程序初步 -3

- **for 语句和 while 语句**
- 循环程序
- 交互式计算程序
- 编码的概念
- 二进制编码
- 整数、实数、字符等的编码
- 编码的计算

循环，for 循环语句（复习）

- 循环语句实现一段代码的重复执行
  - 一类复合语句，执行中可能多次执行其成分语句
  - Python 提供两种循环语句：for 语句和 while 语句
- for 语句根据循环控制器（迭代器）的要求实现重复执行
- 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）

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，其中可以使用变量，做任何事情。前面 for 段称为循环头部

for 循环程序实例

- 生成一个华氏和摄氏温度对照表

  这个程序产生一系列输出

- 求阶乘

  基本定义和参数

- 例：通过再一层循环，做三次阶乘计算

  循环可以嵌套任意层

  但过多嵌套会使程序变得很难理解

- for 语句用于描述事先可以确定方式的较为规范的重复执行
  - 按照 range 函数描述的方式循环，后面还会介绍其他控制 for 循环执行的方式
  - 比较简单的重复计算，应首先考虑能否用 for 语句描述

while 循环语句

- while 是功能更强大的循环语句，可用于描述各种复杂循环
- while 语句

  while 条件：
      语句组

  语法。语句组是循环体，第一行是循环头部

  执行方式（语义）：

  1. 条件为真时执行语句组，而后重复执行整个语句
  2. 条件为假整个 while 语句结束
- 用 for 语句写的程序，都可以改用 while 语句

  如前面写的求阶乘函数，生成摄氏华氏温度转换表
- 如果适合用 for 语句描述，程序更简单清晰。建议尽量用 for

两种循环语句的比较

- 如果不能事先确定循环方式或次数，就只能用 **while** 语句描述

  一种情况：用户输入控制循环的次数

  例：阶乘计算器（假设用户输入负数就结束）

- **for** 与 **while** 语句的对比
  - 都能实现循环体（成分）的重复执行
  - 控制重复（和结束）的方式不同
    - **for** 基于由其他结构（迭代器）确定的值序列
    - **while** 基于逻辑条件（成立则继续）
  - 如果两者都能用，用 **for** 写的程序代码通常更清晰简单

    适合用 **for** 的改用 **while**，需要引入变量，自己做增量操作
  - **while** 可能出现无穷循环（在 **IDLE** 用 **ctrl-c** 中断执行）

例：求平方根

- 求 x 的平方根
  - 数学定义：x 的平方根是满足 y*y = x 的非负数 y
  - 计算描述（怎么做）：
    1) 任取 z，
    2) 求出 y = x/z
    3) 如 y*y 足够接近 x，接受 y 作为 x 的平方根的近似值
    4) 取 z = (z + y)/2，回到 2) 重复
- 分析
  - 用 y*y == x 控制结束，出现什么情况？
  - 通过允许误差判断结束，求出的是近似值

    要求误差更小，会导致计算的时间更长

    应根据需要确定

变量更新

- 循环语句的体里经常需要更新一些变量，常见的如

  n = n + step

  fact = fact * n

- Python 提供了一组扩充赋值操作符

  `+=    -=    *=    /=    //=   %=   **=`

  ```
  fact *= n
  n += 1
  ```

- 还有几个（相应运算符是 >>, <<, &, ^, |）

  `>>=    <<=    &=    ^=    |=`

  这些是二进制位运算符，可以自己查手册。后面可能会说一下

循环控制

- while 和 for 语句都在语句头部描述循环控制

  通过循环继续的逻辑条件或循环变量的取值方式控制执行

  每次迭代执行整个循环体，然后再检查确定是否结束循环
- 有时需要在循环体的中间决定终止或控制循环体的执行。有两个循环控制语句，只能用在循环体里面：
  - break 语句使当前循环立即终止。语句形式：break
  - continue 语句结束循环体本次执行，回到循环头部

