北京航空航天大學
BEIHANG UNIVERSITY



大学计算机基础

（理科类）

第7讲 Python实现自定义数据结构

北京航空航天大学

第6讲回顾

【实验任务 4-2】颜色代码转换器。现给出三个以空格隔开的数字，分别代表某颜色的红、绿、蓝三个通道强度值，有效强度值范围为 **0 ~ 255**（十进制）的整数。当输入的强度值超出有效范围时，必须舍入到最近的有效值（如：如果一个通道的输入强度值为 300，应舍入到 255；如果是 -50，应舍入到 0）。

请输出红、绿、蓝三个通道所确定的颜色的十六进制表示。当某一通道转换为十六进制数不足两位时，要**用 0 补齐**；所有的字母字符（A, B, C, D, E, F）均**大写**。

- **输入样例1**

  1 2 3

- **输出样例1**

  #010203

- **输入样例2**

  -20 275 125

- **输出样例2**

  #00FF7D

【程序纠错】

- **错在哪?**

```
#（1）输入
a, b, c = map(int, input().split()) #某颜色红、绿、蓝三个通道强度值

#（2）将单独的一个颜色转换成十六进制数
if a<0:                 #如果是负数，应舍入到 0
    a1='00'
elif a>255:
    a1='FF'             #大于255，舍入到 255，对应的十六进制数为'FF'
else:
    a1=hex(a)[2:]       #hex()返回值的前2位为'0x'。故采取分片的方法，只取数值部分
    if len(a1)==1:      #如果只有1位
        a1='0'+a1       #前面补0
    a1.upper()          #返回转换为大写字母的副本

if b<0:
    b1='00'
elif b>255:
    b1='FF'
else:
    b1=hex(b)[2:]
    if len(b1)==1:
        b1='0'+b1
    b1.upper()
```

【程序纠错】（续）

```
if c<0:
    c1='00'
elif c>255:
    c1='FF'
else:
    c1=hex(c)[2:]
    if len(c1)==1:
        c1='0'+c1
    c1.upper()

ans='#'+a1+b1+c1    #将红、绿、蓝的颜色强度值（十六进制数）连接成一个字符串

#（3）输出
print(ans)
```

```
1 2 3
#010203
>>> ==========
>>>
-20 275 125
#00FF7d
>>> ==========
>>>
2 21 28
#02151c
```

- **思考：这样写的代码好吗？如何优化？**

1、如何定义函数?

- **给函数起一个有意义的名字**
  - ✓ 如“to_hex”
  - ✓ 当一个程序中有好几个函数时，最好不要用f1、f2……命名，以免分不清含义
- **明确函数的参数是什么?**
  - ✓ 一般是**关键变量**，传入不同的参数，会得到不同的计算结果
  - ✓ 如本题的颜色强度值（十进制数）
- **明确函数的返回值是什么?**
  - ✓ 一般采用**return语句**，返回函数计算的结果
  - ✓ 如果在不同情况下，有不同的计算结果，则在函数体中，采用**if语句**进行分支操作，每种情况下返回相应的计算结果

2、如何调用函数？

调用函数的格式 **函数名(<实参>)**

- 如果函数在定义时有**形参**，则在调用函数时必须给函数提供实际参数（**实参**）。如“**to_hex(r)**”
- 调用函数时，**实参被传递给形参**，然后执行函数体，直到遇到return语句或者执行完函数体中所有的语句
  - ✓ 若有return语句，将返回“return”之后的**表达式的值**
  - ✓ 若没有return语句，则返回**None**，执行点转移至调用点之后的代码

目　录

重点

本节课主要内容

- 一、数据结构
- 二、抽象数据类型
- 三、线性结构—栈
- 四、线性结构—队列
- 五、实验4总结

一、数据结构

北京航空航天大学

数据类型

- **数据类型是某一类值的集合以及定义在此集合上的一组操作的总称**
  - 提供对二进制数据的解释，将数据与所需解决的问题进行关联
- 数据类型分为两大类：**基本数据类型**和**抽象数据类型**
  - **基本数据类型**包括**简单**数据类型和**结构化**数据类型
  - **抽象数据类型**是一个数据模型及定义在该数据模型上的一组操作。是程序设计语言中没有的、由用户**自定义**的数据类型。Python中通过创建**类**来定义

数据结构

- 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**就是**指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7.1】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

