廣東工業大學
Guangdong University of Technology

Practice 1：Python Numerical Method

GongYi龚怡

2167570874@qq.com

1.1 Python Numerical computing program

- Features of Python：
  - Simple and efficient expression ability
  - Equipped with various modules

- This course uses software：

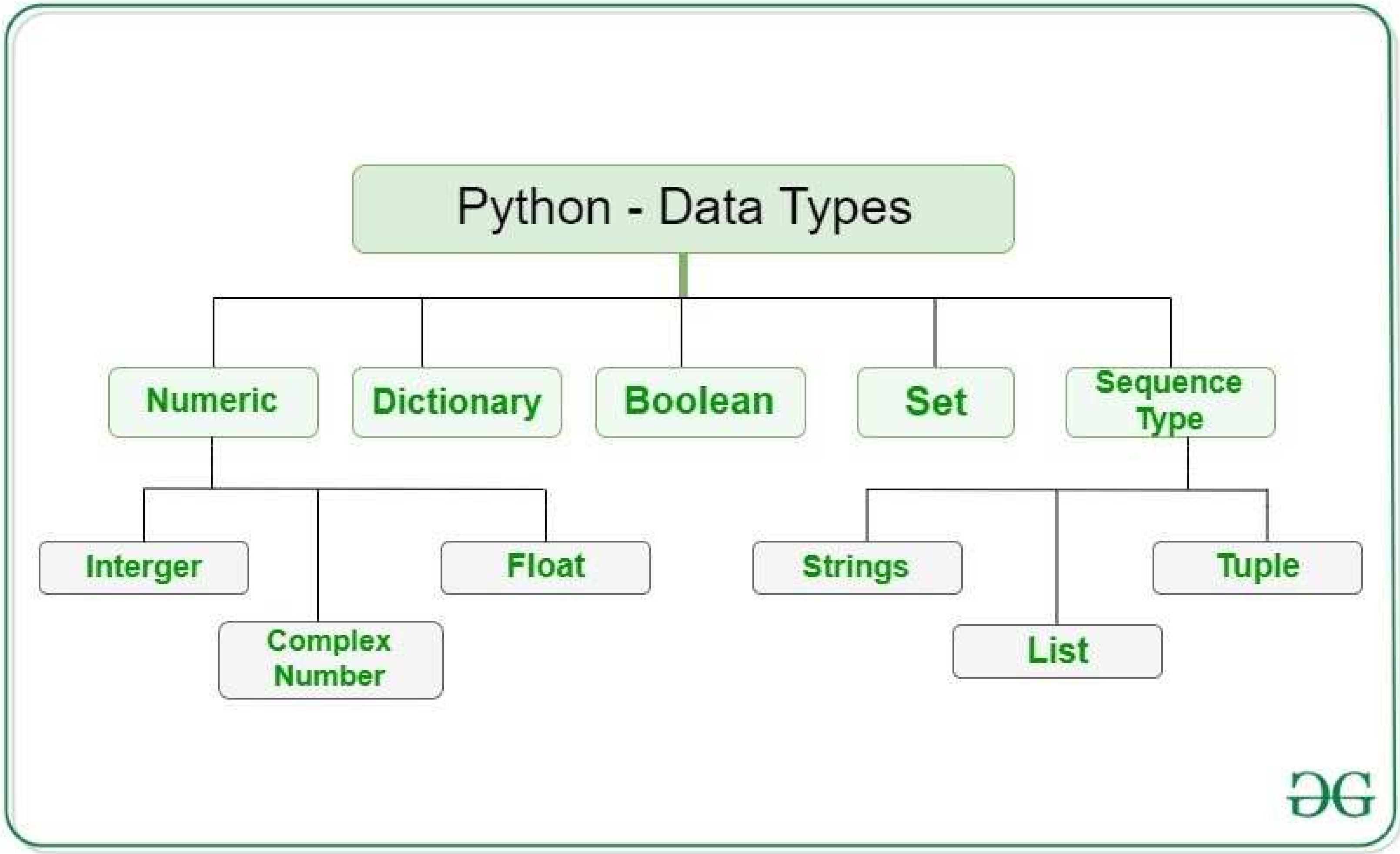
https://pan.baidu.com/s/1iXhXryPJG-YNYF-RedTZ1Q

