WEEK 01 ***ADVANCE TASK***

BRAINNEST

CANDIDATE'S FULL NAME: Daniel Mula Tarancón

PROJECT STATEMENT

The goal of this project is to create a weather app that shows the current weather conditions and forecast for a specific location.

Here are the steps you can take to create this project:

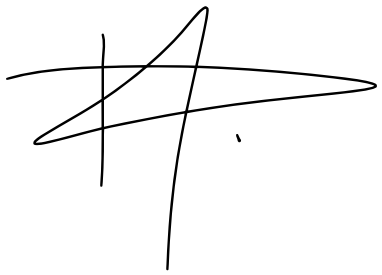
Use the request library to make an API call to a weather service (e.g. OpenWeatherMap) to retrieve the weather data for a specific location.

Use the json library to parse the JSON data returned by the API call.

Use the tkinter library to create a GUI for the app, including widgets such as labels, buttons and text boxes.

Use the Pillow library to display the weather icons.

Use the datetime library to display the current time and date.



INDEX

1. How did I approach this project?

Firstly I started review some documentation related to tkinter and Pillow. And after playing around a little and design the GUI I wanted on piece of sheet.

Then I decided that I could create a Class and add to that class all the elements of my Weather App.

So when I execute that Class the constructor method would return all I wanted.

2. Python libraries used along this project

In order to accomplish this task, these are the Python libraries I used:

tkinter —> For creating the GUI and its elements (buttons, labels…).

Pillow —> For inserting some images so I enhance the project.

requests —> For requesting data from in this case the Openweather API (current weather and forecast)

json —> So I could work with the data requested.

datetime —> Since I am using dates and times in these project, this is necessary library.

webbrowser —> I used this library so I can redirect the user to a website.

As you can see I also imported the file key (which is not include in the Git repository since it contains my personal API key) so I could request the data I needed.

```
import tkinter as tk
from tkinter import *
from PIL import ImageTk, Image
import requests
import json
from datetime import datetime, timedelta
from key import my_weather_key
import webbrowser

```

3. Getting into details

Here I will expose the details and parts of my code. Which I decided to divide them in the following way:

3.1. Blocks of code

My code is divided in three different parts:

- In one section I import all the libraries I need.

- In the other section I create and build my Class (which I've called 'Weather')

- And the last part is the 'dunder main' block where I control the execution of the script.

3.1.1. Importing libraries

I do not want to be redundant. This part is already explained in the second chapter of this report.

3.1.2. The Class

As I've mentioned before I've called my Class 'Weather'. This Class is composed by: A constructor method (def__init__(self, master); A method to build the grid (def build_grid(self)); A method to build the buttons(def build_buttons(self)); A method to build the labels (def build_labels(self)); A method to request the current weather (def request_info(self)); A method to get info about the country codes (def info(self)); A method to request the forecast (def forecast(self)).

A) def __init__(self, master)

```python
def __init__(self, master):
    self.master = master
    self.mainframe = tk.Frame(self.master, bg='white')
    self.mainframe.pack(fill=tk.BOTH, expand=True)
    self.master.title('Weather App')

    self.build_grid()
    self.build_buttons()
    self.build_labels()

    self.input_city = Entry(self.mainframe)
    self.input_city.grid(row=0, column=2)
    self.input_country = Entry(self.mainframe)
    self.input_country.grid(row=1, column=2)
    self.input_forecast = Entry(self.mainframe)
    self.input_forecast.grid(row=11, column=3)

    image_source = Image.open('source.jpeg')
    tk_image_source = ImageTk.PhotoImage(image_source)
    self.image_source_label = Label(self.mainframe, image=tk_image_source)
    self.image_source_label.image = tk_image_source
    self.image_source_label.grid(row=0, column=3)
```

