

**Cognorise Infotech**

Internship Program

App development

**NAME**: TANISHA THAKUR

**FROM**: AURORA DEEMED TO BE UNIVERSITY

**TASK – 1**

**CALCULATOR APP**

Create a straightforward calculator application capable of executing fundamental arithmetic functions, including addition, subtraction, multiplication, and division.

Enhance the app’s capabilities by incorporating features such as input clearance and the ability to handle decimal numbers.

import tkinter as tk

from tkinter import ttk

# Define the main application class

class CalculatorApp:

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

self.root = root

self.root.title("Calculator")

self.expression = ""

self.result\_var = tk.StringVar()

self.create\_widgets()

def create\_widgets(self):

# Display screen

display = ttk.Entry(self.root, textvariable=self.result\_var, font=('Arial', 20), justify='right', state='readonly')

display.grid(row=0, column=0, columnspan=4, sticky='nsew')

# Button layout

buttons = [

'7', '8', '9', '/',

'4', '5', '6', '\*',

'1', '2', '3', '-',

'0', '.', '=', '+',

'C'

]

row\_val = 1

col\_val = 0



for button in buttons:

if button == 'C':

btn = ttk.Button(self.root, text=button, command=self.clear\_input)

elif button == '=':

btn = ttk.Button(self.root, text=button, command=self.calculate\_result)

else:

btn = ttk.Button(self.root, text=button, command=lambda b=button: self.update\_expression(b))

btn.grid(row=row\_val, column=col\_val, sticky='nsew')

col\_val += 1

if col\_val > 3:

col\_val = 0

row\_val += 1

# Configure grid layout

for i in range(5):

self.root.rowconfigure(i, weight=1)

self.root.columnconfigure(i, weight=1)

def update\_expression(self, value):

self.expression += str(value)

self.result\_var.set(self.expression)



def calculate\_result(self):

try:

self.result\_var.set(str(eval(self.expression)))

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

except Exception as e:

self.result\_var.set("Error")

self.expression = ""

def clear\_input(self):

self.expression = ""

self.result\_var.set("")

# Initialize and run the application

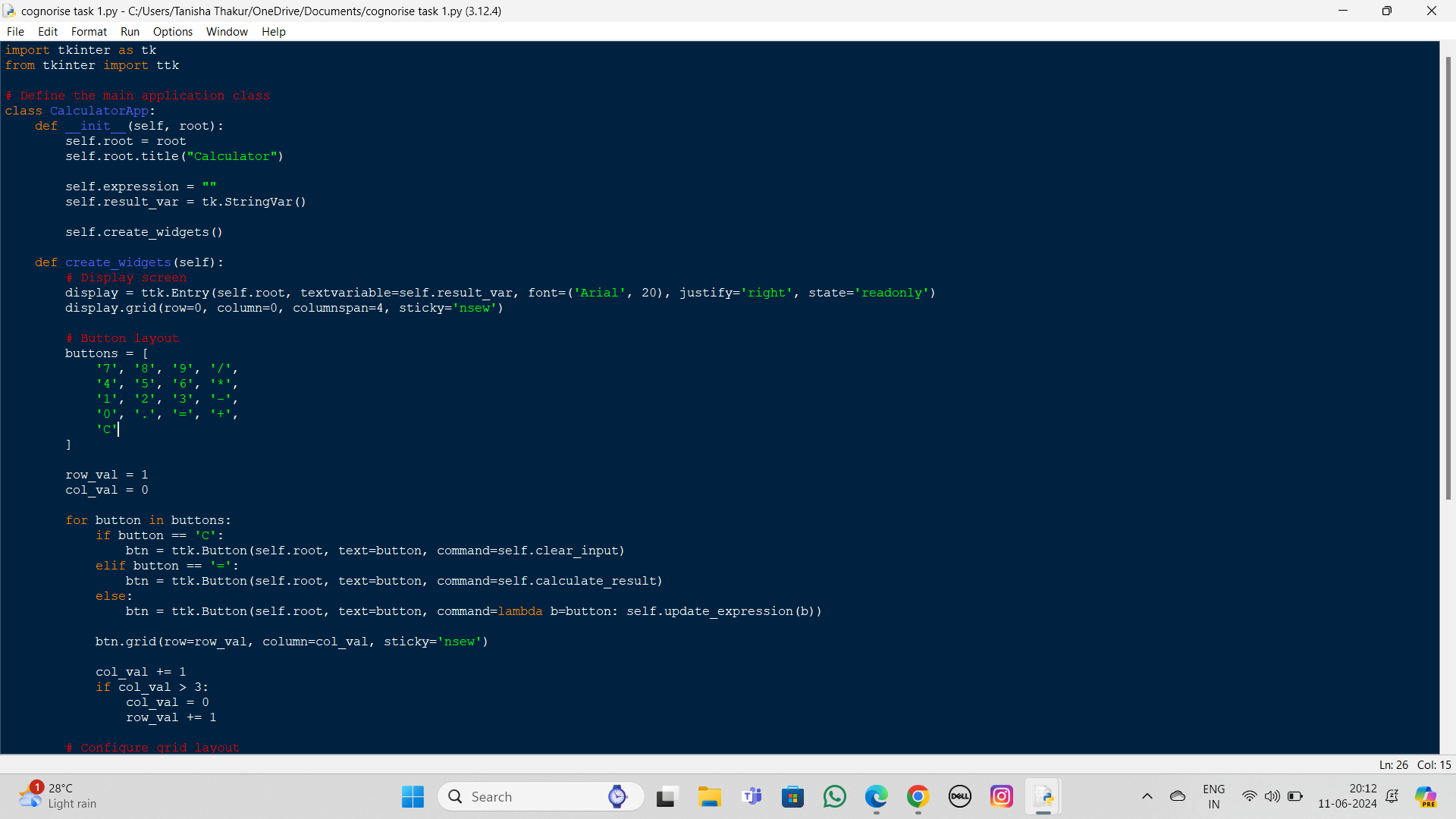
if \_\_name\_\_ == "\_\_main\_\_":

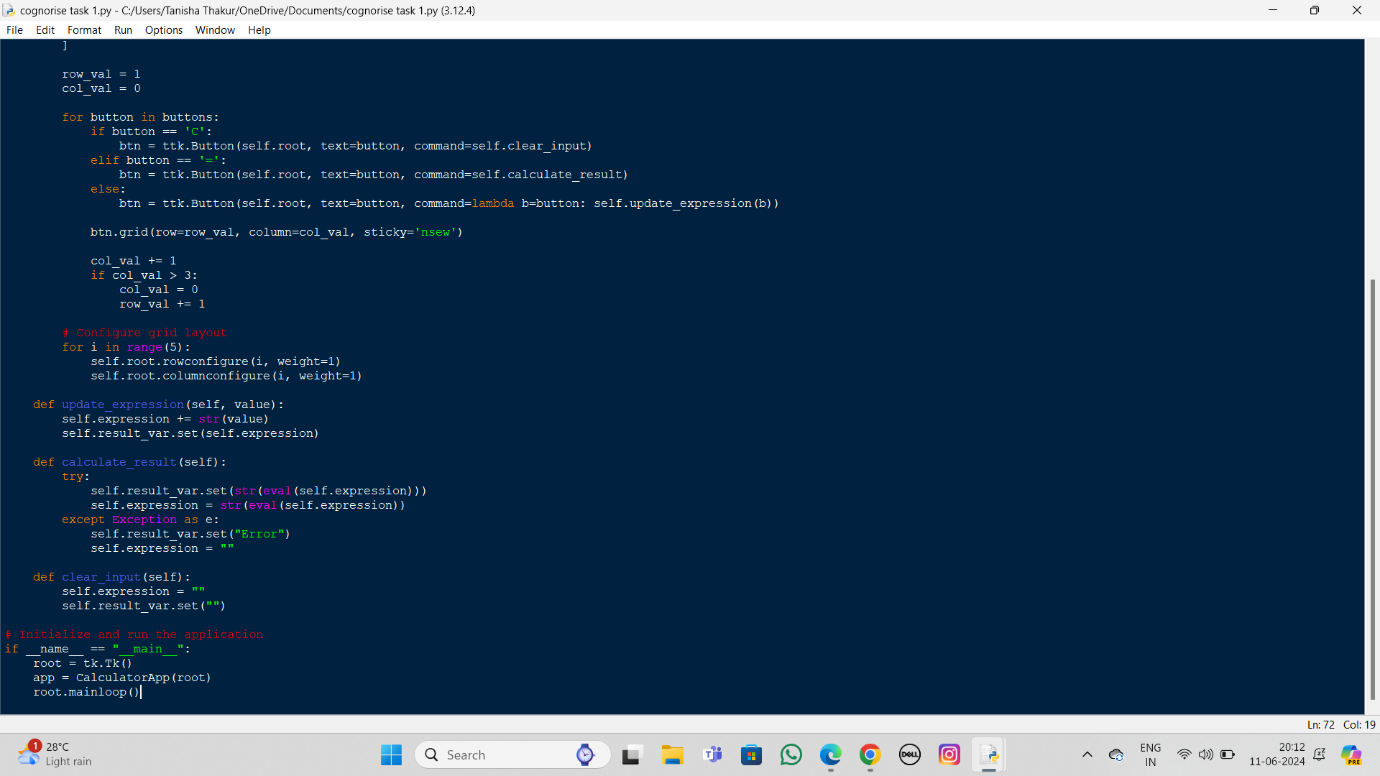
root = tk.Tk()

app = CalculatorApp(root)

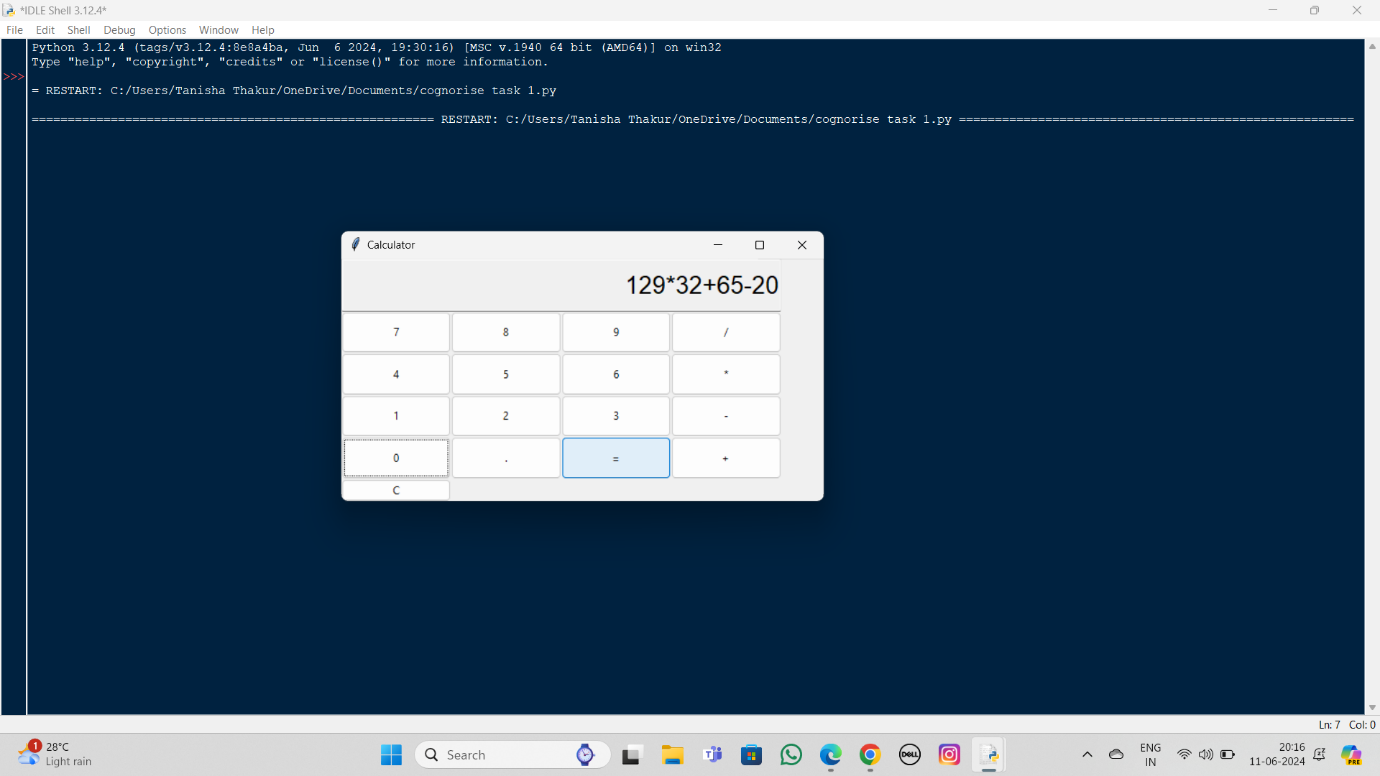
root.mainloop()

