Final.py

```python
import tkinter as tk
from tkinter import Label, messagebox
import math

# Prime Checker Function
def is_prime(n):
    if n == 2:
        return True
    if n < 2 or n % 2 == 0:
        return False
    for i in range(3, int(math.sqrt(n)) + 1, 2):
        if n % i == 0:
            return False
    return True

# Combined Prime Check and Factor Calculation
def check_number2():
    try:
        m = int(entry_prime.get())  # Get input from "ENTER Check Number"

        # Check if the number is prime
        if is_prime(m):
            prime_result = "Prime"
            # result_entry.config(text="THAT IS PRIME", bg="green")
        else:
            prime_result = "Not Prime"
            # result_entry.config(text="NOT PRIME", bg="red")

        # Update the result in the "OUTPUT / RESULT" field
        result_entry.delete(0, tk.END)
        result_entry.insert(0, f"{prime_result}")
    except ValueError:
        messagebox.showerror("Error", "Enter a valid number!")
# Function to get factors of a number
def get_factors(num):
    factors = []
    for i in range(1, int(num**0.5) + 1):
        if num % i == 0:
            factors.append(i)
            if i != num // i:  # Avoid duplicate factors
                factors.append(num // i)
    factors.sort()  # Sort the factors
    return factors

# Adjusted function to handle both Prime Check and Factorization
def check_number():
    try:
```

```
        m = int(entry_prime.get())  # Get input from "ENTER Check Number"

        # Calculate factors
        factors = get_factors (m)

        # Update the result in the "OUTPUT / RESULT" field
        result_entry.delete(0, tk.END)
        result_entry.insert(0, f"Factors: {factors}")
    except ValueError:
        messagebox.showerror("Error", "Enter a valid number!")



# Modulus Calculation with Quotient and Remainder
def calculate_modulus(event=None):  # Add event parameter to handle Enter key
    """Calculates q (quotient) and r (remainder), and handles errors for negative m."""
    global m, n, q, r
    try:
        # Get values for m and n from the entry fields
        m = int(entry_first.get())
        n = int(entry_second.get())

        # Validate values
        if m < 0:
            raise ValueError("m cannot be negative. Enter a positive value for m.")
        if n <= 0:
            raise ValueError("n must be greater than 0.")

        # Calculate quotient (q) and remainder (r)
        q = m // n
        r = m - q * n

        # Update the result field
        result_entry.delete(0, tk.END)
        result_entry.insert(0, f"{m} = {q}({n}) + {r}")

    except ValueError as e:
        messagebox.showerror("Input Error", str(e))
    except ZeroDivisionError:
        messagebox.showerror("Math Error", "Division by zero is not allowed.")

def calculate_mod_method(method):
    try:
        m = int(entry_first.get())
        n = int(entry_second.get())

        # Check if m is positive and show error if so
        if m > 0:
            messagebox.showerror("Input Error", "Value of m must be negative.")
```

```
            return  # Exit the function early if m is positive

        if method == "method1":
            #Calculating modulus for negative m by : m+n
            result = m
            if m < 0:
                while result < 0:
                    result += n
                if result == 0:  # Special condition to handle exact multiples
                    result += n
                messagebox.showinfo (f"Value: {result}")

        elif method == "method2":
            # Updated logic for method2: By rneg= n - rnag
            result = m
            if m < 0:
                q = int(-m / n)
                r = -m - (q * n)
                a = n - r
                q = int((m - a) / n)
                result = f"{m} = {q}({n}) + {a}"

        elif method == "method3":
            # By : m = -q*n+r
            q = -(m // n)
            r = m + q * n
            result = r
            messagebox.showinfo (f"m = -q*n+r" , result)

        result_entry.delete(0, tk.END)
        result_entry.insert(0, f"Result: {result}")
    except ValueError:
        messagebox.showerror("Error", "Enter valid numbers!")




