Report On

Scientific Calculator

Submitted in partial fulfillment of the requirements of the Course project in Semester IV of Second Year Computer Engineering

by

Hemant Shivalkar (Roll No. 40)

Rahul Singh (Roll No. 41)

Supervisor

Prof. Sneha Mhatre

**Vidyavardhini's College of Engineering & Technology**

**Department of Computer Engineering**



**(2023-24)**

**Vidyavardhini's College of Engineering & Technology**

**Department of Computer Engineering**

**CERTIFICATE**

This is to certify that the project entitled “Scientific Calculator” is a Bonafide work of Hemant Shivalkar (Roll No. 40) Rahul Singh (Roll No. 41) submitted to the University of Mumbai in partial fulfillment of the requirement for the Course project in semester IV of Second Year Computer Engineering.

**Supervisor**

Prof. Sneha Mhatre

Internal Examiner External Examiner

Dr Megha Trivedi

Head of Department

Dr. H.V. Vankudre

Principal

**Index**

1. Abstract
2. Problem Statement
3. Module Description
4. Block Diagram

5.Code

6.Output

7.Conclusion

8.Reference

9.Code

**1.Abstract**

A scientific calculator is a digital tool designed to perform complex mathematical operations, typically beyond basic arithmetic. It includes functions like trigonometry, logarithms, exponential, and statistical calculations, providing users, especially students, engineers, and scientists, with a convenient and efficient means to solve intricate mathematical problems.

1

**2.Problem Statement**

Design a scientific calculator application that can perform basic arithmetic operations, and advanced scientific functions such as trigonometry, logarithms, exponentials, etc. Ensure the calculator can handle multi-digit numbers, decimal points, and scientific notation. The user interface should be intuitive and user-friendly.

2

**3. Module Description**

The scientific calculator module will provide advanced mathematical functionalities, including trigonometry, logarithms, exponentials, and constants. It will support multi-digit numbers, decimal points, and scientific notation. The module will follow standard mathematical conventions and provide accurate results. It will also include error handling for invalid inputs and provide a history of calculations for user reference.

**4.Block Diagram**

Start

Initialize GUI

Select Folder

Organize files based on extension

Display Success message box

End

**5. Code**

import tkinter as tk

import math as m

import numpy as np

import sympy as sym

import statistics as st

from functools import reduce

import scipy

'''

This is a scientific calculator built using python -n-built libraries- Tkinter, math and statistics- and external libraries

viz: numpy and sympy. The various buttons are the functionalities of the calculator. The major aim of this project is to turn this calculator into a mathematical software that can be used freely by research scientists, engineers and mathematicians in Africa.

'''

btn\_paremeters = {

'padx': 1,

'pady': 1,

'bd': 4,

'fg': 'white',

'bg': 'grey',

'font': ('arial', 10),

'width': 6,

'height': 2,

'relief': 'ridge',

'activebackground': "#666666"

}

btn\_paremeters\_2 = {

'padx': 1,

'pady': 1,

'bd': 4,

'fg': 'black',

'bg': 'powder blue',

'font': ('arial', 10),

'width': 6,

'height': 2,

'relief': 'ridge',

'activebackground': "#666666"

}

btn\_paremeters\_3 = {

'padx': 1,

'pady': 1,

'bd': 4,

'fg': 'black',

'bg': 'pink',

'font': ('arial', 10),

'width': 6,

'height': 2,

'relief': 'ridge',

'activebackground': "#666666"

}

global const

const = 180

pi = 22/7

def Sin(x):

return m.sin(x \* (const/180))

def Cos(x):

return m.cos(x\*const)

def Tan(x):

return m.tan(x)

def Floor(x):

return m.floor(x)

def Ceil(x):

return m.ceil(x)

def ArcSin(x):

return m.acos(x)

def ArcCos(x):

return m.acos(x)

def ArcTan(x):

return m.atan(x)

def GCD(x,y):

if x> y:

return m.gcd(x, y)

else:

return m.gcd(y,x)

def fmod(x, y):

return m.fmod(x, y)

def Factorial(x):

return m.factorial(x)

def Sqrt(x):

return m.sqrt(x)

def Log(x):

return m.log(x)

def perm(x, y):

if x>=y:

return Factorial(x)/Factorial(x-y)

else:

return 'Incorrect input!'