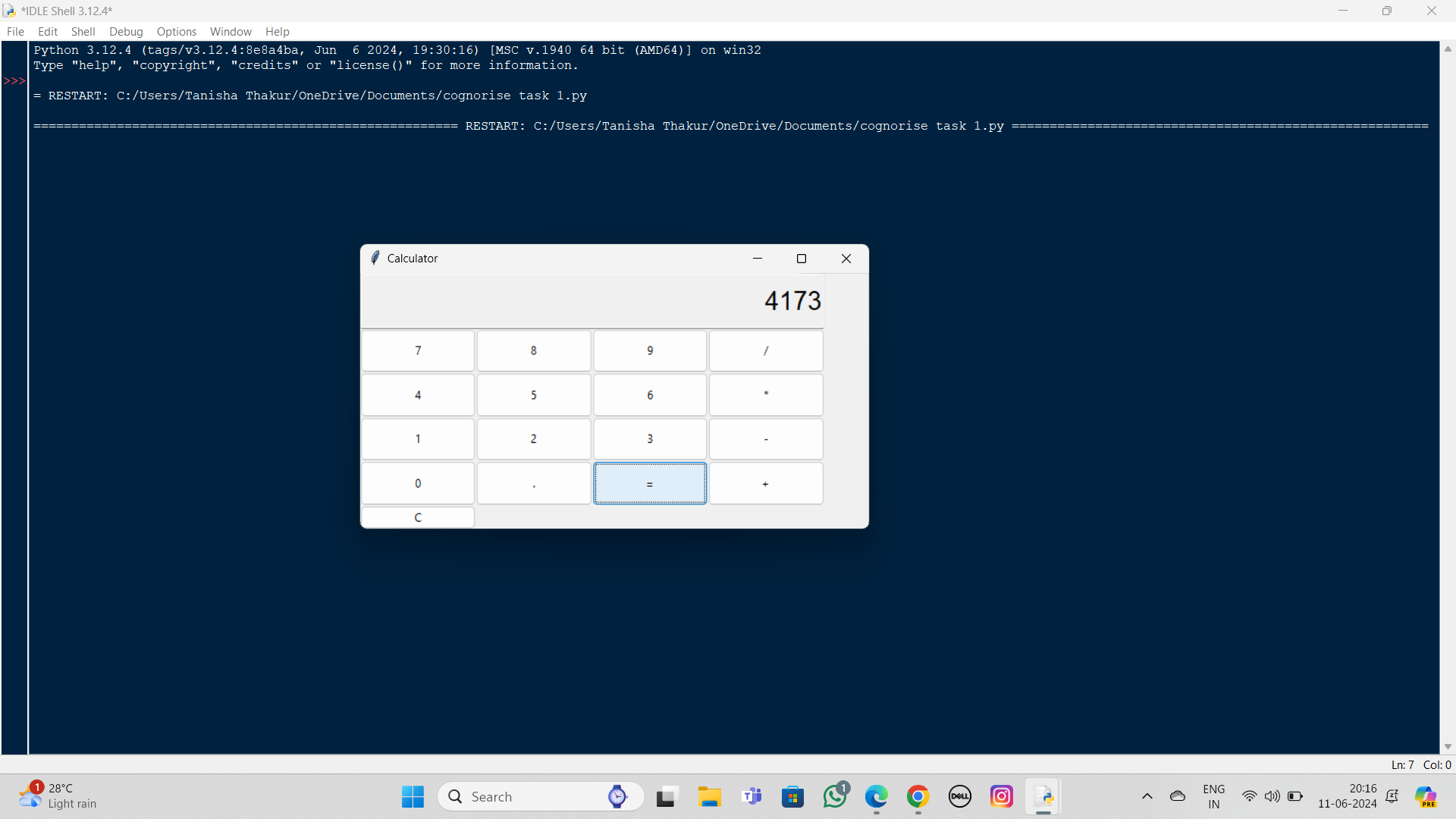








Output:



**Explanation of the Code**

1. **Imports and Class Definition**:
   * Import tkinter and ttk from the tkinter library.
   * Define the CalculatorApp class to encapsulate the calculator's functionality.
2. **Constructor (\_\_init\_\_ method)**:
   * Initialize the root window and set the title.
   * Initialize the expression string and result variable.
   * Call create\_widgets to set up the GUI components.
3. **Creating Widgets**:
   * Create the display entry to show the current expression/result.
   * Create buttons for digits, operators, and actions (clear, equals).
   * Use a loop to place buttons in a grid layout.
4. **Button Actions**:
   * update\_expression: Appends the pressed button's value to the current expression.
   * calculate\_result: Evaluates the current expression and displays the result.
   * clear\_input: Clears the current expression and the display.
5. **Grid Layout Configuration**:
   * Configure the grid layout to make the buttons expand to fill the available space.



**TASK – 2**

**TO – DO LIST**

Interface Design:

Design a basic user interface (UI) with a text input field for adding tasks and a list view to display them.

Task Addition:

Implement functionality to add tasks entered by the user to the list view.

Task Removal:

Allow users to remove tasks from the list by tapping on them.

Basi Functionality:

Ensure the app can handle adding and removing tasks without crashing.

import tkinter as tk

from tkinter import messagebox

class ToDoApp:

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

self.root = root

self.root.title("To-Do List App")

self.task\_entry = tk.Entry(root, width=50)

self.task\_entry.pack(pady=10)



self.add\_button = tk.Button(root, text="Add Task", command=self.add\_task)

self.add\_button.pack(pady=5)

self.task\_listbox = tk.Listbox(root, width=50, height=15)

self.task\_listbox.pack(pady=10)

self.task\_listbox.bind('<Double-1>', self.remove\_task)

def add\_task(self):

task = self.task\_entry.get()

if task:

self.task\_listbox.insert(tk.END, task)

self.task\_entry.delete(0, tk.END)

else:

messagebox.showwarning("Warning", "You must enter a task.")

def remove\_task(self, event):

selected\_task\_index = self.task\_listbox.curselection()

if selected\_task\_index:

self.task\_listbox.delete(selected\_task\_index)

else:

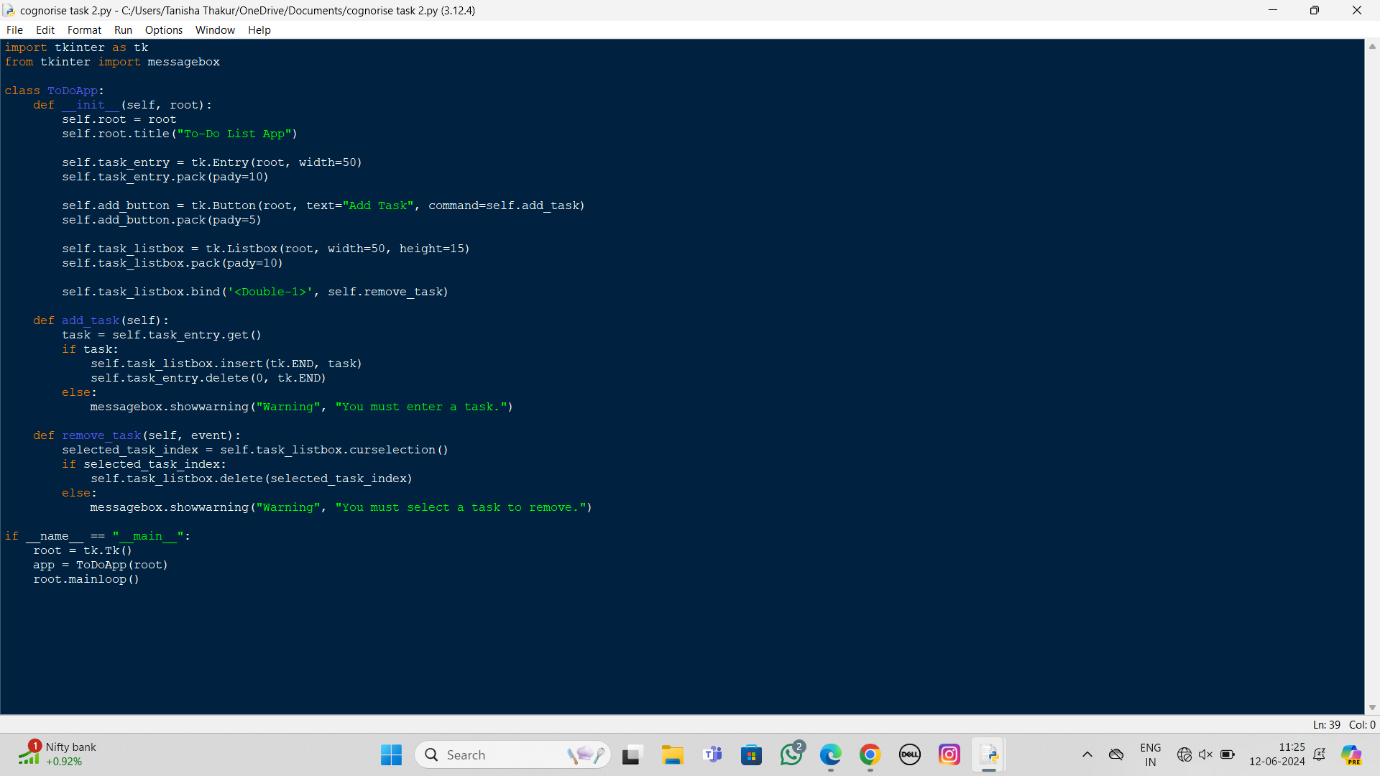
messagebox.showwarning("Warning", "You must select a task to remove.")

if \_\_name\_\_ == "\_\_main\_\_":

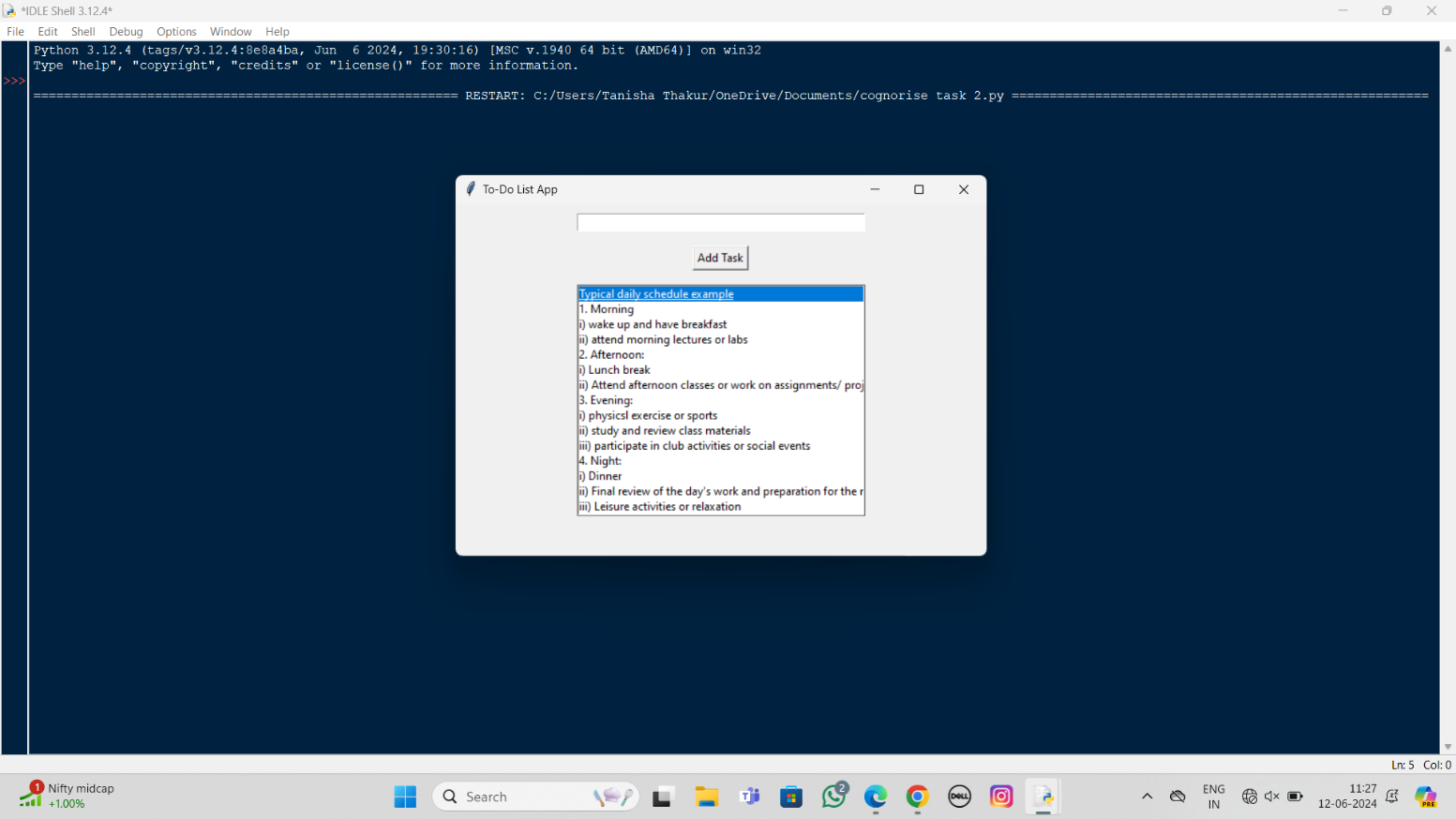
root = tk.Tk()

 app = ToDoApp(root)