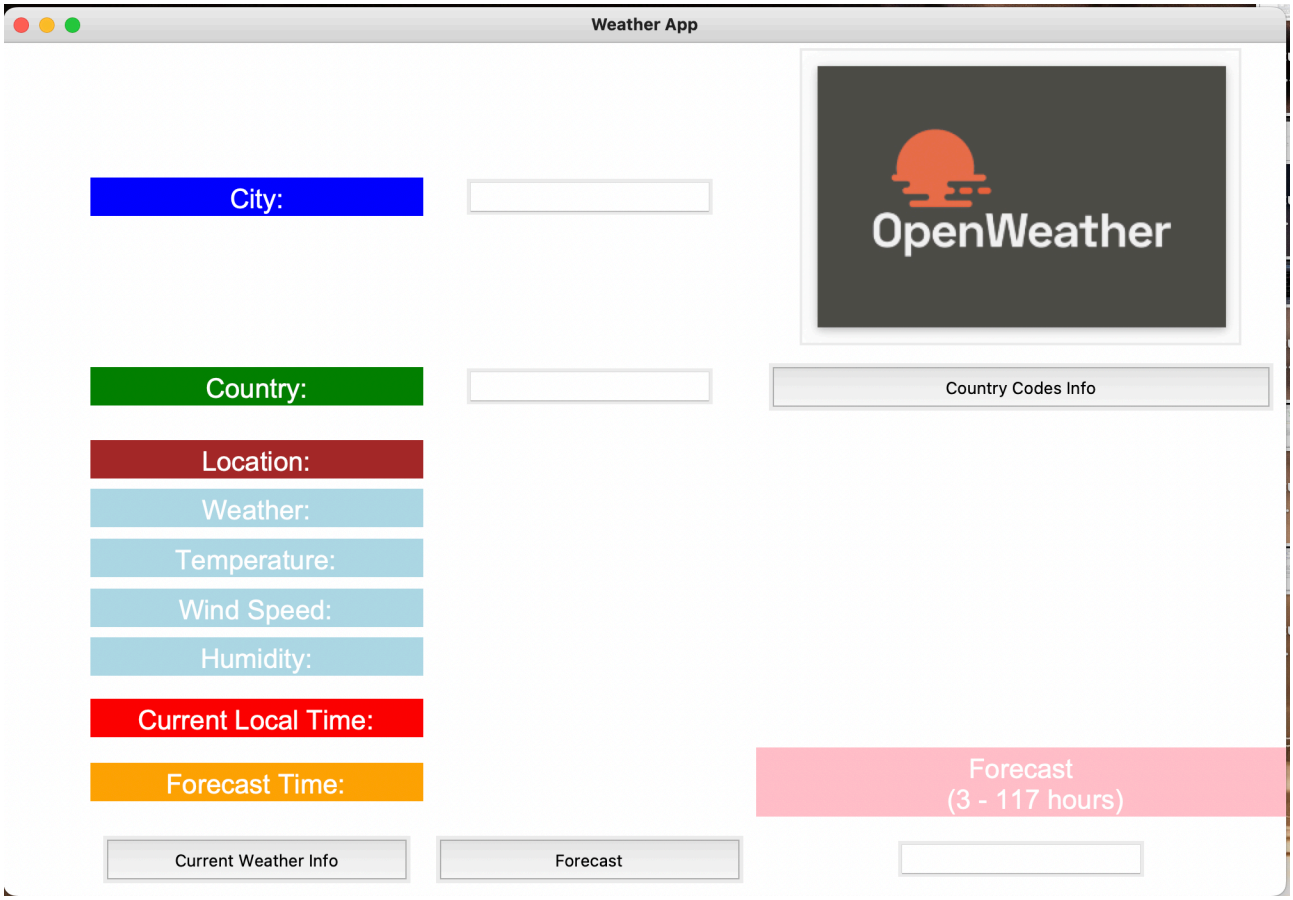
I create the parameter ‘master’ so I can pass the argument ‘root = tk.TK()’ that will create the main window of my GUI or the root window.

