Python 语言程序设计

与大数据分析

作者

高萁 苏庆栋
任腾龙 刘元

排版：聂义莹

版本号：V1.0

此版本包括 Python 语言程序设计
与大数据分析第 1-10 章内容

修订时间：2020.8

山软智库-让知识回归平凡

第一章 Python简介

1.1 Python的产生及发展

Python的创始人是荷兰人**吉多·范罗苏姆** （Guido van Rossum）。

1989年圣诞节期间，在阿姆斯特丹，Guido为了打发时间，决心开发一个新的脚本解释程序，作为ABC语言的一种继承。该编程语言的名字Python（大蟒蛇的意思），取自他非常感兴趣的一部英剧《蒙提.派森的飞行马戏团》（Monty Python's Flying Circus）。

2000年10月，Python 2.0正式发布，解决了其解释器和运行环境的诸多问题，标志着Python的进一步完善。

2008年12月3日Python 3发布，在语法层面和解释器内部做了很多重大改进，不完全兼容Python 2。

2011年1月，它被TIOBE编程语言排行榜评为2010年度语言。

2008年至今，用Python编写的几万个库函数进行了版本升级。至今，绝大多是Python函数库采用Python 3.0系列语法和解释器。

1.2 Python的特点

Python语言作为一种被广泛使用的通用脚本编程语言，具有很多鲜明的特点，包括但不限于：

- 语法简洁。实现相同功能，使用Python的代码行数仅为其他编程语言的1/10~1/5。
  典型的例子：程序员入门经典语句之“HelloWorld!”如果使用Python实现仅需要一句！————
  print("HelloWorld!")
- 与平台无关。Python程序可在任何安装了解释器的计算机中执行，因此，Python程序可以不经修改地实现跨平台运行。
- 粘性扩展。Python语言提供了接口和库函数可以很好地集成C、C++、Java的语言的代码。
- 通用灵活。Python语言可用于各领域的应用程序。如科学计算、数据处理、机器人等。
- 模式多样。尽管Python 3.0解释器内部采用了面向对象方式实现，但是Python语法层面为使用者同时支持面向对象和面向过程两种编程方式。
- 类库丰富。Python解释器提供了几百个内置库和函数库。此外，还有世界各地的程序员贡献的十几万个第三方函数库可供使用，具有良好的编程生态。

还有诸如强制可读等特点并未详细介绍。后续我们将带领读者走进Python的世界，体会Python之美。

第二章 实例解析

本章我们将以温度转换程序（TempConvert.py）、Python蟒蛇绘制程序（DrawPython.py）为例,介绍Python的程序的组成部分，语法元素，库函数等，使大家对Python程序有一个初步的、基本的理解。

2.1 温度转换程序

```
#TempConvert.py
TempStr = input("请输入带有符号的温度值:")
if TempStr[-1] in ['F','f']:
    C=(eval(TempStr[0:-1]))/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C','c']:
    F=1.8*eval(TempStr[0:-1])+32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误。")
```

2.1.1 IPO方法

每个计算机程序都用来解决特定的计算问题。无论程序规模如何，每个程序都有统一的运算模式：输入数据、处理数据和输出数据。即IPO（Input，Process，Output）编程方法。

- 输入是程序的开始。它有很多种类型，如：文件输入、控制台输入、网络输入等。
- 处理是程序对输入数据进行计算产生结果的过程。
- 输出即将计算的结果按要求的格式展示出来。

2.1.2 注释

注释，是程序员在代码中加入一行或多行特殊格式的文字，用以对代码进行解释说明，提高代码可读性。如温度转换程序中的第一行文字，即为一行注释，它不参与程序的编译运行，仅作为提供给代码阅读者的提醒、解释语句。
Python中的注释语句有两种：单行注释和多行注释。

- 单行注释，以“#”开头，“#”后的内容都将被认定为注释，可从某一行的中间开始。
- 多行注释以三个单引号作为开头和结尾，其之间的内容都将被认定为注释。

举例：

```
print("这是第一句。") #单行注释
#print("这是第二句。")
print("这是第三句。")
'''
print("这是第四句。")
这是多行注释
'''
```

上述程序输出结果：

```
这是第一句。
这是第三句。
```

2.1.3 程序框架

Python语言采用严格的“缩进”来表明程序的格式框架。缩进指的是每一行代码开头的空白，类似word中的“缩进”，用来表示代码行之间的层次关系。

- Python代码编写中，用Tab键实现缩进，也可以使用多个空格（一般为4个），但**两者不混用**。建议采用4个空格方式。
- 不需要缩进的代码行顶格书写。
- Python对语句之间的层次关系没有限制，可无限嵌套。

如本节中，温度转换程序中，第四五行缩进了四个空格，第三行语句与第四五行语句即为一个所属关系。第三行语句为“第一层”，第四五行“属于”第三行。

2.1.4 input()函数

Input()函数用于从控制台获取用户输入，无论用户输入的内容为什么类型，Input()函数的返回值均为字符串类型。
在获取用户输入时，Input()函数可以包含一些提示性文字，格式如下：

```
变量=Input(提示性文字)
```

举例：

```
temp=Input("请输入内容：")
```

2.1.5 命名与保留字

与我们所熟悉的其他编程语言相同地，Python也使用变量来保存和表示具体的数据值。为了便于区分和使用，需要给它们一个“名字”，“起名”过程称为“命名”。
Python变量命名规则：

- 允许采用大写字母、小写字母、数字、下划线和汉字等字符及其组合命名。
- 名字的首字母不能是数字。
- 中间不能出现空格。
- 长度没有限制。（语法上没有限制，但实际上受限于计算机存储资源。）
- 对大小写敏感。如temp和Temp是两个不同的名字。
- **不能与保留字相同。**

保留字：
保留字，也称关键字，指被编程语言内部定义并保留使用的标识符。每种程序语言都有一套保留字，程序员编写程序时不能定义与保留字相同的标识符。

Python 保留字如下：

保留字	说明
and	用于表达式运算，逻辑与操作
as	用于类型转换
assert	断言，用于判断变量或条件表达式的值是否为真
break	中断循环语句的执行
class	用于定义类
continue	继续执行下一次循环
def	用于定义函数或方法
del	删除变量或序列的值
elif	条件语句，与if,else结合使用
else	条件语句，与if,elif结合使用，也可用于异常和循环语句
except	except包含捕获异常后的操作代码块，与try,finally结合使用
False	布尔类型数据的一个取值，意味“假”
for	for循环语句
finally	用于异常语句，出现异常后，始终要执行finally，包含的代码块，与try，except结合使用
from	用于导入模块，与import结合使用
global	定义全局变量
if	条件语句，与else，elif结合使用
import	用于导入模块，与from结合使用
in	判断变量是否在序列中
is	判断变量是否为某个类的实例
lambda	定义匿名变量
not	用于表达式运算，逻辑非操作
or	用于表达式运算，逻辑或操作
pass	空的类，方法，函数的占位符
print	打印语句
raise	异常抛出操作
return	用于从函数返回计算结果
True	布尔类型数据的一个取值，意味“真”
try	try包含可能会出现异常的语句，与except，finally结合使用
while	while的循环语句

保留字	说明
with	简化python的语句
yield	用于从函数依此返回值
None	

2.1.6 字符串

字符串是 Python 中最常用的数据类型。
字符串就是用两个双引号""或者两个单引号''括起来的一个或多个字符。温度转化程序中的第2、3、5、6、8、10行均有字符串。
字符串是字符的序列,可以按照单个字符或字符段进行索引。字符串索引有两种方式：正向递增和反向递减。假设一字符串长度为L，正向递增序号从0开始，向右递增，最右侧是L-1；反向递增序列从最右侧-1开始向左排序，号码递减，则最左侧第一个字符的递减索引号为-L。如下图所示：

举例：
温度转换程序中的第3、6行，'TempStr[-1]'即表示了访问TempStr这个字符串的反向递减索引体系中序号为-1的字符，也就是“倒数第一个字符”。
Python也可以访问字符串的某一个区间，采用$[N:M]$（N，M为字符串的索引号）的格式，表示从N到M（但不包含M）的字符串，且可以混合使用正向递增序号和反向递减序号。
举例：
温度转换程序中的第4、7行，'TempStr[0:-1]'即表示了访问TempStr这个字符串的**第0至第倒数第一个（不包括最后一个）**字符段。

```
temp='HelloWorld!'
print(temp[-1])
#输出结果是倒数第一个字符: !
print(temp[5:-1])
#输出结果是:World
```

2.1.7 赋值语句

程序中最核心的代码功能是产生或者计算新数据值，这些代码称为表达式，类似数学中的计算公式。表达式的运算结果由操作的数据类型和相关运算符决定。
例如温度转换程序中，第2、4、7行都包含表达式语句。他们有一个共同点：都含有“=”。
Python语言中，“=”表示“**赋值**”，意为**将右侧表达式的计算结果赋给左侧变量**。此外，若有多个变量需要赋值，可以使用同步赋值语句。格式如下：

> 变量1，变量2，...变量n = 表达式1，表达式2，...，表达式n

对于变量的赋值，要注意其“**后者覆盖前者**”的性质。举例：

```
temp=1
#此时将0赋值给temp，temp值为1
temp=2*2+3
#此时将表达式的运算结果1给temp，temp为7
```

x,y互换程序：

```
x=5
y=7
x=y
y=x
'''上述语句并不能实现x，y的互换,原因是：在执行第三行时，将y的值赋给x，会将x原来的值覆盖，执行完第三行后，x，y均等于7。第四行又将7复制了一遍。
'''
```

正确互换示例：

```
x=5
y=7
t=x
x=y
y=t
'''使用一个中间变量暂时存储数值，就像两杯不同的饮料互换杯子需要使用第三个杯子作“流转”用。
'''
```

或者：

```
x,y=y,x
```

2.1.8 分支

分支语句是控制程序结构的一类重要语句，它的作用是根据判断条件选择不同的代码块执行，可以理解为“**分情况讨论**”。
举例：

```
if 条件1:
    代码块1
elif 条件2:
    代码块2
#......
elif 条件n:
    代码块n
else:
    代码块n+1
```

else不需要再设置判断条件，表示其余情况排除后的剩余情况。
温度转换程序中的第3、6、9行使用了if-elif-else结构，其中第3行的条件语句'TempStr[-1] in ['F','f']'表示判断字符串变量TempStr的倒数第一个字符是否在由'F'和'f'组成的集合内，即：最后一个字符是否是'F'或'f'。如果是，则条件语句的判断结果为"True(真)"，就会进入该分支执行相应代码块。如果否，将不会进入该分支执行代码块，转而进行下一个分支的条件判断。

2.1.9 eval()函数

eval()函数是Python语言中一个十分重要的函数，它能够以Python表达式的方式解析并执行字符串。简单地说，eval()函数的作用是将输入的字符串转换成Python语句（如果可行的话），并执行该语句。在温度转换程序的第3、6行，eval(TempStr[0:-1])将输入的字符串的[0:-1]部分转换成数字，假设用户输入的是"32C",经过eval()函数将得到整数类型（在后续数据类型章节我们将讲到）的32。
举例：

```
x=2
print(eval('x*2'))
#输出结果为：4
t='string'
print(eval('t*2'))
#输出结果为：stringstring
temp="'string'"
print(eval(temp))
#eval将temp的字符串外层双引号去掉后，解释为字符串'string'
```

2.1.10 print()函数

print()函数用于输出字符信息。可直接在括号内填入字符串输出。输出变量时，需要格式化输出方式，使用format()方法讲变量整理为期望的输出格式。
分析如下语句：

```
print("转换后的温度是{:.2f}C".format(C))
```

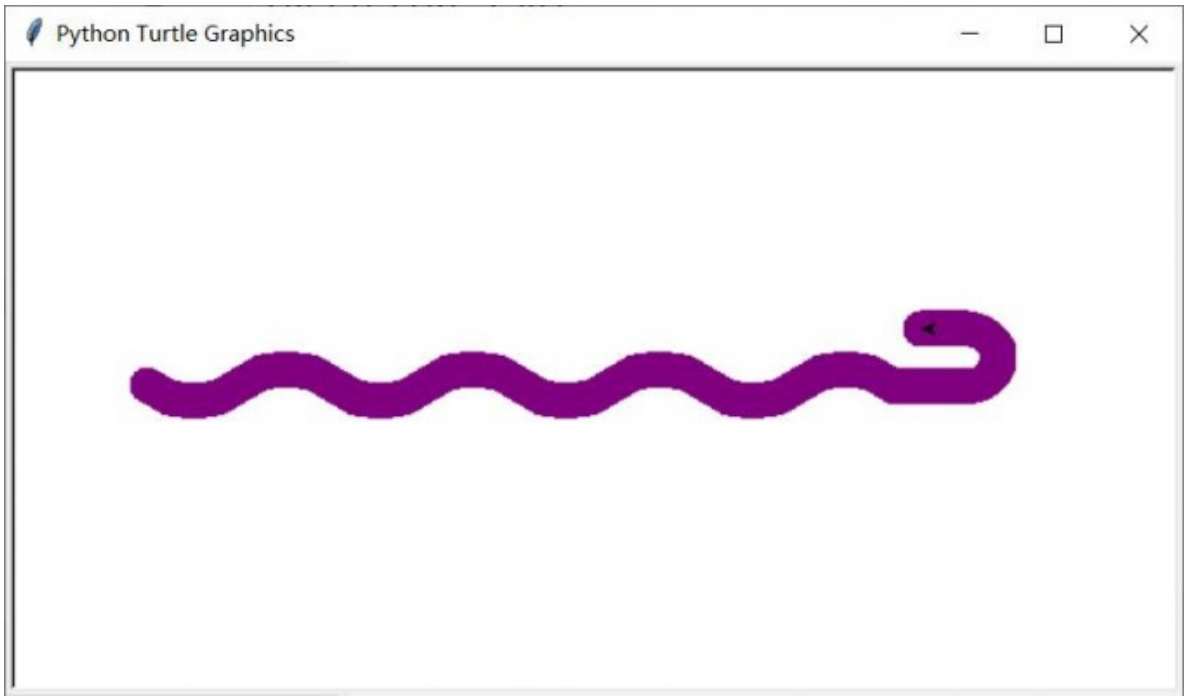
输出内容为：转换后的温度是{ }C。但需要注意的是，{ }内容需要填充，这就是format函数的作用，将format函数的参数填充到{ }内。{:.2f}中的内容表示规范数据格式，输出数值格式为两位小数。
在后续章节我们将会进一步介绍字符串格式输出方法。

2.2 Python蟒蛇绘制程序

给定一个蟒蛇绘制程序：

```
#DrawPython.py
import turtle
turtle.setup(650,350,200,200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40,80)
    turtle.circle(-40,80)
turtle.circle(4,80/2)
turtle.fd(40)
turtle.circle(16,180)
turtle.fd(40*2/3)
```

运行效果如下图：



我们将依此为例简要介绍Python库函数的使用、绘画库中的坐标体系和画笔控制函数。

2.2.1 库函数

Python的第三方程序，即可重用代码可统称为“库”。Python内置的库称为标准库，其他库称第三方库。这种通过使用并利用库中函数进行编程的方法称为“模块编程”。
使用import引用库函数有两种方式。
第一种：

```
import 库名
```

使用函数的格式如下：

```
库名.函数名(函数参数)
```

第二种：

```
from 库名 import 函数名1,函数名2,...函数名n
```

或者

```
from 库名 import *
# *是通配符，导入该库所有函数
```

使用第二种方式，不再需要每次使用函数时使用库名，可直接使用函数名调用。
如使用第二种方式时的“绘画蟒蛇”程序：

```
#DrawPython.py
from turtle import *
setup(650,350,200,200)
penup()
fd(-250)
pendown()
pensize(25)
```

```
pencolor("purple")
seth(-40)
for i in range(4):
    circle(40,80)
    circle(-40,80)
circle(4,80/2)
fd(40)
circle(16,180)
fd(40*2/3)
```

