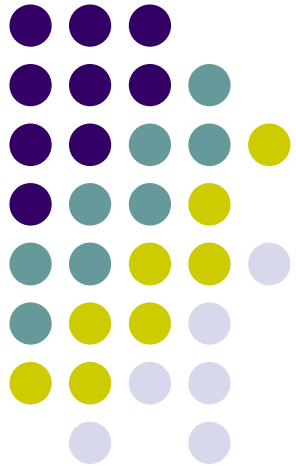
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

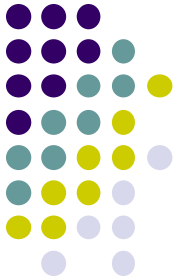
第2章数据表示
Data Representation



申丽萍

lpshen@sjtu.edu.cn

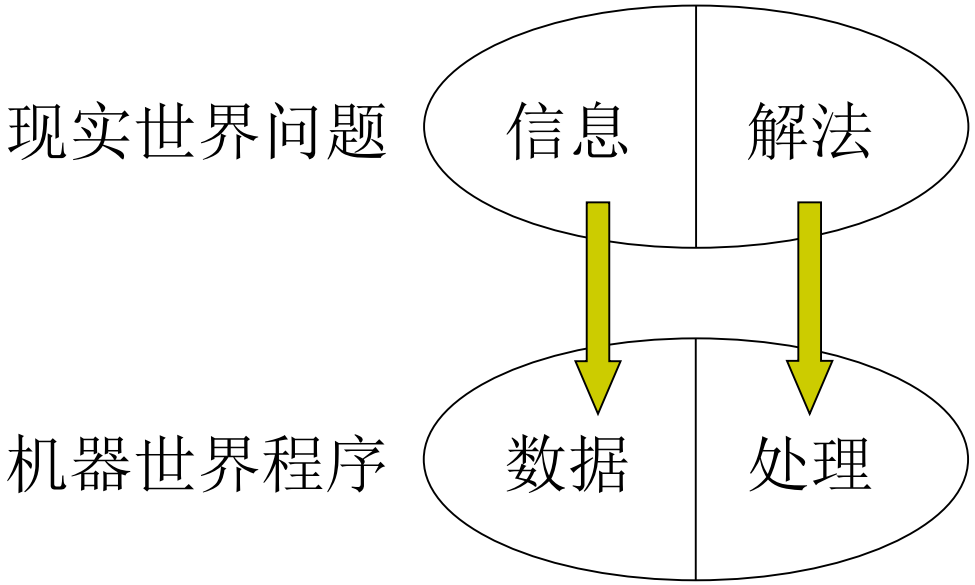
第2章 用数据表示世界



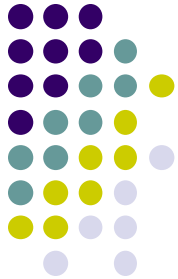
- 数据与数据类型
- 数值数据类型
- 数学库模块
- 布尔类型
- 字符串类型
- 集合体类型：列表、元组、字典
- 输入输出

数据处理

- 计算机=数据处理机器
- 计算=数据+处理
- 问题求解=信息表示+解法表示



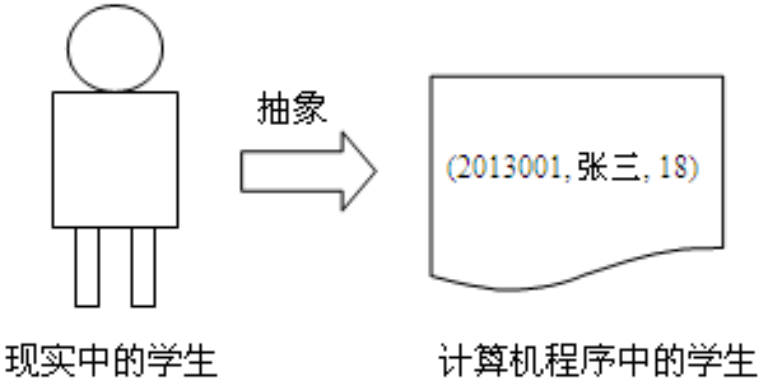
初识Python – 基本成分



- Python程序由模块(module) 组成
  - 可以保存成脚本文件或源文件，后缀名为py。
  - 模块中代码强制缩进表示隶属关系。
- 模块中包含语句(statement) （语句包含定义和指令）
  - 如：print “hello”，可以直接在命令提示符>>>后面输入
- 语句中包含表达式 (expression)
  - 如：x + 1，表达式中包含常量数据和变量数据
- 表达式由数据（data）和作用于数据的运算符组成。
  - 如：数据1 运算符 数据2
  - 如：x + 1

数据

- 抽象:抽取与问题求解有关的信息进行计算机表示。抽象是简化问题的重要手段。
- 数据:现实世界事实或信息在计算机中的符号化表示.
  - 温度:表示为35,或95,或"摄氏35度"
  - 学生:



常量与变量

两种符号化表示:

- 常量/数值：其数据值由字面决定，并且不可改变。
  - 例如: "Hello,World!", 3.14, ...
- 变量：用符号化名字（标识符）表示可变的数据。
  - 标识符构成规则：以字母或下划线开头，后面跟随若干个字母、数字和下划线。如_student, firstName等。
  - Python通过赋值语句来定义变量。例如:

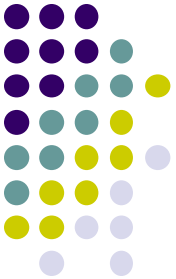
```
>>> s = "Hello, World!"
>>> print s
```

```
>>> s = 3.14
```

  - 变量赋值后才有意义，否则就是“未定义的”。
  - Python中的同时赋值:

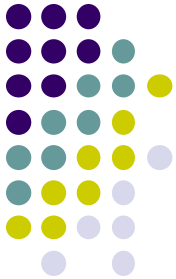
```
>>> x,y = "hello",3.14
>>> x,y=y,x
```

数据类型



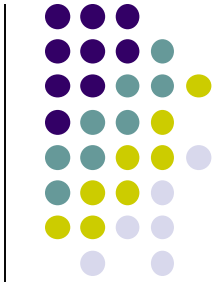
- 为了更精细地表示信息,编程语言提供不同的数据类型。
  - 不同类型具有不同的存储方式和处理方式.
  - 每种类型包含一个合法值的集合,以及一个合法运算的集合.
- 为何要区分数据类型?
  - 类型信息可为系统利用。编译器/解释器利用类型检查,可以发现程序错误.
- 数据类型：
  - 数值类型： 整数类型 int，浮点数类型 float等
  - 字符串 str
  - 布尔类型 bool
  - 集合体类型：列表 list, 元组 tuple，字典dict，文件file等
  - type(<expr>)函数返回<expr>的值的类型

第2章 用数据表示世界



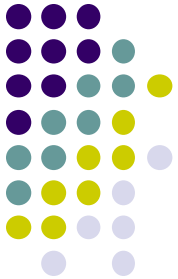
- 数据与数据类型
- 数值数据类型
- 数学库模块
- 布尔类型
- 字符串类型
- 列表类型
- 元组类型

数值数据类型



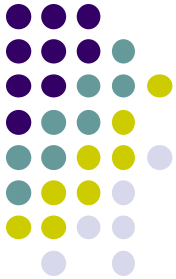
- 整数类型int
  - 不带小数点，如: 123，-456， 0
  - 可存储整数的精确值
- 长整数类型long
  - 超出整数int范围的整数，数值后面加后缀”L”或”l”，如2147483648L
  - 任意长度（当然受限于内存容量）
  - long类型的运算由程序实现
- 浮点数类型float
  - 带小数点，如：3.14 -2.718 13. 0.0
  - 计算机只能存储浮点数的近似值!
- 经验:能用int就不用long，能用int就不用float.

