《Python语言入门》简介

张永伟

北京大学 软微学院 语言信息工程系

2009年10月29日

北京大学

人生苦短,我用Python!

课程相关信息

- 隶属课程:
  - 《计算机科学技术基础-C》
- 主讲教师:
  - 俞敬松（ **yjs@ss.pku.edu.cn** ）
- 课程助教:
  - 张永伟（**zhangywibb@gmail.com**）
- 课程讲义:
  - **http://www.pkumti.net/**

课程简介

- 课程目标
  - 能使用python语言编写简单的程序
  - 掌握程序设计的基本原理方法
- 课程内容
  - python语言基本语法
  - 正则表达式
  - 程序开发实践

课程简介

- 1. Python语言引论
  - 1.1 python语言历史
  - 1.2 python语言特点
  - 1.3 python开发环境
  - 1.4 第一个python程序

课程简介

- 2. python基本语法
  - 2.1 python文件类型
  - 2.2 python编码规则
  - 2.3 变量和常量
  - 2.4 数据类型
  - 2.5 运算符与表达式

课程简介

- 3. python控制语句
  - 3.1 结构化程序设计
  - 3.2 条件语句
  - 3.3 循环语句
    - 3.3.1 while 循环
    - 3.3.2 for 循环
    - 3.3.3 break和continue语句

课程简介

- 4. 内置数据结构
  - 4.1 元组
  - 4.2 列表
  - 4.3 字典
  - 4.4 序列

课程简介

- 5. 字符串与正则表达式
  - 5.1 字符串操作
  - 5.2 正则表达式
  - 5.3 文件的高级操作

课程简介

- 6. 函数、模块与异常处理
  - python程序的结构
  - 函数
  - 模块
  - 异常处理

课程简介

- 7. 面向对象编程*
  - 面向对象编程
  - 类
  - 类的属性
  - 类的方法
  - 类的继承

课堂组织

- 课堂时间分配
  - 讲解理论知识　　66.67%
  - 程序实践　　33.33%
- 课堂点名
  - 课堂不点名

考核方式

- 考核方法
  - 《Python语言入门》成绩计入《计算机科学技术基础-C》总成绩
  - 开卷、但不能与Ta人有任何形式的交流
  - 只考编程，不考概念

- 成绩计算方法
  - 课堂作业 30%
  - 课后作业 20%
  - 最后一节课考试 50%

教材与参考书

- 教材
  - 孙广磊，征服Python:语言基础与典型应用，人民邮电出版社，2007年09月
  - 丘恩 (Wesley J.Chun)著、宋吉广译，Python核心编程(第2版)，人民邮电出版社，2008年07月
  - Mark Lutz 著 侯靖译，Python学习手册(第3版)，机械工业出版社，2009年08月
  - 周伟 宗杰，Python开发技术详解，机械工业出版社，2009年08月
  - Ben Forta著 杨涛 等译，正则表达式必知必会，人民有点出版社，2007年12月

教材与参考书

- 参考书
  - 《Python 学习笔记》
  - 《Python语言入门》
  - 《Python 从新手到高手》
  - 《OReilly - Python Cookbook》
  - 《Programming Python, 3rd Edition》
  - 《Python Programming for the Absolute Beginner》
  - Python 自带文档

所有参考书在课程主页上均有下载

好好想想，有没有问题?

谢　谢！

北京大学

《Python语言入门》课程 第一讲 Python语言引论

张永伟

北京大学 软微学院 语言信息工程系

2009年10月29日

北京大学

课堂调查

- 举手统计
  - 接触过计算机编程语言？
  - 写过200行代码以内的程序？
  - 写过超过200行代码的程序？

Python语言历史

- python语言起源
  - 在1989年末，Guido van Rossum 为了打发圣诞节的无聊，创造了python。
- python版本
  - 最新版本为3.1.1
  - 课程采用版本2.6.x

Python与其他语言比较

- Python与其他语言比较
  - 比Tcl更强大
  - 比Perl的语法和设计更简单
  - 比Java更简单、更易于使用
  - 比C++更简单、更易于使用
  - 比Visual Basic更强大也具备跨平台特性
  - 比Ruby更成熟、语法更具可读性
  - 具备SmallTalk和Lisp等动态类型的特性，但是对于开发者定制系统的终端用户来说更简单，也更接近传统变成语言的语法。

Python的缺点

- Python的缺点
  - 运行速度不够快
- 开发速度与运行速度之间的矛盾
  - 至今还没有一门编程语言，开发速度比Python快，运行速度比C快

Python语言特点（1）

- python语言特点
  - 面向对象
    - 降低了结构化程序设计的复杂性，使得程序设计更贴近显示生活。
  - 简单
    - 学习简单
    - 使用简单
  - 跨平台
    - 一次编写、到处运行。

Python语言特点（2）

- python语言特点
  - 功能强大
    - 动态类型、自动内存管理、大型程序支持、易于扩展、丰富的内置对象类型、丰富的内置工具、丰富的库工具、丰富的第三方工具
  - 应用广泛
    - 数据库、网络、图形图像、科学计算、web开发、操作系统扩展等

python开发环境

- python版本
  - 2.6.x
  - 主页： http://python.org
- 开发环境(IDE)
  - python IDE列表
    - http://wiki.python.org/moin/IntegratedDevelopmentEnvironments
  - PyScripter
    - 主页： http://code.google.com/p/pyscripter/
  - Ulipad
    - 主页： http://code.google.com/p/ulipad/

第一个python程序（1）

- 搭建运行环境
  - 下载并安装python 2.6.x 软件包
  - 下载并安装PyScripter
- 代码输入
  - 输入第一个python程序代码
- 代码保存
  - 将python程序代码保存为以.py结尾的文件
- 程序运行
  - 按快捷键F11即可以运行编写的python程序

第一个python程序（2）

```
#coding:utf-8
"""
多行注释2
"""
#单行注释

#输出字符串"Hello Python"，字符串用成对的单引号，
#或双引号，或三个单引号，三个单引号扩起来。
print 'Hello Python';          #分号可以省略

#定义两个整型变量，并分别赋值，8，9
a=8
b=9

#条件判断语句，注意缩进。
#缩进是初学者最容易犯的错误之一。
if a>b:
    print "a>b";
else:
    print "a<=b";
```

注意：分号、冒号，引号均为英文符号。

注意：缩进的一致性，一般用Tab符号缩进

第一个python程序（3）

```
#coding:utf-8
#程序的交互,及python中文问题

#提示输入一个字符串
inputString=raw_input("input one String:\t")
#打印输入的字符串

#print “你刚才输入的是: \t” +inputString
print "你刚才输入的是: \t"+inputString.decode('gbk').encode('utf-8')
#print "你刚才输入的是: \t"+inputString.decode('utf-8').encode('gbk')

#提示输入一个整数
inputInt=raw_input("input one Integer:\t")
#打印输入的整数
print inputInt
#计算加法
print inputInt+"+2="+str(int(inputInt)+2)
```

Python中文问题值得注意！

试一试如果输入的不是一个数字会发生什么状况？

课后作业

- 搭建、熟悉python开发环境
  - 下节课开始使用PyScripter开发python程序
- 程序作业
  - 编写一个程序，分别要求用户输入2个人的考试成绩，输出两个人成绩的平均值。
  - 求平均值的公式为 (a+b)/2
  - 测试程序的输出结果与你的预期一样么？如果不一样，能否查阅相关资料，解决这个问题。

好好想想，有没有问题?

