函数参数

- 计算框架
- 函数的函数参数
  - 匿名函数
  - **lambda** 表达式
- 随机数生成与模拟
- **print** 的参数和使用
- 字符串的使用和操作

递归函数，作用域和环境

- 递归函数没带来新的作用域问题

  递归函数的定义同样构成一个局部作用域

- 递归函数的每个调用建立一个新局部名字空间。考虑

```
def get_coins(n):
  def ccoins(k, n):
    if n == 0:
    ...
    return ccoins(...) ...
  return ccoins(6, n)

get_coins(20)
```

- 调用 get_coins(20) 在全局环境下建立一个局部名字空间
- get_coins调用 ccoins 建立局部名字空间，其外围是 get_coins 的名字空间
- ccoins 递归调用 ccoins 建立新局部名字空间，外围仍是 get_coins 的名字空间。再递归的情况类似

再看求立方根

- **cbrt** 的主函数体是一个操作框架
  - 逼近，就是一种不断迭代，基于一种能求出更好近似值的方法和一种（选定的）判断结束的标准
  - 这一框架不仅能用于描述求立方根的其他方法，还可能适合其他通过逼近方法求解问题的工作

    为此，只需针对具体问题找出一种求出更好近似值的方法
- 同样程序框架可能用于实现其他程序

  例如求平方根的程序等
- 可以考虑保存这种程序框架
  - 遇到类似问题时，拷贝这个框架，修改后再用
  - 但这种做法不好，基于修改程序的方式重用，不是好办法

逼近计算的框架

- 有没有可能在编程语言里描述这种计算框架？
- 注意这种计算框架的特点
  - 基本计算过程是共同的
  - 其中有些计算片段是具体的，来自需要解决的具体问题
- 回顾前面定义的函数，例如计算最大公约数
  - 基本计算过程是共同的
  - 作为计算对象的整数对是具体的，通过参数提供
- 问题：需要把计算中一些具体的计算片段参数化

  如果能这样做，就能描述上面这种计算框架

  所用语言（Python）是否支持函数的参数化？

函数的函数参数

- Python 支持以函数作为函数的参数
  - 把一些函数参数化，使我们能描述计算的框架
  - 其中的一些具体计算片段通过函数的参数提供
- 实例：逼近计算框架和求立方根函数
- 注意：
  - Python中函数的参数不需要说明类型，说函数参数是约定
  - 在函数体里，函数参数只能作为函数，在调用中使用
  - 提供给函数参数的实参，应该满足函数体里相应的函数调用式的要求（调用的形式和类型符合需要，包括实参个数和各参数的类型）。这些靠写程序的人保证
- 同一计算框架，可以用于实现不同的计算，如求平方根

例：数值积分

- 函数参数的另一个应用：数值积分，可能对许多函数做这种计算
- 这是一种很有用的数值计算，求函数定积分的数值近似
  - 简单算法采用矩形法
  - 用一个函数参数表示被积函数
  - 另外两个参数表示积分区间
  - 固定区间划分，通过全局变量设定（也可以用参数）
- 函数的实现用一个循环

```
for i in range(division) :
    s += f(a + i*d)
return s*d
```

考虑了浮点加法的累积误差和减少乘法的次数。看程序

例：数值积分（分析和讨论）

- 区间的划分数取多少合适？
  - 用户容易设定吗？
  - 用全局变量设定合适吗？
- 能不能自动确定合适的区间划分？
- 一种方法：通过多次计算积分值得到合理的近似值
  - 取一种初始划分，计算一次积分值
  - 加细划分后再次计算，例如区间数加倍
  - 用前后两次积分值的相对误差判断结果是否足够好
  - 如果认为不够好，就重新加细和计算

  这种方法的实现作为习题

匿名函数

- 重看前面借助 **apprMethod** 定义的求平方根函数和求立方根函数
  - 为完成这两个函数，我们定义了几个辅助函数
  - 这些辅助函数很特殊，多半只使用一次
  - 把它们定义在全局环境里并命名，不是最合适的方式
- 问题：能不能直接描述函数对象，不命名而直接使用？
- 数学描述 **f(x) = sin(x) + x** 属于不好的描述方法
  - 没区分函数定义和命名
  - 本身也有歧义（下面不讨论这个问题）
- 严格处理函数的定义和应用，应该使用 **λ-表达式**的概念（前面讲过，由数学家 **A Church** 开发，**lambda 表达式**）

  Python 语言有描述 **lambda** 表达式的结构

lambda 表达式

- **lambda** 表达式是 **Python** 的一种表达式
  - 用于描述小的匿名函数
  - 可以作为函数使用，作用于合适的参数就能完成计算
- 语法形式：

  lambda 参数, ... : 表达式
- 语义：

  建立一个匿名的函数对象，以 参数, ... 为形参。求值时建立所列参数与实际参数的约束，计算出表达式的值作为结果
- 使用：可以直接作为函数使用，也可以作为函数对象，传给需要函数参数的函数等
- 现在重新考虑基于逼近方法函数定义 **sqrt** 和 **cbrt**

