@[toc]

作业基本信息

这个作业属于哪个课程	**< 2301-计算机学院-软件工程>**
这个作业要求在哪里	< 软工实践第一次作业>
这个作业的目标	< 实现一个科学计算器具备的基本功能，并且UI设计较美观 >
其他参考文献	< Python_GUI Math库 python2exe>

解题思路描述

> 1. 拿到这个题目本来是准备做一个微信小程序实现计算器，后来了解到不建议使用小程序，又考虑到从头学前端的时间问题，经过在csdn查阅相关资料后，准备使用`Python`的`GUI-tkinter`库进行实现。
> 2. 初步想法是先进行一个简单计算器的设计，比如加减乘除法，实现一些基本功能，然后在此基础上进行功能的丰富，实现科学计算等功能。
> 3. 又想到自己在使用计算器的时候，对历史记录的查询十分看重，因此想开发一个具有历史记录查询功能的计算器方便用户对自己过去输入过的计算式进行查询。
> 4. 一个计算器，一个软件是否受人欢迎，除了它的功能较为完善以外，其应该有着以一个较为美观的外表，因此UI设计也是相当重要。(实则还有很大进步空间)

Gitcode项目地址

Python实现科学计算器并测试

PSP表格

PSP	**Personal Software Process Stages**	**预估耗时（分钟）**	**实际耗时（分钟）**
Planning	计划	20	30
• Estimate	• 估计这个任务需要多少时间	20	30
Development	开发	490	725
• Analysis	• 需求分析 (包括学习新技术)	120	100
• Design Spec	• 生成设计文档	30	20
• Design Review	• 设计复审	20	20
• Coding Standard	• 代码规范 (为目前的开发制定合适的规范)	15	20
• Design	• 具体设计	30	35
• Coding	• 具体编码	180	260

PSP	Personal Software Process Stages	预估耗时（分钟）	实际耗时（分钟）
• Code Review	• 代码复审	20	30
• Test	• 测试（自我测试，修改代码，提交修改）	120	240
Reporting	报告	190	230
• Test Repor	• 测试报告	120	130
• Size Measurement	• 计算工作量	10	10
• Postmortem & Process Improvement Plan	• 事后总结, 并提出过程改进计划	60	90
	合计	700	985

接口设计和实现过程

> **本次设计科学计算器共用到两个类 `class BTN(Button)` 和 `class Calculator`**

- **类1**： `class BTN(Button)`
  - **含义**：自定义 **按钮类** 从`tkinter.Button`中**继承**而来
  - **功能**：给按钮设置统一的风格，减少冗余代码
  - **与 类2 的关系**：`Calculator`类用 `BTN`类 自定义按钮
  - **代码展示**：（仅展示了定义计算器第一行的按钮制作）

```
class BTN(Button):
    # 自定义按钮类，设置显示文本你，显示为位置，以及对应函数
    def __init__(self,frame,text,fun, args,bg='white',fg='black'):
        super(BTN,self).__init__()
        self.x,self.y,self.w,self.h = args[0],args[1],args[2],args[3]
        self.btn =
Button(frame,text=text,bg=bg,fg=fg,command=fun).place(x=self.x, y=self.y,
width=self.w, height=self.h)

#  第一行按钮制作
    self.button_cube_root = BTN(self.master, 'X^1/3', fun=lambda:
self.getNum('**(1/3)'),args=('10', '150', '60', '40')) # x^(1/3)
    self.button_XY = BTN(self.master, 'X^y', fun=lambda:
self.getNum('**'),args=('90', '150', '60', '40')) #todo 次方项如何开发  显示 x^
    self.button_left = BTN(self.master, '(', fun=lambda:
self.getNum('('),args=('170','150','60','40'))
    self.button_right = BTN(self.master, ')', fun=lambda:
self.getNum(')'),args=('250', '150', '60', '40'))
    self.button_c = BTN(self.master, 'C',  fun=self.clear,args=('330',
'150','60','40'),fg='red')
    self.button_back = BTN(self.master, 'Back', fun=self.back, args=('410',
'150', '60', '40'))
    self.button_division = BTN(self.master, text='÷',  fun=lambda:
self.getNum('÷'),args=('490', '150', '60', '40'),fg='red')
```

