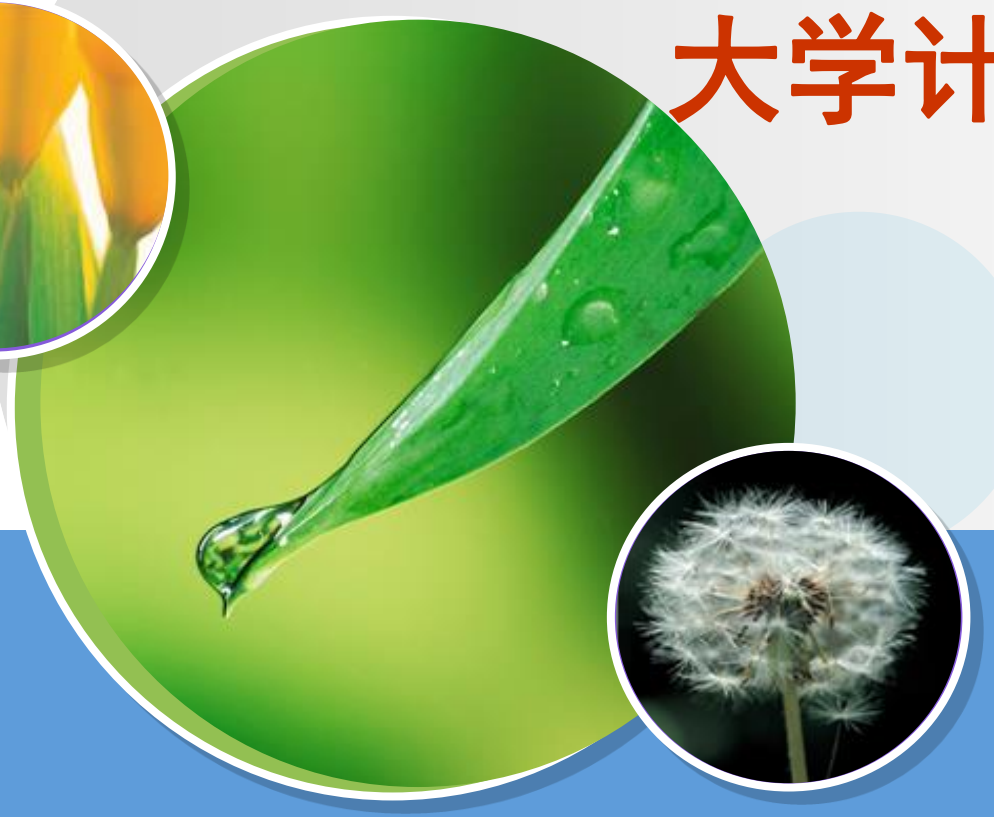
上课课件

北京航空航天大學
BEIHANG UNIVERSITY

沙河高校联盟共享课程

# 大学计算机基础（Python编程）





北京航空航天大学

# 第3章 程序设计基础与数据结构（三）

共6学时

# 本讲重点和难点

**重点**

- **字典的创建**（两种方法）**和基本操作**（访问、遍历、添加键值对、删除、成员检测）
- **字典方法**：clear、get、pop、keys、values
- **Python实现栈的方法**
- **Python实现队列的方法**

**难点**

- 字典的熟练使用
- 利用Python实现栈或队列

# 3.5 Python结构化数据类型

- 3.5.1 概述
- 3.5.2 列表
- 3.5.3 字符串
- 3.5.4 字典

# 预习任务

1、自学中国大学MOOC上北航《大学计算机基础》**MOOC “第5讲 Python的基本语法”、“第6讲数据与数据结构”、“第7讲 Python实现自定义数据结构”**视频和课件，及时完成**单元测验**和**课堂讨论**。

2、预习**教材**《面向计算思维的大学计算机基础》**“第3章 程序设计基础与数据结构”**中**“3.2 Python的基本语法“、 3.4 数据与数据结构“、” 3.5 Python实现自定义数据结构”** 。

3、预习**课件** “《大学计算机基础》-**第3章 程序设计基础与数据结构（三）【预习用】”** 。

# 3.5.3 字符串

- **字符串**：用双引号" "或单引号' '包裹起来的一系列字符
  - 例： " world " ，或' world’
  - '123'代表字符串，而并非数字123
- **字符串是字符序列，可通过位置索引访问每个字符**

**<字符串>[<索引>]**

- **字符串的访问**
  - **使用索引从字符串中提取单个字符**

    'xyz'[2]获取字符'z'
  - **使用切片提取连续的子串**

    'xyz'[1:3] 获取子串 'yz‘
  - **可以间隔一定的步长，提取子串**

- **字符串操作**
  - 序列的**通用操作**：**索引、分片、连接、乘法、成员检测**
  - **字符串方法**：find、join、lower、upper、replace、split、strip、translate……

```
>>> s='123456789'
>>> s[1:8:2]
'2468'
```

# 字符串的遍历

- **字符串的遍历**　要掌握！
  - **方法**：使用**for**语句遍历。**简洁！建议采用**

```
#string_traversal_1.py
word='hardwork'
print('word中的所有字母是：')
for i in word:
    print(i)
```

循环变量为字符串中每个字符

```
>>>
word中的所有字母是：
h
a
r
d
w
o
r
k
```

```
#string_traversal_2.py
word='hardwork'
print('word中的所有字母是：')
for i in range(len(word)):
    print(word[i])
```

循环变量为索引

# 字符串的应用示例：【例3. 11】

**【例3.11】**计算任意一个单词所对应的总分数。

- 定义如下的规则
  - 英文字母a/A对应1分、b/B对应2分、c/C对应3分……
  - 每个英文单词的得分即为各个字母分数的和
  - 试计算任意一个单词所对应的总分数
  - 将结果显示在屏幕上

# 补充知识： ASCII码

- **西文字符的编码**
  - 在计算机内部，西文字符的编码采用**ASCII码**（American Standard Code for Information Interchange，美国信息交换标准交换代码）
  - 国际通用的是**7位**ASCII码（**基础ASCII码**），用7位二进制数表示一个字符的编码
  - 基础ASCII码可以表示**128**个不同字符
    - ✓ 普通字符**94**个： 26个大写英文字母、26个小写英文字母、10个阿拉伯数字、通用运算符号（+、-、×、÷等）及标点符号等共32个
    - ✓ 控制字符或通信专用字符**34**个
    - ✓ 基础ASCII码每个字符都对应一个数值，称为该字符的**ASCII码值**。其排列次序为$b_6b_5b_4b_3b_2b_1b_0$，编码为000 0000~111 1111（十进制数**0~127**）
- ASCII码的编排有一定规律，字母A～Z、a～z都是**按顺序**编排的。而且小写字母比大写字母的码值**大32**

**计算机内部用一个字节存放一个7位ACII码，最高位置0**

# 基础ASCII码编码表

| $b_6b_5b_4$ $b_3b_2b_1b_0$ | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| 0000 | NUL | DLE | SP | 0 | @ | P | ` | p |
| 0001 | SOH | DC1 | ! | 1 | A | Q | a | q |
| 0010 | STX | DC2 | ” | 2 | B | R | b | r |
| 0011 | ETX | DC3 | # | 3 | C | S | c | s |
| 0100 | EOT | DC4 | $ | 4 | D | T | d | t |
| 0101 | ENQ | NAK | % | 5 | E | U | e | u |
| 0110 | ACK | SYN | & | 6 | F | V | f | v |
| 0111 | BEL | ETB | ’ | 7 | G | W | g | w |
| 1000 | BS | CAN | ( | 8 | H | X | h | x |
| 1001 | HT | EM | ) | 9 | I | Y | i | y |
| 1010 | LF | SUB | * | : | J | Z | j | z |
| 1011 | VT | ESC | + | ; | K | [ | k | { |
| 1100 | FF | FS | , | < | L | \ | l | \| |
| 1101 | CR | GS | - | = | M | ] | m | } |
| 1110 | SO | RS | . | > | N | ^ | n | ~ |
| 1111 | SI | US | / | ? | O | - | o | DEL |

“A”的ASCII码值是65

“B”的ASCII码值是66

“a”的ASCII码值是97

“b”的ASCII码值是98

# 【例3. 11】分析

- 因为同一个大小写字母分值相同，为简化处理，先使用字符串的**lower方法**将输入的单词word转换为小写字符串

  **new_word=word.lower()**

- **如何迅速获得某个字母对应的分数?**
  - 内置函数**ord()**：获取某个字符的ASCII码
  - **基础ASCII码**中字母A～Z、a～z都是**按顺序**编排的，“a”的ASCII码值是97，“b”的ASCII码值是98
  - 字母e的分数= **ord('e') - ord('a') + 1** = 5
- **如何计算某个单词所对应的总分数?**
  - **for语句**遍历单词的每个字母，计算每个字母的分数
  - 将所有字母分数累加