整数的限制



- 计算机中的数值是现实数值的一种抽象,两者可能有不同的行为!
  - 例如:整数集是无穷集,但int是有穷的.
- 二进制是计算机表示数据的基础
  - *n*位二进制只能表示2^n个不同值
  - 整数若用32位表示,则int范围为 $-2^{31} \sim 2^{31}-1$，即 -2147483648 ~ 2147483647
  - 试试看：输出2147483647和2147483648
    - 早期版本会导致溢出! Python 2.7解决了溢出问题,直接转换成大整数类型.
  - 小测验:怎样计算$2^{31}-1$使其结果不带L?

计算是次序的艺术



- int类型的范围: $-2^{31} \sim 2^{31}-1$

```
>>> 2**31 - 1
2147483647L
```

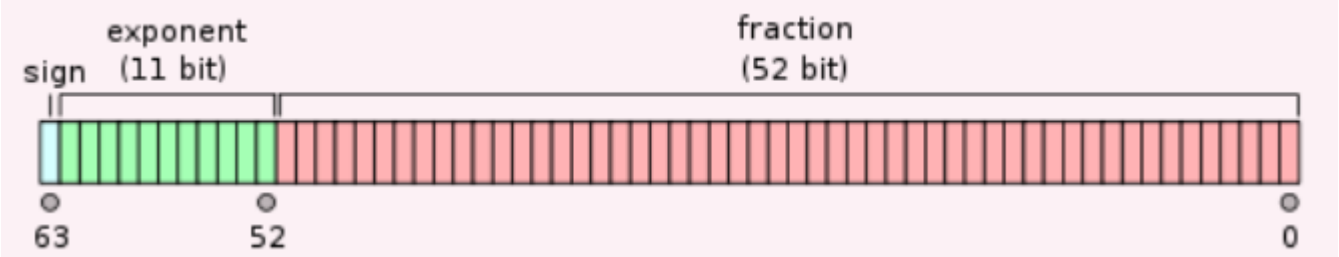
  - 未超出int范围,为什么结果是long?
- 如何计算$2^{31}-1$,并使结果在int中?

```
>>> 2**30 - 1 + 2**30
2147483647
```

- 计算思维与数学思维是不同的!

实数的表示

- 定点表示：小数点的位置固定不变。
- 浮点表示：小数点位置不固定。一个浮点数分成尾数和阶码两部分。阶码表示小数点在该数中的位数，尾数表示数的有效数值。
  - N=246.135，其浮点表示可为：N = 246135 * 10^{-3}
  - 浮点数能表示巨大的数值，很大时用科学表示法:1.234e+12
    - 试验出的pytho最大数:1.79e+308
  - 有些浮点数无法精确表示，只能存储近似值
    - 1.2-1.0=0.19999999999999996

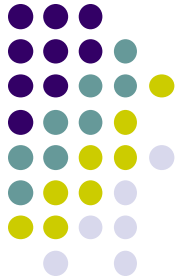


数值运算符

运算符	整数	浮点数
+	加	加
–	减	减
*	乘	乘
/	除(结果取整)	除
**	乘方	乘方
%	余数	(商取整时的)余数
abs()	绝对值	绝对值

- 整数、长整数和浮点数可以混合运算:
  - 运算数都是整数,结果为整数;
  - 整数和长整数运算，结果为长整数
  - 至少有一个运算数是浮点数,结果为浮点数.

类型转换



- 在混合类型表达式中,Python自动类型转换:

  int→ long → float

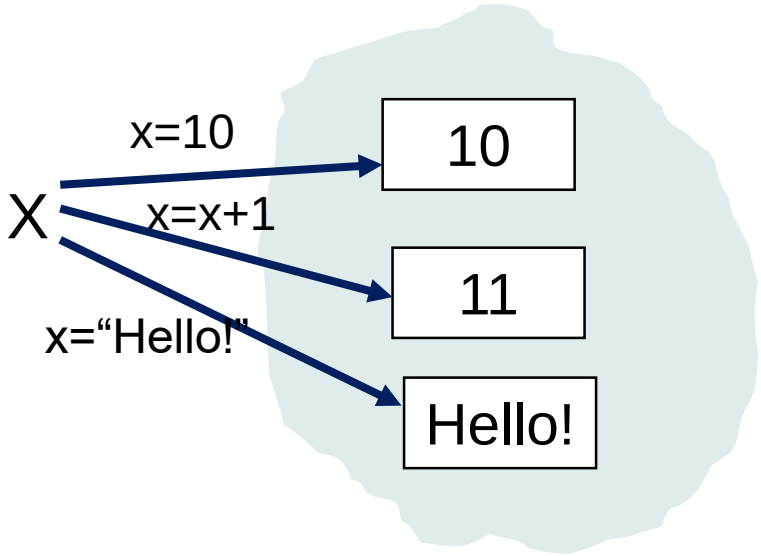
- 人工类型转换:利用int(), long(), float()
  - 思考:求整数的平均值时用哪个好?

```
avg = sum / count
avg = float(sum) / count
avg = float(sum / count)
```

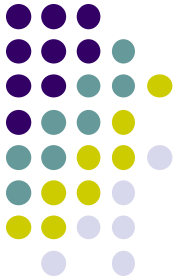
- 浮点数取整时如何做到四舍五入?
  - int(x+0.5)
  - round()

Python的动态类型

- 如何理解x=x+1，fact=fact*f ？
- Python采用动态类型化。
  - 变量不是某个预先定义数据类型的固定内存单元的标识，
  - 变量是对内存中某个数据的引用，引用可动态改变。
  - 变量不需要预先定义数据类型，类型随引用动态改变。

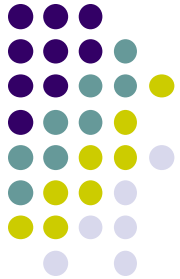


第2章 用数据表示世界



- 数据与数据类型
- 数值数据类型
- 数学库模块
- 布尔类型
- 字符串类型
- 列表类型
- 元组类型

数学库



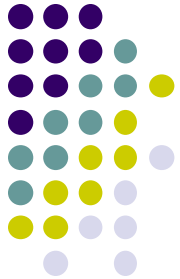
- 库:包含有用定义的模块.
  - 最常见的是函数定义
  - 一般由系统提供,也可自己开发给别人用.
- 数学库:包含有用的数学函数.
- 数学库的两种导入方式:

```
import math
from math import *
```

