组合数据对象-I

- 组合数据对象

  为什么需要？有什么用？

- list（表）
- 表的构造
- 表的基本操作
- 用表解决问题（编程实例）

组合数据

- 在程序里，用变量保存计算得到的（中间）结果
  - 在后续计算中使用，或作为最终结果输出
  - 维护一些信息，支持计算过程的进行
- 在前面程序中，变量里保存都是基本类型的数据
  - 如整数、浮点数、字符串等
  - 每个变量是有名字的独立个体，不同变量互不相关
- 计算中也经常希望能统一地掌握一批数据。例如
  - 记录一个整数的所有素因子
  - 记录条钢切分中各长度分段的安排
  - 保存一个向量或一个矩阵的所有元素
  - 等等

组合数据

- 可以定义多个变量保存多项数据，但这种做法不适合有些情况：
  - 需要保存的数据值数量不定
  - 需要写循环以统一方式逐个处理这些变量的值
  - 需要把一组不定数量的值送进函数或作为函数返回值
- 如过能用一个变量记录一组数据，就能方便地处理这些问题
  - 需要组合数据对象，其值包含一批更基本的元素
  - 希望组合对象能像简单对象一样赋给变量，传进传出函数；希望能个别地处理或以统一方式处理其成分（数据）
- 对组合数据对象的希望：
  - 能包装起一批元素，可以访问和使用其中的元素
  - 又能作为一个整体保存、传递和使用

组合数据对象：表（list）

- Python 提供了多种组合数据的机制
  - 其中最常用的是 list（表，列表），list 是类型名
  - 其他机制后面介绍
- list 是 Python 的一种内置的序列类型
  - 一个表对象可包含一组元素，如 [1, 2, 3]
  - 没有元素的表称为空表，用 [ ] 表示
  - 作为序列类型的对象，每个元素在表中有确定的位置（下标）
  - 通过下标表达式访问表元素。下标表达式里可以包含变量，实际访问的元素由表达式的值确定
  - 表对象可以作为变量的值，可以作为函数的参数和返回值。程序里可以通过变量和函数实现对表对象的操作

表的构造和操作

- 可以通过直接描述建立新的表，还可以通过操作构造新表
- 用 [...] 直接描述，以表达式的形式列出表元素，逗号分隔

  例：[ ] 得到空表，[1, 2, 3] 得到一个包含三个整数的表
  - 如果表元素很少，比较方便
  - 如果各元素之间没有关系，没有统一的描述方式
  - 元素可以写任意表达式，建立表时以表达式的值作为元素
- 表的最基本操作是元素取值和赋值
  - 表元素用下标表达式描述
  - 表 lst 的元素下标从 0 到 len(lst) - 1（注意，表是一种序列）
  - lst[2] 取得 lst 里位置 2（下标为 2）的元素
  - lst[2] = 'a' 把 lst 里下标为 2 的元素改为字符串 '2'

表的构造和操作

- 用 **list(...)** 构造（是做类型转换）

  list 的参数应该是一个序列或者迭代器

  list("abc")　　得到的表相当于写 **["a", "b", "c"]**

  可以利用 **range** 构造顺序值有规律的表。例：

  list(range(100))　　得到的表里按顺序包含元素 **0, 1, …, 99**

- 如果参数不是序列或迭代器，用 **list** 转换将出错

  list(1234) 出错

  Python 把迭代器和序列统称 **iterable**，意为可迭代对象

  list 的参数必须是可迭代对象（注意，空表也是可迭代对象）

  list(1234) 的出错信息说 **1234** 不是可迭代对象

表的构造和操作

- 通过表操作构造新表
  - 表是序列，序列都可以用 + 和 * 操作
  - **lst1 + lst2** 得到一个新表，其中顺序排列两个表的元素
  - **lst * n**（**n** 为整数）得到包含 **lst** 元素 **n** 次的新表
  - **lst.append(x)** 把 **x** 加入 **lst** 作为最后一个元素
- 常用技术
  - 构造已知大小的元素类型和值相同的表（表是序列）

    一定长度全 **0** 值的表　　可以写 **[0] * n**

    一定长度无确定值的表　可以写 **[None] * n**
  - 其他表操作后面介绍，其中一些操作的结果也是表

处理表的基本方法

- **len(lst)** 得到表 **lst** 的长度（元素个数）
- 实际程序里经常需要顺序处理表里的每个元素。常用模式：

```
for i in range(len(lst)) : # 最容易想到的写法
    ... lst[i] ...

for x in lst :  # 简化写法，因为表是可迭代对象
   ... x ...
```