谢　谢！

北京大学

《Python语言入门》课程 第三讲 python控制语句

张永伟

北京大学 软微学院 语言信息工程系

2009年11月19日

北京大学

上节课作业

- 作业：编写一个程序，分别要求用户输入2个人的考试成绩，输出两个人成绩的平均值。
- 答案1:

```
#编写一个程序，分别要求用户输入2个人的考试成绩，输出两个人成绩的平均值。

#要求分别输入2个学生的成绩
score1S=raw_input("请输入第一个学生的成绩：");
score2S=raw_input("请输入第二个学生的成绩：");
#分别将两个人的成绩做类型转换，字符串类型变为整型
score1=int(score1S);
score2=int(score2S);
#计算平均值
averageScore=1.0*(score1+score2)/2
print "average score is "+str(averageScore)
```

上节课作业

- 答案2:

```
#编写一个程序，分别要求用户输入2个人的考试成绩，输出两个人成绩的平均值。

#要求分别输入2个学生的成绩
score1=input("请输入第一个学生的成绩: ");
score2=input("请输入第二个学生的成绩: ");
#计算平均值
averageScore=1.0*(score1+score2)/2
#输出结果
print "average score is ",averageScore;
```

input()函数支持用户输入数字或表达式

课程内容

- 课程内容
  - 结构化程序设计
  - 条件语句
  - 循环语句
    - **while** 循环
    - **for** 循环
    - **break**和**continue**语句
  - 文件

结构化程序设计

- 结构化程序设计方法
  - 自顶向下、逐步求精;
  - 把需要的问题分解为若干个小任务来完成，再对每个小任务进行设计，逐步求精;
  - 结构化程序设计是面向对象程序设计的基础。

- 结构化程序设计的三种基本结构
  - 顺序结构
  - 选择结构
  - 循环结构

结构化程序设计的三种基本结构

- 顺序语句
  - 语句处于同一语句块内，程序执行时，所有语句按顺序依次执行
- 条件语句
  - 又称选择语句、判断语句；
  - 条件语句是指根据条件表达式的不同结果，使程序选择执行不同代码块的语句。
- 循环语句
  - 循环语句是指在满足某个条件的情况下，使程序重复执行同一个代码块的语句。

条件语句

- 条件语句基本形式

```
if <条件1>:          #条件为真时，执行缩进的语句块，为假时继续判断elif的条件
        语句块1      #要用缩进来表示语句块处于if语句之中
elif <条件2>:        #当条件为真时，执行缩进的语句，当条件为假时执行else
        语句块2
else:                #前边所有的条件都为假，则执行下面的缩进语句
        语句块3
```

- 条件语句的执行 过程
  - 如果条件1为真，则执行if语句块；否则进入elif语句的条件判断，如果条件2为真则执行语句块2；如果所有条件都不满足，则执行else语句块3。

条件语句注意事项

- 条件语句注意事项
  - else语句与和它最近的if语句配对
  - if，elif，else语句可以嵌套
  - elif语句可以省略
  - else语句也可以省略
- 问题：条件为真时，执行相应语句块。什么情况下条件为真呢？
  - 0，True，[3,4]，[]，5，False，””，“python”，None， 3.0， 0.0？

条件的真假

- 条件的真假
  - 任何非零数字或非空对象都为真;
  - 数字0，空对象以及特殊对象None都被认作是假;
  - 比较和相等测试会返回True或False（1和0的特殊版本）;
  - 布尔and和or运算表达式会返回真或假的操作对象。

条件语句例子

- 例子1：分数及格与否的判断

```
score=60;
#只有if语句
if score<60:        #如果分数小于60则输出"not
pass"
    print "not pass";

#标准的if,else语句
if score<60:        #如果分数小于60则输出"not
pass!"
    print "not pass!";
else:               #否则输出"pass!"
    print "pass!";
```

■ 例子2：分数及格与否的判断

```
score=60;

#复杂一点的if,elif,else语句
if score<60:      #如果分数小于60则输出"not
pass!!"
    print "not pass!!";
#如果分数大于等于60分或小于80分，则输出"pass but not good!!"
elif score>=60 and score <80:
    print "pass but not good!!";
else:                  #否则输出"good!!"
    print "good!!";
```

例子代码：条件语句

```
#coding=gbk
#File: 03-IfElse.py

#定义一个分数
score=60;
#只有if语句
if score<60:     #如果分数小于60则输出"not pass"
    print "not pass";

#标准的if,else语句
if score<60:     #如果分数小于60则输出"not pass!"
    print "not pass!";
else:            #否则输出"pass!"
    print "pass!";

#复杂一点的if,elif,else语句
if score<60:     #如果分数小于60则输出"not pass!!"
    print "not pass!!";
#如果分数大于等于60分或小于80分，则输出"pass but not good!!"
elif score>=60 and score <80:
    print "pass but not good!!";
else:             #否则输出"good!!"
    print "good!!";
```

```
#复杂一点的if,elif,else语句
if score==0:     #如果分数等于0分则输出"0 score!!!"
    print "0 score!!!";
elif score<60:   #如果分数小于60则输出"not pass!!!"
    print "not pass!!!";
#如果分数大于等于60分或小于80分，则输出"pass but not good!!!"
elif score>=60 and score <80:
    print "pass but not good!!!";
else:            #否则输出"good!!!"
    print "good!!!";
```

while循环

- **while循环语句基本形式1**

```
while <条件1>:                    #条件为真时，执行缩进的语句，否则跳过while循环
        语句块1
```

- **while循环的执行 过程**
  - 当循环条件为true时，执行while中的语句块；循环表达式的值false时，跳出while循环。
- 例子3：每次循环去除一个字符，直到字符串为空

```
#每次循环去除一个字符，直到字符串为空
x="python";
while x:
    print x
    x=x[1:]
```

while循环例子

- 例子4：计算1到100的和

```
#计算1到100的和
i=1;
sum=0;
while i<=100:
      sum=sum+i;
      i=i+1;
print "result is: "+str(sum);
```

while循环完整形式

- **while循环语句基本形式2（完整形式）**

```
while <条件1>:                    #当条件为真时，执行缩进的语句
        语句块1
        if <条件2>:
                语句块2
                break;
        if <条件3>:
                语句块3
                continue;
else:
        语句块4
```

- **while循环的执行 过程**
  - 当循环条件为**true**时，执行**while**中的语句块；循环表达式的值**false**时，跳出**while**循环。当**while**循环没有被**break**语句中断时，则跳出循环后会继续执行**else**语句块。

break、continue、pass语句

- **break**
  - 跳出最近的整个循环（跳出整个循环语句）
- **continue**
  - 跳出本次循环（后面的代码）（继续执行下一个循环）
- **pass**
  - 空占位语句，不做任何事情
- **break语句和continue语句的区别**
  - **break语句跳出整个循环;**
  - 而**continue**语句只是跳出当前循环（**continue**后面的语句将不再执行），会继续执行下一次循环。

while循环例子

- 例子5：打印所有小于10及其大于或等于0的所有偶数

```
#打印所有小于10及其大于或等于0的偶数

i=10;     #定义一个整型变量，并赋值10
#执行while循环，只有当i==0时才会跳出这个循环
while i:
      i=i-1;          #每执行依次循环体都将i值减1
      if i%2!=0:      #如果不是偶数，则跳过本次循环
                      #（即不执行continue后面的语句）
          continue;
      print i,        #print语句以,结尾则不会自动换行
```