Then I create a FRAME using the tkinter class Frame and attached to my root window. On this Frame called ‘self.mainframe’ I’ll build my grid and I’ll put the rest of the widgets.

I decided to use a grid because I found easier to display what I wanted on the places I wanted.

An eventually using pack( ) I add the Frame to my root window.

As you can see the rest of the lines of code inside my constructor are the methods I want or need to execute once I execute my Class. Also as you can see I create some boxes so I can get some input (as you can see in the app image) below. And eventually I add the image that represent the source of data I am using in this case openweather.org.



B) def build_grid(self)

```
def build_grid(self):
    self.mainframe.columnconfigure(0, weight=1)
    self.mainframe.columnconfigure(1, weight=1)
    self.mainframe.columnconfigure(2, weight=1)
    self.mainframe.columnconfigure(3, weight=1)
    # Adding the rows
    self.mainframe.rowconfigure(0, weight=1)
    self.mainframe.rowconfigure(1, weight=1)
    self.mainframe.rowconfigure(2, weight=1)
    self.mainframe.rowconfigure(3, weight=1)
    self.mainframe.rowconfigure(4, weight=1)
    self.mainframe.rowconfigure(5, weight=1)
    self.mainframe.rowconfigure(6, weight=1)
    self.mainframe.rowconfigure(7, weight=1)
    self.mainframe.rowconfigure(8, weight=1)
    self.mainframe.rowconfigure(9, weight=1)
    self.mainframe.rowconfigure(10, weight=1)
    self.mainframe.rowconfigure(11, weight=1)
```

Inside this method I create the grid that so I can structure all my app in a neat way.

C) def build_buttons(self)

```
def build_buttons(self):

    self.weather_info_button = tk.Button(self.mainframe, text='Current Weather Info', command=self.request_info)
    self.weather_info_button.grid(row=11, column=1, sticky='nsew', pady=10, padx=10)

    self.country_codes_info_button = tk.Button(self.mainframe, text='Country Codes Info', command=self.info)
    self.country_codes_info_button.grid(row=1, column=3, sticky='nsew', pady=10, padx=10)

    self.weather_forecast_button = tk.Button(self.mainframe, text='Forecast', command=self.forecast)
    self.weather_forecast_button.grid(row=11, column=2, sticky='nsew', pady=10, padx=10)
```

Inside this method as you can see I've created three buttons.

- Current Weather Info: Once the user press this button gets the current weather info of the place selected. (This button triggers the self.request_info method).

- Country Codes Info: Once the user press this button is redirected to a webpage where he/she can verify the country code. (This button triggers the self.info method).

- Forecast: Once the user press this button gets the forecast selected (from 3 to 117 hours). (This button triggers the self.forecast method).

D) def build_labels(self)

```
def build_labels(self):
    city_label = Label(self.mainframe, text='City:', font=('Arial', 21), bg='blue', fg='white')
    city_label.grid(row=0, column=1, sticky='ew')
    country_label = Label(self.mainframe, text='Country:', font=('Arial', 21), bg='green', fg='white')
    country_label.grid(row=1, column=1, sticky='ew')
    location_label = Label(self.mainframe, text='Location:', font=('Arial', 21), bg='brown', fg='white')
    location_label.grid(row=3, column=1, sticky='ew')
    weather_label = Label(self.mainframe, text='Weather:', font=('Arial', 21), bg='light blue', fg='white')
    weather_label.grid(row=4, column=1, sticky='ew')
    temperature_label = Label(self.mainframe, text='Temperature:', font=('Arial', 21), bg='light blue', fg='white')
    temperature_label.grid(row=5, column=1, sticky='ew')
    wind_label = Label(self.mainframe, text='Wind Speed:', font=('Arial', 21), bg='light blue', fg='white')
    wind_label.grid(row=6, column=1, sticky='ew')
    humidity_label = Label(self.mainframe, text='Humidity:', font=('Arial', 21), bg='light blue', fg='white')
    humidity_label.grid(row=7, column=1, sticky='ew')
    date_label = Label(self.mainframe, text='Current Local Time:', font=('Arial', 21), bg='red', fg='white')
    date_label.grid(row=9, column=1, sticky='ew')
    forecast_date_label = Label(self.mainframe, text='Forecast Time:', font=('Arial', 21), bg='orange', fg='white')
    forecast_date_label.grid(row=10, column=1, sticky='ew')
    forecast_label = Label(self.mainframe, text='''Forecast
(3 - 117 hours)''', font=('Arial', 21), bg='pink', fg='white')
    forecast_label.grid(row=10, column=3, sticky='ew')
```

