北京航空航天大學
BEIHANG UNIVERSITY

大学计算机基础

（理科类）

第5讲 Python的基本语法（一）

北京航空航天大学

目　录

北京航空航天大學
BEIHANG UNIVERSITY

5.1 程序控制结构

北京航空航天大学

程序控制结构

- 结构化程序设计有**3种**基本结构：**顺序控制结构**、**选择控制结构**和**迭代控制结构**
- **顺序控制结构**
  - 串行程序**按序**执行语句，当语句执行完毕则停止
- **选择控制结构（分支结构）**
  - 根据**条件**进行分支
    - ✓ if语句
- **迭代控制结构（循环结构）**
- 在一定**条件**下重复执行**相同**的程序段
    - ✓ while语句，for语句

1、选择控制结构

- **选择结构（分支结构）：根据条件的判断确定应该执行哪一条分支的语句序列**
  - 最简单的分支控制语句为**条件语句（if语句）**
  - 条件语句结构分为三种形式
    - ✓ **非完整性**if语句
    - ✓ **二重选择**的if语句
    - ✓ **多重选择**的if语句

条件语句结构

(1) 非完整性if语句

```
if 条件表达式:
    语句序列
```

(2) 二重选择的if语句

```
if 条件表达式:
    语句序列1
else:
    语句序列2
```

- 其中“条件表达式”可以为**布尔**表达式或**关系**表达式，或**布尔变量**。

条件语句结构：（3）多重选择的if语句

（3）多重选择的if语句

（有**两个或两个以上**条件）

```
#if_example3.py
#判断输入的数是正数、负数或零

num=float(input('Enter a number:'))

if num>0:
    print ('The number is positive')
elif num<0:
    print ('The number is negtive')
else:
    print ('The number is zero')
```

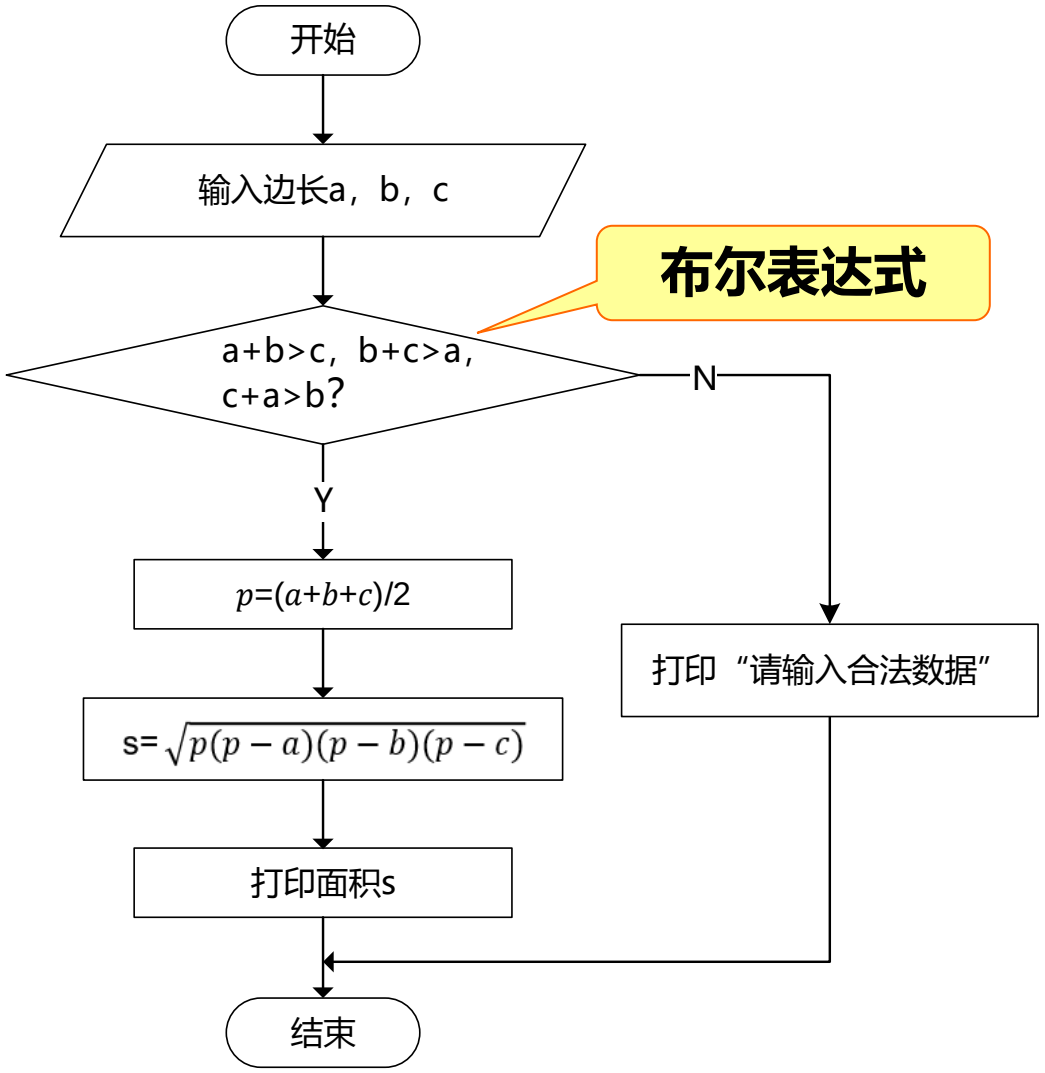
```
if 条件表达式1:
        语句序列1
elif 条件表达式2:
        语句序列2
else:
        语句序列3
```

```
Enter a number:5
5.0 is positive
>>> ==============
>>>
Enter a number:-19
-19.0 is negtive
>>> ==============
>>>
Enter a number:0
0.0 is zero
```

条件语句示例：【例5. 1】

- **【例5.1】输入三角形的三条边长，求三角形的面积。**

- **设计思路**
  - 设a、b、c为三角形的三条边长，构成三角形必须满足：**a+b>c，b+c>a，c+a>b**
  - 则根据**海伦公式**，三角形的面积 **s=$\sqrt{p(p-a)(p-b)(p-c)}$**，其中，**p = (a + b + c)/2**

