Weather forecast Project Documentation

By Andre Mathias and Daniil Malish

Project Overview:

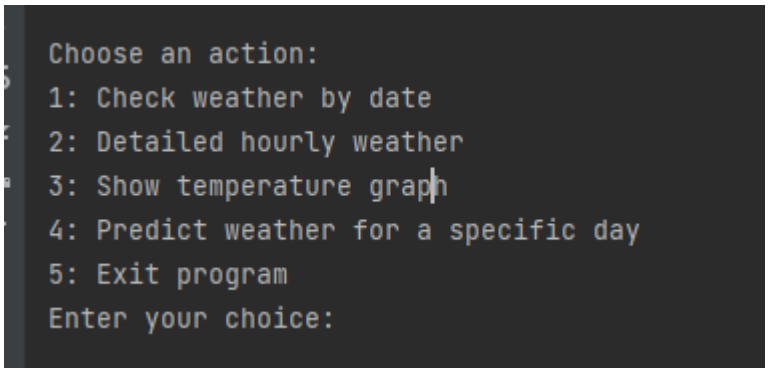
The purpose of this exercise work is to write code that allows for user input, for a certain date, and given that date certain weather-related actions such as drawing a temperature graph for the present day, simply representing weather conditions over a period or to make weather predictions for future dates. This project is very easy to use and is meant for all kinds of users who are simply curious about the current, past or future weather.

Installation:

This project makes use of a few external libraries such as requests, matplotlib, datetime and random. These libraries must sometimes be installed with pip, in case they are not already present. Aside from this, a simple IDE and a python version is needed.

Features and Usages:

The program contains five main methods:



1: checking the weather overview for a certain date that is not in the future

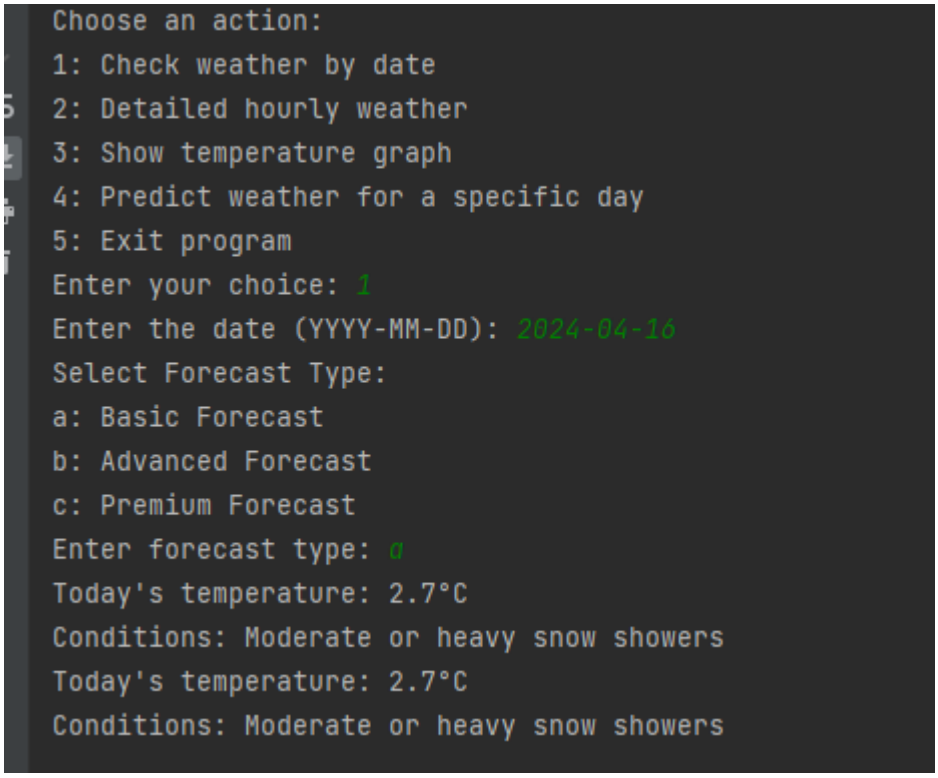
2: printing the hourly temperature for a given date that is not in the future

3: plot a temperature/h diagram for a given date that is not in the future

4: predict the temperature for a day in the future

1:

When the user answers with 1, they are given an initial prompt to pick the type of information they would like to receive, from a to c. These are the basic, advanced and premium forecasts, which each print more information than the previous one. They then expect a date input from the user in YY-DD-MM format, and once provided the code will then give a description of the weather that day based on information from an API call. An example usage would be



In case a future date is entered by a user, method 4, so the method for predicting the weather instead, will be called.