0据说既不是基数也不是偶数？

while循环例子

- 例子6：打印所有小于10及其大于或等于0的最大偶数

```
#打印所有小于10及其大于或等于0的最大偶数

i=10;     #定义一个整型变量，并赋值10
#执行while循环，只有当i==0时才会跳出这个循环
while  i:
      i=i-1;          #每执行依次循环体都将i值减1
      if  i%2==0:     #如果是偶数，则输出这个数，
                      #并跳出整个循环
          print  i;
          break;    #跳出整个循环
```

while循环例子

- 例子7：判断一个数是否为质数

```
#判断一个数是否为质数
sy=raw_input("请输入一个正数: ");      #输入一个正数
y=int(sy);                             #类型转换
x=y/2;                                 #因子变量
while x>1:
    if y%x==0:                         #如果y可以被某个因子整除则不是质数
        print y, 'has factor', x;
        break;
    x=x-1;
else:
    print y, 'is prime';               #如果y没有被某个因子整除，则是质数
```

break用法：只要y可以被某个因子整除，则说明y即不是质数，此时就可以跳出程序了。

else用法：如果整个过程中没有被可以整除的因子，也即没有被break语句中断循环，则会执行else语句里面的内容。

for循环

- **for循环语句基本形式**

```
for <目标对象> in <对象集合>:
        if <条件1>:
                语句块1
                break;    #终止循环
        if <条件2>:
                语句块2
                continue;          #使用continue跳过其它语句，继续循环
        语句块3
else:                              #如果for循环未被break终止则执行else块中的语句
        语句块4
```

- **for循环的执行 过程**
  - 当执行**for**循环时，会逐个将对象集合中的元素赋给目标对象，然后为每个元素执行循环体。如果离开循环体前没有执行过**break**语句（即对象集合中的每个元素都被访问过了），则会执行**else**语句。

for循环

- 对象集合都可以有哪些？
  - 字符串、列表、元组、其它内置可迭代对象以及我们自己通过类所创建的新对象。

- 例子8：依次输出列表中的每个元素

```
#依次输出列表中的每个元素
aList=['apple','orange','banana'];   #定义一个列表
#每次循环都将列表中的一个元素赋值给目标对象fruit
for fruit in aList:
    print fruit;                     #输出目标对象
```

for循环例子

- 例子9：计算列表中所有元素的和

```
#计算列表中所有元素的和
sum=0;
for x in [1,2,3,4,5]:
    sum=sum+x;
print "sum="+str(sum);
```

for循环例子

- 例子10：分别输出一个字符串内每一个字符

```
#分别输出一个字符串内每一个字符
s="python"
for c in s:
    print c,
```

range()函数

- for循环经常和range()函数一起使用
- range()函数语法
  - range([start,]stop[,step])
  - range()函数返回值为一个整数列表，列表的元素值由range的三个参数决定
    - start：可选参数，表示列表起始值，默认为0
    - stop：必须参数，表示列表终止值
    - step：可选参数，表示步长，即列表中元素递增的数值，默认值为1

range()函数例子

- **range()函数例子**

```
>>> range(10)    #默认起始值为0，步长为1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(3,8)   #默认步长为1
[3, 4, 5, 6, 7]
>>> range(3,8,1)
[3, 4, 5, 6, 7]
>>> range(3,8,2)#指定步长为2
[3, 5, 7]
>>> range(5,-5,-1)
[5, 4, 3, 2, 1, 0, -1, -2, -3, -4]
>>> range(5,-5)
[]
```

for循环例子

- 例子11：依次输出列表中的每个元素

```
#依次输出列表中的每个元素
aList=['apple','orange','banana'];   #定义一个列表
#每次循环都将列表中的一个元素赋值给目标对象fruit
for x in range(len(aList)):
    print aList[x];                  #输出目标对象
```

for循环例子

- 例子12：在0～99的数中查找用户输入的值

```
#在0～99的数中查找用户输入的值
sx=raw_input("输入x的值");
x=int(sx);
for y in range(0,100):
    if x==y:
        print "find the number "+str(x);
        break;
else:
    print "did't find the number"
```

文件（1）

- 文件
  - 打开一个文件的函数为内置函数open()。
- 文件语法
  - file_object=open(filename，mode='r'，buffersize=-1)
  - 其中：
    - filename：要打开的文件的名字，可以是相对路径也可以是绝对路径
    - mode：可选参数，文件打开的模式
    - buffersize：可选参数，缓冲区的大小

文件（2）

- 文件打开模式

文件模式	含义
r	以读方式打开
w	以写方式打开
+	以读写方式打开
a	以再文件末尾追加方式打开
b	以二进制方式打开

- 模式的组合
  - 不同模式之间可以组合，例如’**wb+**’表示以二进制读写模式打开；’**r+**’表示以读写模式打开。

文件（3）

- 常用文件操作

input=open("input.txt","r")	创建输入文件
output=open("outpout.txt","w")	创建输出文件
aString=input.read()	把整个文件读入单一字符串
aString=input.read(N)	读取之后的N个字节到一个字符串
aString=input.readline()	读取下一行（包括行末标志符）到一个字符串
aList=input.readlines()	读取整个文件到字符串列表
output.write(aString)	写入字节字符串到文件
output.writelines(aList)	把列表内所有字符串写入文件
output.close()	关闭文件
output.flush()	把输出缓冲区刷到硬盘中，但不关闭文件

例子代码：文件

```
#coding=gbk
#File:03-File.py

#拷贝文件,定义一个函数，该函数名字为copyfile
def copyfile():
    #以只读方式打开文件”1.txt “
    inputFile=open("1.txt","r");
    #以只写方式打开文件”2.txt “
    outputFile=open("2.txt","w");

    #读取整个文件到字符串列表
    lineList=inputFile.readlines();

    #遍历所有字符串列表
    for line in lineList:
        outputFile.write(line); #将每一行写入到第二个文件中

    #关闭文件
    inputFile.close();  #关闭第一个文件
    outputFile.close(); #关闭第二个文件

#调用自定义的函数copyfile
copyfile();
```

课后作业

- 程序作业
  - 编写一个程序，读入一个文件（**normal.txt**）中的所有内容，将文件中的所有字符均改为大写，并另存为另一个文件（**upper.txt**）
  - 将一个字符串**s**的所有字符全部大写的函数为**s.uper()**；其中**s**为待转换的字符串。

好好想想，有没有问题?

谢　谢！

北京大学

《Python语言入门》课程 第四讲
内置数据结构

张永伟

北京大学 软微学院 语言信息工程系

2009年12月1日

北京大学

课程内容

- 课程内容
  - 序列
  - 列表
  - 元组
  - 字典

序列

- 序列是具有索引和切片能力的集合。
- 列表、元组和字符串具有通过索引访问某个具体的值，或通过切片返回一段切片的能力。
- 列表、元组、字符串都属于序列。

列表

- 列表含义
  - 列表（List）是Python中非常重要的内置数据类型。列表由一系列元素组成，所有的元组被包含在一对方括号中。列表被创建将后，可以执行添加、删除、修改操作。
  - 列表中可包含任意的Python数据信息，如字符串、数字、列表、元组等。
- 列表的几个例子:
  - list = [“a”, “b”, “c”]，定义字符列表。
  - list = [1, 2, 3, 4]，定义数字列表。
  - list = [[1,2,3,4], ["a","b","c"]]，定义列表的列表。
  - list = [(1,2,3,4), ("a","b","c")]，定义元组列表。
  - list((1,2))把一个元组转换成一个列表[1,2]，list('test')可把字符串转换成['t','e','s','t']列表。

列表

- 列表须知
  - 列表内元素的个数及元素的值可以改变;
  - 列表内元素用中括号（[]）包裹;
  - 列表内不同元素之间采用逗号（,）分隔;
  - 列表内可以包含任何数据类型，也可以包括另一个列表;
  - 列表可以通过序号来访问其中的成员;
  - 可以对列表进行插入、删除、排序，修改列表中某元素等操作。

