上机1：Python数值分析

龚怡

2167570874@qq.com

1.1 Python数值计算程序

- 编程语言Python的特点：
  - 简洁、高效的表达能力
  - 配备各种软件库，即模块
- 本课程使用软件：
  - Anaconda，Python 3

    https://www.anaconda.com/download

    百度网盘Python3.8.3

    https://pan.baidu.com/s/1iXhXryPJG-YNYF-RedTZ1Q

    密码: 57fs

目录

- 理论教学：Python核心数据类型
- 实验1：编码运行理论课程序1.1-1.5
- 实验2：减少运算次数的实验
- 实验3：求解非线性方程的二分法实现

Python核心数据类型

表 4-1：内置对象

对象类型	字面量 / 构造示例
数字	1234, 3.1415, 3+4j,0b111,Decimal(),Fraction()
字符串	'spam', "Bob's" ,b'a\x01c',u'sp\xc4m'
列表	[1,[2,'three'],4.5],list(range(10))
字典	{'food':'spam','taste':'yum'}, dict(hours=10)
元组	(1,'spam',4,'U'),tuple('spam'),namedtuple
文件	open('egg.txt'),open(r'C:\ham.bin','wb')
集合	set('abc'), {'a', 'b', 'c'}
其他核心类型	类型、None、布尔型
程序单元类型	函数、模块、类（位于本书第四、五、六部分）
Python 实现相关类型	已编译代码、调用栈跟踪（位于本书第四、八部分）

Python数字

- 整数：可以增长为任意位数的数字（内存允许）
- 浮点数对象：带有小数点或科学计数标志e或E

字面量	解释
`1234, -24, 0, 99999999999999`	整数（无大小限制）
`1.23, 1., 3.14e-10, 4E210, 4.0e+210`	浮点数
`0o177, 0x9ff, 0b101010`	Python 3.X 中的八进制、十六进制和二进制字面量
`0177, 0o177, 0x9ff, 0b101010`	Python 2.X 中的两种八进制、十六进制和二进制字面量
`3+4j, 3.0+4.0j, 3J`	复数字面量
`set('spam'), {1, 2, 3, 4}`	集合：2.X 和 3.X 中的构造形式
`Decimal('1.0'), Fraction(1, 3)`	小数和分数扩展类型
`bool(X), True, False`	布尔类型和字面量

Python整数

- 写成十进制的数字的串
- 十六、八、二进制在代码中都对应整数对象，只是特定值的不同语法表示
- 内置函数hex(*I*)，oct(*I*)和bin(*I*)把一个整数转换为十六、八、二进制表示的字符串
- int(*str*, *base*)根据给定*base*进制转为十进制整数

Python浮点数

- 浮点数对象在表达式中，Python将启用浮点数（而不是整数）的运算法则。
- 浮点数在标准Cpython中采用C语言中的“双精度”来实现，其精度与用来构架Python解释器的C编译器所给定的双精度一样。

Python表达式运算符

表 5-2：Python 表达式运算符及程序

运算符	描述
yield x	生成器函数 send 协议
lambda args: expression	创建匿名函数
x if y else z	三元选择表达式（仅当 y 为真时，x 才会被计算）
x or y	逻辑或（仅当 x 为假时，y 才会被计算）
x and y	逻辑与（仅当 x 为真时，y 才会被计算）
not x	逻辑非
x in y, x not in y	成员关系（可迭代对象、集合）
x is y, x is not y	对象同一性测试
x < y, x <= y, x > y, x >= y	大小比较、集合的子集和超集
x == y, x != y	值等价性运算符
x | y	按位或、集合并集
x ^ y	按位异或、集合对称差集
x & y	按位与、集合交集
x << y, x >> y	将 x 左移或右移 y 位

Python表达式运算符

x + y	加法、拼接
x - y	减法、集合差集
x * y	乘法、重复
x % y	求余数、格式化
x / y, x // y	真除法、向下取整除法
-x, +x	取负、取正
~x	按位非（取反码）
x ** y	幂运算（指数）
x[i]	索引（序列、映射等）
x[i:j:k]	分片
x(...)	调用（函数、方法、类，其他可调用对象）
x.attr	属性引用
(...)	元组、表达式、生成器表达式
[...]	列表、列表推导
{...}	字典、集合、集合与字典推导