2.2.2 坐标体系

在本节中，绘画程序的成功实现依赖于对“画笔”位置和行进方向的精准控制，那么怎么实现呢？我们很容易想到，使用坐标系会是一个简洁有效的方式。

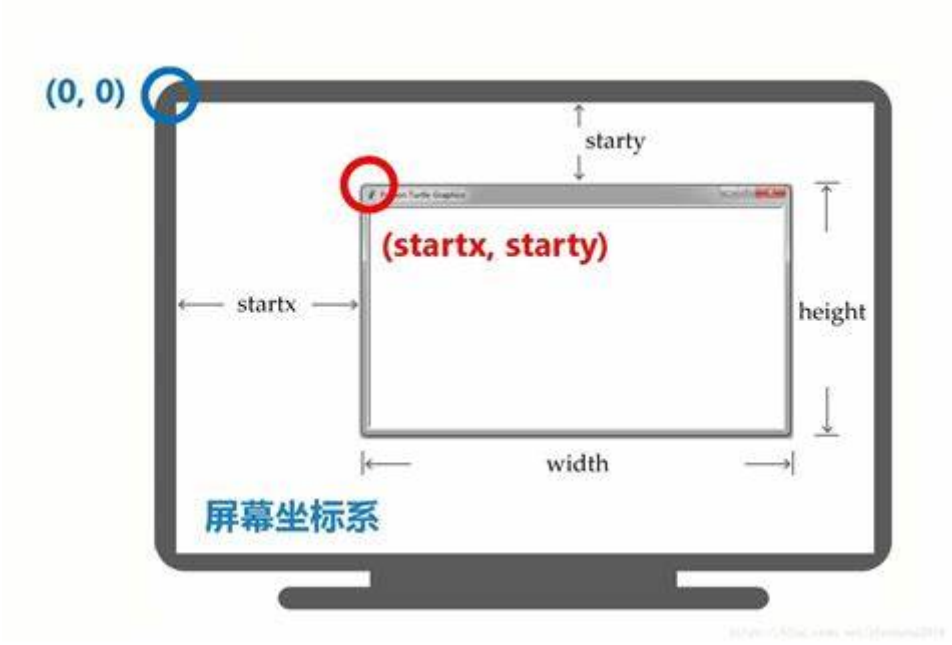
程序第3行

```
turtle.setup(650,350,200,200)
```

即实现了这一想法。setup()函数定义如下：
setup(width,height,startx,starty)

- width:窗体宽度。若为整数值，表示像素值；若为小数，表示宽度与屏幕比例。
- height:窗口高度。若为整数，表示像素值；若为小数，表示高度与屏幕比例。
- startx:窗口左侧与屏幕左侧的像素距离；若为None，窗口位于水平正中央。
- starty:窗口顶部与屏幕顶部的像素距离；若为None，窗口位于竖直正中央。

示意图：



2.2.3 画笔控制函数

DrawPython程序中的几个画笔控制函数：

函数	别名	作用	参数
penup()	pu()、up()	抬起画笔，执行后移动画笔不会绘制图形	无
pendown()	pd()、down()	落下画笔，执行后移动画笔会绘制图形	无
pensize(size)	width(size)	设置画笔宽度，无参数输入时返回当前宽度	画笔宽度size
pencolor()		设置画笔颜色，无参数输入是返回当前颜色	颜色字符串或(r,g,b)

> 1.颜色字符串：如“purple”“red”等。
> 2.(r,g,b)颜色：采用R（红色）G（绿色）B（蓝色）3中基本颜色及他们的叠加组成各种各样的颜色，构成颜色体系。

部分经典RGB颜色：

英文名称	R G B
white	255 255 255
black	0 0 0
grey	190 190 190
darkgreen	0 100 0
gold	255 215 0
violet	238 130 238
purple	160 32 240

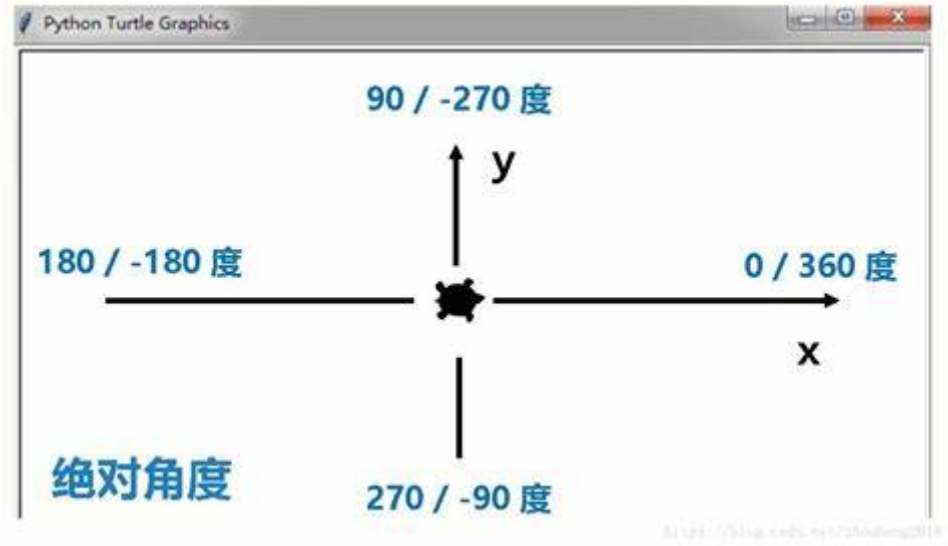
2.2.4形状控制函数

DrawPython程序中的几个绘制形状函数：

函数	别名	作用	参数
fd(distance)	forward(distance)	控制画笔向前行进distance距离	行近距离distanc
seth(to_angle)	setheading(to_angle)	控制画笔原地转动to_angle角度，该角度值是绝对方向角度值	to_angle绝对角度值
circle(radius,extent=None)		根据半径radius绘制	半径radius，

函数	别名	作用	参数
circle(radius,extent=None)		extent角度的弧形	角度extent

绝对角度示意图：



关于circle函数：
radius：弧形半径，当值为正数时，半径在小海龟左侧，值为负数时，半径在右侧。
extent：弧形角度，不设置参数时或设为None时，绘制整个圆形。

第三章 基本数据类型

本章我们的学习目标为：

- 掌握3种数字类型的概念和使用
- 了解3种数字类型在计算机中的使用方法
- 运用Python的标准数字库进行数值计算
- 掌握字符串类型的概念和使用
- 掌握字符串类型的格式化操作方法和应用

3.1 数字类型

3.1.1 数字类型概述

表示数字或数值的数据类型称为数字类型，Python提供了3种数字类型：整数、浮点数和复数，分别对应数学中的整数、实数和复数。

3.1.2 整数类型

进制种类	引导符号	描述
十进制	无	默认情况，例如，1010
二进制	0b或0B	由字符0和1组成，例如0b101
八进制	0o或0O	由字符0到7组成，例如0o711
十六进制	0x或0X	由字符0到9、a到f、A到F组成，例如，0xABC

> 内置函数：pow(x,y)用来计算x^y

3.1.3 浮点数类型

Python语言要求所有浮点数必须带有小数部分，小数部分可以是0

浮点数两种表示方法：十进制表示和科学计数法表示。

Python浮点数的数值范围和小数精度受不同计算机系统的限制。

由于Python语言能够支持无限制且准确的整数计算，因此，如果希望获得精度更高的计算结果，往往采用整数而不直接采用浮点数。

简单地说，浮点数类型的取值范围在[2^{-1023} , 2^{1023}],即[-2.225×10^{308},1.797×10^{-308}]之间，运算精度为2.220×10^{-16}。对于高精度科学计算器外的绝大部分运算来说，浮点数类型足够“可靠”，一般认为浮点数类型没有范围限制，运算结果准确。

3.1.4 复数类型

复数可看作是二元有序实数对(a,b)，表示 a + bj，其中，a是实数部分，简称实部，b是虚数部分，简称虚部

复数类型中实数部分和虚数部分的数值都是浮点数类型。对于复数z，可以用z.real和z.imag分别获得它的实数部分和虚数部分。

3.2 数字类型操作

Python解释器为数字类型提供数字运算操作符、数值运算函数、类型转换函数等操作方法。

3.2.1 内置的数值运算操作符

操作符	描述
x+y	x与y之和
x-y	x与y之差
x*y	x与y之积
x/y	x与y之商
x//y	x与y之整数商，即不大于x与y之商的最大整数

|x%y |x与y之商的余数，也成为模运算
-x	x的负值，即x*(-1)
+x	x的本身
x**y	x的y次幂，即x^y

> 上表中操作符由Python解释器直接提供，不需要引用标准或第三方函数库，也叫做内置操作符。

操作符运算的结果可能改变数字类型，3种数字类型之间存在一种逐渐扩展的关系，具体为：

整数 -> 浮点数 -> 复数

> 这是因为整数可以看成是浮点数没有小数的情况，浮点数可以看成是复数虚部为0的情况。
> 基本规则如下：
>
> - 整数之间运算，如果数学意义上的结果是小数，结果是浮点数。
> - 整数之间运算，如果数学意义上是整数，结果是整数。
> - 整数和浮点数混合运算，输出结果是浮点数 。
> - 整数或浮点数与复数运算，输出结果是复数。

上表中所有二元数学操作符（+、-、*、/、//、%、* *）*都有与之对应的增强赋值操作符（*+=、-=、=、/=、//=、%==、**=）。

3.2.2 内置的数值运算函数

函数	描述
abs(x)	x的绝对值
divmod(x,y)	(x//y,x%y)，输出为二元组形式（也成为元组类型）
pow(x,y[,z])	(x**y)%z,[...]表示该参数可以省略，即pow(x,y)，它与 x**y相同
round(x[,ndights])	对x四舍五入，保留ndights位小数。round(x)返回四舍五入的整数值

|max(x_1,x_2,...,x_n) |x_1,x_2,...,x_n 的最大值，n没有限定
|min(x_1,x_2,...,x_n) |x_1,x_2,...,x_n 的最小值，n没有限定|

> abs()可以计算复数的绝对值，复数的绝对值是二维坐标系中复数位置到坐标原点的长度。

3.2.3 内置的数字类型转换函数

浮点数类型转换为整数类型时，小数部分会被舍弃（不使用四舍五入），附属不能直接转化为其他数字类型，可以通过.real和.imag将复数的实部或虚部分别转换，例如：

函数	描述
int(x)	将x转换为整数，x可以是浮点数或字符串
float(x)	将x转换为浮点数，x可以是整数或字符串
complex(re[,im])	生成一个复数，实部为re，虚部为im，re可以是整数、浮点数或字符串，im可以是整数或浮点数但不能为字符串

3.3 模块1：math库的使用

Python数学计算的标准函数库math共提供4个数学常数和44个函数。

3.3.1 math库概述

math库是Python提供的内置数学类函数库，不支持复数类型，仅支持整数和浮点运算。

math库的两种引用方式：

```
import math
```

```
from math import <函数名>
```

3.3.2 math库解析

math库的数学常数（4个）

常数	数学表示	描述
math.pi	Π	圆周率，值为3.141 592 653 589 793
math.e	e	自然对数，值为2.718 281 828 459 045
math.inf	∞	正无穷大，负无穷大为-math.inf
main.nan		非浮点数标记，NaN(Not a Number)

math库的数值表示函数（16个）

函数	数学表示	描述
math.fabs(x)	$\|x\|$	返回x的绝对值
math.fmod(x, y)	x % y	返回x与y的模
math.fsum([x,y,…])	x+y+…	浮点数精确求和
math.ceil(x)	$\lceil x \rceil$	向上取整，返回不小于x的最小整数
math.floor(x)	$\lfloor x \rfloor$	向下取证，返回不大于x的最大整数
math.factorial(x)	x!	返回x的阶乘，如果x是小数或负数，返回ValueError
math.gcd(a, b)		返回a与b的最大公约数
math.frepx(x)	$x = m * 2^e$	返回(m, e)，当x=0，返回(0.0, 0)
math.ldexp(x, i)	$x * 2^i$	返回$x * 2^i$运算值，math.frepx(x)函数的反运算
math.modf(x)		返回x的小数和整数部分
math.trunc(x)		返回x的整数部分
math.copysign(x, y)	$\|x\| * \|y\|/y$	用数值y的正负号替换数值x的正负号
math.isclose(a,b)		比较a和b的相似性，返回True或False
math.isfinite(x)		当x为无穷大，返回True；否则，返回False
math.isinf(x)		当x为正数或负数无穷大，返回True；否则，返回False
math.isnan(x)		当x是NaN，返回True；否则，返回False

math库的幂对数表示函数（8个）

函　数	数学表示	描　述
math.pow(x,y)	xy	返回 x 的 y 次幂
math.exp(x)	ex	返回 e 的 x 次幂，e 是自然对数
math.expml(x)	ex-1	返回 e 的 x 次幂减 1
math.sqrt(x)	$\sqrt{x}$	返回 x 的平方根
math.log(x[,base])	$\log_{base}x$	返回 x 的对数值，只输入 x 时，返回自然对数，即 $\ln x$
math.log1p(x)	$\ln(1+x)$	返回 $1+x$ 的自然对数值
math.log2(x)	$\log x$	返回 x 的 2 对数值
math.log10(x)	$\log_{10}x$	返回 x 的 10 对数值

math库的三角运算函数（16个）

函　数	数学表示	描　述
math.degree(x)		角度 x 的弧度值转角度值
math.radians(x)		角度 x 的角度值转弧度值
math.hypot(x,y)	$\sqrt{x^2+y^2}$	返回(x,y)坐标到原点(0,0)的距离
math.sin(x)	$\sin x$	返回 x 的正弦函数值，x 是弧度值
math.cos(x)	$\cos x$	返回 x 的余弦函数值，x 是弧度值
math.tan(x)	$\tan x$	返回 x 的正切函数值，x 是弧度值
math.asin(x)	$\arcsin x$	返回 x 的反正弦函数值，x 是弧度值
math.acos(x)	$\arccos x$	返回 x 的反余弦函数值，x 是弧度值
math.atan(x)	$\arctan x$	返回 x 的反正切函数值，x 是弧度值
math.atan2(y,x)	$\arctan y/x$	返回 y/x 的反正切函数值，x 是弧度值
math.sinh(x)	$\sinh x$	返回 x 的双曲正弦函数值
math.cosh(x)	$\cosh x$	返回 x 的双曲余弦函数值
math.tanh(x)	$\tanh x$	返回 x 的双曲正切函数值
math.asinh(x)	$\operatorname{arcsinh} x$	返回 x 的反双曲正弦函数值
math.acosh(x)	$\operatorname{arccosh} x$	返回 x 的反双曲余弦函数值
math.atanh(x)	$\operatorname{arctanh} x$	返回 x 的反双曲正切函数值

math库的高等特殊函数（4个）

函　数	数学表示	描　述
math.erf(x)	$\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$	高斯误差函数，应用于概率论、统计学等领域
math.erfc(x)	$\frac{2}{\sqrt{\pi}}\int_x^\infty e^{-t^2}dt$	余补高斯误差函数，math.erfc(x)=1 - math.erf(x)
math.gamma(x)	$\int_0^\infty x^{t-1}e^{-x}dx$	伽玛（Gamma）函数，也叫欧拉第二积分函数
math.lgamma(x)	ln(gamma(x))	伽玛函数的自然对数

3.4 实例3：天天向上的力量

这个实例主要运用math库里的pow函数来进行计算。题目为：一年365天，以第1天的能力值为基数，记为1.0，当好好学习时能力值相比前一天提高1‰。每天努力和每天放任，一年下来的能力值相差多少呢?
这里只放上了实例代码3.1，其它实sanyinh例代码类似，可自行了解。

```
#e3.1DayDayUp365.py
import math
dayup = math.pow((1.0 + 0.001),365)  #提高0.001
daydown = math.pow((1.0 - 0.001),365) #放任0.001
print("向上：{:.2f},向下：{:.2f}.".format(dayup,daydown))
```

运行结果：

> 向上：1.44,向下：0.69.