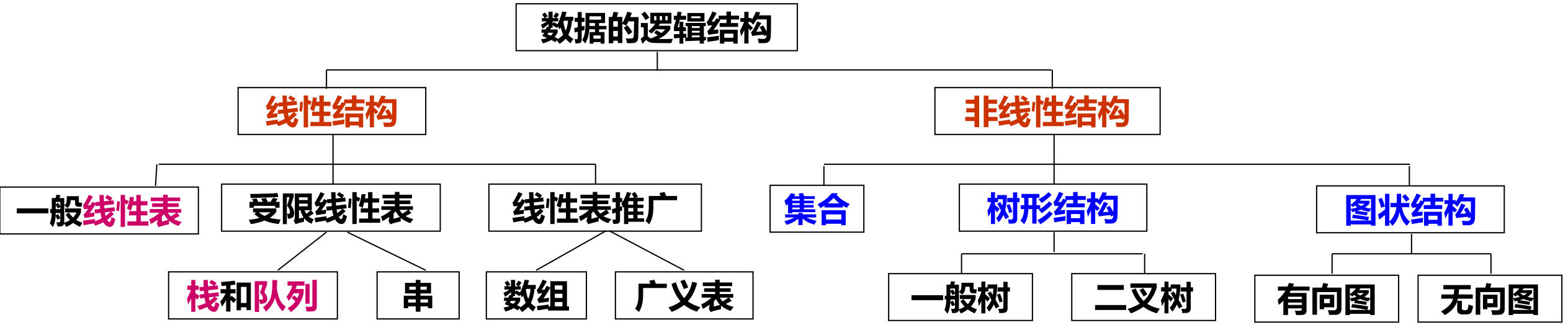
- **数据项分为基本项和组合项，每一个基本项或组合项称为一个字段**
  - ✓ **基本项**：指有独立意义的最小标识单位
  - ✓ **组合项**：由一个或多个基本项组成的有独立意义的标识单位

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

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

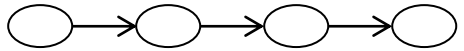


数据逻辑结构层次关系图

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**）；

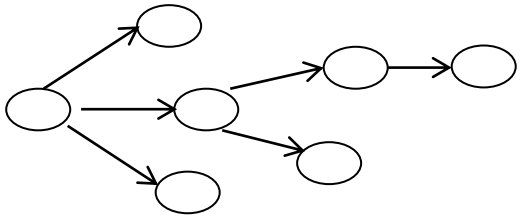
  4）除最后一个以外，集合中的每一个元素都只有一个**后继**（**某个元素的后一个元素**）。