Python运算符

- X / Y 执行真除法（保留商的小数部分）
- X // Y 执行向下取整除法（商的小数去掉）

```
>>> 6 / 2.5

2.4

>>> 6 // 2.5

2.0
```

Python运算符

- 比较运算符可以链式使用，比如
- X < Y < Z 等同于 X < Y and Y < X
- Python3.X中，比较非数字的混合类型的相对大小是不允许的，会引发异常。

数值的显示格式

- 小数位的显示

```
>>> num = 1 / 3.0
>>> num

0.3333333333333333
```

- 使用print显示

```
>>> print("num = ",num)

num =  0.3333333333333333
```

数值的显示格式

- 小数的字符串格式化显示

```
>>> '%e' % num
'3.333333e-01'
>>> '%.2f' % num        #显示2位小数点
'0.33'
```

混合类型向上转换

- 整数与浮点数相加 – 自动向复杂的操作数类型转换

```
>>> 40 + 3.14

43.14
```

- 调用内置函数强制转换类型 Python一般不需要

```
>>> int(3.1415)
3
>>> float(3)
3.0
```

- 自动转换仅限于数值类型。
- 字符串和整数相加会产生错误，除非手动转换类型

超大整数（无限制精度长整数）

- 2的100次幂

```
>>> 2 ** 100

1267650600228229401496703205376
```

- 2的1000000次幂（6个0）

```
>>> 2 ** 1000000    #慢着，你确认要输出？？

>>> len(str(2 ** 1000000))    #先看看有多少个数字吧

301030
```

PI和开平方根

- 圆周率pi

```
>>> import math
>>> math.pi

3.141592653589793
```

- 开平方根

```
>>> import math
>>> math.sqrt(2)          # 等同于2**0.5

1.4142135623730951
```

Python3和Python2在除法上的区别

- Python3.X
- Python2.X

C:\Python33\python
>>> 10 / 4
2.5
>>> 10 / 4.0
2.5
 	
>>> 10 // 4 #向下取整除法
2
>>> 10 // 4.0
2.0
	C:\Python27\python
	>>> 10 / 4
	2
	>>> 10 / 4.0
	2.5
	
	>>> 10 // 4 #向下取整除法
	2
	>>> 10 // 4.0
	2.0

- Python2.X中的 / 执行如同C语言的整数除法
- Python3.X已经去掉，改为真除法，即浮点数

Python3和Python2在除法上的区别

- 后果：Python3.X中的非截断行为可能会影响到大量的Python2.X程序。

- 解决：如果你的程序依赖于截断整数除法，在Python2.X和3.X中都使用 // 操作

奇怪的计算结果

- 加法结果怎么不对？？

```
>>> 1.1 + 2.2

3.3000000000000003
```

- 化整误差是数值编程的基本问题，不只在Python中出现
- Python中的处理方式是使用十进制数（固定精度浮点数）和分数

```
>>> from decimal import *
>>> Decimal("1.1") + Decimal("2.1")

Decimal('3.2')
```

Python中的变量

- 变量在第一次赋值时被创建。
- 变量在表达式中使用时，会被替换成它们的值。
- 变量在表达式中使用之前，必须已被赋值。
- 变量引用对象，而且从不需要事先声明。

目录

- 理论教学：Python核心数据类型
- 实验1：编码运行理论课程序1.1-1.5
- 实验2：减少运算次数的实验
- 实验3：求解非线性方程的二分法实现

1.2 Python实例1-有效位丢失

- 数值误差的实例分析1 – 有效位丢失

$$\sqrt{x+1}-\sqrt{x}$$

什么时候会出问题？

```
# 程序1.1 利用math模块求平方根
import math

x = float(input("输入希望求正平方根的值：")) 	#输入

print("sqrt(", x, ") = ", math.sqrt(x)) 	#输出
```

1.2 Python实例1 $\sqrt{x+1}-\sqrt{x}$

```
result1 = math.sqrt(x+1) - math.sqrt(x)

print("普通计算方法", result1)     #输出
```

- 当 x = 1e15

  输出为1.862645149230957e-08

- 当 x = 1e16

  输出为0.0

近似值应该是5e-09吧！！？

1.2 Python实例1

- 解决方法：避免数值近似相同的数做减法

$$\sqrt{x+1}-\sqrt{x}=(\sqrt{x+1}-\sqrt{x})\frac{\sqrt{x+1}+\sqrt{x}}{\sqrt{x+1}+\sqrt{x}}$$

$$=\frac{1}{\sqrt{x+1}+\sqrt{x}}$$

```
result2 = 1 / (math.sqrt(x+1) + math.sqrt(x))

print("变换公式方法", result2)     #输出
```