列表例子（1）

```
#创建一个列表，里面有4个元素，字符串类型
list = ["apple", "banana", "grape", "orange"]
#输出这个列表
print list
#输出列表第2个元素（从0开始计数）
print list[2]
#向列表中的末尾添加新的元素
list.append("watermelon")
#向列表的第1的位置（从0开始计数）加入新元素
list.insert(1, "grapefruit")
print list
#删除列表中第一个"grape"
list.remove("grape")
print list
#list.remove("a")
#删除列表最后一个元素，并将该元素返回
print list.pop()
print list
```

列表例子（2）

```
#定义一个列表
list = ["apple", "banana", "grape", "orange"]
#索引，输出列表右边数第2个元素（从0开始计数）
print list[-2]
#切片，输出列表第1个到第3个元素（包含第1个元素但不包含第3个元素）
print list[1:3]
#切片，输出列表右边数第3个元素到右边第1个元素，且不包含右边第1个元素
print list[-3:-1]
#定义一个列表，这个列表的元素仍然是列表，相当于二维数组
list = [["apple", "banana"],["grape", "orange"],["watermelon"],["grapefruit"]]
#遍历上面的列表，输出列表内元素
for i in range(len(list)):
    for j in range(len(list[i])):
        print list[i][j], "" ,
    print
```

列表例子（3）

```
list = ["apple", "banana", "grape", "orange"]
#查找"grape"元素在列表中的位置（有多个则只返回第一个）
print list.index("grape")
print list.index("orange")
#判断元素"orange" 是否在list中
print "orange" in list

list1 = ["apple", "banana"]
list2 = ["grape", "orange"]
#列表的追加
list1.extend(list2)
print list1
list3 = ["watermelon"]
#列表的合并
list1 = list1 + list3
print list1
list1 += ["grapefruit"]
print list1
#列表的重复
list1 = ["apple", "banana"] * 2
print list1
```

列表例子（4）

```
#使用列表的sort方法排序
list = ["banana", "apple", "orange", "grape"]
#列表的排序
list.sort()
print "Sorted list:", list
#列表的反转
list.reverse()
print "Reversed list:", list

#使用函数sorted排序,返回一个新的列表
list = ["banana", "apple", "orange", "grape"]
for li in sorted(set(list)):
    print li, "" ,
```

列表所有操作列表

	创建一个列表	
1	L1 = []	建立一个空的列表
2	L2 = [0, 1, 2, 3]	建立一个列表，含有4个元素
3	L3 = ['abc', ['def', 'ghi'], 3.0]	建立一个嵌套的子列表，列表中元素类型不同。
4	L4 = list('zyw')	通过list()工厂方法来建立列表：['z', 'y', 'w ']
	访问列表中的值	
5	L2[i]	索引，访问列表L2第i个元素(列表下标从0开始计数)
6	L2[i:j]	切片，返回含下标从i到j的元素的子列表（包括L2[i],但不包括L2[j]）
7	L3[i][j]	索引的索引，L3[1][1]为'ghi'
	添加元素或列表	
8	L2.append(4)	向列表L2的末尾增加一个元素4
9	L2.insert(i, X)	在列表L2中索引值为i的位置添加元素x
10	L2.extend([5,6,7])	把列表[5,6,7]中的内容添加到列表L2中
11	L1 + L2	合并列表L1和L2,得到一个新的列表
12	L2 * 2	列表的复制，得到包含2个列表L2内元素组成的列表：[0,1,2,3,0,1,2,3]
	删除元素或列表	
13	L2.remove(2)	在列表L2中删除元素2，如果列表中有多个元素2，则只删除第1个元素2
14	L2.pop( )	在列表L2中删除最后一个元素并返回该元素的值
15	L2.pop(i)	在列表L2中删除下标为i的元素并返回该元素的值
16	del L2[i]	删除列表L2中下标为i的元素
17	del L2[i:j]	删除列表L2中下标从i到j的元素（包括L2[i],但不包括L2[j]）
18	del L2	删除列表L2
19	L2[i:j] = []	删除列表L2中下标从i到j的元素（包括L2[i],但不包括L2[j]）
	搜索列表中的元素	
20	L2.index(x)	搜索元素x是否在列表L2中,返回首个x所在的下标值，没搜索到则抛出异常
21	L2.index(x,i)	搜索元素x是否在列表L2[i:]中，返回首个x所在的下标值，没搜索到则抛出异常
22	L2.index(x,i,j)	搜索元素x是否在列表L2[i:j]中，返回首个x所在的下标值，没搜索到则抛出异常
23	3 in L2	搜索元素3是否在列表L2中，搜索到返回True，没有搜索到则返回False。
24	3 not in L2	搜索元素3是否不在列表L2中，没有搜索到则返回True，搜索到则返回False。
	修改列表元素值	
25	L2[i] = 1	将列表L2的第i个元素修改为1。
26	L2[i:j] = L3	列表L2中下标从i到j的元素（包括L2[i],但不包括L2[j]）替换为新列表L3的内容。
	列表的其他操作	
27	len(L2)	求列表L2的长度，L2长度为4
28	max(L2)	求列表L2中所有元素“最大值”
29	min(L2)	求列表L2中所有元素“最小值”
30	sorted(L2)	得到一个新的列表，新列表内元素依次为列表L2中元素顺序排序后得到的元素序列
31	L2.sort()	对列表L2中元素进行顺序排序
32	L2.sort(reverse=True)	对列表L2中元素进行逆序排序
33	L2.reverse( )	原地反转列表L2
34	L2.count(x)	返回元素x在列表L2中出现的次数
35	tuple(L2)	将列表L2转换为一个元组，并返回该元组
36	list(S)	将一个序列S转换为一个列表，并返回该列表

索引

- 索引(S[i])获取特定偏移的元素
  - 第一个元素的偏移为0
  - 负偏移意味着从最后或右边向反方向进行计数
    - S(0)获取了第一个元素
    - S[-2]获取了倒数第二个元素（就像S(len(S)-2)一样）

索引例子

```
#索引例子
tuple = ("apple", "banana", "grape",
"orange")
list = ["apple", "banana", "grape", "orange"]
str = "apple"
print tuple[0]
print tuple[-1]
print list[0]
print list[-1]
print str[0]
print str[-1]
```

切片

- 切片S[i:j]提取对应的部分作为一个序列:
  - 上边界并不包含在内;
  - 如果没有给出分片的边界，分片的边界默认的下边界为0，上边界为序列的长度;
    - S[1:3]获取了从偏移为1的元素，直到但不包括偏移为3的元素
    - S[1:]获取了从偏移为1的元素，直到末尾（偏移为序列长度）之间的元素
    - S[:3]获取了从偏移为0的元素，直到但不包括偏移为3的元素
    - S[:-1]获取了从偏移为0的元素，直到但不包括最后一个元素之间的元素
    - S[:]获取了从偏移为0到末尾之间的元素，这是有效实现列表拷贝的一种方法（得到的列表为新的列表）。
- 扩展的切片S[i:j:k]，其中i，j含义同上，k为递增步长。

切片例子

```
#分片例子
tuple = ("apple", "banana", "grape", "orange")
list = ["apple", "banana", "grape", "orange"]
str = "apple"
print tuple[:3]
print tuple[3:]
print tuple[1:-1]
print tuple[:]
print list[:3]
print list[3:]
print list[1:-1]
print list[:]
print str[:3]
print str[3:]
print str[1:-1]
print str[:]
```

列表例子（3）

```
>>> L = ['spam', 'Spam', 'SPAM!']
>>> L[1] =  'eggs'  # 列表元素的修改
>>> L
['spam', 'eggs', 'SPAM!']
>>> L[0:2] = ['eat', 'more'] #
>>> L # Replaces iSlice assignment:
delete+inserttems 0,1
['eat', 'more', 'SPAM!']
```