牛顿迭代法求根

- 求函数 *f* 的根的牛顿迭代公式是

$$x_{n+1} = x_n - f(x_n)/f'(x_n)$$

  这里要求 *f* 是一个可导的实函数

- 可以定义一个求函数根的函数 **newton**，它有两个函数参数，一个是被求根函数，另一个是其导函数，还有一个初值参数
- **newton** 的使用有些麻烦。对一个函数求根，需要人工求出其导函数，再定义为一个 **Python** 函数
  - 需要人工处理，不自动化
  - 在 **Python** 里可以定义的函数未必有显式的数学表达形式，可能无法写出其导函数的数学表示
- **lambda** 表达式可以用在这里，从一个已有函数，生成另一个函数（下面让它生成给定函数的“导函数”）

生成一个函数

- 微积分中求导是从一个函数得到另一函数，现在考虑数值微分

  D f(x) = (f(x + d) – f(x))/d　　其中 **d** 为很小的正实数

  这并不是真正的导函数，而是导函数的近似函数

- 定义函数 **diff** 实现数值微分，也就是说，把 **diff** 作用到一个函数上将得到它的数值微分函数（可像其他函数用）

  Python 函数 **diff** 很特殊：它从一个函数生成另一个函数

  生成函数的功能有很多应用

- 利用 **diff** 可以实现一个通用的牛顿迭代法求根函数，其参数是
  - 被求根函数和一个初值（不能是导函数的 **0** 点）
  - 返回对这个函数的求根结果
- **lambda** 可用于实现从一个函数“计算”出另一个函数

随机数和模拟

- 计算机完成的计算通常是确定性的
  - 但我们有时希望在计算中引进一些随机因素
  - 例如，做现实世界中的过程的模拟
- **Python** 标准包 **random** 提供随机数功能（标准库手册）
- **random** 包里的几个常用函数
  - **random()** 得到 **[0.0, 1.0)** 中的随机浮点数
  - **randrange(n)**，**randrange(m,n)**，**randrange(m,n,d)** 得到区间里的一个随机整数（参考 **range** 规定）
  - **randint(m, n)** 相当于 **randrange(m, n+1)**
  - **choice(s)** 从字符串 **s** 里随机选出一个字符
  - **seed(n), seed()** 用 **n** 或系统时间重置随机数生成器。用于重启一个随机序列

随机数和模拟

- 已知两个正整数互素的概率为 $6/\pi^2$。现在想写一个程序，验证这一概率的正确性
- 考虑做蒙特卡罗模拟，也就是说，对这个问题做一系列的随机试验，统计其中正反两面的证据。
- 模拟框架（做 n 次试验）：

```
for i in range(n) :
    做试验
    用变量 m 记录通过试验的次数
return m/n
```

- 试验：生成两个随机正整数，判断它们是否互素

  如果互素就是试验通过，否则是没通过，用变量记录通过次数

  求出通过次数除以试验次数的比，由此算出结果

随机数和模拟

- 注意这里的程序
  - 函数 **monteCarlo(test, num)** 描述里一个“抽象的”蒙特卡罗试验的计算框架
  - 参数 **test** 是实现具体试验的函数，它应该是一个无参函数，通过返回真假值表示一次试验是否通过
  - **num** 参数是试验的次数
- **monteCarlo (test, num)** 描述了蒙特卡罗试验的过程
  - 可以用于实现各种具体的蒙特卡罗试验
  - 注意：函数的函数参数，和适当的函数抽象设计在这个程序的组织中起着重要作用
  - 显然，试验中需要用 **Python** 的随机数功能

输出函数 print

- 前面介绍了 print 的最简单用法
  - 它输出一系列表达式的值，空格分隔，结束时换行
  - 简单使用可以满足多数程序输出的需要
  - 可以自己用 str 的转换和拼接等构造所需的输出串
  - 实际上，调用 print 时还有几个可用选项，用于控制 print 函数的行为。下面介绍一些情况
- 函数的选项通过特殊的关键字参数的形式描述
  - 关键字参数可以省略，省略时自动采用该参数的默认值
  - 对应关键字参数的实参形式是 关键字=表达式
  - 自己定义函数时也可以定义关键字参数（后面介绍）
- 下面介绍 print 的两个关键字参数，其他情况见参考材料

输出函数 print

- print 的关键字参数 sep
  - sep 的默认值是只包含一个空格的字符串
  - 作为 sep 值的字符串作为输出项之间的分隔符
  - 例：输出项用逗号空格分隔，print(…., sep=", ")
- 关键字参数 end
  - 默认值是只包含一个换行符的字符串
  - 每次调用 print 的最后输出作为 end 值的字符串
  - 如果不想 print 的最后换行，希望在最后输出逗号和空格，可以写 print(…., end=", ")
- 在调用函数时，所有关键字参数应该写在其他实际参数之后，关键字参数的顺序并不重要，但不能重复