# 【例3.11】程序

```
#（1）输入单词
word=input('请输入你要计算分数的单词：')
new_word=word.lower()

#（2）计算任意长度的字符串对应的总分数

score = 0
for every_item in new_word:
    ascii_value = ord(every_item)
    if ascii_value >= ord('a') and ascii_value <= ord('z'):
        score += ascii_value - ord('a') + 1
    else:
        print('请输入字符！')

#（3）输出结果
print(word,'的分数是：', score)
```

使用字符串的lower方法将其转换为小写字符串

单词总分数，初始值设置为0

使用for语句遍历单词中每个字母

利用ord()函数获取其ASCII码值

计算该字母的分值，并累加

**例3.11-word_score.py**

```
>>>
========== RESTART: E:/
请输入你要计算分数的单词：Hardwork
Hardwork 的分数是： 98
>>>
========== RESTART: E:/
请输入你要计算分数的单词：word
word 的分数是： 60
>>>
```

# 字符串方法

要掌握！请认真学习

## 字符串方法

- 同其他内建数据类型一样，字符串也有许多的方法，常用的如**count、find、join、lower、upper、replace、split、strip、translate**等
- lower、upper、replace、strip、translate方法都返回一个字符串的**副本**，即**原字符串不会被改变**

# 常用字符串方法

| 方法名称 | 含义 | 示例 | 说明 |
| --- | --- | --- | --- |
| count | 统计某个字符在字符串中出现的次数<br><字符串>.count(<某字符>) | >>>'abcabcabcdef'.count('a')<br>3 | |
| find | 在一个较长的字符串中查找子字符串。它返回子串所在位置的**最左端索引**；若没有找到则返回**-1**。<br>**字符串.find(子字符串[,begin, end])** | >>>'AZABCDDeAB'.find('AB')<br>2 | 可选参数**begin，end**指定查找的开始和结束范围；默认查找整个字符串 |
| join | 用某个**连接符**将**字符串列表**中各元素连接起来，产生一个**新的字符串。**<br><'连接符'>.join(<字符串列表>)<br>将列表转换为字符串 | >>>seq=['1', '2', '3', '4', '5']<br>>>> '-'.join(seq)<br>'1-2-3-4-5'<br>>>> ''.join(seq)  #**连接符为空**<br>'12345' | 它是split方法的逆方法。如果连接符**为空**，则是直接把**各元素紧挨着**写在一起 |

# 常用字符串方法（续1）

| 方法名称 | 含义 | 示例 | 说明 |
| --- | --- | --- | --- |
| lower | 返回一个字符串的**副本**，全部为小写字母。<br>**<字符串>.lower()** | >>> name='Peter'<br>>>> new_name=name.lower()<br>>>> new_name<br>'peter' | **原字符串不会被改变** |
| upper | 返回一个字符串的**副本**，全部为大写字母。<br>**<字符串>.upper()** | >>> name='Peter'<br>>>> new_name=name.upper()<br>>>> new_name<br>‘PETER' | **原字符串不会被改变** |
| replace | 将某字符串中所有与某指定子字符串**old**相匹配的项用希望被替换的子字符串new替换掉，得到一个新的字符串，并返回该字符串<br>**<字符串>.replace(old, new[, max])**<br>其中old指示的匹配项被替换为new，可选择最多替换max个 | >>> words='Peter loves China. Peter is a teacher. '<br>>>>new_words=**words.replace('Peter', 'Bob')**<br>>>> new_words<br>'Bob loves China. Bob is a teacher. ' | **原字符串不会被改变。**可实现文字处理中的“**查找并替换**”功能。 |

# 常用字符串方法（续2）

| 方法名称 | 含义 | 示例 | 说明 |
| --- | --- | --- | --- |
| split<br>将字符串转换为列表 | 利用某个分隔符（如“+”)将一个**字符串**分割为序列，返回字符串中所有单词的**列表。**<br>**<字符串>.split([sep[, maxsplit]])**<br>其中sep作为分隔符（如果不特别指定则以**空格**切分单词）；可使用maxsplit指定最大切分数。 | >>> words='Peter loves China.'<br>**>>> words_list=words.split()**<br>>>> words_list<br>['Peter', 'loves', 'China.']<br>**>>>lis=input().split()** | 它是join方法的逆方法。如果不提供任何分隔符，则Python自动把所有空格作为分隔符（如空格、制表符、换行符）。<br>用途：**切分单词；将一行输入的多个数据存入列表** |
| strip | 返回一个字符串的**副本**，它将整个字符串两侧（**不包括内部**）的指定字符去除。如果不指定要去除的字符，则去除所有**空格**字符（空格、制表符和换行符）。<br>**<字符串>.strip([chars])** | >>> words='***SPARM * for *everyone!!!***'<br>>>> words.strip(‘*!’) #去除words中字符串的两侧所有的“*”和“！”<br>'SPARM * for *everyone' | **原字符串不会被改变。**<br>**用途**：**去除**某个长字符串的**两侧**所有**多余的字符。** |

# 字符串与列表的相互转换

- **字符串转列表** -- 字符串方法：**split()**

```
>>> sen = 'Good morning everyone'
>>> sen.split()
['Good', 'morning', 'everyone']
>>> sen = 'A-B-C-D'
>>> sen.split('-')
['A', 'B', 'C', 'D']
```

- **列表转字符串** -- 字符串方法：**join()**

```
>>> seq = ['A','B','C','D']
>>> ''.join(seq)
'ABCD'
>>> '-'.join(seq)
'A-B-C-D'
```

- 字符串的join()方法与split()方法互为**逆操作**
- join方法可以用来为列表中的每个元素添加一个**新**元素（即连接符），并将其连接成一个字符串

# 使用list函数将字符串转换为列表

- **list函数：将一个序列或可迭代对象转换为列表**

**格式：** list(<序列>)

```
>>>string1= list('teachers')
>>>string1
['t', 'e', 'a', 'c', 'h', 'e', 'r', 's']
```

元组

```
>>>string2= list((10,20,30))
>>>string2
[10, 20, 30]
```

- **list函数适用于所有类型的序列（如元组），而不只是字符串！**

# 字符串方法的应用示例：【例3. 12】

【例3.12】切分单词，提取出一篇文章中的所有单词，放入一个列表中。注意去除标点符号。

## 设计思路

- 文章采用**三引号**括起来，赋给一个变量article
- 采用**for循环**，遍历一个列表（存放要去除的标点），使用字符串的**replace**方法，将article中所有的**标点**字符（本例只有**英文逗号**和**句号**）替换为**空格**

```
for i in [', ', '. ']:
    new_article=article.replace(i, ' ')
```

- 再对去掉标点后的字符串使用**split**方法，以**空格**分隔，将单词存入列表

```
words_list=new_article.split()
```

# 【例3.12】程序

**例3.12-string_split.py**

```
#（1）将要切分单词的字符串赋值给变量article
article='''All of us have read thrilling stories in which the hero had only
a limited and specified time to live. Sometimes it was as long as a year,
sometimes as short as 24 hours. But always we were interested in discovering
just how the doomed hero chose to spend his last days or his last hours. '''

#（2）使用replace方法，将article中所有的标点字符替换为空格
for i in [',', '.']:                          #遍历要去除的标点
    new_article=article.replace(i, ' ')       #将article中的','或'.'替换为空格
    article=new_article #将去掉一种标点的新字符串又赋给article，以便再去除其他标点
```

必须赋给新的变量，才能继续替换其他标点。因为原字符串没有被改变

# 【例3.12】程序（续）

```
#（3）对去掉标点后的字符串使用split方法，将字符串转换为列表
words_list=new_article.split()
print (‘new_article:\n’,new_article)        #其中“\n”表示在此处换行
print ('_____')
print ('words_list:\n',words_list)
```

# 【例3.12】程序运行结果

去除标点后

```
>>>
new_article:
 All of us have read thrilling stories in which the hero had only
a limited and specified time to live  Sometimes it was as long as a year
sometimes as short as 24 hours  But always we were interested in discovering
just how the doomed hero chose to spend his last days or his last hours

_____________________________________________________________________
words_list:
 ['All', 'of', 'us', 'have', 'read', 'thrilling', 'stories', 'in', 'which', 'the', 'hero'
, 'had', 'only', 'a', 'limited', 'and', 'specified', 'time', 'to', 'live', 'Sometimes', '
it', 'was', 'as', 'long', 'as', 'a', 'year', 'sometimes', 'as', 'short', 'as', '24', 'hou
rs', 'But', 'always', 'we', 'were', 'interested', 'in', 'discovering', 'just', 'how', 'th
e', 'doomed', 'hero', 'chose', 'to', 'spend', 'his', 'last', 'days', 'or', 'his', 'last',
'hours']
>>>
```