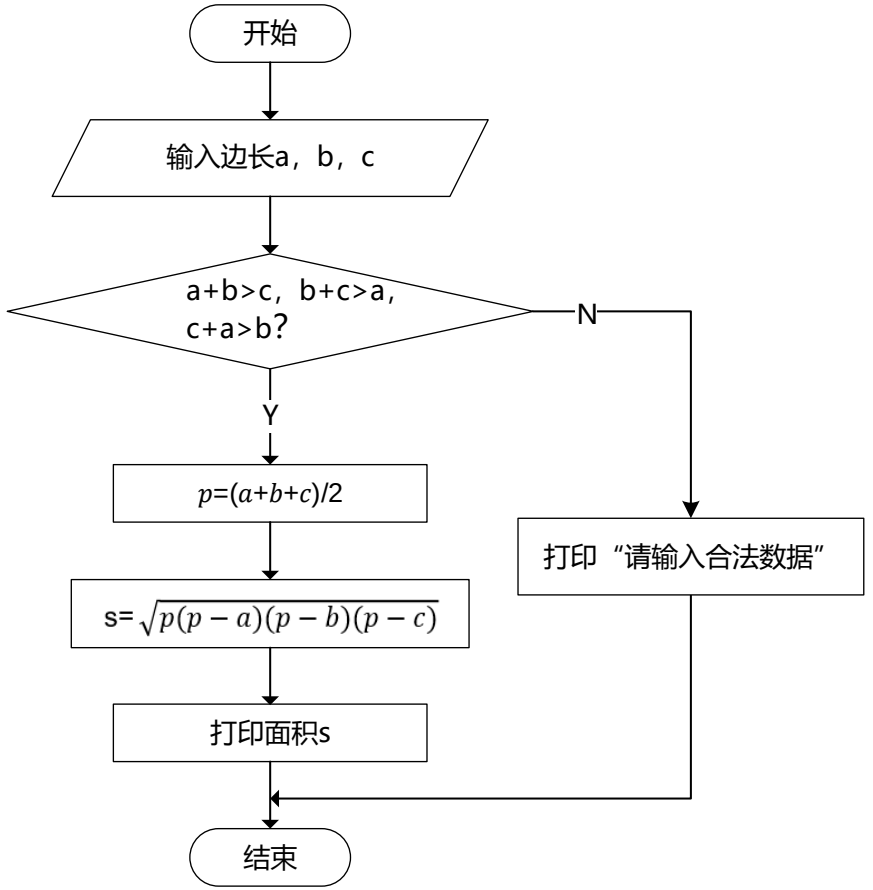


【例5.1】的程序

例5.1-area.py

```
a,b,c = map(float,input().split())

if a+b>c and b+c>a and c+a>b:          #（1）满足构成三角形的条件
    p=(a+b+c)/2
    s=(p*(p-a)*(p-b)*(p-c))**(1/2)
    print('三角形面积s=%.2f' % s)
else:                                  #（2）不满足
    print('请输入合法数据')
```




```
3 4 5
三角形面积s=6.00
>>>
 RESTART: E:\amj\course\Computer\2021\Py
语法\5.1  程序控制结构\if\例5.1-area.py
3 4 9
请输入合法数据
```

- **缩进**（indentation），在Python中有语义含义

条件语句嵌套

- 当某个条件成立时，如果还需要根据另一个条件是否成立来决定执行哪个语句块，可以**嵌套**使用条件语句
- 若True代码块或者False代码块**含有条件语句**
  - 称为**条件语句嵌套**

- **【例5.2】如何判断当前输入整数是否能被2，3整除?**

条件语句嵌套示例：【例5. 2】

- 如何判断当前输入整数是否能被2，3整除?
  - 如果能被2整除
    - ✓ 如果能被3整除
    - ✓ （否则），不能被3整除
  - （否则[不能被2整除]）如果能被3整除
  - （否则），不能被2、3整除
- elif代表“**否则如果**”

```
if x%2 == 0:
    if x%3 == 0:
    else:
elif x%3 == 0:
else:
```

条件语句嵌套

【例5.2】的程序

例5.2-divide by 2 or 3-嵌套.py

条件语句嵌套

```
x = int(input('请输入整数:'))
if x % 2 == 0:
    if x % 3 == 0:
        print('可被2整除，且被3整除')
    else:
        print('可被2整除，不能被3整除')
elif x % 3 ==0:
    print('不可被2整除，但被3整除')
else:
    print('不可被2整除，且不能被3整除')
```

```
>>>
========== RESTART: ====
请输入整数:91
不可被2整除，且不能被3整除
>>>
========== RESTART: ====
请输入整数:99
不可被2整除，但被3整除
>>>
========== RESTART: ====
请输入整数:8
可被2整除，不能被3整除
>>>
========== RESTART: ====
请输入整数:66
可被2整除，且被3整除
>>>
========== RESTART: ====
请输入整数:-33
不可被2整除，但被3整除
>>>
```

复合表达式

- 当条件比较复杂、多于一个时，可以用布尔运算符and、or或not来组合真值，写成复合表达式

例4.1-闰年.py

```
year=int(input('请输入年份：'))

z=(year % 4==0) and (not(year % 100==0)) or (year % 400==0)

if z==True:
    print('%d 是闰年' % year)
else:
    print('%d 不是闰年' % year)
```

复合表达式示例：【例5. 3】

■ **【例5.3】如何求输入三个整数值中的最小值?**

■ **设计思路**

- 获取用户输入的三个整数值：x, y与z
- 如果x < y 且 x < z　　**if x<y and x<z:**
  - ✓ x最小
- （否则），如果y < z　　**elif y<z:**
  - ✓ y最小
- （否则），z最小　　**else:**

复合表达式

【例5.3】的程序

如何求输入三个整数值中的最小值？

- 获取用户输入的三个整数值：x, y与z
- 如果x < y 且 x < z
  - ✓ x最小
- （否则），y < z
  - ✓ y最小
- （否则），z最小

```
x = int(input('请输入整数x: '))
y = int(input('请输入整数y: '))
z = int(input('请输入整数z: '))

if x < y and x < z:
    print(x,'为最小值')
elif y < z:
    print(y,'为最小值')
else:
    print(z,'为最小值')
```

```
>>>
========== RESTART:
======
请输入整数x: 2999
请输入整数y: 0
请输入整数z: -2999
-2999 为最小值
>>>
```

例5.3-if-求最小值-复合表达式.py

条件语句注意事项

- 条件判断
  - **绘制逻辑层次关系图**
    - ✓ 划分变量范围
    - ✓ 每个范围有相应的判断结果
  - **权衡**
    - ✓ 用**if-elif-else**语句
    - ✓ 或**嵌套**条件语句

- **课后练习：改写【例5.3】为嵌套条件语句**

```
if 条件表达式1:
    执行语句1
elif 条件表达式2:
    执行语句2
else:
    执行语句3
```

```
if 条件表达式1:
    执行语句1
else:
    if 条件表达式2:
        执行语句2
    else:
        执行语句3
```

2、迭代控制结构

- **有时候，需要重复执行一些语句多次。如何编写简洁、高效的程序呢?**
- **迭代控制结构（循环结构）：使同一段程序执行多次的一种程序控制结构**
- **循环体：**重复执行的语句序列
- 两种循环（迭代）语句
  - **for语句**
  - **while语句**

（1）for语句

- **如果需要对一个集合（序列或其他可迭代对象）的每个元素都执行同一个代码块，适合采用for语句**
  - **可迭代对象**指可以按次序迭代的对象
  - **利用for语句，可以遍历列表、元组或字典中的每个元素（键）**

```
for 变量 in 序列或其他可迭代对象:
    代码块
```

例：打印列表中每个元素

list_traversal1.py

```
name=['Alice', 'Helen', 'Peter', 'John']
print('name中的所有名字是：')
for each_item in name:
    print(each_item)
```