元组

- 元组与列表类似，与列表不同的是，元组中的元素一旦确立就不能被改变。
  - 元组可以使用在不希望数据被其他操作改变的场合。

元组例子

```
#定义一个元组，里面有4个元素
tuple = ("apple", "banana", "grape", "orange")
#索引
print tuple[-1]
print tuple[-2]
#切片
tuple2 = tuple[1:3]
tuple3 = tuple[0:-2]
tuple4 = tuple[2:-2]
#元组切片的返回值仍然是元组
print tuple2
print tuple3
print tuple4
#元组元素不可以修改，否则会报错
tuple[0]='error'
```

元组例子

```
#coding=gbk
#File:04-Tuple.py

#创建一个元组
T1 = ()      #建立一个空的元组
T2 = (0, 1, 2, 3)        #建立一个元组，含有4个元素
T3 = ('abc', ('def', 'ghi'), 3.0)    #建立一个嵌套的子元组，元组中元素类型不同。
T4 = tuple('zyw')        #通过tuple()工厂方法来建立元组: ('z', 'y', 'w ')


#访问元组中的值
print T2[1] #索引，访问元组T2第1个元素(元组下标从0开始计数)
print T2[2:3]            #切片，返回含下标从2到3的元素的子元组（包括T2[2],但不包括T2[3]）
print T3[1][1]           #索引的索引，T3[1][1]为'ghi'
#添加元素或元组

print T1 + T2            #合并元组T1和T2,得到一个新的元组
print T2 * 2             #元组的复制，得到包含2个元组T2内元素组成的新元组:  (0,1,2,3,0,1,2,3)

#搜索元组中的元素
print T2.index(2)        #搜索元素2是否在元组T2中,返回首个2所在的下标值，没搜索到则抛出异常
print T2.index(3,2)      #搜索元素3是否在元组T2[2:]中，返回首个3所在的下标值，没搜索到则抛出异常
print T2.index(3,2,4)    #搜索元素x是否在元组T2[2:4]中，返回首个3所在的下标值，没搜索到则抛出异常
print 3 in T2            #搜索元素3是否在元组T2中，搜索到返回True，没有搜索到则返回False。
print 3 not in T2        #搜索元素3是否不在元组T2中，没有搜索到则返回True，搜索到则返回False。

#元组的其他操作
print len(T2)            #求元组T2的长度，T2长度为4
print max(T2)            #求元组T2中所有元素“最大值”
print min(T2)            #求元组T2中所有元素“最小值”
print sorted(T2)         #得到一个新的元组，新元组内元素依次为元组T2中元素顺序排序后得到的元素序列
print T2.count(2)        #返回元素x在元组T2中出现的次数
print tuple('zyw')       #将元组T2转换为一个元组，并返回该元组

#删除元组
del T2
print T2
```

元组所有操作列表

	创建一个元组	
1	T1 = ()	建立一个空的元组
2	T2 = (0, 1, 2, 3)	建立一个元组，含有4个元素
3	T3 = ('abc', ('def', 'ghi'), 3.0)	建立一个嵌套的子元组，元组中元素类型不同。
4	T4 = tuple('zyw')	通过tuple()工厂方法来建立元组：('z', 'y', 'w ')
	访问元组中的值	
5	T2[i]	索引，访问元组T2第i个元素(元组下标从0开始计数)
6	T2[i:j]	切片，返回含下标从i到j的元素的子元组（包括T2[i],但不包括T2[j]）
7	T3[i][j]	索引的索引，T3[1][1]为'ghi'
8	T1 + T2	合并元组T1和T2,得到一个新的元组
9	T2 * 2	元组的复制，得到包含2个元组T2内元素组成的新元组： (0,1,2,3,0,1,2,3)
	搜索元组中的元素	
10	T2.index(x)	搜索元素x是否在元组T2中,返回首个x所在的下标值，没搜索到则抛出异常
11	T2.index(x,i)	搜索元素x是否在元组T2[i:]中，返回首个x所在的下标值，没搜索到则抛出异常
12	T2.index(x,i,j)	搜索元素x是否在元组T2[i:j]中，返回首个x所在的下标值，没搜索到则抛出异常
13	3 in T2	搜索元素3是否在元组T2中，搜索到返回True，没有搜索到则返回False。
14	3 not in T2	搜索元素3是否不在元组T2中，没有搜索到则返回True，搜索到则返回False。
	元组的其他操作	
15	len(T2)	求元组T2的长度，T2长度为4
16	max(T2)	求元组T2中所有元素“最大值”
17	min(T2)	求元组T2中所有元素“最小值”
18	sorted(T2)	得到一个新的列表，新列表内元素依次为元组T2中元素顺序排序后得到的元素序列
19	T2.sort(reverse=True)	对元组T2中元素进行逆序排序
20	T2.count(x)	返回元素x在元组T2中出现的次数
21	tuple(S)	将一个序列S转换为一个元组，并返回该元组
	删除元素或元组	
22	del T2	删除元组T2

字典

- 字典（Dictionary）是由“键-值”对组成的集合，字典中的“值”通过“键”来引用。
  - Python中，键-值对之间用“英文逗号”隔开，并且被包含在一对花括号中。
  - 字典与列表最大的不同是，字典是无需的，字典中的元素是通过键来访问的。
  - 字典也是可变的，可以包含任何其他类型，字典中没有位置的概念。
    - 例如“取字典的第一个元素？”
      - 说法错误，因为字典没有位置概念。

字典例子（1）

```
#使用字母作为索引
dict = {"a" : "apple", "b" : "banana", "g" : "grape", "o" :
"orange"}
print dict
print dict["a"]
#使用数字作为索引
dict = {1 : "apple", 2 : "banana", 3 : "grape", 4 : "orange"}
print dict
print dict[2]
#字典的添加、删除
dict = {1 : "apple", 2 : "banana", 3 : "grape", 4 : "orange"}
dict[5]="organe" #添加（元素不存在）或修改元素（元素存在）
del dict[1] #删除元素（1,"apple"）
print dict
print dict[2]
#使用元组作为索引
dict = {}
dict[("a","p","p","l","e")] = "apple"
dict[("b","a","n","a","n","a")] = "banana"
print dict
print dict[("a","p","p","l","e")]
```

字典例子（2）

```
#字典的添加、删除、修改操作
dict = {"a" : "apple", "b" : "banana", "g" : "grape", "o" : "orange"}
dict["w"] = "watermelon"
del(dict["a"])
dict["g"] = "grapefruit"
print dict.pop("b")
print dict
dict.clear()
print dict

#字典的遍历
dict = {"a" : "apple", "b" : "banana", "g" : "grape", "o" : "orange"}
for k in dict:
    print k,dict[k]

#字典items()的使用
dict = {"a" : "apple", "b" : "banana", "c" : "grape", "d" : "orange"}
#每个元素是一个key和value组成的元组，以列表的方式输出
print dict.items()

#调用items()实现字典的遍历
dict = {"a" : "apple", "b" : "banana", "g" : "grape", "o" : "orange"}
for (k, v) in dict.items():
    print "dict[%s] =" % k, v
```

字典例子（3）

```
dict = {"a" : "apple", "b" : "banana", "c" : "grape", "d" : "orange"}
#输出key的列表
print dict.keys()
#输出value的列表
print dict.values()
#每个元素是一个key和value组成的元组，以列表的方式输出
print dict.items()

dict = {"a" : "apple", "b" : "banana", "c" : "grape", "d" : "orange"}
it = dict.iteritems()
print it

#字典中元素的获取方法
dict = {"a" : "apple", "b" : "banana", "c" : "grape", "d" : "orange"}
print dict
print dict.get("c", "apple")
print dict.get("e", "apple")
#get()的等价语句
D = {"key1" : "value1", "key2" : "value2"}
if "key1" in D:
    print D["key1"]
else:
    print "None"
```