    语句形式：continue
- 注意：for 执行方式在进入时确定，例

```
n = 4
for i in range(n):
    print(i)
    n = n + 1
```

计算的基石：编码

- 人们在交流中用抽象的符号（口头或视觉的）表示事物
  - 用简单符号表示基本事物，例如
    - 用“一”或“1”表示单位数量，用名词指称事物等
  - 用符号的组合表示更复杂的事物，例如
    - 用数字的序列表示任意大的数量
    - 用一串文字表示复杂事物（如带很多定语的名词短语）
- 这里出现了事物与其符号表示之间的一对关联
  - 从事物到其符号表示（表示，表达，**representation**）
  - 从符号表示到被它表达的事物（解释，**interpretation**）
- 要用计算机处理真实世界的问题，就需要在计算机里表示这个问题，表示与该问题有关的各方面信息

数字化和编码

- 用计算机处理问题，需要把信息送入计算机，还要取得结果
  - 需要确定信息的计算机表示方式
  - 需要完成外部信息和计算机内部形式之间的转换（两个方向）
- 实际需要处理的信息丰富多彩，数的计算是基础
  - 需要为数确定计算机内部的表示方式
  - 一切信息都用数表示，以便用计算机处理，称为“数字化”
- 信息的数字化形式也称为信息的编码

  计算的基础是“所有信息”都能编码

  “万物皆数”在自然界不真，但在计算机里“成立”
- 问题：怎样数字化？怎样编码？

编码的概念

- 抽象看，编码就是用一套符号系统化地表示另一些事物（抽象的或具体的事物），支持两个方向的映射：

  被编码事物 ←解码/编码→ 其编码表示

- 例，有一种符号表示的符号集 S_1，要用另一符号集 S_2表示

  从 S_1 到 S_2 是“编码/表示”；从 S_2 回到 S_1 是“解码/解释”

- 例：考虑十进制整数集合，基本符号集 S_1 是十进制数字（注意：十进制数是整数的一种表示）。考虑集合 S_2 的符号包含 a～j 十个字母，定义对应关系 1→a, 2→b, ..., 9→i, 0→j
  - 任何整数（的十进制表示）都可以映射到 S_2 的字母串
  - 两方向的翻译直截了当，编码和解码很容易

- 一种编码，就是人为定义的一对转换规则

  公开转换规则就是“编码/解码”，不公开就是“密码/解密”

编码

- 可以用任意十个符号编码十进制数（直接对应），也可以用更大的符号集。采用大符号集时编码可能较短，即，编码有效率问题
- 也可以用小的符号集合 S_2 编码大符号集合 S_1。简单方法：用 S_2 符号足够长的串表示 S_1 的一个符号
- 用两个符号的集合作为编码符号集合 S_2：
  - 1 个符号的序列可以区分 $2 = 2^1$ 种情况
  - 2 个符号的序列可以区分 $4 = 2^2$ 种情况
  - 一般的：n 个符号的序列可以区分 2^n 种情况
- 用更大符号集只能把编码长度缩短一个线性因子，与两符号相比没有根本改进。从量级上考虑编码效率，两符号已经足够了

  可以用任意两个符号，如 a 和 b，或 4 和 9，或 □和○。实际中最常用的是 0 和 1

编码

- 计算机中用两个符号的集合编码。原因：
  - 理论：两个符号的序列足以有效表示复杂信息。（一个符号的集合表达效率低，表达能力也不够）
  - 实际：常规（电子）器件很容易表示两种不同状态。一个器件的状态表示一个基本数据单位（bit），一系列器件的状态表示一系列基本数据单位，可以表示任意复杂的信息
- 1个位：bit（Binary Digit），b/位/比特
- 8个位：Byte，B/字节
- 存储量/数据量：1KB = 1024B (2^{10}B)　1MB = 1024KB (2^{20}B)

  1GB = 1024MB (2^{30}B,吉)　1TB = 1024GB (2^{40}B, 太)

