程序初步-2

- 复数，字符串
- 数学库
- 变量，标识符和关键字
- 赋值
- 程序、模块和执行
- 输出和输入
- 分支计算和重复计算

复数（complex）

- 复数字面量（语法）

  虚部用（整数/浮点数）加后缀 j 或 J 表示，不能空格

  如 2.0 + 3.2j，2 + 3j

- 或者用 complex(m, n) 构造（complex 是复数类型名）

  m 表示实部，n 表示虚部

- 复数对象的实部和虚部都是浮点数

  2 + 3j 等同于 2.0 + 3.0j

- 混合类型计算

  如果有复数对象参与计算，整数和实数先转换到复数

字符串

- 字符串也是一种基本数据类型（简称串）
  - 类型名：**str**
  - 字面量：一对单引号或一对双引号括起的字符序列，其中可以包括空格和一些特殊字符，如\n 表示换行符。这种形式的字符串只能写在一行里，不能换行
  - 语义：相应的字符串对象
- 设 **s, t** 的值是字符串，**i** 是整数，两个常用操作：
  - **s[i]** 取得 **s** 第 **i** 个字符，首字符 **s[0]**，最后字符 **s[-1]**
  - **s + t** 得到两个串的拼接串，如 **"ab"+"12"** 是 **"ab12"**
  - **len(s)** 得到 **s** 的长度（其中的字符个数）

数学函数（导入函数库）

- 数学函数很重要，计算中经常要用
  - Python 标准库模块 math 提供了许多数学函数
  - 数学函数不是默认功能，使用前需要导入 math 模块

    模块就是做好的 Python 程序，自己也可以做
- 导入模块（几种方式）

  from math import *

  语义：导入 math 库函数，其中所有的函数都直接可用

  import math

  语义：导入 math 库函数，以 math.sin 一类形式使用

  from math import sin, cos, sqrt

  语义：导入 math 库函数 sin, cos 和 sqrt，直接可用

数学函数（使用）

- **math** 库提供的数学函数：

  可以自己查阅“**Python** 标准库手册”（**The Python Standard Library**）**9.2** 节

  执行 **dir(math)** 列出 **math** 模块内容

- 函数调用的形式（语法）

  函数名**(**参数**, ...)**　　其中参数可以是任意表达式

- 函数调用是一种基本表达式，语义：针对给定参数执行函数名指定的函数，结果是函数计算出的值

  函数结果也称返回值。参数应满足函数需要（如类型）

- 已能用 **Python** 完成“函数计算器”能做的所有计算

记录中间结果，变量

- 计算中经常遇到需要记录计算得到的中间结果的情况
- 例：已知三角形三边长分别为 6, 8, 11，求面积。

  面积公式：
  $$a = \sqrt{s(s-x)(s-y)(s-z)}$$
  $$\text{where } s = (x+y+z)/2$$

- 其中 s 用了 4 次

  最好是有办法把算出的 s 值记录下来

- Python 里用于记录信息的机制是变量
  - 变量用名字（标识符）表示，可以记录值（对象）
  - 变量也是一种基本表达式（与字面量类似）。变量写在表达式里，就是要求用它的值（语义）

变量

- 变量的形式（语法要求）：

  常用形式：字母开头的字母数字序列，下划线看作字母

  允许用一般的 unicode 字符序列（但不建议）

  区分大小写，同一字母的大写和小写看作不同东西

- 用变量记录计算的结果（给计算出的对象“取名”）的方式是给变量赋值

- 赋值是 Python 的基本操作，通过赋值语言实现。语法：

  变量 = 表达式

  变量, ... = 表达式, ...　　变量和表达式一一对应

- 语义：顺序求出各表达式的值，然后赋给对应的变量

变量和赋值

- 写在表达式里的变量，就表示使用它保存的值（变量求值）
  - 不能对没赋过值的变量求值，那是错误
  - 如果变量已经有值，再次赋值将改变它的值
    - 变量求值总得到此前最近一次赋给它的值
- 写在赋值符“=”左边的变量表示变量本身，是特殊情况
  - 写在一般表达式里的变量，表示它的值
  - 写在赋值语句（赋值符）左边的变量表示变量本身
- 例：求边长 3, 4, 5 的三角形的面积

```
s = (3 + 4 + 5) / 2
area = sqrt(s * (s – 3) * (s – 4) * (s – 5))
```

- 赋值是语句，不是表达式，执行时产生效果

最简单程序的一般形式

- 最简单的程序，通常就是一系列语句
  - 最常见的语句是赋值语句（**import** 也是语句）
  - 后面还会看到其他语句
- 到现在，已经可以把 **Python** 当作一个高级计算器
  - 基于算术运算符、数学函数等描述表达式
  - 用赋值语句把计算结果记录在变量里