Additionally, a user can only call weather data up to 365 days in the past

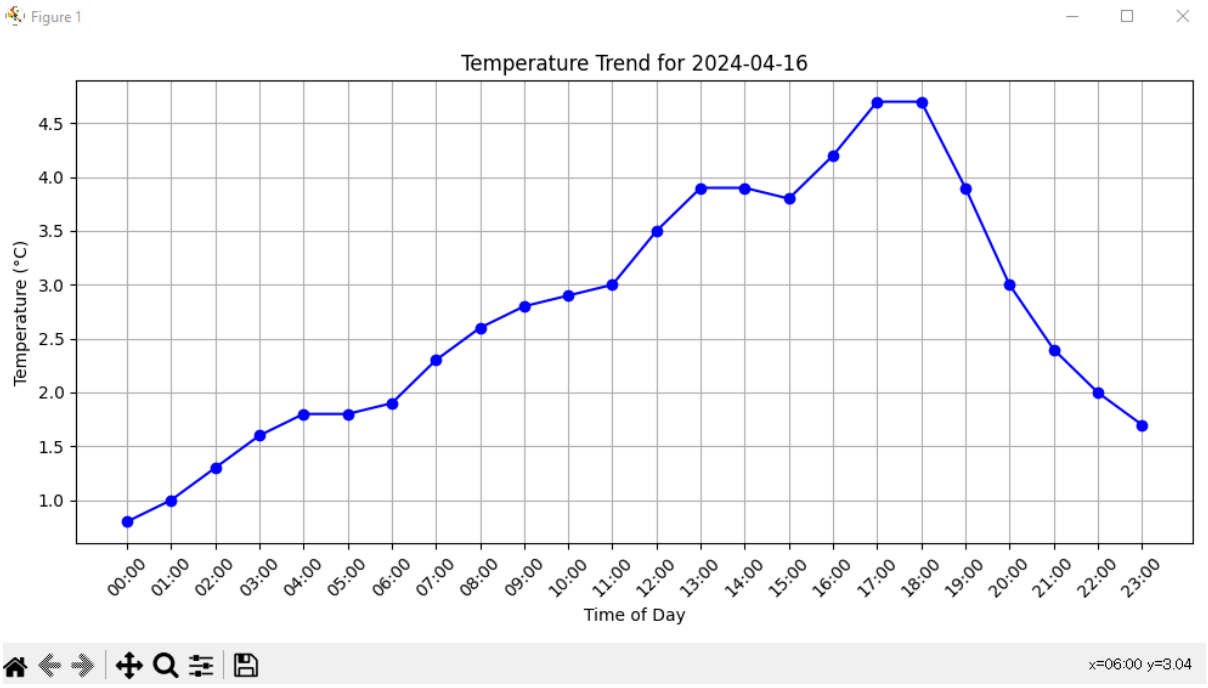
2:

When a user answers with 2, they are given the same prompt asking for a specific date. If they give a date that is either future or too far in the past, the method will inform them and let them try again. If a valid date is entered, hourly information is shown as such:

```
Enter the date for hourly details (YYYY-MM-DD): 2024-04-16
Hourly Forecast:
Time: 2024-04-16 00:00, Temperature: 0.8°C, Condition: Mist
Time: 2024-04-16 01:00, Temperature: 1.0°C, Condition: Patchy rain possible
Time: 2024-04-16 02:00, Temperature: 1.3°C, Condition: Light snow
Time: 2024-04-16 03:00, Temperature: 1.6°C, Condition: Light snow
Time: 2024-04-16 04:00, Temperature: 1.8°C, Condition: Light snow
Time: 2024-04-16 05:00, Temperature: 1.8°C, Condition: Light snow
Time: 2024-04-16 06:00, Temperature: 1.9°C, Condition: Light snow
Time: 2024-04-16 07:00, Temperature: 2.3°C, Condition: Light snow
Time: 2024-04-16 08:00, Temperature: 2.6°C, Condition: Light snow
Time: 2024-04-16 09:00, Temperature: 2.8°C, Condition: Moderate snow
Time: 2024-04-16 10:00, Temperature: 2.9°C, Condition: Moderate snow
Time: 2024-04-16 11:00, Temperature: 3.0°C, Condition: Moderate or heavy snow showers
Time: 2024-04-16 12:00, Temperature: 3.5°C, Condition: Light snow
Time: 2024-04-16 13:00, Temperature: 3.9°C, Condition: Patchy rain possible
Time: 2024-04-16 14:00, Temperature: 3.9°C, Condition: Patchy rain possible
Time: 2024-04-16 15:00, Temperature: 3.8°C, Condition: Patchy rain possible
Time: 2024-04-16 16:00, Temperature: 4.2°C, Condition: Patchy rain possible
```