def comb(x, y):

if x>=y:

return Factorial(x)/(Factorial(x-y)\*Factorial(y))

else:

return 'Wrong input'

def angle\_conversion(x):

pass

class Sci\_Calculator:

def \_\_init\_\_(self, master):

self.master = master

master.title('Scientific Calculator') #name of Calculator

#self-equation to store values

self.expression = ""

self.result = ""

self.input\_txt = tk.StringVar()

self.recall = ''

self.sum\_up = ''

#Mainframe

MainFrame = tk.Frame(self.master, bg = 'gray')

MainFrame.place(relx=0.1, rely=0.1, relwidth = 0.8, relheight=0.8)

#frame for the display

top\_frame = tk.Frame(MainFrame, height = 50, width = 100, bg = 'yellow', relief = 'groove', bd = 4 )

top\_frame.pack(side = tk.TOP)

#the frame for the buttons

bottom\_frame = tk.Frame(MainFrame,height = 700, width = 100, bg= 'grey')

bottom\_frame.pack(side=tk.TOP)

#display on screen

self.screen = tk.Entry(top\_frame, width = 60, background = "grey", foreground = "white", textvariable=self.input\_txt, bd = 5, justify ='right' , cursor = 'tcross')

self.screen.pack()

#Row 3

#factorial button

self.mod = tk.Button(bottom\_frame, text = 'n!', \*\*btn\_paremeters, command = lambda:self.btn\_click('Factorial('))

self.mod.grid(row=3, column = 0)

#cube root button

self.cube\_root = tk.Button(bottom\_frame, text = '₃√', \*\*btn\_paremeters, command = lambda:self.btn\_click('\*\*(1/3)'))

self.cube\_root.grid(row=3, column = 1)

#cube button

self.cube = tk.Button(bottom\_frame, text = 'x^3', \*\*btn\_paremeters,command = lambda: self.btn\_click('\*\*3'))

self.cube.grid(row=3, column = 2)

#Author button

self.ntn = tk.Button(bottom\_frame, text = 'Author', \*\*btn\_paremeters, command = lambda:self.btn\_click('Osikoya Samuel'))

self.ntn.grid(row=3, column = 3)

#antilog button

self.pwr10 = tk.Button(bottom\_frame, text = '10^x', \*\*btn\_paremeters, command = lambda: self.btn\_click('10\*\*') )

self.pwr10.grid(row=3, column = 4)

#exponential button

self.exp = tk.Button(bottom\_frame, text = 'e^x', \*\*btn\_paremeters, command = lambda:self.btn\_click('m.exp('))

self.exp.grid(row=3, column = 5)

#Row 4

#fraction button

self.frac = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = 'x/y', command ='', )

self.frac.grid(row = 4, column = 0)

#square root button

self.rootx = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = '√x', command =lambda : self.btn\_click('Sqrt(') )

self.rootx.grid(row = 4, column = 1)

#square button

self.xsquared = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = 'x^2', command =lambda : self.btn\_click('\*\*2'), )

self.xsquared.grid(row = 4, column = 2)

#power button

self.xpwr = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = 'x^n', command =lambda : self.btn\_click('\*\*') )

self.xpwr.grid(row = 4, column = 3)

#base 10 log button

self.log = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = 'log', command =lambda : self.btn\_click('Log('), )

self.log.grid(row = 4, column = 4)

#natural log button

self.btn\_ln = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = 'ln', command =lambda : self.btn\_click('ln('))

self.btn\_ln.grid(row = 4, column = 5)

#Row 5

#x button

self.alpah\_a = tk.Button(bottom\_frame, text = 'x',\*\*btn\_paremeters, command = lambda: self.btn\_click('x'))

self.alpah\_a.grid(row=5, column = 0)

#y button

self.fact\_b = tk.Button(bottom\_frame, text = 'y', \*\*btn\_paremeters, command = lambda: self.btn\_click('y'))

self.fact\_b.grid(row=5, column = 1)

#help button

self.help\_btn = tk.Button(bottom\_frame, text = 'HELP', \*\*btn\_paremeters, command =lambda :self.btn\_click('visit www.mathsgem.wordpress.com for help'))

self.help\_btn.grid(row=5, column = 2)

#sine inverse button

self.sin\_inv\_btn = tk.Button(bottom\_frame, text = 'Sin^-1', \*\*btn\_paremeters, command = lambda: self.btn\_click('Arcsin('))

self.sin\_inv\_btn.grid(row=5, column = 3)

#cosine inverse button

self.cos\_inv\_btn = tk.Button(bottom\_frame, text = 'Cos^-1', \*\*btn\_paremeters, command = lambda: self.btn\_click('Arcos('))

self.cos\_inv\_btn.grid(row=5, column = 4)