- 数学库函数的调用:例如求平方根

```
math.sqrt(4)
sqrt(4)
```

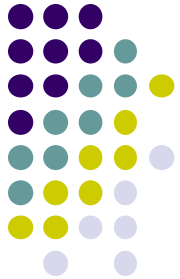
编程例:quadratic.py



```
import math
def main():
    a, b, c = input("Enter three coefficients (a,b,c): ")
    discRoot = math.sqrt(b*b - 4*a*c)
    r1 = (-b + discRoot) / (2 * a)
    r2 = (-b - discRoot) / (2 * a)
    print "The solutions are:", r1, r2
```

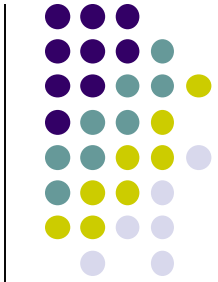
- 用到math.sqrt()函数
- 小测验:不用sqrt()能求平方根吗?
  - 库函数一般效率高.

数学库中的常用函数



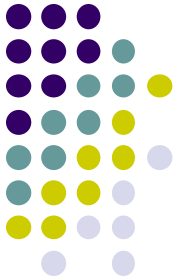
1. 常数π：pi
2. 常数e：*e*
3. 平方根：sqrt(x)
4. 三角函数：sin(x), cos(x), tan(x), asin(x), acos(x), atan(x)
5. 自然对数与常用对数：log(x), log10(x)
6. *e*的x次方：exp(x)
7. ≥x的最小整数：ceil(x)
8. ≤x的最大整数：floor(x)

一种常用算法模式:累积



- 最终结果是由逐个中间结果累积起来形成的.
  - 例如6!的计算:先算6*5,再*4,…,再*1而得.
- 这种累积程序需要一个存放累积结果的变量,累积过程是一个循环:
  - 初始化累积变量
  - 循环直至得到最终结果
  - 计算累积变量的当前累积值
- 初始化:给累积变量一个合适的初值,以便进入循环后能正确计算.
  - 忘记初始化是一个常见编程错误!

例:阶乘计算程序



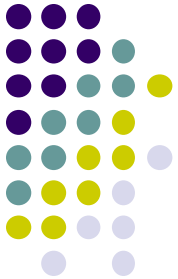
- 具体数(如6)的阶乘

```
fact = 1
for f in [6,5,4,3,2,1]:
  fact = fact * f
```

- 一般情形

```
n = input(" Enter a number: ")
fact = 1
for f in range(n,1,-1):
    fact = fact * f
print "The factorial of ",n, "is",fact
```

阶乘程序的几点注解



- 由于乘法结合律,累积的次序是不重要的.如按下面的循环来累积:

```
for f in [2,3,4,5,6]:
for f in [2,4,6,1,3,5]:
for f in range(2,n+1)
```

- range()函数

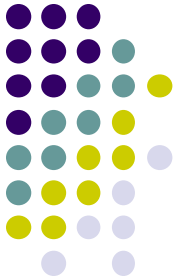
  range(n)返回[0,1,2…n-1]

  range(start,n)返回[start,start+1,…n-1]

  range(start,n,step)返回
  [start,start+step,…start+mstep<n]

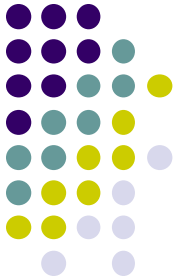
- 计算机科学与程序设计，习惯从0开始计数。

第2章 用数据表示世界



- 数据与数据类型
- 数值数据类型
- 数学库模块
- 布尔类型
- 字符串类型
- 列表类型
- 元组类型

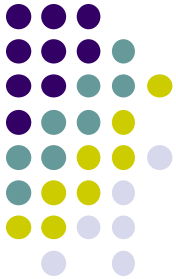
布尔类型 bool



- 布尔类型bool
  - 合法值:True, False.用于表达命题的"真","假“
- 最简单的命题是关系运算，通常出现在if语句
  - 关系运算符有：>, <, >=, <=, ==, !=

```
>>> 3 > 2
True
>>> 4 + 5 == 5 + 4
True
>>> 3! =2
True
>>> "like" < "lake"
False
```

逻辑运算符与布尔表达式



- 三个逻辑运算符（根据优先级次序）：not、and、or
- 简单的关系运算用逻辑运算符联结起来，构成复杂的布尔表达式。

```
>>> (3 > 2) and (5>4)
True
>>> not (4 + 5 == 2 + 7)
False
>>> ("like" < "lake“) or (“B-2”<“f-16”)
True
```

布尔表达式的应用 - if语句

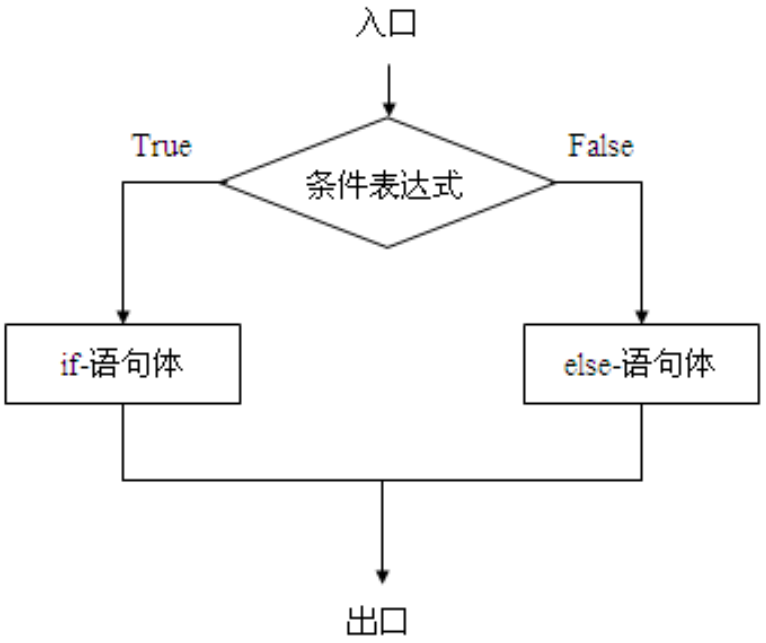
- 语法

  if <布尔表达式>:

  <if-语句体>

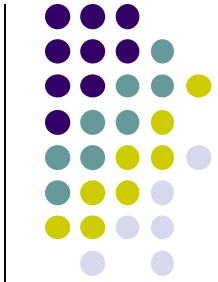
  else:

  <else-语句体>



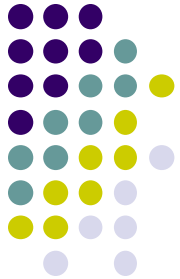
- 语义
  - 若<布尔表达式>为真,执行<if-语句体>,控制转向下一条语句;否则执行<else-语句体>,控制转向下一条语句。

if语句例子（demo04)



