

Final.py

```
1 import tkinter as tk
2 from tkinter import Label, messagebox
3 import math
4
5 # Prime Checker Function
6 def is_prime(n):
7     if n == 2:
8         return True
9     if n < 2 or n % 2 == 0:
10         return False
11     for i in range(3, int(math.sqrt(n)) + 1, 2):
12         if n % i == 0:
13             return False
14     return True
15
16 # Combined Prime Check and Factor Calculation
17 def check_number2():
18     try:
19         m = int(entry_prime.get()) # Get input from "ENTER Check Number"
20
21         # Check if the number is prime
22         if is_prime(m):
23             prime_result = "Prime"
24             # result_entry.config(text="THAT IS PRIME", bg="green")
25         else:
26             prime_result = "Not Prime"
27             # result_entry.config(text="NOT PRIME", bg="red")
28
29         # Update the result in the "OUTPUT / RESULT" field
30         result_entry.delete(0, tk.END)
31         result_entry.insert(0, f"{prime_result}")
32     except ValueError:
33         messagebox.showerror("Error", "Enter a valid number!")
34
35 # Function to get factors of a number
36 def get_factors(num):
37     factors = []
38     for i in range(1, int(num**0.5) + 1):
39         if num % i == 0:
40             factors.append(i)
41             if i != num // i: # Avoid duplicate factors
42                 factors.append(num // i)
43     factors.sort() # Sort the factors
44     return factors
45
46 # Adjusted function to handle both Prime Check and Factorization
47 def check_number():
48     try:
```

```
48     m = int(entry_prime.get()) # Get input from "ENTER Check Number"
49
50     # Calculate factors
51     factors = get_factors (m)
52
53     # Update the result in the "OUTPUT / RESULT" field
54     result_entry.delete(0, tk.END)
55     result_entry.insert(0, f"Factors: {factors}")
56 except ValueError:
57     messagebox.showerror("Error", "Enter a valid number!")
58
59
60
61 # Modulus Calculation with Quotient and Remainder
62 def calculate_modulus(event=None): # Add event parameter to handle Enter key
63     """Calculates q (quotient) and r (remainder), and handles errors for negative m."""
64     global m, n, q, r
65     try:
66         # Get values for m and n from the entry fields
67         m = int(entry_first.get())
68         n = int(entry_second.get())
69
70         # Validate values
71         if m < 0:
72             raise ValueError("m cannot be negative. Enter a positive value for m.")
73         if n <= 0:
74             raise ValueError("n must be greater than 0.")
75
76         # Calculate quotient (q) and remainder (r)
77         q = m // n
78         r = m - q * n
79
80         # Update the result field
81         result_entry.delete(0, tk.END)
82         result_entry.insert(0, f"{m} = {q}({n}) + {r}")
83
84     except ValueError as e:
85         messagebox.showerror("Input Error", str(e))
86     except ZeroDivisionError:
87         messagebox.showerror("Math Error", "Division by zero is not allowed.")
88
89 def calculate_mod_method(method):
90     try:
91         m = int(entry_first.get())
92         n = int(entry_second.get())
93
94         # Check if m is positive and show error if so
95         if m > 0:
96             messagebox.showerror("Input Error", "Value of m must be negative.")
```

```
97         return # Exit the function early if m is positive
98
99     if method == "method1":
100         #Calculating modulus for negative m by : m+n
101         result = m
102         if m < 0:
103             while result < 0:
104                 result += n
105             if result == 0: # Special condition to handle exact multiples
106                 result += n
107             messagebox.showinfo (f"Value: {result}")
108
109     elif method == "method2":
110         # Updated logic for method2: By rneg= n - rnag
111         result = m
112         if m < 0:
113             q = int(-m / n)
114             r = -m - (q * n)
115             a = n - r
116             q = int((m - a) / n)
117             result = f"{m} = {q}({n}) + {a}"
118
119     elif method == "method3":
120         # By : m = -q*n+r
121         q = -(m // n)
122         r = m + q * n
123         result = r
124         messagebox.showinfo (f"m = -q*n+r" , result)
125
126     result_entry.delete(0, tk.END)
127     result_entry.insert(0, f"Result: {result}")
128 except ValueError:
129     messagebox.showerror("Error", "Enter valid numbers!")
130
131
132
133
134 # GCD Calculation
135 def calculate_gcd(method):
136     try:
137         m = int(entry_first.get())
138         n = int(entry_second.get())
139         if method == "gcd1":
140             while n:
141                 m, n = n, m % n
142         elif method == "gcd2":
143             if m < 0 or n < 0:
144                 messagebox.showerror("Error", "Both numbers should be non-negative.")