Access code : 57fs

Contents

- Theory Teaching：Python Core Data Type
- Practice1：coding and runing programs 1.1-1.5
- Practice2：Practice to reduce the number of operations
- Practice3：Bisection algorithm for nonlinear equations

Python Build-in Core Data Type



Python Build-in Core Data Type

- Numbers
- Strings
- List
- Tuple
- Dictionary
- Set

Numerical Types

- int（signed int type）
- long（ Long integer, It can also represent octal and hexadecimal）
- float（ float type）
- complex（ complex type）

Python Number(1)

- int: positive or negative whole numbers (without a fractional part) e.g. -10, 10, 456, 4654654.
- float: any real number with a floating-point representation in which a fractional component is denoted by a decimal symbol or scientific notation e.g. 1.23, 3.4556789e2.
- complex: a number with a real and imaginary component represented as x + 2y.

Python Number Types

- 1234， -24， 0， 999999999999999 integer（No size limit)
- 1.23， 1.， 3.14e-10， 4E210， 4.0e+210 float number
- 0o177， 0x9ff， 0b101010 Two octal, hexadecimal and
- binary literals in Python 3.X
- 0177,0o177， 0x9ff， 0b101010 Two octal, hexadecimal and binary literals in Python 2.X
- 3+4j， 3.0+4.0j， 3j complex number
- set('spam')， {1， 2， 3， 4} Sets: Constructions in 2. X and 3. X
- Decimal('1.0'),Fraction(1， 3） Decimal and fraction extension types
- bool(X)， True, False Boolean types and literals

Python Integer

- A string written as a decimal number
- Hexadecimal, octal, binary in the code corresponds to integer objects, but different syntax representations of specific values
- build-in functions hex(*I*)， oct(*I*) and bin(*I*) conver a integer to a string of hex/oct/binary
- int(*str*, *base*) convert a string to a integer base on giving base

Python Float

- Floating point object in the expression will use float number (not integer)。
- Floating-point numbers are implemented in standard Cpython using the "double precision" of C language，Its precision is the same as the double precision given by the C compiler used to construct the Python interpreter.

Python Expression Operators

operators	
x if y else z	Ternary selection expression
x or y	local or
x and y	logic and
not x	logic not
x in y,x not in y	member relationship
x is y,x is not y	Object identity test
x<y,x<=y, x 〉 y, x >= y	Size comparison
x == y,x != y	Value equivalence operator

Python Expression Operators

operators	
x l y	bitwise or
x^y	Bitwise XOR
x&y	bitwise and
x<<y, X >>y	Move x left or right by y bit

Python Expression Operators

operator	meaning	priority	Associativity
+	addition	These operators have the same priority, but lower priority than the following operators	Left commissure
-	subtraction		
*	multiplication	These operators have the same priority, but higher priority than the above operators	
/	division		
//	Divide by integer		
**	exponentiation		
%	Modular		

Python Operators

- X / Y execute true division（keep the fractional part of the quotient）
- X // Y execute round-down division（remove fractional part of the quotient）

Python Operators

example:

```
>>> 6 / 2.5
2.4
>>> 6 // 2.5
2.0
```

Python Operators

- Comparison operators can be used in chain， for example:
- X < Y < Z Equivalent to X < Y and Y < X
- In Python3.X， Comparing the relative size of non-numeric mixed types is not allowed, and an exception will be thrown.

Display Format of Numerical Value

- Display of decimal places

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

0.3333333333333333
```

- using print() display

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

num =  0.3333333333333333
```

Display Format of Numerical Value

- Formatted display of decimals

```
>>> '%e' % num

'3.333333e-01'

>>> '%.2f' % num        #Display 2 decimal places

'0.33'
```

Convert to operations based on complex data types

- Adding integer and floating point numbers
  Automatically convert to operations based on complex data types

```
>>> 40 + 3.14

43.14
```

Convert to operations based on complex data types

Call built-in function to cast type Python - generally not required

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

- Automatic conversion is limited to numeric types.
- Adding strings and integers will produce errors unless you manually convert the type.