3.5 字符串类型及其操作

3.5.1 字符串类型的操作

字符串是字符的序列表示，可以由一对单引号（'）、双引号（"）或三引号（'''）构成。
3种表示方式如下：

- 单引号字符串：'单引号表示，可以使用"双引号"作为字符串的一部分'
- 双引号字符串："双引号表示，可以使用'单引号'作为字符串的一部分"
- 三引号字符串：'''三引号表示可以使用"双引号"
  '单引号'
  也可以换行
  '''

Python字符串提供区间访问方式，采用[N:M]格式,表示字符串中从N到M（**不包含M**）的子字符串，其中，N和M为字符串的索引符号，可以混合使用正向递增符号和反向递减符号。如果表示中M或者N索引缺失，则表示字符串把开始或结束索引值设置为默认值。

字符串以Unicode编码存储，因此，字符串的英文字符和中文字符都算作一个字符。

反斜杠字符（\）是一个特殊字符，在字符串中表示转义，即该字符与后面相邻的一个字符共同组成了新的含义。例如，\n表示换行、\\表示反斜杠、\'表示单引号、\"表示双引号、\t表示制表符（Tab）等。

3.5.2 基本的字符串操作

Python提供了5个字符串的基本操作符，如下表所示

操作符	描述
x + y	连接两个字符串x与y
x * n 或 n * x	复制n次字符串x
x in s	如果x是s的字串，返回True,否则返回False
str[i]	索引，返回第i个字符
str[N:M]	切片，返回索引第N到第M的字串，其中不包含M

与字符串操作符有关的实例如下：

```
>>>"Python 语言" + "程序设计"
'Python 语言程序设计'
>>>name = "Python 语言" + "程序设计" + "基础"
>>>name
'Python 语言程序设计基础'
>>>"GOAL!" * 3
'GOAL!GOAL!GOAL!'
>>>"Python 语言" in name
True
>>>'Y' in "Python 语言"
False
```

3.5.3 内置的字符串处理函数

Python解释器提供了一些内置函数，其中，有6个函数与字符串处理相关，如下表所示：

函数	描述
len(x)	返回字符串x的长度，也可返回其他组合数据类型元素个数
str(x)	返回任意类型x所对应的字符串形式
chr(x)	返回Unicode编码x对应的单字符
ord(x)	返回单字符表示的Unicode编码
hex(x)	返回整数x对应十六进制数的小写形式字符串
oct(x)	返回整数x对应八进制数的小写形式字符串

每个字符在计算机中可以表示为一个数字，称为编码。字符串则以编码序列方式存储在计算机中。

3.5.4 内置的字符串处理方法

在Python解释器内部，所有数据类型都采用面向对象方式实现，封装为一个类。字符串也是一个类。在面向对象中，这类函数被称为“方法”。字符串类型共包含43个内置方法。这里介绍16个较为常用的方法，如下表所示（其中str代表字符串或变量）：

方法	描述
str.lower()	返回字符串str的副本，全部字符**小写**
str.upper()	返回字符串str的副本，全部字符**大写**
str.islower()	当str所有字符都是**小写**时，返回True，否则返回False
str.isprintable()	当str所有字符都是**可打印的**，返回True，否则返回False
str.isnumeric()	当str所有字符都是**数字**时，返回True，否则返回False
str.isspace()	当str所有字符都是**空格**时，返回True，否则返回False
str.endswith(suffix[, start[,end]])	str[start:end]以suffix结尾时返回True，否则返回False
str.startswith(prefix[,start[,end]])	str[start:end]以prefix结尾时返回True，否则返回False
str.split(sep=None,maxsplit=-1)	返回一个列表，由str根据sep被分隔的部分构成
str.count(sub[,start[,end]])	返回str[start:end]中sub字串出现的次数

方法	描述
str.replace(old,new[,count])	返回字符串str的副本，所有old字串被替换为new，如果count给出，则前count次old出现被替换
str.center(width[,fillchar])	字符串居中函数
str.strip([chars])	返回字符串str的副本，在其左侧和右侧去掉chars中列出的字符
str.zfill(width)	返回字符串str的副本，长度为width，不足部分在左侧添0
str.format()	返回字符串str的一种排版格式
str.join(iterable)	返回一个新字符串，由组合数据类型iterable变量的每个元素组成，元素间用str分隔

3.6 字符串类型的格式化

字符串是程序向控制台、网络、文件等介质输出运算结果的主要形式之一，为了能更好的可读性和灵活性，字符串类型的格式化是运用字符串类型的重要内容之一。Python语言同时支持两种字符串格式化方法，一种类似C语言中printf()函数的格式化方法；另一种采用专门的str.format()格式化方法。本书所用例子均采用第二种方法。

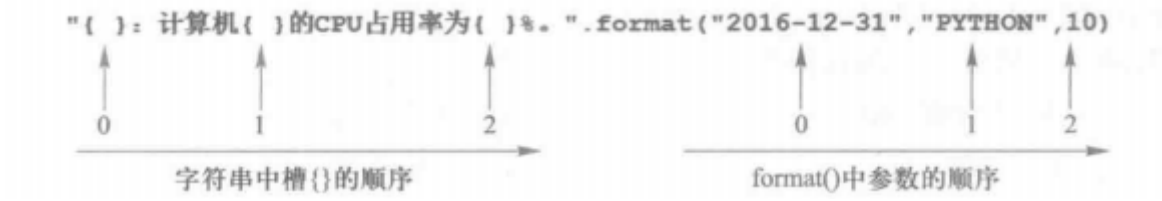
3.6.1 format()方法的基本使用

字符串format()方法的基本使用格式如下：

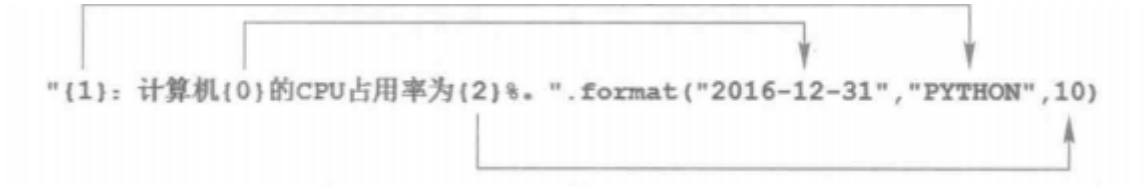
```
<模板字符串>.format(<逗号分隔的参数>)
```

模板字符串由一系列槽组成，用来控制修改字符串中嵌入值出现的位置，其基本思想是将format()方法中逗号分隔的参数按照序号关系替换到模板字符的槽中。

槽用大括号（{}）表示。
如果大括号中没有序号，则按照出现顺序替换，如下图所示。



如果大括号中指定了使用参数的序号，按照序号对应参数替换，如下图所示。



format()方法可以非常方便地连接不同类型的变量或内容，如果需要输出大括号，采用{{表示，}}表示。

3.6.2 format()方法的格式控制

format()方法中模板字符串的槽除了包括参数符号，还可以包括格式控制信息。此时，槽的内部样式如下：

```
{<参数符号>:<格式控制标记>}
```

其中，格式控制标记用来控制参数显示时的格式，格式内容如下图所示。

:	<填充>	<对齐>	<宽度>	<,>	<.精度>	<类型>
引导符号	用于填充的单个字符	< 左对齐 > 右对齐 ^ 居中对齐	槽的设定输出宽度	数字的千位分隔符 适用于整数和浮点数	浮点数小数部分的精度 或 字符串的最大输出长度	整数类型 b,c,d,o,x,X 浮点数类型 e,E,f,%

- <宽度>指当前槽的设定输出字符宽度。
- <对齐>指参数在宽度内输出时的对齐方式，分别用 < 、 > 和 ^ 3个符号表示左对齐、右对齐和居中对齐。
- <填充>指宽度内除了参数外的字符采用什么方式表示，默认采用空格 ，可以通过填充更换。
- <精度>表示两个含义，由小数点（.）开头。对于浮点数，精度表示小数部分输出的有效位数。对于字符串，精度表示输出的最大长度。
- <类型>表示输出整数和浮点数类型的格式规则。

  对于整数类型，输出格式包括输出整数的**二进制**、整数对应的**Unicode字符**、整数的**十进制**方式、整数的**八进制**方式、整数的**小写十六进制**方式和整数的**大写十六进制**方式六种格式。

  对于浮点数类型，输出格式包括输出浮点数对应的**小写字母e的指数形式**、浮点数对应的**大写字母E的指数形式**、浮点数的**标准浮点形式**、浮点数的**百分形式**四种格式。

> 浮点数输出时尽量使用<精度>表示小数部分宽度，有助于更好控制输出格式。

第四章 程序的控制结构

4.1 程序的基本结构

程序设计中引入控制结构的原因：单一的顺序结构无法解决所有问题。

程序的基本结构组成：顺序结构、分支结构、循环结构。

4.2 程序的分支结构

4.2.1 单分支结构：if

在 Python 中，`if` 语句的语法如下所示：

```
if <条件>:
  <语句块>
```

注意以下要点：

- 条件语句两边省略括号（虽然加上也没什么问题，但是标准起见请省略）；
- if 后面有一个英文冒号；
- if 成立后的语句块要与 if 语句本身隔开一个缩进。

在 Python 中，可用于条件判断部分的关系操作符有以下几个：

操作符	数学符号	操作符含义
`<`	$<$	小于
`<=`	$\leq$	小于或等于
`>`	$>$	大于
`>=`	$\geq$	大于或等于
`==`	$=$	等于
`!=`	$\neq$	不等于

和 C++ 等传统语言一样，`=` 指的是赋值，`==`表示逻辑运算“等于”。

> 提示：字符串的比较本质上是字符串对应 Unicode 编码的比较，由此会出现一些有趣的特性，如：字符串的比较将会按照字典序来进行，如：`"P" > "p"` 的结果是 `True`。

4.2.2 二分支结构：if-else

在 Python 中，`if-else` 语句的语法如下所示：

```
if <条件>:
  <语句块1>
else:
  <语句块2>
```

在这里同样要注意 `else` 后面有一个英文冒号。

当然，也可以用一行来写，这种方式适合通过判断返回特定值、对特殊值处理：

```
<表达式1> if <条件> else <表达式2>
```

4.2.3 多分支结构：if-elif-else

在 Python 中，`if-elif-else` 语句的语法如下所示：

```
if <条件1>:
  <语句块1>
elif <条件2>:
  <语句块2>
...
else:
  <语句块n>
```

最后面的 `else` 和 `语句块n` 也可以没有。

注意，在一般的编程语言中，多分支结构可以通过 `switch-case` 语句实现。但是，Python 不一般，它没有 `switch-case` 语句，因此 `if-elif-else` 是一种代替 `switch-case` 的方式。

4.3 程序的循环结构

4.3.1 遍历循环：for 语句

在 Python 中，`for` 语句的语法如下所示：

```
for <循环变量> in <遍历结构>:
  <语句块>
```

其中，遍历结构可以是字符串、文件、组合数据类型，或者是 `range()` 函数等。常用方法：

- 循环 N 次：`for i in range(N):`
- 遍历文件 fi 中的**每一行**：`for line in fi:`
- 遍历字符串 s：`for c in s:`
- 遍历列表 ls：`for item in ls:`

除此之外，`for` 语句还有一种扩展模式：

```
for <循环变量> in <遍历结构>:
  <语句块1>
else:
  <语句块2>
```

`else` 下的语句块会在循环 **正常退出** 时执行。更多内容将会在后面 `continue` 和 `break` 那里讲解。

4.3.2 无限循环：while 语句

在 Python 中，`while` 语句的语法如下所示：

```
while <条件>:
  <语句块>
```

这里的“条件”与 `if` 语句中的一样。执行顺序也与普通的语言一样，先判断一次 `while` 上的条件，然后在开始执行语句块。

与 `for` 相同，`while` 也支持扩展模式：

```
while <条件>:
  <语句块1>
else:
  <语句块2>
```

4.3.3 循环保留字：break 和 continue

`break` 和 `continue` 可以用于辅助控制循环执行。

`break` 可以跳出 **最内层（当前层次）** 的 `for` 或者 `while` 循环，并从循环结束的地方继续运行。

`continue` 可以结束 **当前层次** 循环，即跳过本层循环体重尚未执行的语句：

- 对于 `while` 循环，程序流程继续求解循环条件；
- 对于 `for` 循环，程序流程接着遍历循环列表。

关于 `else` 用在循环中时，有以下规则：

- 循环正常执行完成时，会执行 `else` 中的代码块；
- 循环被 `break` 或 `return` 打断时，不会执行 `else` 中的代码块；
- 循环中使用 `continue` 不影响最终 `else` 中的代码块的执行。

4.4 模块2：random 库的使用

4.4.1 random 库概述

使用 random 库的目的主要是生成随机数，其采用的是梅森旋转算法（Mersenne Twister）来生成伪随机数序列。不过，对于随机性要求更高的应用场景，如加密、解密计算，这样的伪随机数可能无法满足要求。

4.4.2 random 库解析

random 库中常用的随机数生成函数如下：

函数	描述
seed(a=None)	初始化随机数种子，默认值为当前系统时间
random()	生成一个[0.0, 1.0)之间的随机小数
randint(a, b)	生成一个[a, b]之间的整数
getrandbits(k)	生成一个 k 比特长度的随机整数
randrange(start, stop[, setp])	生成一个[start, stop)之间以 step 为步数的随机整数
uniform(a, b)	生成一个[a, b]之间的随机小数
choice(seq)	从序列类型（如列表）中随机返回一个元素
shuffle(seq)	将序列类型中的元素随机排列，返回打乱后的序列
sample(pop, k)	从 pop 类型中随机选取 k 个元素，以列表类型返回

引用方式：

```
import random
```

或者

```
from random import *
```

在调试的时候，我们可以通过 seed() 指定随机种子，一般是一个整数，只要种子相同，每次生成的随机数序列也就相同，这样便于测试和同步数据。

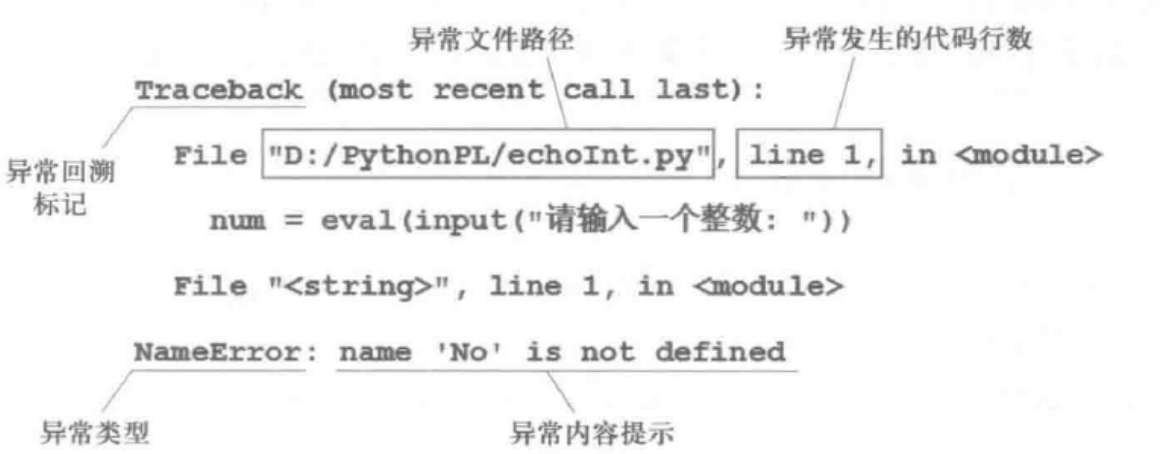
4.5 程序的异常处理

4.5.1 程序处理：try-except 语句

异常一般是这样的，比如说你写了一个语句，用来计算平方：

```
num = eval(input("请输入一个整数"))
print(num ** 2)
```