字典例子（4）

```
#字典的更新
dict = {"a" : "apple", "b" : "banana"}
print dict
dict2 = {"c" : "grape", "d" : "orange"}
dict.update(dict2)
print dict
#udpate()的等价语句
D = {"key1" : "value1", "key2" : "value2"}
E = {"key3" : "value3", "key4" : "value4"}
for k in E:
    D[k] = E[k]
print D
#字典E中含有字典D中的key
D = {"key1" : "value1", "key2" : "value2"}
E = {"key2" : "value3", "key4" : "value4"}
for k in E:
    D[k] = E[k]
print D

#设置默认值
dict = {}
dict.setdefault("a")
print dict
dict["a"] = "apple"
dict.setdefault("a","default")
print dict
```

字典所有操作列表

创建一个字典		
1	D1 = {}	建立字典D1，该字典为空
2	D2 = {'spam': 2, 'eggs': 3}	建立字典D2，里面有两个元素
3	D3 = {'food': {'ham': 1, 'egg': 2}}	建立字典D3，字典内元素还可以包含字典（字典的嵌套）
4	D4=dict((['x',1],['y',2]))	通过dict()工厂方法来建立字典：{'y': 2, 'x': 1}
5	D5={}.fromkeys(('x','y'),1)	通过fromkeys()内置函数来建立字典D5，字典所有元素具有相同值
6	D6 = dict(name='Bob', age=42)	通过dict()工厂方法来建立字典:{'age': 42, 'name': 'Bob'}
访问字典中的值		
7	D2['eggs']	索引，访问键为'eggs'的值,如果不存在，则会抛出错误
8	D3['food']['ham']	索引的索引，访问键为'food'的值（为另一个字典）的键为'ham'的值
9	D2.get('zyw')	访问字典D2中键为'zyw'的值，如果不存在则返回None
10	D2.get('zyw',4)	访问字典D2中键为'zyw'的值，如果不存在则返回指定的值4
添加元素或字典		
11	D2['zyw']=6	向字典D2中增加键-值对('zyw',6)，前提是键'zyw'不存在
12	D2.update(D1)	将字典D1中的键-值对全部添加到字典D2中
13	D2.setdefault('zyw',4)	向字典D2中添加键-值对('zyw',4)，若'zyw'存在，不进行任何操作
删除元素或字典		
14	D2.pop('spam')	在字典D2中删除键为'spam'的键-值对，并返回值，不存在则报错
15	D2.pop('zyw',4)	在字典D2中删除键为'spam'的键-值对，并返回值，不存在返回4
16	D2.clear()	删除字典D2中所有键-值对
17	del D2['spam']	删除字典D2中键为'spam'的键-值对
18	del D2	删除字典D2
搜索字典中的元素		
19	D2.has_key('spam')	搜索'spam'是否是字典D2中的一个键，是返回True，否则返回False
20	'spam' in D2	搜索'spam'是否为字典D2中的一个键，搜索到返回True，没有搜索到则返回False。
21	'spam' not in D2	搜索'spam'是否不为字典D2中的一个键，没有搜索到则返回True，搜索到则返回False。
修改字典元素值		
22	D2['zyw']=6	修改字典D2中键为'zyw'的值为6，前提是键'zyw'存在
字典的其他操作		
23	len(D2)	求字典D2的长度（键-值对的个数）
24	D2.items()	返回一个包含字典D2中所有键-值对的列表
25	D2.keys()	返回一个包含字典D2中所有键的列表
26	D2.values()	返回一个包含字典D2中所有值的列表
27	D2.iter()	返回字典D2的一个迭代子

列表、元组、字典（1）

- 列表、元组、字典的共同点
  - 都属于python内置数据结构;
  - 都用于管理多个元素;
  - 都是线性结构;
  - 都提供的丰富的对管理的元素进行操作的功能;
  - 列表、元组、字典中各元素均采用逗号（,）进行分隔;
  - 列表、元组、字典内各元素类型可以不同。

纠正02课件中的错误

列表、元组、字典（2）

- 列表、元组、字典的不同点
  - 列表内元素用中括号（[]）包裹、元组内元素用小括号（()）包裹、字典内元素用大括号（{}）包裹;
  - 列表、字典内元素个数及元素的值可以修改，元组内的元素个数及元素的值不可以修改（元组可以看成是只读的列表）；

课后作业

- 资料查阅
  - 查阅相关资料，按照课件中给出的列表所有操作表格的形式，列出python中集合（set）的所有操作;
  - 本次作业为python的课后作业，计入python总成绩;
  - 注：列表中有的操作可以直接借鉴。
- 程序作业
  - 练习列表、元组、字典的所有操作，掌握尽可能多的操作。

好好想想，有没有问题?

谢　谢！

北京大学

《Python语言入门》课程 第五讲 字符串

张永伟

北京大学 软微学院 语言信息工程系

2009年12月3日

北京大学

课程内容

- 课程内容
  - 字符串的定义
  - 字符串的删除
  - 字符串的基本操作
  - 转义字符
  - 字符串常用操作
  - 课堂测验

字符串的定义

- 字符串的定义
  - 字符串为引号之间的字符集合，这里引号包括单引号、双引号，三引号（三个连续的单引号或双引号）。

```
>>> s1='I love Python'
>>> s2=str([1,2,3])
>>> s1
'I love Python'
>>> s2
'[1, 2, 3]'
```

字符串的删除

- 删除字符串:
  - del s#s为要删除的字符串的名字

```
>>> s1='I love Python'
>>> s1
'I love Python'
>>> del s1
>>> s1

Traceback (most recent call last):
  File "<pyshell#157>", line 1, in <module>
    s1
NameError: name 's1' is not defined
```

字符串的基本操作

- 字符串属于序列，所以也支持索引、切片的操作。所有序列都有的“in” “not in” “*” “+” len() 也同样适用于字符串，且含义相同。
  - 索引：str[i]
  - 切片：str[i:j]
  - 判断一个sub字符串是不是属于str字符串: sub in str; sub not in str;
  - 字符串的重复：str*3
  - 字符串的连接：str1+str2
  - 求字符串的 长度：len(str)

转义字符（1）

- 转义字符
  - 一个反斜线加一个单一字符可以表示一个特殊字符，通常是一个不可打印的字符。例如’\t’表示一个字符，该字符为制表符。
  - 如果这些特殊字符包含在一个原字符串中，则就失去了他们转义后的意义。
- 原字符串
  - 原字符串的定义即在字符串的前面加一个r，如定义字符串s，s=r'I love\tPython'，输出s时会发现s的内容为'I love\\tPython'，在这里制表符\t即为一个转义的特殊字符，但在所定义的原字符串中，它失去了它的特殊含义。

转义字符（2）

八进制	字符	说明
\0	NUL	空字符Nul
\a	BEL	响铃字符
\b	BS	退格
\t	HT	横向制表符
\n	LF	换行
\v	VT	纵向制表符
\f	FF	换页
\r	CR	回车
\e	ESC	转义
\"	"	双引号
\'	'	单引号
\\	\	反斜杠

转义字符（3）

- 例子:

```
>>> print 'i\tlike\tpython'
i       like    python
>>> print r'i\tlike\tpthon'
i\tlike\tpthon
>>> print '\\i like python\\'
\i like python\
>>> print "I'm a boy"
I'm a boy
>>> print 'I\'m a boy'
I'm a boy
>>> print """I'm a boy"""
I'm a boy
```

- 注意
  - 成对定义的单引号中单引号要转义
  - 成对定义的双引号中双引号要转义
  - 成对的三引号中单引号双引号都不需要转义
  - 成对定义的单引号中双引号不需要转义
  - 成对定义的双引号中单引号不需要转义