遍历列表

range函数

- **range函数**：Python内建的**范围函数**，**用于产生某个整数范围内的整数数字**
  - 指定的范围包含下限，但**不包含上限**
  - **下限为0**时，可以**省略**
  - 步长为正整数时，产生的数字从小到大递增；步长为负整数时，产生的数字从大到小递减；**步长为1**时可以**省略**

range *(下限, 上限, 步长)*

- range函数经常与**for语句**结合起来使用，指定循环迭代的范围

```
for i in range(0,n):
    print(i)
```

for语句示例：【例5.4】

【例5.4】计算1+2+3+4+…+n的累加和。

例5.4-for-累加求和.py

```
n=int(input("请输入n：  "))

sum=0                              #累加和
for i in range(1,n+1):             #i为循环变量
    sum=sum+i

print("最终累加和sum=",sum)
```

为什么是n+1?

```
>>>
请输入n：10
最终累加和sum= 55
>>> ==========
>>>
请输入n：100
最终累加和sum= 5050
```

for语句示例：【例5.5】

- **【例5.5】打印如下所示的九九乘法表。**

```
1x1=1
1x2=2   2x2=4
1x3=3   2x3=6   3x3=9
1x4=4   2x4=8   3x4=12  4x4=16
1x5=5   2x5=10  3x5=15  4x5=20  5x5=25
1x6=6   2x6=12  3x6=18  4x6=24  5x6=30  6x6=36
1x7=7   2x7=14  3x7=21  4x7=28  5x7=35  6x7=42  7x7=49
1x8=8   2x8=16  3x8=24  4x8=32  5x8=40  6x8=48  7x8=56  8x8=64
1x9=9   2x9=18  3x9=27  4x9=36  5x9=45  6x9=54  7x9=63  8x9=72  9x9=81
```

- **总结规律**
  - 行号与**乘数**一致：**1~9**
  - 列号与**被乘数**一致：**1~9**

【例5.5】分析

	j=1	j=2	j=3	j=4	j=5	j=6	j=7	j=8	j=9
i=1	1x1=1								
2	1x2=2	2x2=4							
3	1x3=3	2x3=6	3x3=9						
4	1x4=4	2x4=8	3x4=12	4x4=16					
5	1x5=5	2x5=10	3x5=15	4x5=20	5x5=25				
6	1x6=6	2x6=12	3x6=18	4x6=24	5x6=30	6x6=36			
7	1x7=7	2x7=14	3x7=21	4x7=28	5x7=35	6x7=42	7x7=49		
8	1x8=8	2x8=16	3x8=24	4x8=32	5x8=40	6x8=48	7x8=56	8x8=64	
9	1x9=9	2x9=18	3x9=27	4x9=36	5x9=45	6x9=54	7x9=63	8x9=72	9x9=81

- **双重循环**
  - 外循环变量，**乘数**的变化：循环变量**i=1~9**，按**行**打印
  - 内循环变量，**被乘数**的变化：循环变量**j=1~i**，按**列**打印

```
for i in range(1, 10):
    for j in range(1, i+1):
```

【例5.5】程序

```
for i in range(1,10):          #i为乘数，i=1~9
    for j in range(1,i+1):     #j为被乘数，j=1~i
        print("%dx%d=%d" % (j,i,j*i), end='\t')
        #end='\t'表示输出的末尾以Tab键结束
    print()                    #一行结束，换行
```

例5.5-for-九九乘法表.py

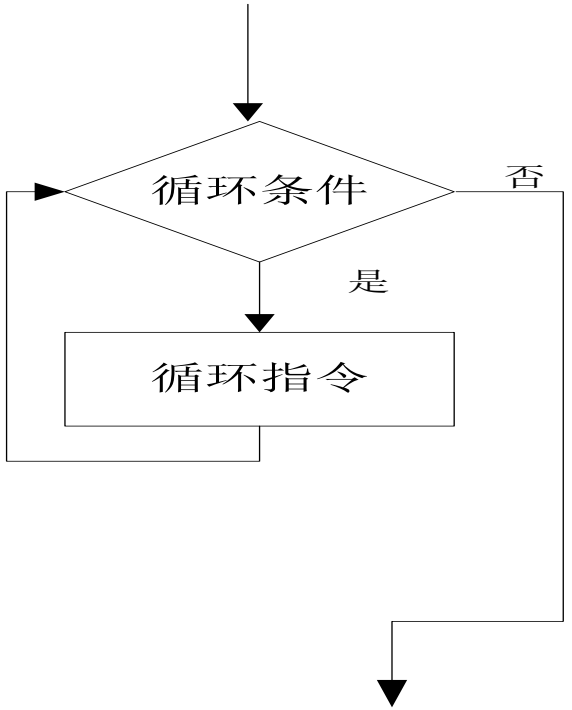
每项“被乘数x乘数=乘积”使用格式化字符串方法完成设置

- 输出**间隔**使用print()的**end**参数来控制，**end='\t'**
- 按行打印，一行内每项的输出没有换行
- **一行结束，使用print()换行**

思考：print语句如果改写成print(“%dx%d=%d” % (i,j,i*j), end= ‘\t’)，符合题目要求吗?

（2）while语句

- **while语句**：有**条件**地执行一条或多条语句
  - **循环条件表达式**为**True**，程序将**执行循环体一次**，之后**再次评估测试**
  - 过程一直重复，**直至**测试条件评值为**False**



while 条件:

code block

- 当**循环次数未知**时，适于使用while语句

while语句注意事项

while 循环条件表达式:

语句序列

改变循环条件表达式的值

……

1.首先判断循环条件表达式是否为真，若不为真，则其后的语句一次也不被执行！

2.在循环体中，**必须有一条改变循环条件表达式的值的语句**！或者**采用break强制退出循环**。否则，循环将无休止地进行下去

while语句示例：【例5.6】

- 【例5.6】猜数字游戏。由玩家通过键盘输入所猜数字
  - 如果玩家猜5，显示“猜对啦！”
  - 如果玩家猜测的数大于5，显示“高了！”
  - 如果猜测的数小于5，显示“低了！”

方法一 使用while语句

- 猜数情况分三类
  - 等于5
  - 小于5
  - 大于5
- 未猜中‘5’时，持续进入游戏
  - answer变量：1表示猜中，0表示未猜中
    - ✓ 初始化： **answer=0**
    - ✓ 进入循环的判断条件：**while(answer==0)**

方法一程序

```
print("欢迎！")
answer = 0
while (answer == 0):
    g = input("猜想数字是:")
    guess = int(g)
    if guess == 5:
        answer = 1
        print("猜对啦！")
    else:
        if guess > 5:
            print("高了！")
        else:
            print("低了！")

print("游戏结束！")
```

循环条件表达式，初始化为0

循环条件判断

用户输入，并类型转换

更改循环条件表达式的值——关键！

必须有此句！否则循环永远不能停止

```
>>>
========== RESTART:
====
欢迎！
猜想数字是:20
高了！
猜想数字是:-20
低了！
猜想数字是:0
低了！
猜想数字是:10
高了！
猜想数字是:5
猜对啦！
游戏结束！
```