是这样没错，但是万一你的用户使坏不给你数字呢？Boom！异常信息甩在了你的脸上：



Python 解释器返回异常信息，并退出程序。

那么这么看来，程序认为执行过程中出了错，所以丢下异常跑了。但是有时候我们不希望它跑路，而是遇到错误以后回来执行其他东西，怎么办？

其实，只要把异常“捕获”就好啦！

要想捕获异常，首先应该知道异常类型，它表明异常发生的原因，也是程序处理异常的依据。然后，我们根据异常类型，使用如下语句：

```
try:
  <语句块1>
except <异常类型>:
  <语句块2>
```

对于刚才的程序，可以修改成如下形式：

```
try:
  num = eval(input("请输入一个整数"))
  print(num ** 2)
except NameError:
  print("输入错误，请输入一个整数！")
```

其实这里，`except` 关键字对标的就是其他语言中的 `catch`。

4.5.2 异常的高级用法

`try-except` 支持多个 `except` 语句，格式如下：

```
try:
  <语句块1>
except <异常类型1>:
  <语句块2>
...
except <异常类型n>:
  <语句块n+1>
except:
  <语句块n+2>
```

另外，类似于 `for` 和 `while` 中的 `else` 语句，在 `try-except` 语句执行完成后，可以通过 `else` 或者 `finally` 来执行一段后续代码：

```
try:
  <语句块1>
except <异常类型1>:
  <语句块2>
else:
  <语句块3>
finally:
  <语句块4>
```

`else` 语句与 `for` 和 `while` 中的一样，try 中 **正常结束且没有发生异常** 时，`else` 中的语句执行。被 `return` 后是不会执行 `else` 的。

`finally` 语句则“管得更宽”，不管有没有异常都会执行。如果有 `return` 语句出现在 `try-except` 语句中时，`finally` 中的内容也会在 `return` 执行前抢先执行。

一般来讲，`try-except` 语句只用来检测极少发生的情况，比如用户输入的合规性或者文件打开是否成功。其他经常容易出现的错误建议使用 `if` 直接判断。

以我个人的理解，`try-except` 是定义了“错误”，是超乎程序预期的部分。本质上来讲，它也是一种分支结构，但是能与正常的业务逻辑代码区分开来。如果滥用 `try-except`，反而会导致整个程序结构混乱，还有可能造成各种奇怪的错误。

那么，什么时候应该使用异常语句呢？emmmm。。。想用就用吧，用户体验才是王道。

第五章 函数和代码复用

在本章对于第四节,第七节的示例和第五节的复用规范以及第八节的内置函数则希望大家能够自行学习并应用在日常代码编写中

5.1 函数的基本使用

5.1.1函数的定义

函数是一段具有特定功能的可重复使用的语句组

我们可以在任何需要的地方调用执行而不用重复编写这些语句，以最简单的温度转换为例：

```
def convertTemp(temp, scale):
  if scale == "c":
   return (temp - 32.0) * (5.0/9.0)
  elif scale == "f":
   return temp * 9.0/5.0 + 32
temp = int(input("Enter a temperature: "))
scale = input("Enter the scale to convert to: ")
converted = convertTemp(temp, scale)
print("The converted temp is: " + str(converted))
```

当我们编写了这样一个语句组时，便可以把他定义为一个简单的函数以此来实现对它的复用。我们可以在一个程序或多个程序中多次调用这个函数，而当需要对它修改时，只需要修改一次，所有调用位置的功能都更新了。这种高效的复用是使用函数的两大目的之一。

而使用函数的另一大目的是降低编程难度。函数是一种功能抽象，利用它可以将一个复杂的大问题分解成一系列简单的小问题，然后将小问题继续细分，当问题被划分到足够简单时，便可以将每个小问题分别封装成函数来处理。

而在Python中，我们使用

```
def<函数名>(<参数列表>);
<函数体>
return<返回值列表>
```

函数名可以是任何有效的Python标识符；参数列表是调用改函数时传递给它的值，可以是任意多个。当有多个参数时用逗号分隔（参见温度转换）。函数体里是每次被调用时执行的代码，当需要返回值时，使用保留字return和返回值列表，否则函数值可以没有return语句，在函数体结束位置将控制权返回给调用者。

函数调用和执行的一般形式如下：
<函数名>(<参数列表>)

5.1.2 函数的调用过程

程序调用一个函数需要执行以下四个步骤：

- 调用程序在调用处暂停执行
- 在调用时将实参复制给函数的形参
- 执行函数体语句
- 函数调用结束给出返回值，程序回到调用前的暂停处继续执行

```
import sys
import random
def compareNum(num1,num2):#定义一个函数
    if(num1>num2):
        return num1
    elif(num1==num2):
        return num1
    else:
        return num2
if __name__=="__main__":
    num1=random.randrange(1,99)
    num2=random.randrange(1,99)#随机获取1-99的一个数
    print compareNum(num1,num2)#在此处传递实参，传递结束后返回正常值
    print "num1=",num1
    print "num2=",num2
```

以上面的程序举例，在程序执行到Print compareNum()时，便用实参num1，num2替换形参，随后用实参代替形参执行函数体内容，当函数执行完毕后，重新回到断点，继续执行剩余的语句。

5.1.3 lambda函数

有些函数如果只是临时一用，而且它的业务逻辑也很简单（比如做个简单加法、取绝对值、简单过滤等）时，就没必要非给它取个名字。在做大的Python项目开发中，过多的函数名会影响代码的可读性。

匿名函数并非没有名字，而是将函数名作为结果返回，语法格式如下：

```
<函数名> = lambda <参数列表>:<表达式>
```

其等价于上文定义的函数

5.2 参数的传递

5.2.1 可选参数和可变数量参数

在定义函数时，如果部分函数不一定需要调用程序输入，那么可以为这些参数直接指定默认值，如果没有传入对应的参数值则使用默认值代替。

```
def func(a, b=5, c=10):
    print('a is', a, 'and b is', b, 'and c is', c)

func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

由于函数调用时需要按顺序输入参数，所以可选参数必须定义在非可选函数的后面。

在函数定义时，也可以设计可变数量参数，通过在参数前增加星号（*）实现。带有星号的可变参数只能出现在参数列表的最后。调用时，这些参数被当作元组类型传入函数。

```
## 例子
def total(a=5, *numbers, **phonebook):
    print('a', a)

    #遍历元组中的所有项目
    for single_item in numbers:
        print('single_item', single_item)

    #遍历字典中的所有项目
    for first_part, second_part in phonebook.items():
        print(first_part,second_part)

print(total(10,1,2,3,Jack=1123,John=2231,Inge=1560))
```

5.2.2 参数的位置和名称传递

在调用函数时通常我们按照位置顺序传递参数，但当参数很多，这种调用参数的方式可读性较差。亦或者是在大型工程中函数可能定义在函数库中，也可能与调用距离很远，可读性较差。这是我们就可以使用按形参命名输入实参的方式。举个栗子：假设我们定义一个func的函数，第一段程序是他的定义，而第二段程序是距离他还很远的调用。此时我们就可以按照形参名称输入实参。

```
func(x1,y1,z1,x2,y2,z2):
return
…
result = func(4,2,3,5,9,0)
…
result = func(x2=4,x1=2,y2=3,z1=5,z2=9,y1=0)
```

由于指定了参数名称，所以顺序可以任意调整。

5.2.3 函数的返回值

return语句用来退出函数并将程序返回到函数被调用的地方继续执行。return语句可以同时将0个，1个或多个函数运算后的结果返回给函数被调用处的变量。函数可以没有return，此时函数并不返回值。

5.2.4 函数对变量的作用

Python函数对变量的作用遵守如下原则：

- 简单数据类型无论是否与全局变量重名，仅在函数内部创建和使用，函数退出后变量被释放。如有全局同名变量，值保持不变。
- 简单数据类型在用global保留字声明后作为全局变量使用，函数退出后该变量保留值且值被函数改变。
- 对于组合数据类型的全局变量，如果在函数内部没有被真实创建的同名变量，则函数内部可以直接使用并修改全局变量的值。
- 如果函数内部真实创建了组合数据类型变量，无论是否有同名全局变量，无论是否有同名全局变量，函数仅对局部变量进行操作，函数退出后局部变量被释放，全局变量值不变。

5.3 datetime库

datetime库以类的方式提供多种日期和时间表达式。

- (1)datetime.date:日期表达类，可以表达年、月、日等。
- (2)datetime.time:时间表达类，可以表示小时、分钟、秒、毫秒等。

- (3)datetime.datetime:日期和时间表示的类，功能覆盖date和time类。
- (4)datetime.timedelta:与时间间隔有关的类。
- (5)datetime.tzinfo:与时区有关的信息表达类。

datetime库解析

detatime.now()获得当前日期和时间对象。

datetime.now()

作用：返回一个datetime类型，表示当前的日期和时间，精确到微秒。

参数：无

调用该函数，执行结果如下：

```
>>> from datetime import datetime
>>> today = datetime.now()
>>> today
datetime.datetime(2019,7,6,17,35,39,758490)
```

datetime.utcnow()获得当前日期和时间对应的UTC（世界标准时间）时间对象：

datetime.utcnow()

作用：返回一个datetime类型，表示当前的日期和时间的UTC表示，精确到
微秒。

参数：无

调用该函数执行结果如下：

```
>>> from datetime import datetime
>>> today = datetime.utcnow()
>>> today
datetime.datetime(2019,7,6,9,53,26,814729)
```

直接使用datetime()构造一个日期和时间对象：

```
datetime(year,month,day,hour=0,minute=0,second=0,microsecond=0)
```

作用：返回一个datetime类型，表示指定的日期和时间，可以精确到微秒。
参数如下。

year：指定的年份，MINYEAR <= YEAR <= MAXYEAR

month：指定的月份，1 <= month <= 12

day：指定的日期，1 <= day <= 月份所对应的日期上限

hour：指定的小时，0 <= hour <= 24

minute：指定的分钟数，0 <= minute < 60

second：指定的秒数，0 <= second < 60

microsecond：指定的微秒数，0 <= microsecond <1000000

其中，hour、minute、second、microsecond参数可以全部或部分忽略。
调用datetime创建一个datetime对象，执行结果如下：

```
>>> from datetime import datetime
>>> someday = datetime(2019,7,6,18,11,57,9)
>>>someday
datetime.datetime(2019,7,6,18,11,57,9)
```

5.4 递归

程序调用自身的编程技巧称为递归（ recursion）。

递归做为一种算法在程序设计语言中广泛应用。 一个过程或函数在其定义或说明中有直接或间接调用自身的一种方法，它通常把一个大型复杂的问题层层转化为一个与原问题相似的规模较小的问题来求解，递归策略只需少量的程序就可描述出解题过程所需要的多次重复计算，大大地减少了程序的代码量。递归的能力在于用有限的语句来定义对象的无限集合。一般来说，递归需要有边界条件、递归前进段和递归返回段。当边界条件不满足时，递归前进；当边界条件满足时，递归返回。

给大家举个最简单的例子：

一个洋葱是一个带着一层洋葱皮的洋葱。

通过以上的介绍，我们大致可以总结出递归的以下几个特点：

- 1、必须有一个明确的结束条件
- 2、每次进入更深一层递归时，问题规模(计算量)相比上次递归都应有所减少
- 3、递归效率不高，递归层次过多会导致栈溢出（在计算机中，函数调用是通过栈（stack）这种数据结构实现的，每当进入一个函数调用，栈就会加一层栈帧，每当函数返回，栈就会减一层栈帧。由于栈的大小不是无限的，所以，递归调用的次数过多，会导致栈溢出）

关于递归还有两个名词，可以概括递归实现的过程

递推：像上边递归实现所拆解，递归每一次都是基于上一次进行下一次的执行，这叫递推

回溯：则是在遇到终止条件，则从最后往回返一级一级的把值返回来，这叫回溯

下面给大家给出两个递归示例，鉴于在之前的Java和c++中大家都已经学习过递归的思想，我们就不再着重讨论。

递归求阶乘

```
# 1! +2! +3! +4! +5! +...+n!
def factorial(n):
    ''' n表示要求的数的阶乘 '''
    if n==1:
        return n # 阶乘为1的时候，结果为1,返回结果并退出
    n = n*factorial(n-1) # n! = n*(n-1)!
    return n  # 返回结果并退出
res = factorial(5) #调用函数，并将返回的结果赋给res
print(res) # 打印结果
```

递归求斐波那契数列

```python
# 1，1，2，3，5，8，13，21，34，55，试判断数列第十五个数是哪个？
def fabonacci(n):
    ''' n为斐波那契数列 '''
    if n <= 2:
        ''' 数列前两个数都是1 '''
        v = 1
        return v # 返回结果，并结束函数
    v = fabonacci(n-1)+fabonacci(n-2) # 由数据的规律可知，第三个数的结果都是前两个数之和，所以进行递归叠加
    return v  # 返回结果，并结束函数
print(fabonacci(15)) # 610     调用函数并打印结果
```

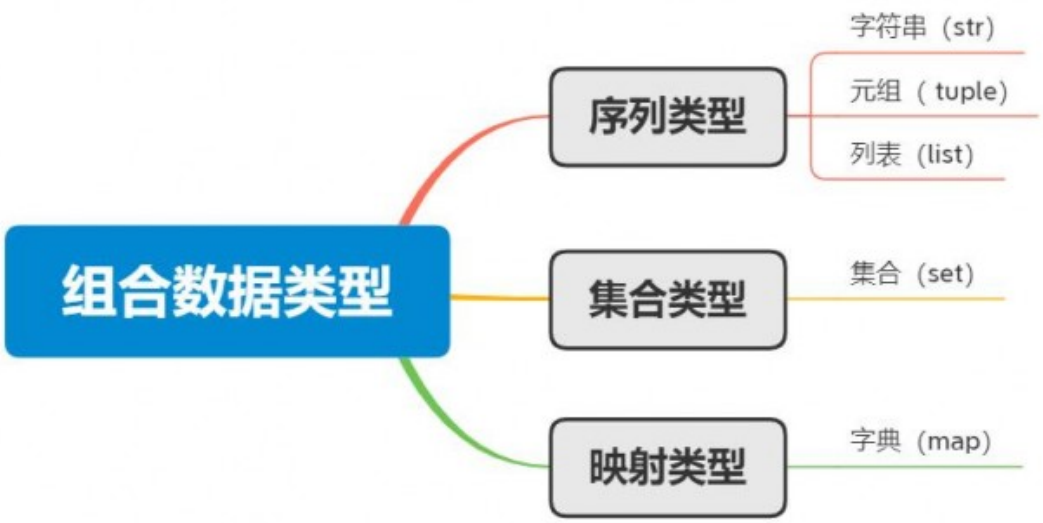
第六章 组合数据类型

6.1 本章概述

计算机不仅对单个变量表示的数据进行处理，更多的时候需要对一系列数据进行批量处理，本章我们介绍组合数据类型。组合数据类型为多个同类型或者不同类型数据提供单一表示。组合数据类型分为三类：序列类型、集合类型和映射类型。

6.2 组合数据类型概述

组合数据类型的分类如下：



序列类型是一个元素向量，元素之间存在先后关系，通过序号访问，元素之间不排他。
集合类型是一个元素集合，元素之间无序，集合中元素唯一存在。
映射类型是“键-值”对数据的组合，每个元素都是一个包含key和value的键值对。

6.2.1 序列类型

序列类型是一维向量元素，元素之间存在先后关系，通过序号访问。由于存在先后关系，所以可以存在相同内容但是位置不同的元素。

序列类型使用相同的索引体系，即正向递增和反向递减。如下图：

字符串是 Python 中最常用的数据类型。我们可以使用引号('或 ")来创建字符串。例如：