- 当 x = 1e16，输出为5e-09

1.3 Python实例2 – 化整误差

- 数值误差的实例分析2 – rounding error

  把0.1累加100万次

```
# 程序1.2
x = 0.0

for i in range (1000000):
    x = x + 0.1

print("sum result = ", x)     #输出
```

输出为100000.00000133288　怎么不是100000？

1.3 Python实例2 － 化整误差

- 原因是化整误差（rounding error）
  - 在运用有效位数的二进制表示数值的电脑中，只要表示实数，不可避免就会出现误差。
    - 十进制的0.1在变为二进制的过程中，成为循环小数

      $(0.1)_{10} = (0.0001100110011\cdots)_2$
    - 发生化整，变成了比0.1稍大的值。
  - 计算过程中不应该采用诸如上述化整误差产生巨大影响的算法。

1.3 Python实例3 – 尾数丢失

- 数值误差的实例分析3 – 尾数丢失

$$10^{10} + \underbrace{10^{-8} + \cdots + 10^{-8}}_{\text{累加10,000,000次}} = 10^{10} + 0.1$$

```
# 程序1.3
x = 1e10
y = 1e-8

for i in range (10000000):
    x = x + y

print(x)    #输出为10000000000.0
```

怎么不是10000000000.1？

1.3 Python实例3 － 尾数丢失

- 数值误差的实例分析3 － 尾数丢失

$$10^{10}+\underbrace{10^{-8}+\cdots+10^{-8}}_{\text{累加}10,000,000\text{次}}=10^{10}+0.1$$

```
# 先累加小的值，再加到大数上
x = 1e10
y = 1e-8

temp = 0
for i in range (10000000):
    temp += y
x += temp
print(x)     #输出
```

这说明了：编码前，要下功夫思考方法

输出是10000000000.1

1.4 Python实例4 – 解决误差的模块

- decimal模块 – 正确管理二进制浮点数

  把0.1累加100万次

```
# 程序1.4
from decimal import *

x = Decimal("0.0")

for i in range (1000000):
    x = x + Decimal("0.1")      #十进制的0.1

print("sum result = ", x)    #输出
```

结果为100000.0
不受化整误差的影响

1.5 Python实例5 – 分数计算模块

- fractions模块 – 直接分数计算

$$\frac{1}{3} \to \mathrm{Fraction}(1,3) \qquad \frac{5}{4} \to \mathrm{Fraction}(5,4)$$

```
# 程序1.5
from fractions import Fraction

                                        #输出5/10和3/15约分
print(Fraction(5, 10), Fraction(3, 15))
                                                   # 1/3 + 1/7
print(Fraction(1, 3) + Fraction(1, 7))
                                             # 5/3 * 6/7 * 3/2
print(Fraction(5, 3) * Fraction(6, 7) * Fraction(3, 2))
```

目录

- 理论教学：Python核心数据类型
- 实验1：编码运行理论课程序1.1-1.5
- 实验2：减少运算次数的实验
- 实验3：求解非线性方程的二分法实现

2 实验2：减少运算次数的实验

- **实验目的：**

  比较不同算法求多项式的运算次数与用时。

```
from time import *                #引入时间库
startT = time()                   #记录起始时间
# 程序写在这里。。。
endT = time()                     #记录结束时间
print("time = %.2g 秒\n" % (endT - startT))
```

```
countMul = 0                      #统计乘法次数
countAdd = 0                      #统计加法次数


print("乘法次数",countMul)
print("加法次数",countAdd)
```

2 实验2：减少运算次数的实验

例：计算函数的值（x=0.1，1，2）

$$f_n(x)=1+2x+3x^2+\cdots+100001x^{100000}$$

- **算法1：**

  直接法

```
#程序1.6
x = 1         #自变量 x
f = 1         #函数值 f

for i in range (100000):      # i从0开始
    f = f + 自己写             #函数

print("result = ", f)        #输出
```