# tan inverse button

self.tan\_inv\_btn = tk.Button(bottom\_frame, text = 'Tan^-1',\*\*btn\_paremeters, command = lambda: self.btn\_click('Arctan('))

self.tan\_inv\_btn.grid(row=5, column = 5)

#Row 6

#enclosed minus button

self.enclosed\_minus\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = '( - )', command = lambda:self.btn\_click('(-'), )

self.enclosed\_minus\_btn.grid(row = 6, column = 0)

#degree conversion button

self.angles\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = 'o \' \"', command ='' )

self.angles\_btn.grid(row = 6, column = 1)

#hyperbolic fucntion button

self.hyp\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters\_3, text = 'hyp', command = self.hyp, )

self.hyp\_btn.grid(row = 6, column = 2)

#sine button

self.Sin\_btn= tk.Button(bottom\_frame, \*\*btn\_paremeters, text = 'Sin', command = lambda: self.btn\_click('Sin('))

self.Sin\_btn.grid(row = 6, column = 3)

#cosine button

self.cos\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = 'Cos', command =lambda : self.btn\_click('cos(') )

self.cos\_btn.grid(row = 6, column = 4)

#tan button

self.btn\_tan = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = 'Tan', command =lambda : self.btn\_click('Tan(') )

self.btn\_tan.grid(row =6, column = 5)

#Row 8

#equation solver function

self.eqn = tk.Button(bottom\_frame, \*\*btn\_paremeters\_3, text = 'EQN', command = self.equation\_solver, )

self.eqn.grid(row = 8, column = 0)

#algebra function button

self.eng\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters\_3, text = 'Alg', command = self.Alg)

self.eng\_btn.grid(row = 8, column = 1)

#left bracket

self.left\_brac = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = '(', command = lambda:self.btn\_click('(') )

self.left\_brac.grid(row = 8, column = 2)

#right bracket

self.right\_brac = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = ')', command =lambda:self.btn\_click(')'))

self.right\_brac.grid(row = 8, column = 3)

#standard form button

self.sd = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = 'sd', command ='', )

self.sd.grid(row = 8, column = 4)

#

self.btn\_ln = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = 'M+', command ='' )

self.btn\_ln.grid(row = 8, column = 5)

#Row 9

#statistics function button

self.stat = tk.Button(bottom\_frame, text = 'STAT', \*\*btn\_paremeters\_3, command = self.Stat)

self.stat.grid(row=9, column = 0)

#Roots function button

self.btn\_ln = tk.Button(bottom\_frame, \*\*btn\_paremeters\_3, text = 'Roots', command =self.Roots )

self.btn\_ln.grid(row = 9, column = 1)

#LCM function button

self.btn\_ln = tk.Button(bottom\_frame, \*\*btn\_paremeters\_3, text = 'LCM', command = self.LCM )

self.btn\_ln.grid(row = 9, column = 2)

#polynomial button

self.polyn = tk.Button(bottom\_frame, text = 'Pol ', \*\*btn\_paremeters)

self.polyn.grid(row=9, column = 3)

#ceiling function button

self.floor = tk.Button(bottom\_frame, text = 'Ceil', \*\*btn\_paremeters, command= lambda: self.btn\_click('Ceil('))

self.floor.grid(row=9, column = 4)

#floor function button

self.floor = tk.Button(bottom\_frame, text = 'Floor', \*\*btn\_paremeters, command= lambda: self.btn\_click('Floor('))

self.floor.grid(row=9, column = 5)

#Row 10

#matrix function button

self.matrix\_btn = tk.Button(bottom\_frame, text = 'MATRIX', \*\*btn\_paremeters\_3, command = self.matrix)

self.matrix\_btn.grid(row=10, column = 0)

#vector function button

self.vec\_btn = tk.Button(bottom\_frame, text = 'VECTOR', \*\*btn\_paremeters\_3)

self.vec\_btn.grid(row=10, column = 1)

#calculus function

self.cube = tk.Button(bottom\_frame, text = 'Calc', \*\*btn\_paremeters\_3, command= self.Calc)

self.cube.grid(row=10, column = 2)

#permutation button

self.perm\_btn= tk.Button(bottom\_frame, text = 'nPr', \*\*btn\_paremeters, command = lambda: self.btn\_click('perm('))

self.perm\_btn.grid(row=10, column = 3)

#combination button

self.comb\_btn = tk.Button(bottom\_frame, text = 'nCr', \*\*btn\_paremeters, command = lambda: self.btn\_click('comb('))

self.comb\_btn.grid(row=10, column = 4)