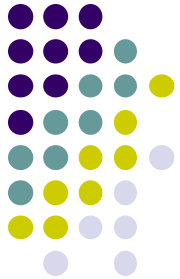
```
# Python program outputs the maximum of 3 values

a,b,c=input("please input 3 numbers (a,b,c):")
max=0
if a>b:
    max=a
else:
    max=b
if c>max:
    max=c
print "The maximum of a,b,c is: ",max
```

第2章 用数据表示世界



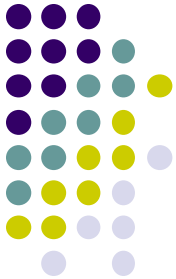
- 数据与数据类型
- 数值数据类型
- 数学库模块
- 布尔类型
- 字符串类型
- 列表类型
- 元组类型

文本数据



- 计算机应用
  - 科学计算
  - 信息管理
- 信息管理中大量的数据都是文本数据.
  - 如:姓名,地址,简历等等
  - 小测验:身份证号码,电话号码等是数值?
- 计算机中用字符串来表示文本数据.

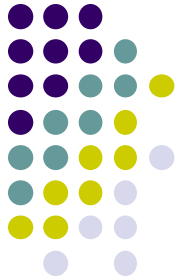
字符串类型 str



- 字符串类型str：字符序列
- 字符：计算机中信息表示的最小单位，分可打印字符如大小写字符、阿拉伯数字和标点等，和不可打印字符如回车、换行等。
- 字符串字面值：用一对引号(单/双/三)括住。

```
'hello world'
"~!@#$%^&*"
"汉字也是字符"
'''line one,
   line two'''
"""line one,
   line two"""
```

字符的转义



- 字符串本身含有引号怎么办?
  - 含有单引号:串用双引号括住

    "I'm a student."
  - 含有双引号:串用单引号括住

    'He said, "OK."'
  - 两者都有时用什么引号?

    He said, " I'm a student."
- 一般的做法:用转义字符“\”来转变字符含义。转义符“\”后面的引号不再解释为界定符，而是普通的引号。

  “He said, \"I‘m a student.\""

  'He said, "I\'m a student."'

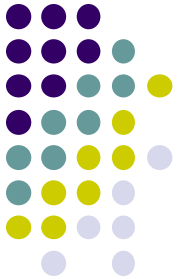
字符串操作:取字符

- 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>]
  $S = s_0, s_1, s_2, s_3, \ldots, s_{n-1}$
  - 对长度为*n*的字符串,索引可以
    - 是大于0的数:自左向右为0 ~ *n*–1, 或者
    - 是负数:自右向左为–1, –2, –3,…, –*n*
  - 例如:若str = "Hello Bob",则

H	e	l	l	o		B	o	b
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

str[0]或str[–9]是'H'
str[5]或str[–4]是' '
str[8]或str[–1]是'b'
str[9]或str[–10]越界出错

取字符例子 (demo07)



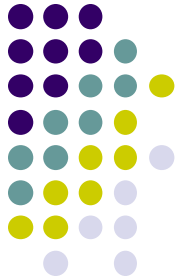
```
# counting vowels in a string without count() function

s=raw_input("please input a string to count vowels:")
vCount=0      # viariable counts the number of vowels.

# range(len(s))返回的正好是字符串的索引号范围!
for i in range(len(s)):
    if s[i] in ["a","e","i","o","u"]:
        vCount=vCount+1

print "The number of vowels in '",s,"' is ",vCount
```

字符串操作:取子串（切分）

- 切分:取一个索引范围内的字符.
  <字符串>[<start>:<end>]
  - 所取子串:位置索引从start ~ end–1
  - start或/和end可省略,缺省值为串的首/尾
  - 例如:若str = "Hello Bob",则
    - str[0:3]是'Hel'
    - str[5:9]是' Bob'
    - str[:5]即str[0:5]
    - str[5:]即str[5,9]
    - str[:]即str[0:9]

H	e	l	l	o		B	o	b
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

字符串切分例子 (demo08)



```
# Counting "bob" appears in a given string.
#eg. if the given string is 'azcbobobegghakl',then the output should be 2

s=raw_input("Please input a string:")
bCount=0                    # counting the "bob"s.

# 注意range范围应该是字符串长度-2，否则后面s[i:i+3]会出界
for i in range(len(s)-2):
    substring=s[i:i+3]          #提取3个字符长度的字串
    if substring=="bob":
        bCount=bCount+1
print "the number of times 'bob' appears in '",s,"' is",bCount
```

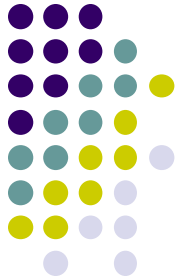
字符串其它操作

Operator	Meaning
+	Concatenation
*	Repetition
<string>[ ]	Indexing
len(<string>)	length
<string>[:]	slicing

- 两字符串的连接 + <string1> + <string2>
  - 例如:"Hello" + "Bob"得到"HelloBob"
- 一个字符串的重复 *
  - 例如:3*"Hi"和"Hi"*3都得到"HiHiHi"
- 子串检测 in
  - 例如:"ok" in "cook"返回True(见后)
- 字符串不能更改(immutable)，只能生成新的字符串。
  - 例如： s[1]=“H” 是错误的。

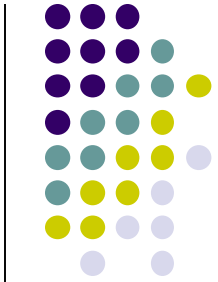
```
>>> s="hello world!"
>>> s[1]="H"

Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    s[1]="H"
TypeError: 'str' object does not support item assignment
```

字符的机内表示



- 与数值一样,计算机内用二进制数表示每一个字符.
  - 因此操作字符串本质上仍然是数值运算.
  - 表示字符的这个数值称为字符的编码.
- 问题:计算机采用什么字符集?其中每个字符用什么编码?
  - 对这个问题的不同回答就导致了许多不同的字符编码系统.