Inside this method I create all the labels I desire for my app. And I insert them into my grid structure.

E) def request_info(self)

```
def request_info(self):
    #Making the API request

    api_key = my_weather_key
    city_name = self.input_city.get()
    country_code = self.input_country.get()
    url = f'https://api.openweathermap.org/data/2.5/weather?q={city_name},{country_code}&appid={api_key}'
    res = requests.get(url)
    data = res.json()
    print(data)

    #Displaying the data

    location_text = Text(self.mainframe, height=2, width=20)
    location = 'Longitude: ' + str(data['coord']['lon']) + '  Latitude: ' + str(data['coord']['lat'])
    location_text.insert(END, location)
    location_text.grid(row=3, column=2, sticky='ew')

    weather_text = Text(self.mainframe, height=2, width=20)
    weather = str(data['weather'][0]['main']) + ' ' + '(' + data['weather'][0]['description'] + ')'
    weather_text.insert(END, weather)
    weather_text.grid(row=4, column=2, sticky='ew')


    #Adding some pictures
    weather_value = str(data['weather'][0]['main'])
    weather_options = ['Clear', 'Clouds', 'Rain', 'Snow']
    img_options = ['sunny.jpeg', 'cloudy.jpg', 'rainny.jpeg','snow.jpeg']
    w_index = weather_options.index(weather_value)
    img = img_options[w_index]
    image_weather = Image.open(img)
    resized_image = image_weather.resize((400,400))
    tk_image_weather = ImageTk.PhotoImage(resized_image)
    self.image_weather_label = Label(self.mainframe, image=tk_image_weather)
    self.image_weather_label.image = tk_image_weather
    self.image_weather_label.grid(row=4, column=3, rowspan=4)


    #-------------------------------------------------------------------------

    temperature_text = Text(self.mainframe, height=2, width=20)
    temperature = str(round(data['main']['temp'] - 273.15,2)) + '°C'
    temperature_text.insert(END, temperature)
    temperature_text.grid(row=5, column=2, sticky='ew')
```