    可以保存任意多的中间结果，积累数据
  - 可以基于以前的结果做进一步计算
- 做交互式计算，人自己控制计算的进程

关键字

- 关键字是 Python 中一组特殊的名字
  - 由语言规定
  - 每个关键字有自己的特殊意义，只能用在特定地方
- 关键字一共 33 个，如下：

  and as assert break class continue def del
  else except elif False finally for from global
  if in import is lambda None nonlocal not or
  pass return raise True try while with yield

- 特别注意：关键字不能用作变量名

  支持 Python 的编辑器认识关键字，以特殊颜色/形式显示

IDLE 执行环境和编辑器

- 在执行环境中的输入只能在一行内编辑
  - 不能修改已经送给系统的输入
  - 不容易利用已经输入的东西
- IDLE 提供了一个编辑器，适合编写长程序，很方便
  - 启动：
    - 用执行环境 file 菜单里的 new window 项
    - 用 file 菜单的 open 项打开一个已有 python 程序文件
  - 执行正在编辑的程序：
    - 保存文件（通过file菜单或按钮，或功能键 Ctrl-S）
    - 用 F5 功能键，或 run 菜单的 run module 项启动执行

模块，执行和输出

- 一个（正在编辑的）**py** 文件是一个模块
  - 执行模块时，表达式的值不在执行环境里显示
  - 要想看到计算结果，就需要使用输出函数
- 标准函数 **print** 产生输出，显示到 **Python** 窗口中

  print(表达式**, ...)**

  效果：顺序求值并输出作为其参数的表达式

  多项输出之间用空格分隔，最后换一行（默认情况）
- **print(…)** 是表达式，其值是个特殊对象，用关键字 **None** 表示，不显示。使用 **print** 的更多细节后面介绍
- 三角形面积程序（模块）

输入

- 假定要计算另一个三角形的面积
  - 一种方法：修改原程序，修改其中的基本数据
  - 太麻烦。利用输入功能可以帮助处理这个问题
- 基本交互式输入，用函数 **input**
  - 形式：**input(提示字符串)**
  - 语义：
    1. 输出“提示字符串”；
    2. 等待用户输入
    3. 返回用户输入的字符序列（做成的字符串对象）
- 程序里的注释，从 # 开始到行尾，执行时跳过，没有语义

输入

- 输入整数（例）

  n = int(input("Input an integer: ")) # 写合适的提示串

  输入浮点数的方式类似：

  x = float(input("…..:"))

- 利用 input 改造三角形计算程序
- 效果和情况：
  - 实现了从输入到输出的计算
  - 把一个专用的完成对一组数据的特殊计算的程序，改造为一个可以用于不同输入数据组的程序
  - 每次使用，需要人工提供输入

检查数据情况（为什么需要判断？）

- 并不是任意三个数都描述了一个三角形
  - 前面程序只能对有意义的三元组完成计算
  - 如果提供的数据不构成三角形，计算就会出错
- 输入来自程序外部（程序的用户）
  - 输入可能不符合程序的需要
  - 为保证程序不出错，应该检查程序的输入
- 求三角形面积的程序，应该检查输入
  - 输入正确时完成计算
  - 输入数据不正确时输出错误信息

条件描述和判断

- 检查数据时需要做比较，以便
  - 判断某些条件是否成立
  - 根据检查的情况，分别处理
- 比较数据对象、检查条件是做逻辑判断
  - 检查两个对象（变量、表达式的值）之间的关系

    需要用关系运算符，写关系表达式
  - 可能需要组合几个简单条件，完成较复杂的判断

    需要用逻辑运算符，写逻辑表达式
  - 逻辑判断的结果也应该是数据，用对象表示

    用一个专门的类型表示关系/逻辑表达式的值

逻辑类型，关系

- 检查数值关系的运算符
  - 等于和不等于

    == !=
  - 序关系运算符

    < <= >= >
  - 关系运算符的优先级低于加减法运算符
  - 做不同数值类型之间比较之前，先转换到相同类型
- 关系判断的结果（关系/逻辑表达式的值）是逻辑类型
  - 类型名：**bool**
  - **bool** 类型只包含两个值：**True**，**False**（真，假）

逻辑运算符

- 用于组合简单的逻辑表达式（条件表达式）
  - 与运算 **and**　　**a and b** 真，当 **a** 和 **b** 都真
  - 或运算 **or**　　**a or b** 真，当 **a** 或 **b** 真
  - 否定运算（一元）**not**　　**not a** 真，当 **a** 假
- 关系运算的连续写法
  - 允许任意连续书写，看作 **and** 的简化
  - **a < b <= c** 相当于 **a < b and b <= c**
  - **a < b >= c** 相当于 **a < b and b >= c**
- 关于逻辑表达式和逻辑判断，有些细节后面讨论

