第五章 复合数据类型

主要内容

- 复合数据类型
- 序列的访问及运算符
- 列表使用
- 元组使用
- 集合使用
- 字典使用

数据容器

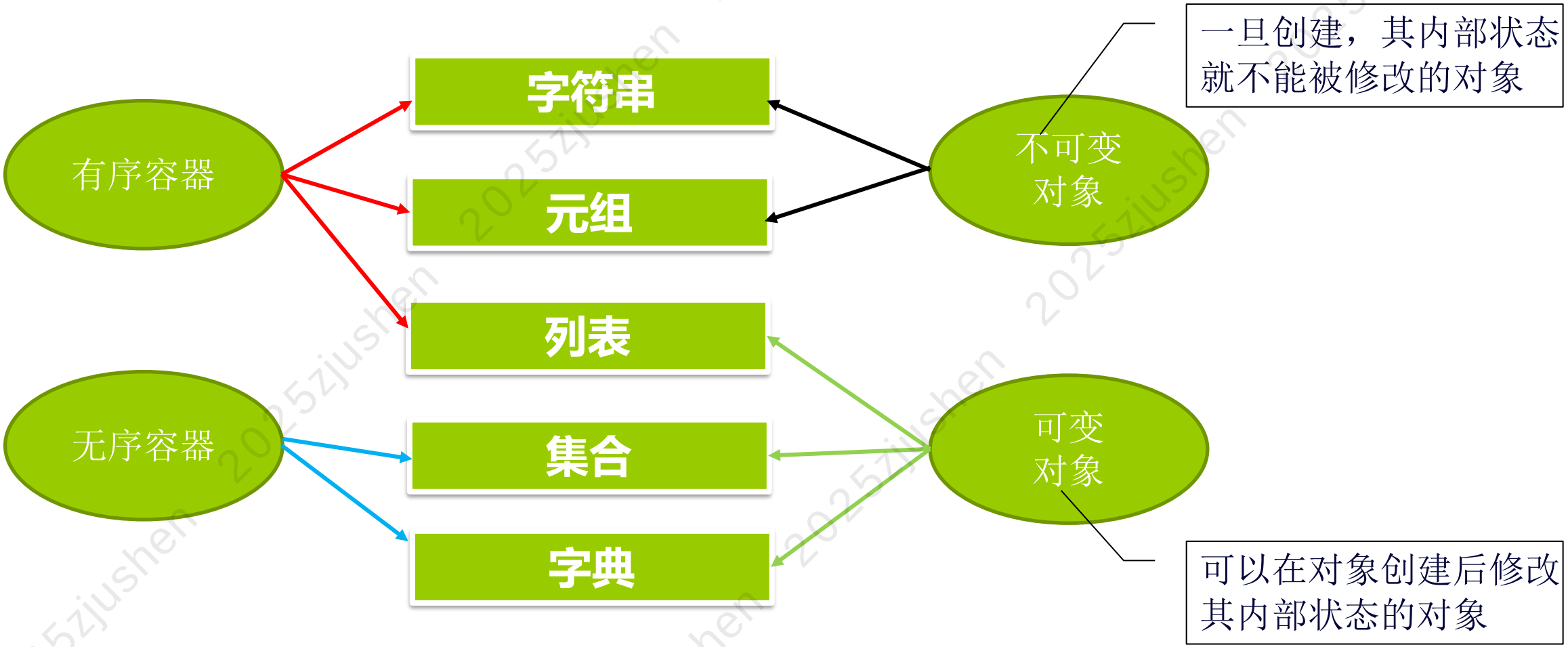
数据容器

为满足程序中复杂的数据表示，Python支持组合数据类型，可以将一批数据作为一个整体进行数据操作，这就是数据容器的概念。

常用的容器类型：

序列（列表（list）,字符串（string）,元组（tuple））字典、集合

容器分类



序列概述

序列（Sequence）

这种容器中可包含多个数据（元素），容器中的数据（元素）有先后次序，每个元素通过用其下标（索引）来访问。序列的下标从0开始，后面下标依次为1,2,3,....。

注：序列是其中一大类数据容器的统称,不是具体的数据类型。

类型	字面量	可变性
字符串	'hello'	不可变对象
列表	['h','e','l','l','o']	可变对象
元组	('h','e','l','l','o')	不可变对象

通用的序列操作

所有的序列类型都可以进行的操作归纳如下表所示。

操作	描述
X1+X2	联接序列X1和X2,生成新序列
X*n	序列X重复n次，生成新序列
X[i]	引用序列X中下标为i的成员
X[i：j]	引用序列X中下标为i到j-1的子序列
X[i：j：k]	引用序列X中下标为i到j-1的子序列，步长为k
len(X)	计算序列X中成员的个数
max(X)	序列X中的最大值
min(X)	序列X中的最小值
v in X	检查v是否在序列X中，返回布尔值
v not in X	检查v是否不在序列X中，返回布尔值

访问单个数据

- 用[ ]来访问序列中的一个元素。比如访问字符串中的某个字符：

```
>>>prompt = 'hello'
>>>print(prompt[0])
h
>>>print(prompt[4])
o
```

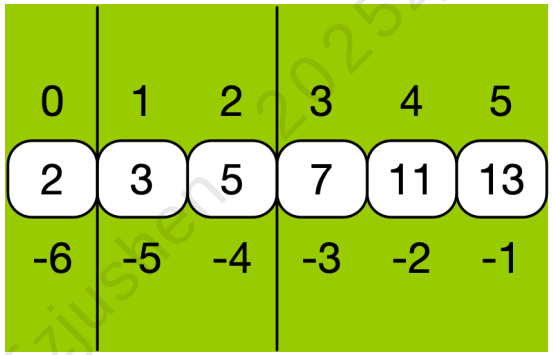
- 假设序列中的元素个数是n，下标的有效范围是0到n-1，或者-1到-n。
- 如果下标的绝对值大于n-1，则会发生下标越界错误。
- 如果下标的值为负数，表示从序列的最后一个元素往前引用，比如：

```
>>>prompt = 'hello'
>>>print(prompt[-1],prompt[-4])
o e
```

