Calculator

Calculator Code

```
import tkinter as tk
import math
import tkinter.messagebox as messagebox
import tkinter.ttk as ttk
import datetime
import matplotlib.pyplot as plt
import numpy as np
from sympy import symbols, lambdify, solve, Eq, sympify
import random

# function for addition operation
def add_numbers():
    try:
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        result = num1 + num2
        label_result.config(text="Result: "+str(result))
        calculation_history.append(f"{num1} + {num2} = {result}") # Update the calculation history
    except ValueError:
        label_result.config(text="Invalid input! Please enter numeric values.")

# function for subtraction operation
def subtract_numbers():
```

```
    try:
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        result = num1 - num2
        label_result.config(text="Result: "+str(result))
    except ValueError:
        label_result.config(text="Invalid input! Please enter numeric values.")


# function for multiplication operation
def multiply_numbers():
    try:
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        result = num1 * num2
        label_result.config(text="Result: "+str(result))
    except ValueError:
        label_result.config(text="Invalid input! Please enter numeric values.")


# function for division operation
def divide_numbers():
    try:
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        if num2 == 0:
            label_result.config(text="Cannot divide by zero!")
        else:
```

```
            result = num1 / num2
            label_result.config(text="Result: "+str(result))
    except ValueError:
        label_result.config(text="Invalid input! Please enter numeric values.")


# function for remainder operation
def remainder_numbers():
    try:
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        result = num1 % num2
        label_result.config(text="Result: "+str(result))
    except ValueError:
        label_result.config(text="Invalid input! Please enter numeric values.")


# function for square root operation
def sqrt_numbers():
    try:
        num = float(entry1.get())
        if num < 0:
            label_result.config(text="Cannot calculate square root of a negative number!")
        else:
            result = math.sqrt(num)
            label_result.config(text="Result: "+str(result))
    except ValueError:
        label_result.config(text="Invalid input! Please enter numeric values.")
```

```
# function for exponentiation operation operation
def exponent_numbers():
    try:
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        result = num1 ** num2
        label_result.config(text="Result: "+str(result))
    except ValueError:
        label_result.config(text="Invalid input! Please enter numeric values.")

# function for logarithm operation
def logarithm_numbers():
    try:
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        if num1 <= 0 or num2 <= 0:
            label_result.config(text="Cannot take logarithm of a non-positive number!")
        else:
            result = math.log(num1, num2)
            label_result.config(text="Result: "+str(result))
    except ValueError:
        label_result.config(text="Invalid input! Please enter numeric values.")

# function for percentage operation
def percentage_numbers():
```

```
    try:
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        result = num1 * (num2 / 100)
        label_result.config(text="Result: "+str(result))
    except ValueError:
        label_result.config(text="Invalid input! Please enter numeric values.")


# function to clear input fields
def clear_inputs():
    try:
        entry1.delete(0, tk.END)
        entry2.delete(0, tk.END)
        label_result.config(text="Result:")
    except ValueError:
        label_result.config(text="Invalid input! Please enter numeric values.")


# define conversion rates as a directory
conversion_rates = {
    'metres': {'feet': 3.28084, 'inches': 39.3701},
    'feet': {'metres': 0.3048, 'inches': 12},
    'inches': {'metres': 0.0254, 'feet': 0.0833333},
    'celsius': {'fahrenheit': lambda x: (x * 9/5) + 32, 'kelvin': lambda x: x + 273.15},
    'fahrenheit': {'celsius': lambda x: (x - 32) * 5/9, 'kelvin': lambda x: (x + 459.67) * 5/9},
    'kilograms': {'pounds': 2.20462, 'ounces': 35.274},
    'pounds': {'kilograms': 0.453592, 'ounces': 16},
```

```
    'ounces': {'kilograms': 0.0283495, 'pounds': 0.0625}
}


def convert_units():
    from_unit = from_unit_var.get()
    to_unit = to_unit_var.get()
    try:
        value = float(entry1.get())
        if from_unit == to_unit:
            converted_value = value
        elif from_unit in conversion_rates and to_unit in conversion_rates[from_unit]:
            conversion_func = conversion_rates[from_unit][to_unit]
            if callable(conversion_func):
                converted_value = conversion_func(value)
            else:
                converted_value = value * conversion_func
        else:
            messagebox.showerror("Invalid Conversion", "Cannot convert from selected units.")
            return

        # update the result label
        label_result.config(text=f"Result: {converted_value:.2f} {to_unit}")
    except ValueError:
        label_result.config(text="Invalid input! Please enter numeric values.")