2 实验2：减少运算次数的实验

例：计算函数的值（x=0.1，1，2）

$$f_n(x) = 1 + 2x + 3x^2 + \cdots + 100001x^{100000}$$

- **填写上机报告中的以下表格**

x	算法	函数结果f	乘法次数	加法次数	用时(秒)
0.1	算法1				
	算法2				
1	算法1				
	算法2				
2	算法1				
	算法2				

2 实验2：减少运算次数的实验

例：计算函数的值（x=0.1，1，2）

$$f_n(x) = 1 + 2x + 3x^2 + \cdots + 100001x^{100000}$$

- **算法2（秦九韶法）：**

$$f_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

$$\begin{cases} S_n = a_n \\ S_k = xS_{k+1} + a_k, \quad k = n-1, n-2, \cdots, 1, 0 \\ f_n(x) = S_0 \end{cases}$$

2 实验2：减少运算次数的实验

- **for循环使用range函数的说明**

for i in range (10): #这里的range (10) 即range(0, 10, 1)，按步长1生成从0开始到10-1的数列 使用list(range(0, 10)) 可以输出 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for i in range (4,0,-1): #按步长-1生成从4开始结束于0-(-1)的数列 使用list(range(4,0,-1)) 可以输出 [4, 3, 2, 1]

2 实验2：减少运算次数的实验

- **算法2（秦九韶法）：**

#程序1.7

$$f_n(x) = 1 + 2x + 3x^2 + \cdots + 100001x^{100000}$$

```
from time import *    #时间统计库

x = 1              #自变量 x
powN = 100000      #最后一个数的幂次
aN = powN + 1      #最后一个系数值
countMul = 0       #统计乘法次数
countAdd = 0       #统计加法次数

startT = time() #记录起始时间

S = aN              #函数值
for i in range (p          ): # i从powN开始到1
    S =                     #迭代函数
    countAdd += 1           #此处只统计算法的加法，忽略i的计数
    countMul += 1           #每次增加的乘法次数

endT = time()    #记录结束时间

print("result = ", S)       #输出
print("乘法次数",countMul)
print("加法次数",countAdd)
print("time = %.2g 秒\n" % (endT - startT))
```

自己写

自己写

目录

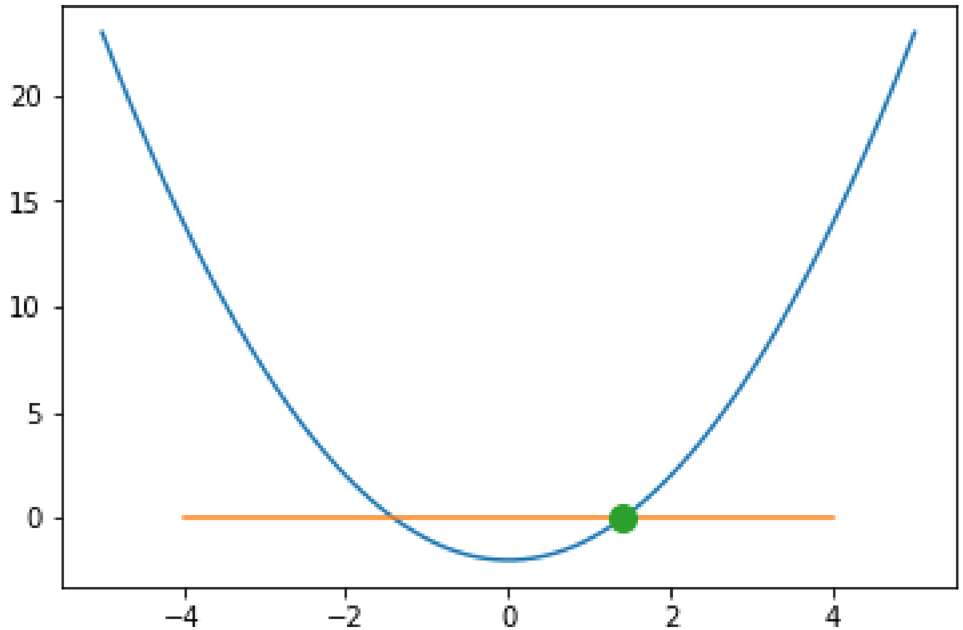
- 理论教学：Python核心数据类型
- 实验1：编码运行理论课程序1.1-1.5
- 实验2：减少运算次数的实验
- 实验3：求解非线性方程的二分法实现