访问部分数据

如果要访问序列中的一部分元素，可以使用切片（slice）。切片通过冒号分隔两个下标来实现。如右图所示，切片a[1:3] 表示包含从下标1开始到下标3前面的下标2为止的部分元素的子序列（列表）。

```
>>>a = [2,3,5,7,11,13]
>>>print(a[1:3])
[3,5]
>>>print(a[0:5:2])
[2, 5, 11]
```



切片使用第3个参数，那么该参数表示切片选择元素的步长。

练习

序列a

请写出下列切片的结果：

1. a[1:-3] 切片使用负的下标访问

   结果：[3,5]

2. a[2:] 切片省略第2个下标

   结果：[5, 7, 11, 13]

3. a[:3] 省略第1个下标，第二个下标为正数

   结果：[2, 3, 5]

4. a[:-3] 省略第1个下标，第二个下标为负数

   结果：[2, 3, 5]

5. a[:-3:-1] 省略第1个下标，第二个下标为负数,第三个参数是负数时

   结果： [13, 11]

6. a[4:0:-1] 切片第3个参数为负数时

   结果是：[11, 7, 5, 3]

关于切片Tip：

- 步长是正数顺序访问
- 步长为负数逆序访问
- 步长不能为0

序列的运算符

- 加号（+）:连接2个序列

```
>>>a = [2, 3, 5, 7, 11, 13]
>>>b = [4,6,8,9,10,12]
>>>print(a+b)
[2, 3, 5, 7, 11, 13, 4, 6, 8, 9, 10, 12]
```

- 乘号（*）:重复序列

```
>>> [1,2,3]*3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```


序列的运算符

in: 可以检查特定的值是否是序列中的一部分。

```
>>>a = [2,3,5,7,11,13]
>>>3 in a
    True
>>> [2,3] in [2,3,5,7,11,13]
    False
>>>[2,3] in [[2,3],5,7,11,13]
    True
```

```
>>> ‘e’ in ‘hello’
    True
>>>‘he’ in ‘hello’
    True
```

计算函数

- len()函数返回序列内部元素的个数。

```
>>>len([2,3,5,7])
        4
>>>len('hello world')
        11
```

- min()和max()函数计算序列中的最小值和最大值

```
>>> max([2,3,5,7,11,13])
        13
>>>min('好好学习天天向上')
        '上'
```

注：字符串的大小是按照其Unicode 编码来比较的。

列表

列表概念

列表（list）

- 由一系列按照指定顺序排列的元素组成。列表中的元素可以是不同类型。
- 列表的表示用方括号[ ]将元素括起来，元素之间用逗号（,）分隔
- 列表是序列类型的一种，序列所有的特性和操作对于列表都是成立的，除此之外，列表还有自己的特殊操作。

创建列表

1. 直接使用列表的字面量。

```
a = [ ]     # 创建一个空列表
a = [2,3,5,7,11,13]
a = [0]*5  # 快速生成[0, 0, 0, 0, 0]
```

2. 使用list()将其他数据类型转换成一个列表。

```
a = list('hello')
a的内容是: ['h', 'e', 'l', 'l', 'o']
list(range(1,10,2))
结果是：[1,3,5,7,9]
```


列表的元素类型可以是任何类型，也包括列表类型。当列表的元素是列表时，可以构成多维列表，如同一个矩阵。

```
matrix = [
    [1, 2, 3, 4, 5],
    [3, 0, 8,11,14],
    [5, 6, 9,12,16],
    [7, 0, 0, 0, 0],
    [9,11,17, 0,15]
    ]
```

	[0]	[1]	[2]	[3]	[4]
[0]	1	2	3	4	5
[1]	3	0	8	11	14
[2]	5	6	9	12	16
[3]	7	0	0	0	0
[4]	9	11	17	0	15

- 用matrix[0][0]访问其中第一行第一列的元素
- 矩阵的每一行都是一个列表。

基本的列表操作

列表元素的赋值

和字符串不同，列表中的元素可以被修改。

```
a = [1,3,5,7,11]
a[0] = 2
print(a)
```

输出：

```
[2,3,5,7,11]
```

切片赋值

切片表示列表的一部分，可以被赋值，接受另外一个列表，替换切片那部分元素。

```
name = list('Perl')
name[2:] = list('ar')
print(name)
```

结果是：

```
['P', 'e', 'a', 'r']
```

基本的列表操作

- 删除元素

在 Python 中，del 是一个关键字，用于**删除对象的引用**。

它的核心作用是解除变量名与对象之间的绑定关系，而不是直接销毁对象本身。

```
name = [‘Alice’, ‘Kim’, ‘Karl’, ‘John’]
del name[2]      #解除了name[2]对‘Karl’字符串的引用，列表结构发生了变化
print(name)
```

结果是：

```
['Alice', 'Kim', 'John']
```


列表的方法

列表的常用方法	描述
L.append(x)	在列表L尾部追加x
L.clear()	移除列表L的所有元素
L.count(x)	计算列表L中x出现的次数
L.copy()	列表L的备份
L.extend(x)	将可迭代对象x扩充到列表L中
L.index(value[,start[,stop]])	计算在指定范围内value的下标
L.insert(index,x)	在下标index的位置插入x
L.pop(index)	返回并删除下标为index的元素，默认是最后一个
L.remove(value)	删除值为value的第一个元素
L.reverse()	倒置列表L
L.sort()	对列表元素排序

列表的方法

- 追加：append()

用append()在列表后面增加一个元素。