#exponential button

self.exp = tk.Button(bottom\_frame, text = 'e^x', \*\*btn\_paremeters, command = self.btn\_click('m.exp('))

self.exp.grid(row=10, column = 5)

#Row 11

#seven button

self.seven\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters\_2, text = '7', command = lambda:self.btn\_click('7'), )

self.seven\_btn.grid(row = 11, column = 0)

#eight button

self.eight\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters\_2, text = '8', command = lambda:self.btn\_click('8'))

self.eight\_btn.grid(row = 11, column = 1)

#nine button

self.nine\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters\_2, text = '9', command =lambda:self.btn\_click('9'), )

self.nine\_btn.grid(row = 11, column = 2)

#delete function button

self.Del\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = 'DEL', command =self.btn\_clear )

self.Del\_btn.grid(row = 11, column = 3)

#clear all button

self.Ac\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = 'AC', command = self.btn\_clearAll, )

self.Ac\_btn.grid(row = 11, column = 4)

#GCD function button

self.btn\_ln = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = 'GCD', command = lambda: self.btn\_click('GCD(') )

self.btn\_ln.grid(row = 11, column = 5)

#Row 12

#four button

self.four\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters\_2, text = '4', command =lambda:self.btn\_click('4'), )

self.four\_btn.grid(row = 12, column = 0)

#five button

self.five\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters\_2, text = '5', command =lambda:self.btn\_click('5') )

self.five\_btn.grid(row = 12, column = 1)

#six button

self.six\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters\_2, text = '6', command =lambda:self.btn\_click('6'), )

self.six\_btn.grid(row = 12, column = 2)

#multiplication button

self.times\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = 'x', command =lambda:self.btn\_click('\*') )

self.times\_btn.grid(row = 12, column = 3)

#division button

self.div\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = '/', command =lambda:self.btn\_click('/'), )

self.div\_btn.grid(row = 12, column = 4)

#complex function button

self.complx = tk. Button(bottom\_frame, text = 'CMPLX',\*\*btn\_paremeters, command = '')

self.complx.grid(row=12, column = 5)

#Row 13

#Row 14

#one button

self.one\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters\_2, text = '1', command =lambda:self.btn\_click('1'), )

self.one\_btn.grid(row = 14, column = 0)

#two button

self.two\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters\_2, text = '2', command =lambda:self.btn\_click('2') )

self.two\_btn.grid(row = 14, column = 1)

#three button

self.three\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters\_2, text = '3', command =lambda:self.btn\_click('3'), )

self.three\_btn.grid(row = 14, column = 2)

#plus button

self.plus\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = '+', command =lambda:self.btn\_click('+') )

self.plus\_btn.grid(row = 14, column = 3)

#minus button

self.minus\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = '-', command =lambda:self.btn\_click('-'), )

self.minus\_btn.grid(row = 14, column = 4)

#Distribution button

self.distr = tk.Button(bottom\_frame, text = 'DISTR', \*\*btn\_paremeters)

self.distr.grid(row=14, column = 5)

#Row 16

#zero button

self.zero\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = '0', command =lambda:self.btn\_click('0'), )

self.zero\_btn.grid(row = 15, column = 0)

#dot button

self.dot\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = '.', command =lambda:self.btn\_click('.') )

self.dot\_btn.grid(row = 15, column = 1)

#comma button

self.exp\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = ',',command=lambda:self.btn\_click(','), )

self.exp\_btn.grid(row = 15, column = 2)

#Ans button

self.ans\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = 'Ans', command =self.Answer )

self.ans\_btn.grid(row = 15, column = 3)

#equal button

self.equal\_btn = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = '=', command =self.btn\_equal, )

self.equal\_btn.grid(row = 15, column = 4)

#mod n button

self.btn\_ln = tk.Button(bottom\_frame, \*\*btn\_paremeters, text = 'mod n', command =lambda :self.btn\_click('fmod('))

self.btn\_ln.grid(row = 15, column = 5)