- **类 2**： `class Calculator`
  - **含义**：计算器类，用于计算器初始化
  - **功能**：有10多个功能属性，30+ 个按钮属性，5个计算函数（`back`,`get_history`,`getNum`,`clear`,`run`）
  - **关键所在**：
    - 设置两个不同的计算式，**分别存储用于显示给用户看的表达式 和实际计算的表达式**，每次输入计算符号或者数字的时候，两个式子都进行存储，并且**注意不是完全相同的存储方式**（如次方符号 显示表达式存`^`，而计算表达式存`**`）

```
def getNum(self, num):
    if self.flag:    # 用于处理输入=号计算后再输入新的表达式自动情况的功能
        self.screen_equation.set('')
        self.equation.set('0')
        self.result.set('')
        self.flag = 0
    temp_equ = self.equation.get()  # 输入算式
    temp_screen_equation = self.screen_equation.get()
    if temp_equ == '0' and (num not in ['.', '+', '-', '*', '÷']):  #处理
算式初始的特殊情况
        temp_equ = ''
        temp_screen_equation = ''
    temp_screen_equation = temp_screen_equation + num
    temp_equ = temp_equ + num
    self.equation.set(temp_equ)

 self.screen_equation.set(temp_screen_equation.replace('**','^').replace
('*','x').replace('pi','π'))  # 替换一部分字符，让显示更符号用户渴望看到的
```

    - **计算**：计算表达式的**结果使用Python的eval函数进行计算**，但是要注意表达式的正确与否，并**给出错误原因**，而且考虑到实际需求，计算只保留了5位小数。

```
def run(self):
    temp_equ = self.equation.get()
    self.equ_history.append(temp_equ)
    temp_screen = self.screen_equation.get()
    self.screen_history.append(temp_screen)
    try:
        answer = '%.5f' % eval(temp_equ)  #  保留两位小数
        self.result.set(str(answer))
    except (ZeroDivisionError):  #  除0错误，语法错误返回Error
        self.result.set(str('Error: 除零错误'))
    except NameError:
        self.result.set(str('Error: 请加上括号'))
    except SyntaxError:
        self.result.set(str('Error: 语法错误，请正确输入'))
    self.res_history.append(self.result.get())
```

    - **查询历史记录：** 当用户每按一次CE的按钮，**都会返回前一次计算的表达式和结果。**此功能用三个历史列表分别存储历史记录。

```
def get_history(self):    # 查询历史记录和结果
    temp_equ = self.equation.get()
    pre_idx = self.equ_history.index(temp_equ)
    new_idx = (pre_idx - 1 + len(self.equ_history)) %
len(self.equ_history) # 取出上一个计算式子和结果
    self.equation.set(self.equ_history[new_idx])
    self.result.set(self.res_history[new_idx])
    self.screen_equation.set(self.screen_history[new_idx]
```