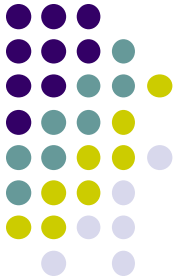
编码标准



- 不同计算机若用不同编码,则彼此无法沟通. 标准化：
  - ASCII:单字节编码,但只用到7位(0~127)
    - 96个可打印字符,32个控制字符
  - ISO/IEC 8859-1(Latin-1):单字节用满8位(0~255)
  - GB2312:两字节(7445字符/6763汉字)
  - GBK:两字节(21886字符/21003汉字),中文缺省编码
  - GB18030:最多四字节(76556字符/70244汉字)
  - ISO/IEC 10646或Unicode:最多四字节.

```
>>> sys.getdefaultencoding()
'ascii'
>>> sys.getfilesystemencoding()
'mbcs'
>>> locale.getdefaultlocale()
('zh_CN', 'cp936')
```

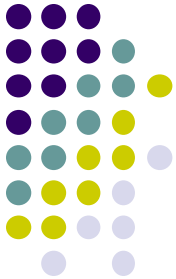
字符与编码



- 求给定字符的编码: ord()
  `ord('a')`可得`97`
- 求给定编码的字符:chr()
  `chr(97)`可得`'a'`
- 可见Python 2.7默认编码为ASCII.
  - Q:非ASCII字符怎么办?
  - A:用Unicode字符串

```
>>>print u'A\xc4B'
AÄB
```

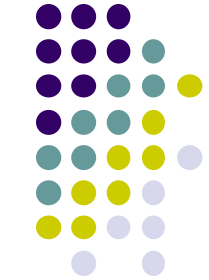
Python程序中使用汉字



- 中文Windows的缺省编码是GBK

```
>>> str1="汉"
>>> str2=u"汉"
>>> len(str1)
2
>>> len(str2)
1
>>> print repr(str1)
'\xba\xba'
>>> print repr(str2)
u'\u6c49'
>>> print '\xba\xba\xc4'
ººÄ
>>> print u'\u6c49\xc4'      # 当要输入多种字符集时，用unicode
汉Ä
```

编码与解码

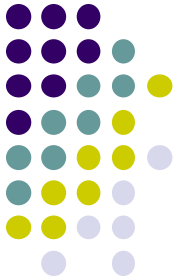


- 编码与解码是用计算机解决问题时常用的方法
  - 底层的字符编码用Unicode
  - 打开文件处理数据时通常要解码成unicode
  - 保存成文件时通常要编码
- 解码　　<string>.decode(<standardCoding>)
- 编码　　<unicode>.encode(<standardCoding>)

```
>>> str1="汉"
>>> str2=u"汉"
>>> s1=str1.decode("GBK")
>>> len(s1)
1
>>> s1
u'\u6c49'
>>> str2.encode("UTF-8")
'\xe6\xb1\x89'
>>> s2=str2.encode("UTF-8")
>>> len(s2)
3
>>> print s2
汉
>>> s22=s2.decode("GBK")

Traceback (most recent call last):
  File "<pyshell#53>", line 1, in <module>
    s22=s2.decode("GBK")
UnicodeDecodeError: 'gbk' codec can't decode byte 0x89 in position 2: incomplete
 multibyte sequence
```

UTF-8编码的字符串不能用GBK解码

数值与字符串的互相转换



- eval()函数:将字符串当作数值表达式进行计算.
  - 语法: `eval(<string>)`
  - 例如: `eval("3+4*5")`
  - 例如 `eval(“a>8 and True”)`

```
>>> a=10
>>> eval("a>8 and True")
True
```

- 直接用某种数据类型进行转换：
  - 例如：`int(“123”),long(“123”), float(“123”)`
  - 例如：`bool(“True”)`
- str()函数:将数值当作字符串
  - 语法: str(<expr>)
  - 例如: str(3+4*5)

```
>>> str(3+4*5)
'23'
```

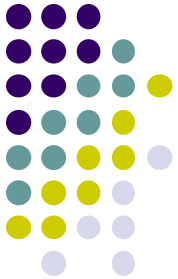
string库

- 模块string：包含了很多有用的字符串处理函数

函数	含义	类
capitalize(s)	首字母大写	字母处理
upper(s)	全部大写	
lower(s)	全部小写	
capwords(s)	每个单词首字母大写	
count(s, sub)	子串sub在s中出现的次数	搜索相关
find(s, sub)	子串sub在s中首次出现的位置	
find(s,sub,start,end)	子串sub在s中指定起始和结束点首次出现的位置	
rfind(s,sub)	从右边开始查找子串sub在s中首次出现的位置	
startswith(s,start)	是否以start开头	
endswith(s, end)	是否以end结尾	
replace(s, sub, newsub)	将s中所有子串sub替换成newsub	
split(s, ch)	将s按ch分割拆分成子串列表。默认按空格分隔	
strip(s)	去掉s中所有前导和尾部空格	格式相关
lstrip(s)	去掉s中所有前导空格	
rstrip(s)	去掉s中所有尾部空格	
ljust(s, width)	获取固定长度，左对齐，右边不够用空格补齐	
rjust(s, width)	获取固定长度，右对齐，左边不够用空格补齐	
join(list)	将列表list中的所有字符串合并成一个字符串	

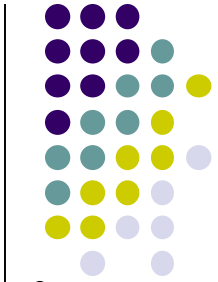
注: 所有函数都可以用两种方式调用，如upper(s), 或者 s.upper().

string库 – 例子



```
>>> from string import *
>>> capwords("hello world!")
'Hello World!'
>>> count("知之为知之，不知为不知","不知")
2
>>> find("知之为知之，不知为不知","不知")
12
>>> rfind("知之为知之，不知为不知","不知")
18
>>> print replace("知之为知之，不知为不知","知","zhi")
zhi之为zhi之，不zhi为不zhi
>>> split("a-b-c-d","-")
['a', 'b', 'c', 'd']
```

编程实例:序列查找



在一个字符串序列里查找目标：利用位置规律来定位（定长）。

例1： 查月份：

```
months="JanFebMarAprMayJunJulAugSepOctNovDec"
m = input("Enter month number (1-12): ")
pos = (m-1)*3
monthAbbr = months[pos:pos+3]
```

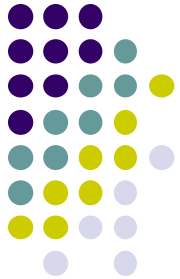
例2： 在一个字符串里查找单词“bob”

```
# This is demo09 to output the
# longest alphabetical sequences in a string

s=raw_input('Please input the string: ')
start=0
# find all the alphabetical sequences
strList=[]    # for the alphabetical sequences
for i in range(len(s)-1):
    if s[i]>s[i+1]:
        strList=strList+[s[start:i+1]]
        start=i+1
strList=strList+[s[start:]]
```

第2章 用数据表示世界

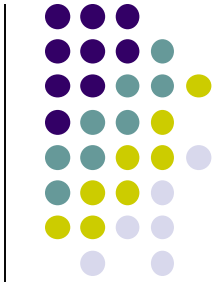
- 数据与数据类型
- 数值数据类型
- 数学库模块
- 布尔类型
- 字符串类型
- 集合体类型：列表、元组、字典
- 输入输出

数据集合体



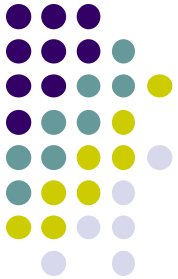
- 很多程序都需要处理大量类似数据的集合.
  - 文档中的大量单词,
  - 海量的Internet数据
  - 实验得到的数据如DNA序列, ...
- 原子类型： int, long, float, bool都是“原子”值。
- str类型是由多个字符组成的序列。
- 有没有一个对象能包含很多数据?
  - 例如: range(5) = [0,1,2,3,4]
  - 例如: string.split("This is it.") = ['This','is','it']
- 集合体类型：能够用一个变量来存储大量数据的类型，包括列表、元组、字典和文件。