3 实验3： 求解非线性方程的二分法实现

- **例：** 用二分法求解方程在[1.3, 1.5]上的解。

$$f(x) = x^2 - 2 = 0$$

- 步骤1： 画图看看解的位置



■ 画图：

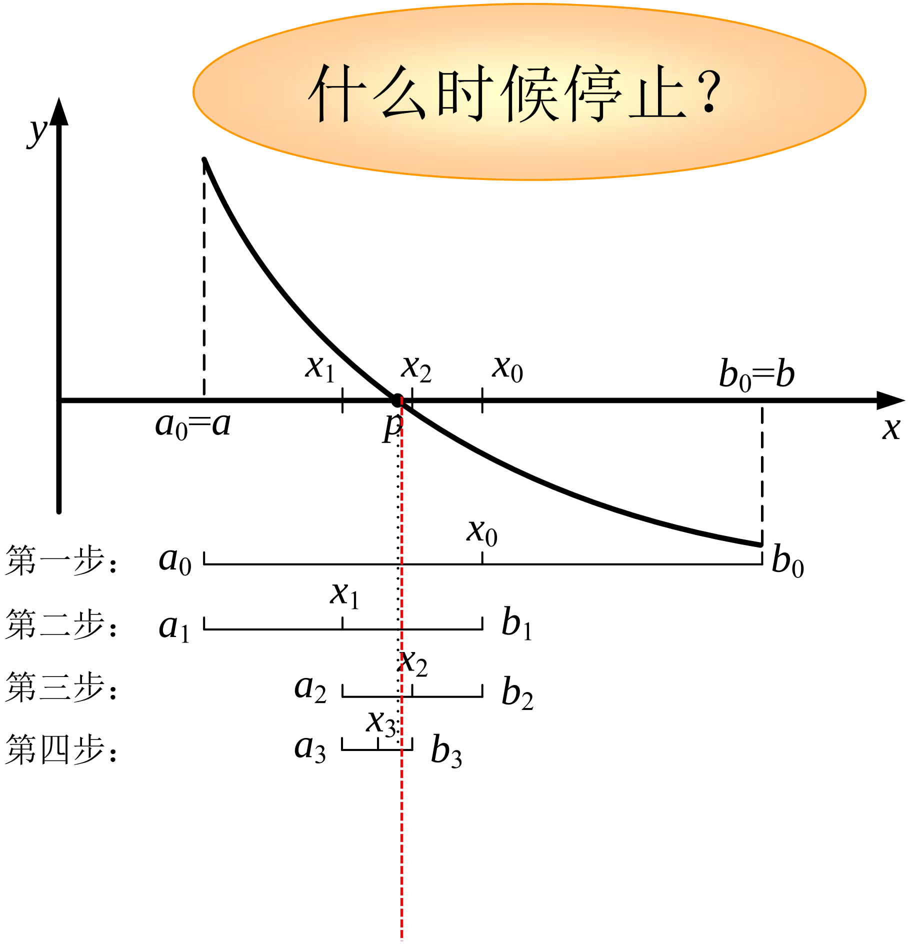
- 调用绘图库：matplotlib
- 调用数值计算程序包：NumPy

#程序1.8

```
#使用NumPy科学计算程序包
import numpy as np
#用plt输入matplotlib的pyplot
import matplotlib.pyplot as plt

#设定x轴的范围和精度，生成一组等间距的数据
x = linspace(-5,5,100)          #获得x坐标数组
#step = 0.01       #画图点之间的步长距离
#x = np.arange(-5,5+step,step)  #获得x坐标数组

y = x * x -2        #函数y = f(x)值

plt.figure()        #创建figure对象

#设定为1个图表表示
#plt.subplot(1,1,1)

#线形图
plt.plot(x, y, label = 'line')  #绘制关于x和y的折线图
plt.plot([-4,4], [0,0])         #绘制点(-4,0)到点(4,0)的直线
#绘制符合要求的方程解
plt.plot(2 ** 0.5, 0, marker = 'o', markersize = 10)
```