3:

When a user answers with 3, they are yet again given the prompt asking for a specific date. There are again validation methods used to confirm the validity, and in case it is valid, using matplotlib a graph will be plotted using the API data for that day.



4:

When a user answers with 4, they asked to provide a date whose temperature the code will attempt to predict. This however is not modelled with a complex machine learning algorithm, but rather with a very trivial algorithm: for each day with unknown temperature since the last known day until the target day, add a temperature fluctuation of +-2°C. While this can produce accurate answers for results several days in advance, if the user enters a prompt for a date that is years in the future, possible temperatures of hundreds of degrees can be reached.

```
Choose an action:
1: Check weather by date
2: Detailed hourly weather
3: Show temperature graph
4: Predict weather for a specific day
5: Exit program
Enter your choice: 4
Enter the date for prediction (YYYY-MM-DD): 2040-01-01
Predicting for a future day.
Predicted average temperature for 2040-01-01: 76.6°C
```

5:

If the user presses 5, the program will simply close.

Relevant classes:

The most relevant classes in the project structure that are used in every exercise are

WeatherAPI

WeatherData

Application

The WeatherAPI class simply has the task to fetch the weather data for a given day from Turku

The WeatherData class is the class in which our weather information is stored. That is, we are creating objects of the WeatherData class that contain dictionaries which store our information, which is then called by the application class. This class also contains most relevant methods for displaying the information and plotting the graphs. Display_basic_info is for simply presenting information about the day when needed.