方法一

使用while 语句

方法二 使用while True语句

- 猜数情况分三类
  - 等于5
  - 小于5
  - 大于5
- 未猜中‘5’时，持续进入游戏
  - **使用无限循环，while True:**
- 猜中‘5’时， 使用**break**跳出本重循环

方法二程序

```
print("欢迎！")
while True:
    g = input("猜想数字是:")
    guess = int(g)
    if guess == 5:
        print("猜对啦！")
        break
    else:
        if guess > 5:
            print("高了！")
        else:
            print("低了！")

print("游戏结束！")
```

持续进入循环

用户输入，并类型转换

打印输出相应内容

跳出本重循环——关键！

```
====================== RESTART:
===
欢迎！
猜想数字是:21
高了！
猜想数字是:-21
低了！
猜想数字是:0
低了！
猜想数字是:11
高了！
猜想数字是:6
高了！
猜想数字是:3
低了！
猜想数字是:5
猜对啦！
游戏结束！
```

方法二

使用while True语句

思考：哪种方法更简洁?

迭代编程注意事项

- 每迭代一次
  - for中**循环变量的值**
    - ✓ 每次迭代后会**自动更新**
    - ✓ 当超出循环范围，或遇到**break**语句，则终止循环
  - while**循环条件**
    - ✓ 在循环中**显式更新**（如本题中**i = i + 1**）
    - ✓ 或者使用**break**语句，强制终止循环
  - **无限循环**用**while True**实现，需要时用**break**终止循环

循环控制语句：continue和break语句

1、 continue语句

- **终止当前循环**，并忽略continue后面的语句；回到循环顶端，**提前进入下一轮循环**

2、 break语句

- 在while循环和for循环中均可使用
- 一般放在**if选择结构**中，一旦遇到break语句，则立即跳出循环，使得**整个循环提前结束，继续执行循环结构后面的语句**

> - 除非**break语句**让代码更简单或更清晰，否则不要轻易使用

while True语句示例【例5.7】

【例5. 7】 输入正整数n，求1~n之间的全部奇数之和。

- **设计思路**
  - 设x是1~n之间的任意一个整数，初值为0
  - 采用**while True**循环语句
    - ✓ x逐次加1
    - ✓ （1）如果x是偶数，直接进行下一轮循环（**continue**语句）；
    - ✓ （2）如果x大于n，终止循环（**break**语句）；
    - ✓ （3）如果x是奇数，则对x累加求和

【例5. 7】程序

```
#例5.7-while True示例-求1~n的全部奇数之和.py

n = int(input("输入任意一个正整数："))  #输入任意一个正整数

x=0                    #1~n之间的任意一个整数，初值为0
ans=0
while True:            #无限循环
    x += 1             #x逐次加1
    if x%2 == 0:       #（1）如果x是偶数，直接进行下一轮循环
        continue
    elif x > n:        #（2）如果大于n，终止循环
        break          跳出循环
    else:              #（3）如果是奇数，则累加求和
        ans += x

print("ans=",ans)
```

```
输入任意一个正整数：10
ans= 25
>>>
 RESTART: E:\amj\course\Compu
语法\5.1  程序控制结构\while\
输入任意一个正整数：100
ans= 2500
>>>
```

北京航空航天大學
BEIHANG UNIVERSITY

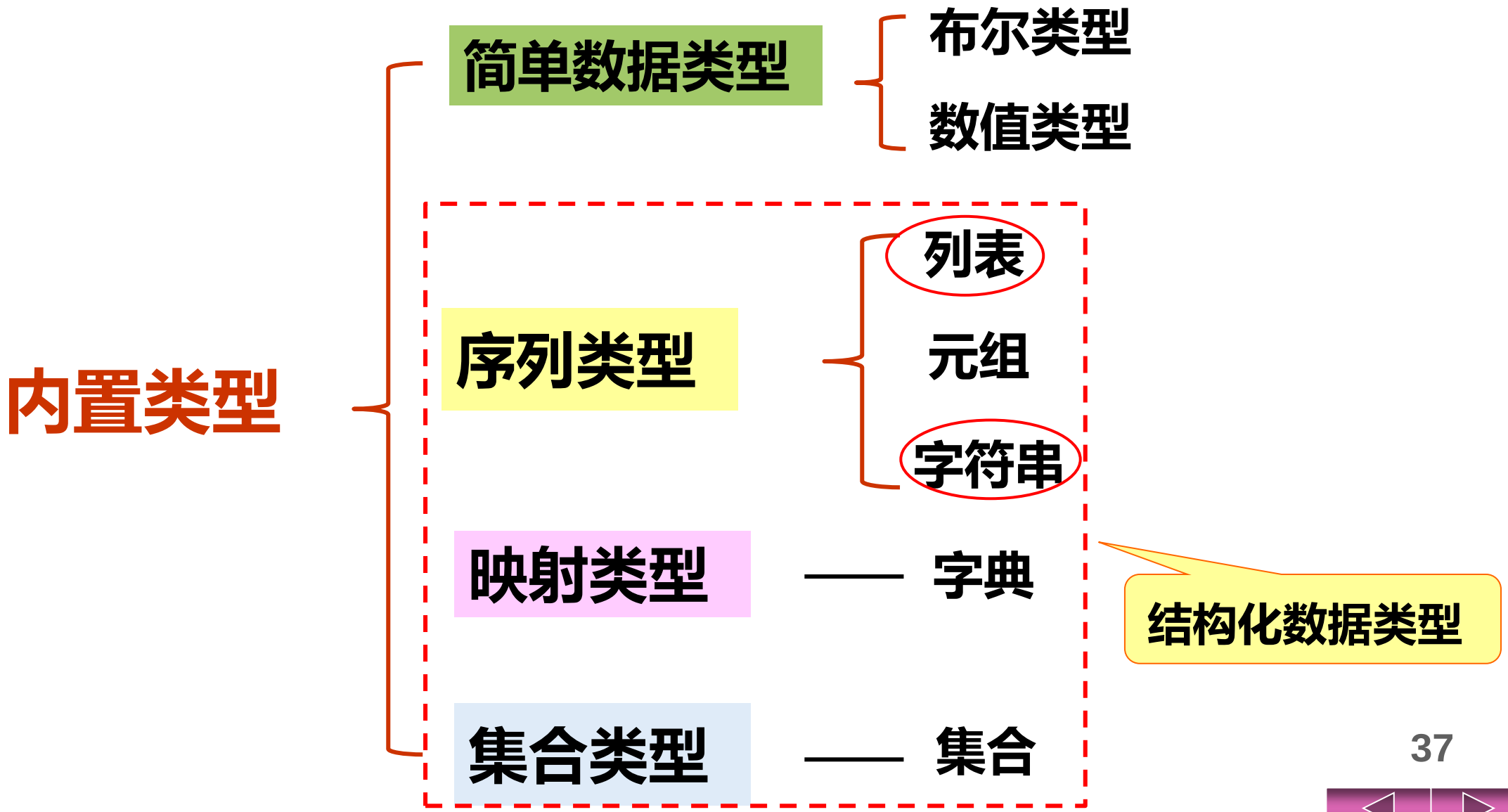
5.2 结构化数据类型简介

北京航空航天大学

5.2.1 结构化数据类型

- **问题**：***少量数据可以用单独的变量名存储，具有相同属性的批量数据如何存储?***
- **Python常用内置类型**：简单数据类型、序列类型、映射类型和集合类型
  - **简单数据类型**包括**布尔类型**和**数值类型**，**没有内部结构**
- **序列类型、映射类型和集合类型属于结构化数据类型**
  - 数据内部由若干分量组成（数据有“**内部结构**”）
  - 数据之间存在特定的**逻辑关系**
- 对于具有**相同属性**的一组数据——可以采用特定的**数据结构**来存储