```
a = [2,3,5,7,11]
a.append(13)
print(a)
```

结果是：

```
[2, 3, 5, 7, 11, 13]
```


列表的方法

- 扩展：extend()

用extend()把可迭代对象中的元素逐一添加到列表的后面。

```
lst = [1, 2]

lst.extend([3,4])

lst.extend("ab")

print(lst)结果就是：[1, 2, 3, 4, 'a', 'b']
```


列表的方法

- 插入：insert()

insert(index, obj)用来将一个数据（obj）插入到列表的指定位置（index）。

```
a = [2,3,5,6,11]
a.insert(2,4)
a.insert(12,13)    #指定插入的位置(12)不存在，加到最后
print(a)
```

结果是：

```
[2, 3, 4, 5, 6, 11, 13]
```

列表的方法

- 查找：index()

index()用于在列表中查找某个数据第一出现的位置（下标）。

```
a = [100,120,101,35]
print(a.index(120))
```

输出：

```
1
```

注:如果查找的数据在列表中不存在，则会发生错误。（怎么避免错误发生？）

列表的方法

- 删除：remove()

用remove()删除某个数据在列表中第一次出现的元素。

a = [2,3,5,7,5,11]

a.remove(5)

print(a)

输出：

[2, 3, 7, 5,11]

注:如果要删除的数据不在列表中，则会发生错误。

- 易错点：

```
a=[1,2,3,3,5]
for i in a:
    if i==3:
        a.remove(i)
print(a)
```

期望输出结果为：[1, 2, 5]

实际输出结果为：[1, 2, 3, 5]

怎么修改？

```
a=[1,2,3,3,5]
for i in a[:]:
    if i==3:
        a.remove(i)
```

列表的方法

- 弹出：pop()

用pop()删除并返回列表中指定下标（位置）的数据，如果不指定下标，则删除最后一项。

```
a = [2,3,5,7,11]
print(a.pop())
print(a.pop(2))
print(a)
```

输出：

```
11
5
[2, 3, 7]
```


列表的方法

- 反转：reverse()

用reverse()将列表反转。该方法没有返回值。

```
a = [2,3,5,7,11]
a.reverse()
print(a)
```

结果是：

```
[11, 7, 5, 3, 2]
```


列表的方法

- 排序：sort()

sort()用于将列表中数据按一定规则进行排序。该方法没有返回值。

可以设置参数reverse，**reverse = True** 降序，**reverse = False** 升序（默认值）。

```
a = [8,2,10,5,3,11]
a.sort()
print(a)
```

输出：

```
[2, 3, 5, 8, 10, 11]
```

```
a = [8,2,10,5,3,11]
a.sort(reverse = True)
print(a)
```

输出：

```
[11, 10, 8, 5, 3, 2]
```


列表方法的实际应用（*）

append() + pop() 是天然的栈（**后进先出，LIFO**）操作组合操作，时间复杂度均为O(1)。如撤销、括号匹配、路径回退、DFS 等。

append() + pop(0) 可以实现队列（先进先出，**FIFO**），但效率比较低，因此可以使用 collections.deque（一种队列的数据结构）。

有兴趣的同学可以使用deepseek进行探究学习。

列表推导式

列表推导式是将某种操作应用到序列，从一个或者多个列表快速简洁地创建新列表的一种方法，又被称为列表解析。

它还可以将循环和条件判断结合，从而避免语法冗长的代码，同时提高程序性能。

列表推导式

基本推导式：这里的 expression 可以是多种形式，算术表达式、函数、条件表达式等等

```
[ expression  for  item  in  iterable ]
nl = [2*number  for  number  in [1,2,3,4,5] ]    #结果是：[2, 4, 6, 8, 10]
slen = [len(s)  for s  in ['apple','banana','peach','watermelon']] #结果是：[5, 6, 5, 10]
cl = [number if number % 2 else -number  for number in range(1,8)] #结果是：[1, -2, 3, -4, 5, -6, 7]
```

嵌套循环：可以实现多层迭代逻辑，其执行顺序与传统的嵌套 for 循环一致

```
[exp for outer in outer_iterable for inner in inner_iterable]
pl = [(x, y) for x in range(1,3) for y in range(3)]  #结果是：[(1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]
```

条件过滤：添加条件过滤，将原有迭代对象中符合条件的元素找出来形成新列表

```
[expression  for  item in  iterable  if  condition]
nl = [ number  for number  in  range(1,8)  if number % 2 == 1]   #结果是：[1, 3, 5, 7]
```

推导式练习

- 生成数列求和

  要求：计算1+1/2+...+1/20之和

  实现：sum([1/i for i in range(1,21)]) #结果为：3.597739657143682

- **多条件文本处理**

  要求：从words中筛选出同时满足长度大于等于5且以”s“结尾单词，并转为全大写。 words = ['apples','Students','cats','people’]

  实现：[word.upper() for word in words if len(word)>=5 and word[-1]=='s‘]

  #结果为： ['APPLES', 'STUDENTS']

元组

元组概念

- 元组（tuple）

元组是不可修改的任何类型的数据序列。元组像列表一样可以表达任何类型、任意数量的数据的序列。

元组的字面量用圆括号( )而不是方括号[ ]。

(1, 3.2, 5, 7.0, 9)

('not', 'and', 'or')

创建元组

1.用元组的字面量

```
d = (100,20)
print(d)
```

输出：

```
(100, 20)
```

注意：若元组中只有一个元素要写成(100,)

2.用tuple()方法，把其他序列类型转换成元组。

```
a = tuple([2,3,5,7,11])
print(a)
```

输出：

```
(2, 3, 5, 7, 11)
```

元组使用

- 元组是不可修改的，即不能对元组中的元素进行增加，删除，修改或排序。
- 元组不如append()、insert()、remove()等会修改内容的方法。
- 元组常用方法

元组常用方法	描述
T.count(x)	计算x元素出现的次数
T.index(x)	计算X元素的下标

元组的优点

- 元组速度比列表快。
- 元组不允许修改数据，可以起到数据保护作用
- 元组可以作为字典的“键”，也可做为集合的元素，而列表不可以。

适用场景：存储常量数据、字典键、函数返回值等需保证数据完整性的场景。

练习题

统计单词

求一句英文句子的单词数。单词是字母数字串，中间没有空格。