列表类型



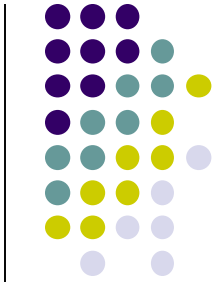
- 列表(List):是一种数据集合体.
  - 是数据项的有序序列
  - 例如：`[], [1,2,3] [1,"two",3.0,True]`
- 数据整体用一个名字表示
  - 例如: seq = ['abc', 2, True]
- 数据成员通过位置索引引用
  - 例如: seq[2]=True

列表操作



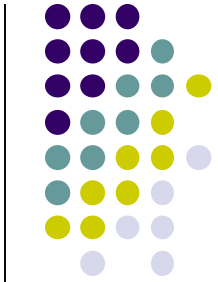
- 类似字符串操作:
  - 合并: `<seq> + <seq>`
  - 重复: `<seq> * <int_expr>`
  - 索引: `<seq>[<index_expr>]`
  - 分段: `<seq>[<start>:<end>]`
  - 长度: `len(<seq>)`
  - 迭代: `for <var> in <seq>: ...`
- 删除列表成员:
  - `del <seq>[<start>:<end>]`

列表操作(续)



- 专用于列表的方法:
  - 追加: `<list>.append(x)`
  - 排序: `<list>.sort()`
  - 逆转: `<list>.reverse()`
  - 查找x的索引: `<list>.index(x)`
  - 在i处插入x: `<list>.insert(i,x)`
  - 数x的个数: `<list>.count(x)`
  - 删除x: `<list>.remove(x)`
  - 按索引取出成员: `<list>.pop(i)`
  - 隶属: `x in <list>`

列表操作-索引



- 索引操作和字符串类似
  - 通过在序列中的位置编号来访问成员
    <列表>[<位置编号>]
  - 例如

```
>>> x = [1,"two",3.0,True]
>>> x[0]
1
>>> x[-1]
True
>>> x[1+1]
3.0
```

列表操作-子列表

- 子列表操作和字符串类似
  - 指定序列中的开始和结束位置

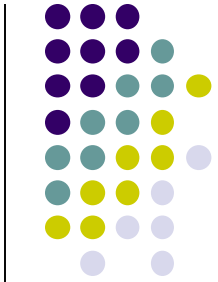
    <列表>[<开始位置>:<结束位置>]
  - 例如

```
>>> x = [1,"two",3.0,True]
>>> x[0:2]
[1, 'two']
>>> x[1:]
['two', 3.0, True]
>>> x[:-1]
[1, 'two', 3.0]
```

- 列表也有+和*操作,意义和字符串类似

```
>>> [1,3,5]+[2,4]
[1, 3, 5, 2, 4]
>>> 4*[3.0,True]
[3.0, True, 3.0, True, 3.0, True, 3.0, True]
```

与列表有关的几个内建函数



- 求列表长度len()

```
>>> x=4*[3.0,True]
>>> len(x)
8
```

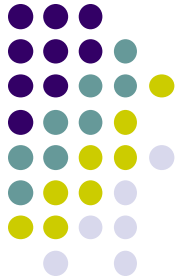
- 删除列表成员del()

```
>>> x=[1,2,3]
>>> del x[1]
>>> x
[1, 3]
```

- 产生整数列表range()

```
>>> range(1,10,2)
[1, 3, 5, 7, 9]
```

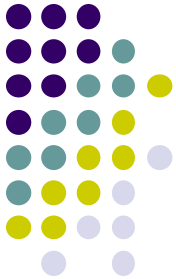
列表与字符串



- 回顾: 字符串是字符序列,可通过索引引用串的组成部分.
- 列表与字符串的区别:
  - 列表的成员可以是任何数据类型,而字符串中只能是字符;
  - 字符串不能增删改,而列表可以

```
>>> x=[1, True, "spring"]
>>> x[0]=6
>>> x
[6, True, 'spring']
>>> del x[1]
>>> x
[6, 'spring']
```

列表与数组



- list与其他语言中的数组array相似,但不同
  - 列表是动态的,而数组是定长的
  - 列表可以增删成员
  - 不要求各成员都是相同类型的
  - 成员本身也可以是列表
  - 例如

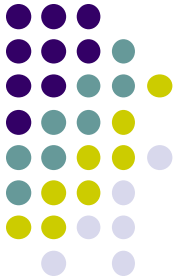
```
[2,"apples"]
[1,"two",3.0,True]
[[1,"apple"],[2,"pears"]]
```

元组类型

- 元组类型tuple
  - 用圆括号括起的成员集合体，如(), (8,) (3,“6",True)
  - 和列表基本相同,只是不能删改成员

```
>>> print (8),(8,)
8 (8,)
>>> t1=(3,"3",True)
>>> t1[1]
'3'
>>> t1[0:2]
(3, '3')
>>> t1+(4,5,6)
(3, '3', True, 4, 5, 6)
>>> t1[1]=8

Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    t1[1]=8
TypeError: 'tuple' object does not support item assignment
```

有序集合体：str, list,tuple

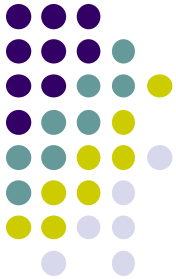


- 大量数据按照次序排列的集合体称为序列(sequence)
- 序列通用的操作：
  - 索引：s[i], s[i:j]
  - 检测x是否在序列：x in s, x not in s
  - 排序：sorted(s, cmp, key, reverse)　vs　list.sorted(cmp, key, reverse)
    - 例子：wordCount=[("a",10),("data",15),("we",20),("the", 16),("key",6)]

```
>>> wordCount=[("a",10),("data",15),("we",20),("the", 16),("key",6)]
>>> sorted(wordCount,lambda x,y:cmp(x[1],y[1]))
[('key', 6), ('a', 10), ('data', 15), ('the', 16), ('we', 20)]
>>> wordCount
[('a', 10), ('data', 15), ('we', 20), ('the', 16), ('key', 6)]
>>> sorted(wordCount,key=lambda x:x[1])
[('key', 6), ('a', 10), ('data', 15), ('the', 16), ('we', 20)]
>>> wordCount
[('a', 10), ('data', 15), ('we', 20), ('the', 16), ('key', 6)]
>>> import operator
>>> sorted(wordCount,key=operator.itemgetter(1))
[('key', 6), ('a', 10), ('data', 15), ('the', 16), ('we', 20)]
>>> wordCount
[('a', 10), ('data', 15), ('we', 20), ('the', 16), ('key', 6)]
>>> wordCount.sort(key=lambda x:x[1],reverse=True)
>>> wordCount
[('we', 20), ('the', 16), ('data', 15), ('a', 10), ('key', 6)]
```

```
>>> t1=(10,5,8,3,1,7)
>>> t1.sort()

Traceback (most recent call last):
  File "<pyshell#41>", line 1, in <module>
    t1.sort()
AttributeError: 'tuple' object has no attribute '
>>> sorted(t1)
[1, 3, 5, 7, 8, 10]
```

字典:无序集合体



- 列表实现了索引查找:按给定位置检索.
- 很多应用需要“键-值”查找:按给定的键,检索相关联的值.
- 字典类型(dict):存储“键-值对”.
  - 创建: dict = {k_1:v_1, k_2:v_2, ... , k_n:v_n}
  - 检索: dict[<k_i>]返回相关联的<v_i>
  - 值可修改:dict[<k_i>] = <new_value>
  - 键类型常用字符串,整数;值类型则任意.
  - 存储:按内部最有效的方式,不保持创建顺序.