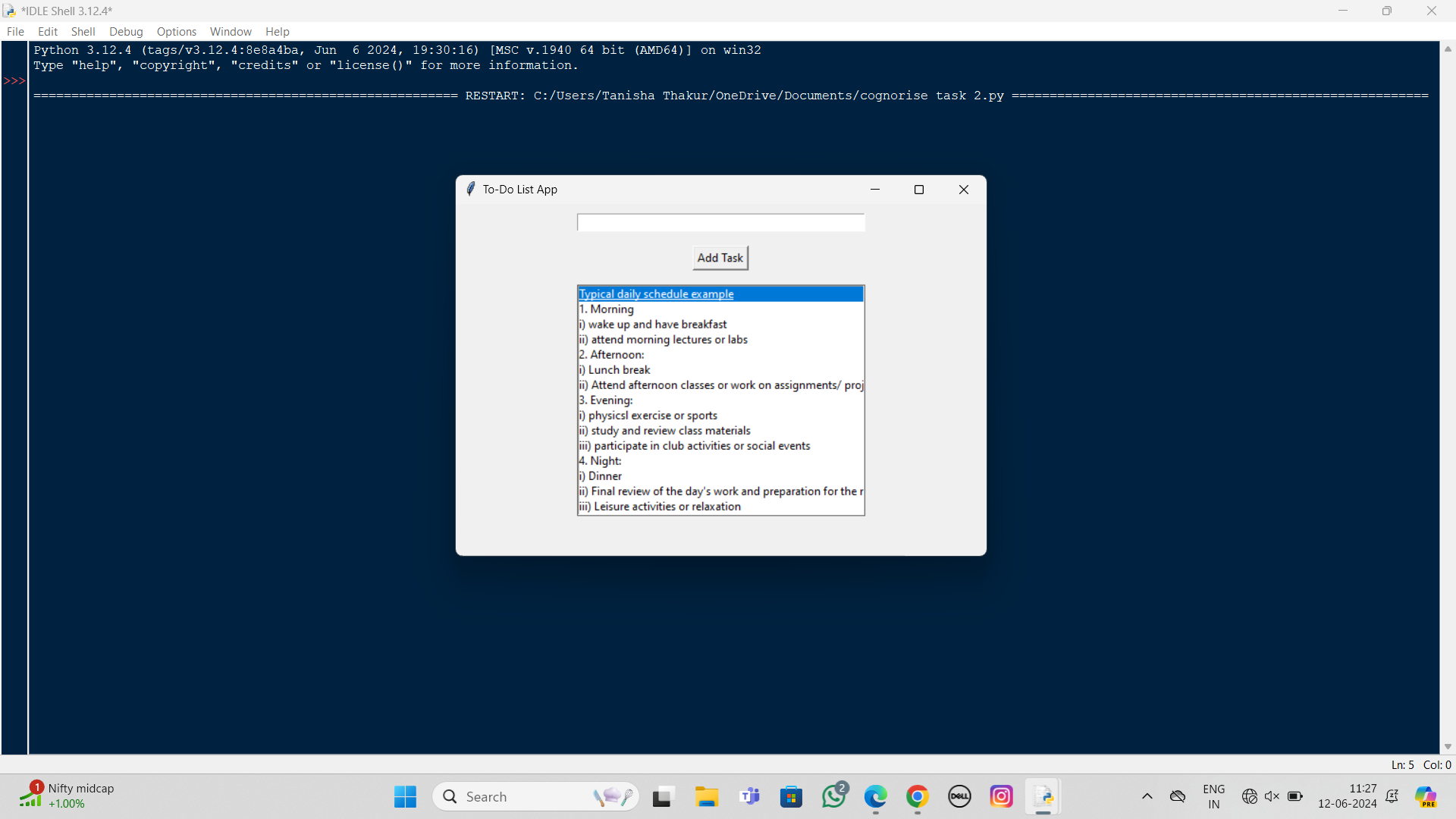
root.mainloop()

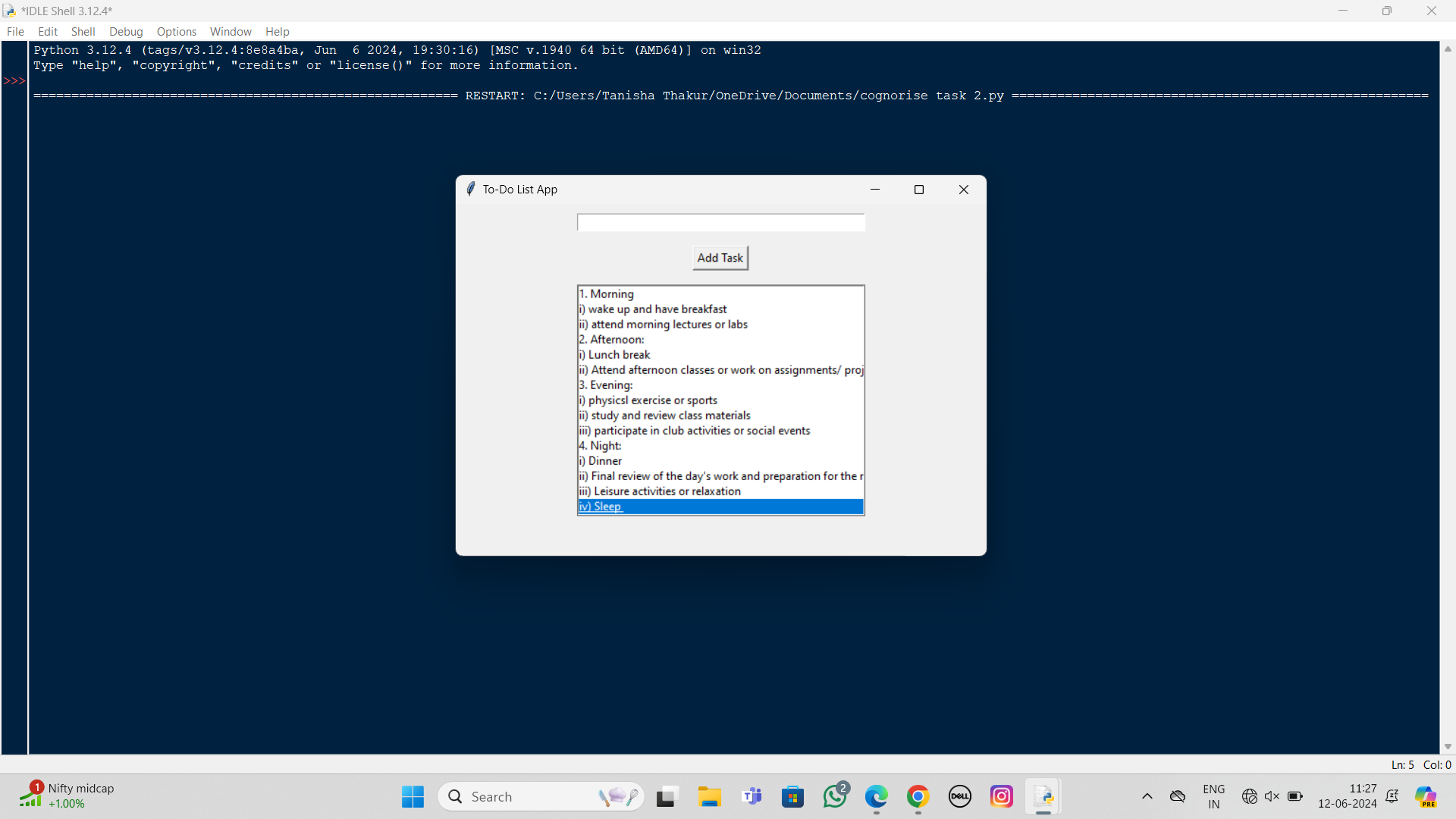


Output:









**Explanation:**

1. **Interface Design**:
   * We create the main application window using tk.Tk().
   * Entry widget is used for the text input field to enter new tasks.
   * Button widget is used to trigger the addition of tasks.
   * Listbox widget is used to display the list of tasks.



1. **Task Addition**:
   * The add\_task method retrieves the text from the Entry widget, adds it to the Listbox, and then clears the input field.
2. **Task Removal**:
   * The Listbox is configured to bind a double-click event to the remove\_task method. This method removes the selected task from the Listbox.
3. **Basic Functionality**:
   * Basic error handling is added using messagebox.showwarning to ensure tasks are not added if the input field is empty, and tasks can only be removed if one is selected.

**TASK – 3**

**ALARM CLOCK APP**

Build an alarm clock app that allows users to set and manage alarms

* Home Screen: Display current time and date
* Option to set a new alarm
* Alarm settings: Time picker to set the alarm time
* Option to choose the alarm tone.
* Alarm Management: List of all set alarms with on/off toggle.
* Snooze and Dismiss: Snooze or dismiss the alarm when it rings
* User interface: User-friendly and visually appealing design.



import tkinter as tk

from tkinter import ttk, messagebox

from datetime import datetime

import time

import threading

import winsound

class AlarmClockApp:

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

self.root = root

self.root.title("Alarm Clock App")

self.current\_time\_label = tk.Label(root, font=("Helvetica", 48))

self.current\_time\_label.pack(pady=20)

self.current\_date\_label = tk.Label(root, font=("Helvetica", 24))

self.current\_date\_label.pack(pady=10)

self.update\_time()

self.set\_alarm\_button = tk.Button(root, text="Set New Alarm", command=self.open\_set\_alarm\_window)

self.set\_alarm\_button.pack(pady=10)

self.alarm\_frame = tk.Frame(root)

 self.alarm\_frame.pack(pady=10)

self.alarms = []

def update\_time(self):

now = datetime.now()

current\_time = now.strftime("%H:%M:%S")

current\_date = now.strftime("%A, %B %d, %Y")

self.current\_time\_label.config(text=current\_time)

self.current\_date\_label.config(text=current\_date)

self.root.after(1000, self.update\_time)

def open\_set\_alarm\_window(self):

self.set\_alarm\_window = tk.Toplevel(self.root)

self.set\_alarm\_window.title("Set Alarm")

self.time\_picker = ttk.Combobox(self.set\_alarm\_window, values=[f"{i:02d}:{j:02d}" for i in range(24) for j in range(60)])

self.time\_picker.pack(pady=10)

self.alarm\_tone = ttk.Combobox(self.set\_alarm\_window, values=["Tone 1", "Tone 2", "Tone 3"])

self.alarm\_tone.pack(pady=10)

self.save\_button = tk.Button(self.set\_alarm\_window, text="Save Alarm", command=self.save\_alarm)

self.save\_button.pack(pady=10)

 def save\_alarm(self):

alarm\_time = self.time\_picker.get()

alarm\_tone = self.alarm\_tone.get()

if alarm\_time and alarm\_tone:

alarm = {"time": alarm\_time, "tone": alarm\_tone, "active": True}

self.alarms.append(alarm)

self.update\_alarm\_list()

self.set\_alarm\_window.destroy()

else:

messagebox.showwarning("Warning", "Please set both time and tone.")

def update\_alarm\_list(self):

for widget in self.alarm\_frame.winfo\_children():

widget.destroy()

for idx, alarm in enumerate(self.alarms):

alarm\_label = tk.Label(self.alarm\_frame, text=f"{alarm['time']} - {alarm['tone']}")

alarm\_label.grid(row=idx, column=0, padx=10)

toggle\_button = tk.Button(self.alarm\_frame, text="On" if alarm['active'] else "Off",

command=lambda idx=idx: self.toggle\_alarm(idx))

toggle\_button.grid(row=idx, column=1, padx=10)

delete\_button = tk.Button(self.alarm\_frame, text="Delete", command=lambda idx=idx: self.delete\_alarm(idx))

delete\_button.grid(row=idx, column=2, padx=10)



def toggle\_alarm(self, idx):

self.alarms[idx]['active'] = not self.alarms[idx]['active']

self.update\_alarm\_list()

def delete\_alarm(self, idx):

del self.alarms[idx]

self.update\_alarm\_list()

def check\_alarms(self):

while True:

now = datetime.now().strftime("%H:%M")

for alarm in self.alarms:

if alarm['active'] and alarm['time'] == now:

self.trigger\_alarm(alarm)

time.sleep(1)

def trigger\_alarm(self, alarm):

for \_ in range(5):

winsound.Beep(1000, 1000)

snooze\_button = tk.Button(self.root, text="Snooze", command=self.snooze\_alarm)

snooze\_button.pack(pady=10)

dismiss\_button = tk.Button(self.root, text="Dismiss", command=snooze\_button.pack\_forget)

dismiss\_button.pack(pady=10)

def snooze\_alarm(self):