- **例如：通迅录、成绩单、花名册**

- **【例7.1】市话用户信息表**

（2）树形结构

（2）树形结构



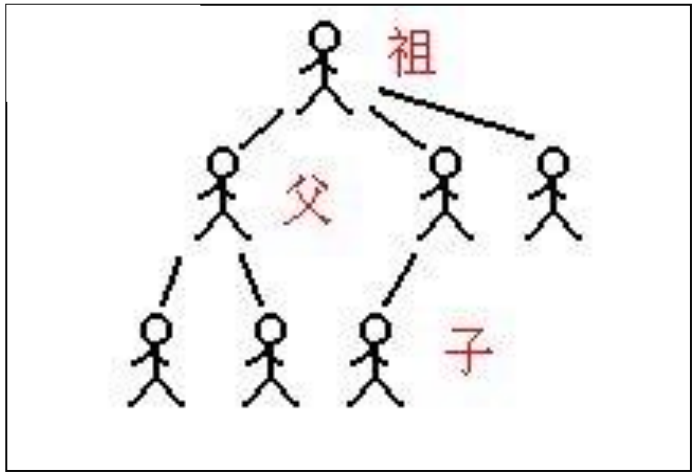
结构中的数据元素之间存在一对多的关系

- **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

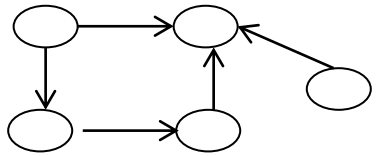
- **例如**

  - ✓ **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）

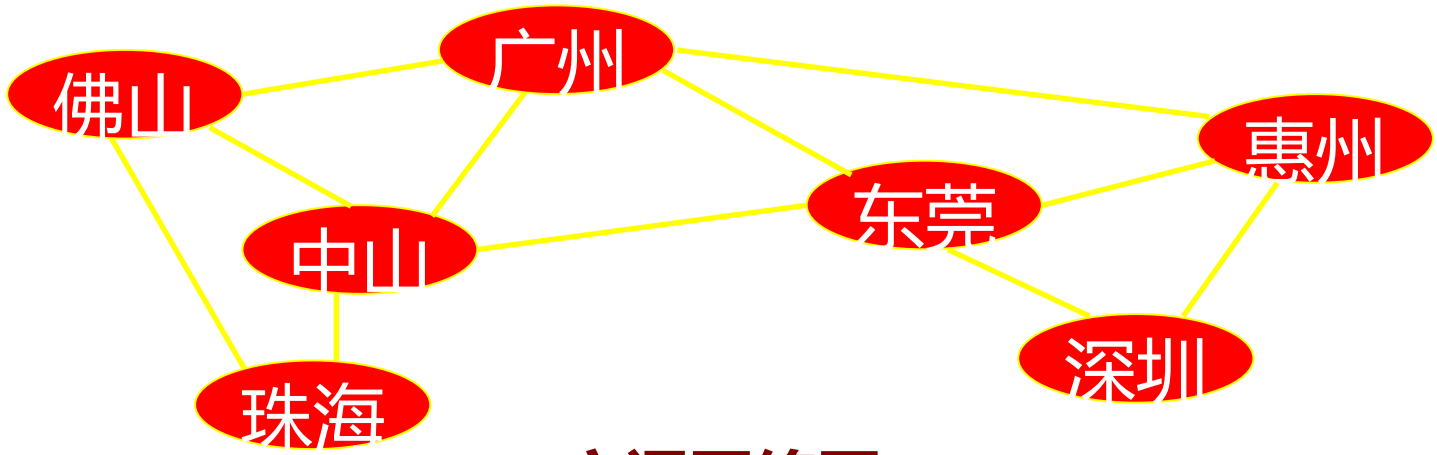


结构中的数据元素之间存在多对多的关系

- **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**

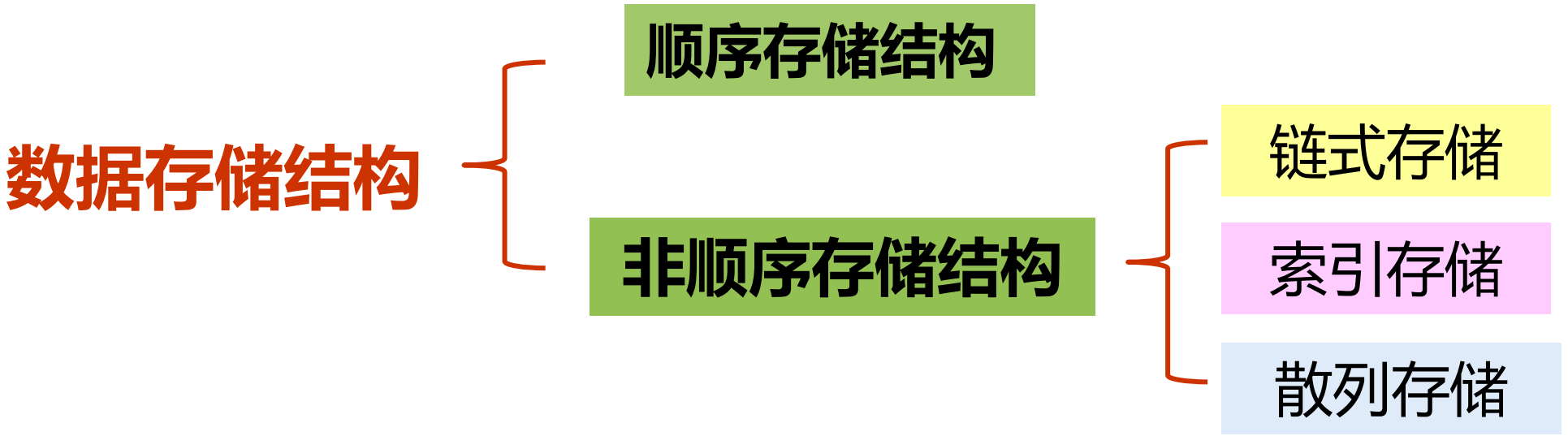


交通网络图

数据的存储结构

- 数据的**逻辑结构**只是描述了数据元素之间的逻辑关系，**与数据在计算机中的存储方式无关**
- 如果数据在计算机中无规律地存储，则数据元素在内存中难以迅速找到，将大大影响数据处理的速度！
- 两种存储结构
  - **顺序存储结构**：使**逻辑**上**相邻**的元素在存储器中的**位置也相邻**
  - **非顺序存储结构**：对逻辑上相邻的元素不要求其物理地址相邻

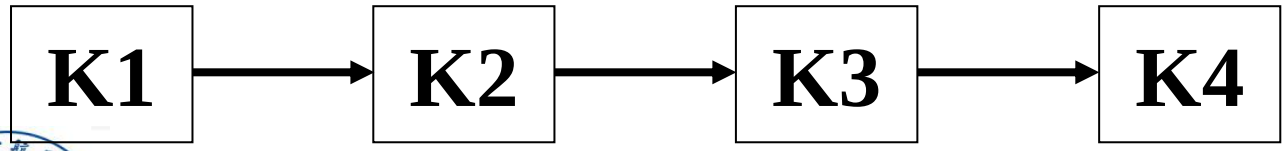
数据存储结构分类



- **数据的逻辑结构和存储结构是密不可分的两个方面**
  - 算法的**设计**取决于所选定的**逻辑**结构
  - 算法的**实现**依赖于所采用的**存储**结构