字典例

- 缩略语字典

  abbr = {'etc':'cetera', 'cf':'confer', 'ibid':'ibidem'}

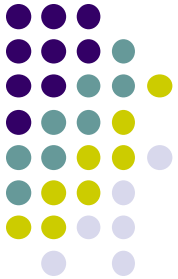
```
>>> abbr={'etc':'cetera','cf.':'confer', 'e.g.':'for example','ibid':'ibidem'}
>>> abbr['etc']
'cetera'
>>> abbr['etc']='et cetera'
>>> abbr
{'cf.': 'confer', 'etc': 'et cetera', 'ibid': 'ibidem', 'e.g.': 'for example'}
```

- 月份映射表

  month = {1:'Jan', 2:'Feb', 3:'March',4:'April'}

```
>>> month = {1:'Jan', 2:'Feb', 3:'March',4:'April'}
>>> month[4]
'April'
>>> month[5]

Traceback (most recent call last):
  File "<pyshell#24>", line 1, in <module>
    month[5]
KeyError: 5
>>> month[5]='May'
>>> month
{1: 'Jan', 2: 'Feb', 3: 'March', 4: 'April', 5: 'May'}
```

字典操作



- 键存在性:<dict>.has_key(<key>)
- 键列表:<dict>.keys()
- 值列表:<dict>.values()
- 键值对列表:<dict>.items()
- 删除键值对:del <dict>[<key>]
- 清空字典:<dict>.clear()

```
>>> month.has_key(5)
True
>>> month.keys()
[1, 2, 3, 4, 5]
>>> month.values()
['Jan', 'Feb', 'March', 'April', 'May']
>>> month.items()
[(1, 'Jan'), (2, 'Feb'), (3, 'March'), (4, 'April'), (5, 'May')]
>>> del month[1]
>>> month
{2: 'Feb', 3: 'March', 4: 'April', 5: 'May'}
>>> month.clear()
>>> month
{}
```

编程实例:词频统计 assign3

- 统计文档中单词的出现次数.
- 用字典结构:
  - 用很多累积变量显然不好!
  - counts:{<单词>:<频度计数>}
- 分词

```
for ch in ",.;:\n":
    text.replace(ch, ' ')
    wordlist = string.split(text)
```

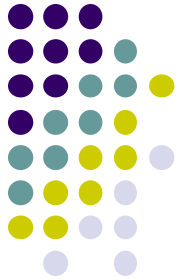
- 单词首次出现时字典里查不到会出错:

```
for w in wordlist:
    try:
        wordCounts[w]=wordCounts[w]+1
    except KeyError:
        wordCounts[w]=1
```

- 如何输出前n个最频繁的单词? 根据频度进行排序生成列表:

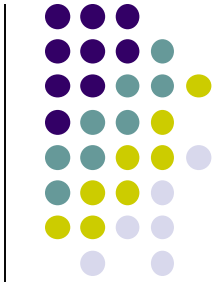
```
Countlist=sorted(counts.items(), key=lambda count:count[1],reserse=True)
```

第2章 用数据表示世界



- 数据与数据类型
- 数值数据类型
- 数学库模块
- 布尔类型
- 字符串类型
- 列表与元组类型
- 输入输出

输入 – 代码中静态输入



- 程序中数据如何提供?
  - 编程时提供

```
def main():
    name = "Lucy"
    age = 7
    birthYear = 2012 - age
    print name,"was born in", str(birthYear)+"."

main()
```

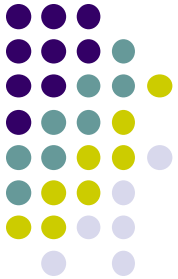
输入- input()

- 程序中数据如何提供?
  - 编程时提供
  - 运行时输入:

    <变量> = input(<提示>)

```
>>> x = input("请输入:")
>>> n,a = input("请输入姓名和年龄")
def main():
    name = input("请输入姓名: ")        'Lucy'
    age = input("请输入年龄: ")          20
    birthYear = 2015 - age
    print name,"was born in",str(birthYear)+"."

main()
```

输入- raw_input()



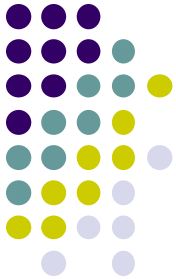
- 另一种输入函数

  <变量> = raw_input(<提示>)

  - input将输入内容作为表达式来求值,而raw_input将输入整体视为字符串.
  - 例如

```
>>> n = raw_input("输入姓名")   Lucy
>> a = raw_input("输入年龄")    20
>>> 2015 - eval(a)
```

input与raw_input



- 例:比较

```
>>> x=input("your name please:")
your name please:'Liping'
>>> x=input("your age please:")
your age please:2*8+2
>>> x
18
>>> 2000+x
2018
```

```
>>> x=raw_input("your name please:")
your name please:Liping
>>> x=raw_input("your age please:")
your age please:2*8+2
>>> x
'2*8+2'
>>> 2000+eval(x)
2018
```

- 由此可见：
  - input()将输入当作表达式
  - raw_input()将输入当成字符串数据

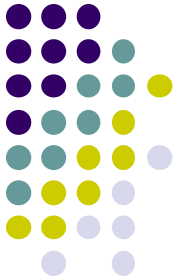
字符串的输入

- 错误输入:

```
>>> x=input("your name please:")
your name please:Liping

Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    x=input("your name please:")
  File "<string>", line 1, in <module>
NameError: name 'Liping' is not defined
```

- 原因:input()是把输入当成表达式来计算的!
- 解决方法:
  - 输入时加上引号
  - 使用raw_input()

输出语句print



- 语法

  print

  print <表达式>

  print <表达式1>, <表达式2>, ... , <表达式n>

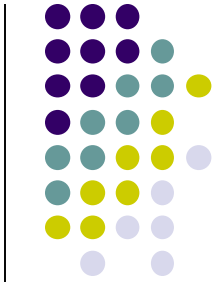
  print <表达式1>, <表达式2>, ... , <表达式n>,

- 例子：

```
>>> print

>>> print 3+4
7
>>> print 3,4,3+4
3 4 7
>>> print 3+    4
7
>>> print "The answer is ",       3+4,
The answer is  7
```

```
print "3+4="       3+4=
print 3+4          7
print "3+4=",      3+4= 7
print 3+4
```