二分法（Bisection or Binary-search method）

- **原理**：若 $f \in C[a, b]$，且 $f(a) \cdot f(b) < 0$，则 f 在 (a, b) 上必有一根。



第一步，取$a_0=a$，$b_0=b$，中点 $x_0=\dfrac{a_0+b_0}{2}$，如果$f(x_0)=0$，那么$p=x_0$，即x_0为根，算法停止。否则，如果$f(a_0)f(x_0)<0$，取$a_1=a_0$，$b_1=x_0$；如果$f(x_0)f(b_0)<0$，取$a_1=x_0$，$b_1=b_0$，此时方程的有根区间缩小为$[a_1,b_1]$。

第二步，区间$[a_1,b_1]$的中点为 $x_1=\dfrac{a_1+b_1}{2}$，如果$f(x_1)=0$，那么$p=x_1$，即x_1为根，算法停止。否则，如果$f(a_1)f(x_1)<0$，取$a_2=a_1$，$b_2=x_1$；如果$f(x_1)f(b_1)<0$，取$a_2=x_1$，$b_2=b_1$，此时方程的有根区间缩小为$[a_2,b_2]$。

重复上述步骤，不断缩小有根区间，直到找到满足精度的解（如左图所示）。

二分法解方程程序

#程序1.9

```
a = 2           # f(x) = x*x - a
LIMIT = 1e-20   #终止条件

#方程函数f()定义
def f(x):
    """函数值的计算"""
    return x * x - a
# f()函数结束

#-----主执行部分------
    #初始设置
xlow = float(input("请输入x值下限:"))
xup = float(input("请输入x值上限:"))

#循环处理
iter = 0      #迭代计数
while (xup - xlow) * (xup - xlow) > LIMIT: #满足终止条件前循环
                                #计算新的中值点
                                #迭代计数加1
        自己写                   #中点函数值为正
                                #更新xup
                                #中点函数值为负
                                #更新xlow
    print("{:.15g} {:.15g} {:.15g}".format(iter,xlow, xup))
```

3 实验3：求解非线性方程的二分法实现

- **填写上机报告中的以下表格**

迭代次数	下限x_{low}	上限x_{up}	$(x_{up}-x_{low})/2$	$f((x_{up}-x_{low})/2)$ 的正负性
0	1.3	1.5	1.4	<0

上机课任务

- 完成程序1.1-程序1.9
- 完成“上机1实验报告.docx”，需补充内容，并通过email发送给老师。

附录：实验2参考答案

例：计算函数的值（*x*=0.1，1，10）

$$f_n(x) = 1 + 2x + 3x^2 + \cdots + 100001x^{100000}$$

- **参考答案**

计算机配置

Intel(R) Core(TM) i7-7920HQ CPU @ 3.10GHz 3.10 GHz

4.00 GB

64 位操作系统

x	算法	函数结果f	乘法次数	加法次数	用时(秒)
0.1	算法1	1.234567901234568	5000050000	100000	0.062
	算法2	1.234567901234568	100000	100000	0.034
1	算法1	5000150001	5000050000	100000	0.084
	算法2	5000150001	100000	100000	0.034
2	算法1	长度30109	5000050000	100000	13
	算法2	长度30109	100000	100000	0.36

附录：实验2参考答案

参考程序1.6:

```
from time import *    #时间统计库

x = 1          #自变量 x
f = 1          #函数值 f
countMul = 0     #统计乘法次数
countAdd = 0     #统计加法次数

startT = time()   #记录起始时间

for i in range (100000):        # i从0开始
    f = f + (i+2) * x**(i+1)        #函数
    countAdd += 1        #此处只统计算法的加法，不含i+1
    countMul += i+1        #每次增加的乘法次数

endT = time()    #记录结束时间

print("result = ", f)      #输出
print("乘法次数",countMul)
print("加法次数",countAdd)
print("time = %.2g 秒\n" % (endT - startT))
```

附录：实验2参考答案

参考程序1.7：

```
from time import *    #时间统计库

x = 1                #自变量 x
powN = 100000        #最后一个数的幂次
aN = powN + 1        #最后一个系数值
countMul = 0         #统计乘法次数
countAdd = 0         #统计加法次数

startT = time() #记录起始时间

S = aN               #函数值
for i in range (powN,0,-1): # i从powN开始到1
    S = x * S + i           #迭代函数
    countAdd += 1           #此处只统计算法的加法，忽略i的计数
    countMul += 1           #每次增加的乘法次数

endT = time()    #记录结束时间

print("result = ", S)     #输出
print("乘法次数",countMul)
print("加法次数",countAdd)
print("time = %.2g 秒\n" % (endT - startT))
```