顺序存储结构

- **顺序存储结构：将逻辑上相邻的元素存储在物理位置相邻的存储单元中**
- 常用于存储具有**线性结构**（如**线性表**、**栈**、**队列**、**串**和**数组**）的数据
- 逻辑相邻的结点在物理位置上一定也相邻，易于查找任意一个结点数据



逻辑结构

物理结构

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

北京航空航天大學
BEIHANG UNIVERSITY

二、抽象数据类型

北京航空航天大学

抽象数据类型

- **抽象数据类型是一个数据模型及定义在该数据模型上的一组操作**
  - 抽象数据类型由**用户自己定义**，用来扩展程序设计语言的数据类型
  - 比如**直线**、**分数**抽象数据类型
- **Python通过创建一个新的类（class），来定义一个新的数据类型，对某类数据进行特定的操作**

类

- **类**是一组具有**共同特性**（包括属性、操作、方法、关系、行为）的所有对象成员的**抽象描述**
  - 具有**相同或相似性质**的对象的**抽象**就是类。因此，类的**具体化**就是对象，或类的**实例**是对象
  - 类具有**属性**，它是对象的**状态**的抽象，用**数据结构**来描述类的**属性**
  - 类具有**操作**，它是对象的**行为**的抽象，用**操作名**和实现该操作的**方法**来描述
- **实例化**
  - 创建了一个类，即可创建该类的**实例**——**对象**

如何创建类？

首字母大写

```
class 类名:
    <类属性的定义>
    def __init__(self,<参数>):
        <实例属性的定义>
    def <成员函数名> (self,<参数>):
        <成员函数的定义>
```

构造方法，创建该类的数据对象，定义实例属性

定义类的方法

构造方法

- 所有类需要提供**构造方法**（constructor），用以创建该类的数据对象，**初始化**对象的**属性**
  - 在Python中，构造方法由__init__标识

类的名字

形参列表

```
class Line :
    def __init__(self , A, B, C):
        self.A = A
        self.B = B
        self.C = C
```

注意：“__init__”中的“__“是两个下划线

self参数代表将来要创建的对象自身

构造方法的形参列表

- **形参列表第一项为self，后面为代表实例属性的变量（如果有）**
  - **当创建类的实例时，将实参传递给形参**
  - self为特殊参数，是指向对象自身的引用，必须为形参的**第一项**，但是**在调用时不会有实参对应**
  - **本例形参列表包含self、A、B、C**
    - ✓ 构造方法中成员变量（实例属性）**self.A、 self.B、 self.C**分别定义实例对象的A、B、C

【例7.2】创建类和使用类

【例7.2】在平面直角坐标系上，任意一条直线都可以由Ax+By+C=0来表示。请你定义一个**直线的类**。

- 这个类的参数有3个，分别为：A、B、C。
- 假定这个类的方法有2个，分别为：

（1）计算原点（0，0）到该直线的**距离**，以return返回结果

（2）计算该直线与x、y坐标轴围成的三角形的**面积**（若直线与坐标轴重合，则面积为0），以return返回结果

- 要求：生成一条**直线的实例**，输出该条直线实例的所有属性（即A、B、C），输出2个方法所返回的结果

【例7.2】的Python程序

```
#1、定义一个直线的类
class Line:
    # （1）初始化
    def __init__(self, A, B, C):  #Ax+By+C=0，类的参数：A、B、C
        self.A = A                #定义实例属性
        self.B = B
        self.C = C
    # （2）计算原点到直线距离
    def cal_dis_0(self):
        distance = abs(self.C) / ((self.A * self.A + self.B * self.B) ** 0.5)
        return distance
```

构造方法：创建类的数据对象，定义实例属性

定义成员函数

【例7. 2】的Python程序（续）

```
# （3）计算直线与坐标轴围成的面积
  def cal_area(self):
      if self.A == 0 or self.B == 0 or self.C == 0:
          return 0
      b = -self.C / self.B                    #截距
      a = -self.C / self.A                    #横坐标
      return abs(a * b / 2)

#2、创建直线的实例
line1 = Line(3,4,5)

# 3、输出直线实例的属性、方法结果
print("该直线实例的属性：A=%d，B=%d，C=%d" % (line1.A, line1.B, line1.C))
print("原点到直线距离：%.3f" % line1.cal_dis_0())    # 计算原点到直线距离
print("与坐标轴围成图形的面积：%.3f" % line1.cal_area()) #计算面积
```