# function for Pi constant
```

```
def pi_constant():
    result = math.pi
    entry1.delete(0, tk.END)
    entry1.insert(tk.END, str(result))

# function for Euler's constant
def e_constant():
    result = math.e
    entry1.delete(0, tk.END)
    entry1.insert(tk.END, str(result))

# funtion to toggle between light and dark mode
def toggle_theme():
    global is_dark_mode

    if is_dark_mode:
        window.configure(bg=light_bg_color)
        label1.configure(bg=light_bg_color)
        label2.configure(bg=light_bg_color)
        is_dark_mode=False
    else:
        window.configure(bg=dark_bg_color)
        label1.configure(bg=light_bg_color)
        label2.configure(bg=light_bg_color)
        is_dark_mode=True
```

```
# declare a list to store the history of calculations
calculation_history = []

def show_history():
    history_window = tk.Toplevel(window)
    history_window.title("Calculation History")

    # create a text widget to show the history
    history_text = tk.Text(history_window, height=10, width=40, font=("Arial", 12))
    history_text.pack()

    # create a label to display the history
        history_label = tk.Label(history_window, text="Calculation History:", fg=light_fg_color,
font=("Arial", 12, "bold"))
    history_label.pack()

    # insert each calculation from the history list into the text widget
    for calculation in calculation_history:
        history_text.insert(tk.END, calculation + '
')

    #disable editing in the text widget
    history_text.configure(state='disabled')

def copy_to_clipboard():
    result = label_result.cget("text")
```

```
    window.clipboard_clear()
    window.clipboard_append(result)
    messagebox.showinfo("Copied", "Result has been copied to the clipboard.")


def save_history():
    try:
        now = datetime.datetime.now()
        filename = f"calculation_history_{now.strftime('%Y%m%d%H%M%S')}.txt"
        with open(filename, "w") as file:
            for calculation in calculation_history:
                file.write(calculation + "
")
        messagebox.showinfo("History Saved", "Calculation history has been saved to a file.")
    except Exception as e:
        messagebox.showerror("Error", "Failed to save calculation history.")


# funtion for plotting a graph
def plot_graph():
    try:
        expression = entry3.get()
        x = symbols('x')
        y = lambdify(x, expression, 'numpy')
        x_vals = np.linspace(-10, 10, 400)
        y_vals = y(x_vals)
        plt.plot(x_vals, y_vals)
        plt.xlabel('x')
```

```
        plt.ylabel('y')
        plt.title('Graph of ' + expression)
        plt.grid(True)
        plt.show()
    except Exception as e:
        messagebox.showerror("Error", str(e))

# function for equation solving
def solve_equation():
    try:
        equation = entry3.get()
        x = symbols('x')
        y = symbols('y')
        eq = Eq(eval(equation), y)
        x_value = float(entry1.get())
        eq_with_x_value = eq.subs(x, x_value)
        solution = solve(eq_with_x_value, y)
        label_result.config(text="Solutions: " + str(solution))
    except Exception as e:
        label_result.config(text="Error: "+ str(e))

def factorial_number():
    try:
        num = int(entry1.get())
        if num < 0:
            label_result.config(text="Cannot calculate factorial of a negative number!")
```

```
        else:
            result = math.factorial(num)
            label_result.config(text="Result: "+str(result))
    except ValueError:
        label_result.config(text="Invalid input! Please enter an integer")

# function for generating a random number for dice
def dice():
    try:
        min_value = 1
        max_value = 7
        result = int(random.uniform(min_value, max_value))
        label_result.config(text="Result: " + str(result))
    except ValueError:
        label_result.config(text="Invalid input! Please enter numeric values.")