```
tmp="python"
tmp2='python2'
```

列表可以作为一个方括号内的逗号分隔值出现。列表的数据项不需要具有相同的类型。例如：

```
list1 = ['python', 'Abc', 2020]
list2 = [1, 2, 3, 4, 5 ]
list3 = [['a', 'b', 'c'], [1, 2, 3]]
#对于嵌套情形可以使用多级索引
print(list3[0])
#输出结果为['a', 'b', 'c']
print(list3[0][1])
#输出结果为b
```

Python 的元组与列表类似，不同之处在于元组的元素**不能修改**。列表使用方括号，元组使用小括号。创建元组只需要在括号中添加元素，并使用逗号隔开即可。但括号在不混淆语义的情况下不是必需的。例如：

```
tmp="Cat","Dog"
tmp2=("ABC",2020,('blue','red'))
#对于嵌套情形可以使用多级索引
print(tmp2[2])
#输出结果为('blue','red')
print(tmp2[2][1])
#输出结果为red
```

序列都可以进行的操作包括索引，切片，加，乘，检查成员。下面介绍序列类型12个通用的操作符和函数。

操作符	**描述**
x in s	如果x是序列s的元素，返回True，否则返回False
x not in s	如果x是序列s的元素，返回False，否则返回True
s + t	连接两个序列s和t
s*n 或 n*s	将序列s复制n次
s[i]	索引，返回s中的第i个元素，i是序列的序号
s[i: j]	切片，返回序列s中第i到j个元素子序列（不含第j个）
s[i: j: k]	步骤切片，返回序列s中第i到j个以k为步数的子序列
len(s)	返回序列s的长度
min(s)	返回序列s的最小元素， s中元素需要可比较
max(s)	返回序列s的最大元素， s中元素需要可比较
s.index(x)或s.index(x, i, j)	返回序列s从i开始到j位置中**第一次**出现元素x的位置
s.count(x)	返回序列s中出现x的总次数

6.2.2 集合类型

集合类型的含义与数学上的同名概念一致。包含0个或多个数据的**无序**组合，**元素不重复**，元素类型只能是固定的数据类型，如整数、浮点数、字符串等。

> tips:
> python中判定是否是固定数据类型主要考察类型是否能够进行哈希运算。能够运行哈希运算的类型都可以作为集合元素。
> **哈希运算**可以将任意长度的二进制映射为较短的固定长度的二进制值，这个二进制值称为哈希值。

前面提到过，集合是无序组合，它没有索引和位置的概念，不能分片，集合中的元素可以进行添加和删除。集合用大括号表示，可以使用大括号生成一个集合。例如：

```
tmp={425,"python",(10,"SDU"),2020}
```

set()函数也可以用来生成集合，输入可以是任何组合数据类型，输出是无序且无重复的集合。例如：

```
t=set((425,"python",(10,"SDU"),2020,425))
print(t)
#输出
#{425, 2020, 'python', (10, 'SDU')}
```

注意：由于集合是无序的，所以集合的打印效果与定义的顺序可能不一致。

下面给出集合类型的10个操作符：

操作符	描述
S-T	返回一个新的集合，这个集合包含在S中但不在T中的所有元素
S-=T	更新集合S，包括在S中但不在T中的所有元素
S&T	返回一个新集合，这个集合中的元素既在S中又在T中
S&=T	更新集合S，包括既在S中又在T中的元素
S^T	返回一个新集合，这个集合中的元素包括S与T中非相交的元素。
S^=T	更新集合S，包括集合S与T中非相交的元素。
S | T	返回一个新集合，这个集合包含S与T中的所有元素。
S | =T	更新集合S，包括S与T中的所有元素。
S<=T	判断S是否是T的子集关系。如果是，返回True，否则返回False。
S>=T	判断S是否是T的超集关系。如果是，返回True，否则返回False。

下面介绍了10个集合类型的操作函数：

操作函数	描述
S.add(x)	如果数据项x不在集合S中，将x增加到S
S.clear()	清除集合S中的所有元素
S.discard(x)	移除集合S中的元素x，就算S中没有x也不会报错
S.remove(x)	移除集合S中的元素x,如果S中没有x，则会报错KeyError
S.pop()	从集合中随机返回S的一个值。若S为空，则产生KeyError错误
S.copy()	返回集合S的一个副本
len(S)	返回集合S中的元素的数目
x in S	判断元素x是否包含在集合S中
x not in S	判断元素x是否不在集合S中
S.isdisjoint(T)	如果集合S和T没有相同元素，返回True。

6.2.3 映射类型

映射类型是“键-值”数据的组合，每个元素是一个键值对，即元素是（key,value），元素之间是无序的。键值对是一种二元关系，键表示一个属性，值是属性的内容。6.4节我们将深入介绍。

6.3 列表类型和操作

6.3.1 列表类型的概念

列表(list)是包含0个或多个对象引用的有序序列，属于序列类型。长度可变，支持增加、删除和替换，没有长度限制，元素类型可以不同。
列表用中括号表示，也可以用list()函数将元组或字符串转化为列表。例如：

```
t1=[2020,"python",[10,"SDU"],2020]
#[2020,'python',[10,'SDU'],2020]
t2=list((2020,"python",[10,"SDU"],2020))
#[2020,'python',[10,'SDU'],2020]
t3=list("列表类型的概念")
#['列','表','类','型','的','概','念']
```

列表之间的赋值操作必须显式进行。简单的将一个列表赋值给另一个列表不会生成新的列表对象。例如：

```
l=[2020,"SDU"]
t=l
#此时t成为列表l的一个引用，两者均指向同一份数据
l[0]=20
print(t)
#输出[20,"SDU"]
```

6.3.2 列表类型的操作

函数或方法	描述
ls[i]=x	替换ls第i项为x
ls[i:j]=lt	用列表lt替换列表ls中第i到j项数据（不含第j个）
ls[i:j:k]=lt	用列表lt替换列表ls中第i到j项以k为步数的数据（不含第j个）
del ls[i:j]	删除列表ls第i到第j项数据，等价于ls[i:j]=[]
del ls[i:j:k]	删除列表ls第i到第j项以k为步数的数据
ls+=lt	将列表lt元素增加到列表ls中
ls*=n	更新列表ls，其元素重复n次
ls.append(x)	在列表最后添加元素x
ls.clear()	删除所以元素
ls.copy()	生成一个新列表，赋值ls所有元素
ls.insert(i,x)	在列表ls第i位置增加元素x
ls.pop(i)	将ls第i元素取出并删除该元素
ls.remove(x)	将列表中出现的第一个元素x删除
ls.reverse()	列表中的元素反转

6.4 字典类型和操作

6.4.1 字典类型的概念

python中使用字典实现映射关系。字典中键值对之间没有顺序，不能重复，简单讲字典可以说是元素为键值对的集合。通过大括号建立，建立模式如下：

{<键1>:<值1>,<键2>:<值2>,...,<键n>:<值n>}
例如：

```
dict={"中国":"北京","法国":"巴黎","美国":"华盛顿"}
#关于国家和首都映射关系的字典
```

直接使用大括号可以创建一个空的字典，而不是集合。生成空集合需要使用set()函数。

字典通过字典变量和键进行访问和修改：

```
dict={"中国":"北京","法国":"巴黎","美国":"华盛顿"}
s=dict["中国"]
print(s)
#输出 北京
dict["中国"]="大北京"
print(dict["中国"])
#输出 大北京
```

6.4.2 字典类型的操作

下面介绍一些字典操纵的函数和方法：

函数和方法	描述
dict.keys()	返回一个包含字典中键的列表
dict.values()	返回一个包含字典中所有值的列表
dict.items()	返回一个包含字典中(键, 值)对元组的列表
dict.get(key,default)	若键存在,返回它对应的值value，如果字典中不存在此键，则返回default的值
dict.pop(key[, default])	如果字典中key 键存在，删除并返回dict[key]，否则返回默认值
dict.popitem()	在字典中随机取出一个键值对，以元组(key,value)形式返回
dict.clear()	删除字典中所有元素
del dict[key]	删除字典中某一键值对
key in dict	如果键在字典中返回True，否则返回False

字典可以通过for-in语句进行元素遍历。例如：

```
dict={"中国":"北京","法国":"巴黎","美国":"华盛顿"}
for key in dict:
    print(dict[key])
'''输出
北京
巴黎
华盛顿
'''
```

6.5 模块4：jieba库的使用

6.5.1 jieba库概述

jieba库是Python中一个重要的第三方中文分词函数库。其功能是，将一个中文句子，运用比较智能的方式拆分为符合人们认知的一系列词语。jieba库的原理是利用一个中文词库，将待分词的内容与词库进行比对，通过图结构和动态规划方法找到最大概率的词组。

jieba库支持三种分词模式：
1.精确模式。将句子精确分开，适合文本分析。
2.全模式。将句子中所有可以成词的成语快速扫描出来，但不能消除歧义。
3.搜索引擎模式。在精确模式基础上，对长词再次切分。

注意jieba库不是python自带库，而是第三方库，需要pip指令安装。

```
pip install jieba
#或者
pip3 install jieba
```

6.5.2 jieba库解析

jieba库常用分词函数：

函数	描述
jieba.cut(s)	精确模式，返回一个可迭代的数据类型
jieba.cut(s,cut_all=True)	全模式，输出文本s中所有可能的单词
jieba.cut_for_search(s)	搜索引擎模式，适合搜索引擎建立索引的分词效果
jieba.lcut(s)	精确模式，返回一个列表类型，建议使用
jiaba.lcut(s,cut_all=True)	全模式，返回一个列表类型，建议使用
jieba.lcut_for_search(s)	搜索引擎模式，返回一个列表类型，建议使用
jieba.add_word(w)	向分词词典增加新词w

第七章 文件和数据格式化

通过前面的学习，我们已经掌握了 Python 语言最基本的语法和功能，那么下面我们要讲的东西就跟数据的存储有很大关系了。你可曾想过，几行不起眼的代码，也能够编辑图片呢？你可曾想过，你自己用 Python 写的代码，竟然能编辑和生成与 Excel 兼容的文件呢？本章知识见解将会带你走进 Python 的文件世界。在这里，我们将传授你有关 Python 文件输入输出的秘诀，并为你详细讲述几种主流的标准文件格式。话不多说，让我们一同走进本章的知识见解吧！

7.1 文件的使用

7.1.1 文件概述

相信你对于文件一点也不陌生了吧？你交的每一次实验报告，你每天填的健康表格，不都是文件吗？

你要知道，文件里面是可以存放各种数据的。那么，这跟我们的 Python 程序的“输入输出数据”有什么样的区别和联系呢？

我们自己写 Python 的时候，经常会要求用户输入一些数据，并基于输入内容进行一些运算。例如第一个输入要求是一个数字，第二个输入要求是一个字符串，最后我们根据输入进行运算，然后再输出到屏幕上。

有时候为了调试程序，我们难免要重复运行它，每次运行都要输入重复的内容。那么这些数据我可否先存储到一个文件里，然后自动读取它呢？

回答了上一个问题，我们再来联想一下，一个`.docx`文件，被 Word 程序打开，这是不是可以说 Word 这个程序的运行需要以一个`.docx`作为输入，并读取里面的数据呢？

这么看来，文件中的数据确实可以被 Python 程序输入，而且 Python 程序的运行结果也确实可以输出到文件中。那么，怎么做呢？

7.1.2 文件的打开关闭

文件就像一个盒子。开始的时候是密封状态，存放在磁盘上。我们如果想读取文件中的数据，就必须把这个包装“打开”：

```
<变量名> = open(<文件名>, <打开模式>)
```

文件名可以是一个路径，告诉 Python，我要打开的盒子到底存放在哪里。而打开模式就是说：我拿到这个盒子以后我要怎么用它，我是单纯的打开看看（只读），还是我要往里面放东西（写）呢？

下面的表格列举了文件的打开模式及其含义：

文件的打开模式	含义
'r'	只读模式，如果文件不存在，则返回异常 `FileNotFoundError`。不指定打开模式则默认为此模式
'w'	覆盖写模式，如果文件不存在则创建，存在则完全覆盖
'x'	创建写模式，如果文件不存在则创建，存在则返回异常 `FileExistsError`
'a'	追加写模式，如果文件不存在则创建，存在则在文件最后追加内容
'b'	二进制文件模式
't'	文本文件模式，默认值
'+'	与 r w x a 一起使用，在原功能基础上增加同时读写功能

打个比方：

```
textfile = open('7.1.txt', 'rt')
```

这就是指定要打开 `7.1.txt` 文件，打开方式为文本文件模式，只读，并把这个文件的代表对象存放到 `textfile` 变量中。

当然，用文件需要打开“盒子”，自然也需要关闭。一般的操作系统中，一个文件在同一时间只能被一个程序使用，这就要求你尽快使用，用完尽快还回去。

通常来讲，Python 程序在结束时会自动帮我们关闭已经打开的文件。不过我们也可以调用以下函数来手动关闭文件：

```
<变量名>.close()
```

7.1.3 文件的读写

打开了文件，我们就可以按照我们想要的方式来阅读它了。

你可以打开一个 `.docx` 文件，并查看它的内部。你打开可能会看到许多乱码，这是因为 Word 采用了二进制的方式存储数据，而它的 Word 程序则包含了读取文件的程序段。

我们自己的文件也一样，需要定义好对应的格式，并保证输入和输出都按照我们定义的格式有序进行。

我们的 Python 提供了以下几种方式来进行文件的读取：

操作方法	含义
.readall()	读入整个文件内容，返回一个字符串或者字节流*
.read(size=-1)	从文件中读入整个文件内容，如果给出参数，读入前 size 长度的字符串或者字节流
.readline(size=-1)	从文件中给读入一行内容，如果给出参数，读入该行前 size 长度的字符串或者字节流
.readlines(hint=-1)	从文件中读入所有航，以每行为元素形成一个列表，如果给出参数，读入 hint 行

*：字符串或字节流取决于文件的打开模式，如果是以文本方式打开（t），则返回字符串，否则（b）返回字节流，下同。

同样的，有读入也可以有输出。以下表格提供了 Python 的文件输出函数：

操作方法	含义
.write(s)	向文件写入一个字符串或者字节流
.writelines(lines)	将一个元素全为字符串的列表写入文件
.seek(offset)	改变当前文件操作指针的位置，offset的值：0 - 文件开头，1 - 当前位置，2 - 文件结尾

7.2 PIL 库的使用

7.2.1 什么是 PIL 库?

PIL （Python Image Library）是一个第三方的 Python 语言库。

要安装 PIL，请在控制台使用以下命令：

```
pip install pillow
```

或者

```
pip3 install pillow
```

PIL 库支持各种对图像的操作，例如存储、读取、显示、处理。

7.2.2 图像类——Image类

7.2.2.1 概述

Image 类是 PIL 库中非常重要的类，一个 Image 对象代表一个图片，引入方式如下：

```
from PIL import Image
```

在 PIL 中，任何一个图像文件都可以用 Image 对象表示。

7.2.2.2 图像的读取与创建

Image 对象的读取和创建函数如下表：

方法	描述
Image.open(filename)	根据参数加载图像文件
Image.new(node, size, color)	根据给定参数创建一个新的图像
Image.open(StringIO.StringIO(buffer))	从字符串中获取图像
Image.frombytes(mode, size, data)	根据像素点 data 创建图像
Image.verify()	对图像文件的完整性进行检查，返回异常

例如：

```
im = Image.open("D:\\pycodes\\birdnest.jpg")
```

这里说明一下，由于反斜杠 `\` 在 Python 中具有特殊的含义，因此需要使用两个反斜杠。实际上解析出来以后，两个反斜杠会被解析成一个。

在上面的代码中，我们使用的是绝对路径，这就非常具体地指定了一个文件的位置。但是这有个问题，如果我把这个 Python 程序拿到其他电脑上去用，那么我必须保证文件在另外一台电脑上的存储位置完全相同，这就有点难了，而且操作起来也不是很方便。有没有一种办法能够直接指定一个路径，能够随着 Python 程序的存储位置进行改变呢？

答案是可以的，使用相对路径即可：