调用成员函数

访问实例的属性

【例7.2】的程序运行结果

```
>>>
该直线实例的属性：A=-2, B=5, C=2
原点到直线距离：0.371
与坐标轴围成图形的面积：0.200
>>>
```

- **如何创建类和使用类?**

（1）在“**class <类名>:**”下面利用**构造方法**创建**类的数据对象**，定义**成员函数**来描述类的各种**行为**

（2）创建类的一个或多个**实例**：**实例名=类名.(<实参>)**

（3）采用“**Object.attribute**”输出实例的**属性值**，采用“**Object.function( )**”输出**方法的结果**

北京航空航天大學
BEIHANG UNIVERSITY

三、线性结构—栈

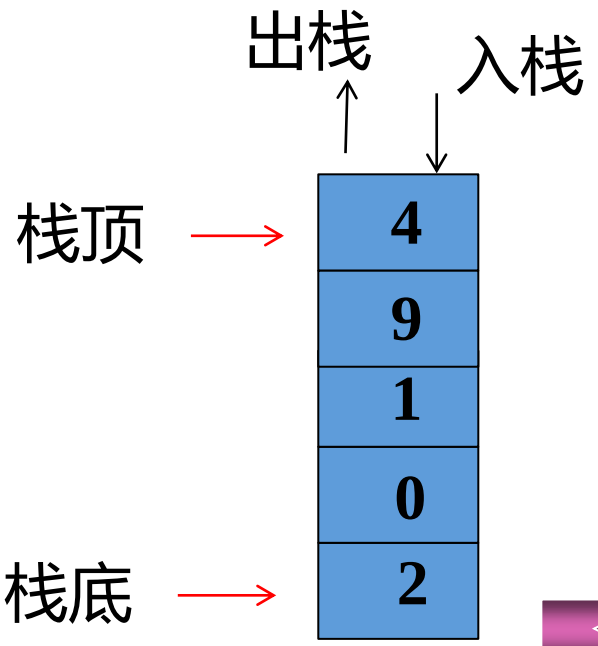
北京航空航天大学

线性结构

- **线性结构的定义**
  - **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，……，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**（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、栈的抽象数据类型

- **栈的抽象数据类型**
  - **数据元素：可以是任意类型的数据，但必须同属于一个数据对象**
  - **数据关系：栈中元素之间是线性关系**
  - 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
  - **基本操作：** push(item)、pop()、peek()、isEmpty()、size()等

栈的基本操作

- **元素的添加及移除均从栈顶位置开始操作**
  - **push(item)，压栈（元素入栈）。**向栈顶添加元素，**参数为向栈添加的元素**，无返回值
  - **pop()，弹栈（元素出栈）。**将**元素从栈顶移除**，无需参数，返回栈顶元素，此时**栈被修改**
  - **peek()，返回栈顶元素。但并不移除**，无需参数，此时**栈不被修改**
  - **isEmpty()，测试栈是否为空。**无需参数，返回布尔值
  - **size()，返回栈内元素的数量。**无需参数，返回整型值
  - **get_items()，返回栈内元素的列表。**无需参数，返回列表

- **思考：*pop()与peek()有何不同?***

3、栈的实现

- **Python中实现栈有两种方法**
  - **方法一：通过类的定义来实现栈**
    - ✓ 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

【讨论1】

- **分别使用列表的什么方法实现压栈和弹栈?**

（1）定义抽象数据类型“栈”

（1）在Python中定义抽象数据类型“栈”

Stack_def.py

```
class Stack:
    def __init__(self):                 #栈的初始化
        self.items = []                 #创建数据对象——栈，初始值为空列表

    def push(self, item):               #压栈（元素入栈）
        self.items.append(item) #在列表末尾增加新的对象

    def pop(self):                      #弹栈（元素出栈）
        return self.items.pop()     #移除列表中最后一个元素，并返回该元素的值
```

创建的数据对象：栈

欲添加的元素

定义抽象数据类型“栈”（续）

成员函数

```
def peek(self):                 #返回栈顶元素
    return self.items[len(self.items)-1]

def isEmpty(self):              #测试栈是否为空
    return self.items == []

def size(self):                 #返回栈内元素的数量
    return len(self.items)

def get_items(self):            #返回栈内元素的列表
   return  self.items
```

【举手发言】

- **请思考：下面两种操作有何区别?**

（1）弹栈

```
def pop(self):
        return self.items.pop()
```

（2） 返回栈顶元素

```
def peek(self):
        return self.items[len(self.items)-1]
```

还可以怎么写?

