana', 'cherry')`，创建一个列表，包含元组中每个元素的大写形式。
- 2 创建一个列表，对于 0 到 9 的每个整数，如果是偶数则放入字符串 "even"，如果是奇数则放入字符串 "odd"。

3 假设有一个包含文件路径的列表

`file_paths = ['report.pdf', 'image.png', 'archive.zip', 'notes.txt']`, 创建一个新列表, 只包含其中所有 `.txt` 文件的名称 (不含扩展名)。

4 给定元组列表

`student_data = [('Alice', 95), ('Bob', 88), ('Charlie', 92)]`, 创建一个 `{name: score}` 形式的字典。

5 给定字典 `code_to_name = {101: 'Math', 102: 'Physics', 103: 'Chemistry'}`, 创建一个新字典, 将键和值互换。

6 将两个列表 `['name', 'age', 'gender']` 和 `['Tom', 20, 'man']` 合并为一个字典

7 给定列表 `sizes = ['S', 'M', 'L']` 和 `colors = ['Red', 'Blue']`, 创建一个字典, 键为 `'Red-S'` 这样的组合, 值为 `(color, size)` 这样的元组。

8 使用上面的 `student_scores` 字典, 创建一个新字典, 键不变, 值变为对应的等级 (例如, 90 分及以上为 'A', 80-89 为 'B', 否则为 'C')。

9 给定字符串 `text = "Hello, World!"`, 创建一个集合, 包含字符串中所有不重复的小写字母 (忽略大小写和非字母字符)。