#function for statisticsq

def Stat(self):

print('Choose task to perform:\n1. Mean \n2. Mode\n3. Median\n4. Standard Deviation\n5. Variance')

choice = float(input('Enter the appropriate number:'))

if choice ==1: #calculates mean

list\_1 = []

list\_2 = []

x = input('Enter data points separated by a space: ')

list\_1 = x.split()

for i in list\_1:

list\_2.append(float(i))

print('The mean is ', sum(list\_2)/len(list\_2))

elif choice ==2: #calculates mode

list\_1 = []

list\_2 = []

x = input('Enter data points separated by a space: ')

list\_1 = x.split()

for i in list\_1:

list\_2.append(float(i))

print('The mode is/are', st.mode(list\_2))

elif choice ==3:

list\_1 = []

list\_2 = []

x = input('Enter data points separated by a space: ')

list\_1 = x.split()

for i in list\_1:

list\_2.append(float(i))

print('The median is', st.median\_grouped(list\_2))

elif choice ==4:#calculates standard deviation

list\_1 = []

list\_2 = []

list\_3= []

list\_4 = []

a = input('Is the frequecy? y/n: ')

if a == 'y' or a == 'Y':

x = input('Enter data points separated by a space: ')

list\_1 = x.split()

for i in list\_1: # to convert the string to floats

list\_2.append(int(i)) # list of data points

y = input('Enter the frequncy of each data point accordinnly separated by space:')

list\_3 = y.split()

for j in list\_3: # to convert the string to floats

list\_4.append(int(j)) # list of frequency

c = [(x\*y) for x,y in zip(list\_2, list\_4)]

d = sum(c)/ sum(list\_4) # the mean

e = [(k - d)\*\*2 for k in list\_2 ] # list of the square of the devation

f = [(m\*n) for m,n in zip(list\_4, e)] #list of the product of the frequency and the deviation

print('The standard deviation is', m.sqrt(sum(f)/sum(list\_4)))

elif a =='N' or a =='n':

list\_1 = []

list\_2 = []

x = input('Enter data points separated by a space: ')

list\_1 = x.split()

for i in list\_1:

list\_2.append(int(i))

d = sum(list\_2)/len(list\_2)

list\_3 = [(k - d)\*\*2 for k in list\_2]

print('The standard deviation is ', m.sqrt(sum(list\_3)/len(list\_2)))

elif choice ==5: #calculates variance

list\_1 = []

list\_2 = []

list\_3= []

list\_4 = []

a = input('Is the frequecy? y/n: ')

if a == 'y' or a == 'Y':

x = input('Enter data points separated by a space: ')

list\_1 = x.split()

for i in list\_1: # to convert the string to floats

list\_2.append(int(i)) # list of data points

y = input('Enter the frequncy of each data point accordinnly separated by space:')

list\_3 = y.split()

for j in list\_3: # to convert the string to floats

list\_4.append(int(j)) # list of frequency

c = [(x\*y) for x,y in zip(list\_2, list\_4)]

d = sum(c)/ sum(list\_4) # the mean

e = [(k - d)\*\*2 for k in list\_2 ] # list of the square of the devation

f = [(m\*n) for m,n in zip(list\_4, e)] #list of the product of the frequency and the deviation

print('The variance is', sum(f)/sum(list\_4))

elif a =='N' or a =='n':

list\_1 = []

list\_2 = []

x = input('Enter data points separated by a space: ')

list\_1 = x.split()

for i in list\_1:

list\_2.append(int(i))

d = sum(list\_2)/len(list\_2)

list\_3 = [(k - d)\*\*2 for k in list\_2]

print('The variance is ', sum(list\_3)/len(list\_2))

else:

print('Wrong input!')

#function to expand and simplify algebraic expressions

def Alg(self):

#declared variables

a, b, c ,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z = sym.symbols('a, b, c ,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z')

print('Choose task to perform:\n1. Expand \n2. Simplify')

self.choice = float(input('Entter the task to perform enclosed in even parentheses:'))

#expands algebraic expressions e.g (x + y)\*\*5

if self.choice ==1:

expression = input('Enter the expression to expand: ')

print(sym.expand(expression))

#simplifes algebraic expressions

elif self.choice ==2:

expression = input('Enter the expression to expand: ')

print(sym.simplify(expression))

else:

print('Wrong Input!')