```
sentence="This    is a  pen   "
words=sentence.split()
print(len(words))
```

程序运行：

4

打分程序

- 设计一个打分程序，计算去掉一个最高分，一个最低分后一名选手的最后平均得分。

思路：输入若干分数

如何获得最高分和最低分

如何去掉这两个分数

如何计算平均分

```
score = list(map(float,input().split()))

slen = len(score)
smin = min(score)
smax = max(score)

score.remove(smin)
score.remove(smax)


avescore = sum(score) / (slen - 2)

print(avescore)
```

例题：排序

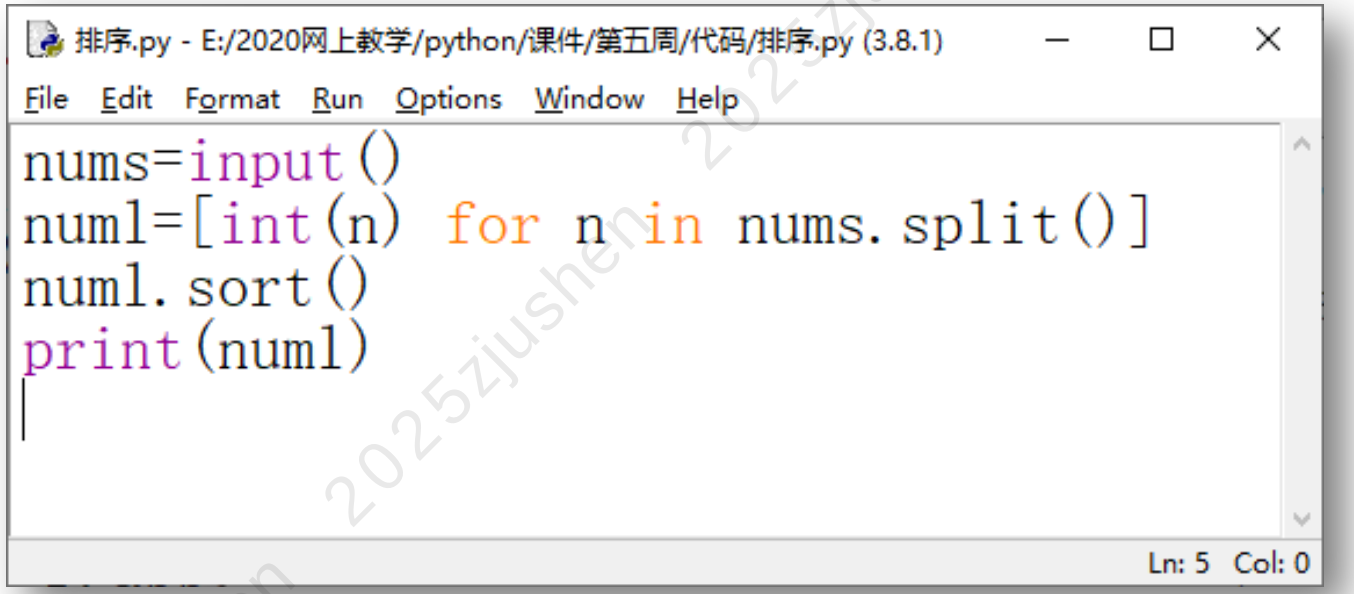
- 在一行中输入若干个整数，至少输入一个整数，整数之间用空格分割，要求将数据按从小到大排序输出。

程序输入：

5 -76 8 345 67

程序输出：

[-76, 5, 8, 67, 345]



输入四个字符串，求这些字符串的最大长度

思路：

输入字符串

计算字符串长度

比较长度

输出结果

```
length = 0
for i in range(4):
    a = len(input())
    if a > length:
        length = a
print(length)
```

取首字母并变成大写

题目要求：提示用户输入名字（拼音）姓与名之间用空格分隔，然后输出姓名中的首字符大写组合

如：

```
请输入名字：zhang san
ZS
```

思路：

输入姓名

取首字符并转换

合并

输出结果

```
name = input("请输入名字：").split()
n = [i[0].upper() for i in name]
outn = ''.join(n)
print(outn)
```

字符串常用方法

- 聚合字符串：字符串的join()方法

将一个可迭代对象（列表、元组、字符串）中各个**字符串类型**的元素以指定的字符串（即调用方法的字符串）连接组合成新一个字符串。

该方法在需要**高效合并多个字符串**时是首选工具，尤其适合处理结构化数据（如CSV、路径、日志等）。通过指定分隔符，可以灵活生成符合格式要求的字符串。

例1:

```
separator = “|”    # 分隔符是调用方法的字符串
result = separator.join([“a”, “b”])      #   "a|b”
```

例2:

```
d = [2025,3,25]
separator = "-"
result = separator.join([str(i) for i in  d])  # "2025-3-25"
```

输出图形

题目要求：提示用户输入图形的行数，然后输出如下图形

```
请输入行数：5
    a
   bbb
  ccccc
 ddddddd
eeeeeeeee
```

```
n = int(input('请输入行数:'))
for i in range(n):
    print(f"{' '*(n-1-i)}{chr(ord('a')+i)*(2*i+1)}")
```

思路：

输入行数

循环输出每一行

每一行可以看着两部分空格和字母

加密

编写一个明文密文转换程序。

加密方法：凯撒密码（A->C、B->D…Y->A、Z->B），

如：输入：CHINA

输出：EJKPC

方法一：生成密码表，查表转换

