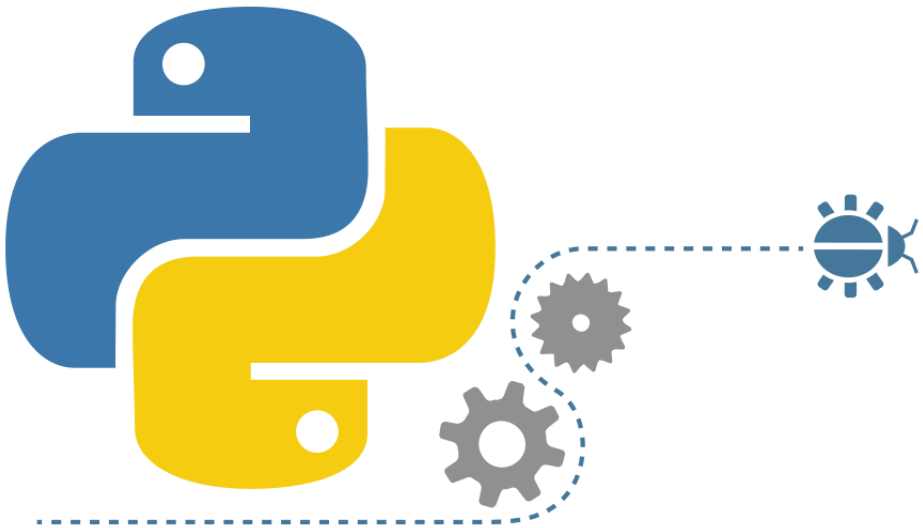
Python交互式编程——实验报告

实验名称： 用Python做计算器，做数学题

姓名： 张展邦

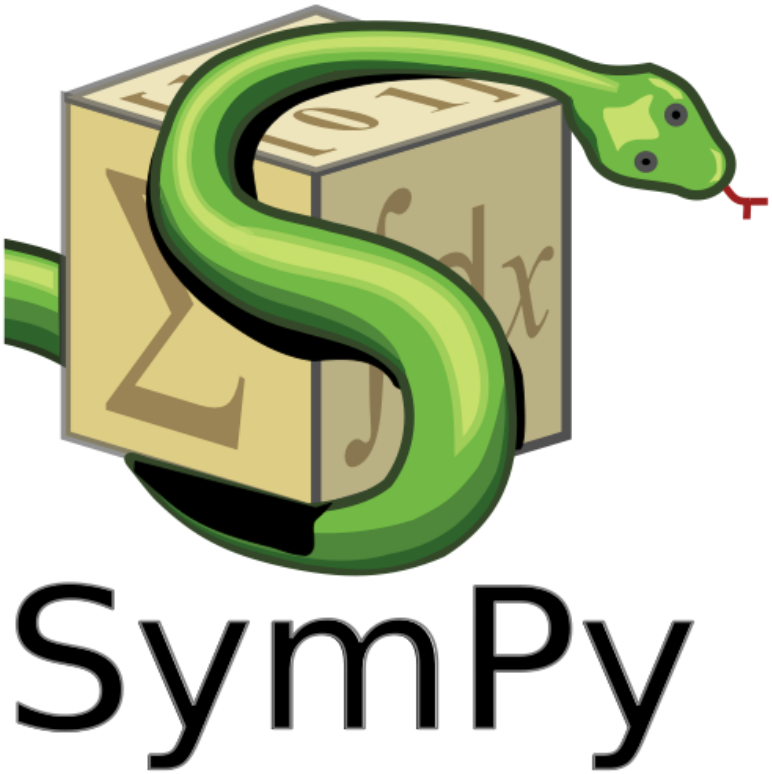
学号：18342134



一、实验目的

1. 了解一种“解释型”语言 python
2. 使用 python 做一些简单的科学计算

二、实验环境



- 编程工具：Python（Anaconda）
- Sympy强力支持
- 操作系统：Windows 10

三、使用 python 作为计算器

3.1 使用简单表达式

- 基本的四则运算都可以直接计算出
- 另：**变量万能**

```
>>> sum = 3-1
>>> print(sum)
2
>>> 1+1
2
>>> sum = 2+3
>>> print(sum)
```

```
5
>>> sum *= 2
>>> print(sum)
10
```

- 除法分两种，用'/'，保留小数部分；用'//'，舍弃小数部分（向下取整）

```
>>> 1/3
0.3333333333333333
>>> 5//3
1
```

- 幂运算，用符号'**'，以下是 $ 3^2 $的计算例子

```
>>> 3**2
9
```

3.2 使用数学函数

Python里用专门用于数学计算的库math

- 导入math

```
>>> import math
```

- math有什么，用help函数

```
>>> help(math)
```

常用常数

- e = 2.71828828459045
- pi = 3.141592653589793

常用函数

- math.factorial(x) = $ x! $
- math.exp(x) = $ e^x $
- math.pow(x,y) = $ x^y $ //开根号时y为分数
- math.gcd(x,y) = $ (x,y) $ //x,y的最大公因数
- math.log(x, [base=math.e]) = $ log _(base) x $ //不打系统默认base = math.e
- 三角函数类的使用与实际无异，只需前面加'math.'