#function to find the limits, derivative, and integral of single variable functions

def Calc(self):

#declaration of all possible variables

a, b, c ,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z = sym.symbols('a, b, c ,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z')

print('Choose the task to perform:\n1.Derivative \n2. Limits \n3. Improper Integral \n4. Definite Integral')

self.choice = float(input('Enter the nnumber here: ')) #user's task to perform

#derivate of function

if self.choice ==1:

expression = input('Enter the function to differentiate: ')

var\_of\_func = input('Enter the variable e.g x, y or z: ')

order\_of\_dydx = input('Enter the order of the derivative eig 1, 2, or 3:')

print(sym.diff(sym.simplify(expression), var\_of\_func, order\_of\_dydx))

#limit of fucnction

elif self.choice == 2:

expression = input('Enter the expression of limit: ')

var\_of\_func = input('Enter the variable e.g x, y or z: ')

pt\_of\_limit = input('Enter the point of limit. If point of limit is infinity please enter d: ')

print(sym.limit(sym.simplify(expression), var\_of\_func, pt\_of\_limit))

#integral of function

elif self.choice ==3:

expression = input('Enter the expression to inegrate beginning special functions with sym. e.g sin(x) as sym.sin(x): ')

var\_of\_func = input('Enter the variable e.g x, y or z: ')

print(sym.integrate(sym.simplify(expression)))

#definite integral

elif self.choice == 4:

expression = input('Enter the expression to inegrate beginning special functions with sym. e.g sin(x) as sym.sin(x)')

var\_of\_func = input('Enter the variable e.g x, y or z: ')

lower\_limit = input('Enter the value of the lower limit:')

upper\_limit = input('Enter the value of the upper limit:')

print(sym.integrate(sym.simplify(expression), (var\_of\_func, lower\_limit, upper\_limit)))

else:

print('Wrong Input')

#fucntion to find the roots of polynomial equations

def Roots(self):

a = []

b = []

c = input('Enter the coeeficients of the polynomial in ascending powers of the variable separated by space: ')

a = c.split()

for i in a:

b.append(float(i))

v = np.polynomial.Polynomial(b)

print(v.roots())

#function to find the lcm of denominator

def LCM(self):

a = []

b = []

c = input('Enter the the numbers (max of 7 numbers) separated by a space: ') #accepts the denominators

a = c.split()

for i in a:

b.append(int(i))

arr = np.array(b)

print(np.lcm.reduce(arr)) #prints the lcm

#Matrix function

def matrix(self):

''' This fucntion evaluates the product, inverse and determinant of matrices

'''

print('Choose task to perform:\n1. Matrix Multiplication\n2. Determinant of Matrix\n3. Inverse of Matrix')

self.choice = int(input('Enter the task to pperfom eg 1,2 3'))

if self.choice ==1:

print('How manny matrices for multip[l;ication(min of 2 and max 0f 4)?')

choice = int(input('Enter the number of matrices here: '))

if choice ==2:

R\_1 = int(input('Enter the number of rows for the first matrix: '))#function for statistics#function for statistics

C\_1 = int(input('Enter the number of columns for the first matrix: '))

print('Enter all entries(row-wise) in a single line separated by space: ')

entries\_1 = list(map(float, input().split())) #accepts all the elements in matrix on a single line

R\_2 = int(input('Enter the number of rows for the 2nd matrix: '))

C\_2 = int(input('Enter the number of columns for the 2nd matrix: '))

print('Enter all entries(row-wise) in a single line separated by space: ')

entries\_2 = list(map(float, input().split()))

matrix\_1 = np.array(entries\_1).reshape(R\_1, C\_1)

matrix\_2 = np.array(entries\_2).reshape(R\_2, C\_2)

print(sym.Matrix(matrix\_1)\*sym.Matrix(matrix\_2))

elif choice==3:

R\_1 = int(input('Enter the number of rows for the first matrix: '))

C\_1 = int(input('Enter the number of columns for the first matrix: '))

print('Enter all entries(row-wise) in a single line separated by space: ')

entries\_1 = list(map(float, input().split()))

R\_2 = int(input('Enter the number of rows for the 2nd matrix: '))

C\_2 = int(input('Enter the number of columns for the 2nd matrix: '))

print('Enter all entries(row-wise) in a single line separated by space: ')

entries\_2 = list(map(float, input().split()))

R\_3 = int(input('Enter the number of rows for the 3RD matrix: '))

C\_3 = int(input('Enter the number of columns for the 3RD matrix: '))

print('Enter all entries(row-wise) in a single line separated by space: ')

entries\_3 = list(map(float, input().split()))

matrix\_1 = np.array(entries\_1).reshape(R\_1, C\_1)

matrix\_2 = np.array(entries\_2).reshape(R\_2, C\_2)

matrix\_3 = np.array(entries\_3).reshape(R\_3, C\_3)

print(sym.Matrix(matrix\_1)\*sym.Matrix(matrix\_2)\*sym.Matrix(matrix\_3))

elif choice ==4:

R\_1 = int(input('Enter the number of rows for the first matrix: '))