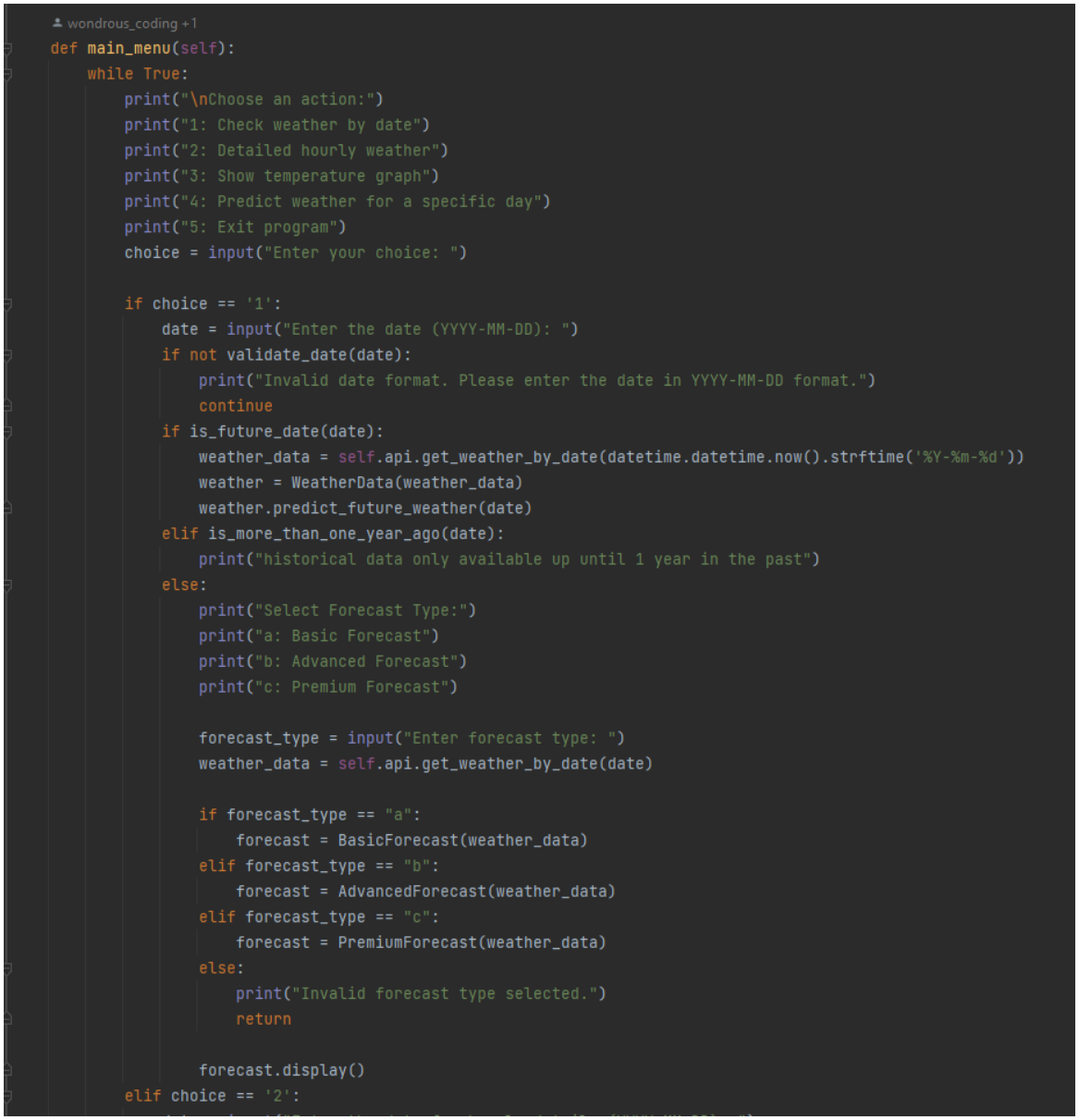
Display_hourly_info is for printing the temperature for every hour reading that day

Predict_future_weather is a “stupid” algorithm that attempts to guess the temperature given the previously outlined rules

Plot_daily_temperature uses Matplotlib to plot a graph showing the temperature progression throughout the day.

```python
wondrous_coding
class WeatherData:
    wondrous_coding
    def __init__(self, data):
        if data is None:
            raise ValueError("No data provided for weather information.")
        self.location = data['location']
        self.forecast = data['forecast']['forecastday'][0]

    wondrous_coding
    def display_basic_info(self):
        location_info = (self.location['name'], self.location['region'], self.location['country'])
        weather_info = self.forecast['day']
        condition = weather_info['condition']['text']
        max_temp = weather_info['maxtemp_c']
        min_temp = weather_info['mintemp_c']
        average_temp = weather_info['avgtemp_c']
        humidity = weather_info['avghumidity']
        uv_index = weather_info['uv']

        print(f"Weather Information for {location_info[0]}, {location_info[1]}, {location_info[2]}:")
        print(f"Condition: {condition}")
        print(f"Max Temp: {max_temp}°C, Min Temp: {min_temp}°C, Avg Temp: {average_temp}°C")
        print(f"Humidity: {humidity}%")
        print(f"UV Index: {uv_index}")

    wondrous_coding
    def display_hourly_info(self):
        print("Hourly Forecast:")
        for hour in self.forecast['hour']:
            time = hour['time']
            temp = hour['temp_c']
            condition = hour['condition']['text']
            print(f"Time: {time}, Temperature: {temp}°C, Condition: {condition}")

    wondrous_coding
    def plot_daily_temperature(self):
        times = [hour['time'][11:] for hour in self.forecast['hour']]
        temperatures = [hour['temp_c'] for hour in self.forecast['hour']]

        plt.figure(figsize=(10, 5))
        plt.plot(times, temperatures, marker='o', linestyle='-', color='b')
        plt.title(f"Temperature Trend for {self.forecast['date']}")
        plt.xlabel('Time of Day')
        plt.ylabel('Temperature (°C)')
        plt.xticks(rotation=45)
        plt.grid(True)
        plt.tight_layout()
        plt.show()

    wondrous_coding
    def predict_future_weather(self, target_date_str):
        current_date = datetime.datetime.now()
```