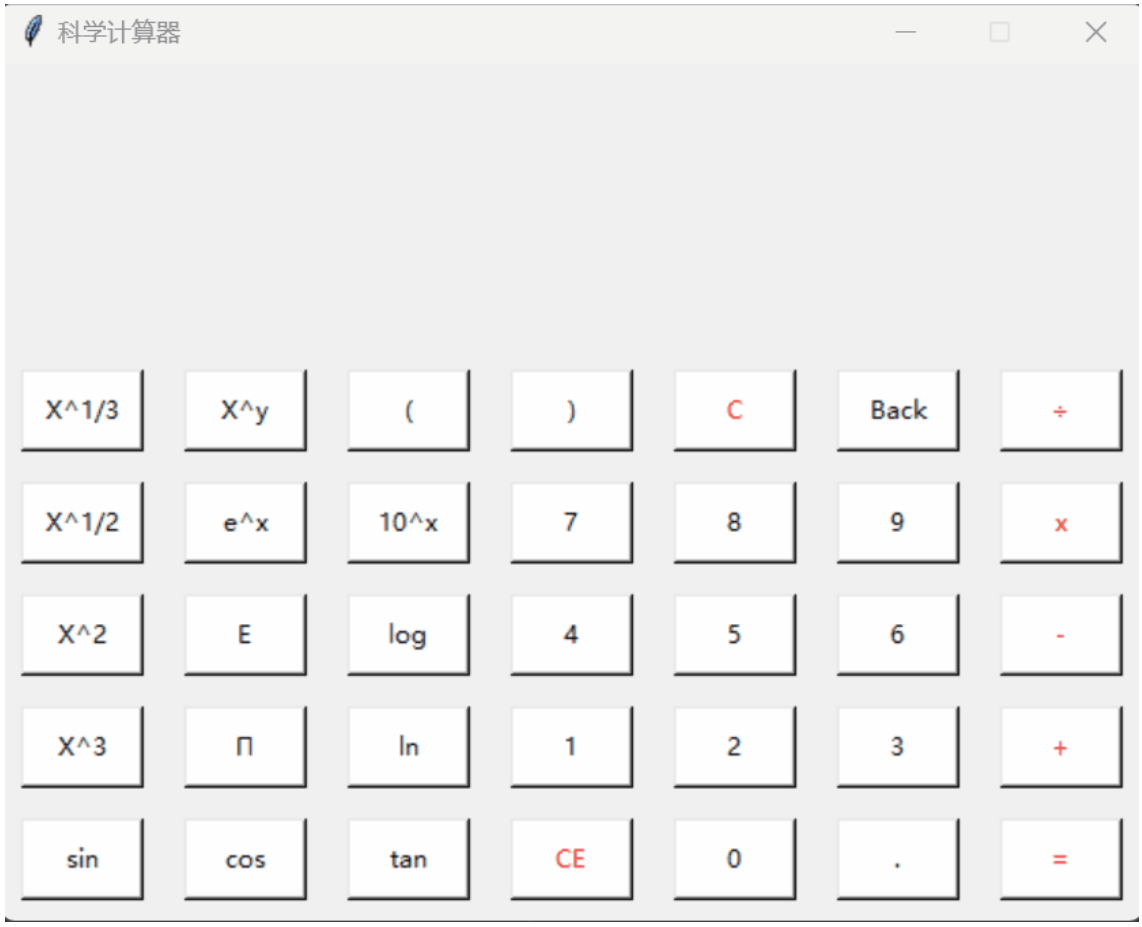
- **撤销一位数字or算符**：需要同时对显示表达式和计算表达式同时更新，并且显示表达式每次不是只撤回一个字符。因为(`sin`,`cos`,`tan`,`log`**等多个字符表示的算符需要一次撤销几个字符**，需要特殊处理)

```
def back(self):
    self.flag = 0   # 设置初始开始标志
    temp_equ = self.equation.get()
    temp_screen = self.screen_equation.get()
    self.screen_equation.set(temp_screen[:-1])
    '''
    elif temp_equ[-3:] == 'log':  # 特殊处理一些复杂算符
        self.equation.set(temp_equ[:-3])
        self.screen_equation.set(temp_screen[:-3])
    elif temp_equ[-3:] == 'sin':
        self.equation.set(temp_equ[:-3])
        self.screen_equation.set(temp_screen[:-3])
    '''
```