（2）创建栈对象的实例

- 完成对Stack的定义后，便可以在同一程序使用该类，或在另一程序中**导入Stack类，再使用该类**

```
from Stack_def  import Stack
```

- 然后**创建Stack的对象实例**，进而**调用栈的方法**进行相应的操作

课后练习

【例7.2】在Python中创建抽象数据类型“栈”的实例，并测试“栈”的所有操作。

【例7.3】栈的应用–后缀表达式

【例7.3】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7.3】设计思路

设计思路

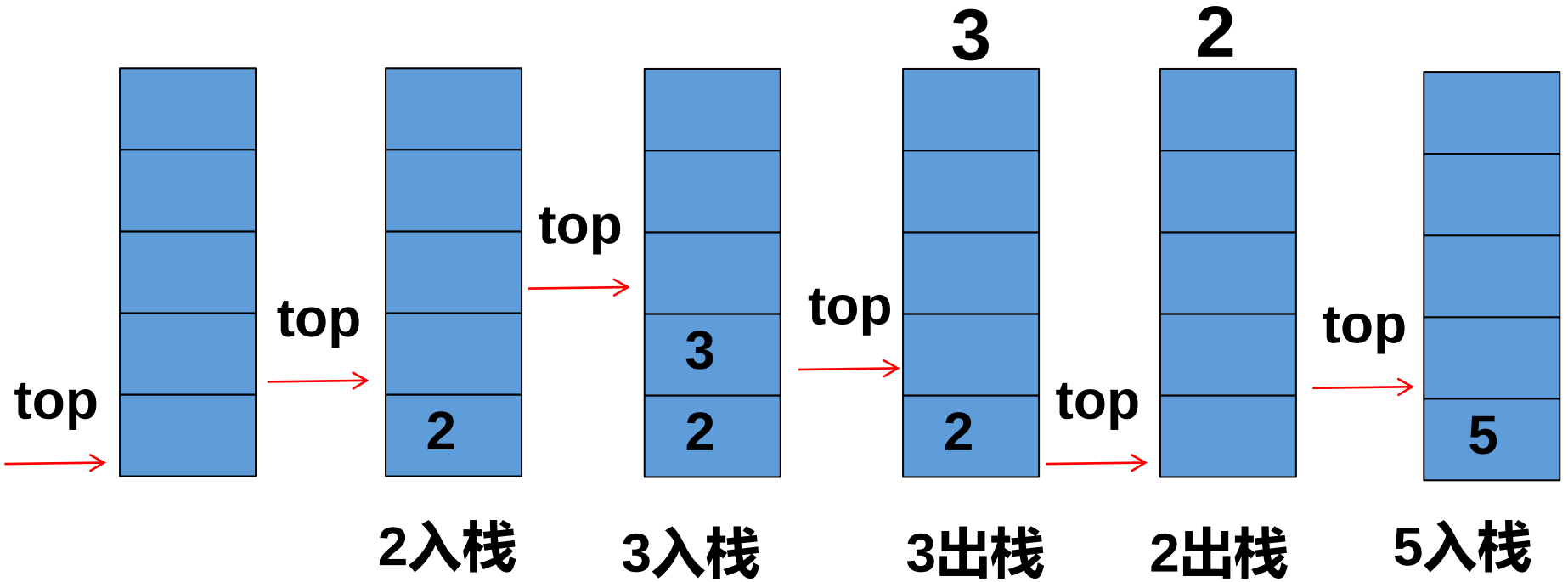
- 对于二元运算，对后缀表达式求值可以建立一个**栈**S，用于存储**操作数**
- **从左至右扫描**后缀表达式，如果读到**操作数**就将它**压入栈**S中
- 如果读到**运算符**，则**取出S**中由栈顶向下的**2项**作为操作数进行**运算**，再将运算的**结果压入S**中
- 重复此过程，直至扫描完后缀表达式，最后S栈顶的数值即为表达式的值