- ‘live’、’hours’后面的句号均被去除了！

# 3.5.4 字典

- **字典**是非常有用的**结构化**数据类型，Python中**唯一内建的映射类型**
  - 用于存储**值**与**名字**相关联的某一类特殊数据
    - ✓ 例如**通讯簿**，一个人的工作单位、电话号码、地址等信息都与其**姓名**相关联

**例：people={‘张三’：‘82337600’，‘李四’：‘82337601’，‘王五’：‘82337602’}，人名是键，电话号码是值**

  - 字典中的值没有特定的顺序，打印出来是**随机**的
  - **字典由一对“{}”括起来的若干“键/值”对构成**

# 键值对

- **键值对**由键与值组成，之间以“:”分隔

- （1）键必须是**唯一**的，同一个字典中**各个键**，必须**各不相同**
- （2）键必须是任意**不可变**类型，如**数字**、**字符串**或**元组**，**不能是列表**
- （3）值可以是简单数据类型、序列类型，也可以是字典或集合
  - 例：一个人名，可以有电话号码、工作单位、地址等值

```
people_new={‘张三’:{‘电话’:‘82337600’, ‘地址’:‘新主楼G105’},
            ‘李四’:{‘电话’:‘82337601’, ‘地址’:‘新主楼G106’},
            ‘王五':{'电话':'82337602', '地址':'新主楼G107'}}
```

# 1、创建字典

## 创建字典

- **字典**由多个键和值构成的对组成，每个键值对称为**项**，键与值之间用冒号（：）隔开，项与项之间用逗号“，”隔开，整个字典由一对大括号括起来。空字典如{}所示

```
>>> {01: 'apple', 02: 'pear', 03: 'banana'}
SyntaxError: invalid token
```

```
>>> {1: 'apple', 2: 'pear', 3: 'banana'}
{1: 'apple', 2: 'pear', 3: 'banana'}
```

方法一　直接在{}中写出各项

```
>>> phonebook={'Alice':'02341', 'Peter':'09102', 'Bob':'03356'}
>>> phonebook                          #访问整个字典
{'Bob': '03356', 'Peter': '09102', 'Alice': '02341'}
>>> phonebook['Bob']                   #查找某个键对应的值
‘03356'
```

# dict函数

方法二

用dict函数，通过**其他字典**或者**（键，值）**这样的序列对（即**元组**）来创建字典

- （1）先创建一个列表，其中每个元素是一个表示**（键，值）**的元组，其中键可以是数字、字符串或元组：

  **列表名=[(键1, 值1)，(键2, 值2)，…]**

- （2）再用dict函数创建字典，将列表中的每个元组变成一个键值对：

  **字典名=dict(列表名)**

# dict函数（续）

```
>>> items=[('name','Alice'),('age',20),('sex', 'female')]   #创建列表，每个元素是一个表示（键，值）的元组
>>> d=dict(items)                                           #用dict函数创建字典，将列表items中的每个元组变成一个键值对
>>> d                                                       #访问整个字典
{'age': 20, 'name': 'Alice', 'sex': 'female'}
>>> d['name']                                               #查找某个键对应的值
'Alice'
```

当访问整个字典时，打印顺序是**随机**的

# 2、字典的基本操作

- **字典的基本操作与序列类似**，其中d为字典，k为键
  - ✓ **len(d)**：返回d中项的个数
  - ✓ **d[k]**：返回关联到键k上的值，**如果k是一个字符串，则必须用单引号括起来！**
  - ✓ **d[k]=v**：将值v关联到键k上，相当于**替换**某个已有的键对应的值，或者在字典中**添加**一个键/值对
  - ✓ **del d[k]**：**删除**键为k的项
  - ✓ **k in d**：检查d中是否含有键为k的项，并返回值“True”或“False”

必须先创建字典d

- **访问某个键对应的值时，采用<字典>[<键>]的形式**

# 循环遍历字典元素

- 采用for语句遍历字典的所有键，访问对应的值

```
#for_dic.py

phonebook={'Alice':'2341', 'Peter':'9102', 'Bob':'3356'} #字典
for key in phonebook:                          #遍历整个字典，key表示键
    print (key, 'corresponds to', phonebook[key])      #打印每个键对应的值
```

```
>>>
Bob corresponds to 3356
Peter corresponds to 9102
Alice corresponds to 2341
```

使用print函数打印时，键和值都不显示引号

# 【课堂练习】：DoReMi

| 字母名 | C | D | E | F | G | A | B |
|---|---|---|---|---|---|---|---|
| 唱名 | do | re | mi | fa | sol | la | si |

- 请编写一个程序，创建字典（以字母名和唱名作为键值对）；根据键盘输入的字母名，输出对应的唱名。
  - 输入：一行，为一个大写字母
  - 输出：一行，为字母对应的唱名
- **字典写出前3个键值对即可**

# 【课堂练习】程序

**work1-DoReMi.py**

```
dic={'C':'do','D':'re','E':'me','F':'fa','G':'sol','A':'la','B':'si'}

char=input()          #输入字母

print(dic[char])      #通过键析取值，打印唱名
```

```
>>>
C
do
>>> ====
>>>
D
re
```

## 3、字典方法

- clear、copy、deepcopy、get、keys、pop、popitem、update、values等

### ① clear方法

- clear方法清除字典中的所有的项，原地操作，返回None
- clear方法的调用：<待清除的字典名>.clear()

```
>>> d={'name':'Alice','age':23}
>>> d.clear()
>>> d
{}
```

思考：字典的clear方法与del语句相同吗?

# ② 字典的get方法

- 字典的**get()方法**可以根据键返回值，如果字典中不存在输入的键，则返回None　　**调用**：**<字典名>.get(<键>[, 默认值])**
  - 当get()方法有两个参数时，第一个为**键**，第二个为设置的默认返回值。如果字典中不存在输入的键，则返回该默认值（如下例中的'Not found'）。

```
>>> tel = {'gree': 4127, 'jack': 4098, 'shy': 4139}
>>> tel.get('gree')
4127
>>> tel.get('jack', 'Not found')
4098
>>> tel.get('rose', 'Not found')
'Not found'
```

rose不存在

# ③ 字典的pop方法

## ③ pop方法

- **pop方法获得对应于给定键的值，然后将这个键值对从字典中移除**
- pop方法的调用：**<被访问的字典名>.pop(<要移除键值对的键>)**

```
>>> d={'name':'Alice','age':23}
>>> d.pop('name')
'Alice'
>>> d
{'age': 23}
```

**思考1：字典的pop方法与get方法有何区别?**

**思考2：列表的pop方法是如何调用的？括号里是什么?**

**<被访问的列表名>.pop(<要移除元素的索引>)**

# ④ keys方法

- keys()方法返回字典中所有的键，顺序不定
- keys方法的调用：<字典名>.keys()

```
>>> tel = {'gree': 4127, 'shy': 4139 ,'jack': 4098}
>>> tel.keys()
dict_keys(['jack', 'shy', 'gree'])
```

- 如果需要它有序，则要将其转换成列表，再使用sort()方法

```
>>> name=tel.keys()
>>> lis=list(name)
>>> lis
['gree', 'shy', 'jack']
```

```
>>> lis.sort()
>>> lis
['gree', 'jack', 'shy']
```

# ⑤ values方法

- values方法返回字典中所有的值
- values方法的调用：<字典名>.values()
- 返回值的列表中可以包含重复的元素

```
>>> dic = {'gree': 95, 'jack': 88, 'shy': 73,
'alice':95}
>>> score=dic.values()                    #获取所有的值
>>> score
dict_values([95, 95, 73, 88])

>>> lis_score=list(score)                 #转换为列表
>>> lis_score
[95, 95, 73, 88]
```

# 【例3.13】字典应用：统计单词出现次数

【例3.13】通过键盘输入一串字符（假定这串字符不包括标点符号，只包含英文字符，各单词之间以一个空格分隔），试统计其中各单词出现的字数，并以字典的形式输出。

## 设计思路

- 采用字符串的**split()方法**提取空格分隔的单词，存入一个列表lis
- 创建一个空字典word_dic
- 采用**for循环**，遍历列表lis，提取单词次数，记录在字典中
  - ✓ 如果某单词在字典中，则字典word_dic中以该单词为键的值加1；

```
word_dic[word] += 1
```