- `Python2exe`：可以通过下载`Pyinstall`包，在命令对给定的python文件进行转换。具体操作可看 python2exe

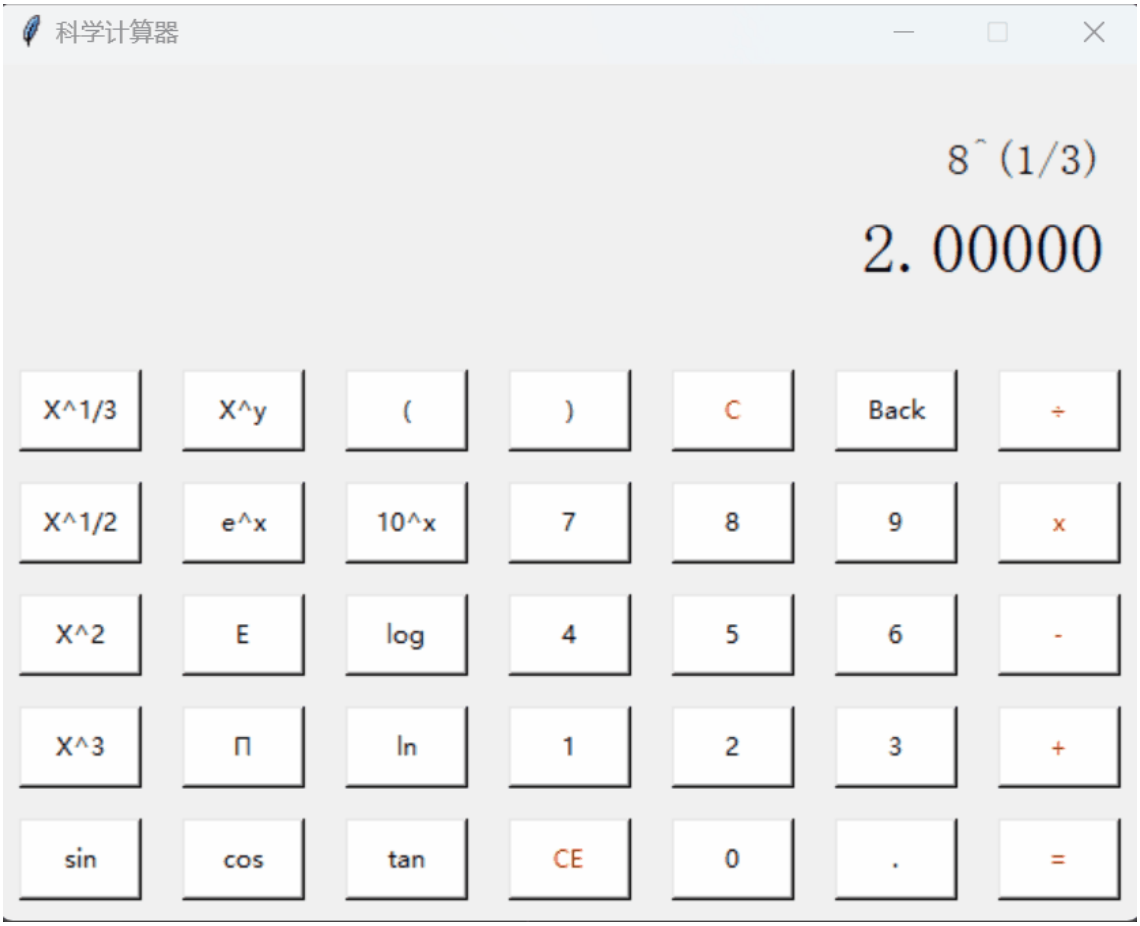
关键功能展示

- **加减乘除运算：**



- **科学计算**

- **展示历史记录**



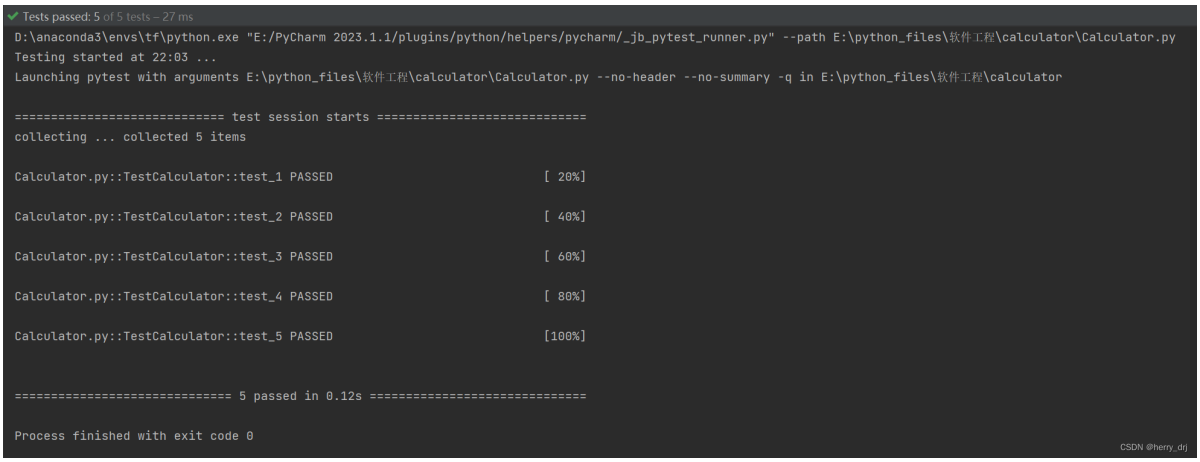
性能改进与优化

- 原本在Calculator中每一个按钮都要定义一个实例，然后再进行一次位置放置，考虑到按钮个数较多，并且一般颜色较为统一，因此**考虑自定义一个新类**，对按钮颜色设计等进行统一设计，，同时也减少了相似代码的冗余。
- 考虑到用户期望看到的算数表达式一般是 18×17　$18 \div 6$ **而不是** $18 * 17$　$18/6$ 这种，因此考虑将**用于显示和用于计算的字符串分开存储**，这样有利于用户观察计算式。
- 考虑到用户需要的查询过去某个时刻计算的表达式，因此我新增了**查询历史记录**这个功能。
- 使用测试发现，每次进行新的计算式的时候都需要clear,因此我重新优化了，使得每次计算完结果的时候**每次输入新的表达式的时候，自动清零开始。**
- back表达式要一个一个删除，但是**对特殊字符需要考虑连续删除多个字符**的。

单元测试与性能分析

单元测试