常用的Python内置类型



数据结构

- **数据结构**：通过某种方式（如对元素进行编号）组织在一起的、具有**相同属性**的数据元素（数字或字符）的**集合**
- Python的数据结构：**序列**（sequence），**映射**（mapping）（**字典** dictionary），**集合**（set）
  - **序列**：**由整数索引的对象的有序集合**。数据成员是有序排列的，可以通过元素的位置访问一个或多个成员元素
  - **索引**：**序列中的每个元素被分配一个序号，即元素的位置**。第一个索引是0，第二个索引是1，依此类推

索引：　0　1　2　3　4

  - 例如列表：[‘a’ , ‘b’ , ‘c’ , ‘d’ , ’ e’]

5.2.2 序列的通用操作

- **序列：由整数索引的对象的有序集合**
- **序列的通用操作**
  - **索引**（indexing）：通过元素编号**访问（获取）**序列中的某个元素
  - **分片**（slicing）：访问序列中的**一定范围（间隔一定步长）内**的元素
  - **加**（adding）：使用加号**连接**两个或两个以上的序列成为一个**新**序列
  - **乘**（multiplying）：将原序列**重复**若干次，连接成一个**新**序列
  - **检查成员资格**：使用**成员资格运算符in**检查一个值是否在某个序列中
- **内建函数**：计算序列**长度 len(x)** 、找出最大元素**max(x)**和最小元素**min(x)**、求和函数**sum(x)……**

（1）索引（indexing）

- **索引**：通过元素编号访问（获取）序列中的某个元素
  - 序列名后跟一对**方括号**，将要访问的元素编号括起来。
  - **正数索引**：**最左边**的元素编号为**0**，从左到右依次为0、1、2、……
  - **负数索引**：从**最右边**（即最后1个元素）开始计数，即最后1个元素的编号为**-1**，倒数第二个元素的编号为-2，……

```
>>> greeting='Hello'
>>> greeting[0]
'H'
>>> greeting[-5]
'H'
```

访问字符串中的最左侧元素

负数索引：访问字符串中的从右至左的第5个元素

（2）分片（slicing）

- **分片**：访问序列中的一定范围内的元素
  - 提取序列的一部分

步长为1时可以省略

格式：**<列表名>[索引1：索引2：步长]**

```
>>> greeting = 'Hey, man!'
>>> greeting[5:8]
'man'
```

greeting[5:7]，则只能提取‘ma’

  - **在普通的分片中，默认步长为1（一般是隐式设置），**返回**索引1**和**（索引2-1）**之间的所有元素

注意：返回的元素不包括第2个索引对应的元素！

副本

```
>>> numbers = [1,4,9]
>>> numbers[:]
[1, 4, 9]
>>> y = numbers[:]
>>> y
[1, 4, 9]
>>> y.pop()
9
>>> y
[1, 4]
>>> numbers
[1, 4, 9]
```

希望访问整个列表，则将两个索引都置空

赋给另一个变量，生成**副本**

改变副本

不会影响原列表

- **提示**：通过**将两个索引都置空，**可以方便地**产生一个原列表的副本**。这时如果对原列表的**副本**进行任何**操作**（如修改元素、删除某元素、排序），**不会影响到原列表**

（3）加（adding）

- **序列相加**
  - **连接**两个或两个以上的序列成为一个**新**序列

原序列不变

```
>>> [1, 2, 3] + [4, 5, 6]
[1, 2, 3, 4, 5, 6]
>>> 'Hello, ' + 'world!'
'Hello, world!'
>>> [1, 2, 3] + 'world!'
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    [1, 2, 3] + 'world!'
TypeError: can only concatenate list (not "str") to list
```

相同类型

只能将列表与列表或者字符串与字符串进行adding操作
无法将字符串add至列表

- **注意**：**必须是相同类型的序列才能进行连接操作。**
  列表和字符串是无法连接在一起的！

（4）乘（multiplying）

- **序列乘法**：用数字x乘以一个序列
  - 将原序列**重复x次**，生成一个新的序列

```
>>> 'hello' * 5
'hellohellohellohellohello'
>>> [37, 38, 39] * 4
[37, 38, 39, 37, 38, 39, 37, 38, 39, 37, 38, 39]
```

- **注意**：**并不是将序列中每个元素乘以x！**

（5）检查成员资格（使用in）

- **检查成员资格**：**使用in运算符检查一个值是否在某个序列中**
  - 如果在，则返回**True**
  - 如果不在，则返回**False**
- **应用**：for语句对列表、字符串、字典的遍历

in_遍历序列.py

```
name=['Alice', 'Helen', 'Peter', 'John']
print('name中的所有名字是：')
for each_item in name:
    print(each_item)

word='hardwork'
print('word中的所有字母是：')
for each_item in word:
    print(each_item)
```

```
name中的所有名字是：
Alice
Helen
Peter
John
word中的所有字母是：
h
a
r
d
w
o
r
k
```

in示例：【例5.8】

- **应用：**用于程序执行某些安全策略的检查

【例5.8】检查用户输入的名字是否在用户列表中。

例5.8-in_example.py

```
name=input('Enter your user name：  ')

users=['Alice', 'Peter', 'Bob', 'John', 'Helen'] # 合法用户
if name in users:
    print('合法用户')
else:
    print('非法用户')
```

```
>>>
Enter your user name：John
合法用户
>>> ================================
>>>
Enter your user name：Mingjing Ai
非法用户
```

北京航空航天大學
BEIHANG UNIVERSITY

5.3 列表

北京航空航天大学

列表

- **列表**（List）：值的有序序列，每个值由索引来识别
  - 用一对**方括号**包裹多个元素，各个元素通过**逗号**分隔
  - 如：[1, 10, 100, 1000]

    ['Alice', 'Helen', 'Peter', 'John']

[0, 1, 2, 3, 4, 5, 6]
[0.0, 1.1, 2.2, 3.3]
[‘a’, ‘b’, ‘c’, ‘d’, ’e’]
[‘a’, 1.1, 2, ‘d’, 3.3]
[[0,1], [1.1, 3], ’c’]