# GCD Calculation
def calculate_gcd(method):
    try:
        m = int(entry_first.get())
        n = int(entry_second.get())
        if method == "gcd1":
            while n:
                m, n = n, m % n
        elif method == "gcd2":
            if m <0 or n <0:
                messagebox.showerror("Error", "Both numbers should be non-negative.")
                return
```

```python
            # Function to compute GCD using the prime factors method
            def get_factors(num):
                factors = []
                for i in range(1, int(num**0.5) + 1):
                    if num % i == 0:
                        factors.append(i)
                        if i != num // i:  # To avoid adding square roots twice
                            factors.append(num // i)
                return factors

            def gcd(a, b):
                # Get the factors of both numbers
                factors_a = get_factors(a)
                factors_b = get_factors(b)

                # Find the common factors
                common_factors = list(set(factors_a) & set(factors_b))

                # Return the greatest common factor
                return max(common_factors)

            m = gcd(m, n)
            messagebox.showinfo("GCD using the prime factors", m)

        elif method == "gcd3":
            # Function to compute GCD using the Successive Difference method
            def last_num(a, b):
                check = True
                if a <0 or b <0:
                    check = False
                while a != b:
                    if check == False:
                        messagebox.showerror("Error", "Both numbers should be non-negative.")
                        return
                    elif a > b:
                        a = a - b
                    else:
                        b = b - a
                return b

            m = last_num(m, n)
            messagebox.showinfo("That By Successive Difference", m)

        elif method == "gcd4":
            # Function to compute GCD using a Algorithm
            def compute_GCD(a, b):
                if b == 0:
                    return a
                else:
```

```python
                    return compute_GCD(b, a % b)

            m = compute_GCD(m, n)
            messagebox.showinfo("That By Algorithm Method", m)

        result_entry.delete(0, tk.END)
        result_entry.insert(0, f"GCD: {m}")
    except ValueError:
        messagebox.showerror("Error", "Enter valid numbers!")

# LCM Calculation
def calculate_lcm(method):
    try:
        m = int(entry_first.get())
        n = int(entry_second.get())

        if method == "lcm1":
            # Function to compute GCD using the Euclidean Algorithm
            def gcd(a, b):
                while b != 0:
                    a, b = b, a % b
                return a

            # Function to compute LCM using the formula: LCM = (a * b) / GCD(a, b)
            def lcm(a, b):
                return abs(a * b) // gcd(a, b)

            lcm_value = lcm(m, n)
            messagebox.showinfo("LCM by Thorey:", lcm_value)

        elif method == "lcm2":
            # Function to compute LCM (Least Common Multiple) using the 'Tree' method
            def compute_LSM(a, b):
                if b > a:
                    hight = a
                else:
                    hight = b
                value = hight  # To calculate the next index
                while True:
                    if hight % a == 0 and hight % b == 0:
                        return hight
                    else:
                        hight = hight + value  # Go to the next index

            lcm_value = compute_LSM(m, n)
            messagebox.showinfo("LCM by Tree Method:", lcm_value)

        result_entry.delete(0, tk.END)
        result_entry.insert(0, f"LCM is: {lcm_value}")
```

```python

    except ValueError:
        messagebox.showerror("Error", "Enter valid numbers!")

# Tkinter GUI Design
root = tk.Tk()
root.title("Math Operations")
root.geometry("900x600")
root.config(bg="#19535f")

# Input Fields That title and what it take
tk.Label(root, text="Enter Check Number", fg="white", bg="#2b76d4", font=("Arial", 12)).place(x=50, y=10)