可迭代对象（序列或迭代器）都可作为 **for** 循环变量的取值源

- 也可以处理表中几个元素或一段元素
  - 处理个别元素时，直接通过下标访问
  - 处理表中一段元素，可以利用 **for** 循环和 **range**，通过下标表达式访问指定元素

实例：向量乘法

- 向量可以很自然地表示为（浮点数的）表
  - 现在考虑向量的乘法（点积）
  - 函数以表示向量的两个表为参数，返回一个浮点数
- 设计：
  - 先确定两个向量等长，否则返回 **0.0**（或者报告错误）
  - 循环乘两个向量的各对应项并累积
  - 返回结果
- 函数定义很简单

表的一些情况

- 表可以包含任何类型的对象作为元素
  - 允许一个表的元素属于不同类型
  - 最常用的是同类型元素的表
- 表的元素可以是任何数据对象，包括表。例如

  [[1, 2, 3, 4], ["a", "b", "c", "d"], ["age", 23]]
- 例：假设函数的参数是一个表，其中每项是一个两项的表，分别表示单价和计重，求总价
- 注意 **for** 语句头部，取固定个数元素的表元素的简化写法

  for x, y in lst : ... # 如果 **lst** 的元素都是两个成员的表

实例：生成斐波那契序列

- 写函数生成一个保存前 20 项的斐波那契序列的表
  - 显然需要一个表，可以用 [0]*20 生成
  - 需要先给第 0 和 1 项赋值
  - 后面的项基于已有值的项计算，计算规则是统一的，可以（也应该）用一个循环描述
- 不难写出程序
- 很容易定义一个函数，让它返回 n 项斐波那契序列的表
  - 虽然表在函数里定义，但其存在期不依赖于函数
  - 可以从函数里返回任何对象，包括表对象
- 基础表也可以用 [1, 1] + [0]*18 生成

实例：多项式求值

- 可以用表 **p** 里保存一元多项式，其中 **p[i]** 存系数 a_i

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_k x^k$$

- 写一个函数，计算这种表示的多项式在给定点 **x** 的值

  evalPoly(p, x) 应返回 **p** 在 **x** 的值

- **p** 是一个 **k+1** 项的表，按顺序取项计算就可以

```
val = 0
for i in range(0, len(p)) :
    val += p[i] * x**i
return val
```

函数实现很简单

实例：多项式求值

- 显然：
  - 可以不用 **x**i**
  - 而是保存 **x** 的前一幂次，每次乘 **x**
  - 不难修改函数的定义，实现这一方法

- 还可以利用多项式的 **Horner** 范型和求值规则

$$((\cdots((a_k x + a_{k-1}) \cdot x + a_{k-2}) \cdots + a_2) \cdot x + a_1) \cdot x + a_0$$

需要从表的最后一项向前处理

内置函数 **reversed** 生成反转迭代器，放在 **for** 语句里描述迭代方式。函数很简单，关键是建立好初始状态

实例：筛法求素数

- 一个经典的求素数方法称为筛法，方法是
  - 取从 2 开始的自然数序列（直到某个 n）
  - 2 是素数，从它之后的序列中划去 2 的所有倍数
  - 找到序列中下一个未划去的元素，它也是素数，从它之后的序列中划去它的所有倍数
  - 重复此操作直至序列中只剩素数（无法再划去任何的数）
- 显然，必须用一个对象表示自然数序列，实现这个算法
- 考虑用一个表实现这个自然数序列
  - 记录一串自然数
  - 通过对表的操作，实现筛法过程

实例：筛法求素数（细节考虑）

- 有关表内容的考虑（数据设计）
  - 例如，在表里保存自然数本身
  - 从 0 开始，元素与下标对应，可能操作方便
  - 用将元素置 0 表示该数已经被划掉
  - 完全可以采用其他方式，只要操作方便，表示清晰
- 操作的步骤（主要就是对表的操作）：
  - 建立自然数表，如称 numbs，建立其初始状态（包括把前两个元素置 0（0 和 1 不是素数）
  - 从 2 开始循环，直至完成工作
  - 对每个新确定的素数，划去其倍数（一个内层循环）

实例：筛法求素数（实现细节）

- 实现：
  - 如果 **n** 不是素数，它一定有小于等于其平方根的因子。循环到不小于 **n** 的平方根就可以结束
  - 考虑用一个条件为 **True** 的 **while** 循环，在循环体里检查结束条件，满足时 **break**
  - 检查并划去倍数的循环可以用一个 **for** 描述
- 运行发现结果表包含很多无用的 **0**，应去掉。可能方法：
  - 另做一个只包含素数的表，利用 **append** 操作构造
    - 展示了如何构造一个事前不知道大小的表
  - 也可以删除原表中的 **0**，但未必方便，效率也是问题