> 通过定义了一个测试函数类，和五个测试函数，进行计算器的基础算法测试包括（+ - x ÷ ）、科学计算（log、ln、x^y、sin、cos、tan等等）



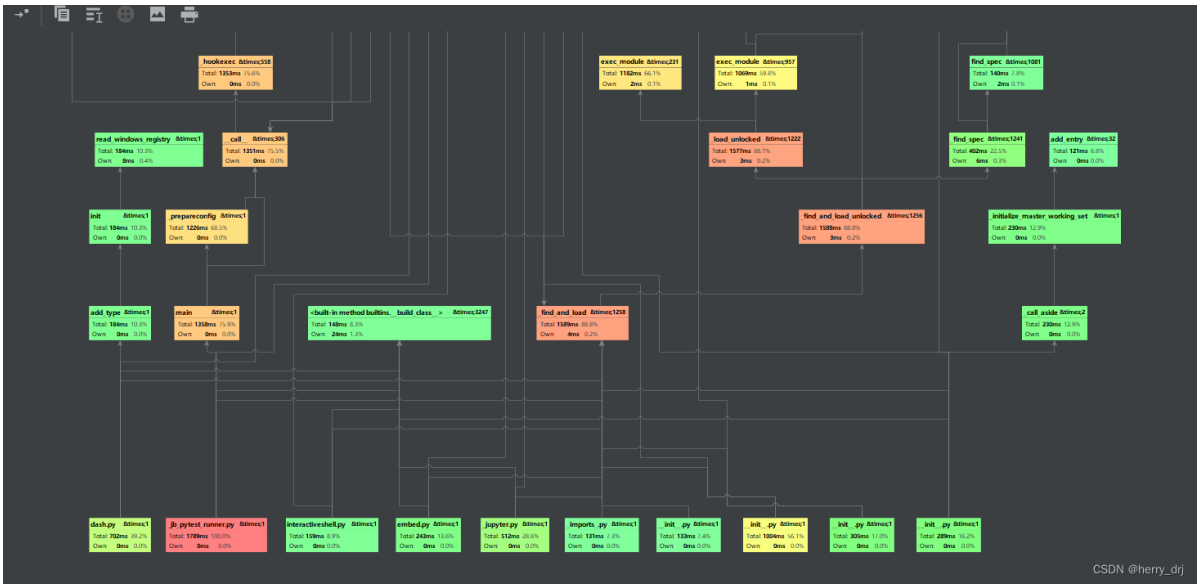
单元测试结果图

部分代码：

```
# 测试部分代码 完整代码见github链接
class TestCalculator(unittest.TestCase):
     def test_4(self):
         Cal = Calculator(root)
         Cal.equation.set('sin(pi/2)+cos(pi/2)+tan(pi/4)')
         result = Cal.run()
         self.assertEqual(result, '2.0')
     def test_5(self):
         Cal = Calculator(root)
         Cal.equation.set('(log10(10**5)-1)/2')
         result = Cal.run()
         self.assertEqual(result, '2.0')
if __name__ == '__main__':
     unittest.main()
```