# create the main window
window = tk.Tk()
window.title("Calculator")

# define colors for light and dark modes
light_bg_color = "white"
light_fg_color = "black"
dark_bg_color = "black"
dark_fg_color = "white"
```

```
# set the initial mode to light mode
is_dark_mode = False

# configure the styling options
window.configure(bg=light_bg_color)    # set the background color of the window

# create the widgets
label1 = tk.Label(window, text="Number 1:", fg=light_fg_color, font=("Arial", 12, "bold"))
entry1 = tk.Entry(window, bg=light_bg_color, fg=light_fg_color, font=("Arial", 12))

label2 = tk.Label(window, text="Number 2:", fg=light_fg_color, font=("Arial", 12, "bold"))
entry2 = tk.Entry(window, bg=light_bg_color, fg=light_fg_color, font=("Arial", 12))

label3 = tk.Label(window, text="Equation:", fg=light_fg_color, font=("Arial", 12, "bold"))
entry3 = tk.Entry(window, bg=light_bg_color, fg=light_fg_color, font=("Arial", 12))

# create the constant buttons
button_pi = tk.Button(window, text="Pi", command=pi_constant, fg=light_fg_color, font=("Arial", 12,
"bold"), padx=10, pady=5)
button_e = tk.Button(window, text="e", command=e_constant, fg=light_fg_color, font=("Arial", 12,
"bold"), padx=10, pady=5)

# create the widgets for unit conversion
label_from_unit = tk.Label(window, text="From Unit:", fg=light_fg_color, font=("Arial", 12, "bold"))
from_unit_var = tk.StringVar(window)
from_unit_var.set('metres') # set the default selected unit
```

```
from_unit_menu = tk.OptionMenu(window, from_unit_var, *conversion_rates.keys())


label_to_unit = tk.Label(window, text="To Unit:", fg=light_fg_color, font=("Arial", 12, "bold"))
to_unit_var = tk.StringVar(window)
to_unit_var.set('feet') # set the default selected unit
to_unit_menu = tk.OptionMenu(window, to_unit_var, *conversion_rates.keys())


button_add = tk.Button(window, text="Addition", command=add_numbers, fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)
button_subtract = tk.Button(window, text="Subtract", command=subtract_numbers, fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)
button_multiply = tk.Button(window, text="Multiply", command=multiply_numbers, fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)
button_divide = tk.Button(window, text="Divide", command=divide_numbers, fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)
button_remainder = tk.Button(window, text="Remainder", command=remainder_numbers, fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)
button_sqrt = tk.Button(window, text="Root", command=sqrt_numbers, fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)
button_exponent = tk.Button(window, text="Exponent", command=exponent_numbers, fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)
button_logarithm = tk.Button(window, text="Logarithm", command=logarithm_numbers, fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)
button_percentage = tk.Button(window, text="Percentage", command=percentage_numbers, fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)
button_units = tk.Button(window, text="Convert Unit", command=convert_units, fg=light_fg_color,
```

```
font=("Arial", 12, "bold"), padx=10, pady=5)
button_clear = tk.Button(window, text="Clear", command=clear_inputs, fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)
# create the factorial button
button_factorial = tk.Button(window, text="Factorial", command=factorial_number, fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)
# create the random number generator button
button_dice = tk.Button(window, text="Dice", command=dice, fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)

# create the toggle theme button
button_toggle_theme = tk.Button(window, text="Toggle Theme", command=toggle_theme, fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)

label_result = tk.Label(window, text="Result:")

# set the layout of the widgets
label1.grid(row=0, column=0)
entry1.grid(row=0, column=1)

label2.grid(row=1, column=0)
entry2.grid(row=1, column=1)

label3.grid(row=10, column=0)
entry3.grid(row=10,column=1)
```

```
button_add.grid(row=2, column=0)

button_subtract.grid(row=2, column=1)

button_multiply.grid(row=2, column=2)

button_divide.grid(row=3, column=0)

button_remainder.grid(row=3, column=1)

button_sqrt.grid(row=3, column=2)

button_exponent.grid(row=4, column=0)

button_logarithm.grid(row=4, column=1)

button_percentage.grid(row=4, column=2)

button_dice.grid(row=4, column=3)

button_units.grid(row=5, column=0)

button_clear.grid(row=5, column=1)

button_toggle_theme.grid(row=5, column=2)

button_factorial.grid(row=5, column=3)


# set the layout of the widgets for unit conversion

label_from_unit.grid(row=6, column=0)

from_unit_menu.grid(row=6, column=1)


label_to_unit.grid(row=6, column=2)

to_unit_menu.grid(row=6, column=3)


# set the layout of the widgets for constant buttons

button_pi.grid(row=7, column=0)

button_e.grid(row=7, column=1)
```

```
# create the widgets for the result display
label_result = tk.Label(window, text="Result:", fg=light_fg_color, font=("Arial", 12, "bold"))
label_result.grid(row=8, column=1)

# create a history button
button_history = tk.Button(window, text="History", command=show_history, fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)
button_history.grid(row=8, column=3)

# create the copy button
button_copy = tk.Button(window, text="Copy", command=copy_to_clipboard, fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)
button_copy.grid(row=8, column=4)

# create a save history button
button_save_history = tk.Button(window, text="Save History", command=save_history, fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)
button_save_history.grid(row=9, column=0)

# create the plot button
button_plot = tk.Button(window, text='Plot', command=plot_graph, fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)
button_plot.grid(row=9, column=1)

# create the solve button
button_solve = tk.Button(window, text="Solve Equation", command=solve_equation,
```

```
fg=light_fg_color, font=("Arial", 12, "bold"), padx=10, pady=5)

button_solve.grid(row=9, column=2)


label_result.grid(row=7, column=2, columnspan=3)


window.geometry("700x400")


# run the event loop

window.mainloop()
```