- 是Python中最具灵活性的**有序集合对象**类型。与字符串不同，列表**可以包含任何种类的对象**：数字、字符串、自定义对象甚至其他列表
- 列表是**可变对象**，支持在原处修改，可以通过指定的偏移值和分片、列表方法调用、删除语句等方法实现对列表的改变
  - 序列的**通用操作**：**索引、分片、连接**和**乘法、成员资格检查**

1、列表的主要性质

- **可变长度**、**异构**以及任意**嵌套**
  - 列表可以根据需要增长或缩短（长度可变），并且可以包含任何类型的对象，并支持任意的嵌套
- **可变**序列
  - 列表支持在原处的修改
  - 也可以响应针对**序列**的操作，如索引、切片以及合并
  - 当**合并**或**切片**应用于列表时，返回**新**的列表——**原列表不变**

```
>>> list3=[1,2,3,'a','b']
>>> list3+['add1', 'add2']
[1, 2, 3, 'a', 'b', 'add1', 'add2']
>>> list3
[1, 2, 3, 'a', 'b']
```

2、列表的创建

- **列表的创建**

0 1 2 3 4 5

```
a_list = [ 'a', 'b', 0, 'z', 2020, 'example' ]
```

-6 -5 -4 -3 -2 -1

异构

  - 建立a_list列表，则计算机建立了包含6个元素的可以按照**序号**（**索引**）访问的数据对象
  - 各个元素都有序号（**索引**），两种方式：**正数**索引，**负数**索引
  - **列表的创建，通过[ ]赋予列表对应的值**
  - **[ ]**表示空列表，如：n_list = [ ]
  - **异构**：列表可以包含多种类型的数据（不仅仅是数字）
  - 列表可以**嵌套**，也就是构建多维的列表

二维列表：a_list = [[10, 20, 30]， [40, 50, 60]，[70, 80, 90]]

3、列表的各种操作

- **列表的各种操作**
  - **访问**列表中的元素
  - **增加**元素
  - **删除**元素
  - **修改**元素
  - **查找**元素
  - 与其他数据类型的**转换**
  - 其他**内建函数**

列表的操作（1）：访问列表

- 示例：

  0 1 2 3 4 5

  a_list = ['a', 'b', 0, 'z', 2020, 'example']

  -6 -5 -4 -3 -2 -1

- **列表按照索引序号访问**
  - a_list[0] == a_list[-6]
  - a_list[-1] == a_list[5]
  - a_list[3] == a_list[?]
  - ***如果索引越界会怎么样?*** 比如a_list[6]和a_list[-7]

```
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    a_list[6]
IndexError: list index out of range
```

列表的操作：切片访问多个元素

- 示例：

```
                0   1   2   3      4           5
a_list = [ 'a', 'b', 0, 'z', 2020, 'example' ]
              -6  -5  -4  -3     -2          -1
```

- **切片访问多个元素**
  - 通过指定两个索引值，可以从列表中获取称作“**切片**”的某个部分，返回值是一个**新**列表，按顺序从第一个切片索引开始，截止但不包含第二个切片索引

    a_list[1:3] == ['b', 0]

    不包含a_list[3]

    思考：a_list[0:4:2]=？
  - 如果左切片索引为**零**，可以将其**留空**而将零隐去

    a_list[:3] 与 a_list[0:3] 相同
  - 如果右切片索引为列表的**长度**，也可以将其**留空**

    a_list[3:] 与 a_list[3:6] 相同，因为该列表有六个元素

列表的操作（2）：增加元素

- 四种方法可以给列表**增加新的元素**
  - **方法一**：+连接运算符：将别的列表元素添加到本列表尾部，产生一个**新**的列表（**不改变原列表**）

```
>>> list3=[1,2,3,'a','b']
>>> list3+['add1', 'add2']
[1, 2, 3, 'a', 'b', 'add1', 'add2']
>>> list3
[1, 2, 3, 'a', 'b']
```

  - **方法二**：append(x)方法，将括号中的**对象元素**添加到列表末尾
  - **方法三**：extend(L)方法，将括号中**序列**的**元素**追加到列表末尾
  - **方法四**：insert(i, x)方法，将元素x添加到索引i指定位置

- **注意**：**方法二~方法四均修改了原列表！**

append()方法：在末尾添加一个对象

- **append(x)方法**，将括号中的对象元素（只有**一个**）添加到列表末尾，相当于**a[len(a):] = [x]**
- 示例：
  - list4=[1, 2, 3, 'add1', 'add2']
  - list4.append('app1')　　这时候list4里面是什么？
    [1, 2, 3, 'add1', 'add2', 'app1']
  - list4.append('app1', 'app2')　为什么错误？——**只能有一个参数**
  - list4.append(['app1', 'app2']) list4里面又是什么？

说明：

- append(x)添加一个元素到列表中，['app1', 'app2']则为一个新的元素，不过这个元素是一个**列表**而已

extend()方法在末尾添加多个对象

- **extend(L)方法**，将序列中的**各个**元素逐个**追加**到列表末尾
- 通过添加指定列表的所有元素来扩充列表，相当于**a[len(a):]=L**

修改了原列表

- 示例：
  - list4 =[1, 2, 3, 'add1', 'add2']
  - list4.extend('app1')　　这时候list4里面是什么？

与list4.append('app1') 相同吗?

[1, 2, 3, 'add1', 'add2', 'a', 'p', 'p', '1']

  - list4.extend([0,1,2])　　list4里面又是什么？

[1, 2, 3, 'add1', 'add2', 0, 1, 2]

  - list4.extend('app1', 'app2') 为什么错误？——因为**只能有一个参数**

```
TypeError: extend() takes exactly one argument (2 given)
```

- **说明**：extend(L)将序列中所有元素**逐个**添加到原列表中

insert() 方法

- **insert(i,x) 方法**将单个元素插入到列表中指定位置，第一个参数是列表中将被顶离原位的第一个元素的位置索引
- i是准备插入到其前面的那个元素的索引，例如a.insert(0,x)将x插入到整个列表之前，而a.insert(len(a), x)相当于a.append(x)
- 示例：
  - list4 =[1, 2, 3, 'add1', 'add2']
  - list4.insert(1, 'insert')
  - 则将在原来索引为1的位置，增加新的元素，其余元素顺序后移
    - ✓ [1, 'insert', 2, 3, 'add1', 'add2']

列表的操作（3）：删除元素

- 三种途径**删除元素**
  - **方法一**：使用**del语句**删除列表中的**某个**或**连续几个**元素
  - **注意**：del语句还可以删除其他元素，如字典元素

```
>>> names=['Alice', 'Helen', 'Peter', 'John']
>>> del names[1]      #删除某个元素
>>> names
['Alice', 'Peter', 'John']

>>> del names[0:2]     #删除连续的几个元素
>>> names
['John']

>>> lst = [1, 2, 3]
>>> del lst[0:]          #删除所有元素
>>> lst
[]
```