Very large integer（Unlimited precision long integer）

- 2^{100}

```
>>> 2 ** 100

1267650600228229401496703205376
```

- $2^{1000000}$

```
>>> 2 ** 1000000   #Wait, are you sure you want to output this value？？

>>> len(str(2 ** 1000000))    #Let's take a look how many numbers there are.

301030
```

PI and Square Root

- Pi (圆周率)

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

3.141592653589793
```

- Square root

```
>>> import math
>>> math.sqrt(2)          # Equivalent to 2**0.5

1.4142135623730951
```

Differences in division between Python 3 and Python 2

- Python3. X
- Python2. X

Python3. X	Python2. X

C:\Python33\python
>>> 10 / 4
2.5
>>> 10 / 4.0
2.5
 	
>>> 10 // 4 **#Round-down division**
2
>>> 10 // 4.0
2.0
	C:\Python27\python
	>>> 10 / 4
	2
	>>> 10 / 4.0
	2.5
	
	>>> 10 // 4 **#Round-down division**
	2
	>>> 10 // 4.0
	2.0

Differences in division between Python 3 and Python 2

- In Python 2.X, “/” like integer division in C language
- In Python3.X change to true division, i.e. floating point number

Differences in division between Python3 and Python2

- Aftermath： In Python3.X, Non-truncated behavior may affect a large number of Python 2.X programs.

- Solution： If your program depends on truncating integer division, use // operation in Python both 2.X and 3.X

Strange Calculation Result

- What's wrong with addition？？

```
>>> 1.1 + 2.2

3.3000000000000003
```

- Rounding error is the basic problem of numerical programming， not only in Python
- In Python, decimal numbers (fixed precision floating point numbers) and fractions are used to deal with this problem

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

Decimal('3.2')
```

Python's Variables

- The variables is created at the first assignment
- When variables are used in an expression, they are replaced with their values
- Variables must be assigned before being used in an expression
- Variables refer to objects and never need to be declared beforehand

Contents

1.2 Python Example1- Significant bit loss

- Numerical error analysis1 - significant bit loss

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

When goes wrong?

```
# program 1.1 using module math to take a square root
import math

x = float(input("input the number： "))     #input

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

1.2 Python Example 1 $\sqrt{x+1}-\sqrt{x}$

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

print("general calculation method", result1)     #output
```

- when x = 1e15

output: 1.862645149230957e-08

- when x = 1e16

output： 0.0

The approximate value should be 5e-09！！？

1.2 Python Example 1

- Solution： avoid subtraction of numbers with nearly identical values

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

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

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

print("method of transformation formula", result2)
#output
```

- When x = 1e16， output: 5e-09

1.3 Python Example 2 - Rounding error

- Example analysis of numerical error2 – rounding error
- Add 0.1 million times

```
# program 1.2
x = 0.0

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

print("sum result = ", x)    #output
```

output: 100000.00000133288 Why not 100000?

1.3 Python Example 2 - Rounding error

- Rounding error
  - If numeric values are stored in a computer with binary significant digits, the real number has rounding error inevitable

1.3 Python Example 2 - Rounding error

- Rounding error

  Decimal 0.1 converts to binary, it becomes infinite loop decimals.

  - $(0.1)_{10} = (0.0001100110011\ldots)_2$
  - It is rounded， and slightly larger than 0.1.

- In the calculation process, algorithms can generate the rounding error above should not be used.

1.3 Python Example 3 – mantissa loss

- Example analysis 3 – mantissa loss

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

```
# program 1.3
x = 1e10
y = 1e-8

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

print(x)     #output 10000000000.0
```

Why not10000000000.1 ?

1.3 Python Example 3 – mantissa loss

- Example analysis 3 – mantissa loss

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

1.3 Python Example3 – mantissa loss

- Example analysis 3 – mantissa loss

```
# Add the small values first, and then add them to the large ones
x = 1e10
y = 1e-8

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

It is important to think method before coding

output is 10000000000.1

1.4 Python Example4 – module **to r**esolve errors

- Decimal module – Efficient management of binary floating-point numbers
- Add 0.1 one million times

```
# program 1.4
from decimal import *

x = Decimal("0.0")

for i in range (1000000):
    x = x + Decimal("0.1")       #decimal 's 0.1

print("sum result = ", x)     #output
```

result is 100000.0
Not affected by rounding error

1.5 Python Example5–module fractions

- Fractions module– Direct fractions calculation

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

```
# program 1.5
from fractions import Fraction


                                        #out put 5/10 and 3/15
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))
```

Contents

- Theory Teaching：Python Core Data Type
- Practice1：coding and runing programs 1.1-1.5
- Practice2：Practice to reduce the number of operations
- Practice3：Bisection algorithm for nonlinear equations