四、使用Python做高数题目

Python里有SymPy这个无敌函数库，运用它可以轻松解决高数和线代作业（非证明题）

SymPy的计算本质是**符号计算**，不是数值计算

4.1高等数学

- **SymPy的符号开方：root(x,2) = sqrt(x)**

```
>>> f = root(x, 2)
```

- **首先从SymPy里导入所有的函数，设置符号**

```
>>> from __future__ import division
>>> from sympy import *
>>> x, y, z, t = symbols('x y z t')
>>> a, b, c, n = symbols('a b c n')
>>> f, g, h = map(Function, 'fgh')
```

- **表达式表示**

```
>>> f = 1/x + (x*sin(x) - 1)/x
```

- **表达式化简：simplify**

```
>>> f.simplify()
>>> print(f)
sin(x)
```

- **求极限：limit(f, x, 极限)**（无穷：oo）

$$\lim_{n\to\infty} \sqrt{n+1} - \sqrt{n}$$

```
>>> f = root(n+1, 2)-root(n, 2)
>>> limit(f, n, oo)
0
```

$$\lim_{x\to 0} \frac{\sqrt{1+x} - 1}{x + \sin x}$$

```
>>> f = (root(1+x, 2)-1)/(x+sin(x))
>>> limit(f, x, 0)
1/4
```

$$\lim_{x\to 0} \frac{\sqrt[3]{1+3x} - \sqrt[3]{1-2x}}{x + x^2}$$

```
>>> f = (root(1+3x, 3)-root(1-2x, 3))/(x+x**2)
>>> limit(f, x, 0)
5/3
```

4.2线性代数

- 矩阵的表达

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
>>> N = Matrix([[1,0,0],[0,1,0],[1,0,1]])
>>> N
Matrix([ [1, 0, 0], [0, 1, 0], [1, 0, 1]])
```

- 矩阵的相乘

$$\begin{bmatrix} 2 & 3 \\ 1 & -5 \end{bmatrix} \begin{bmatrix} 4 & 3 & 6 \\ 1 & -2 & 3 \end{bmatrix} = \begin{bmatrix} 11 & 0 & 21 \\ -1 & 13 & -9 \end{bmatrix}$$

```
>>> A = Matrix([[2,3],[1,-5]])
>>> B = Matrix([[4,3,6],[1,-2,3]])
>>> A*B
Matrix([ [11, 0, 21], [-1, 13, -9]])
```

- 矩阵求逆

$$\begin{bmatrix} 2 & 5 \\ -3 & -7 \end{bmatrix}^{-1} = \begin{bmatrix} -7 & -5 \\ 3 & 2 \end{bmatrix}$$

```
>>> A = Matrix([[2,5],[-3,-7]])
>>> A**-1
Matrix([ [-7, -5], [ 3, 2]])
```

- 矩阵的转置

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}^{T} = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

```
>>> A = Matrix([[1,2,3],[4,5,6],[7,8,9]])
>>> A.T
Matrix([ [1, 4, 7], [2, 5, 8], [3, 6, 9]])
```

- 矩阵的行列式

$$\begin{vmatrix} 1 & 6 & 0 \\ 2 & 4 & -1 \\ 0 & -2 & 0 \end{vmatrix} = -2$$

```
>>> A = Matrix([[1,6,0],[2,4,-1],[0,-2,0]])
>>> A.det()
-2
```

- 简化行阶梯形：rref()返回第一项为简化阶梯型矩阵，第二项为主元位置列表（以0开始）

$$\begin{bmatrix} 1 & 3 & 5 & 7 \\ 3 & 5 & 7 & 9 \\ 5 & 7 & 9 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
>>> A = Matrix([[1,3,5,7],[3,5,7,9],[5,7,9,1]])
>>> A.rref()
(Matrix([ [1, 0, -1, 0], [0, 1, 2, 0], [0, 0, 0, 1]]), (0, 1, 3))
```