145             return
```

```
146     # Function to compute GCD using the prime factors method
147     def get_factors(num):
148         factors = []
149         for i in range(1, int(num**0.5) + 1):
150             if num % i == 0:
151                 factors.append(i)
152                 if i != num // i: # To avoid adding square roots twice
153                     factors.append(num // i)
154         return factors
155
156     def gcd(a, b):
157         # Get the factors of both numbers
158         factors_a = get_factors(a)
159         factors_b = get_factors(b)
160
161         # Find the common factors
162         common_factors = list(set(factors_a) & set(factors_b))
163
164         # Return the greatest common factor
165         return max(common_factors)
166
167     m = gcd(m, n)
168     messagebox.showinfo("GCD using the prime factors", m)
169
170 elif method == "gcd3":
171     # Function to compute GCD using the Successive Difference method
172     def last_num(a, b):
173         check = True
174         if a < 0 or b < 0:
175             check = False
176         while a != b:
177             if check == False:
178                 messagebox.showerror("Error", "Both numbers should be non-negative.")
179                 return
180             elif a > b:
181                 a = a - b
182             else:
183                 b = b - a
184         return b
185
186     m = last_num(m, n)
187     messagebox.showinfo("That By Successive Difference", m)
188
189 elif method == "gcd4":
190     # Function to compute GCD using a Algorithm
191     def compute_GCD(a, b):
192         if b == 0:
193             return a
194         else:
```

```
195         return compute_GCD(b, a % b)
196
197     m = compute_GCD(m, n)
198     messagebox.showinfo("That By Algorithm Method", m)
199
200     result_entry.delete(0, tk.END)
201     result_entry.insert(0, f"GCD: {m}")
202 except ValueError:
203     messagebox.showerror("Error", "Enter valid numbers!")
204
205 # LCM Calculation
206 def calculate_lcm(method):
207     try:
208         m = int(entry_first.get())
209         n = int(entry_second.get())
210
211         if method == "lcm1":
212             # Function to compute GCD using the Euclidean Algorithm
213             def gcd(a, b):
214                 while b != 0:
215                     a, b = b, a % b
216                 return a
217
218             # Function to compute LCM using the formula: LCM = (a * b) / GCD(a, b)
219             def lcm(a, b):
220                 return abs(a * b) // gcd(a, b)
221
222             lcm_value = lcm(m, n)
223             messagebox.showinfo("LCM by Thorey:", lcm_value)
224
225         elif method == "lcm2":
226             # Function to compute LCM (Least Common Multiple) using the 'Tree' method
227             def compute_LSM(a, b):
228                 if b > a:
229                     hight = a
230                 else:
231                     hight = b
232                 value = hight # To calculate the next index
233                 while True:
234                     if hight % a == 0 and hight % b == 0:
235                         return hight
236                     else:
237                         hight = hight + value # Go to the next index
238
239             lcm_value = compute_LSM(m, n)
240             messagebox.showinfo("LCM by Tree Method:", lcm_value)
241
242     result_entry.delete(0, tk.END)
243     result_entry.insert(0, f"LCM is: {lcm_value}")
```

```
244
245     except ValueError:
246         messagebox.showerror("Error", "Enter valid numbers!")
247
248 # Tkinter GUI Design
249 root = tk.Tk()
250 root.title("Math Operations")
251 root.geometry("900x600")
252 root.config(bg="#19535f")
253
254 # Input Fields That title and what it take
255 tk.Label(root, text="Enter Check Number", fg="white", bg="#2b76d4", font=("Arial",
256 12)).place(x=50, y=10)
257 entry_prime = tk.Entry(root, font=("Arial", 14), width=10)
258 entry_prime.place(x=60, y=35)
259
260 tk.Label(root, text="Enter First Number", fg="white", bg="#2b76d4", font=("Arial",
261 12)).place(x=250, y=10)
262 entry_first = tk.Entry(root, font=("Arial", 14), width=10)
263 entry_first.place(x=260, y=35)
264