字符串转换

- 字符串转换相关函数
  - **str(obj)** 将其他类型内容转换为字符串
  - **int(obj)** 将字符串或整数转换为浮点数
  - **float(obj)** 将字符串或整数转换为浮点数
  - **long(obj)** 将字符串转换为长整型

字符串常用操作

- **str.capitalize()**
  - 将字符串首字符大写，并返回新的首字符大写后的字符串。

```
>>> 'i like python'.capitalize()
'I like python'
```

字符串常用操作

- **str.center(width[, fillchar])**
  - 返回一个长为width的新字符串，在新字符中，原字符串居中，将fillchar指定的符号（默认为空格）填充至其前后。

```
>>> 'i like python'.center(20,'*')
'***i like python****'
```

字符串常用操作

- **str.count(sub[, start[, end]])**
  - 返回sub在str里面出现的次数，如果start，end指定，则返回str中指定范围内sub出现的次数。

```
>>> 'i like python'.count('i')
2
>>> 'i like python'.count('i',1)
1
>>> 'i like python'.count('i',1,8)
1
```

字符串常用操作

- **str.endswith(suffix[, start[, end]])**
  - 判断字符串是否以suffix结束，如果start，end指定，则返回str中指定范围内str子串是否以suffix为结尾。如果是返回True，否则返回False。

```
>>> 'i like python'.endswith('thon')
True
>>> 'i like python'.endswith('Python')
False
```

字符串常用操作

- **str.find(sub[, start[, end]])**
  - 判断sub是否在str中，如果start，end指定，则返回str中指定范围内sub出现的位置。在则返回在整个str中开始的索引值，不在则返回-1。

```
>>> 'i like python'.find('i')
0
>>> 'i like python'.find('i',1)
3
>>> 'i like python'.find('like')
2
>>> 'i like python'.find('zyw')
-1
```

字符串常用操作

- **str.index(sub[, start[, end]])**
  - 同find()函数功能相同，只不过当没有查找到sub时会抛出ValueError异常。

```
>>> 'i like python'.index('i')
0
>>> 'i like python'.index('i',1)
3
>>> 'i like python'.index('like')
2
>>> 'i like python'.index('zyw')

Traceback (most recent call last):
  File "<pyshell#34>", line 1, in <module>
    'i like python'.index('zyw')
ValueError: substring not found
```

字符串常用操作

- **str.isalnum()**
  - 如果str至少含有一个字符，并且所有的字符都是字母或数字则返回True，否则返回False。(空格即不是字母也不是数字)

```
>>> 'i like python'.isalnum()
False
>>> 'ilikepython'.isalnum()
True
>>> 'ilikepython1348869'.isalnum()
True
>>> '1348869'.isalnum()
True
```

字符串常用操作

- str.isalpha(): 判断是不是都是字母字符
- str.isdigit(): 判断是不是都是数字
- str.islower(): 判断是不是都是小写
- str.isspace(): 判断是不是都是英文空格
- str.istitle(): 判断是不是都是标题化文本
- str.isupper(): 判断是不是都是大写

字符串常用操作

```
str.isalpha()
如果str至少含有一个字符，并且所有的字符都是字母则返回True，否则返回False。(空格不是字母)
>>> 'i like python'.isalpha()
False
>>> 'ilikepython'.isalpha()
True
>>> 'ilikepython1348869'.isalpha()
False
>>> '1348869'.isalpha()
False

str.isdigit()
如果str至少含有一个字符，并且所有的字符都是数字则返回True，否则返回False。
>>> 'i like python'.isdigit()
False
>>> 'ilikepython'.isdigit()
False
>>> 'ilikepython1348869'.isdigit()
False
>>> '1348869'.isdigit()
True
```

字符串常用操作

```
str.islower()
如果str至少含有一个区分大小写的字符，并且所有的区分大小写的字符都是小写则返回True，否则返回False。
>>> 'I like python'.islower()
False
>>> 'i like python'.islower()
True
>>> ''.islower()
False
>>> '1348869'.islower()
False

str.isspace()
如果str至少含有一个字符，并且所有的字符都是英文空格则返回True，否则返回False。
>>> 'I like python'.isspace()
False
>>> '          '.isspace()
True
>>> '  '.isspace()
False
>>> ''.isspace()
False
```

字符串常用操作

str.istitle()
如果str是标题化字符串，并且至少还有一个有大小写变化的字符，则返回True，否则返回False。标题化字符串指的是字符串内所有单词的第一个有大小写变化的字符是否被大写，而其余字母小写。

```
>>> 'I Like Python'.istitle()
True
>>> 'I Like Python2009'.istitle()
True
>>> '2009'.istitle()
False
>>> 'I Like 2009Python'.istitle()
True
>>> 'I Like 2009 Python'.istitle()
True
```

str.isupper()
如果str至少含有一个区分大小写的字符，并且所有的区分大小写的字符都是大写则返回True，否则返回False。

```
>>> 'i like python'.isupper()
False
>>> 'I LIKE PYTHON'.isupper()
True
>>> ''.isupper()
False
>>> '1348869'.isupper()
False
```

字符串常用操作

- **str.join(seq)**
  - 以str作为分隔符，将序列seq中所有元素（必须为字符串或转换为字符串）合并为一个新的字符串。

```
>>> '**'.join(['3','4','5'])
'3**4**5'
>>> '**'.join(('3','4','5'))
'3**4**5'
>>> '**'.join('python')
'p**y**t**h**o**n'
>>> '**'.join([3,4,5])

Traceback (most recent call last):
  File "<pyshell#62>", line 1, in <module>
    '**'.join([3,4,5])
TypeError: sequence item 0: expected string, int found
```

字符串常用操作

- **str.ljust(width[, fillchar])**
  - 返回一个原字符串左对齐，并用指定符号fillchar（默认为英文空格）填充至长度为width的新字符串。如果width<len(str)，则返回str。

```
>>> 'i like python'.ljust(20,"*")
'i like python*******'
>>> 'i like python'.ljust(5,"*")
'i like python'
```

字符串常用操作

- **str.lower()**
  - 返回一个str中所有字符均小写的新字符串。

```
>>> "Big".lower()
'big'
>>> '3355'.lower()
'3355'
```

字符串常用操作

- **str.lstrip([chars])**
  - 与strip()函数功能类似，只删除str右侧连续包含chars字符串指定的字符。chars不指定时，默认为英文空格。

```
>>> '   spacious   '.lstrip()
'spacious   '
>>> 'www.example.com'.lstrip('cmowz.')
'example.com'
```

字符串常用操作

- **str.replace(old, new[, count])**
  - 将str中的old字符串替换为new字符串，并将替换后的新字符串返回，如果count指定，则只替换前count个字符串。

```
>>> 'I like like like
Python'.replace('like','love')
'I love love love Python'
>>> 'I like like like
Python'.replace('like','love',2)
'I love love like Python'
>>> 'I like like like
Python'.replace('love','like',2)
'I like like like Python'
```

字符串常用操作

- **str.rfind(sub[, start[, end]])**
  - 与find()函数功能类似，只不过是从str右边开始查找。

```
>>> 'i like python'.rfind('i')
3
>>> 'i like python'.rfind('i',1)
3
>>> 'i like like like
python'.rfind('like')
12
>>> 'i like python'.rfind('love')
-1
```

字符串常用操作

- **str.rindex(sub[, start[, end]])**
  - 与index()函数功能类似，只不过是从str右边开始查找。

```
>>> 'i like python'.rindex('i')
3
>>> 'i like python'.rindex('i',1)
3
>>> 'i like like like python'.rindex('like')
12
>>> 'i like python'.rindex('love')

Traceback (most recent call last):
  File "<pyshell#117>", line 1, in <module>
    'i like python'.rindex('love')
ValueError: substring not found
```