2 Practice 2： Reduce the Mumber of Operations

- **Purpose：**

Compare the polynomial operation numbers and times among different algorithms.

```
from time import *                #import time lib
startT = time()                   #record starting time
# your code is here 。。。
endT = time()                     #record ending time
print("time = %.2g secs\n" % (endT - startT))
```

```
countMul = 0          #Statistical multiplication times
countAdd = 0          #Statistical addition times

print("multiplication times",countMul)
print("addition times",countAdd)
```

2 Practice 2：Reduce the Mumber of Operations

Example：Calculate function value (x=0.1，1，2)

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

- **Algorithm1：**

direct method

```
#program 1.6
x = 1        #variable x
f = 1        #the value of  f

for i in range (100000): # i start from 0
  f = f + [your code here]        #function

print("result = ", f)        #output
```

2 Practice 2：Reduce the Mumber of Operations

Example：Calculate function value (x=0.1，1，2)

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

Fill in the following form in your report

2 Practice 2：Reduce the Mumber of Operations

x	algorithm	result of f	Multiplications	Additions	Time(secs)
0.1	algorithm1				
	algorithm2				
1	algorithm1				
	algorithm2				
2	algorithm1				
	algorithm2				

2 Practice 2：Reduce the Mumber of Operations

Example：Calculate function value (x=0.1，1，2)

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

- **Algorithm2（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 Practice 2：Reduce the Mumber of Operations

- range() in for-loop

```
for i in range (10):

#here range (10) as range(0, 10, 1)，Generate a
sequence from 0 to (10-1) in steps of 1, using
list(range(0, 10)) can output:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
for i in range (4,0,-1):

#Generate a sequence of numbers starting from 4 and
ending at 0 - (- 1) in steps of - 1
using list(range(4,0,-1)) can output:
[4, 3, 2, 1]
```

2 Practice 2：Reduce the Mumber of Operations

- Algorithm 2（秦九韶法）：

#program 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  Your codes   # i从powN开始到1
    S =  Your codes          #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))
```

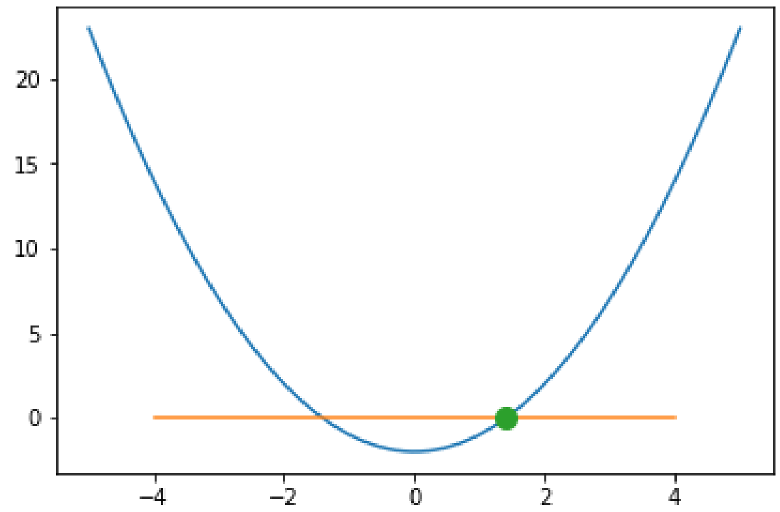
Contents

- Theory Teaching：Python Core Data Type
- Practice1：coding and runing programs 1.1-1.5
- Practice2：Practice to reduce the number of operations
- Practice3：Bisection algorithm for nonlinear equations

- **Example**：bisection method applied to $f(x)$ on [1.3, 1.5]

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

- step1：draw the graph of $f(x)$



Drawing:

- call drawing lib： matplotlib
- call numerical computing lib： NumPy

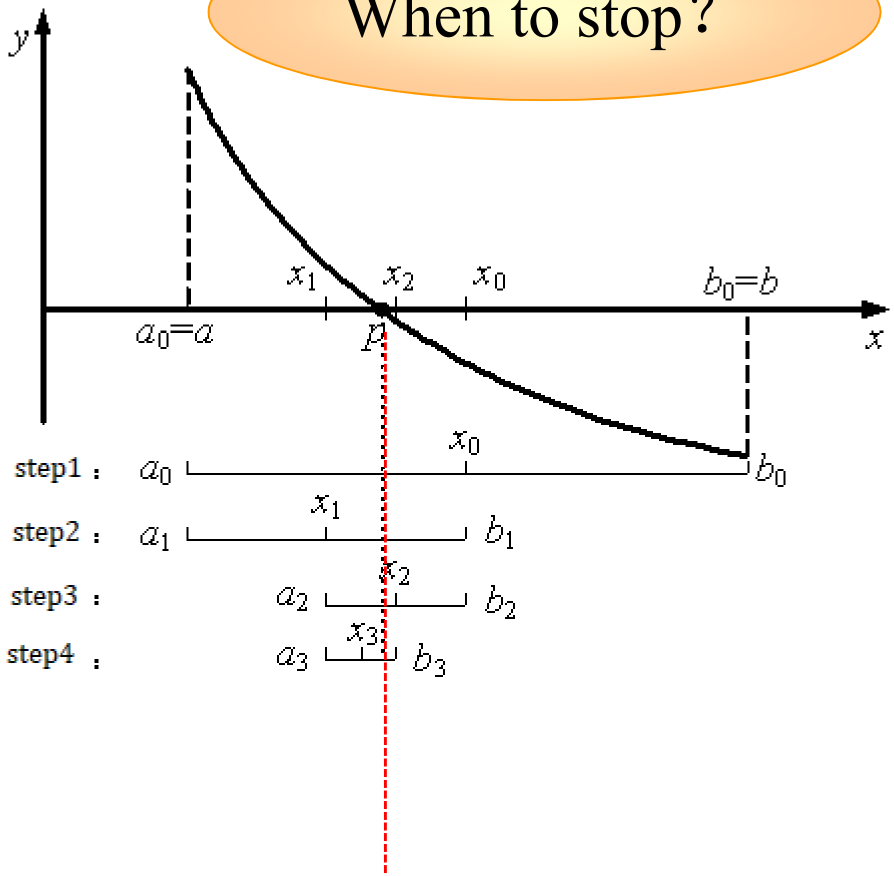
#program 1.8