  1PB = 1024TB (2^{50}B, 派)　1EB = 1024PB (2^{60}B, 艾) ……

编码

- 计算机内部的统一编码：
  - 基本符号用 0 和 1 表示，能区分两种情况，复杂信息用一串 0/1 表示，二进制串。n 位二进制串可表示 2^n 种不同情况
  - 一切信息进入计算机，都转换为二进制串
- 五彩缤纷的大千世界，在计算机里得到了大统一

  进入计算机需编码；要知道一段代码的意义就需解码
- 把一类信息放入计算机，理论上说，可采用任意的编码方式
  - 为了交换信息，用计算机处理，就需要确定编码方式
  - 计算机需要广泛使用，因此人们制定了编码标准。要理解计算机信息处理，需要了解一些最重要的编码方式
- 需要编码的对象很多：各种数值，包括整数，实数等；字符，包括英文字母，中文字；更复杂的信息体，如图像，声音等

数制

- 以进制方式表示数是人类的重要智力发明，只用有穷的（很少）几个符号，就可以表示无穷多不同的数
- 数的进制：

  (整)数是客体，进制表示形式是数的表示形式（也是“编码”）

  整数不一定要用十进制表示（手指/十进制，电子元件/二进制）
- 十进制数 D 的值可以用如下方式分解（计算）：

  $D = k_n*10^n + \ldots + k_0*10^0 + k_{-1}*10^{-1} + \ldots + k_{-m}*10^{-m}$

  基数为10，系数/数字 $k_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

  例：$46.37_{(10)} = 4\times10^1 + 6\times10^0 + 3\times10^{-1} + 7\times10^{-2}$
- 数的常见表示形式：

  二进制, 八进制, 十进制, 十二进制, 十六进制, 六十进制

数的二进制表示（编码）

- $B = k_n*2^n + \ldots + k^0*2_0 + k_{-1}*2^{-1} + \ldots + k_{-m}*2^{-m}$

  基数是 2，系数（数字）属于 {0, 1}

- $110110_{(2)} = 1*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0$

  $= 32 + 16 + 0 + 4 + 2 + 0 = 54_{(10)}$

- $11.01_{(2)} = 1*2^1 + 1*2^0 + 0*2^{-1} + 1*2^{-2}$

  $= 2 + 1 + 0.0 + 0.25 = 3.25_{(10)}$

- 运算规则（加法和乘法，很简单）。位计算规则：

  0+0 = 0　　0+1 = 1　　1+0 = 1　　1+1 = 10（进位）

  0*0 = 0　　0*1 = 0　　1*0 = 0　　1*1 = 1

  二进制数的加法和乘法基于这些规则实现，减法转换为加法（方法后面介绍），除法比较复杂，从略

整数表示和计算

- 二进制编码长且不易读，与十进制之间转换比较麻烦
- 八进制描述较短，与二进制转换方便，1位对应于二进制3位：

  000→0　010→2　100→4　110→6

  001→1　011→3　101→5　111→7

  八进制用数字 0 到7 表示。八进制到十进制的转换：

  $1234_{(8)} = 1\times 8^3+2\times 8^2+3\times 8^1+4 = 668$

- 十六进制通常用 1~9 和 A~F 表示数字

  $1234_{(16)} = 1\times 16^3+2\times 16^2+3\times 16^1+4 = 4660$

  十六进制的1位对应二进制4位：

  0000→0　0001→1　0010→2　0011→3　0100→4　0101→5
  0110→6　0111→7　1000→8　1001→9　1010→10
  1011→11　1100→12　1101→13　1110→14　1111→15

整数的编码

- 人习惯于十进制表示，进入计算机就需要转为二进制表示，从计算机输出，需要转回十进制（人容易读）

  应该用数值相同的二进制数表示原来的十进制数（并不必须）

  在这种情况下，编码和解码就是同一个整数的数制转换

- 十进制整数数到二进制数的转换，一般用除余法。例：

```
2 | 13
  ------
 2 | 6 ------- 1
   ------
  2 | 3 ------- 0
    ------
   2 | 1 ------- 1
     ------
       0 ------- 1
```