 # Implement snooze functionality

pass

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

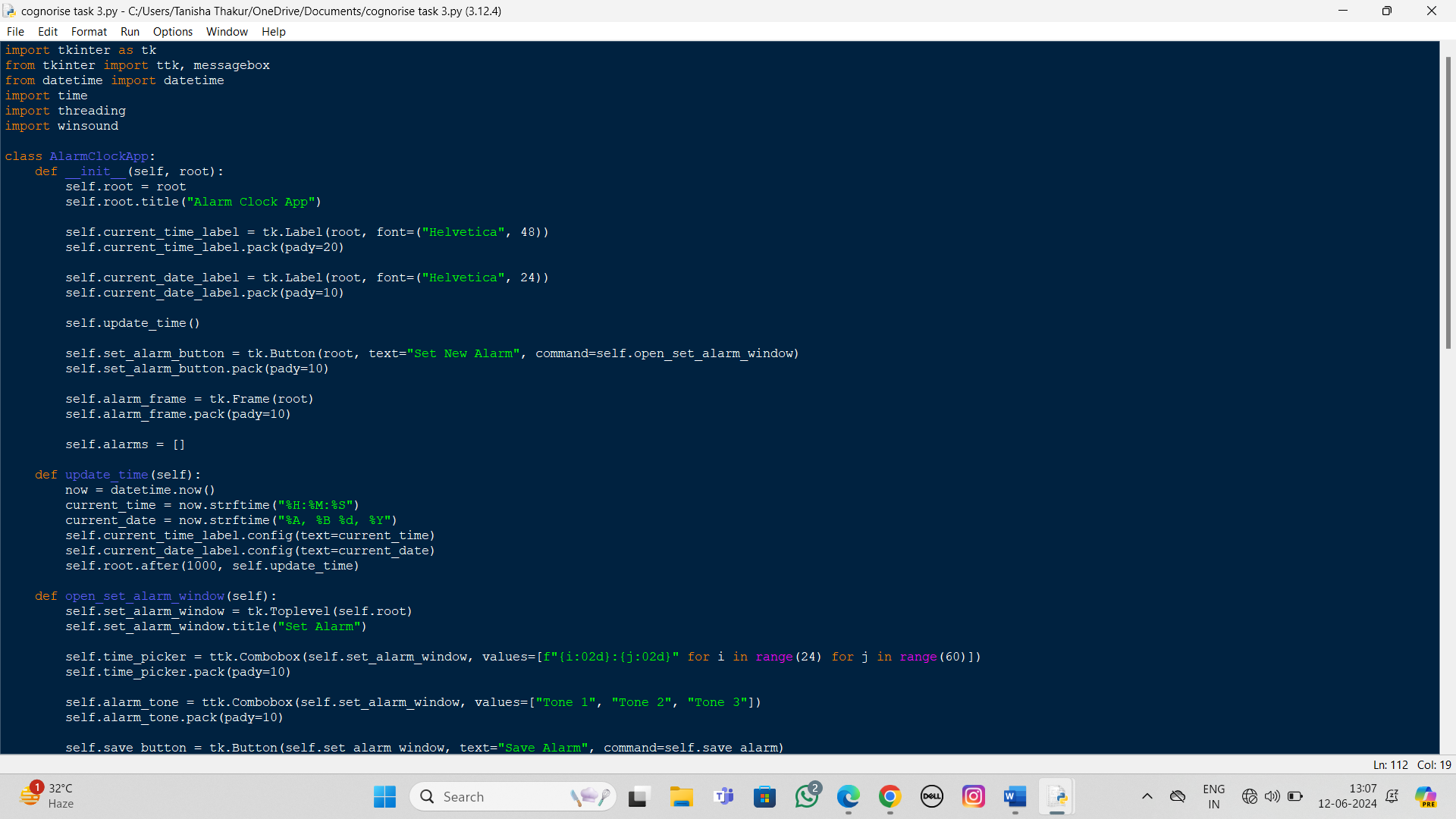
app = AlarmClockApp(root)

alarm\_thread = threading.Thread(target=app.check\_alarms)

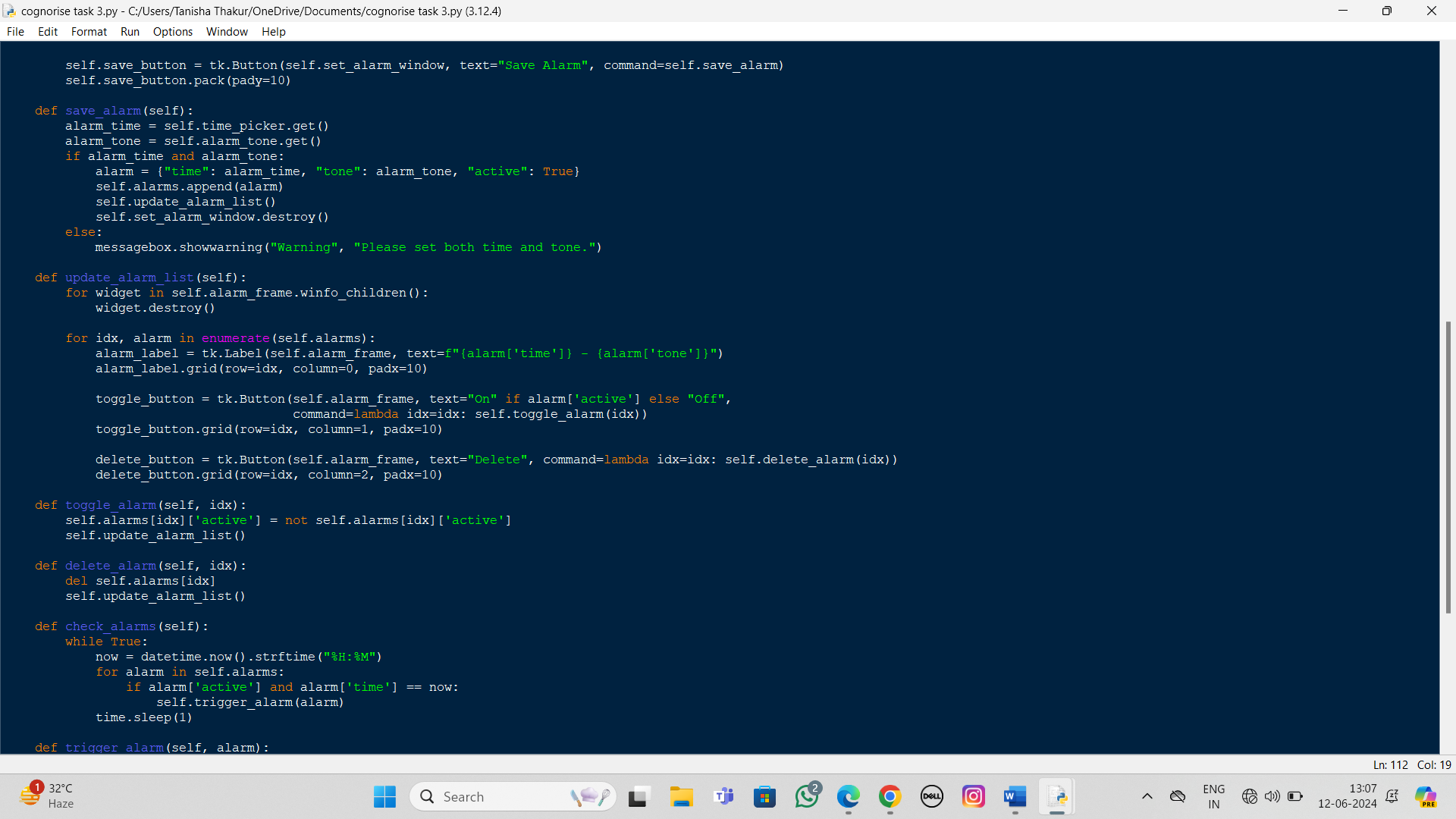
alarm\_thread.daemon = True

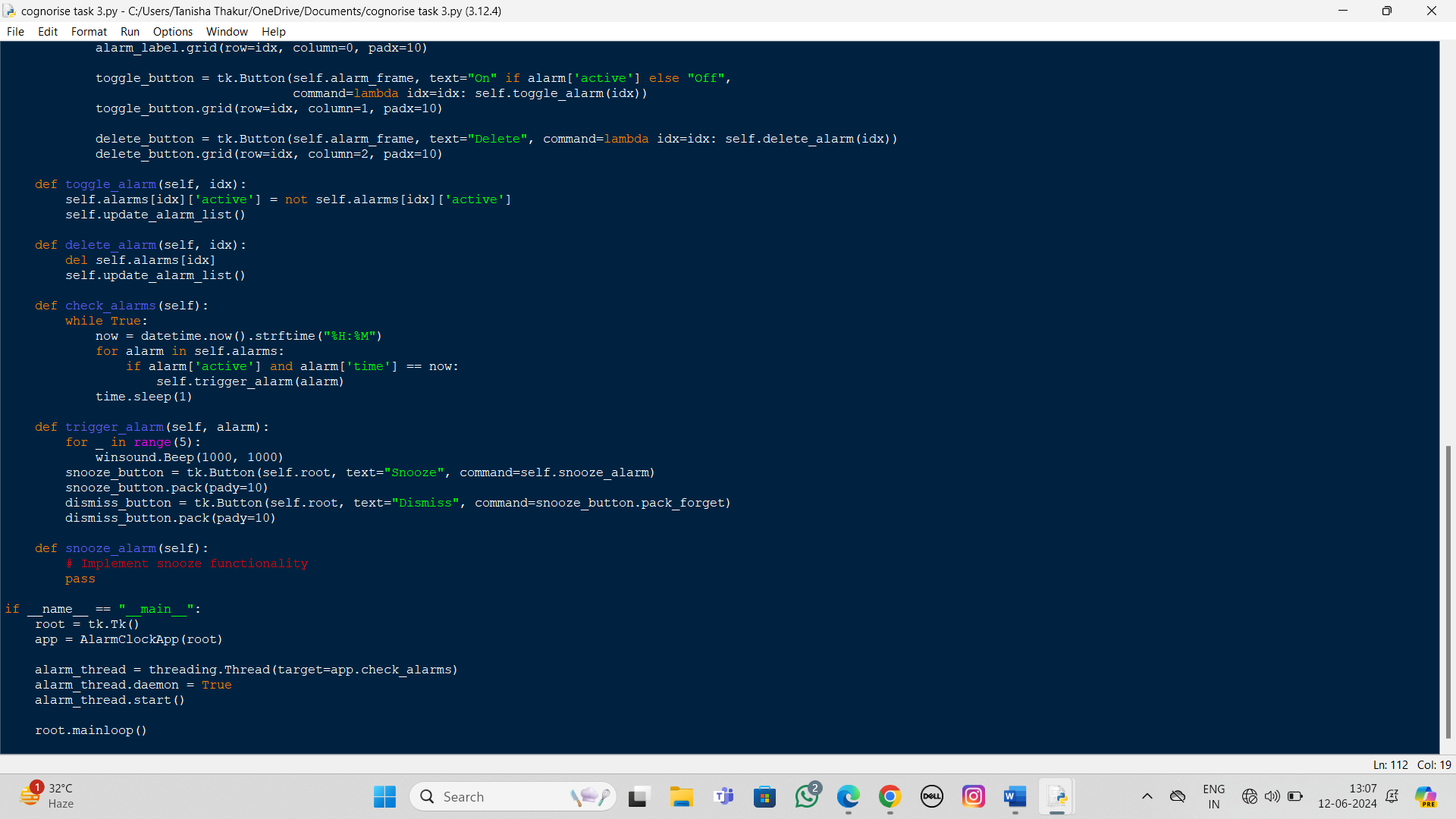
alarm\_thread.start()

root.mainloop()