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

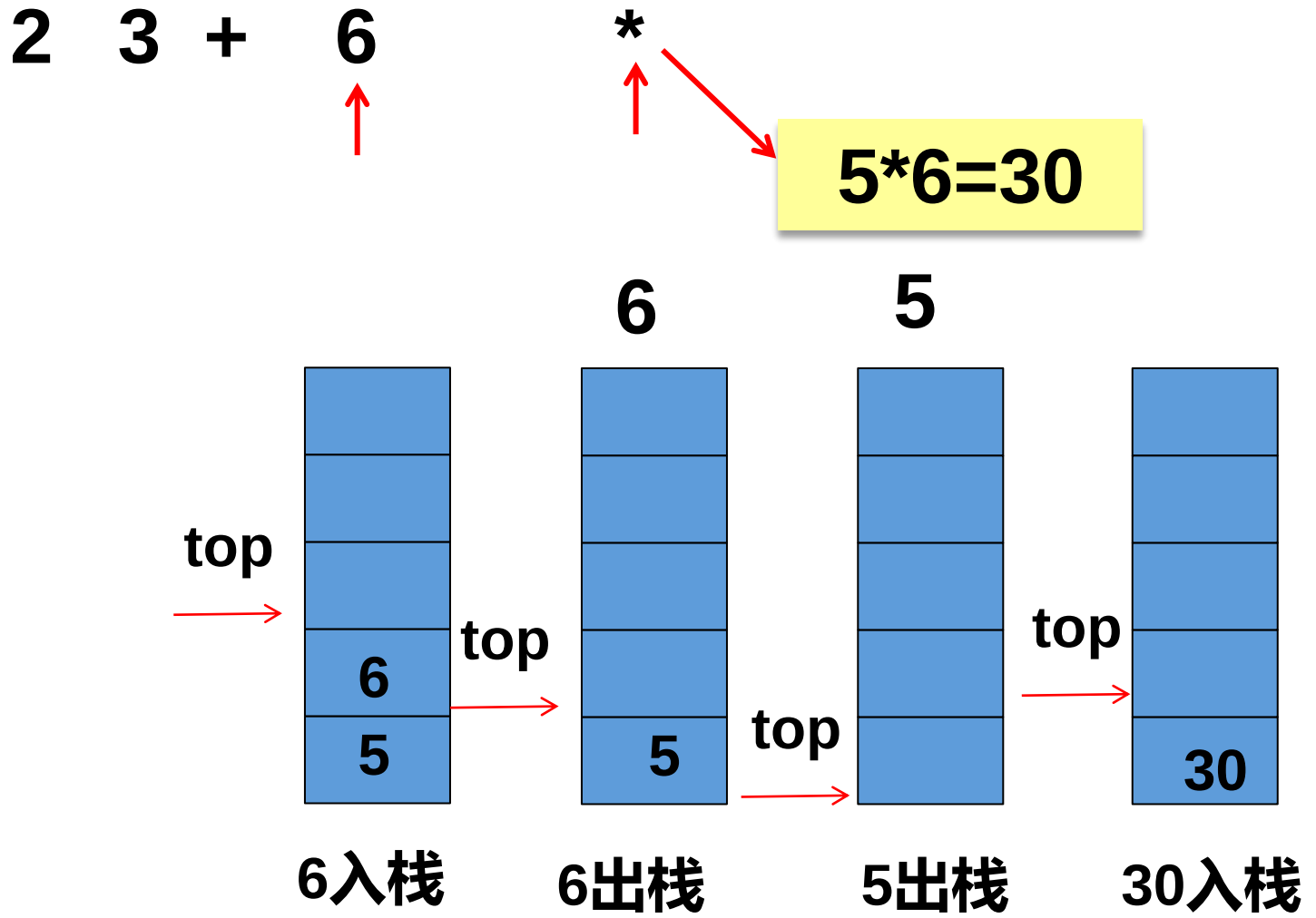
2 3 + 6 *

2+3=5



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

方法一：使用列表模拟栈

```
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

7.3-后缀表达式【列表】.py

【例7.3】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

方法二：创建栈类的实例来模拟栈

7.3-后缀表达式【栈类】.py

导入栈类

实例化

调用栈类的方法

```
from Stack_def import Stack

n = input().split()
s=Stack()
for i in n:
    if len(i) > 1 or i.isdigit():
        s.push(int(i))
    else:
        b=s.pop()
        a=s.pop()
        if i == '+':
            total = a + b
        elif i == '-':
            total = a - b
        elif i == '*':
            total = a * b
        elif i == '/':
            total = a / b
        s.push(total)
print("%.2f"%s.peek())
```

```
#将一行后缀表达式读入
#创建栈Stack的实例

#i为数字
#若为负数则len(i)必然>1
#此时i为运算符

#b先取出（后放入），应该是第二个运算元素
#根据不同算符计算出结果

#将这一步的答案放回栈中
#获取栈顶元素，即为最终计算结果
```

北京航空航天大學
BEIHANG UNIVERSITY

四、线性结构—队列

北京航空航天大学

1、队列的定义

- **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、队列的抽象数据类型

- 队列抽象数据类型可由如下的**结构**和**操作**进行定义
  - **class Queue**：**创建一个队列类**
  - **enqueue(item)**：**向队尾添加新的元素。**参数item为向队列添加的元素，无返回值
  - **dequeue()**：**将元素从队首移除。**无需参数，**返回队首**元素，此时队列**被修改**
  - **isEmpty()**：**测试队列是否为空。**无需参数，返回布尔值
  - **size()**：**返回队列内元素的数量。**无需参数，返回整型值
  - **get_items()**，**返回队列内元素的列表。**无需参数，返回列表

3、Python中如何实现队列？

- **方法一**：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
  - 采用**列表**存储队列的各元素，按照人们的习惯，列表的位置0存放队首元素，末尾存放队尾元素
  - $Q=[a_0,a_1\ldots,a_{i-1},a_i,a_{i+1},\ldots,a_{n-1}]$

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

【例7.4】定义抽象数据类型-队列

【例7.4】在Python中定义抽象数据类型“队列”。