C\_1 = int(input('Enter the number of columns for the first matrix: '))

print('Enter all entries(row-wise) in a single line separated by space: ')

entries\_1 = list(map(float, input().split()))

R\_2 = int(input('Enter the number of rows for the 2nd matrix: '))

C\_2 = int(input('Enter the number of columns for the 2nd matrix: '))

print('Enter all entries(row-wise) in a single line separated by space: ')

entries\_2 = list(map(float, input().split()))

R\_3 = int(input('Enter the number of rows for the 3RD matrix: '))

C\_3 = int(input('Enter the number of columns for the 3RD matrix: '))

print('Enter all entries(row-wise) in a single line separated by space: ')

entries\_3 = list(map(float, input().split()))

R\_4 = int(input('Enter the number of rows for the 3RD matrix: '))

C\_4 = int(input('Enter the number of columns for the 3RD matrix: '))

print('Enter all entries(row-wise) in a single line separated by space: ')

entries\_4 = list(map(float, input().split()))

matrix\_1 = np.array(entries\_1).reshape(R\_1, C\_1)

matrix\_2 = np.array(entries\_2).reshape(R\_2, C\_2)

matrix\_3 = np.array(entries\_3).reshape(R\_3, C\_3)

matrix\_4 = np.array(entries\_4).reshape(R\_4, C\_4)

print(sym.Matrix(matrix\_1)\*sym.Matrix(matrix\_2)\*sym.Matrix(matrix\_3)\*sym.Matrix(matrix\_4))

elif self.choice==2:

R\_1 = int(input('Enter the number of rows for the first matrix: '))

C\_1 = int(input('Enter the number of columns for the first matrix: '))

print('Enter all entries(row-wise) in a single line separated by space: ')

entries = list(map(float, input().split()))

matrix\_1 = np.array(entries).reshape(R\_1, C\_1)

print(round(np.linalg.det(matrix\_1), 2))

elif self.choice ==3:

R\_1 = int(input('Enter the number of rows for the first matrix: '))

C\_1 = int(input('Enter the number of columns for the first matrix: '))

print('Enter all entries(row-wise) in a single line separated by space: ')

entries = list(map(float, input().split()))

matrix\_1 = np.array(entries).reshape(R\_1, C\_1)

print(np.linalg.inv(matrix\_1))

def equation\_solver(self):

'''This function solves linear system of equations '''

R\_1 = int(input('Enter the number of equations: '))

C\_1 = int(input('Enter the number of unknowns: '))

print('Enter all coefficients(row-wise) in a single line separated by space: ')

entries\_1 = list(map(float, input().split())) #gets matrix input from user

R\_2 = R\_1

C\_2 = int(input('Enter 1: '))

print('Enter all the constants of the equations accordingly: ')

entries\_2 = list(map(float, input().split()))

matrix\_1 = np.array(entries\_1).reshape(R\_1, C\_1)

matrix\_2 = np.array(entries\_2).reshape(R\_2, C\_2)

print(np.linalg.inv(matrix\_1)\*sym.Matrix(matrix\_2))

#to display items on screen

def btn\_click(self, x):

if len(self.expression)>=100:

self.expression = self.expression

self.input\_txt.set(self.expression)

else:

self.expression = self.expression + str(x)

self.input\_txt.set(self.expression)

#for backspace

def btn\_clear(self):

self.expression = self.expression[:-1]

self.input\_txt.set(self.expression)

#evaluate expressions on screen

def btn\_equal(self):

self.result = str(eval(self.expression))

self.expression = self.result

self.input\_txt.set(self.expression)

#clears all expressions screen

def btn\_clearAll(self):

self.expression =""

self.input\_txt.set(self.expression)

def hyp(self):

choice = int(input('1.sinh\n2.cosh\n3.tanh'))

if choice==1:

user\_entry= float(input('Enter number: '))

print(m.sinh(user\_entry))

elif choice==2:

user\_entry= float(input('Enter number: '))

print(m.cosh(user\_entry))

elif choice==3:

user\_entry= float(input('Enter number: '))

print(m.sinh(user\_entry))

else:

print('Wrong input!')

def Answer(self):

self.answer = self.sum\_up

self.expression = self.expression + self.answer

self.input\_txt.set(self.expression)

# uses whatever is stored in memory\_recall

def memory\_recall(self):

if self.expression == "":

self.input\_txt.set('0' + self.expression + self.recall)

else:

self.input\_txt.set(self.expression + self.recall)