即：$13_{(10)} = 1101_{(2)}$

写程序实现这种转换，方法：

二进制串开始为空（没有字符）

反复做，直至数变成 **0**：

  数除2的余数字符加在二进制串前面

  数本身除以2

整数的编码

- 二进制整数到十进制数的转换

  可以认为就是求二进制数的“值”，公式：

$$
\begin{aligned}
b_{n-1}\ldots b_1 b_0 &= b_{n-1}\times 2^{n-1} + \ldots + b_1 \times 2 + b_0 \\
&= ((\ldots(b_{n-1}\times 2) + \ldots)\times 2 + b_1)\times 2 + b_0
\end{aligned}
$$

- 写程序计算，输入是二进制字符串，算出其数值

  需要不断乘 2 并加相应二进制位值（0或1），应该用循环

  可以通过下标使用二进制串的内容

- Python 的字符串可以当作迭代器，得到的序列是字符串里顺序的各个字符。利用这个情况，可以稍许简化程序
- Python 里可以用二进制/八进制/十六进制形式写整数。例：

  0b100110110, 0o7654321, 0xacceded

数的编码和计算

- 实际计算机硬件采用定长二进制串表示正整数。常见：
  - 32 位编码，表示范围 $0 \sim 2^{32}-1 = 4294967295$
  - 64 位编码，范围 $0 \sim 2^{64}-1$ = 1844,6744,0737,0955,1615
  - 近似公式： $2^{10} = 1024 \approx 10^3$
- Python 的任意大整数通过软件（程序）技术实现，其中用多个计算机整数表示一个超出计算机表示范围的整数
- Python 里可以用二进制/八进制/十六进制形式写整数。例：

  0b100110110, 0o7654321, 0xacceded

  - 注意：用不同形式写出都是整数，例如

    Python 程序里写 14，0b1110, 0o16, 0xE 都表示整数 14

带符号整数的编码

- 负数通常用补码。补码的计算：

  求出其绝对值的二进制编码，各位求反后加 1
- 例：求 - 103 的 8 位二进制补码编码

  103 的 8 位二进制码是 01100111

  - 103 的补码：求反得 10011000，加一得 10011001
- 采用补码的意义：
  - 整数相加可以采用同样的计算规则，正数/负数相加可以直接计算，舍去进位（在确定的表示范围内）
  - 减法可以通过求减数的补码后相加得到
- 带符号数的表示范围：32位整数为 $-2^{31} \sim 2^{31}-1$，64位整数表示为 $-2^{63} \sim 2^{63} - 1$

小数的进制转换

- 表示实数时，需要做十进制浮点数与二进制浮点数之间的转换。整数的转换以及讨论过，现在考虑小数的转换
- 二进制小数到十进制，同样是在十进制中求值：

$$0.b_1b_2\cdots b_n = b_1 \cdot 2^{-1} + b_2 \cdot 2^{-2} + \cdots + b_n \cdot 2^{-n}$$

  例如，$0.1011_{(2)} = 0.5_{(10)} + 0.125_{(10)} + 0.0625_{(10)} = 0.6875_{(10)}$

- 十进制小数到二进制小数，用不断乘 2 并搜集整数部分的方法

  如 $0.5 \times 2 = 1.0$。取整数部分 1，二进制数是 $0.1_{(2)}$

  $0.375 \times 2 = 0.75$，$0.75 \times 2 = 1.5$，$0.5 \times 2 = 1$，$0.375_{(10)} = 0.011_{(2)}$

  注意：十进制小数一般不能用有限位二进制小数表示

  例：$0.1_{(10)} = (0.000110011\cdots)_{(2)}$, $0.7_{(10)} = (0.101100110\ldots)_{(2)}$