```
im = Image.open("data\\birdnest.jpg")
```

在这里，我们只需要在程序的目录下建立一个 `data` 文件夹，然后把 `birdnest.jpg` 文件放进去就可以了。你可以发现，这里的“相对”指的是相对于程序文件所在目录的位置。

7.2.2.3 文件的保存与缩略图的制作

能打开，就能保存。

使用 Image 对象的 `save()` 函数就可以保存图片：

```
im.save('processed.jpg')
```

当然，我们还可以生成一个缩略图：

```
im.thumbnail((128, 128))
```

其中，128 是指定的缩略图尺寸。

7.2.2.4 图像的旋转和缩放

Image 类可以对图像进行旋转和缩放，方法如下表：

方法	描述
Image.resize(size)	按 size 大小调整图像，生成副本
Image.rotate(angle)	按 angle 角度旋转图像，生成副本

7.2.2.5 图像的像素和通道处理

Image 类还可以进行像素处理和通道处理这样的高级方法，如下：

方法	描述
Image.opint(func)	根据函数 func 的功能对每个元素进行运算，返回图像副本
Image.split()	提取 RGB 图像每个颜色通道，返回图像副本
Image.merge(mode, bands)	合并通道，其中 mode 表示色彩，bands 表示新的色彩通道
Image.blend(im1, im2, alpha)	将两幅图片 im1 和 im2 按照如下公式插值后生成新的图像：`im1 * (1.0 - alpha) + im2 * aplha`

7.2.3 图像的过滤和增强

PIL 哭也的 ImageFilter 类和 ImageEnhance 类提供了过滤图像和增强图像的方法。

ImageFilter 提供的方法如下：

方法	描述
ImageFilter.BLUR	图像的模糊效果
ImageFilter.CONTOUR	图像的轮廓效果
ImageFilter.DETAIL	图像的细节效果
ImageFilter.EDGE_ENHANCE	图像的边界加强效果
ImageFilter.EDGE_ENHANCE_MORE	图像的阈值边界加强效果
ImageFilter.EMBOSS	图像的浮雕效果
ImageFilter.FIND_EDGES	图像的边界效果
ImageFilter.SMOOTH	图像的平滑效果
ImageFilter.SMOOTH_MORE	图像的阈值平滑效果
ImageFilter.SHARPEN	图像的锐化效果

使用 ImageFilter 的方法如下：

```
Image.filter(ImageFilter.<function>)
```

而 ImageEnhance 类则提供了更高级的图像增强功能，例如调整色彩、亮度、对比度、锐化等，如下表：

方法	描述
ImageEnhance.enhance(factor)	对选择属性的树枝增强 factor 倍
ImageEnhance.Color(im)	调整图像的色彩平衡
ImageEnhance.Contrast(im)	调整图像的对比度
ImageEnhance.Brightness(im)	调整图像的宽度
ImageEnhance.Sharpness(im)	调整图像的锐度

7.3 一二维数据的格式化和处理

7.3.1 数据组织的维度

数据的维度这个概念仅在本书有出现，因此简单介绍一下：

一维数据，其实可以理解成是一个数组，数组中元素可以是有序的，也可以是无序的。

二维数据，更像是一张 Excel 表格，有行和列。

你会发现，一维数据和二维数据的区别是，要想找到一个一维数据只需要一个参数，而找到一个二维数据需要两个参数（行和列）。

那么以此类推，需要更多参数来找到的数据，我们可以称之为高维数据。

7.3.2 一二维数据的存储格式

一维数据相对比较简单，有不同的存储方式，例如：

- 元素之间用一个或者多个空格隔开，如 `中国 美国 日本 德国 法国 英国 意呆利`
- 元素之间用逗号隔开，如 `中国,美国,日本,德国,法国,英国,意呆利`
- 当然也可以用其他符号隔开

而二维数据则可以由多条一维数据组成。

一种国际通用的二维数据存储格式叫做 CSV（Comma-Seperated Values，逗号分隔值），它的规则如下：

- 纯文本格式，通过单一编码表示字符。
- 以行表示单位，开头不留空行，行之间没有空行。
- 每行表示一个一维数据，多行表示二维数据。
- 以逗号（英文，半角）分隔每列数据，列数据为空也要保留逗号。
- 对于表格数据，可以包含或不包含列名，包含列名时要把列名放置在文件第一行。

7.3.3 一二维数据的表示和读写

对于一维和二维数据，我们都可以使用 Python 中的列表来进行表示。一维数据对应一维列表，二维数据对应二维列表。

在数据的读写中通常会用到以下方法：

- `str.split(',')` 将字符串按照某个字符串分割成列表
- `",".join(list)` 将一个列表组合成一个字符串，中间用某个字符串隔开
- `str.replace('A', 'B')` 将其中所有的 A 字符串都换成 B 字符串

7.4 高维度数据的格式化

高维度数据的保存方式与低维度略有不同。高维度数据采用的方式是 键值对。就好比你手上拿着一大堆钥匙，每个钥匙能够对应一个数据盒子。要想得到对应的数据，你就需要使用对应的钥匙。

JSON 格式就是用于存储键值对的一种通用数据格式，它有如下约束规则：

- 数据保存在键值对中。
- 键值对之间用逗号分隔。
- 大括号用于保存键值对数据组成的对象。
- 方括号用于保存键值对数据组成的数组。

例如，这里就有一个 Json 数据的栗子：

```
"智库 Python 组成员" : [
  {
    "姓氏" : "高",
    "名字" : "萁",
    "级别" : "组长"
  },
  {
    "姓氏" : "任",
    "名字" : "腾龙",
    "级别" : "成员"
  },
  {
    "姓氏" : "苏",
    "名字" : "庆栋",
    "级别" : "成员"
  },
  {
    "姓氏" : "刘",
    "名字" : "元",
    "级别" : "成员"
  }
]
```

~~兄弟萌把猝不及防打在公屏上~~

7.5 json 库的使用

7.5.1 概述

Json 库是用来处理 Json 格式的 Python 标准库，导入方式如下：

```
import json
```

json 库主要包含两类函数，一类是解析函数，一类是操作函数。

操作函数主要完成对外部 Json 格式数据和程序内部数据类型之间的转换。

解析函数主要用于解析键值对内容。

7.5.2 json 库解析

json 库包含两个过程：

- 编码（encoding）：将 Python 数据类型变换成 Json 格式的过程。
- 解码（decoding）：将 Json 格式中解析数据对应到 Python 数据类型的过程。

本质上，编码和解码是数据的序列化和反序列化的过程。

json 库的操作函数如下：

函数	描述
json.dumps(obj, sort_keys=False, indent=None)	将 Python 的数据类型转换为 Json 格式，编码过程
json.loads(string)	将 Json 格式字符串转换为 Pythoh 的数据类型，解码过程
json.dump(obj, fp, sort_keys=False, indent=None)	与 dumps() 功能一致，输出到文件 fp
json.load(fp)	与 loads() 功能一致，从文件 fp 读入

对于 `json.dumps` 函数，其中：

- `obj` 可以是 Python 的列表或者字典类型。当输入字典类型时，它将自动变为 Json 格式字符串，默认按照原本顺序存放。
- `sort_keys` 参数能够让字典元素按照 key 进行排序。
- `indent` 参数能够增加数据缩进，使得其拥有更高的可读性。

拓展：Python 中常用的数据类型转换函数：

函数	参数
int(x [, base])	将字符串 x 转换为一个整数
float(x)	将字符串 x 转换为一个浮点数
complex(real [, imag])	根据 real 和 imag 创建一个浮点数
str(x)	将对象 x 转换为字符串
repr(obj)	将对象 obj 当作 Python 语句执行，返回结果的字符串形式
eval(str)	计算字符串中的有效 Python 表达式，返回结果
ruple(s)	将序列 s 转换为一个元组
list(s)	将序列 s 转换为一个列表
chr(x)	将一个整数转换为一个字符
unichr(x)	将一个整数转换为 Unicode 字符
ord(x)	将一个字符转换为它的整数值
hex(x)	将一个整数转换为一个十六进制字符串
oct(x)	将一个整数转换为一个八进制字符串

第八章 程序设计方法论

学习目标：

- 了解计算思维的概念。
- 掌握自顶向下的设计方法。
- 掌握自底向上的执行过程。
- 了解计算生态和模块编程思想。
- 掌握Python第三方库的安装方法。
- 掌握Python源文件的打包方法。

8.1 计算思维

计算思维是人类科学思维活动的重要组成部分。人类在认识世界、改造世界过程中表现出3种基本的思维特征：以实验和验证为特征的实证思维，以物理学科为代表：以推理和演绎为特征的逻辑思维，以数学学科为代表；以设计和构造为特征的计算思维，以计算机学科为代表。理解计算思维，除了认识概念本身，还要清晰地认识它的时代特性。

在程序设计范畴，计算思维主要反映在理解问题的计算特性、将计算特性抽象为计算问题、通过程序设计语言实现问题的自动求解等几个方面。

8.2 实例15：体育竞技分析

本节使用一种从各种球类比赛中抽象的一般规则，规则定义如下：两个球员在一个有4面边界的场地上用球拍击球。开始比赛时，其中一个球员首先发球。接下来球员交替击球，直到可以判定得分为止，这个过程称为回合。
当一名球员未能进行一次合法击打时，回合结束。未能打中球的球员输掉这个回合。如果输掉这个回合的是发球方，那么发球权交给另一方；如果输掉的是接球方，则仍然由这个回合的发球方继续发球。
总之，每回合结束，由赢得该回合的一方发球。球员只能在他们自己的发球局中得分。首先达到15分的球员赢得一局比赛。
在计算机模拟中，运动员的能力级别将通过发球方赢得本回合的概率来表示。因此，一个0.6概率的球员可以在他的发球局有60%的可能性赢得1分。程序首先接收两个球员的水平值，然后利用这个值采用概率方法模拟多场比赛。程序最后会输出比赛运行结果。

```
该问题的IPO描述如下。
输入:两个球员(球员A和B)的能力概率，模拟比赛的场次
处理:模拟比赛过程
输出:球员A和B分别赢得球赛的概率
```

抽象这个问题时，将球员失误、犯规等可能性一并考虑在能力概率中，在每局比赛中，球员A先发球。

```
一个期望的输出结果如下。
模拟比赛数量: 500
球员A获胜场次: 268 (53.6%)
球员B获胜场次: 232 (46.4%)
```

体育竞技分析程序需要面对不确定事件。一个球员赢得了50%的发球权，并不意味着剩下的每一局他都是胜利者。

> 模拟和仿真
> 模拟(simulation)是抽象原系统某些行为特征并用另一系统来表示这些特征)的过程，通常用于设计初期的模型验证。仿真(emulation)则更进一步，需要模仿系统真实能做的事情，接收同样的数据，获得同样的结果，只不过实现的过程不同。仿真一般用于处理兼容性问题或在资源有限的条件下实现系统原型。

8.3 自顶而下和自底而上

程序需要采用自顶而下的设计方法，采用自底而上的执行方法。

8.3.1 自顶而下设计

顶层设计:
自顶而下设计中最重要的就是顶层设计。大多数程序都可以将IPO描述直接用到程序结构设计中，体育竞技分析从用户处得到模拟参数，最后输出结果。

步骤1:打印程序的介绍性信息。

```
def main():
    printIntro()
```

步骤2:获得程序运行需要的参数，即probA、probB、n。

```
def main():
    printIntro()
    probA,probB,n = getInputs()
```

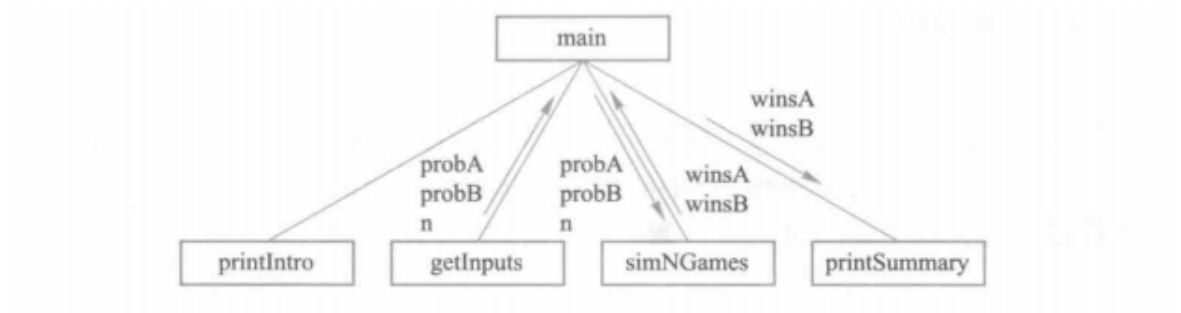
步骤3:利用球员A和B的能力值probA和probB,模拟n次比赛。
probA、probB、n。

```
def main():
    printIntro()
    probA,probB,n = getInputs()
    winsA,winsB = simNGames(n,probA,probB)
```

步骤4:输出球员A和B获胜比赛的场次及概率。
probA、probB、n。

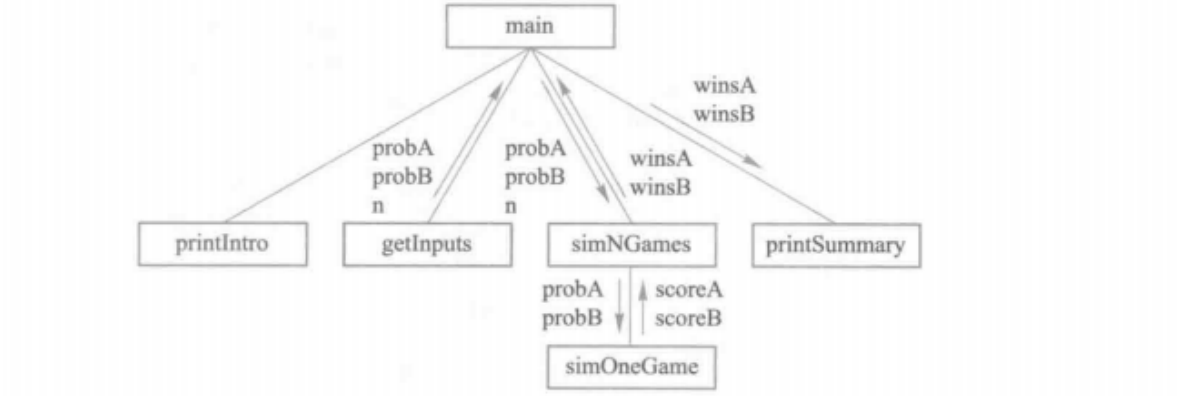
```
def main():
    printIntro()
    probA,probB,n = getInputs()
    winsA,winsB = simNGames(n,probA,probB)
    printSummary(winsA,winsB)
```

第n层设计
上述设计可以表示为下图：

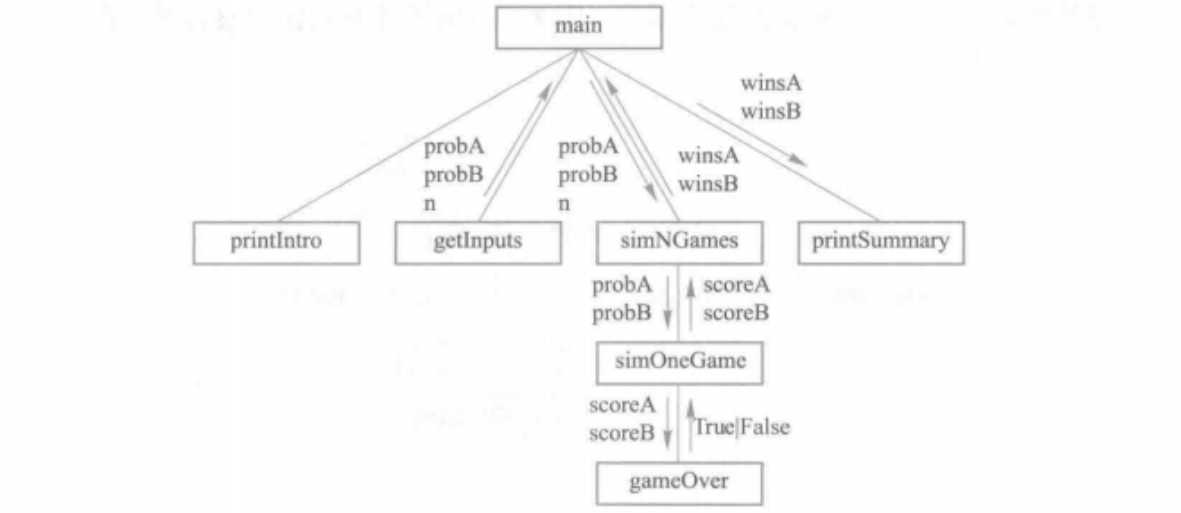


自顶而下设计的第二阶段是实现或进一步抽象第2层函数。

printIntro()函数应该输出一个程序介绍。
getIntro()函数根据提示得到3个需要返回主程序的值。
simNGames()函数是整个程序的核心，基本思想模拟n场比赛，并跟踪记录每个球员赢得了多少比赛。这个函数需要知道每个球员的概率，返回两个球员的最终得分，下图给出了这个设计对整体结构的更新。



siMOneGame()函数中需要根据比赛规则编写代码，两个球员A和B持续对攻直至比赛结束。可以采用无限循环结构知道比赛结束条件成立。同时需要跟踪记录比赛得分，保留发球局标记。
这里进一步设计了gameOver()函数，用来表示一次比赛结束的条件，封装该函数有助于简化根据不同规则修改函数的代价，提高代码的可维护性。程序结构图更新如下：



printSummary()函数最后输出比赛结果。

最后代码实现如下：

