Explanation

This Python code is for a **Weather App** built using the **Tkinter** library for the GUI, and it fetches weather data from the OpenWeather API based on the user's input. Let me break it down for you:

**1. Library Imports:**

* tkinter: For building the GUI.
* geopy.geocoders.Nominatim: Used for geolocation (converting city name to latitude/longitude).
* timezonefinder.TimezoneFinder: Finds the timezone of the location based on its coordinates.
* datetime: Handles date and time, allowing display of local time.
* requests: For making API calls to fetch weather data from OpenWeather.
* pytz: For time zone handling.

**2. Tkinter Setup:**

python

Copy code

root = Tk()

root.title("Weather App")

root.geometry("900x500+300+200")

root.resizable(False, False)

* This initializes the main window (root), sets the title to "Weather App", and defines the window size and position.
* The resizable(False, False) disables resizing of the window.

**3. getWeather() Function:**

This is the core function of the app, triggered when the user searches for a city's weather.

* **Step 1: Get user input**:

python

Copy code

city = textfield.get()

Retrieves the city name entered by the user.

* **Step 2: Geolocation**:

python

Copy code

geolocator = Nominatim(user\_agent="weather\_app")

location = geolocator.geocode(city)

Uses the Nominatim geocoder to get the latitude and longitude of the city. If the city isn't found, an error message is shown.

* **Step 3: Timezone and Local Time**:

python

Copy code

obj = TimezoneFinder()

result = obj.timezone\_at(lng=location.longitude, lat=location.latitude)

home = pytz.timezone(result)

local\_time = datetime.now(home)

current\_time = local\_time.strftime("%I:%M %p")

Uses the TimezoneFinder to find the timezone based on coordinates, then displays the local time in that timezone.

* **Step 4: Fetch Weather Data**:

python

Copy code

api\_key = "646824f2b7b86caffec1d0b16ea77f79"

api = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={api\_key}&units=metric"

json\_data = requests.get(api).json()

Uses the OpenWeather API to get weather data for the city. The response is in JSON format and includes information such as temperature, weather conditions, humidity, pressure, and wind speed.

* **Step 5: Extract Data**:

python

Copy code

condition = json\_data['weather'][0]['description'].capitalize()

temp = json\_data['main']['temp']

pressure = json\_data['main']['pressure']

humidity = json\_data['main']['humidity']

wind = json\_data['wind']['speed']

Extracts the relevant weather information like condition, temperature, pressure, humidity, and wind speed.

* **Step 6: Update the UI**:

python

Copy code

t.config(text=f"{temp}°C")

c.config(text=f"{condition} | FEELS LIKE {temp}°C")

w.config(text=f"{wind} m/s")

h.config(text=f"{humidity} %")

d.config(text=condition)

p.config(text=f"{pressure} hPa")

Updates various labels in the UI with the extracted weather data.

**4. User Interface (UI):**

The UI consists of several widgets like Label, Button, and Entry to create the input field, display weather information, and style the interface.

* **Search Box**:

python

Copy code

textfield = tk.Entry(root, justify="center", width=17, font=("poppins", 25, "bold"), bg="#404040", border=0, fg="white")

This is where the user inputs the city name. It's styled with font and color options.

* **Search Button**:

python

Copy code

myimage\_icon = Button(image=Search\_icon, borderwidth=0, cursor="hand2", bg="#404040", command=getWeather)

A button is linked to the getWeather() function, which triggers when clicked.

* **Display Labels**:

python

Copy code

label1 = Label(root, text="Wind", font=("Helvetica", 15, "bold"), fg="white", bg="#1ab5ef")

There are several labels for displaying the weather data like wind speed, humidity, weather description, and pressure.

**5. Main Loop:**

python

Copy code

root.mainloop()

This keeps the application running until the user manually closes it.

**Key Features:**

* **Error Handling**: If the city is not found or there’s an API issue, the app shows error messages using messagebox.showerror().
* **Real-time Data**: Fetches current weather and time based on the city’s geolocation and updates the UI dynamically.
* **Simple UI**: The design uses Labels and Entry widgets to create a clean interface.

Let me know if you need any specific part explained further!