  - ✓ 如果某单词不在字典中，则word_dic创建新的键值对，以该单词为键，值为1

```
word_dic[word]=1
```

# 【例3.13】程序

**例3.13-words_count.py**

```
#（1）提取空格分隔的单词，存入一个列表lis
lis=input().split()

#（2）创建一个空字典
word_dic={}

#（3）采用for循环，遍历列表lis，提取单词次数，记录在字典中
for word in lis:
    if word in word_dic:
        word_dic[word] += 1      #以该单词为键的值加1
    else:
        word_dic[word] = 1       #以该单词为键，初值为1

print (word_dic)
```

同word_dic[word] = word_dic[word]+1

创建新的键值对

# 【例3. 13】程序运行结果

```
>>>
to be or not to be
{'not': 1, 'be': 2, 'to': 2, 'or': 1}
```

```
>>>
I am a Chinese I love my country You are Chinese too You love our country too
{'I': 2, 'Chinese': 2, 'are': 1, 'am': 1, 'country': 2, 'love': 2, 'too': 2,
'You': 2, 'our': 1, 'a': 1, 'my': 1}
```

# 【例3.14】字典应用：统计水果信息

**【例3.14】**已知某超市购进了一批水果，采购人员将水果名称、单价信息存储在一个字典中，水果名称、重量信息存储在另一个字典里。试编写程序：能够根据输入的任何一个水果名称**查找**其**单价**并输出；提取**所有水果名称**存入一个列表，并输出；提取所有水果的**重量**并求和输出；计算**总金额**并输出。

- **输入**
  - ✓ 一行，为字符串，表示要查找的水果名称。
- **输出**
  - ✓ 四行，第1行为查找到的水果单价
  - ✓ 第2行为所有水果名称，列表形式
  - ✓ 第3行为水果的总重量，浮点数
  - ✓ 第4行为水果的总金额，浮点数

# 【例3.14】设计思路

- **设计思路**

**1、创建两个字典price、 weight**

分别以“水果名称：单价”、“水果名称：重量”为键值对

**2、查找单价**

采用字典的get()方法，在字典price中根据输入的水果名称（键）查找其单价（值）

**<字典名>.get(<键>)**

**3、提取所有水果名称**

采用字典的keys()方法，在字典price中提取所有水果名称（键），再用list函数转换为列表

**list(<字典名>.keys())**

# 【例3.14】设计思路（续）

**4、计算水果的总重量**

（1）采用字典的**values()方法**，提取字典**weight**中所有水果的**重量**（值），再用list函数转换为列表

**<字典名>.values()**

（2）采用**sum函数**对列表元素求和

**5、计算总金额**

（1）采用字典的**keys()方法**获得字典price中所有的**键**

（2）采用**for循环**，遍历字典price的键，根据键去查找price和weight字典的值，二者**相乘**，得到每种水果的金额；再**累加**，得到总金额

```
money=0
for i in price.keys():                    #遍历字典price的键（水果名称）
    money += price[i]*weight[i]
```

# 【例3.14】程序

**例3.14-fruits_count.py**

```
#1、创建字典
price={'pear':5, 'banana':4, 'peach':4.5, 'apple':4.3, 'strawberry':10, 'pineapple':2.6}
weight={'pear':100, 'banana':150, 'peach':150, 'apple':200, 'strawberry':80, 'pineapple':180}

#2、输入要查找的水果名称
lookup=input()

#3、提取信息并计算
#（1）根据输入的水果名称查找其单价
ans=price.get(lookup)          #get()方法根据键返回值

#（2）提取所有水果名称存入一个列表
name=list(price.keys())        #keys()方法以列表的形式返回字典中所有的键；再用list函数转换为真正的列表
```

# 【例3.14】程序（续）

```
#（3）提取所有水果的重量并求和输出
weights=list(weight.values())      #values方法以列表的形式返回字典中所有的值；再用list函数转换为真正的列表
#print('各水果重量',weights)        #【打印中间结果】
total=sum(weights)                 #sunm函数对列表元素求和

#（4）计算总金额
money=0
for i in price.keys():             #遍历字典price的键（水果名称）
    money += price[i]*weight[i]    #i的单价乘以重量，得到i的金额；累加

#4、输出
print(lookup,'单价是',ans)
print('所有水果名称：',name)
print('所有水果的重量是',total,'斤')
print('水果总金额是',money,'元')
```

**掌握**：利用字典中的值进行算术运算

# 【例3.14】程序运行结果

```
>>>
peach
peach 单价是 4.5
所有水果名称： ['pear', 'pineapple', 'strawberry', 'apple', 'banana', 'peach']
各水果重量 [100, 180, 80, 200, 150, 150]
所有水果的重量是 860 斤
水果总金额是 3903.0 元
```

# 字典类型小结

- 由大括号将元素括起，每个元素是“键:值”的形式
  - d[k]：返回d中键为k的值，若k不存在会提示出错；
  - d[k] = v：将值v与键k关联，若已有值则覆盖；
  - del d[k]：从d中将键为k的项移除；
  - d.clear()：清除d中所有的项，原地操作，返回{}；
  - d.get(k,v)：如果k在d中，返回d[k]，否则返回v；
  - d.pop(k)：从d中将键为k的项移除，并**返回**相应的**值**；
  - d.keys()：返回包含d中**所有键**的列表；
  - d.values()：返回d中**所有值**的列表；
  - k in d：若k在d中，则返回True；
  - len(d)：返回d中元素的数量；
  - for k in d：依据d中keys进行遍历，此过程中请勿删除字典元素

# 3.6 Python实现自定义数据结构

- 3.6.1 数据结构概述
- 3.6.2 线性结构—栈
- 3.6.3线性结构—队列

# 3.6.1 数据结构概述

- 计算机要处理的数据并不是杂乱无章的，它们往往存在内在的联系
- 通常把具有**相同属性**的一类数据元素，以某种方式（如对元素进行编号）组织在一起，形成特定的**数据结构**

**数据结构**

**按一定的逻辑结构组成的一批数据，使用某种存储结构将这批数据存储于计算机中，并在这些数据上定义了一个运算集合**

【例】市话用户信息表

| 序号 | 用户名 | 电话号码 | 用户住址 | |
|---|---|---|---|---|
| | | | 街道名 | 门牌号 |
| 00001 | 万方林 | 3800235 | 北京西路 | 1659 |
| 00002 | 吴金平 | 3800667 | 北京西路 | 2099 |
| 00003 | 王 冬 | 5700123 | 瑶湖大道 | 1987 |
| 00004 | 王三 | 5700567 | 瑶湖大道 | 2008 |
| 00005 | 江 凡 | 8800129 | 学府大道 | 5035 |

# 数据结构中的术语

## 数据结构中的术语

- **数据项**（Data Item）（**字段**）：**是数据的具有独立含义的不可分割的最小标识单位，数据项是对客观事物某一方面特性的数据描述**
  - ✓ 如【例1】表中的序号、用户名、电话号码等都是数据项
- **数据元素**（Data Element）：**是数据的基本单位，通常作为一个整体考虑和进行处理**（又称**结点**、**记录**）。一个数据元素由若干数据项组成
  - ✓ 如【例1】表中的包含序号、用户名、电话号码等具体数值的每一行数据都是一个数据元素
- **数据对象**（Data Object）：**是性质相同的数据元素的集合，是数据的一个子集**。如【例7.1】表中的所有数据元素

# 数据项和数据元素的例子

【例】某电信公司的市话用户信息表。

基本项　　组合项

| 序号 | 用户名 | 电话号码 | 用户住址 | |
|---|---|---|---|---|
| | | | 街道名 | 门牌号 |
| 00001 | 万方林 | 3800235 | 北京西路 | 1659 |
| 00002 | 吴金平 | 3800667 | 北京西路 | 2099 |
| 00003 | 王　冬 | 5700123 | 瑶湖大道 | 1987 |
| 00004 | 王三 | 5700567 | 瑶湖大道 | 2008 |
| 00005 | 江　凡 | 8800129 | 学府大道 | 5035 |

1个数据元素（记录）

5个数据元素

- **数据项分为基本项和组合项，每一个基本项或组合项称为一个字段**
  - ✓ **基本项**：指有独立意义的最小标识单位
  - ✓ **组合项**：由一个或多个基本项组成的有独立意义的标识单位