265 tk.Label(root, text="Enter Second Number", fg="white", bg="#2b76d4", font=("Arial",
266 12)).place(x=450, y=10)
267 entry_second = tk.Entry(root, font=("Arial", 14), width=10)
268 entry_second.place(x=475, y=35)
269
270 tk.Label(root, text="OUTPUT / RESULT", fg="white", bg="#ec9713", font=("Arial",
271 12)).place(x=670, y=10)
272 result_entry = tk.Entry(root, font=("Arial", 14), width=20)
273 result_entry.place(x=640, y=35)
274
275 # Buttons for that Color , withd
276 button_check_number = tk.Button(root, text="Check Number", command=check_number2,bg="#53be41"
277 , width=18)
278 button_factors = tk.Button(root, text="Factors", command=check_number,bg="#53be41", width=18)
279 button_pos_mod = tk.Button(root, text="Pos Mod",
280 command=calculate_modulus,bg="#2d4793",fg="#da1e14" , width=18)
281 button_neg_mod1 = tk.Button(root, text="Neg Mod1", command=lambda: calculate_mod_method↵
282 ("method1"),bg="#2d4793",fg="#da1e14", width=18)
283 button_neg_mod2 = tk.Button(root, text="Neg Mod2", command=lambda: calculate_mod_method↵
284 ("method2"),bg="#2d4793" ,fg="#da1e14", width=18)
285 button_neg_mod3 = tk.Button(root, text="Neg Mod3", command=lambda: calculate_mod_method↵
286 ("method3"),bg="#2d4793",fg="#da1e14" , width=18)
287 button_gcd1 = tk.Button(root, text="GCD1", command=lambda: calculate_gcd("gcd1"),bg="yellow"
288 , width=18)
289 button_gcd2 = tk.Button(root, text="GCD2", command=lambda: calculate_gcd("gcd2"),bg="yellow"
290 , width=18)
291 button_gcd3 = tk.Button(root, text="GCD3", command=lambda: calculate_gcd("gcd3"),bg="yellow"
292 , width=18)
```

```
282 button_gcd4 = tk.Button(root, text="GCD4", command=lambda: calculate_gcd("gcd4"),bg="yellow"
    , width=18)
283 button_lcm1 = tk.Button(root, text="LCM1", command=lambda: calculate_lcm("lcm1"),bg="#9e339b"
    , width=18)
284 button_lcm2 = tk.Button(root, text="LCM2", command=lambda: calculate_lcm("lcm2")
    ,bg="#9e339b" , width=18 )
285
286 # Place buttons on the window
287 button_check_number.place(x=60, y=110)
288 button_factors.place(x=650, y=110)
289
290 button_pos_mod.place(x=60, y=220)
291 button_neg_mod1.place(x=260, y=220)
292 button_neg_mod2.place(x=460, y=220)
293 button_neg_mod3.place(x=660, y=220)
294
295 button_gcd1.place(x=60, y=350)
296 button_gcd2.place(x=260, y=350)
297 button_gcd3.place(x=460, y=350)
298 button_gcd4.place(x=650, y=350)
299
300 button_lcm1.place(x=60, y=500)
301 button_lcm2.place(x=650, y=500)
302
303
304 #Text Buttons :
305 txt1 = Label(text = 'Cckeck Prime or Not : ',fg= 'white' , bg = '#19535f', font=20)
306 txt2 = Label(text = 'Return Factors for Number : ',fg= 'white' , bg = '#19535f', font=20)
307 txt3 = Label(text = 'Mod When m post : ',fg= 'white' , bg = '#19535f', font=20)
308 txt4 = Label(text = 'Mod m_nag=> sum ',fg= 'white' , bg = '#19535f', font=20)
309 txt5 = Label(text = 'Mod m_nag=> r_nag ',fg= 'white' , bg = '#19535f', font=20)
310 txt6 = Label(text = 'Mod m_nag=>-q law ',fg= 'white' , bg = '#19535f', font=20)
311 txt7 = Label(text = 'GCD by=> Prime-Factors ',fg= 'white' , bg = '#19535f', font=20)
312 txt8 = Label(text = 'GCD by=> Tree',fg= 'white' , bg = '#19535f', font=20)
313 txt9 = Label(text = 'GCD by=> Successive',fg= 'white' , bg = '#19535f', font=20)
314 txt10= Label(text = 'GCD by=> Algorithm',fg= 'white' , bg = '#19535f', font=20)
315 txt11= Label(text = 'LCM by=> Prime-Factors ',fg= 'white' , bg = '#19535f', font=20)
316 txt12= Label(text = 'LCM by=> Tree ',fg= 'white' , bg = '#19535f', font=20)
317
318 # Text Place :
319 txt1.place(x='50' , y = '80')
320 txt2.place(x='620' , y = '80')
321
322 txt3.place(x='50' , y = '180')
323 txt4.place(x='250' , y = '180')
324 txt5.place(x='450' , y = '180')
325 txt6.place(x='650' , y = '180')
326
327 txt7.place(x='50' , y = '290')
```

```
328 txt8.place(x='270' , y = '290')
329 txt9.place(x='450' , y = '290')
330 txt10.place(x='650' , y = '290')
331
332 txt11.place(x='50' , y = '450')
333 txt12.place(x='650' , y = '450')
334 # Render Buttons
335 # for i, (text, command) in enumerate(buttons):
336 #     tk.Button(root, text=text, bg="yellow", font=("Arial", 10), width=15,
337 #               command=command).place(x=50, y=50 + i * 40)
338 root.mainloop()
```