entry_prime = tk.Entry(root, font=("Arial", 14), width=10)
entry_prime.place(x=60, y=35)

tk.Label(root, text="Enter First Number", fg="white", bg="#2b76d4", font=("Arial", 12)).place(x=250, y=10)
entry_first = tk.Entry(root, font=("Arial", 14), width=10)
entry_first.place(x=260, y=35)

tk.Label(root, text="Enter Second Number", fg="white", bg="#2b76d4", font=("Arial", 12)).place(x=450, y=10)
entry_second = tk.Entry(root, font=("Arial", 14), width=10)
entry_second.place(x=475, y=35)

tk.Label(root, text="OUTPUT / RESULT", fg="white", bg="#ec9713", font=("Arial", 12)).place(x=670, y=10)
result_entry = tk.Entry(root, font=("Arial", 14), width=20)
result_entry.place(x=640, y=35)


# Buttons for that Color , witdh
button_check_number = tk.Button(root, text="Check Number", command=check_number2,bg="#53be41", width=18)
button_factors = tk.Button(root, text="Factors", command=check_number,bg="#53be41", width=18)
button_pos_mod = tk.Button(root, text="Pos Mod", command=calculate_modulus,bg="#2d4793",fg="#da1e14" , width=18)
button_neg_mod1 = tk.Button(root, text="Neg Mod1", command=lambda: calculate_mod_method("method1"),bg="#2d4793",fg="#da1e14", width=18)
button_neg_mod2 = tk.Button(root, text="Neg Mod2", command=lambda: calculate_mod_method("method2"),bg="#2d4793" ,fg="#da1e14", width=18)
button_neg_mod3 = tk.Button(root, text="Neg Mod3", command=lambda: calculate_mod_method("method3"),bg="#2d4793",fg="#da1e14" , width=18)
button_gcd1 = tk.Button(root, text="GCD1", command=lambda: calculate_gcd("gcd1"),bg="yellow", width=18)
button_gcd2 = tk.Button(root, text="GCD2", command=lambda: calculate_gcd("gcd2"),bg="yellow", width=18)
button_gcd3 = tk.Button(root, text="GCD3", command=lambda: calculate_gcd("gcd3"),bg="yellow", width=18)
```

```
button_gcd4 = tk.Button(root, text="GCD4", command=lambda: calculate_gcd("gcd4"),bg="yellow", width=18)
button_lcm1 = tk.Button(root, text="LCM1", command=lambda: calculate_lcm("lcm1"),bg="#9e339b", width=18)
button_lcm2 = tk.Button(root, text="LCM2", command=lambda: calculate_lcm("lcm2"),bg="#9e339b" , width=18 )

# Place buttons on the window
button_check_number.place(x=60, y=110)
button_factors.place(x=650, y=110)

button_pos_mod.place(x=60, y=220)
button_neg_mod1.place(x=260, y=220)
button_neg_mod2.place(x=460, y=220)
button_neg_mod3.place(x=660, y=220)

button_gcd1.place(x=60, y=350)
button_gcd2.place(x=260, y=350)
button_gcd3.place(x=460, y=350)
button_gcd4.place(x=650, y=350)

button_lcm1.place(x=60, y=500)
button_lcm2.place(x=650, y=500)


#Text Buttons :
txt1 = Label(text = 'Ckeck Prime or Not : ',fg= 'white' , bg = '#19535f', font=20)
txt2 = Label(text = 'Return Factors for Number : ',fg= 'white' , bg = '#19535f', font=20)
txt3 = Label(text = 'Mod When m post : ',fg= 'white' , bg = '#19535f', font=20)
txt4 = Label(text = 'Mod m_nag=> sum ',fg= 'white' , bg = '#19535f', font=20)
txt5 = Label(text = 'Mod m_nag=> r_nag ',fg= 'white' , bg = '#19535f', font=20)
txt6 = Label(text = 'Mod m_nag=>-q law ',fg= 'white' , bg = '#19535f', font=20)
txt7 = Label(text = 'GCD by=> Prime-Factors ',fg= 'white' , bg = '#19535f', font=20)
txt8 = Label(text = 'GCD by=> Tree',fg= 'white' , bg = '#19535f', font=20)
txt9 = Label(text = 'GCD by=> Successive',fg= 'white' , bg = '#19535f', font=20)
txt10= Label(text = 'GCD by=> Algorithm',fg= 'white' , bg = '#19535f', font=20)
txt11= Label(text = 'LCM by=> Prime-Factors ',fg= 'white' , bg = '#19535f', font=20)
txt12= Label(text = 'LCM by=> Tree ',fg= 'white' , bg = '#19535f', font=20)

# Text Place :
txt1.place(x='50' , y = '80')
txt2.place(x='620' , y = '80')

txt3.place(x='50' , y = '180')
txt4.place(x='250' , y = '180')
txt5.place(x='450' , y = '180')
txt6.place(x='650' , y = '180')

txt7.place(x='50' , y = '290')
```

```
txt8.place(x='270' , y = '290')
txt9.place(x='450' , y = '290')
txt10.place(x='650' , y = '290')

txt11.place(x='50' , y = '450')
txt12.place(x='650' , y = '450')
# Render Buttons
# for i, (text, command) in enumerate(buttons):
#     tk.Button(root, text=text, bg="yellow", font=("Arial", 10), width=15,
command=command).place(x=50, y=50 + i * 40)
root.mainloop()

```