```
#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)
```

Bisection method

- Assume $f \in C[a, b]$, with $f(a) \cdot f(b) < 0$, then it follows that there exists a root $\alpha \epsilon (a, b)$.

When to stop?



Step1, let $[a_0, b_0] = [a, b]$, $x_0=(a_0+b_0)/2$, if $f(x_0)=0$, then $p=x_0$ is the root, stops. Otherwise, if $f(a_0)f(x_0) < 0$, let $a_1=a_0$, $b_1=x_0$; if $f(x_0)f(b_0) < 0$, let $a_1=x_0$, $b_1=b_0$, the interval becomes $[a_1, b_1]$

Step2, midpoint of $[a_1, b_1]$ is $x_1=(a_1+b_1)/2$, if $f(x_1)=0$, then $p=x_1$ is the root, stops. Otherwise, if $f(a_1)f(x_1) < 0$, let $a_2=a_1$, $b_2=x_1$; if $f(x_1)f(b_1) < 0$, let $a_2=x_1$, $b_2=b_1$, the interval becomes $[a_2, b_2]$

Repeat above steps, the interval will be scaled down until the root satisfied the accuracy (left figure)

Program of bisection algorithm

#program 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
     Fill in your code          #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 Practice 3： Bisection algorithm for nonlinear equations

- **Fill out the following form in your report**

Iterations	Lower limit x_{low}	upper limit x_{up}	$(x_{up} - x_{low})/2$	Sign of $f((x_{up} - x_{low})/2)$
0	1.3	1.5	1.4	< 0

Practise:

- Perform program1.1-1.9
- Complete document “Practice 1.docx”,

  submit it to CG in pdf format

Appendix： keys to practice 2

Example： calculate the value of the function （x=0.1， 1， 10）

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

computer configuration

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

4.00 GB

64 位操作系统

Appendix：keys to practice 2

Keys

x	Algorithm	Value of f	Multiplications	additions	time(secs)
0.1	Algorithm 1	1.234567901234568	5000050000	100000	0.062
	Algorithm 2	1.234567901234568	100000	100000	0.034
1	Algorithm 1	5000150001	5000050000	100000	0.084
	Algorithm 2	5000150001	100000	100000	0.034
2	Algorithm 1	长度30109	5000050000	100000	13
	Algorithm 2	长度30109	100000	100000	0.36

Appendix： keys to practice 2

Source Code 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))
```

Appendix： keys to practice 2

Source code 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))
```