- 转换到有限位会引入误差。包含整数和小数部分时分别计算

实数的表示：浮点数

- 科学记数法，用尾数加数阶（指数）的形式：

  例：**0.008961**×**10^8，23.465**×**10^-12**

  不唯一，如 **0.008961**×**10^8 = 8.961**×**10^5,**

  规范表示：尾数都用一位整数

- 计算机浮点数采用二进制的科学记数法（*b* 和 *e* 都是 **0/1**）

$$\pm 1.b_1 b_2 \cdots b_n \times \pm e_0 e_1 \cdots e_m$$

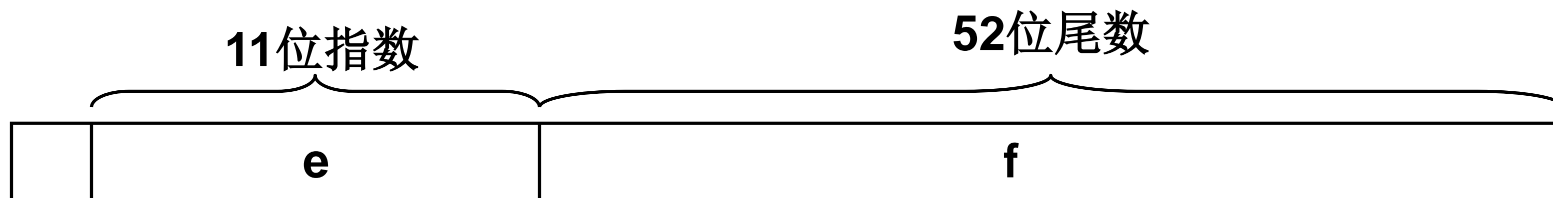
  包含 **4** 部分：指数符号和绝对值，尾数符号和绝对值。采用规范表示整数部分总是 **1**，可省略

- 计算机用定长二进制串表示浮点数。指数长度决定浮点数的表示范围，尾数长度决定数的精度

- 目前**CPU**大都采用 **IEEE 754** 标准。单精度**32**位表示，**8**位指数**23**位尾数。范围约±**(10^-45 ～ 10^38)**，约**7**位**10**进制有效数字

IEEE 754 双精度数

- 约 16～17位十进制精度：



数的符号位

f 是 52 位二进制数。规范化要求 f 的首位为 1，因为总是 1 可以省去，概念上 f 是 53 位二进制数 1.f，e 是 11 位二进制数，表示范围 –1022~1023（另外两个值另有它用）

表示的数值是：$s \times 1.f \times 2^{\pm e}$，指数和尾数都是 0 表示 0.0

全精度（尾数的个位非 0 的情况）范围：$\pm 2.23 \times 10^{-308}$ 到 $\pm 1.80 \times 10^{308}$，绝对值最小的数与 0 距离很大

IEEE 标准引入非规范数，可表示比 $1.0...0 \times 2^{-1023}$ 更小的数，用指数为 0 尾数 f 不全 0 表示 $\pm 0.e \times 2^{-1022}$。最小正数是 $0.0...01 \times 2^{-1022} = 2^{-52} \times 2^{-1022} = 2^{-1074} \approx 0.5 \times 10^{-323}$

浮点数计算

- 浮点数是近似数，计算是近似计算
  - 加减需要3步：对位，运算，规范化

    计算时就是两个二进制数相加
  - 乘法是尾数相乘指数相加，然后规范化，取高位
  - 除法比较复杂，不讨论
- 浮点计算误差的原因：
  - 精度有限，造成的表示误差
  - 转换误差，十进制小数一般不能转换为等值的二进制小数
  - 有限精度的计算误差