格式化输出



- 格式化运算符%

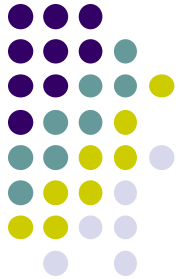
  <模板串> % （<数据1>,……, <数据n>）

  - %用来分隔格式定义与数据
  - 格式化运算的结果是一个字符串
- 模板串: 描述填入的值的输出形式，出现在字符串中

  % <width>.<precision><type>

  - % 标记“空位”,输出时用相应值填入
  - 三种类型字符type：decimal, float, string
  - 宽度width: 用多少位置显示数值.
    - 省略或指定为0:根据值的实际长度显示.
    - 宽度超出值的长度时:右对齐显示
      - 宽度前加负号:左对齐.
  - 精度precision：用于浮点数输出，表示小数点后保留几位数字

格式化输出实例

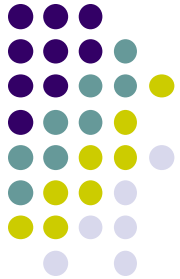


```
>>> "Hello %s %s, you may have already won $%d" % ("Mr.", "Smith", 10000)
'Hello Mr. Smith, you may have already won $10000'
>>> 'This int, %5d, was placed in a field of width 5' % (7)
'This int,     7, was placed in a field of width 5'
>>> 'This int, %10d, was placed in a field of witdh 10' % (10)
'This int,          10, was placed in a field of witdh 10'
>>> 'This float, %10.5f, has width 10 and precision 5.' % (3.1415926)
'This float,    3.14159, has width 10 and precision 5.'
>>> 'This float, %0.5f, has width 0 and precision 5.' % (3.1415926)
'This float, 3.14159, has width 0 and precision 5.'
>>> 'Compare %f and %0.20f' % (3.14, 3.14)
'Compare 3.140000 and 3.14000000000000012434'
```

- 浮点数如何精确化?
  - 银行应用要求精确表示金额,故不宜用浮点数及浮点运算.
  - 解决办法:以“分”为单位,用整数表示金额.

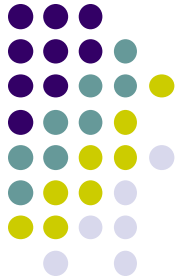
```
>>> x=96666666
>>> print "You have %d.%02d" % (x/100, x%100)
You have 966666.66
```

文件处理



- 文件:对存储在磁盘上的一组数据予以命名.
- 典型的数据组织粒度:
  - 基本数据项
  - 若干数据项构成固定结构的记录
  - 若干记录构成文件
- 例:
  - 基本数据项:学号,姓名,年龄
  - 一个学生的记录:{学号,姓名,年龄}
  - 一个文件:全体学生的记录

文本文件

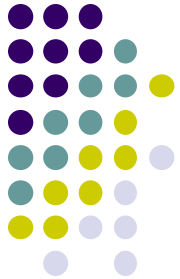


- 文件中是文本数据
  - 相应地有二进制数据.
- 可视为存储在磁盘上的字符串.
  - 单行字符串
  - 多行字符串
    - 行尾(EOL):用特殊字符,如新行(\n)字符.
    - Python用\n表示新行字符,该字符在显示时被解释成新行字符. 例:

```
print “first line\nsecond line”
```

```
>>> print 'first line\nsecond line'
first line
second line
```

文件处理:打开文件



- 打开文件:将磁盘文件与一个程序变量关联,做好读写准备.

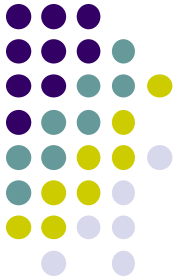
  `<filevar> = open(<filename>,<mode>)`

  - `<mode>`: 'r','w','a'
- 例如

```
infile = open("myfile","r")
outfile = open("myfile","w")
oldfile = open("myfile","a")
```

  - 写打开要小心!可能破坏现有文件

文件处理:读写文件



- 读文件:读出文件内容

  `<filevar>.read()`

  `<filevar>.readline()`

  `<filevar>.readlines()`

  - 点表示法:程序中文件是对象!
  - 要有文件当前读写位置的概念!
- 写文件:将新内容写入文件.

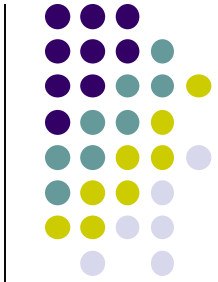
  `<filevar>.write(<string>)`

  - 若想写多行内容,在string后加\n
- 关闭文件:取消文件变量与磁盘文件的关联.

  `<filevar>.close()`

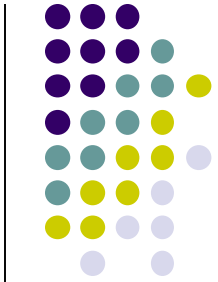
  - 系统会将缓存中的文件内容输出到磁盘，并释放资源。

文件读写演示



```
>>>newfile = open("test.txt",'w')
>>>newfile.write("First line\n")
>>>newfile.write("Second line\n")
>>>newfile.close()
>>>infile = open("test.txt","r")
>>>infile.readline()
>>>infile.readline()
>>>infile.readline()
>>>infile.close()
>>>oldfile.open("test.txt","a")
>>>oldfile.write("Third line")
>>>oldfile.close()
>>>infile = open("test.txt","r")
>>>infile.readlines()
```

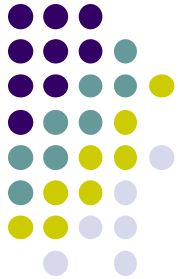
编程实例:批处理



- 通过文件实现成批数据的输入输出
  - 这种情况不适合用交互方式输入

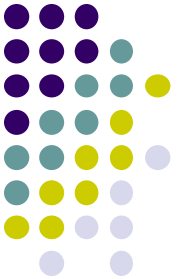
```
infile = open(infileName,'r')
outfile = open(outfileName,'w')
for line in infile.readlines():
    first, last = string.split(line)
    uname = string.lower(first[0]+last[:7])
    outfile.write(uname + '\n')

infile.close()
outfile.close()
```

assign2- 简单的数据计算程序

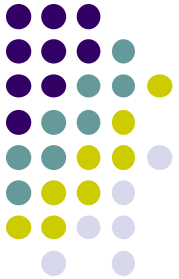


- 课本76页习题第13，23题，附加统计元音字母的题目。
- 注意从ftp://public.sjtu.edu.cn/ct/assignments 下载详细作业2文档
- 上机时间：10月8日 8：00~9：40
- 上机地点：电院4号楼313机房
- 截止日期：10月8日 24：00

assign3- 文本文件中的单词计数



- 注意从ftp://public.sjtu.edu.cn/ct/assignments 下载详细作业3文档
- 上机时间：10月22日 8：00~9：40
- 上机地点：电院4号楼313机房
- 截止日期：10月22日 24：00



End