列表的操作（3）：删除元素（续）

- **方法二**：按照元素值删除： **remove (x)**，移除某个值在列表中的**第一个**匹配项
- **方法三**：使用**pop方法**删除指定位置元素并弹出
  - ✓ pop **([i])** 删除第i个元素并弹出
  - ✓ pop () 删除最后一个元素并弹出
- 示例：
  - ✓ list4=[1,2,3, 'test', 'pop1', 'pop2']
  - ✓ del list4[2] 删除索引为2的元素
  - ✓ list4.remove ('test') 删除元素 'test'
  - ✓ list4.pop(2) 弹出索引为2的元素，返回值为该元素
  - ✓ list4.pop() 弹出最后一个元素

- **注意：pop方法中i两边的方括号表示这个参数是可选的，而不是要求输入一对方括号，会经常在Python库参考手册中遇到这样的标记**

列表的遍历

列表的遍历

要掌握！

遍历：逐一访问列表中的每个元素

- 使用for语句遍历

```
# list_traversal1.py
name=['Alice', 'Helen', 'Peter', 'John']
print('name中的所有名字是：')
for each_item in name:
    print(each_item)
```

循环变量为列表中每个元素

```
name中的所有名字是：
Alice
Helen
Peter
John
>>>
```

4、列表的主要方法

列表方法

要掌握!

- **方法**：是针对对象属性的各种**操作**。是能执行特定功能的程序语句块，即**函数**
- **方法的调用**：**对象.方法(参数)**
- Python为列表定义了多个方法，用于**检查**或**修改**列表中的内容，如：
  - ✓ append
  - ✓ count
  - ✓ extend
  - ✓ index
  - ✓ insert
  - ✓ pop
  - ✓ remove
  - ✓ reverse
  - ✓ sort

列表方法的使用

方法名称	含义	示例	说明

append	在列表末尾追加**一个**新的对象
	.append(<追加对象>)
		>>> prices=[1, 2, 3]
		>>> prices.append(4)
		>>> prices
		[1, 2, 3, 4]
			在恰当位置**修改**原来的**列表**（即列表名不变），而不是返回一个修改过的新列表。
			一次只能追加**一个**对象

count	统计某个元素在列表中出现的次数 **.count(<某元素>)**	>>> ['to', 'be', 'or', 'not', 'to', 'be'].count('to') 2	

extend	在列表的末尾一次性追加另一个序列中的**多个**值
	.extend(<另一个序列>)
		>>> x=[1, 2, 3]
		>>> y= 'abc'
		>>> x.extend(y)　　#在x列表的末尾一次性追加字符串y中的各字符
		>>> x
		[1, 2, 3, 'a', 'b', 'c']
			与连接操作有区别。
			extend**修改**了原来的**列表**，连接操作则不修改原始序列，而返回一个全新的序列。

- **思考：append（附加）与extend（扩展）有何区别?**

列表方法的使用（续1）

方法名称	含义	示例	说明

index	从列表中找出某个值的第一个匹配项的索引位置
	.index(<要匹配的值>)
		>>> greeting=['Good', 'morning', 'everyone', 'she', 'said']
		>>> greeting.index('everyone')
		2
			可用于**查找**某个**元素**在列表中的**位置**

insert	将**一个**对象插入列表中指定位置
	.insert(<索引>,<待插入对象>)
		>>> numbers=[1, 2, 3, 5, 6, 7]
		>>>numbers.insert(3, 'four')
		#在第3个编号位置插入 “four”
		>>> numbers
		[1, 2, 3, 'four', 5, 6, 7]
			一次只能插入一个对象。可以使用**分片赋值**实现插入**多个对象**的操作

```
>>> numbers=[1, 2, 3, 8, 9, 10]
>>> numbers[3:3]= [4,5,6,7]  #在第3个元素之前，插入新的元素
>>> numbers
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

使用分片赋值
实现插入操作

列表方法的使用（续2）

方法名称	含义	示例	说明

pop	移除列表中索引所指的一个元素（**默认为最后一个**），并返回该元素的值
	.pop(<索引>)
		>>> x=[4, 5, 6, 7]
		>>> x.pop()
		#括号中不写参数则移除列表中最后一个元素
		7
		>>> x.pop(2)
		6
			pop方法是唯一一个既能**修改原列表**又能**返回元素值**（除了None）的列表方法

remove	移除某个值在列表中的**第一个**匹配项
	.remove(<要移除的元素>)
		>>> x=['to‘ , 'be', 'or', 'not', 'to', 'be']
		>>> x.remove('be')
		>>> x
		['to', 'or', 'not', 'to', 'be']
			如果列表中有多项都与remove要移除的值相同，也只移除**第一个**匹配项。
			与pop方法不同，remove方法**修改原列表，但不返回值**

- **思考：** pop与remove有何区别?

列表方法的使用（续3）

方法名称	含义	示例	说明

reverse	将列表中的元素**反向**存放
	.reverse()
		>>> x=[4, 5, 9, 8]
		>>> **x.reverse()**
		>>> x
		[8, 9, 5, 4]
			该方法**修改原列表**，但**不返回值**

sort	在原位置对列表进行**排序** **.sort()**按从小到大顺序排序， 或**.sort(key=<参数>)** 或**.sort(reverse=<参数>)** 指明列表是否**反向排序（从大到小）**	>>>x = [4,6,2,1,7,9] >> x.sort() >>> x [1, 2, 4, 6, 7, 9] >>>y= [4,6,2,1,7,9] >>>**y.sort(reverse =True)** >>>y [9,7,6,4,2,1]	该方法**修改原列表**，让其中的元素能按一定的顺序排列。但**不返回值** **高级排序参见扩展与提高内容**

- 如果列表元素是**字符串**，sort方法也可以对其排序
- 如果每个字符串包含不止一个字符，先按最左边字符的ASCII码值大小排序；当最左边的字符相同时，再按第2个字符的ASCII码值大小排序

列表方法示例：【例5.9】

【例5.9】通过键盘一次输入一批数据（一行），每个数据之间间隔一个空格。对这些数据进行逆置，将逆置后数据输出。

【例5.9】设计思路

设计思路

- **通过键盘输入的一行多个数据，如何存储?**
  - ✓ 采用字符串的split()方法，按照空格将输入的多个数据分开，存入一个列表：**s1 = input().split()**
  - ✓ 这里s1为列表，列表中的每个元素都是字符串
- 采用for循环，使用列表的**pop方法**，每次将列表s1中最后一个元素取出
  - ✓ 假定s1=[a_0, a_1, ..., a_{n-1}]
- 在for循环中，再使用列表的**append方法**，将该元素添加入s2列表的末尾
  - ✓ 则s2=[a_{n-1}, ..., a_1,a_0]

【例5.9】程序

例5.9-列表-翻转.py

```
s1 = input().split()
print('最初列表s1中所有元素是：',s1)

s2=[]          #空列表，存储翻转的数据
for i in range(0,len(s1)):
    s2.append(s1.pop())    #每次将s1中最后一个元素添加到s2
```

将其添加至列表s2**末尾**

将列表s1中**最后**一个元素取出