字符串常用操作

- **str.rjust(width[, fillchar])**
  - 返回一个原字符串右对齐，并用指定符号fillchar（默认为英文空格）填充至长度为width的新字符串。如果width<len(str)，则返回str。

```
>>> 'i like python'.rjust(20,"*")
'*******i like python'
>>> 'i like python'.rjust(5,"*")
'i like python'
```

字符串常用操作

- **str.rsplit([sep[, maxsplit]])**
  - 与split()函数功能类似，只不过是从maxsplit指定时，从右边开始分割maxsplit次。

```
>>> '1<>2<>3'.rsplit('<>')
['1', '', '2', '3']
>>> '1<>2<>3<>4<>5<>6'.rsplit('<>',3)
['1<>2<>3', '4', '5', '6']
```

字符串常用操作

- **str.rstrip([chars])**
  - 与strip()函数功能类似，只删除str右侧连续包含chars字符串指定的字符。chars不指定时，默认为英文空格。

```
>>> '   spacious   '.rstrip()
'   spacious'
>>> 'mississippi'.rstrip('ipz')
'mississ'
```

字符串常用操作

- str.split([sep[, maxsplit]])
  - 以sep字符串为分隔符对str进行分割得到一个字符串列表，如果maxsplit指定，则仅分割maxsplit次（这样，列表中最多有maxsplit+1个元素），如果没有指定maxsplit，则全部分割。如果sep指定，则多个连续的sep字符串不会被认为是一个sep字符串，分割结果也会包含相应的空串。指定sep情况下分割一个空的字符串会得到 ['']。如果sep没有指定，多个连续的空格会被认为是一个分隔符，返回的字符串列表中也不含有空串。sep省略或为None时分割一个空串或只含有英文空格的字符串会得到[]

字符串常用操作

- **str.split([sep[, maxsplit]])**

```
>>> '1<>2<>3'.split('<>')
['1', '2', '3']
>>> '1<><>2<>3'.split('<>')
['1', '', '2', '3']
>>> ''.split('<>')
['']
>>> '   1    2 3  '.split()
['1', '2', '3']
>>> '   1    2 3  '.split(None)
['1', '2', '3']
>>> '   1    2 3  '.split(None,1)
['1', '2 3  ']
>>> "           ".split(None)
[]
>>>
```

字符串常用操作

- **str.splitlines([keepends])**
  - 返回一个列表，列表中每一个元素为字符串str的一行（用采用换行符分隔），元素可能为空。如果keepends没有指定或指定为假，则返回的列表元素中不包含换行符，如果指定，且不为假，则每个元素末尾都包含换行符。

字符串常用操作

- **str.splitlines([keepends])**

```
>>> "I love Python\nHe loves Python\nAnd she also loves
Python\n\n".splitlines(1)
['I love Python\n', 'He loves Python\n', 'And she also loves Python\n',
'\n']
>>> "I love Python\nHe loves Python\nAnd she also loves
Python\n\n".splitlines()
['I love Python', 'He loves Python', 'And she also loves Python', '']
>>> "I love Python\nHe loves Python\nAnd she also loves
Python\n\n".splitlines(0)
['I love Python', 'He loves Python', 'And she also loves Python', '']
```

字符串常用操作

- **str.startswith(prefix[, start[, end]])**
  - 判断字符串是否以prefix开始（prefix在这里可以为字符串或一个元组，为元组时则表示以任何一个元组中的字符串开始），如果start，end指定，则判断str指定范围内的子串是否以prefix开始。如果是返回True，否则返回False。

```
>>> 'i like python'.startswith('i like')
True
>>> 'i like python'.startswith('zyw')
False
```

字符串常用操作

- **str.strip([chars])**
  - 返回一个新的字符串，字符串的首尾中连续含有chars字符串内字符的字符被删除。如果不指定chars，则默认删除str的首尾连续英文空格。

```
>>> '   spacious   '.strip()
'spacious'
>>> 'www.example.com'.strip('cmowz.')
'example'
```

字符串常用操作

- **str.swapcase()**
  - 返回一个新的字符串，字符串中大写字符改为小写，小写字符改为大写。

```
>>> '2009 I like Python'.swapcase()
'2009 i LIKE pYTHON'
```

字符串常用操作

- **str.title()**
  - 返回标题化的字符串：字符串内所有单词首字母大写，其余字母小写。如果单词以没有大小写变化的字符开始，则忽略这些字符，而将单词的第一个具有大小写变化的字符大写。整个单词都没有字符有大小写变化，则不改变这个单词。

```
>>> "i like pen of green".title()
'I Like Pen Of Green'
>>> '2009i like python'.title()
'2009I Like Python'
>>> '2009i 2009like 2009python2009 2009'.title()
'2009I 2009Like 2009Python2009 2009'
```

字符串常用操作

- **str.upper()**
  - 返回一个str中所有字符均大写的新字符串。

```
>>> "Big".upper()
'BIG'
>>> '1348869'.upper()
'1348869'
```

字符串常用操作

- **str.zfill(width)**
  - 返回长度为width的新字符串，str位于新字符串右端，前面填充0。如果str长度大于width，则str会被返回。一般用于数字字符串。

```
>>> 'python'.zfill(20)
'00000000000000python'
>>> 'python'.zfill(2)
'python'
>>> '2102'.zfill(20)
'00000000000000002102'
>>> for i in range(1,13):
        print str(i).zfill(2),'月',

01 月 02 月 03 月 04 月 05 月 06 月 07 月 08 月 09 月
10 月 11 月 12 月
```

python课程考试

- 分数计算
  - 课后作业50分
  - 课堂测验50分

- 课堂测验
  - 四道题目，前两道题目满分50分，后两道题目满分70分
  - 四道题目任选其一
  - 前两道题目容易些，第两道题目难一些，多出的20分为奖励分
  - 根据自己实际掌握的情况选作。

课堂测验

- 试题一
  - 要求用户输入一个英文句子，统计该英文句子中含有单词的数目。
  - 提示，英文单词间可以认为都用英文空格进行分割。
  - 测试数据:
    - 输入： **I like python very much.**
    - 输出： 5

课堂测验

- 有如下一篇文档，要求编写程序，将中英文句子分开到两个不同的文件中。

I love China.
我爱中国。
I like Python.
我喜欢Python。
I will be always with you.
我将永远陪伴你。

I love China.
I like Python.
I will be always with you.

我爱中国。
我喜欢Python。
我将永远陪伴你。

- 测试数据：
  - 如上所示，中英文文本文件名为“zh_en.txt”，生成的中文文本文件名为“zh.txt”，生成的英文文本文件名为“en.txt”

课堂测验

- 试题三
  - 要求用户输入一个英文句子，统计该英文句子中含有单词的数目以及词的种数。
  - 提示，英文单词间可以认为都用英文空格进行分割。
  - 测试数据:
    - 输入1: I like python very much.
    - 输出1: 5，5 (50分)
    - 输入2: I like python very very much.
    - 输出：6，5 (20分)

课堂测验

- 有如下一篇文档，要求编写程序，将中英文句子分开到两个不同的文件中。

```
I love China.
我爱中国。

I like Python.
我喜欢Python。

I will be always with you.
我将永远陪伴你。
```

```
I love China.
I like Python.
I will be always with you.
```

```
我爱中国。
我喜欢Python。
我将永远陪伴你。
```

- 测试数据：
  - 如上所示，中英文文本文件名为“zh_en.txt”，生成的中文文本文件名为“zh.txt”，生成的英文文本文件名为“en.txt”

好好想想，有没有问题?

谢　谢！

北京大学