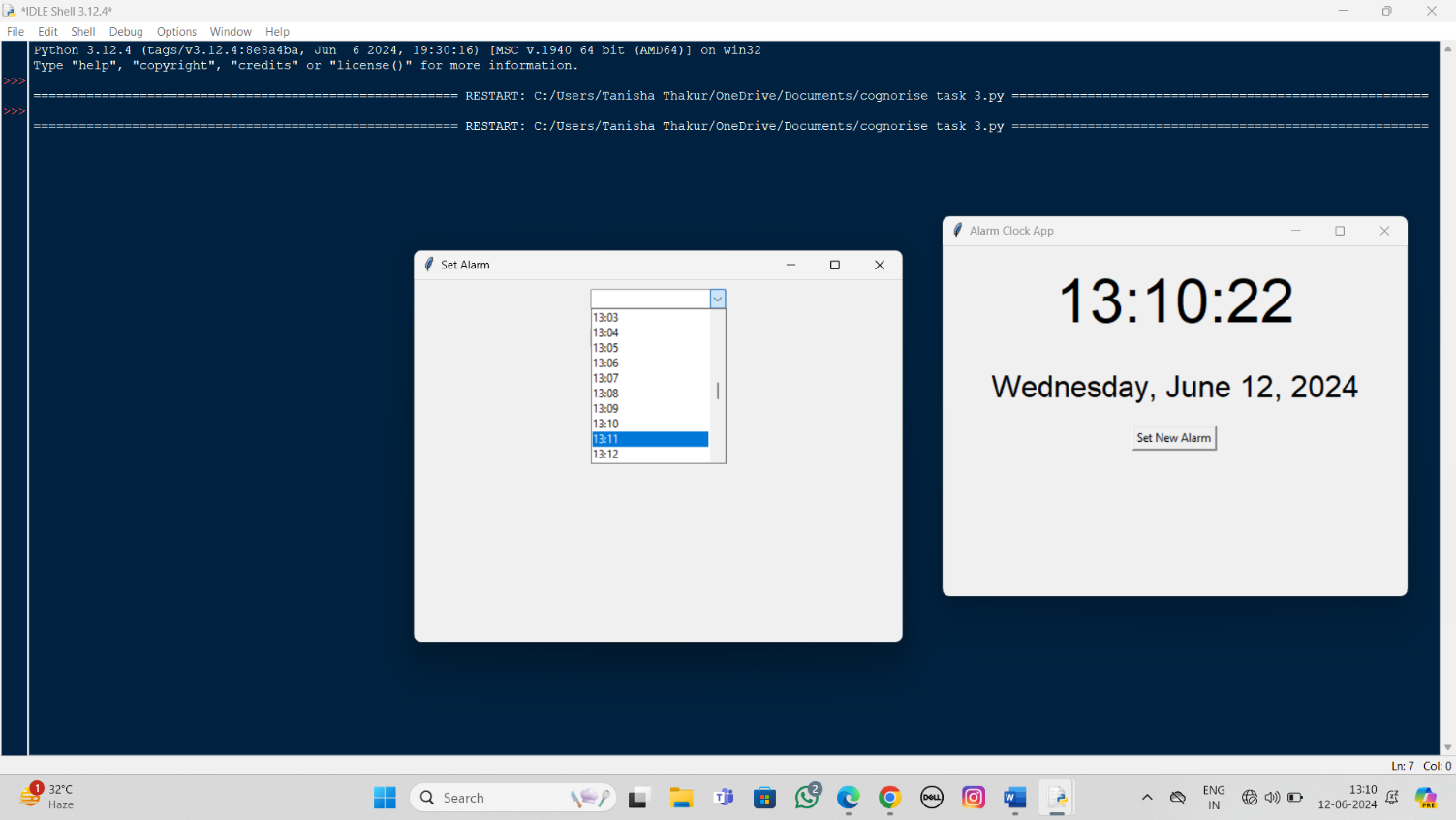


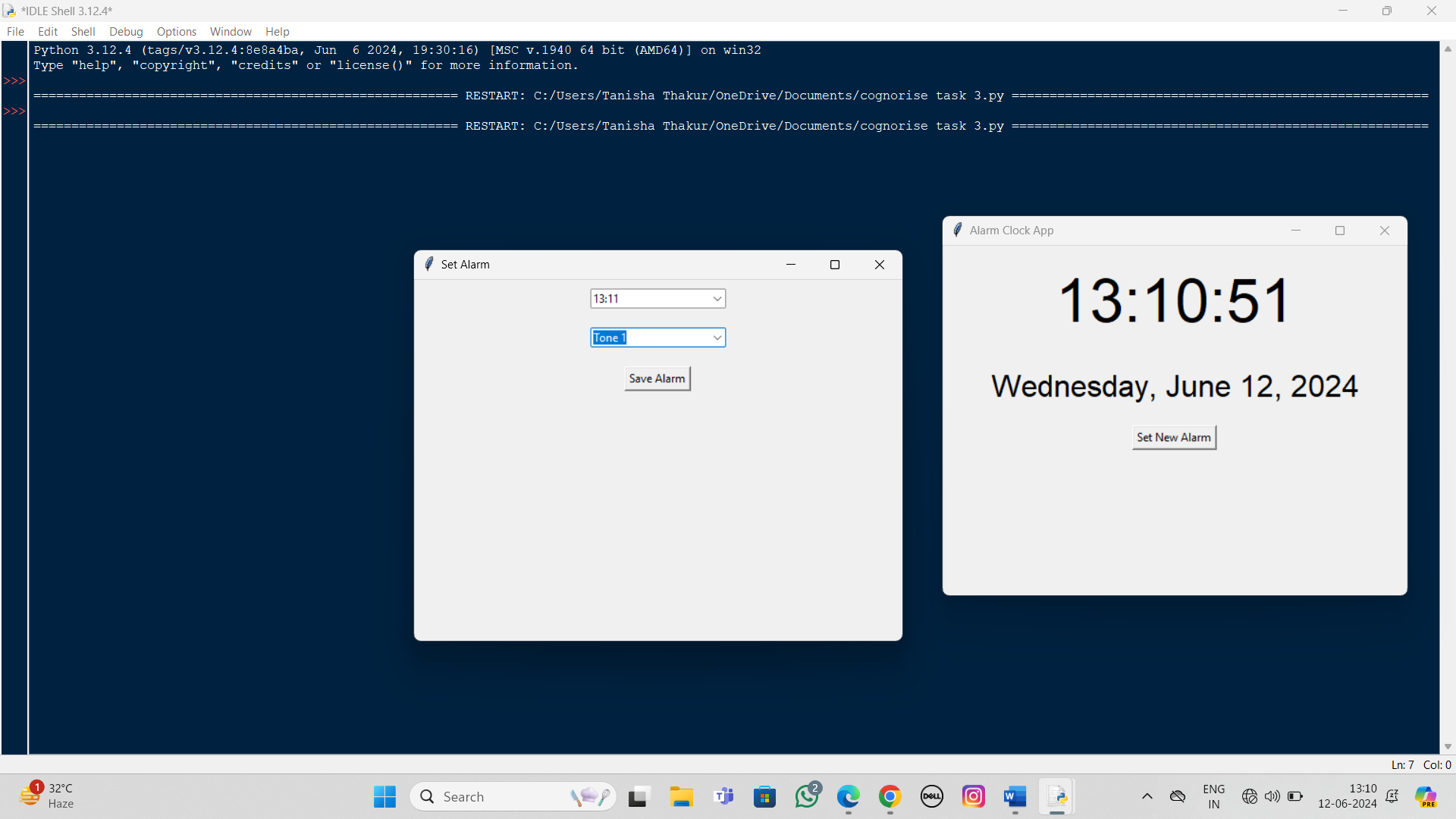




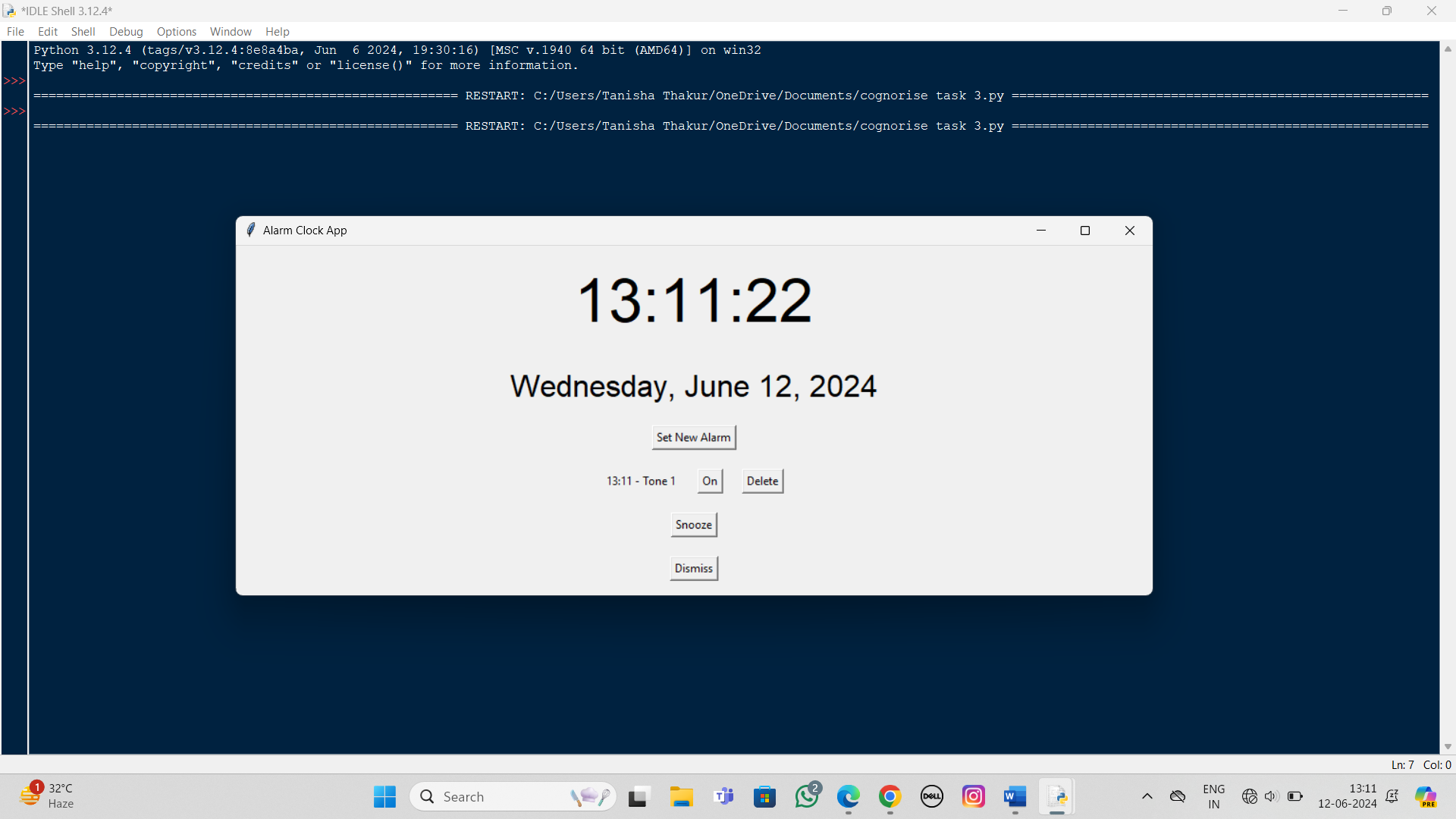


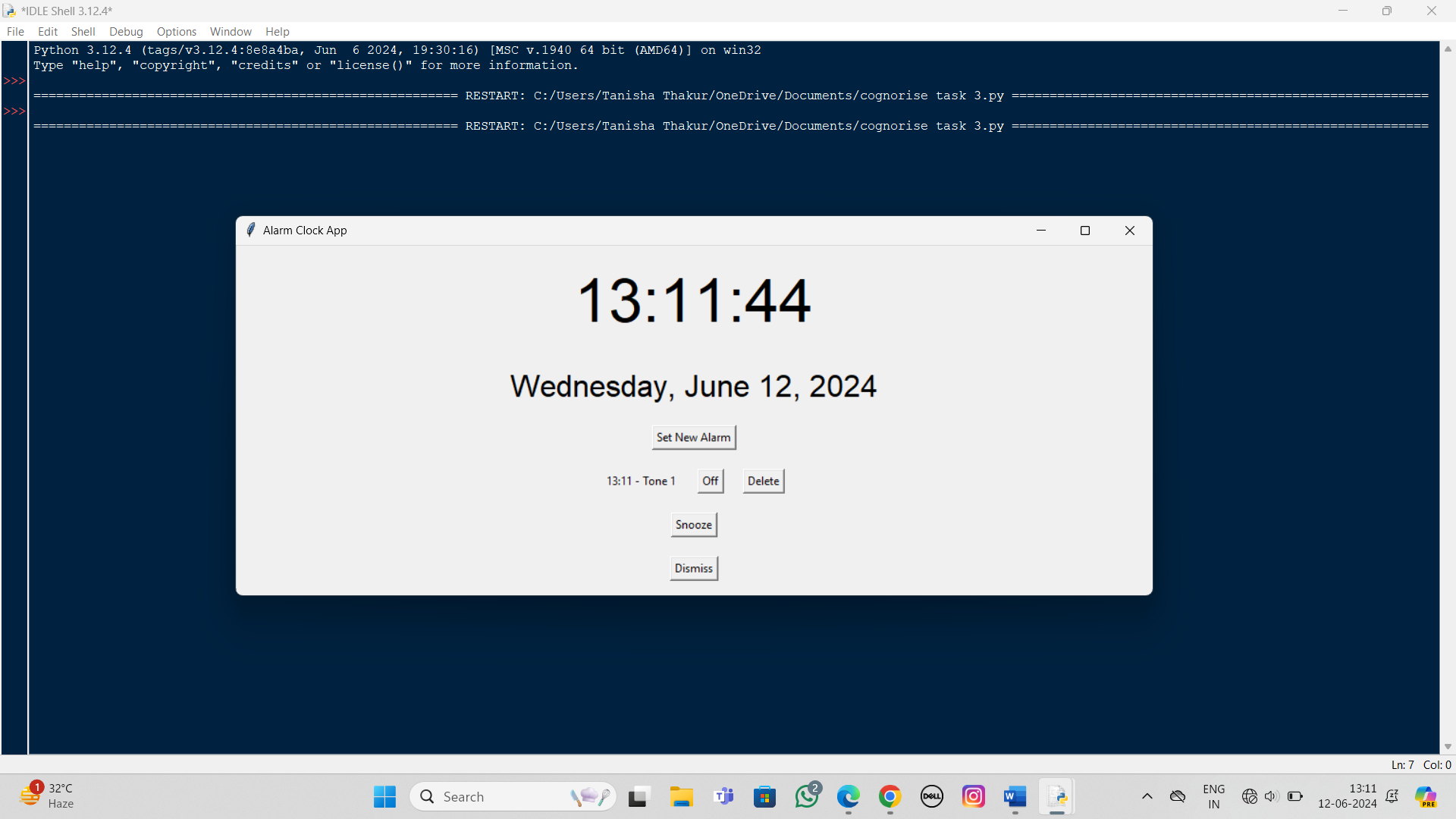
Output:











**Explanation**

1. **Home Screen**:
   * update\_time method keeps updating the current time and date every second.
   * Button to set a new alarm opens a new window to set the alarm.
2. **Set a New Alarm**:
   * open\_set\_alarm\_window opens a new window to set alarm time and tone.
   * save\_alarm saves the new alarm and updates the alarm list.
3. **Alarm Management**:



update\_alarm\_list updates the list of alarms with toggle and delete options.

* + toggle\_alarm toggles the alarm on/off.
  + delete\_alarm deletes the selected alarm.

1. **Check Alarms**:
   * check\_alarms runs in a separate thread to continuously check the current time against the set alarms.
   * trigger\_alarm is called when an alarm goes off, and it plays a beep sound.
2. **Snooze and Dismiss**:
   * trigger\_alarm includes placeholder buttons for snooze and dismiss.

**TASK – 4**

**BMI CALCULATOR**

BMI Calculator App: Develop a user-centric Body Mass Index (BMI) calculator app that facilitates input of weight and height. Upon user input, the app swiftly computes the BMI by dividing weight (in kilograms) by the square of height (in meters). Notably, the app's distinctive feature is its capability to classify the BMI result.

Based on the calculated BMI, the app classifies users into distinct categories such as underweight, normal weight, overweight, or other relevant classifications. This classification mechanism delivers personalized insights into one's weight status. The app's interface will be intuitive and minimalist, prioritizing seamless user interaction. By combining calculation and classification, the app empowers users to make informed health-related decisions while fostering a greater understanding of their own well-being.