```
table = [chr(i) for i in range(65,91)]
new_table = table[2:]+table[:2]
s = input()
news = [new_table[table.index(i)] for i in s]
print(''.join(news))
```

方法二：使用字符串的translate方法

```
import string

pt=string.ascii_uppercase
ct=pt[2:]+pt[:2]

table=str.maketrans(pt,ct)

ins=input("请输入明文:")
outs=ins.translate(table)
print(outs)
```

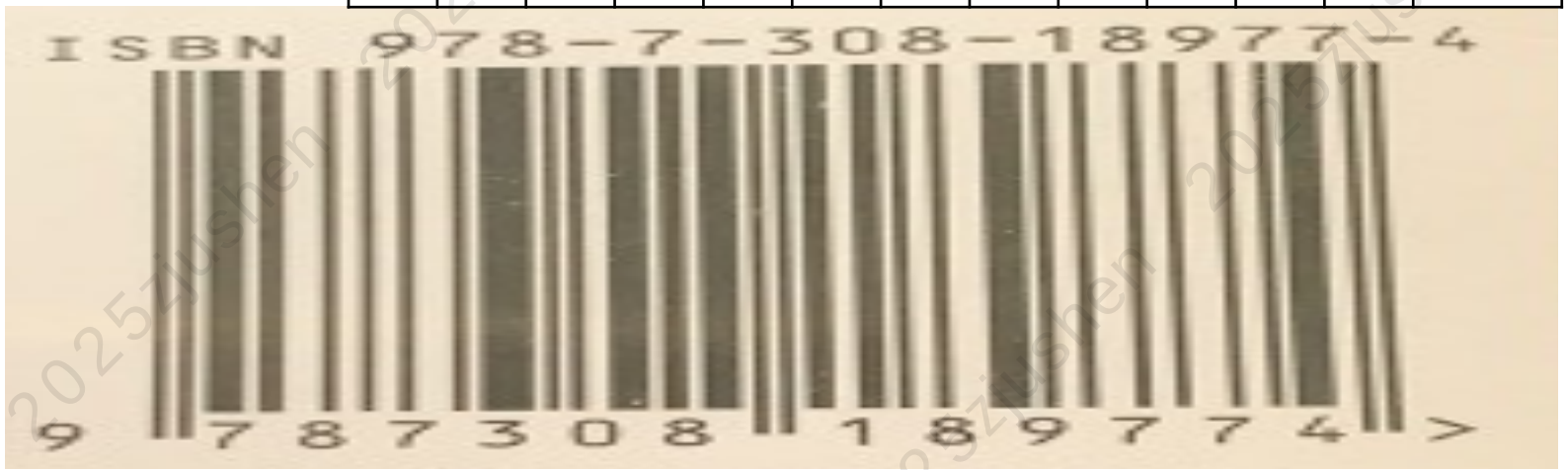
课后习题：书号验证

设计一个查询书号是否正确的程序，要求输入书号，并判断该书号是否正确

书号的验证是通过最后一位的验证码来进行。我们以13位的书号为例：

前12位数依次乘以1和3，然后求它们的和除以10的余数，最后用10减去这个余数，就得到了校验码。如果余数为0，则校验码为0，以此类推

9	7	8	7	3	0	8	1	8	9	7	7	4
1	3	1	3	1	3	1	3	1	3	1	3	
9	21	8	21	3	0	8	3	8	27	7	21	136



```python
#ISBN13位版本
isbn = input("请输入13位ISBN: ").split('-')
s=''.join(isbn)
w = sum([int(s[i]) if i%2 == 0 else int(s[i])*3 for i in range(12)])
m = w % 10
n = 10 - m
#p = '0' if n==10 else str(n)
if n == 10:
    p = '0'
else:
    p = str(n)
if p == isbn[4]:
    print("书号正确")
else:
    print("书号错误")
```


数据科学：集中趋势度量——均值、中值和众数（*）

- 均值（Mean）：均值是所有数据值加起来除以数据点的数量。
- 中值（Median）：中值是将数据集按大小顺序排列后位于中间位置的值。

  如果数据集数量是奇数，中值是中间的数值；

  如果是偶数，则是中间两个数值的平均值；
- 众数（Mode）：众数是一组数据中出现次数最多的值。

课后任务：

了解statistics模块

```
# 使用statistics模块
from statistics import *
data = [98, 76, 88, 49, 65, 97, 65, 76]
print(mean(data),median(data),mode(data),sep = ',')
```

集合

集合概念

集合（set）是一类容器，元素没有先后顺序，并且元素的值不重复。

集合的字面量用花括号{ }

{'apple', 'orange', 'pear', 'banana'}

{1,5,7}

	列表（List）	集合（Set）
有序性	有序	无序
重复性	允许重复元素	不允许重复元素
索引访问	支持	不支持
元素访问方式	通过索引访问	通过遍历访问
用途	需要保持元素顺序或允许重复元素的场景	需要去重或快速判断元素是否存在的场景
操作复杂度	索引访问O(1)，查找O(n)	查找、插入、删除O(1)

功能/方法	列表（List）	集合（Set）
添加元素	append(x), extend(iterable), insert(i, x)	add(elem)
移除元素	remove(x), pop(i), clear()	remove(elem), discard(elem), pop(), clear()
元素是否存在	x in list（使用in关键字）	elem in set（使用in关键字）
访问/获取元素	通过索引：list[index]	不支持索引访问
元素个数	len(list)	len(set)
排序	sort(), reverse()（原地排序）	无（但可以使用sorted(set)得到排序后的列表）
复制	copy()	copy()
迭代	支持for循环迭代	支持for循环迭代
成员测试后获取索引	index(x)（若不存在则抛出ValueError）	无（集合无序，不支持索引）
计数	count(x)	无（集合不允许重复元素）
集合操作	无	union(), intersection(), difference(), symmetric_difference(), issubset(), issuperset()

创建集合

- 直接给变量赋值一个集合字面量