性能分析

> python的性能分析可以 通过分析**函数执行的频繁程度、哪些函数经常阻塞、每个函数的使用时间**以找到性能较低的函数进行优化。（~~斯，但是计算器这个好像也没啥特别需要优化性能的~~）



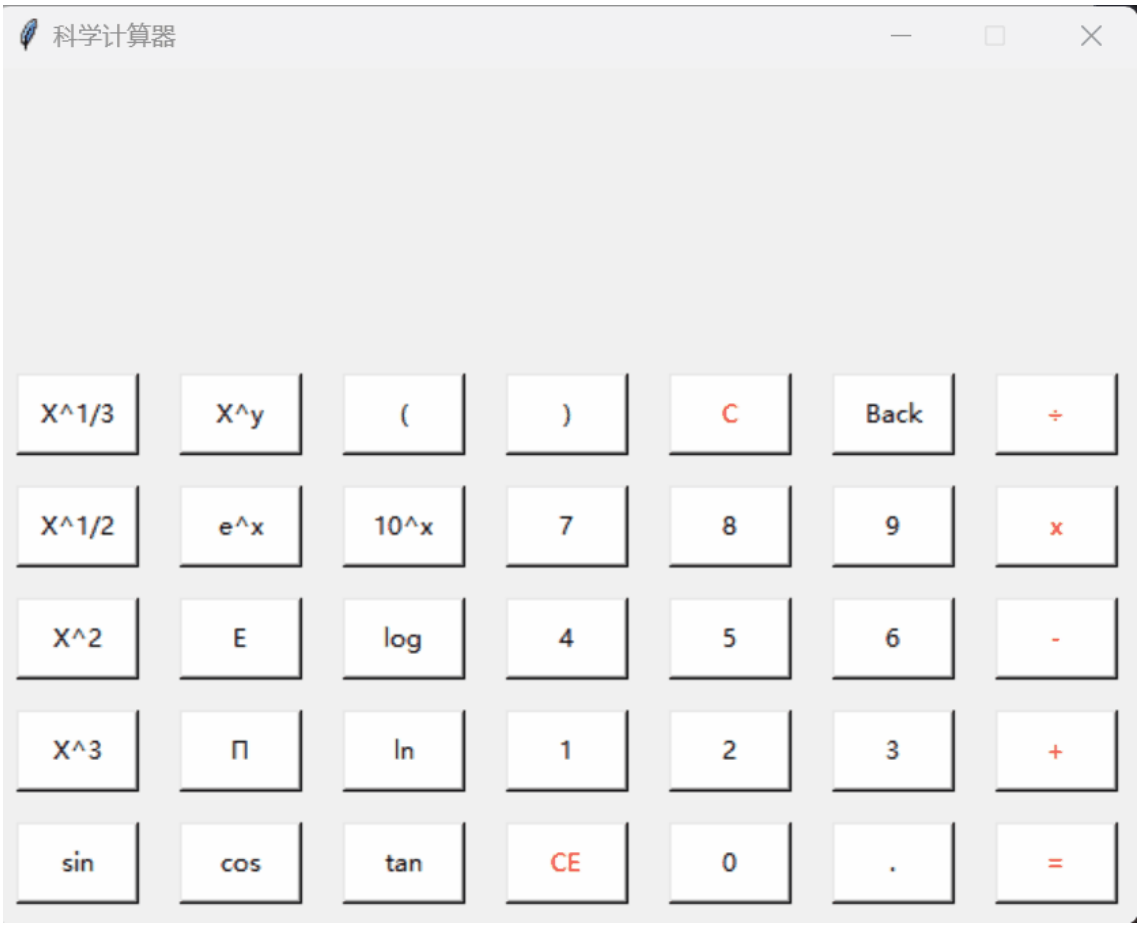