import tkinter as tk

from tkinter import messagebox

class BMICalculatorApp:

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

self.root = root

self.root.title("BMI Calculator")

self.weight\_label = tk.Label(root, text="Weight (kg):")

self.weight\_label.pack(pady=5)

self.weight\_entry = tk.Entry(root)

self.weight\_entry.pack(pady=5)

self.height\_label = tk.Label(root, text="Height (m):")

self.height\_label.pack(pady=5)

self.height\_entry = tk.Entry(root)

self.height\_entry.pack(pady=5)

self.calculate\_button = tk.Button(root, text="Calculate BMI", command=self.calculate\_bmi)

self.calculate\_button.pack(pady=20)

self.result\_label = tk.Label(root, text="", font=("Helvetica", 14))

self.result\_label.pack(pady=10)

 def calculate\_bmi(self):

try:

weight = float(self.weight\_entry.get())

height = float(self.height\_entry.get())

if height <= 0 or weight <= 0:

messagebox.showwarning("Warning", "Please enter positive values for weight and height.")

return

bmi = weight / (height \*\* 2)

classification = self.classify\_bmi(bmi)

self.result\_label.config(text=f"BMI: {bmi:.2f} ({classification})")

except ValueError:

messagebox.showwarning("Warning", "Please enter valid numbers for weight and height.")

def classify\_bmi(self, bmi):

if bmi < 18.5:

return "Underweight"

elif 18.5 <= bmi < 24.9:

return "Normal weight"

elif 25 <= bmi < 29.9:

return "Overweight"

else:

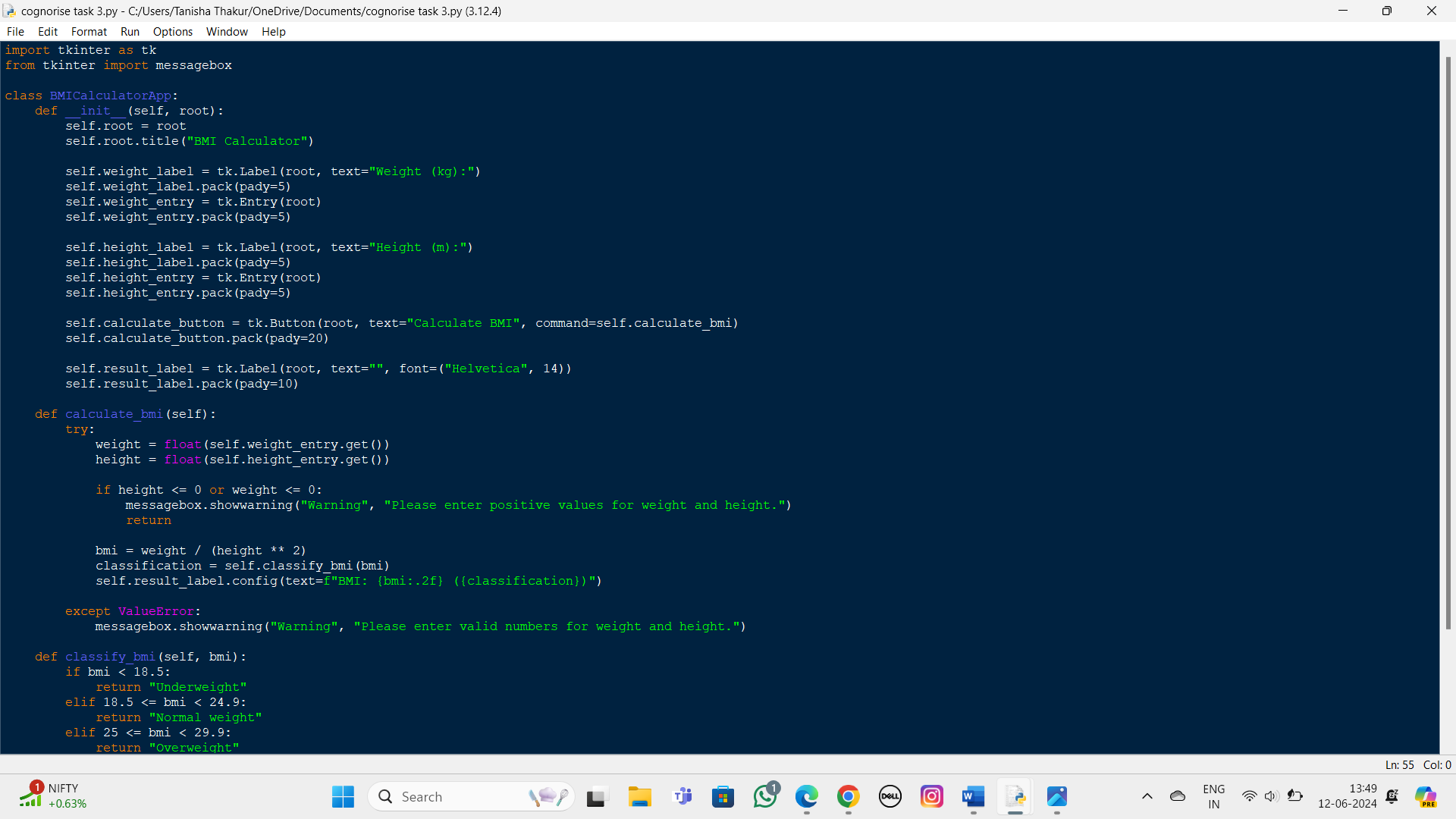
return "Obesity"

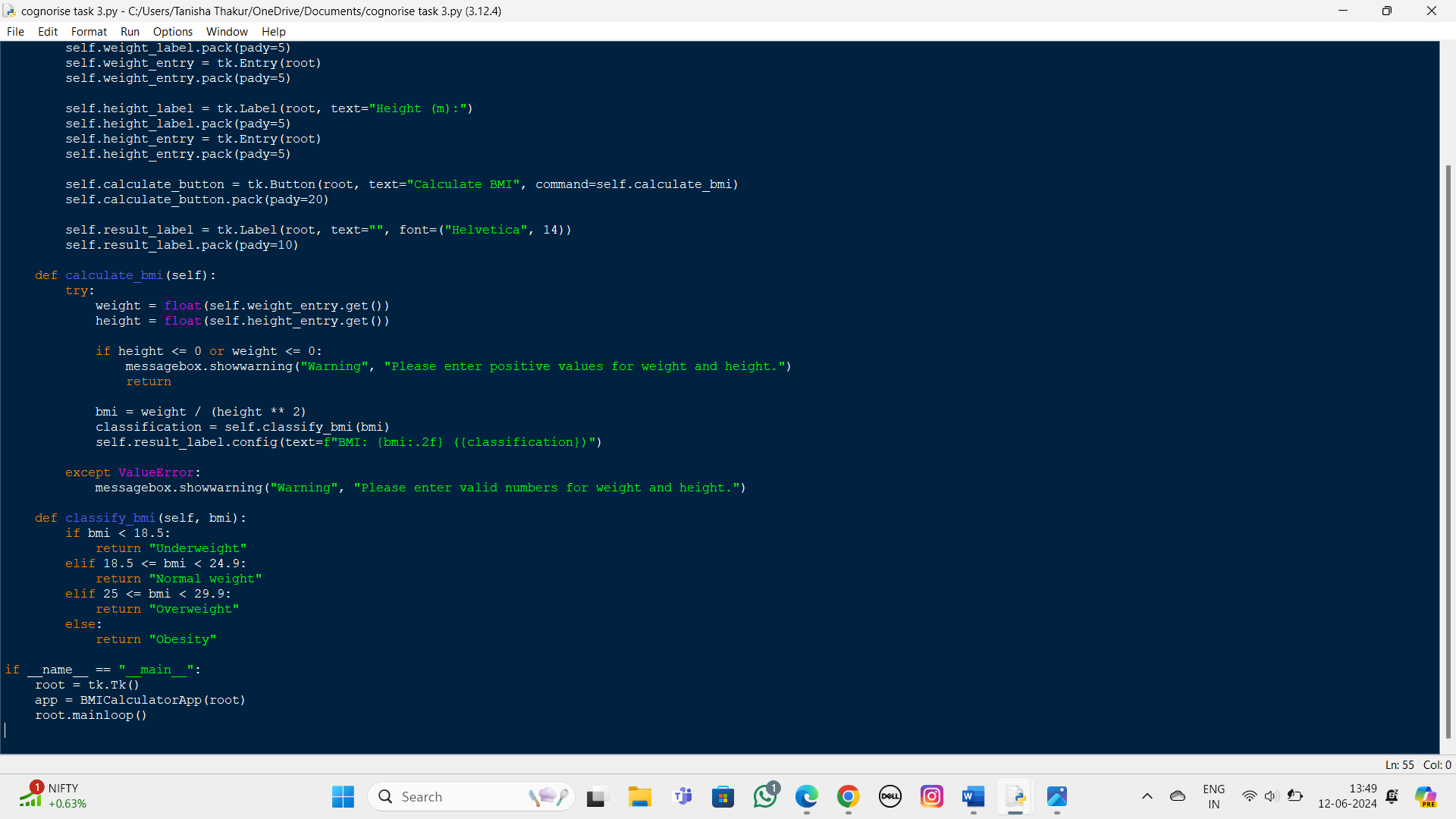
if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = BMICalculatorApp(root)