```
fruit = {'apple', 'orange', 'pear', 'banana'}
```

- 使用set()创建一个空集合

```
emp = set()
```

注：emp = {} #创建一个空字典

- 使用set()将列表或元组转换成集合

```
prime = set([1,3,5,7,11])
```

结果是：

```
{1, 3, 5, 7, 11}
```

注：

因为集合的值不重复，所以创建集合的时候，python会消除重复的值。

```
fruit = {'apple', 'orange', 'apple', 'pear', 'orange' }
```

结果是：

```
{'orange', 'pear', 'apple'}
```

增加和删除集合元素

- 增加元素：

  add()：单个元素

  update()：多个元素（以可迭代对象形式进行添加）

- 删除元素：

  discard() 如果存在则删除，不存在无效果

  remove() 如果存在则删除，不存在会抛出异常

  pop() 如果存在删除任意一个元素并返回，不存在会抛出异常

  clear() 删除所有元素，剩下一个空集合

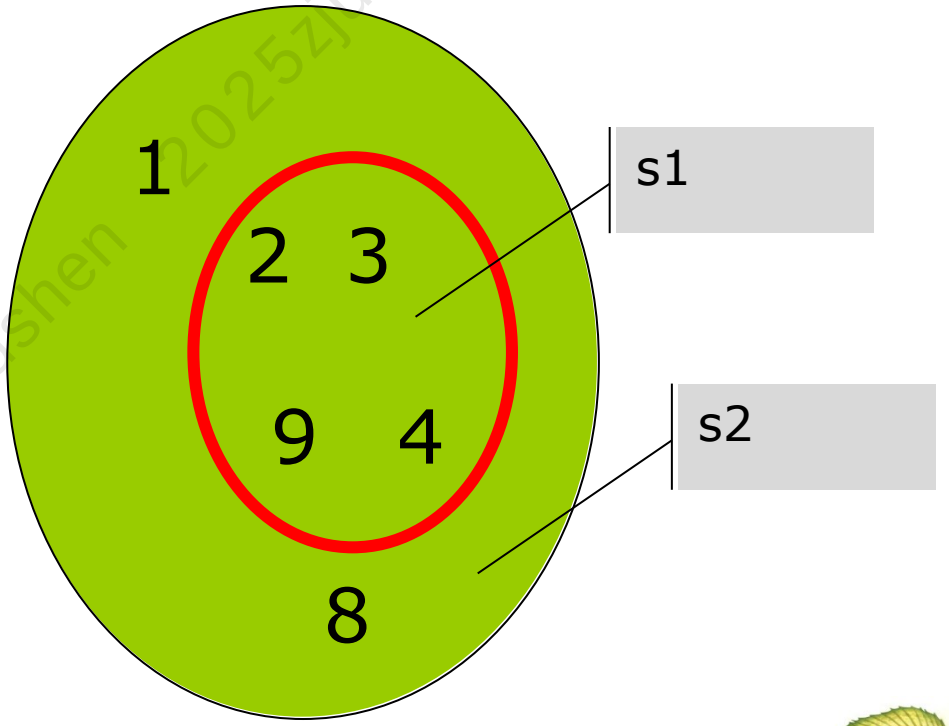
子集和超集

- 概念

  如果集合s1中的元素，都在集合s2中，则称s1为s2的子集，s2则为s1的超集。

- 使用s1.issubset(s2)来判断s1是否为s2的子集。
- 使用s2.issuperset(s1)来判断s1是否为s2的超集。

```
>>> s1 = {2, 3, 9, 4}
>>> s2 = {1, 2, 4, 3, 9, 8}
>>> s1.issubset(s2)
True
>>> s2.issuperset(s1)
True
```



关系运算判断

■ 使用关系运算符==和!=

判断2个集合是否包含完全相同的元素。

```
>>> s1 = {2, 3, 9, 4}
>>> s2 = {3, 4, 2, 9}
>>> s1 == s2
True
```

■ 使用关系运算符<,<=,>,>=。

- 如果s1是s2的真子集，则s1<s2是True
- 如果s1是s2的子集，则s1<=s2是True
- 如果s1是s2的真超集，则s1>s2是True
- *如果s1是s2的超集，则s1>=s2是True*

注：s1是s2的真子集的意思是s1是s2的子集，但是s2中至少有一个s1中不存在的元素； s1是s2的真超集的意思是s1是s2的超集，但是s1中至少有一个s2中不存在的元素。

集合运算

- 通过集合的函数或运算符进行集合的并集、交集、差集和对称差的集合运算。

假设2个集合：s1 = {3,5,7,11}, s2 = {3,4,5,6,7}

运算	方法	运算符	示例	结果	说明
并集	union()	|	s1 | s2	{2,3,4,5,6,7,11}	结果是包含两个集合中所有元素的新集合
交集	intersection()	&	s1 & s2	{2,3,5,7}	交集是只包含两个集合中都有的元素的新集合
差集	difference()	-	s1 - s2	{11}	s1-s2的结果是出现在s1但不出现在s2的元素的新集合
对称差	symmertric_difference()	^	s1 ^ s2	{4,6,11}	结果是一个除了共同元素之外的所有元素

- 某班级学生参加乒乓球（A）和羽毛球（B）两项活动，以下是参加这两项活动的学生名单，一个人可能参加两个活动，现在需要统计共有多少人参加活动、同时参加两项活动的人数、只参加乒乓球或者羽毛球的人数。

A = {'刘晨', '吴子轩', '周雨桐', '张一诺', '徐佳琪', '朱浩然', '李俊杰', '王天佑'}

B = {'徐明辉','张一诺', '徐佳琪', '李俊杰', '王天佑', '周晨', '孙雪', '张俊', ', '朱佳琪'}