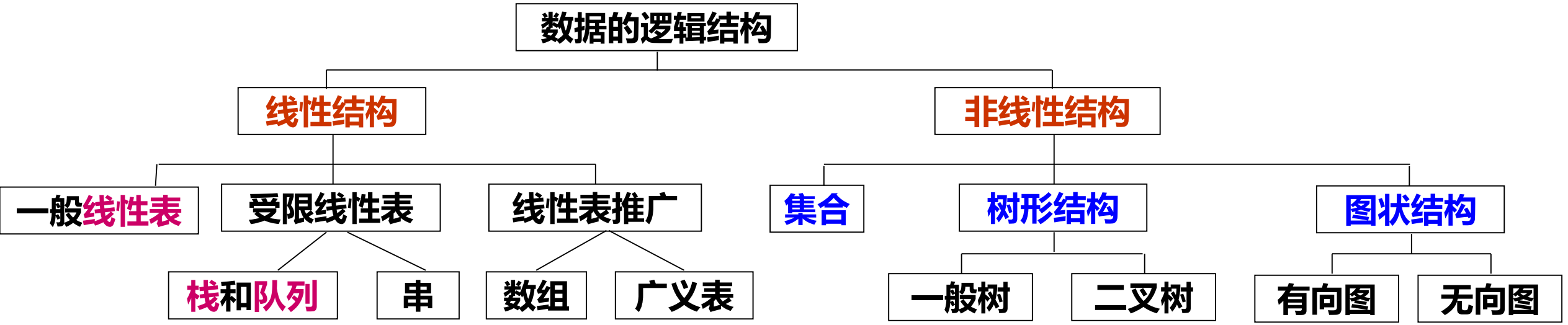
# 数据结构的三个层次

- 数据结构的三个组成部分
  - **逻辑结构**： 数据元素之间逻辑关系的描述
    - ✓ **线性结构，树形结构，网状结构，集合结构**
  - **存储结构**（物理结构）： 数据元素在计算机中的存储及其逻辑关系的表现
    - ✓ **顺序存储结构，非顺序存储结构**
  - **数据操作**： 对数据要进行的运算

**详见教材和MOOC**

# 数据逻辑结构的分类

- 数据的逻辑结构分为**线性**结构和**非线性**结构两大类

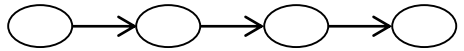



数据逻辑结构层次关系图

**受限线性表**：“**操作受限**”的线性表

# （1）线性结构

## （1）线性结构



**结构中的数据元素之间存在一对一的关系**

- **特点**

  1）存在唯一一个被称作“**第一个**”的元素；

  2）存在唯一一个被称作“**最后一个**”的元素；

  3）除第一个以外，集合中的每一个元素都只有一个**前驱**（**某个元素的前一个元素**）；

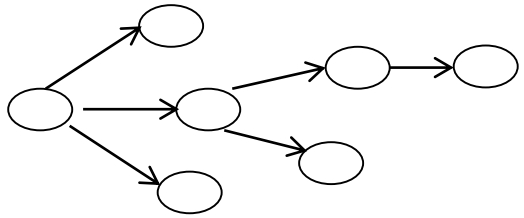
  4）除最后一个以外，集合中的每一个元素都只有一个**后继**（**某个元素的后一个元素**）。

- **例如：通迅录、成绩单、花名册**

- **【例】市话用户信息表**

# （2）树形结构

## （2）树形结构



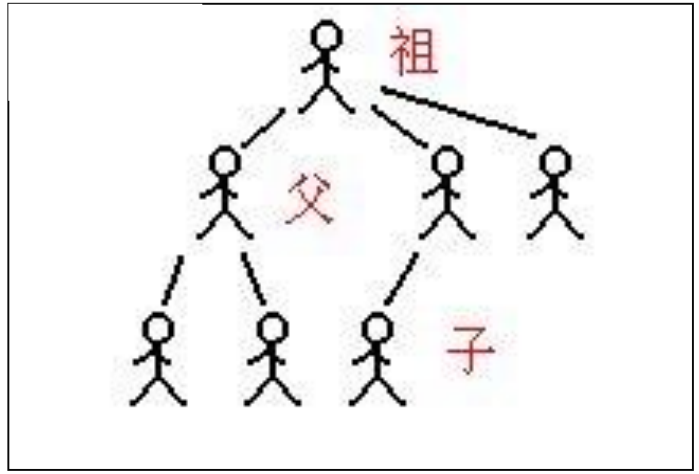
**结构中的数据元素之间存在一对多的关系**

- **特点**

  结点间具有**分层次**的连接关系：一个结点可能包含若干个子结点

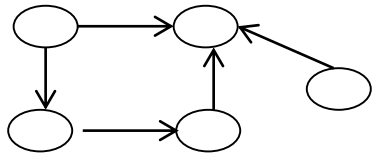
- **例如**

  - ✓ **单位的组织结构，家谱，磁盘目录文件系统**



# （3）网状结构

**（3）网状结构（图状结构）**

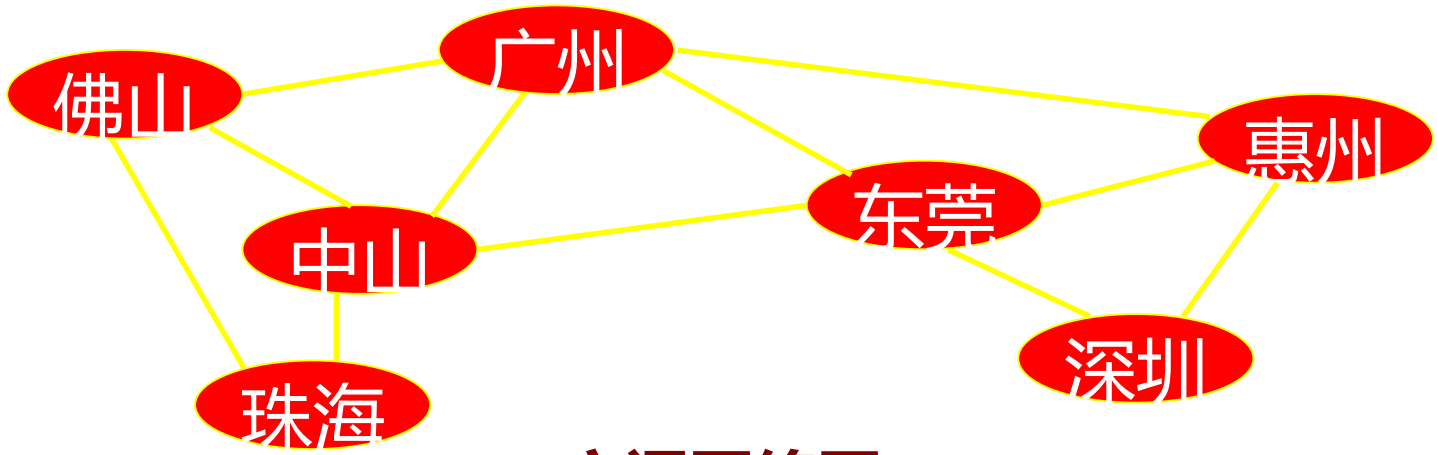


**结构中的数据元素之间存在多对多的关系**

- **特点**

  结点间的**连接**是**任意**的，结点之间**不存在包含**关系

- **例如：交通网络图，通信网络**




**交通网络图**

# 数据结构的运算

- **建立**（Create）一个数据结构
- **消除**（Destroy）一个数据结构
- **访问**（Access）一个数据结构

最基本的运算

- **插入**（Insert）：在一个数据结构中增加一个新的结点
- **删除**（Delete）：从一个数据结构中删除一个结点
- **查找**（Search）（检索）：在一个数据结构中查找满足条件的结点
- **修改**（Modify）：对一个数据结构（中的数据元素）进行修改
- **排序**（Sort）：将一个数据结构中所有结点按某种顺序重新排列
- **输出**（Output）：将一个结构中所有结点的值打印、输出

# 3.6.2 线性结构——栈

**线性结构**

**若结构是非空有限集，有且仅有一个开始结点和一个终端结点，并且所有结点都最多只有一个直接前驱和一个直接后继**，则这样的数据结构称为**线性结构**

- 可表示为：$(a_1, a_2, \ldots\ldots, a_n)$
- 某个元素的前一个元素称为该元素的**直接前驱元素**
- 某个元素的后一个元素称为该元素的**直接后继元素**