```
#queue_class_good.py
#列表的位置0存放队首元素，末尾存放队尾元素。每次在列表末尾插入新元素，在列表的位置0移除元素
#这种方法更符合人们的习惯

class Queue:
    def __init__(self):          #创建一个新的空队列
        self.items = []          #空列表

    def isEmpty(self):           #测试队列是否为空
        return self.items == []

    def enqueue(self,item):      #向队尾添加新的元素
        self.items.append(item)      #在列表的末尾添加一个元素

    def dequeue(self):           #将元素从队首移除
        return self.items.pop(0)     #将列表位置0的元素移除

    def size(self):              #返回队列内元素的数量
        return len(self.items)

    def get_items(self):         #返回队列内元素的列表
        return  self.items
```

self.items即是创建的数据对象——队列

欲添加的元素

例7.4-queue_class_good.py

利用列表模拟队列

- **方法二：利用列表来模拟队列**
  - 使用append()方法把元素添加到队尾
  - 使用pop(0) 方法把队首的元素删除

```
>>> queue = ['a', 'b', 'c']
>>> queue.append('d')
>>> queue.append('e')
>>> queue
['a', 'b', 'c', 'd', 'e']
>>> queue.pop(0)
'a'
>>> queue
['b', 'c', 'd', 'e']
>>> queue.pop(0)
'b'
>>> queue
['c', 'd', 'e']
```

【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

【例7.5】烫手的山芋

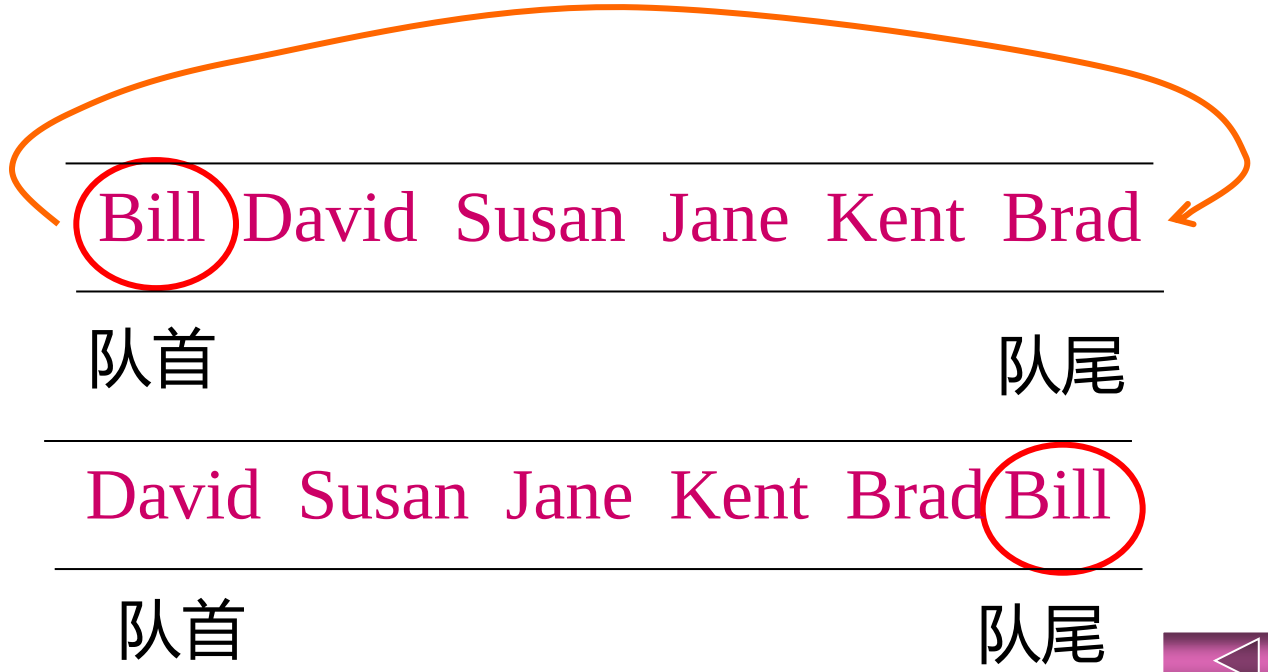
【例7.5】儿童游戏：烫手的山芋（hotpotato）。

- 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7.5】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7.5】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7.5】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7.5】的程序

例7.5-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7.5】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7.5】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