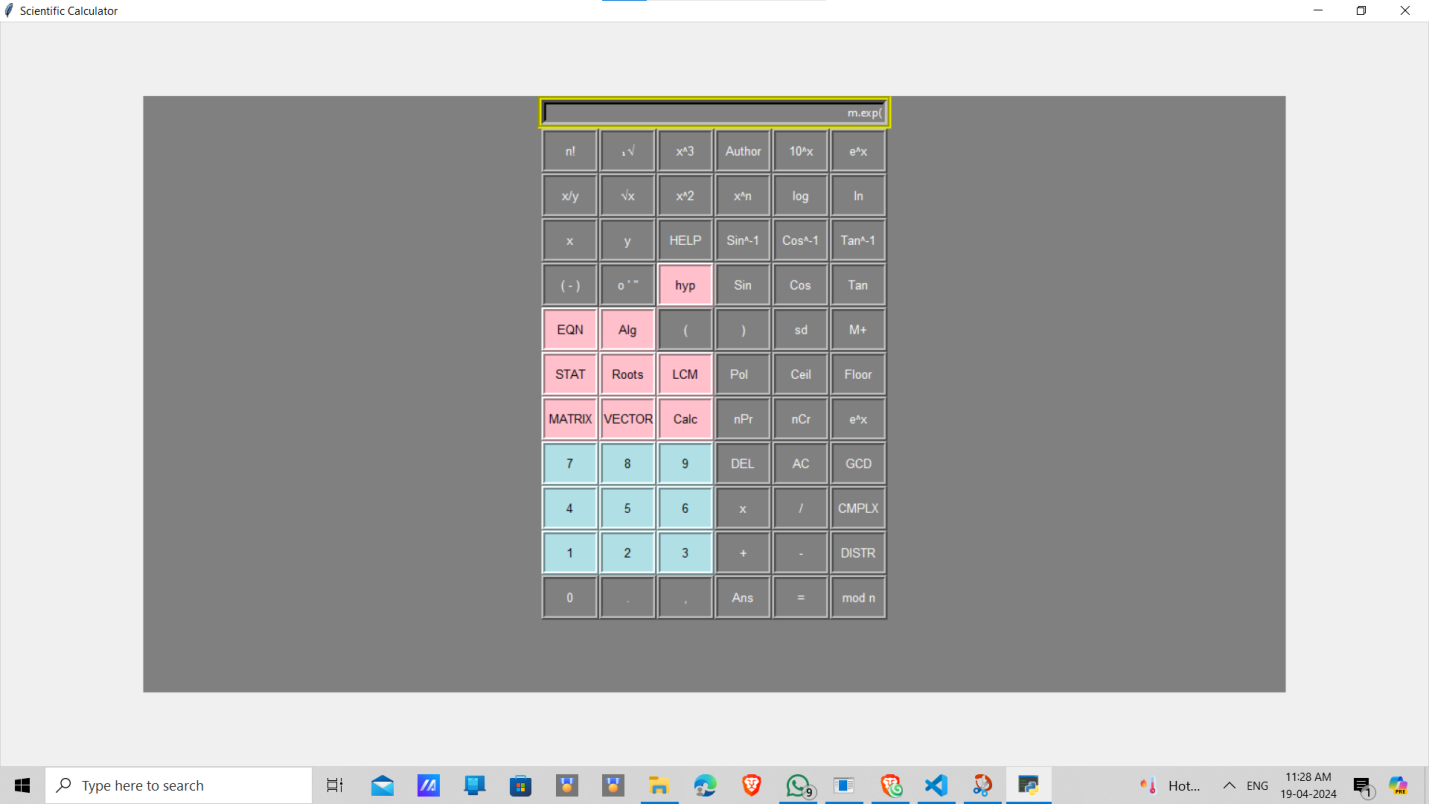
root = tk.Tk()

first\_gui = Sci\_Calculator(root)

root.geometry('')

root.mainloop()

**6.Results**



**7.Conclusion**

In conclusion, the development and implementation of a scientific calculator have been successfully achieved. The calculator provides a comprehensive set of mathematical functions, including basic arithmetic operations and advanced scientific capabilities. It handles multi-digit numbers, decimal points, and scientific notation with precision and accuracy. The user interface is intuitive and user-friendly, ensuring ease of use for a diverse range of users. The module has been tested extensively to ensure reliability and robustness. It is a valuable tool for students, professionals, and anyone in need of a feature-rich and accurate scientific calculator. The project's success underscores the potential of innovative technology in enhancing mathematical practices and education.

**8.References**

Stackoverflow,

tutorialpoints ,

github,

Openai