```
print(f"参加羽毛球和乒乓球的一共有多少人:{len(A|B)}人")
print(f"同时参加羽毛球和乒乓球的有多少人:{len(A&B)}人")
print(f"参加羽毛球但不参加乒乓球的有多少人:{len(A-B)}人")
print(f"参加乒乓球但不参加羽毛球的有多少人:{len(B-A)}人")
```

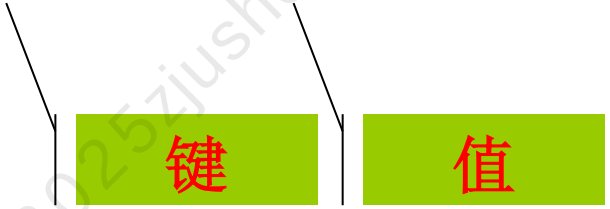

字典

概念

字典是一个用“键”做索引来存储的数据的集合。一个键和它所对应的数据形成字典中的一个条目。这种通过名称来访问其各个值得数据结构称为“映射”

用花括号{ }来表示，每个元素用冒号分隔 键 和 数据（值）。

students = {3180101:’张三’, 3180102:’李四’, 3180105:’王五’, 3180110:’赵六’}



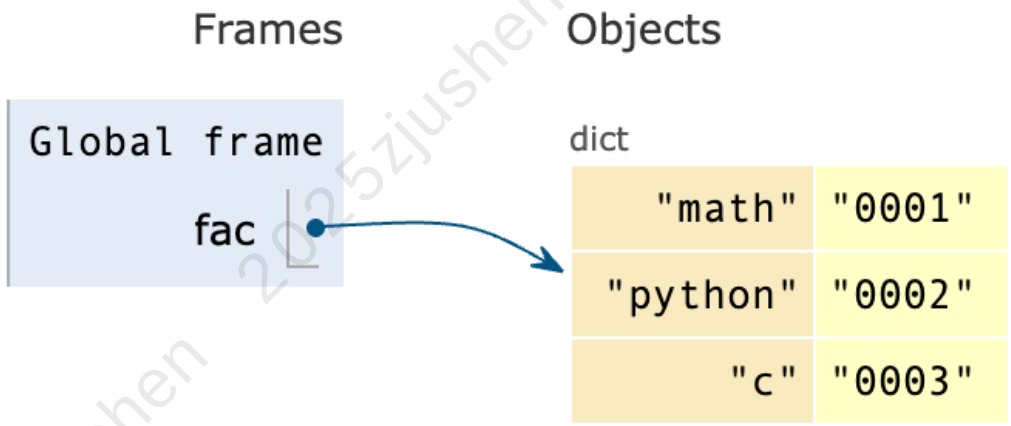
3180101	‘张三’
3180102	‘李四’
3180105	‘王五’
3180110	‘赵六’

- 不可变对象可作为字典的键，如： 数字，字符串，元组（元组内的数据也应是不可变对象）
  可变对象不可以作为字典的键，如：列表，字典等

创建字典

方法	示例	适用场景
{}	d = {}	创建空字典
显式键值对	s = {'name': 'Alice', 'age': 30}	需要复杂键或非字符串键时
dict.fromkeys()	d = dict.fromkeys(['a', 'b'], 0)	批量初始化相同值的键
关键字参数	s = dict(name='Alice', age=30)	键是简单字符串且符合标识符规则
zip() 组合	dict(zip(keys, values))	从两个列表构建键值对

如何实现该字典的构建？



访问字典的键

<字典>.keys(), 返回字典所有键的对象(类似于集合)

a set-like object providing a view on D's keys

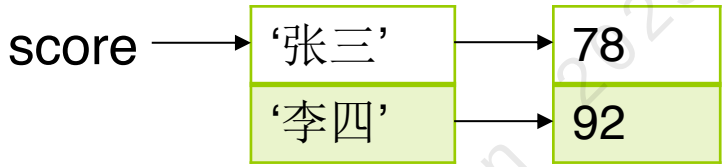
```
>>> score = {'张三':78, '李四':92}
>>>list( score.keys())
['张三, '李四']
```


访问字典的值

1.直接用[ ]运算符，用<字典> [键]的形式

例：score = {'张三':78, '李四':92}

print(score['张三'])



如果提供的键不存在，会抛出keyerror的异常。

2.用get()，用<字典>.get(键)的形式

score.get(‘张三’)

如果提供的键不存在，返回None。

3. <字典>.values()返回所有值的对象（类似于列表）

an object providing a view on D's values

注：字典不是有序容器，所以不能通过位置来访问，值只能通过与之关联的键访问，3.7版本之后可以保持插入顺序。

```
>>> score = {'张三':78, '李四':92}
>>> score['王五']
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    score['王五']
KeyError: '王五'
```

访问字典的键和项

<字典>.items(), 返回字典键值对组成的类似于集合的对象

a set-like object providing a view on D's items