```
>>>
10 20 30 40 80 70 60
最初列表s1中所有元素是： ['10', '20', '30', '40', '80', '70', '60']
最终s2中所有元素是： ['60', '70', '80', '40', '30', '20', '10']
最终s1中所有元素是： []
```

s2与s1反向

sort方法如何保持原列表不变？

- **sort方法**、**reverse方法**用于在原位置对列表进行排序或反向存放，这意味着经过操作，**原列表**被**改变**了
- **但如果用户需要一个排好序或反向存放的列表副本，同时又保留原列表不变，怎么做?**

- **方法一**：先采用分片（调用**x[:]**）**复制整个列表**给另一个变量（y），再对该变量（y）排序，则原列表不变

不能写为y=x

```
x=[4,6,2,1,7,9]          #原始数据

y=x[:]                   #获取x的一个副本，以免修改了x本身
y.sort()                 #对副本进行排序

print('排序后的y: ',y)
print('此时列表x: ',x)
```

```
排序后的y:  [1, 2, 4, 6, 7, 9]
此时列表x:  [4, 6, 2, 1, 7, 9]
```

copy模块的deepcopy函数

- **方法二：采用copy模块的deepcopy函数进行深拷贝**
  - 修改s2，s1并没有随之变化

```
>>> import copy
>>> s1=[1,2,3,7,6]
>>> s2=copy.deepcopy(s1)
>>> s1
[1, 2, 3, 7, 6]
>>> s2.reverse()
>>> s2
[6, 7, 3, 2, 1]
>>> s1
[1, 2, 3, 7, 6]
```

- **或者采用copy模块的copy函数**

```
>>> s1
[1, 2, 3, 7, 6]
>>> s3=copy.copy(s1)
>>> s3.reverse()
>>> s3
[6, 7, 3, 2, 1]
>>> s1
[1, 2, 3, 7, 6]
```

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']
```

详见第6讲中“字符串”

- **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'
```

- **内建函数**：**list**和**str**，与**split**有什么差别？自己试一试？

其他内建函数

- 其他内建函数
  - **list**(seq)：将其他序列类型（如字符串）转换为列表

```
>>> list('abcd')
['a', 'b', 'c', 'd']
```

自行练习

  - **len()**：取得列表元素个数
  - **sum()**：列表求和
  - **max()**：求最大值
  - **min()**：求最小值

技巧1：如何在一个列表中一次删除多个重复元素？

删除列表中同一个元素

```
lis1 = [0,1,'a',2,'a','a',3,4]
print("原始lis1为：",lis1)
lis1 = [x for x in lis1 if x != 'a']    #列表中元素为x，x是lis1中不是a的所有元素
print('删除所有a后，lis1=',lis1)
```

保留元素的条件

```
原始lis1为： [0, 1, 'a', 2, 'a', 'a', 3, 4]
删除所有a后，lis1= [0, 1, 2, 3, 4]
```

删除列表中多个元素

```
del_list = ['a','b']
lis2 = [0,'a',1,2,'b','a',3,4]
print("原始lis2为：",lis2)
lis2 = [x for x in lis2 if x not in del_list]    #列表中元素为x，x是lis2中不包括del_list中任一元素的所有元素
print('删除所有a和b后，lis2=',lis2)
```

保留元素的条件

```
原始lis2为： [0, 'a', 1, 2, 'b', 'a', 3, 4, 'b']
删除所有a和b后，lis2= [0, 1, 2, 3, 4]
```

列表一次删除多个重复元素.py

不要使用for语句和remove方法删除多个重复元素！

- 对于包含若干个字符的列表，**在for语句中使用列表的remove方法删除给定列表中的字符，容易漏删除！**

列表一次删除多个重复元素【error】.py

```
#for语句中使用列表的remove方法删除给定列表中的词，容易漏删除

word1=['a','b','c','d']
del_list = ['a','b']                          #要删除的字符列表
print("原始word1为：",word1)
for i in word1:
    if i in del_list:
        word1.remove(i)                       #word1中的'b'将不会被删除
print("删除'a'和'b'后，word1最终变为：",word1)
```

```
原始word1为:  ['a', 'b', 'c', 'd']
删除'a'和'b'后，word1最终变为:  ['b', 'c', 'd']
```

出错原因分析

出错原因分析

- 错误来源是**Python的迭代器机制**
- 遍历word1
  - ✓ **访问 word1[0]**。由于 **word1[0]= 'a'**，则调用remove方法，将其从word1中**删除**；因为remove方法修改原列表，故**word1中的所有元素会前移一个位置**，即原来的word1[1]变成了word1[0]，原来的word1[2]变成了word1[1]……。
  - ✓ **访问word1[1]**。但其实看的是原列表中的word1[2]。而原列表中的**word1[1]已移到word1[0]的位置**，被**认为已经遍历**过了。则原列表中的**word1 [1]**（即‘b’）就会**漏删除**

- **结论**：**如果原字符串中有两个相邻的停用词，则第2个停用词不会被删除！**

巧妙：从右往左遍历列表删除多个重复元素

- **巧妙**：采用for循环，从右往左遍历，每次判断列表中最后一个元素是否为要删除的元素，若是，则删除。**不会漏删除！**
- range函数当步长为**负数**时，则生成的数字是从大到小

列表一次删除多个重复元素【right】.py

```
word1=['a','b','c','d','a','b','c','d']
del_list = ['a','b']                          #要删除的字符列表
print("原始word1为：",word1)

for i in range(len(word1)-1,-1,-1): #步长取-1，生成的数字是从大到小。因为range函数不包括上限，所以上限取-1，实际上最后i=0
    if word1[i] in del_list:                  #若word1中第i个元素在列表del_list中
        word1.remove(word1[i])                #则从word1中删除

print("删除'a'和'b'后，word1最终变为：",word1)
```

```
原始word1为： ['a', 'b', 'c', 'd', 'a', 'b', 'c', 'd']
删除'a'和'b'后，word1最终变为： ['c', 'd', 'c', 'd']
```

技巧2：列表的初始化

一维列表的初始化

列表初始化.py

```
#一维列表的初始化
lis3=['a']*5                    #方法一：将原序列重复若干次，连接成一个新序列。比方法二更简单
lis4=[0 for i  in range(5)]     #方法二：for语句
```

```
一维列表lis3初始化后:  ['a', 'a', 'a', 'a', 'a']
一维列表lis4初始化后:  [0, 0, 0, 0, 0]
```

二维列表的初始化

```
#二维列表的初始化
lis5=[['a']*3  for i in range(6)]                    #方法一：元素重复三次，形成子列表；子列表重复6次。
lis6=[[0 for i  in range(3)] for j in range(6)] #方法二：双重for循环。每个子列表中共3个元素，有6个子列表
```

```
二维列表lis5初始化后:  [['a', 'a', 'a'], ['a', 'a', 'a'], ['a', 'a', 'a'], ['a', 'a', 'a'],
['a', 'a', 'a'], ['a', 'a', 'a']]
二维列表lis6初始化后:  [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

北京航空航天大學
BEIHANG UNIVERSITY



实验1

北京航空航天大学