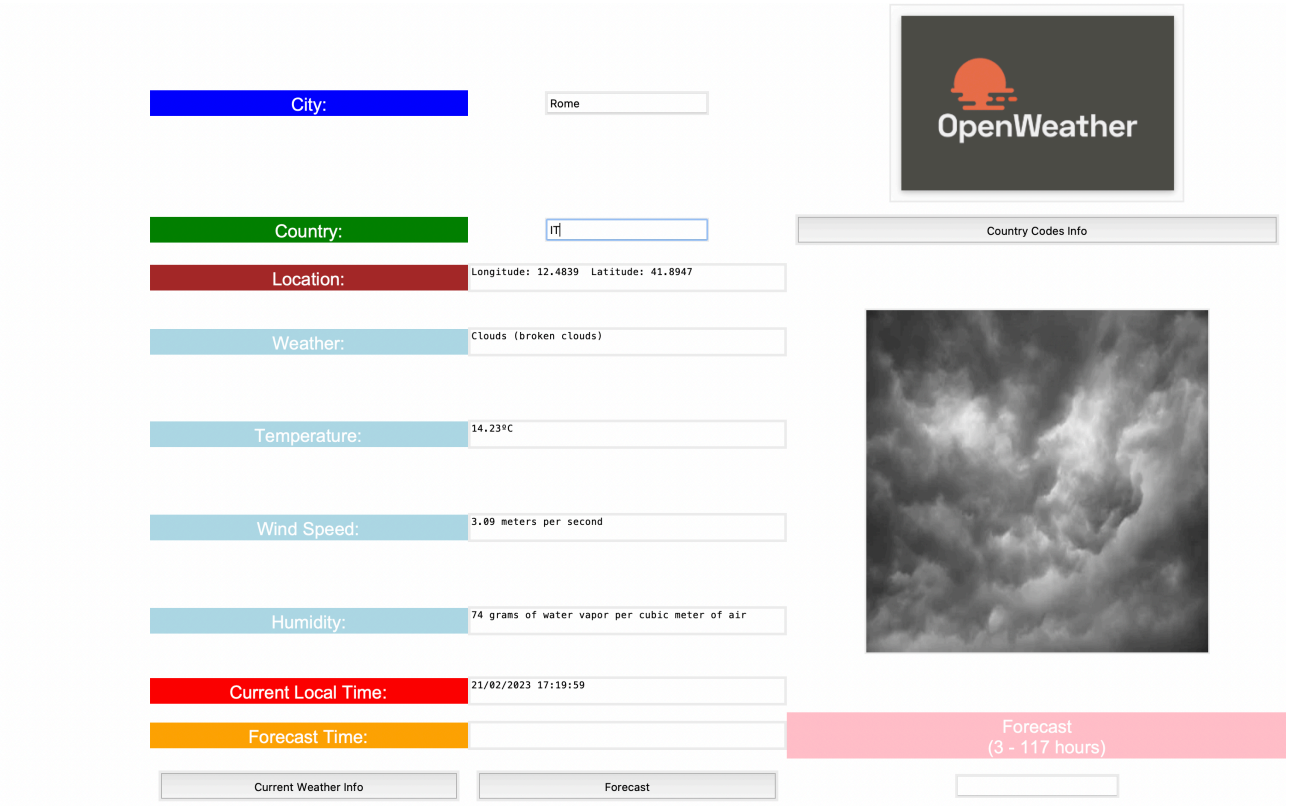
```
wind_text = Text(self.mainframe, height=2, width=20)
wind_speed = str(data['wind']['speed']) + ' meters per second'
wind_text.insert(END, wind_speed)
wind_text.grid(row=6, column=2, sticky='ew')

humidity_text = Text(self.mainframe, height=2, width=20)
humidity = str(data['main']['humidity']) + ' grams of water vapor per cubic meter of air'
humidity_text.insert(END, humidity)
humidity_text.grid(row=7, column=2, sticky='ew')

date_text = Text(self.mainframe, height=2, width=20)
tz_offset = timedelta(seconds=data['timezone'])
utc_time = datetime.utcnow()
local_time = utc_time + tz_offset
date = local_time.strftime("%d/%m/%Y %H:%M:%S")
date_text.insert(END, date)
date_text.grid(row=9, column=2, sticky='ew')

f_date_text = Text(self.mainframe, height=2, width=20)
f_date = ''
f_date_text.insert(END, f_date)
f_date_text.grid(row=10, column=2, sticky='ew')
```

Firstly I make the API request using the inputs given by the user.

Then I display the weather data into the grid so everything is in order.

As you can see I've added some images that will be shown depending on the weather conditions.

Eventually I display the local time of the place requested. And I display nothing in the forecast time text box because this is not the forecast.