字符串（复习）

- 字符串是一种基本数据类型
  - 类型名：**str**
  - 字面量形式：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

    还可以用连续三个单引号或三个双引号作为“括号”
  - 语义：得到相应的字符串对象
- 几个问题
  - 单个单引号或双引号括起的字符串中间不能换行
  - 连续写出几个字符串自动拼成一个长字符串
  - 三引号形式的字符串中间可以换行，换行符，换行后的空格等都看作字符串的内容

字符串

- 特殊字符（常用的，其他见参考材料）

  \n 表示换行字符　　**** 表示字符 ****

  \' 表示单引号　　**\"** 表示双引号

  其他见语言手册 **2.4.1, String and Bytes literals** 一节

- 两个主要的字符串构造操作（**s, t** 表示字符串，**n** 表示整数）

  s + t　拼接两个字符串，得到拼接串

  结果字符串的前一段是 **s** 的拷贝，后一段是 **t** 的拷贝

  "abc" + "123" 得到 **"abc123"**

  s * n 或 **n * s**　做出 **s** 的 **n** 个拷贝拼接而成的串

  两种写法等价。如，**"ab" * 2** 和 **2 * "ab"** 都得到 **"abab"**

字符串操作

- 字符串的“长度”就是其中的字符个数

  len(s) 得到字符串 **s** 的长度

  长度等于 **0** 的串称为空串

- 字符串里每个字符有一个位置，位置称为“下标”，从 **0** 开始计数，直至 **len(s) - 1**

- 下标表达式 **s[k]** 确定 **s** 中下标为 **k** 的字符

  s[0]　给出 **s** 的首字符

  s[-1]　给出 **s** 的末字符，类似地可写 **s[-2]** 等

  s[n]　给出 **s** 第 **n** 个字符

  取字符，实际得到的是指定位置的字符形成的单字符串

  如果下标超出这个字符串的范围，系统报错

在字符串上循环

- 可以通过下标在字符串上循环

```
i = 0
while i < len(s) :
    print(s[i])
    i += 1
```

```
for i in range(len(s)):
    print(s[i])
```

- Python 把字符串看作一种序列（sequence），序列对象的特点是包含有序的一系列元素，常需要逐一操作其中的元素。字符串的元素就是其中的字符
- 为了方便对序列中各元素的操作，Python 允许直接把序列对象作为迭代器，用在 for 语句里和其他需要迭代器的地方

  用 for 描述输出字符串里字符，很方便（其他操作类似）

```
for c in s:
    print(c)
```

字符串切片（slice）

- 对于序列对象，可以做切片。一个对象的切片就是用从该对象里选出的一些元素做成的另一个同类型对象
- 下面用字符串介绍 **Python** 的切片描述方法（其他序列对象的切片操作也都是这样描述）：

  s[m : n] 得到由 **s** 里下标为 **m, m+1, ..., n-1** 的字符构成的字符串（**[ ]** 里的表达式描述切片）

  s[: n] 得到由 **s** 里下标为 **0, 1, ..., n-1** 的字符构成的串

  s[m :] 是 **s** 下标为 **m, m+1, ..., len(s)-1** 的字符构成的串

  s[:] 得到 **s** 的一个拷贝

  s[m:n:d] 得到 **s** 中下标为 **m, m+1, ..., n-1** 位置的字符中间隔为 **d** 的字符构成的串。这里的 **m, n** 也可以省略，表示从头开始或到结束，但 **:** 不能省略。例如 **s[::2]**

字符串是“不变”对象

- 建立了一个字符串之后，不能修改其内容
  - 前面所有操作都是建立新字符串（不是修改已有字符串）
  - 对于基本类型的对象做运算，都是创建新对象
- 一些例子：

```
y = 2
x = y + 1
x += 3
ss = s = "Thank"
s += " you! "
s *= 2
print(ss)
print(s)
```

字符串处理函数

- 定义函数，判断某个字符是否在一个字符串里出现
  - 显然需要两个参数，检索用字符和被检索字符串
  - 需要一个个比较，用 **for** 循环最简单
  - 返回 **True** 或 **False** 表示出现或不出现
- 统计一个字符在一段正文里出现的次数
  - 正文也用字符串表示
  - 用 **for** 循环比较一个个字符
  - 用一个计数变量，初始值为 **0**，发现要找的字符就加一
  - 返回计数值
- 实际上，**str** 类型已经提供了这些操作

字符与编码

- 每个字符有一个编码
- 内置函数 **ord(c)** 给出字符 **c**的编码（**c** 必须是只包含一个字符的字符串）
  - 例如 **ord("a")** 得到 **97**
  - 不同字符的编码不同，字符也根据编码排大小（排序），称为字符序
- 从编码得到字符
  - 内置函数 **chr(n)** 给出编码为整数 **n** 的字符
  - 字符与编码一一对应