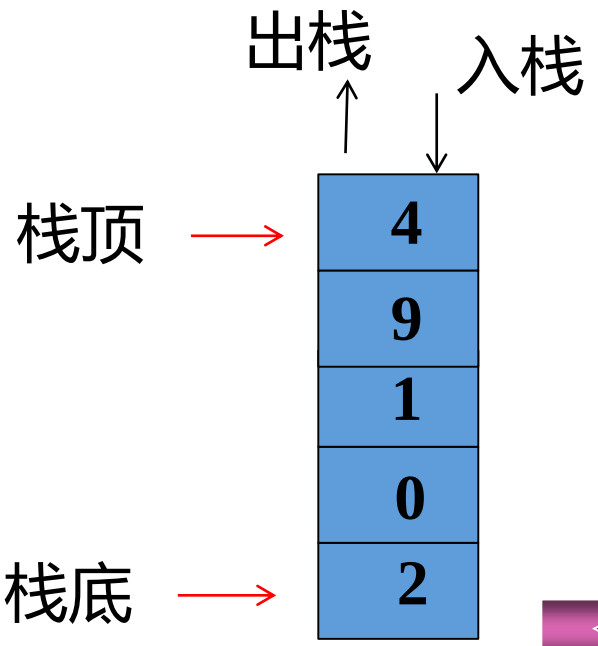
- 线性结构包括**线性表**、**栈**、**队列**、**字符串**、**数组**等
- 最典型、最常用的是---**线性表**

## 1、栈的定义

栈

**栈**（Stack）**是元素的有序集合，是限定在表尾进行添加或者移除元素操作的线性表**。又称**后进先出**（Last In First Out，**LIFO**）或**先进后出**（First in Last Out，**FILO**）线性表

- **栈顶**（Top）：**允许进行插入、删除操作的一端**，又称为**表尾**。用**栈顶指针**（top）来指示栈顶元素
- **栈底**（Bottom）：**栈的固定端**，又称为**表头**。
- **空栈：不含元素的栈**
- **入栈**或**进栈（压栈）**：栈的插入运算
- **出栈**或**退栈（弹栈）**：栈的删除运算



## 2、栈的基本操作

- **元素的添加及移除均从栈顶位置开始操作**
  - **压栈（元素入栈）**：向栈顶添加元素
  - **弹栈（元素出栈）**：将**元素从栈顶移除**，返回栈顶元素，此时**栈被修改**
  - **返回栈顶元素**：**但并不移除**，此时**栈不被修改**
  - **测试栈是否为空**：返回布尔值，若空为True，非空为False
  - **返回栈内元素的数量**：返回整型值
  - **返回栈内元素的列表**：返回列表

## 3、栈的实现

方法一

- **通过类的定义来实现栈**

自学教材或MOOC

✓ 通过**类的创建**定义栈这个抽象数据类型，对栈的操作通过定义**类的方法**实现

方法二

- **直接使用列表模拟栈，调用列表方法描述栈的操作**

✓ $s=[a_0,a_1\ldots,a_{i-1},a_i,a_{i+1},\ldots,a_{n-1}]$

栈底元素

栈顶元素

# 【课堂讨论1】

- **分别使用列表的什么方法实现压栈和弹栈?**

# 【课堂讨论1】答案

- **压栈（入栈）**：利用列表的**append**方法，每次在列表**末尾**增加一个新的元素
- **弹栈（出栈）**：利用列表的**pop()**方法，移除列表中**最后**那个元素

# 方法二：列表模拟栈的基本操作

| 栈操作 | 语句 | 操作后栈内容 | 返回值 |
|---|---|---|---|
| 创建一个空栈 | s=[] | [] | |
| 压栈 | s.append('I') | ['I'] | |
| 压栈 | s.append('am') | ['I','am'] | |
| 返回栈顶元素 | x=s[-1] | ['I','am'] | x='am' |
| 压栈 | s.append(18) | ['I','am',18] | |
| 返回栈内元素的数量 | len(s) | ['I','am',18] | 3 |
| 返回栈内元素的列表 | y=s | ['I','am',18] | y=[ ‘I’ ,'am',18] |
| 测试栈是否为空 | if s==[]: | ['I','am',18] | False |
| 弹栈 | s.pop() | ['I','am'] | 18 |
| 弹栈 | s.pop() | ['I'] | ‘am’ |

# 【举手发言】

- 弹栈与返回栈顶元素有何不同?
- 返回栈顶元素有哪几种写法?

# 【举手发言】答案

- **弹栈**是**移除**栈顶元素，并**返回**值
- **返回栈顶元素**只是**获取**栈顶元素，并**返回**值，但并不从栈顶删除该元素
- 假定栈为s，返回栈顶元素有两种方法
  - x=s[-1] #采用负数索引
  - x=s[len(s)-1] #采用正数索引

# 课后练习1

**【课后练习1】**在Python中利用列表模拟“栈”，并测试“栈”的所有操作。

# 【例3.15】栈的应用–后缀表达式

**【例3.15】根据给定的后缀表达式，求表达式的值，结果保留两位小数。**

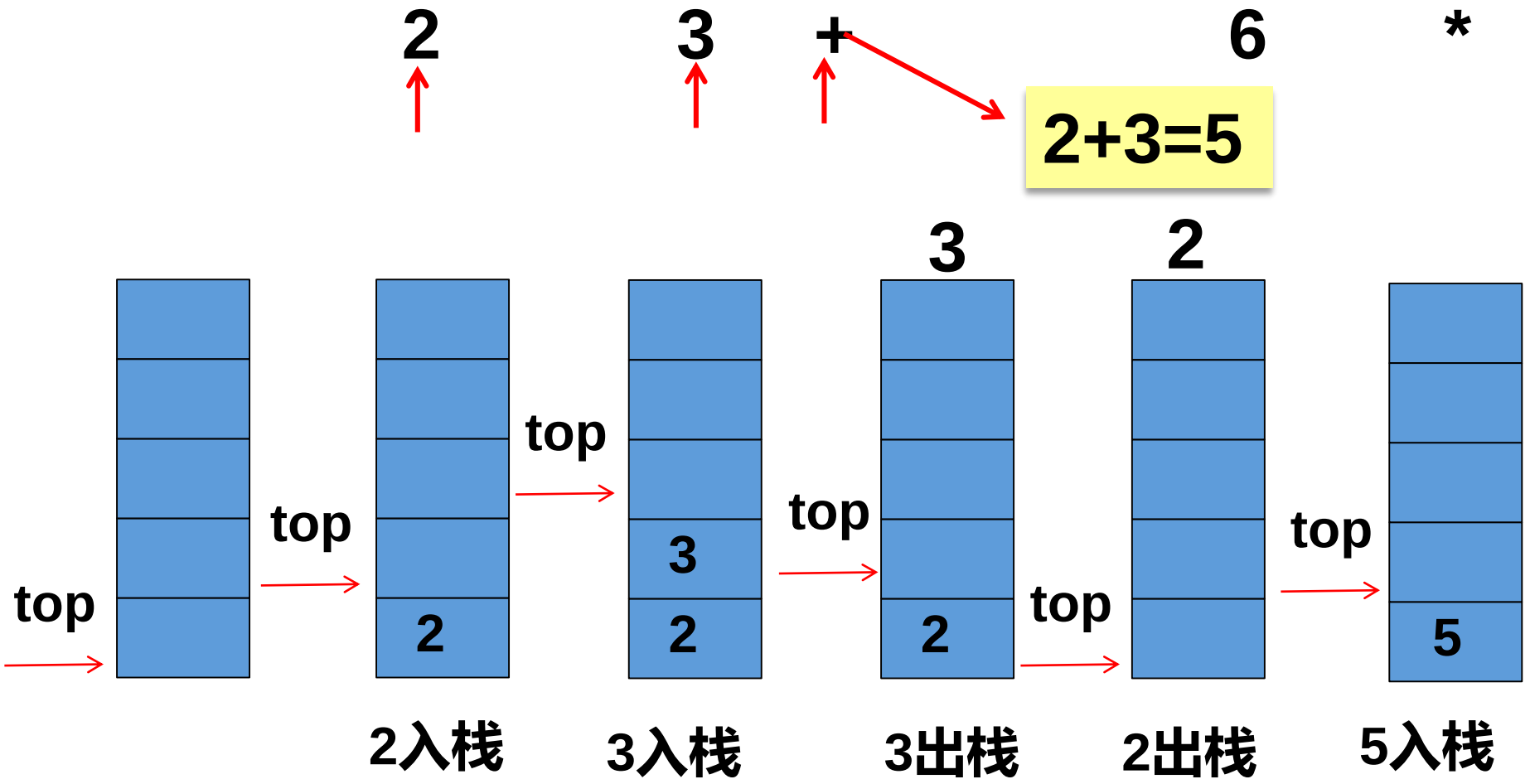
- **有效的运算符**包括+、-、*、/（四则运算）
- 每个运算对象均为**整数**（但计算过程可能产生小数），规定给定的后缀表达式总是**有效的**
- **后缀表达式**：**不包含括号**、运算符放在两个运算对象的**后面**、所有的计算按运算符出现的**顺序**严格**从左向右**进行，即无需考虑运算符的优先规则的表达式。【例】 2 3 + 6 *