The application class runs our application by utilizing a simple while true loop.



```
wondrous_coding +1
def main_menu(self):
    while True:
        print("\nChoose an action:")
        print("1: Check weather by date")
        print("2: Detailed hourly weather")
        print("3: Show temperature graph")
        print("4: Predict weather for a specific day")
        print("5: Exit program")
        choice = input("Enter your choice: ")

        if choice == '1':
            date = input("Enter the date (YYYY-MM-DD): ")
            if not validate_date(date):
                print("Invalid date format. Please enter the date in YYYY-MM-DD format.")
                continue
            if is_future_date(date):
                weather_data = self.api.get_weather_by_date(datetime.datetime.now().strftime('%Y-%m-%d'))
                weather = WeatherData(weather_data)
                weather.predict_future_weather(date)
            elif is_more_than_one_year_ago(date):
                print("historical data only available up until 1 year in the past")
            else:
                print("Select Forecast Type:")
                print("a: Basic Forecast")
                print("b: Advanced Forecast")
                print("c: Premium Forecast")

                forecast_type = input("Enter forecast type: ")
                weather_data = self.api.get_weather_by_date(date)

                if forecast_type == "a":
                    forecast = BasicForecast(weather_data)
                elif forecast_type == "b":
                    forecast = AdvancedForecast(weather_data)
                elif forecast_type == "c":
                    forecast = PremiumForecast(weather_data)
                else:
                    print("Invalid forecast type selected.")
                    return

                forecast.display()
        elif choice == '2':
```

It is from here that a lot of the logic is implemented, and that information is called and used.

WeatherData objects are used in this class to achieve the desired outcome.

Lesser important classes are the forecast classes that are only used in option 1:

```
AndreSebaMa +1
class Forecast:
    AndreSebaMa +1
    def __init__(self, data):
        if data is None:
            raise ValueError("No data provided for weather information.")
        self.forecast_day = data['forecast']['forecastday'][0]['day']
        self.forecast_hours = data['forecast']['forecastday'][0]['hour']
        self.display()

    wondrous_coding
    def display(self):
        raise NotImplementedError("Subclasses must implement this method.")

AndreSebaMa +1
class BasicForecast(Forecast):
    AndreSebaMa +1
    def display(self):
        print(f"Today's temperature: {self.forecast_day['avgtemp_c']}°C")
        print(f"Conditions: {self.forecast_day['condition']['text']}")

wondrous_coding +1
class AdvancedForecast(Forecast):
    AndreSebaMa +1
    def display(self):
        print("Today's temperature:")
        for hour in self.forecast_hours:
            print(f"{hour['time'][11:]} - Temp: {hour['temp_c']}°C, Condition: {hour['condition']['text']}

AndreSebaMa +1
class PremiumForecast(Forecast):
    AndreSebaMa +1
    def display(self):
        print("Detailed Forecast with Predictive Analysis:")
        print(f"Today's temperature: {self.forecast_day['avgtemp_c']}°C")
        for hour in self.forecast_hours:
            feels_like = hour.get('feelslike_c', 'N/A')
            print(f"{hour['time'][11:]} - Temp: {hour['temp_c']}°C, Feels Like: {feels_like}°C")
```

These serve a mostly cosmetic function and are not used outside of the first option. The classes all inherit from the Forecast class.

Outside of the classes, there are several validation methods to ensure proper formatting and avoid errors

```
wondrous_coding
def validate_date(date_str):
    try:
        datetime.datetime.strptime(date_str, '%Y-%m-%d')
        return True
    except ValueError:
        return False


wondrous_coding
def is_future_date(date_str):
    today = datetime.datetime.now()
    date = datetime.datetime.strptime(date_str, '%Y-%m-%d')
    return date > today


wondrous_coding
def is_more_than_one_year_ago(date_str):
    today = datetime.datetime.now()
    one_year_ago = today - datetime.timedelta(days=365)
    input_date = datetime.datetime.strptime(date_str, '%Y-%m-%d')
    return input_date < one_year_ago
```

The resting report that leads to these can be found in GitHub