```
>>> score = {'张三':78, '李四':92}
>>> score.items()
dict_items([('张三', 78), ('李四', 92)])
```


修改和增加项

字典属于可变容器，所以可以修改或增加项

1. score['李四'] = 89

#把score中键为'李四'的数据修改为89

2. score['王五'] = 100

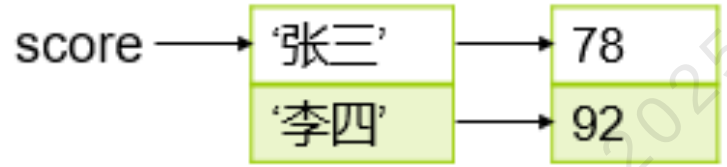
#score中没有键为‘王五’的元素，则增加一项，键为’王五’,数据为100。

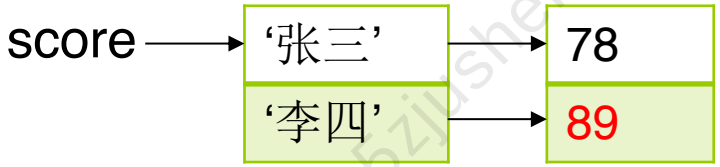
print(score）

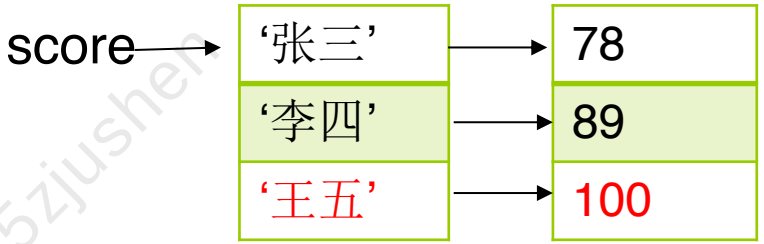
输出：{'张三': 78, '李四': 89, '王五': 100}

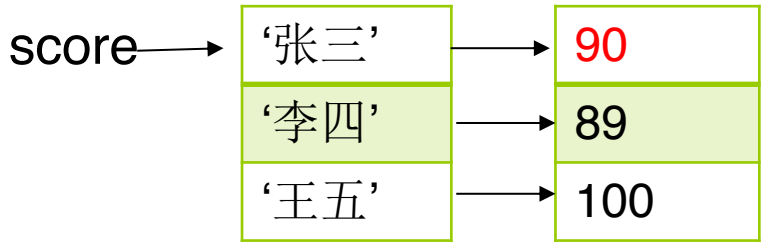
3.update()，用于更新字典中的键/值对，可以修改存在的键对应的值，也可以添加新的键/值对到字典中。

score.update({‘张三’:90})









删除项

1. 用del语句

   del score['张三']

2. 用pop方法，该方法会返回正在删除的项的值

   score.pop(‘张三’)

注：如果指定键不存在，则会抛出KeyError异常。

解决方案：

可以用in和not in运算符检测一个键是否在字典中存在。

增加默认值 score.pop(‘张三’,”不存在”)

3. clear()删除所有条目

常用操作

- 字典大小

  用函数len()得到字典的条目的数量。

  len(score)

- 用==和!=比较2个字典是否相同（键和值都相同，而项的顺序没有关系）

```
score = {'张三':78, '李四':92, '王五':89}
mark = {'张三':78, '李四':92, '王五':91}
print(score == mark)
```

输出：

False

遍历字典

```
score = {'张三':78, '李四':92, '王五':89}
for name in score:
    print(f'{name}:{score[name]}')
```

输出

```
张三:78
李四:92
王五:89
```

```
for item in score.items():
    print(f'{item[0]}:{item[1]}')
```

```
for key, value in score.items():
    print(f'{key}:{value}')
```

字典的方法

函数	返回值和说明
keys()	返回由全部的键组成的一个序列
values()	返回由全部的值组成的一个序列
items()	返回一个序列，其中的每一项是一个元组，每个元组由键和它对应的值组成
clear()	删除所有条目
get(key)	返回这个键所对应的值
pop(key)	返回这个键所对应的值，同时删除这个条目

性能比较

特性	列表(list)	集合(set)	字典(dict)
有序性	✅ 保持插入顺序	❌ 无序	✅ 3.7+保持插入顺序
元素唯一性	❌ 允许重复元素	✅ 自动去重	✅ 键唯一
元素类型	任意对象	可哈希对象	键：可哈希对象 值：任意对象
访问方式	索引/切片	只能遍历	键访问
时间复杂度	查找O(n) 插入O(1)	查找O(1) 交并差O(n)	查找O(1) 插入O(1)

复制操作

- 在Python中，复制操作的行为因对象类型（可变/不可变）和复制方式（直接赋值、浅拷贝、深拷贝）的不同而有显著差异。

类型	可变性	直接赋值	切片/浅拷贝	深拷贝
列表	可变	共享引用	外层独立，嵌套共享	完全独立
字典	可变	共享引用	外层独立，嵌套共享	完全独立
字符串	不可变	共享引用（无副作用）	完整切片为原对象	无意义（始终独立）
元组	不可变	共享引用（无副作用）	完整切片为原对象	无意义（始终独立）

```
a = [1, 2, 3]
b = a    # b和a指向同一列表
b.append(4) # 修改b会影响a
print(a)        # [1, 2, 3, 4]
```

```
a = [[1, 2], [3, 4]]
b = a[:]       # 浅拷贝
b[0].append(5) # 修改嵌套列表会影响a
print(a)       # [[1, 2, 5], [3, 4]]
```

```
import copy
a = [[1, 2], [3, 4]]
b = copy.deepcopy(a)
b[0].append(5)   # 不影响a
print(a)             # [[1, 2], [3, 4]]
```

综合实践

《电商用户行为分析系统开发》

任务背景：

某电商平台需要分析用户行为日志，原始数据采用字典结构存储：

```
user_behavior = {
    '用户ID': ['行为1', '行为2', ...],
    ...
}
```

目标：掌握以下知识点

- 字典的键值对存储与遍历
- 集合的自动去重特性
- 列表推导式与条件过滤
- 字典的get()方法计数模式
- 数据结构间的协同工作场景

请编写Python代码实现以下三个核心功能：

功能要求：

1. **行为类型提取**
   - 输出所有出现过的**唯一**行为类型
   - 要求：使用集合实现去重
2. **高价值用户筛选**
   - 找出所有完成"purchase"行为的用户
   - 要求：使用列表推导式实现
3. **行为热度统计**
   - 统计每种行为出现的总次数
   - 要求：使用基础字典实现计数逻辑