# 【例3.15】设计思路

- **设计思路**
  - 对于二元运算，对后缀表达式求值可以建立一个**栈**S，用于存储**操作数**
  - **从左至右扫描**后缀表达式，如果读到**操作数**就将它**压入栈**S中
  - 如果读到**运算符**，则**取出S**中由栈顶向下的**2项**作为操作数进行**运算**，再将运算的**结果压入S**中
  - 重复此过程，直至扫描完后缀表达式，最后S栈顶的数值即为表达式的值

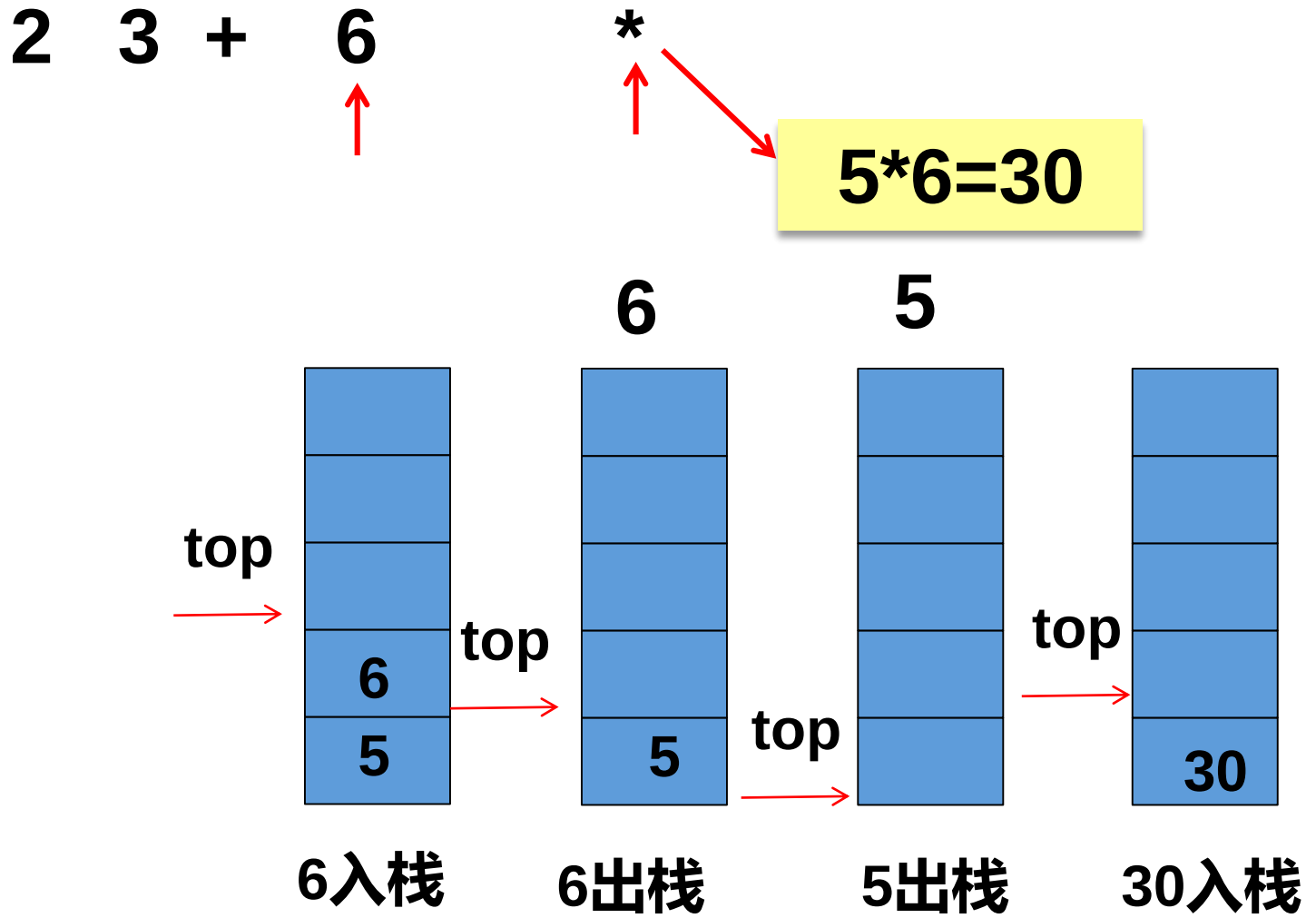
# 分析运算过程

【例】后缀表达式：2 3 + 6 *，其运算过程如下： 2+3=5， 6*5=30



# 分析运算过程（续）

- **思考：*按照中缀表达式a-b，先出栈的操作数应该赋给a还是b？***



# 采用伪代码描述算法

input 后缀表达式n

**遍历n中的字符**

if 字符是**数字**

push 数字**入栈**

else

运算符

pop 两个数字**出栈**

进行**运算**

push 计算结果入栈

输出栈顶元素

# 【例3.15】Python程序

```
n = input().split()      #将一行后缀表达式读入
stack = []                                  #栈stack
for i in n:
    if len(i) > 1 or i.isdigit():           #i为数字
        stack.append(int(i))                #若为负
    else:                                   #此时i为运算符
        b=stack.pop()                       #b先取出（后放入），应该是第二个运算元素
        a=stack.pop()                       #根据不同算符计算出结果
        if i == '+':
            s = a + b
        elif i == '-':
            s = a - b
        elif i == '*':
            s = a * b
        elif i == '/':
            s = a / b
        stack.append(s)                     #将这一步的答案放回栈中
print("%.2f"%stack[0])
```

判断是不是负数或数字

是，入栈

否则，i为运算符

数字出栈

计算结果入栈

3.15-后缀表达式【列表】.py

# 【例3.15】程序运行结果

**例：后缀表达式5 -7 2 / + 1 -**

```
5 -7 2 / + 1 -
操作数压栈后，stack变为： [5]

操作数压栈后，stack变为： [5, -7]

操作数压栈后，stack变为： [5, -7, 2]

stack中两个操作数弹栈后，stack变为： [5]
a= -7 ,b= 2
运算符为 /
中间运算结果s= -3.5
中间运算结果s压栈后stack变为： [5, -3.5]
```

-7/2=-3.5

```
stack中两个操作数弹栈后，stack变为： []
a= 5 ,b= -3.5
运算符为 +
中间运算结果s= 1.5
中间运算结果s压栈后stack变为： [1.5]

操作数压栈后，stack变为： [1.5, 1]

stack中两个操作数弹栈后，stack变为： []
a= 1.5 ,b= 1
运算符为 -
中间运算结果s= 0.5
中间运算结果s压栈后stack变为： [0.5]

0.50
```

5+(-3.5)=1.5

1.5-1=0.5

**最终计算结果**

# 3.6.3 线性结构——队列

## 1、队列的定义

**队列**

**队列**（Queue）**是元素的有序集合，向队列的一端（队尾rear）添加新的元素，而在另一端（队头front）移除现有元素**

- 队列是一种**先进先出**（First In First Out，FIFO）的**线性表**
- **队首**（front）：**允许进行元素删除的一端**
- **队尾**（rear）：**允许进行插入元素的一端**
- 队列中没有元素时称为**空队列**
- **入队**：**队列的插入操作**
- **出队**：**队列的删除操作**

## 2、队列的基本操作

- **入队**：向队尾添加新的元素
- **出队**：将元素从队首**移除**，**返回队首**元素，此时队列**被修改**
- **测试队列是否为空。**返回布尔值，若空为True，非空为False
- **返回队列内元素的数量**：返回整型值
- **返回队列内元素的列表**：返回列表

# 3、队列的实现

方法一

- **通过类的定义来实现队列** 自学教材或MOOC
  - ✓ 通过**类的创建**定义队列这个抽象数据类型，对队列的**操作**通过定义**类的方法**实现

方法二

- **直接使用列表模拟队列，调用列表方法描述队列的操作**
  - ✓ $Q=[a_0,a_1\ldots,a_{i-1},a_i,a_{i+1},\ldots,a_{n-1}]$