性能分析图

异常处理措施与展示

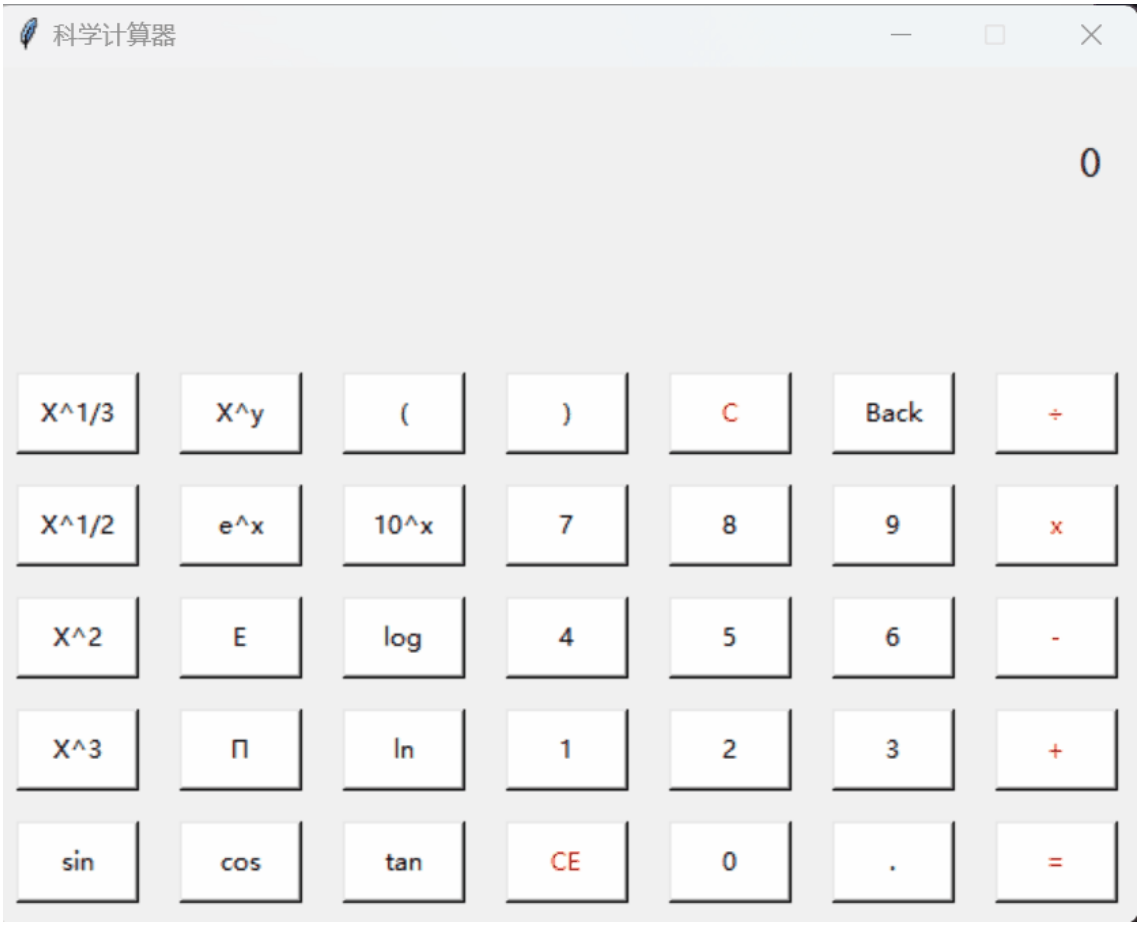
> 在使用计算器的时候,难免会有**输入错误表达式**的时候,因此在这种时候需要及时报错提醒用户，最好有具体的原因!

计算机错误输入分为以下几种:

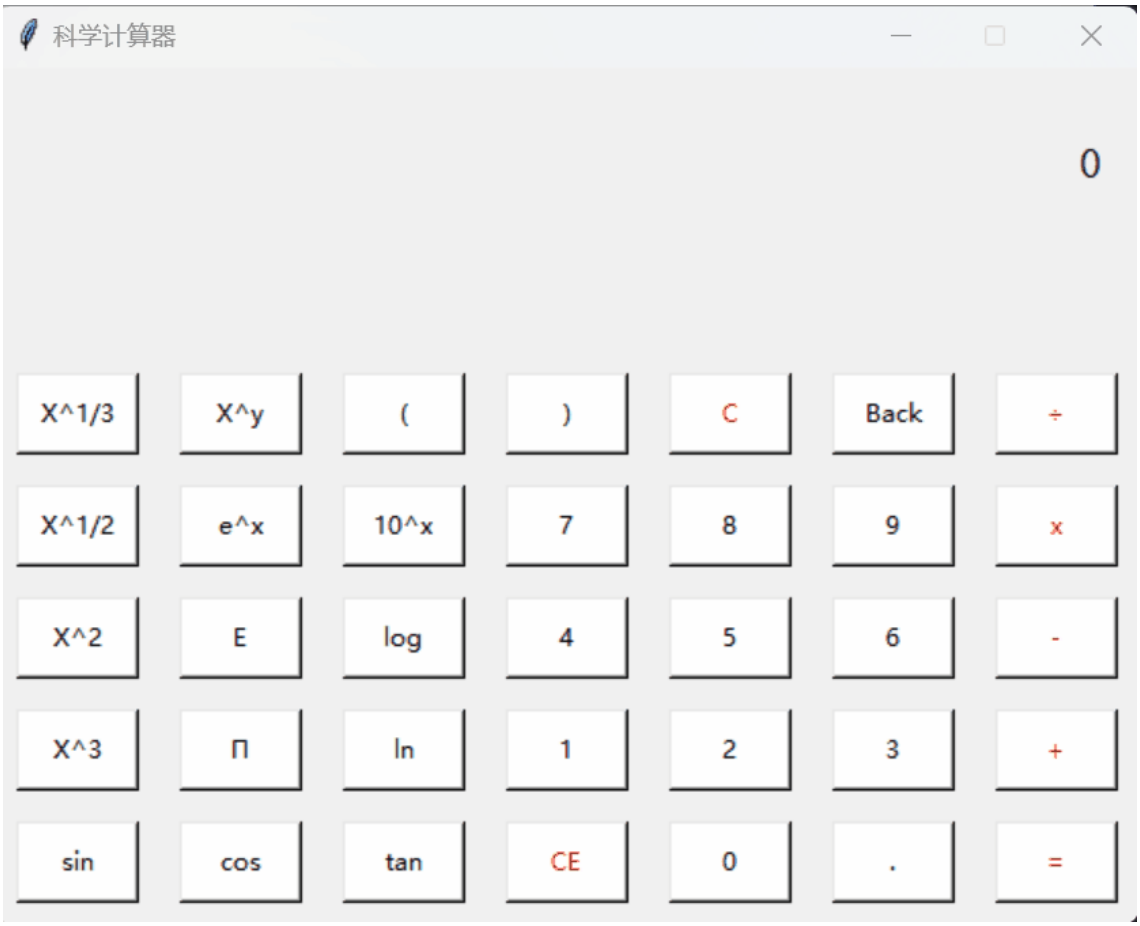
- **除数分母为零的**错误，需要返回错误并给出错误原因;



- 使用科学计算e，log的时候**未加括号**，不能计算，会导致括号错误；



- 输入次方等符号，需要前面先有一个数字，否则会导致**语法错误**；



代码：

```
try:
    answer = '%.5f' % eval(temp_equ)  #  保留两位小数
    self.result.set(str(answer))
except (ZeroDivisionError):  #  其他除0错误，或语法错误返回Error
    self.result.set(str('Error: 除零错误'))
except NameError:
    self.result.set(str('Error: 请加上括号'))
except SyntaxError:
    self.result.set(str('Error: 语法错误，请正确输入'))
```

心得体会

- 本次自己动手编程实现了一个**科学计算器**,体会到 **一个项目从无到有一个完整的过程**，包括: 开发前的计划,开发时候的需求分析,生成设计文档,实现需求的具体编码,以及开发完成后的单元测试等.
- 在项目中发现在**编写代码实现具体功能**的时候,往往没有计划好的那种顺利,一般是错误频出, **新的问题不断出现,不断解决的一个过程,** 比如(我在实现过程中发现,用于计算的表达式和用于存储的表达式不能一起存储, 查询历史记录的各种问题).
- 学会了`Python`的GUI库`tkinter`熟悉了其使用,会使用该库进行简单的**可视化软件制作.**
- 提高了编程中**debug的能力**,会用debug工具进行调试python代码.
- 学会了使用python工具进行**单元测试和异常处理**, 有利于更好地完整地完成一个项目.