    误差会在计算中累积，趋势是越来越大

浮点数计算

- 两个重要情况
  - 很大的数加减很小的数，可能没效果，如

    1.23e20 + 2.33　　　**3.14e10 – 6.18e-10**

  - 两个相近的数相减，差的相对误差可能变得非常大，结果的精确度急剧恶化，甚至完全失去价值

    假设两个数均为 **16** 位十进制精度，它们的前**10**位相同，两数求出的差至多只有 **6** 位精度

- **IEEE** 浮点数表示还包括特殊值
  - 正/负无穷，**NaN**（**Not a Number**）
  - **Python** 用串 **"inf"** 和 **"Infinity"** 表示无穷，计算出错将得到 **"nan"**。用 **float("infinity")** 可得到表示正无穷的浮点数

文字信息的编码

- 文字信息是字符的序列，基于字符编码
  - 字符集：计算机上可用的基本字符集合
  - 编码的基本方法：字符集中字符排序，用字符的顺序位置作为编码（是整数，用整数的二进制编码）
- 标准字符集：人们为交换信息而定义的字符编码标准
  - ASCII 是最常用的 7 位字符集，包含十进制数字、英文字母、常用标点符号及特殊控制字符（系统用）。用一个字节编码
  - 中文字符（中文字）有多种编码标准（GB，Big5 等）。中文字符数量大，常用两个字节编码一个字（也可能用更多字节）
  - 随着国际交流日益频繁，尤其是互联网的广泛使用，迫切需要在更大范围方便地共享文字信息。国际码 Unicode 是一种统一的编码系统，在一套编码中表示世界主要文字的字符
- 如何区分一个单元里存储是是整数还是字符？

其他信息的编码和转换

- Python：
  - Python使用 Uicode 作为基础编码
  - 标准内置函数 ord(c) 给出字符 c（单字符的字符串）的编码
  - 标准内置函数 chr(i) 得到整数 i 对应的字符（单字符串）
- 信息进入/送出计算机，都需要做外部形式与对应内部形式的转换。实际转换一般由特殊硬件设备和软件合作完成
- 例：
  - 字符输入转换：人按键盘键，键盘硬件产生表示该键字符的二进制序列送入计算机，输出字符时做反方向的转换
  - 数字照相机：特殊硬件把由镜头得到的光强和色彩信息转换为二进制序列送入计算机，显示时从这种序列生成相应图像

程序的编码

- 计算机中另一类最重要的信息是程序
- 机器语言程序是机器指令序列。指令编码的基本情况：
  - 每种指令用一个数（定长或变长）表示，称为指令字
  - 内存单元顺序编号，从 **0** 开始，称为内存地址
  - 指令中需要用某单元的数据时，给出其地址
    - 存取数据指令和运算指令给出数据的地址
    - 控制转移指令指定转移的目标地址（程序里的地址）
- 高级语言程序用普通的字符文本表示，通常采用计算机上的标准字符集，采用普通的文本编码形式

  Python 采用 **Unicode**（统一码）作为其编码字符集

  对程序进行编码，使它可以由计算机自动处理的思想来自图灵

编码的一般意义

- 理论上说，任意一类可以用有限个符号表达的信息结构，都可以为其设计一种编码方式，方法：
  - 选用一个包含至少两个符号的集合 **R**
  - 定义一个表示函数 **[.]**，可能还有一个解释函数 **[.]**$^{-1}$
  - 使对属于该类的任一 **S**，**[S]** 是 **R** 中符号的序列。（如果有解释函数，则 **[[S]]**$^{-1}$ 与 **S** 有某种“等价关系”）
- **Gödel** 定义了一种从逻辑公式到整数的编码，并基于这一编码证明了“如果一个公理系统包含算术理论（即，足够复杂），它就不可能是完全的”（**Gödel** 不完全性定理）

  这个定理是20世纪人类认识论发展中的一项重要数学成果

  彻底粉碎了希尔伯特公理化数学的梦想（希尔伯特第二问题）

  但是，我们不知道公理化数学能走多远，特别是有人参与时

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

- 循环语句和重复计算
- 浮点计算误差，近似计算，循环控制
- 编码的意义
- 二进制，二进制编码
- 数制转换，转换中的问题