City:
Rome
OpenWeather
Country:
IT
Country Codes Info
Location:
Longitude: 12.4839 Latitude: 41.8947
Weather:
Clouds (broken clouds)
Temperature:
14.23ºC
Wind Speed:
3.09 meters per second
Humidity:
74 grams of water vapor per cubic meter of air
Current Local Time:
21/02/2023 17:19:59
Forecast Time:
Forecast
(3 - 117 hours)
Current Weather Info
Forecast

F) info(self)

```
def info(self):
    url = "https://www.iso.org/obp/ui/#search"
    webbrowser.open_new_tab(url)
```

This is a simple method that redirect the user to a website where can get the proper country code.

G) def forecast(self)

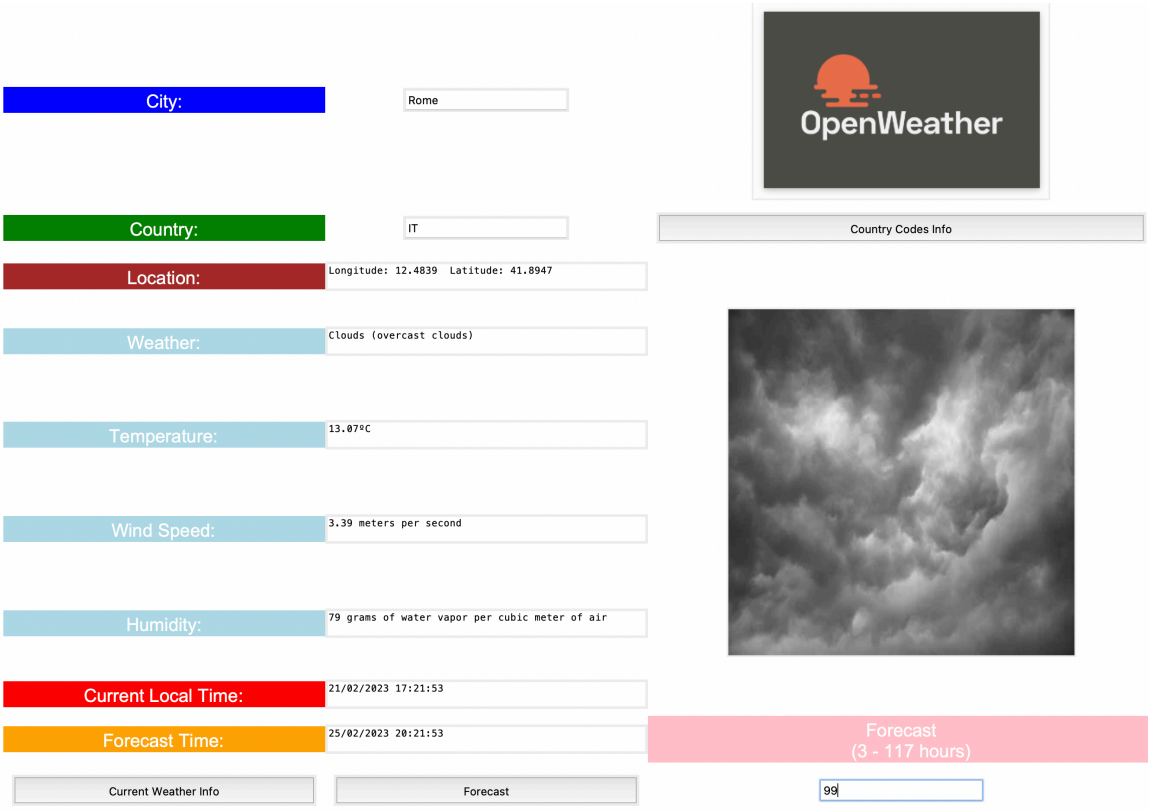
```
def forecast(self):
    #Making the API request

    api_key = my_weather_key
    city_name = self.input_city.get()
    country_code = self.input_country.get()
    forecast_hours = int(self.input_forecast.get())
    url = f'https://api.openweathermap.org/data/2.5/weather?q={city_name},{country_code}&appid={api_key}'
    f_url = f'https://api.openweathermap.org/data/2.5/forecast?q={city_name},{country_code}&appid={api_key}'
    res = requests.get(url)
    f_res = requests.get(f_url)
    data = res.json()
    f_data_gross = f_res.json()
    index = forecast_hours // 3
    f_data = f_data_gross['list'][index]



    # Displaying the data

    location_text = Text(self.mainframe, height=2, width=20)
    location = 'Longitude: ' + str(data['coord']['lon']) + '  Latitude: ' + str(data['coord']['lat'])
    location_text.insert(END, location)
    location_text.grid(row=3, column=2, sticky='ew')

    weather_text = Text(self.mainframe, height=2, width=20)
    weather = str(f_data['weather'][0]['main']) + ' ' + '(' + f_data['weather'][0]['description'] + ')'
    weather_text.insert(END, weather)
    weather_text.grid(row=4, column=2, sticky='ew')
```

```
# Adding some pictures
weather_value = str(f_data['weather'][0]['main'])
weather_options = ['Clear', 'Clouds', 'Rain', 'Snow']
img_options = ['sunny.jpeg', 'cloudy.jpg', 'rainny.jpeg', 'snow.jpeg']
w_index = weather_options.index(weather_value)
img = img_options[w_index]
image_weather = Image.open(img)
resized_image = image_weather.resize((400, 400))
tk_image_weather = ImageTk.PhotoImage(resized_image)
self.image_weather_label = Label(self.mainframe, image=tk_image_weather)
self.image_weather_label.image = tk_image_weather