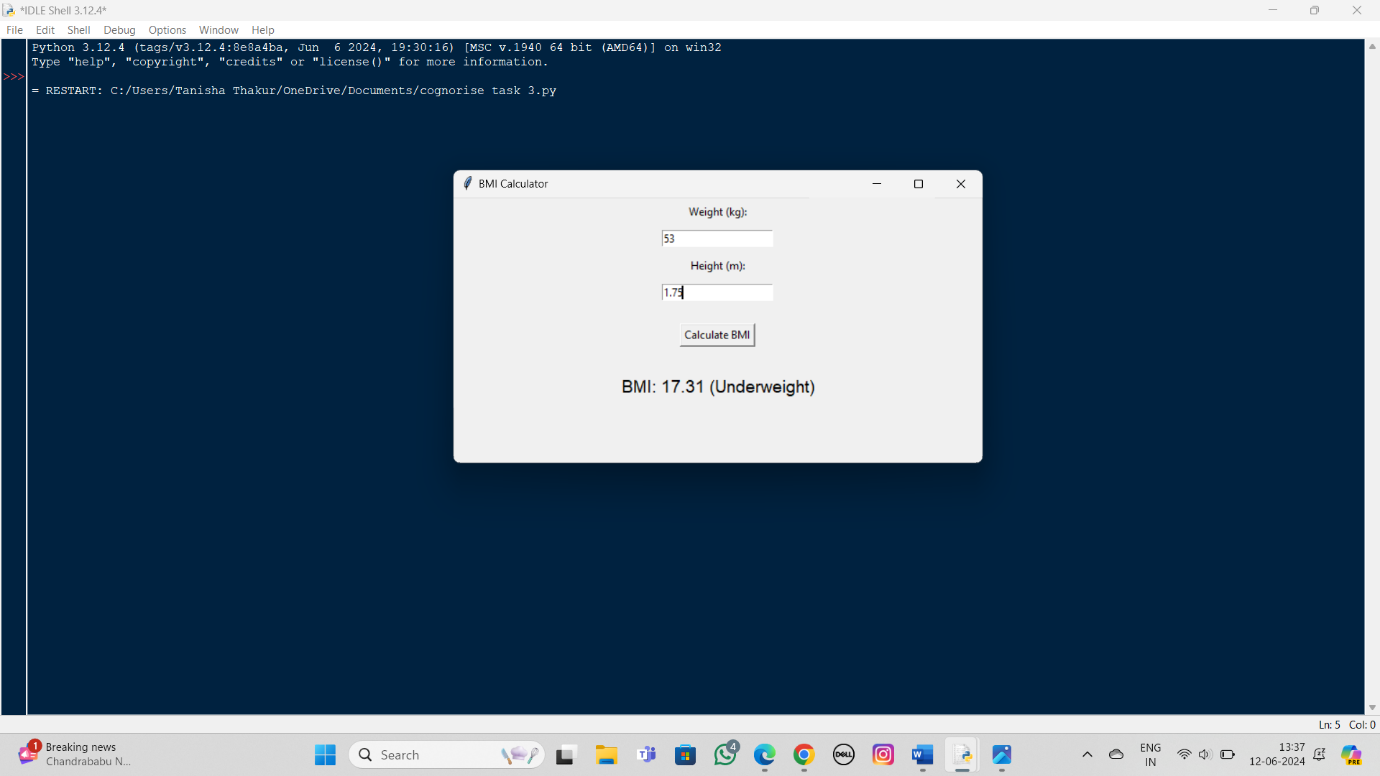
root.mainloop()

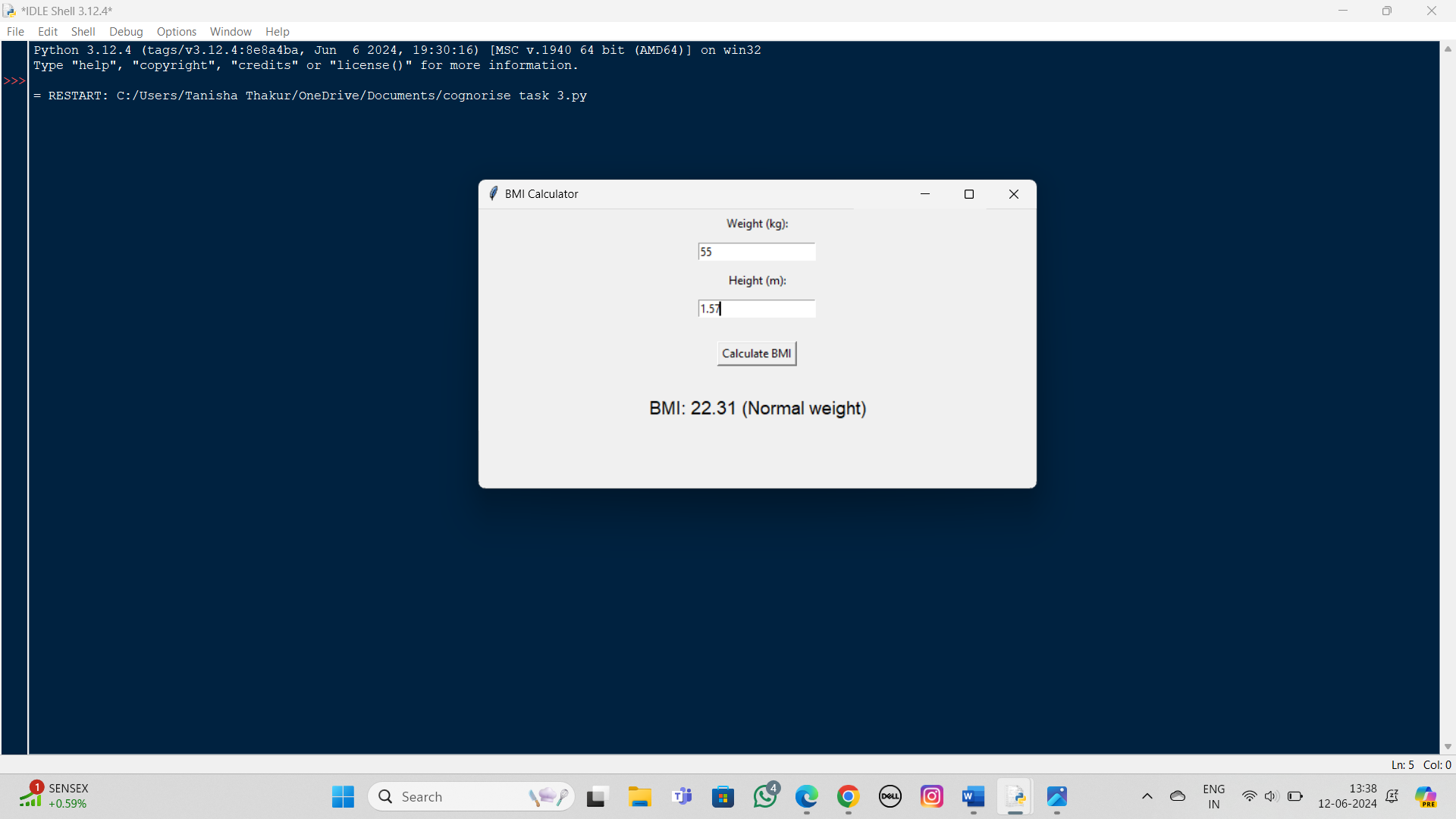




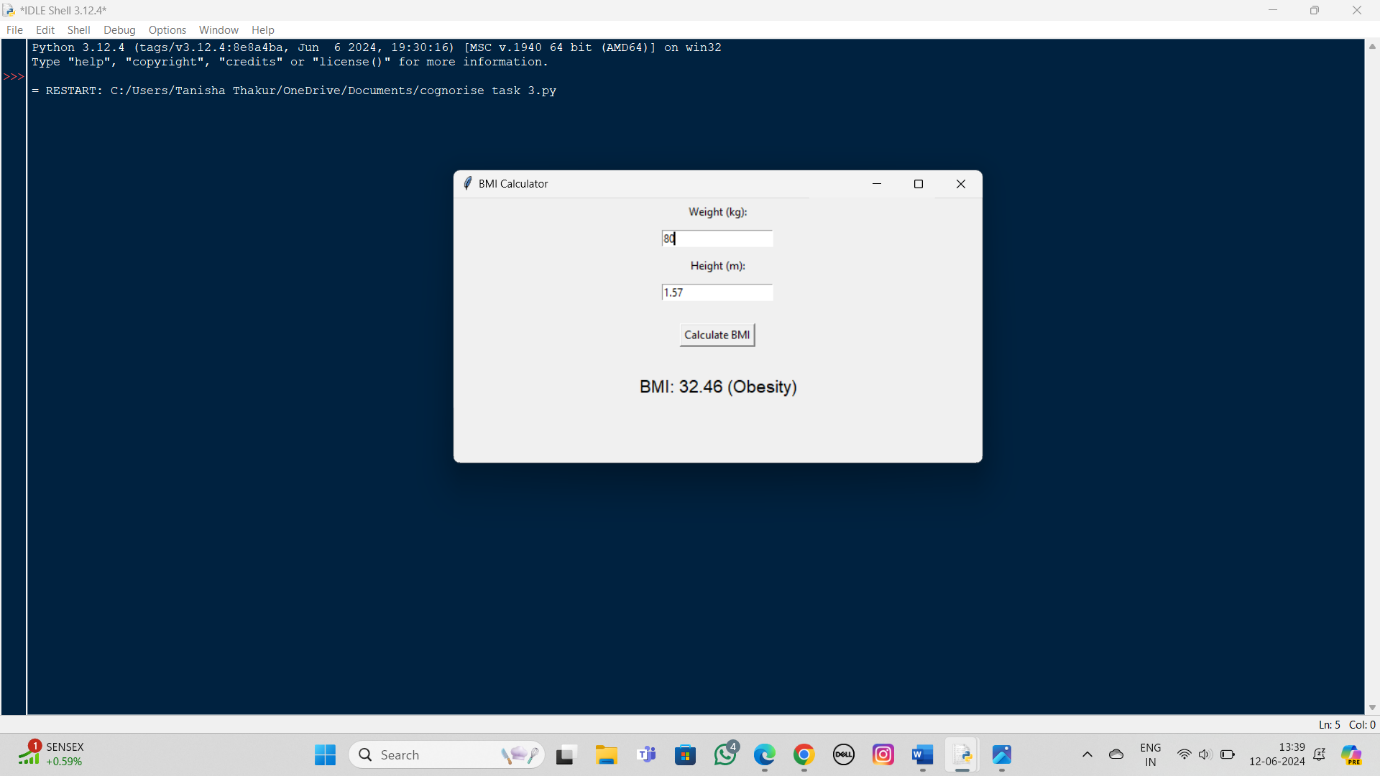


Output:









**Explanation**

1. **User Interface**:
   * The main window is created using tk.Tk().
   * Labels and entry fields are provided for the user to input their weight and height.
   * A button triggers the BMI calculation.
2. **BMI Calculation**:
   * The calculate\_bmi method retrieves the user inputs, converts them to float, and checks for valid positive values.
   * BMI is calculated using the formula: bmi = weight / (height \*\* 2).
3. **BMI Classification**:
   * The classify\_bmi method categorizes the BMI value based on standard classification ranges.
   * The result is displayed with the BMI value and its classification.



**TASK – 5**

**SIMPLE PUZZLE GAME**

Simple Puzzle Game Design: Craft an uncomplicated puzzle game where participants reassemble scrambled puzzle fragments within a 3x3 grid to compose a coherent image. Tailored for beginners, this project offers an accessible introduction to game development. Players engage in a delightful challenge of rearranging jumbled pieces by swapping adjacent elements until the picture is reconstructed correctly.