队首元素

队尾元素

- ✓ **入队**：利用列表的append方法
- ✓ **出队**：利用列表的pop(0)方法

# 方法二：列表模拟队列的基本操作

## 方法二：列表模拟队列的基本操作

| 队列操作 | 语句 | 操作后队列内容 | 返回值 |
|---|---|---|---|
| 创建一个队列 | q=[] | [] | |
| 入队 | q.append('Bill') | ['Bill'] | |
| 入队 | q.append('David') | ['Bill','David'] | |
| 入队 | q.append('Susan') | ['Bill','David', 'Susan'] | |
| 返回队列内元素的数量 | len(q) | ['Bill','David', 'Susan'] | 3 |
| 返回队列内元素的列表 | y=q | ['Bill','David', 'Susan'] | y=['Bill','David','Susan'] |
| 测试队列是否为空 | if q==[]: | ['Bill','David', 'Susan'] | False |
| 出队 | q.pop(0) | ['David', 'Susan'] | 'Bill' |
| 出队 | s.pop(0) | ['Susan'] | 'David' |

# 【课堂讨论2】

- **使用列表实现栈和队列，二者在操作上有何区别?**
  - 操作的位置
  - 使用的方法

# 课后练习2

**【课后练习2】**在Python中利用列表模拟“队列”，并测试“队列”的所有操作。

# 【例3.16】队列的应用–烫手的山芋

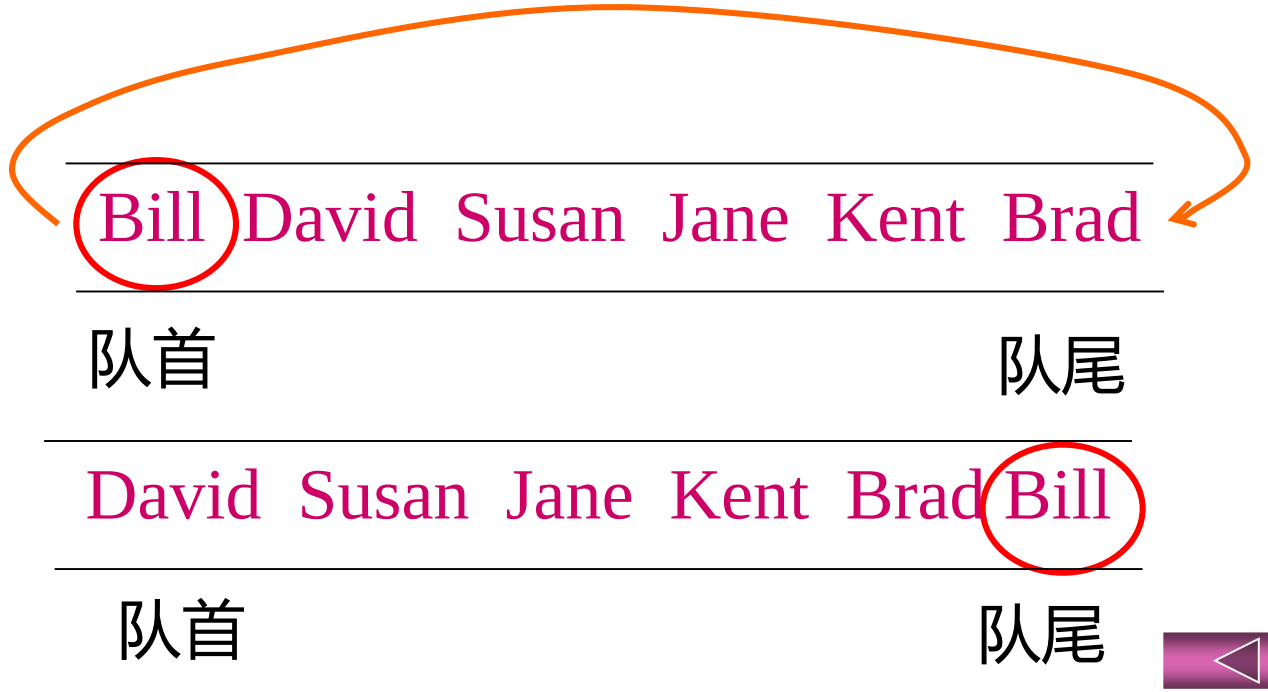
**【例3.16】儿童游戏：烫手的山芋（hotpotato）。**

- 在这个游戏中，孩子们围成一圈，把手里的东西一个传一个。
- 规定经过传递一定的次数后，则停止传递，当时手上拿着山芋的孩子就要退出游戏。烫手的山芋被交给下一个孩子，重新开始此游戏。**每当达到规定的传递次数，就有一个孩子退出游戏**。……直到只剩一个人，则此人为胜利者
- **要求**：通过键盘输入**孩子人数**、孩子的**名字**以及**循环传递的次数**；模拟烫手的山芋的游戏过程。给出游戏胜利者的名字

# 【例3.16】问题分析

- 把圆圈想象成一个**首尾相接的队列**
- 假定开始拿着山芋的孩子站在**队首**，传出山芋后，则被移出队列，然后**将他加入队尾**
- 原来在他后面的孩子则变成队首；下一次，该孩子被移除队首，加入队尾……

**用队列模拟游戏过程**



# 【例3.16】设计思路

## ■ 设计思路

- **程序包括输入过程和处理过程**
  - ✓ **输入过程**包括输入孩子**人数**kidnum、孩子的**名字**name以及循环传递的**次数**num
    - 采用**列表**namelist作为存储所有孩子的**队列**
    - 采用**while语句**按初始顺序分别输入孩子的名字，循环变量为i，循环条件为**i <= kidnum**；并采用列表的**append方法**将其加入到队列namelist的尾部

# 【例3.16】设计思路（续）

- ✓ **处理过程**采用**两重while循环**来实现
  - **外层循环**（while **kidnum > 1** ）处理当剩余孩子数量多于1个的情况，循环变量**kidnum**为剩余孩子数量 。每当达到规定的传递次数num，则用**del**语句**删除此时拿着山芋的孩子，** **kidnum**减1；直至**kidnum**为1，则终止循环

```
del namelist[0]                 # 删除队首
kidnum -= 1                     #孩子总数减1
```

  - **内层循环**（whlie **j <= num**）实现在规定循环传递次数内时对队列的操作（**移除队首的孩子，并添加到队尾**）。循环变量j为循环传递的次数

```
namelist.append(namelist.pop(0))
```

# 【例3.16】的程序

**例3.16-hotpotato.py**

```
#（1）输入过程
namelist = []              #空列表，作为存储参加游戏的所有孩子的队列
kidnum =int( input("总共有多少个孩子：")) #孩子的个数
i = 1
while i <= kidnum:     #按初始顺序输入孩子的名字，将其加入到队列尾部
    name=input("请输入第%d个孩子的名字：" %i)
    namelist.append(name)          #将第i个孩子加入到队列尾部
    i += 1                         #改变循环变量的值


num = int(input("循环传递的次数：")) #循环传递的次数
```

提示语能随i的变化而变化

# 【例3.16】的程序（续）

```
# （2）处理过程——两重while循环
while kidnum > 1:                    #当剩余孩子数量多于1个
    j= 1                             #计循环传递的次数
    print ("----------------------------")
    print ("初始队列为：", namelist)
    while j <= num:                  #当j小于等于规定的循环传递次数 时
        namelist.append(namelist.pop(0))  #将第一个孩子的名字移除，加入队尾
        print ("第",j,"次传递")
        print ("队列变为：",namelist)
        j += 1                       #循环传递次数加1
    print (namelist[0],"退出游戏“)
    del namelist[0]                  # 删除此时拿着山芋的孩子（队首）
    kidnum -= 1                      #孩子总数减1

print (“游戏的胜利者是：",namelist[0])
```

# 【例3.16】程序运行结果

第一轮游戏

第二轮游戏

```
>>>
总共有多少个孩子：3
请输入第 1 个孩子的名字：
a
请输入第 2 个孩子的名字：
b
请输入第 3 个孩子的名字：
c
循环传递的次数：4
--------------------------------
初始队列为： ['a', 'b', 'c']
第 1 次传递
队列变为： ['b', 'c', 'a']
第 2 次传递
队列变为： ['c', 'a', 'b']
第 3 次传递
队列变为： ['a', 'b', 'c']
第 4 次传递
队列变为： ['b', 'c', 'a']
b 退出游戏
--------------------------------
初始队列为： ['c', 'a']
第 1 次传递
队列变为： ['a', 'c']
第 2 次传递
队列变为： ['c', 'a']
第 3 次传递
队列变为： ['a', 'c']
第 4 次传递
队列变为： ['c', 'a']
c 退出游戏
游戏的胜利者是： a
```

经过4次传递以后，**b**变成了**队首**，应退出游戏