self.image_weather_label.grid(row=4, column=3, rowspan=4)

# ------------------------------------------------------------------------
```

```
temperature_text = Text(self.mainframe, height=2, width=20)
temperature = str(round(f_data['main']['temp'] - 273.15, 2)) + '°C'
temperature_text.insert(END, temperature)
temperature_text.grid(row=5, column=2, sticky='ew')

wind_text = Text(self.mainframe, height=2, width=20)
wind_speed = str(f_data['wind']['speed']) + ' meters per second'
wind_text.insert(END, wind_speed)
wind_text.grid(row=6, column=2, sticky='ew')

humidity_text = Text(self.mainframe, height=2, width=20)
humidity = str(f_data['main']['humidity']) + ' grams of water vapor per cubic meter of air'
humidity_text.insert(END, humidity)
humidity_text.grid(row=7, column=2, sticky='ew')
```

```
f_date_text = Text(self.mainframe, height=2, width=20)
delta = int((forecast_hours // 3)*3)
forecast_time = utc_time + tz_offset + timedelta(hours=delta)
f_date = forecast_time.strftime("%d/%m/%Y %H:%M:%S")
f_date_text.insert(END, f_date)
f_date_text.grid(row=10, column=2, sticky='ew')
```

Here I am doing almost the same I did with the self.request_info method but I must do a another request (I need the forecast) and I also display the forecast time.


City:
Rome
OpenWeather
Country:
IT
Country Codes Info
Location:
Longitude: 12.4839 Latitude: 41.8947
Weather:
Clouds (overcast clouds)
Temperature:
13.07°C
Wind Speed:
3.39 meters per second
Humidity:
79 grams of water vapor per cubic meter of air
Current Local Time:
21/02/2023 17:21:53
Forecast Time:
25/02/2023 20:21:53
Forecast
(3 - 117 hours)
Current Weather Info
Forecast
99

3.1.3. Executing (dunder main)

```
if __name__=='__main__':
    root = tk.Tk()
    Weather(root)
    root.mainloop()
```

Using ‘dunder main’ we can control the execution of the script so it only runs when we call it. If the script were imported as a module into another script, the code inside the block wouldn’t be executed.

In this case we won’t have that problem but I think is a good practice to use a ‘dunder main’.

4. Conclusions

Dedicating more time would allow me to improve the App. Nonetheless, I believe the App is nice enough and it achieves its purpose which is getting the current weather and the forecast of a given place.