条件语句

- 条件语句是一种结构。基于逻辑条件执行不同操作
- 语法和语义（基本形式1）：

```
if 条件：
    语句组
```

如果条件为真就执行语句组，否则直接结束

- 语法和语义（基本形式2）：

```
if 条件：
    语句组1
else:
    语句组2
```

如果条件为真则执行语句组1，否则执行语句组2

- 例：改造的三角形面积计算程序

条件语句的形式

- **Python** 程序的退格形式和语义
  - **Python** 程序的格式有语义作用
  - 同层结构垂直对齐，下层结构进一步退格
- **if** 语句的细节
  - 一个 **if** 段（一个“**if** 条件 **:**”行和一个语句组）
  - 最后是可选的 **else** 段（“**else :**”行和一个语句组）
  - 两者之间可以有任意多个 **elif** 段

    形式是一个 “**elif** 条件 **:**” 行和一个语句组
- 语句组（手册里称为 **suite**）总是下一层结构，形式是：

  换一行，再退格后的若干语句，相互对齐

if 语句

- if 的一般形式

```
if 条件 :
    语句组
elif 条件 :
    语句组
… …
else:
    语句组
```

在一个 if 的语句组里还可以有 if 语句（嵌套的 if）

- 例：求二元一次方程的根：

先输入数据，然后求出判别式的值，最后根据判别式的值分三种情况处理

基本操作，顺序和分支控制

- 总结一下至今介绍的计算描述方式
  - 基本操作
  - 操作的组合（控制）
- 已经介绍了几种基本操作（语句，都以表达式作为成分）
  - 赋值语句
  - 输入函数，输出函数（通常单独写为语句）
  - import 语句
- 两种控制（和相关执行方式）
  - 顺序结构（顺序书写，描述顺序执行）
  - 分支结构（if 语句，描述根据条件选择执行）

直线型程序和分支程序

- 只有顺序组合的程序是直线型程序，只有一条执行路径：

  Python 解释器顺序执行程序里的每个语句一次

- 只包含 if 复合语句的程序称为分支程序
  - 程序里的每条基本语句最多执行一次
  - 处理不同数据时执行的语句序列（执行路径）可能不同
  - 可以列举出程序里所有可能的执行路径
- 直线型程序和分支程序描述简单，完成的工作也简单
  - 程序里的每个基本语句至多执行一次
  - 程序长度（语句总条数）是所有执行路径的长度上限
- 要完成复杂的计算，必须突破这种限制和状况

重复计算

- 最基本的复杂计算就是重复计算，例如
  - 求 1 到 10 的整数之和
  - 求 2 到 20 的偶数之乘积，等等
- 基本情况：
  - 重复计算
  - 计算中做的操作有规律
- 简单情况，重复次数确定时，可以通过一系列语句
  - 但太麻烦
  - 也不能处理一般情况
- 编程语言（包括Python）都为描述重复提供了的专门结构

循环，for 循环语句

- 实现重复执行的机制是循环语句
  - 是一类复合语句，执行中可能多次执行其成分语句
  - Python 有两种循环语句：for 语句和 while 语句
- for 语句根据循环一个控制器（迭代器）实现重复执行
- 语法（简单形式）和语义

  for 变量 in 迭代器：
  　语句组

  迭代器描述一个值序列（一系列对象，值）

  语义：对变量顺序取序列中的每个值执行语句组一次

  语句组称为循环体，其中可以用变量。for 段称循环头部

for 循环语句

- for 语句的简单实例（求 0 到 99 之和）：

```
s = 0
for x in range(100):
    s = s + x
```

- range 函数调用得到一类简单迭代器，三种调用形式：
  - range(n) 描述的序列是 0, 1, ..., n-1
  - range(m, n) 描述的序列是 m, m+1, ..., n-1

    n 小于 m 得到空序列，for 直接结束（左闭右开区间）
  - range(m, n, d) 描述序列 m, m+d, m+2*d, ... (小于 n)

    d 为负值时，是逐项减小但大于 n

小结

- 几个关键字：

  and as assert break class continue def del

  else except elif False finally for from global

  if in import is lambda None nonlocal not or

  pass return raise True try while with yield

- 几种数值类型和字符串
- 输入和输出，模块和导入，顺序结构和顺序执行
- 条件和逻辑表达式，if 语句和选择执行，语句组
- 循环结构和重复执行，for 语句