```
from random import random
def printIntro():
    print("这个程序模拟两个选手A和B的某种竞技比赛")
```

```
    print("程序运行需要A和B的能力值（以0到1之间的小数表示）")
def getInputs():
    a = eval(input("请输入选手A的能力值(0-1): "))
    b = eval(input("请输入选手B的能力值(0-1): "))
    n = eval(input("模拟比赛的场次: "))
    return a, b, n
def simNGames(n, probA, probB):
    winsA, winsB = 0, 0
    for i in range(n):
        scoreA, scoreB = simOneGame(probA, probB)
        if scoreA > scoreB:
            winsA += 1
        else:
            winsB += 1
    return winsA, winsB
def gameOver(a,b):
    return a==15 or b==15
def simOneGame(probA, probB):
    scoreA, scoreB = 0, 0
    serving = "A"
    while not gameOver(scoreA, scoreB):
        if serving == "A":
            if random() < probA:
                scoreA += 1
            else:
                serving="B"
        else:
            if random() < probB:
                scoreB += 1
            else:
                serving="A"
    return scoreA, scoreB
def printSummary(winsA, winsB):
    n = winsA + winsB
    print("竞技分析开始，共模拟{}场比赛".format(n))
    print("选手A获胜{}场比赛，占比{:0.1%}".format(winsA, winsA/n))
    print("选手B获胜{}场比赛，占比{:0.1%}".format(winsB, winsB/n))
def main():
    printIntro()
    probA, probB, n = getInputs()
    winsA, winsB = simNGames(n, probA, probB)
    printSummary(winsA, winsB)
main()
```

运行结果如下：

> 这个程序模拟两个选手A和B的某种竞技比赛
> 程序运行需要A和B的能力值（以0到1之间的小数表示）
> 请输入选手A的能力值(0-1): **0.45**
> 请输入选手B的能力值(0-1): **0.5**
> 模拟比赛的场次: **1000**
> 竞技分析开始，共模拟1000场比赛
> 选手A获胜359场比赛，占比35.9%
> 选手B获胜641场比赛，占比64.1%

注意，深入探讨竞技规律的前提是参赛选手水平差别不大，所发现的规律有助于弥补短板，类似于中国男足和巴西男足的竞技规律，似乎没有什么必要去探讨了。（摊手）

设计过程总结

本节结合体育竞技实例介绍了自顶向下的设计过程。从问题输入输出确定开始，整体设计逐渐向下进行。每一层以大体算法描述开始，然后逐步细化成代码，细节被函数封装，整个过程可以概括为以下4个步骤。

步骤1:将算法表达为一系列小问题。

步骤2:为每个小问题设计接口。

步骤3:通过将算法表达为接口关联的多个小问题来细化算法。

步骤4:为每个小问题重复上述过程。

自顶向下设计是一种开发复杂程序最具价值的设计理念和工具，设计过程自然且简单，自顶向下设计通过封装实现抽象，利用了模块化设计的思想。

8.3.2 自底而上执行

执行中等规模程序的最好方法是从结构图最底层开始，而不是从顶部开始，然后逐步上升。或者说，先运行和测试每一个基本函数，再测试由基础函数组成的整体函数，这样有助于定位错误。

8.4 模块7：pyinstaller库的使用

8.4.1 pyinstaller概述

pyinstaller 是一个十分有用的第三方库，它能够在Windows、Linux、 Mac OS X等操作系统下将Python源文件打包，通过对源文件打包，Python程序可以在没有安装Python的环境中运行，也可以作为一个独立文件方便传递和管理。

使用pyinstaller库需要注意以下问题。

（1）文件路径中不能出现空格和英文句号(.)。

（2）源文件必须是UTF-8编码，暂不支持其他编码类型。采用IDLE编写的源文件都保存为UTF-8编码形式，可直接使用。

8.4.2 pyinstaller解析

pyinstaller有一些常用参数，如下:

参数	功能
-h,--help	查看帮助
-v,--version	查看pyinstaller版本
--clean	清理打包过程中的临时文件
-D,--onedir	默认值，生成dist目录
-F,--onefile	在dist文件夹中只生成独立的打包文件
-p DIR,--paths DIR	添加Python文件使用的第三方库路径
-i<.ico or .exe,ID or.icns>
--icon<.ico or .exe,ID or .icns>
指定打包程序使用的图标（icon）文件

pyinstaller命令不需要在Python源文件中增加代码，只需要通过命令行进行打包即可。

8.5 计算生态和模块编程

近20年的开源运动产生了深植于各信息技术领域的大量可重用资源,直接且有力地支撑了信息技术超越其他技术领域的发展速度，形成了“**计算生态**”。产业界广泛利用可重用资源快速构建应用已经是主流产品开发方式。Python 语言从诞生之初致力于开源开放，建立了全球最大的编程计算生态。

在计算生态思想指导下，编写程序的起点不再是探究每个具体算法的逻辑功能和设计，而是尽可能利用第三方库进行代码复用，探究运用库的系统方法。这种像搭积木一样的编程方式，称为“**模块编程**”。每个模块可能是标准库、第三方库、用户编写的其他程序或对程序运行有帮助的资源等。模块编程与模块化设计不同，模块化设计主张采用自顶向下设计思想，主要开展耦合度低的单一程序设计与开发, 而模块编程主张利用开源代码和第三方库作为程序的部分或全部模块，像搭积木一样编写程序。

学了编程能做什么?似乎只能打印字符、挑战汉诺塔。这是大多数编程语言学习预期和学习效果的鸿沟。因为绝大多数编程语言设计用于开发专业功能，而不是用于构建计算生态，因此，需要专业程序员经过漫长学习才能够掌握并开发有价值的程序。Python 语言却不同，它不是其他语言的替代，而是一种真正面向计算生态的语言。

第九章 科学计算和可视化

9.1 前言

开展基本的科学计算需要两个步骤：**组织数据和展示数据**。组织数据是运算的基础，也是将客观世界数字化的必要手段；展示数据是体现运算结果的有力武器。

首先让我们回到线性代数里最先学习的概念—— **矩阵**。数学的矩阵是一个按照阵列排序的复数或实数集合，最早来自于方程组的系数和常数所构成的方阵。矩阵是高等代数学中的常用工具，主要应用于统计数学，物理学，电路学，力学光学，量子物理，计算机图像和动画等领域。科学计算以矩阵作为基础，增加了计算密度，能够表达更为复杂的运算逻辑。

本章将介绍用于组织和运算数据的第三方Python库numpy和展示并绘制专业图表的第三方库matplotlib。

9.2 numpy库的使用

python中用列表保存一组值，可将列表当数组使用。另外，python中有array模块，但它不支持多维数组，无论是时列表还是array模块都没有科学运算函数，不适合做矩阵等科学计算。numpy没有使用python本身的数组机制，而是提供了ndarray对象，该对象不仅能方便地存取数组，而且拥有丰富的数组计算函数。

使用前先我们首先需要导入Numpy模块。

```
import numpy as np
#或
from numpy import *
```

9.2.1 数组的声明及使用

Numpy中定义的最重要使用最频繁的对象就是ndarray的N维数组类型。

它描述相同类型（dtype）的元素集合，可以使用基于零的索引访问集合中的项目，它的类似Python中的列表索引。我们学习numpy库，首先得学习ndarray对象。

ndarray的创建方式如下：

```
numpy.array(object, dtype =None, copy =True, order =None, subok =False, ndmin
=0)
```

各参数解释如下：

- 【object】：数组或嵌套的数列
- 【dtype】：数组元素的数据类型，可选
- 【copy 】：对象能否复制，可选
- 【order】：创建数组的样式，C为行方向，F为列方向，A为任意方向，默认为A
- 【subok】：默认返回一个与基类类型一致的数组
- 【ndmin】：指定生成数组的最小维度

下面让我们来看几个例子

```
#创建一维数组
import numpy as np
list = [1,2,3,4]
oneArray = np.array(list)

print(list)     #[1, 2, 3, 4]  这是python的列表对象
print(oneArray) #[1 2 3 4]     这是一个一维数组
```

```
#创建二维数组
import numpy as np
twoArray = np.array([[1,2],[3,4],[5,6]])

print(twoArray) #[[1 2] [3 4] [5 6]]  这是一个二维数组
```

Numpy库的具体函数如下：

类型	类型代码	说明
int8、uint8	i1、u1	有符号和无符号8位整型（1字节）
int16、uint16	i2、u2	有符号和无符号16位整型（2字节）
int32、uint32	i4、u4	有符号和无符号32位整型（4字节）
int64、uint64	i8、u8	有符号和无符号64位整型（8字节）
float16	f2	半精度浮点数
float32	f4、f	单精度浮点数
float64	f8、d	双精度浮点数
float128	f16、g	扩展精度浮点数
complex64	c8	分别用两个32位表示的复数
complex128	c16	分别用两个64位表示的复数
complex256	c32	别用两个128位表示的复数
bool	?	布尔型
object	O	python对象
string	Sn	固定长度字符串，每个字符1字节，如S10
unicode	Un	固定长度Unicode，字节数由系统决定，如U10

NumPy.ndarray函数和属性：

函数	属性
ndarray.ndim	获取ndarray的维数
ndarray.shape	获取ndarray各个维度的长度
ndarray.dtype	获取ndarray中元素的数据类型
ndarray.T	简单转置矩阵ndarray

更多详细的属性，同学们可以在需要时继续搜索，本章仅提供部分内容。

9.3 matplotlib库的使用

Matplotlib 是 Python 的绘图库。 它可与 NumPy 一起使用，提供了一种有效的 MatLab 开源替代方案。 它也可以和图形工具包一起使用，如 PyQt 和 wxPython。

```
#一个简单的例子
import matplotlib.pyplot as plt
path = ""
plt.plot([4,7,1,9,4])
    #绘图接收一个list，如果只有一个list默认其为Y轴，X轴数据为其索引值，从0开始
plt.ylabel("grade")
plt.axis([-2,8,0,12])
    #axis函数接收一个list，设定横纵坐标尺度，list各个参数分别代表[X初始刻度，X终止刻度，Y起始刻度，Y终止刻度]
plt.savefig(path,dpi = 600)
    #savefig函数用来保存图片至path地址，dpi值表示每英寸具有的像素点数，1英寸=2.54cm
plt.show()
```

接下来我们为大家提供简单的常用函数大全

plt.savefig('test', dpi = 600)： 将绘制的图画保存成png格式，命名为 test

plt.ylabel('Grade')： y轴的名称

plt.axis([-1, 10, 0, 6])： x轴起始于-1，终止于10 ， y轴起始于0，终止于6

plt.subplot(3,2,4)： 分成3行2列，共6个绘图区域，在第4个区域绘图。排序为行优先。也可plt.subplot(324)，将逗号省略。

.plot函数
plt.plot(x, y, format_string, **kwargs): x为x轴数据，可为列表或数组；y同理；format_string 为控制曲线的格式字符串，** kwargs 第二组或更多的（x, y, format_string）

format_string: 由 颜色字符、风格字符和标记字符组成。

颜色字符：'b'蓝色 ；'#008000'RGB某颜色；'0.8'灰度值字符串

风格字符：'-'实线；'--'破折线； '-.'点划线； ':'虚线 ； ''无线条

标记字符：'.'点标记 'o' 实心圈 'v'倒三角 '^'上三角

eg： plt.plot(a, a*1.5, 'go-', a, a*2, '*') 第二条无曲线，只有点

.plot 显示中文字符
pyplot并不默认支持中文显示，需要rcParams修改字体来实现

rcParams的属性：

‘font.family’ 用于显示字体的名字

‘font.style’ 字体风格，正常‘normal’ 或斜体‘italic’

‘font.size’ 字体大小，整数字号或者‘large’ ‘x-small’

eg:

import matplotlib

matplotlib.rcParams[‘font.family’] = ‘STSong’

matplotlib.rcParams[‘font.size’] = 20

设定绘制区域的全部字体变成 华文仿宋，字体大小为20

中文显示2：只希望在某地方绘制中文字符，不改变别的地方的字体

在有中文输出的地方，增加一个属性： fontproperties

eg:

plt.xlabel(‘横轴：时间’, fontproperties = ‘simHei’, fontsize = 20)

pyplot文本显示函数：

plt.xlabel()：对x轴增加文本标签

plt.ylabel()：同理

plt.title(): 对图形整体增加文本标签

plt.text(): 在任意位置增加文本

plt. annotate(s, xy = arrow_crd, xytext = text_crd, arrowprops = dict)

: 在图形中增加带箭头的注解。s表示要注解的字符串是什么，xy对应箭头所在的位置，xytext对应文本所在位置，arrowprops定义显示的属性

eg:

plt.xlabel(‘横轴：时间’， fontproperties = ‘SimHei’, fontsize = 15, color = ‘green’)

plt.ylabel(‘纵轴：振幅’, fontproperties = ‘SimHei’, fontsize = 15)

plt.title(r’正弦波实例 $y = cons(2\pi x)$’ , fontproperties = ‘SimHei’, fontsize = 25)

plt.annotate (r’%mu=100$, xy = (2, 1), xytext = (3, 1.5),

arrowprops = dict(facecolor = ‘black’, shrink = 0.1, width = 2)) # width表示箭头宽度

plt.text (2, 1, r’$\mu = 100$, fontsize = 15)

plt.grid(True)

plt. annotate(s, xy = arrow_crd, xytext = text_crd, arrowprops = dict)

第十章 网络爬虫和自动化

10.1 本章简介

似乎一瞬间全世界都开始讨论网络爬虫，随着网络的迅速发展，如何有效地提取并利用信息很大程度上决定了解决问题的效率。搜索引擎作为辅助程序员检索信息的工具已经有些力不从心。为了更高效地获取指定信息需要定向抓取并分析网页资源，网络爬虫火爆了起来。

本章将讲述编写优质网络爬虫的方法。嗯，别人有的这里也有，而且还更好。

10.2 问题概述

Python语言发展中有一个里程碑式的应用事件，即美国谷歌(Google) 公司在搜索引擎后端采用Python语言进行链接处理和开发,这是该语言发展成熟的重要标志。Python 语言的简洁性和脚本特点非常适合链接和网页处理，因此，在Python的计算生态中，与URL和网页处理相关的第三方库很多。

网络爬虫应用一般分为两个步骤：①通过网络链接获取网页内容；②对获得的网页内容进行处理。这两个步骤分别使用不同的函数库: requests 和beautifulsoup4。
采用pip指令安装requests库，如果在Python 2和Python 3并存的系统中，采用pip3指令，代码如下:

```
:\>pip install requests  #或者pip3 install requests
```

采用pip或pip3指令安装beautifulsoup4库，注意，不要安装beautifulsoup库，后者由于年久失修，已经不再维护了。安装命令如下:

```
:\>pip install beauti fulsoup4   #或者pip3 install beautifulsoup4
```

使用Python语言实现网络爬虫和信息提交是非常简单的事情，代码行数很少，也无须掌握网络通信等方面的知识，非常适合非专业读者使用。**然而，肆意地爬取网络数据并不是文明现象，通过程序自动提交内容争取竞争性资源也不公平。就像那些肆意的推销电话一样，他们无视接听者意愿，不仅令人讨厌也有可能引发法律纠纷。**

10.3 模块10：requests库的使用

10.3.1 requests库概述

requests库是一个简洁且简单的处理HTTP请求的第三方库，它的最大优点是程序编写过程更接近正常URL访问过程。

requests库支持非常丰富的链接访问功能，包括国际域名和URL获取、HTTP长连接和连接缓存、HTTP会话和Cookie保持、浏览器使用风格的SSL验证、基本的摘要认证、有效的键值对Cookie记录、自动解压缩、自动内容解码、文件分块上传、HTTP(S)代理功能、连接超时处理、流数据下载等。

10.3.2 requests库解析

与网页请求相关的函数如下表所示:

函数	描述
get(url[,timeout=n])	对应于HTTP的GET方式，获取网页最常用的方法，可以增加timeout=n参数，设定每次请求超时时间为n秒
post(url,data={'key':'value'})	对应于HTTP的POST方式，其中字典用于传递客户数据
delete(url)	对应于HTTP的DELETE方式
head(url)	对应于HTTP的HEAD方式
options(url)	对应于HTTP的OPTIONS方式
put(url,data={'key':'value'})	对应于HTTP的PUT方式，其中字典用于传递客户数据

get()是获取网页最常用的方式，在调用requests.get0)函数后，返回的网页内容会保存为一个Response对象。其中，get()函数的参数 url 链接必须采用HTTP或HTTPS方式访问，

和浏览器的交互过程一样，requests.get(代表请求过程， 它返回的Response对象代表响应。返回内容作为一个对象更便于操作，Response对象的属性如下表所示，需要采用 <a>.<b> 形式。

属性	描述
status_code	HTTP请求的返回状态，整数，200表示成功，404表示失败
text	HTTP响应内容的字符串形式，即url对应的页面内容
encoding	HTTP响应式内容的编码方式
content	HTTP响应内容的二进制形式

- status_code属性返回请求HTTP后的状态,在处理数据之前要先判断状态情况，如果请求未被响应，需要终止内容处理。
- text 属性是请求的页面内容，以字符串形式展示。
- encoding 属性非常重要，它给出了返回页面内容的编码方式，可以通过对encoding属性赋值更改编码方式，以便于处理中文字符。
- content 属性是页面内容的二进制形式。

除了属性，Response对象还提供一些方法，如下表所示：

方法	描述
json()	如果HTTP响应内容包含JSON格式数据，则该方法解析JSON数据
raise_for_status()	如果不是200，则产生异常

获取一个网页内容的函数建议采用如下代码的第2到第9行，第10和第11行是测试代码。

```
import requests
def getHTMLText():
    try :
        r = requests.get (ur1,timeout = 30)
        r.raise_for_status ()
        r.encoding = 'utf-8'
        return r.text
    except:
        return ""
url = "http://www.baidu.com"
print(getHTMLText(url))
```

10.4 模块11：beautifulsoup4库的使用

10.4.1 beautifulsoup4库概述

beautifulsoup4库，也称为Beautiful Soup 库或bs4库，用于解析和处理HTML和XML。需要注意的是，它不是BeautifulSoup库。它的最大优点是能根据HTML和XML语法建立解析树，进而高效解析其中的内容。

HTML建立的Web页面一般非常复杂，除了有用的内容信息外，还包括大量用于页面格式的元素，直接解析一个Web网页需要深入了解HTML语法，而且比较复杂。beautifulsoup4库将专业的Web页面格式解析部分封装成函数，提供了若干有用且便捷的处理函数。

beautifulsoup4库采用面向对象思想实现，简单地说，它把每个页面当作-一个对象，通过<a>.<b>的方式调用对象的属性(即包含的内容)，或者通过<a>.<b>()的方式调用方法(即处理函数)。

在使用beautifulsoup4库之前，需要进行引用，由于这个库的名字非常特殊且采用面向对象方式组织，可以用from-import方式从库中直接引用BeautifulSoup 类，
方法如下:

```
from bs4 import BeaytifulSoup
```

10.4.2 beautifulsoup4库解析

beautifulsoup4库中最主要的是BeautifulSsoup 类，每个实例化的对象相当于一个页面。采用from-import导入库中的Beautifulsoup类后，使用BeautifulSoup()创建一个BeautifulSoup对象。

创建的BeautifulSoup对象是一个树形结构,它包含HTML页面中的每一个Tag(标签)元素，如<head>、<body>等。 具体来说，HTML中的主要结构都变成了BeautifulSoup对象的一个属性，可以直接用<a>.<b>形式获得，其中<b>的名字采用HTML中标签的名字。下表列出了BeautifulSoup中常用的一些属性。

属性	描述
head	HTML页面的<head>内容
title	HTML页面标题，在<head>之中，由<title>标记
body	HTML页面的<body>内容
p	HTML页面中第一个<p>内容
strings	HTML页面所有呈现在Web上面的字符串，即标签的内容
stripped_strings	HTML页面所有呈现在Web上的非空格字符串

BeautifulSoup属性与HTML的标签名称相同，每一个Tag标签在beautifulsoup4库中也是一个对象，称为**Tag对象。**上例中，title 是一个标签对象。可以通过Tag对象的name、attrs 和string 属性获得相应内容，采用<a>.<b>的语法形式。标签Tag有4个常用属性，如下表所示。

属性	描述
name	字符中，标签的名字，比如div
attrs	字典，包含了原来页面Tag所有的属性，比如href
contents	列表，这个Tag下所有子Tag的内容
string	字符串，Tag所包围的文本，网页中真实地文字

由于HTML语法可以在标签中嵌套其他标签，所以，string 属性的返回值遵循如下原则。
(1)如果标签内部没有其他标签，string 属性返回其中的内容。
(2)如果标签内部还有其他标签，但只有一个标签， string 属性返回最里面标签的内容。
(3)如果标签内部有超过1层嵌套的标签，string 属性返回None (空字符串)。

当需要列出标签对应的所有内容或者需要找到非第一个标签时， 需要用到BeautifulSoup的find()和find_ all()方法。这两个方法会遍历整个HTML文档，按照条件返回标签内容。

```
BeautifulSoup.find_ all (name，attrs,recursive,string,limit)
```

作用:根据参数找到对应标签，返回列表类型。

参数如下。

- name：按照Tag标签名字检索，名字用字符串形式表示，例如div、li。
- attrs：按照Tag标签属性值检索，需要列出属性名称和值，采用JSON表示。
- recursive：设置查找层次，只查找当前标签下一层时使用recursive-False.
- string：按照关键字检索string属性内容，采用string=开始。
- limit：返回结果的个数，默认返回全部结果。

除了find_ all()方法，BeautifulSoup 类还提供一个find()方法，它们的区别只是前者返回全部结果而后者返回找到的第一个结果，find_ all()函数由于可能返回更多结果，所以采用列表形式；find()函数返回字符串形式。

```
BeautifulSoup.find (name,attrs,recursive,string)
```

作用:根据参数找到对应标签，采用字符串返回找到的第一个值。
参数:与find.all()方法一样，略。

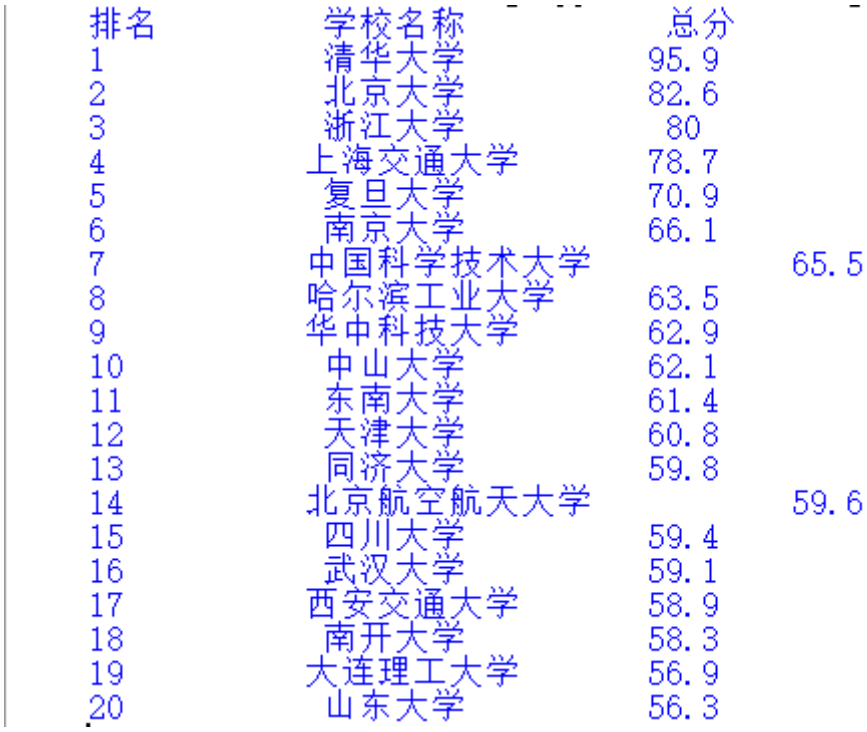
10.5 实例20：中国大学排名爬虫

功能描述：
输入大学排名url链接
输出大学排名信息（排名，大学名称，总分）
程序的结构设计：
从网络上获取大学排名网页内容
提取网页内容中信息到合适的数据结构
利用数据结构展示并输出结果

代码：

```
import requests
from bs4 import BeautifulSoup
import bs4

def getHTMLText(url):
    try:
        r = requests.get(url, timeout=30)
        r.raise_for_status()
        r.encoding = r.apparent_encoding
        return r.text
    except:
        return ""

def fillUnivList(ulist, html):
    soup = BeautifulSoup(html, "html.parser")
    for tr in soup.find('tbody').children:
        if isinstance(tr, bs4.element.Tag):
            tds = tr('td')
            ulist.append([tds[0].string, tds[1].string, tds[3].string])

def printUnivList(ulist, num):
    print("{:^10}\t{:^6}\t{:^10}".format("排名","学校名称","总分"))
    for i in range(num):
        u=ulist[i]
        print("{:^10}\t{:^6}\t{:^10}".format(u[0],u[1],u[2]))

def main():
    uinfo = []
    url = 'http://www.zuihaodaxue.cn/zuihaodaxuepaiming2016.html'
    html = getHTMLText(url)
    fillUnivList(uinfo, html)
    printUnivList(uinfo, 20) # 20 univs
main()
```

运行截图：



排名	学校名称	总分
1	清华大学	95.9
2	北京大学	82.6
3	浙江大学	80
4	上海交通大学	78.7
5	复旦大学	70.9
6	南京大学	66.1
7	中国科学技术大学	65.5
8	哈尔滨工业大学	63.5
9	华中科技大学	62.9
10	中山大学	62.1
11	东南大学	61.4
12	天津大学	60.8
13	同济大学	59.8
14	北京航空航天大学	59.6
15	四川大学	59.4
16	武汉大学	59.1
17	西安交通大学	58.9
18	南开大学	58.3
19	大连理工大学	56.9
20	山东大学	56.3

代码优化：

针对输出后中文对齐问题

原因： 当中文字符宽度不够时,采用西文字符填充；中西文字符占用宽度不同。

解决： 当中文字符宽度不够时,采用西文字符填充；中西文字符占用宽度不同采用中文字符的空格填充chr(12288)

```
def printUnivList(ulist, num):
    tplt = "{0:^10}\t{1:{3}^10}\t{2:^10}"
    print(tplt.format("排名","学校名称","总分",chr(12288)))
    for i in range(num):
        u=ulist[i]
        print(tplt.format(u[0],u[1],u[2],chr(12288)))
```

优化后全代码如下：

```
import requests
from bs4 import BeautifulSoup
import bs4

def getHTMLText(url):
    try:
        r = requests.get(url, timeout=30)
        r.raise_for_status()
        r.encoding = r.apparent_encoding
        return r.text
    except:
        return ""
def fillUnivList(ulist, html):
    soup = BeautifulSoup(html, "html.parser")
    for tr in soup.find('tbody').children:
        if isinstance(tr, bs4.element.Tag):
            tds = tr('td')
            ulist.append([tds[0].string, tds[1].string, tds[3].string])


```

```
def printUnivList(ulist, num):
    tplt = "{0:^10}\t{1:{3}^10}\t{2:^10}"
    print(tplt.format("排名","学校名称","总分",chr(12288)))
    for i in range(num):
        u=ulist[i]
        print(tplt.format(u[0],u[1],u[2],chr(12288)))

def main():
    uinfo = []
    url = 'http://www.zuihaodaxue.cn/zuihaodaxuepaiming2016.html'
    html = getHTMLText(url)
    fillUnivList(uinfo, html)
    printUnivList(uinfo, 20) # 20 univs
main()
```

优化后运行截图;

```
    排名              学校名称              总分
    1                 清华大学             95.9
    2                 北京大学             82.6
    3                 浙江大学              80
    4               上海交通大学           78.7
    5                 复旦大学             70.9
    6                 南京大学             66.1
    7             中国科学技术大学         65.5
    8             哈尔滨工业大学           63.5
    9               华中科技大学           62.9
    10                中山大学             62.1
    11                东南大学             61.4
    12                天津大学             60.8
    13                同济大学             59.8
    14            北京航空航天大学         59.6
    15                四川大学             59.4
    16                武汉大学             59.1
    17              西安交通大学           58.9
    18                南开大学             58.3
    19              大连理工大学           56.9
    20                山东大学             56.3
...
```

10.6 实例21：搜索关键词自动提交

问题的IPO描述：
输入:待查询关键字
处理:自动获得百度搜索结果页面，并对页面内容解析处理
输出:返回链接的标题列表

利用beautifulsoup4库找到data-tools属性值，提取带有title的字符串，可以看到，data-tools 内部由{}形成的数据是典型的JSON格式，可以用json库将其转换成字典，便于操作。

代码：

```
import requests
from bs4 import BeautifulSoup
import re
import json
def getKeywordResult(keyword):
    url = 'http://www.baidu.com/s?wd='+keyword
    try:
```

```
        r = requests.get(url, timeout=30)
        r.raise_for_status()
        r.encoding = 'utf-8'
        return r.text
    except:
        return ""
def parserLinks(html):
    soup = BeautifulSoup(html, "html.parser")
    links = []
    for div in soup.find_all('div', {'data-tools': re.compile('title')}):
        data = div.attrs['data-tools']  #获得属性值
        d = json.loads(data)        #将属性值转换成字典
        links.append(d['title'])    #将返回链接的题目返回
    return links
def main():
    html = getKeywordResult('Python语言程序设计基础(第2版)')
    ls = parserLinks(html)
    count = 1
    for i in ls:
        print("[{:^3}]{}".format(count, i))
        count += 1
main()
```