The game's manageable complexity encourages newcomers to explore the game development process step by step. The 3x3 grid setup serves as an apt foundation for beginners to familiarize themselves with basic game mechanics. This undertaking provides a hands-on opportunity to gain practical insights into graphic rendering, user interaction, and problem-solving in the context of game creation.

import random

# Function to create a solved 3x3 board

def create\_solved\_board():

return [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

# Function to scramble the board

def scramble\_board(board):

moves = ['up', 'down', 'left', 'right']

for \_ in range(100):

 move = random.choice(moves)

make\_move(board, move)

return board

# Function to print the board

def print\_board(board):

for row in board:

print(' '.join(str(cell) if cell != 0 else ' ' for cell in row))

print()

# Function to find the position of the empty space (0)

def find\_empty(board):

for i, row in enumerate(board):

for j, cell in enumerate(row):

if cell == 0:

return i, j

return None

# Function to make a move

def make\_move(board, direction):

i, j = find\_empty(board)

if direction == 'up' and i < 2:

board[i][j], board[i+1][j] = board[i+1][j], board[i][j]

elif direction == 'down' and i > 0:

board[i][j], board[i-1][j] = board[i-1][j], board[i][j]

elif direction == 'left' and j < 2:

board[i][j], board[i][j+1] = board[i][j+1], board[i][j]

 elif direction == 'right' and j > 0:

board[i][j], board[i][j-1] = board[i][j-1], board[i][j]

# Function to check if the board is solved

def is\_solved(board):

return board == create\_solved\_board()

def main():

board = create\_solved\_board()

board = scramble\_board(board)

print("Welcome to the 3x3 Puzzle Game!")

print("Arrange the numbers to form the original image.")

print("Use 'up', 'down', 'left', 'right' to move the empty space.\n")

while not is\_solved(board):

print\_board(board)

move = input("Enter your move (up, down, left, right): ").strip().lower()

if move in ['up', 'down', 'left', 'right']:

make\_move(board, move)

else:

print("Invalid move! Please enter one of 'up', 'down', 'left', 'right'.\n")

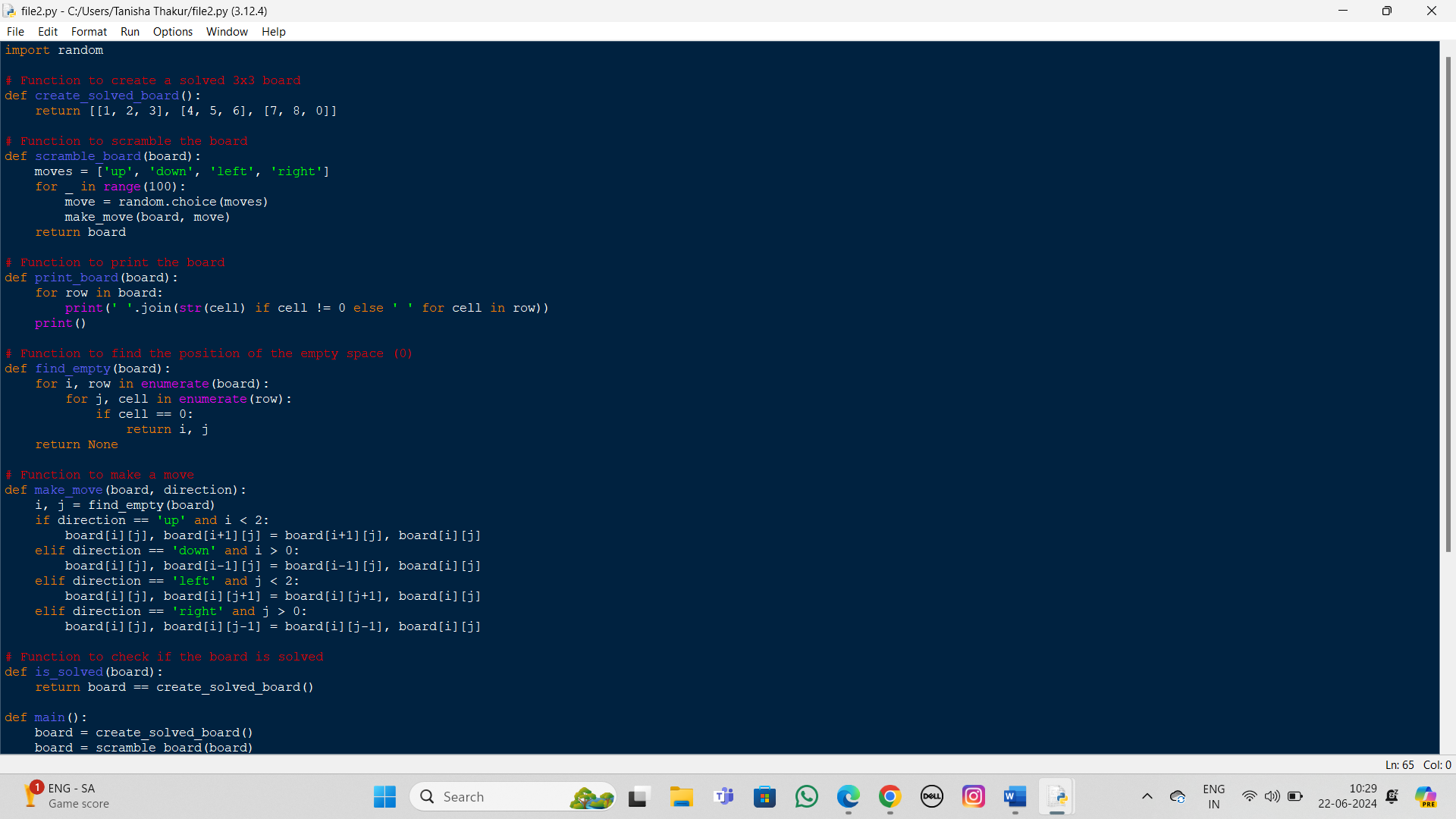
print("Congratulations! You've solved the puzzle!")

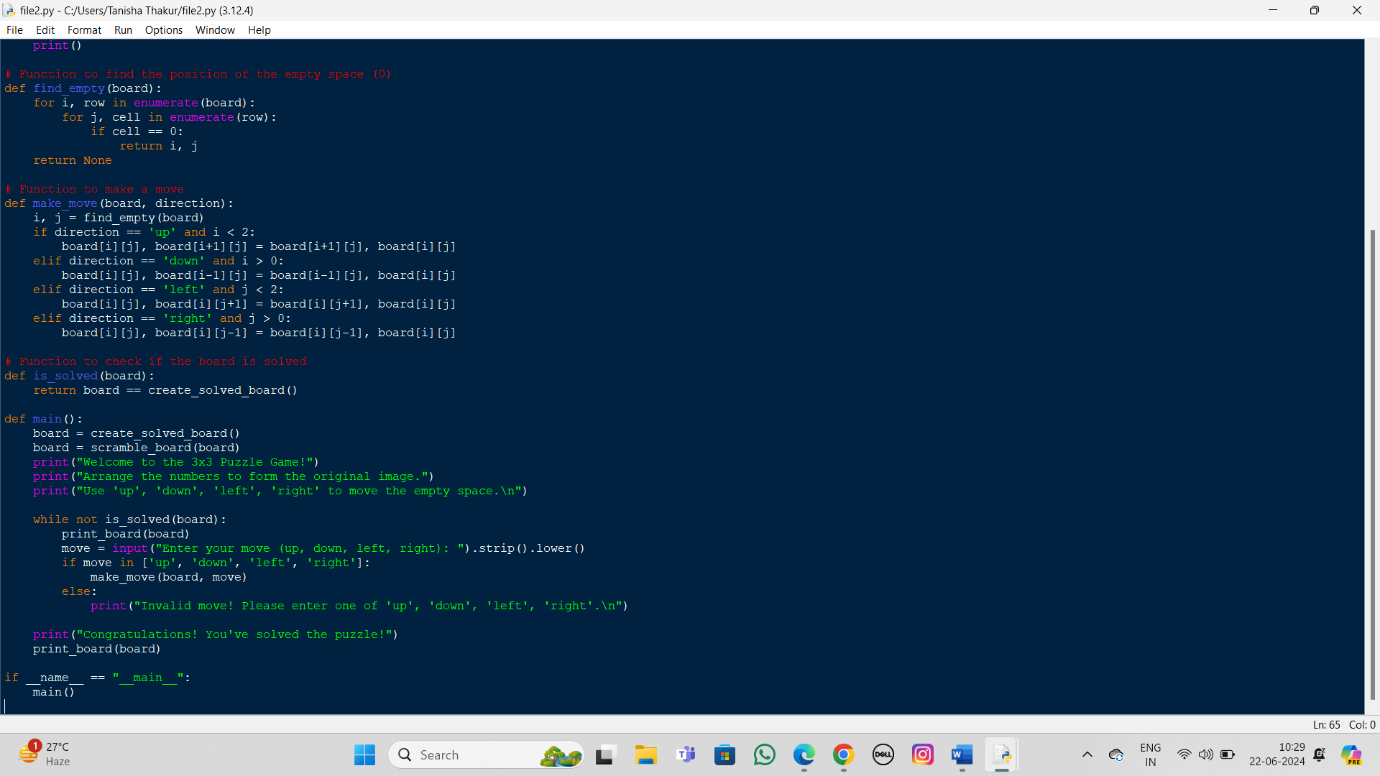
print\_board(board)

if \_\_name\_\_ == "\_\_main\_\_":

main()

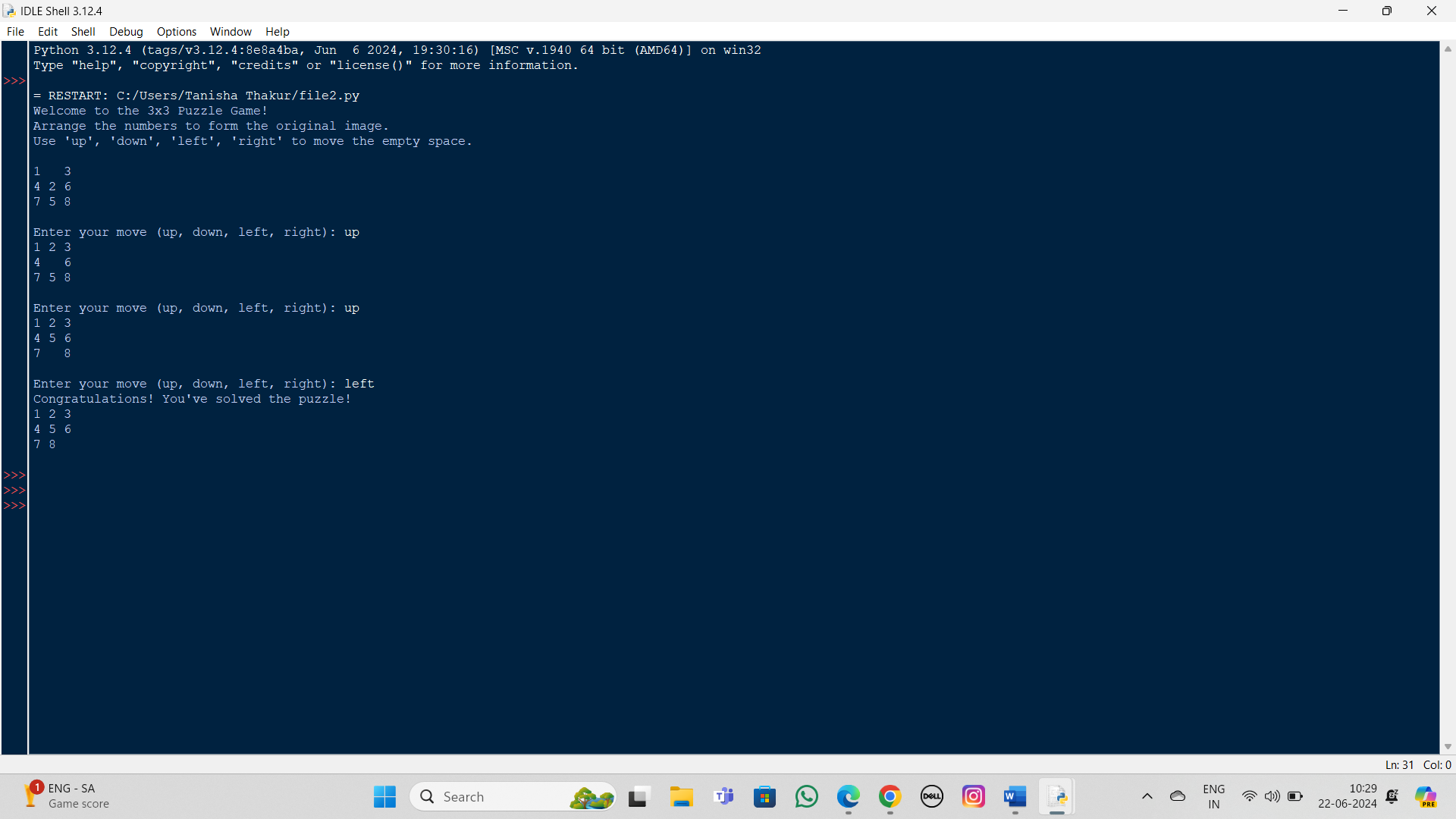








Output:



**Explanation**

1. **create\_solved\_board:** Generates a solved 3x3 puzzle.
2. **scramble\_board:** Randomly scrambles the board to create a puzzle.
3. **print\_board:** Displays the board in a human-readable format.
4. **find\_empty:** Finds the position of the empty space (0).
5. **make\_move:** Moves the empty space in the specified direction if the move is valid.
6. **is\_solved:** Checks if the board is in a solved state.
7. **main:** Manages the game loop, taking user input and updating the board.



**Thank you! Cognorise Infotech for giving me this wonderful opportunity to showcase my knowledge in app development.**

