Weather and air quality display using Python and API

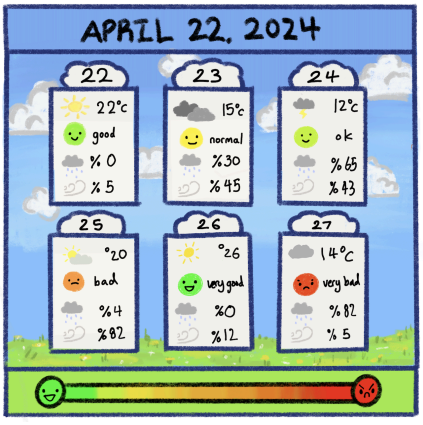
I wanted to try something different instead of only creating games in Python. So, I will complete a weather information display for this project with the Application Programming Interface(API) and Python. The reason I am doing this is because it has caught my attention to understanding the utility of API. It is a confusing topic for me and since one of the ways of using an API is creating a weather display, I thought that this project would enhance my knowledge of API.

What is API?

API is a collection of communication protocols and subroutines of different programs used to communicate between them (Jain). Accurate and reliable data is essential for a weather and air quality display, as its primary purpose is to inform users about the current weather conditions and air quality levels. So, I will find a weather API, collect the weather information, and display the data on a GUI.

To decide which API to use, I considered the specific location where the weather information will be displayed. Since the weather conditions vary by location, I chose to focus on a particular country and city to display: Songdo in Incheon, South Korea. I figured that I needed two different APIs: weather API and air quality API. Since the information I need is the weather conditions in Songdo, I got APIs that are created in South Korea. Local information was a good choice as they are the ones who understand the country the most. For the air quality API, I chose "에어코리아" which is in translation "Air Korea" an organization that collects air quality information all around Korea, and for the weather API, I chose "기상청" which is the Korean Meteorological Administration the National Meteorological Service of the Republic of Korea.

For the GUI, I decided on using Tkinter in Python due to my many experiences with it, I used it many times and understood its functionality. Before I started planning my weather and air quality display, I first sketched out how I wanted the GUI to look like:



For the weather and air quality display GUI, I made it have a landscape background and have each day be a chunk of cloud. I wanted the information to be simple and direct so that users could glance at the GUI and then continue doing whatever they were doing; I was looking more into convenience. Additionally, I focused a lot on the aesthetics of the GUI because I thought the overall look should be appealing to users and be able to easily detect the weather conditions based on what the image is expressing.

Information about this project:

Requirements:

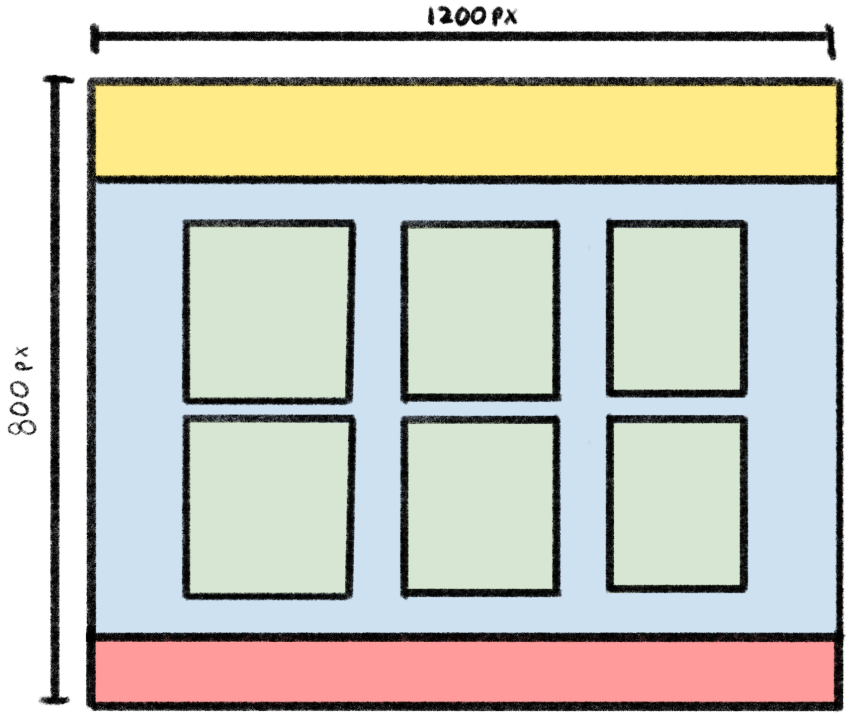
- The GUI has to show the weather and air quality in the following 7 days.
- The weather information must include the temperature, rainfall probability, rainfall, and clearness.
- Images will be used to represent certain times throughout the day and the conditions of the weather

GUI information:

- Screen size: 1200 x 800
- Background: Light Blue
- Date banner on the top: (frame → inside label)
    - Size: 1200 x 150
    - Information: date
    - Background: light blue
- Air quality scale banner: (frame → image Label))
    - Size: 1200 x 150
    - Air quality scare drawing
- Main background:
    - Size: 1200 x 500
    - Background: the landscape background
- Small sections for each day (frame → image+text inside canvas, Label on top)
    - Size: 300 x 450
    - Background: white
    - Information: day, date, weather temperature, air quality, wind strength, bright/dark
        - Temperature: graphic, numbers
        - Air quality: graphic, words

        - Rain probability: image, numbers
        - Rain amount: graphic, numbers
    - Three days only (current, the day after, the day after tomorrow)
    - The API only shows the next two days only
        - When there is no information for the third day put “Loading…” or “N/A”

The sections can be located by looking at this color-coded image:



Function:

- Understanding how to use API
    - Weather information API
    - Air quality information API
- Tikinter → GUI
- Create code to display information obtained through API on GUI
- Background color change (depending on what time of day it is)
    - Morning: 06:00 ~ 11:59
    - Afternoon: 12:00 ~ 16:59
    - Noon: 17:00 ~ 18:59
    - Night: 19:00 ~ 24:00 / 00:00 ~ 5:59

Collecting the weather forecast and air quality information:

Before beginning to program the graphics, I first decided to collect the weather forecast data from the Korean Meteorological Administration's API and the Air quality information from Air Korea's API. These two organizations had provided Open APIs available for everyone; so there isn't anything restricted to collecting the weather information.

There was a guidebook for the Korean Meteorological Administration Open API that provided me with how to use the Open API that they developed. Since the purpose of this API is to easily collect weather information and apply it to different codes, the Korean Meteorological Administration already provided the beginning part of the code. With the personal information like the API access key and selecting the certain day of weather information the beginning of the code looks like the code below:

```
      import requests
      import json
      url =
'http://apis.data.go.kr/1360000/VilageFcstInfoService_2.0/getVilageFcst'
#connect the Python program with the website
      key =
'Ts+NMfEsOgySQHW4+bWRkZP21N8vBTl4bqCkgyB8uzQMES9dcxmiADWp5qopwg9j0CqzjRWst+RYPI
6ElYSbiA==' # API Access Key (personal)
      params = {'serviceKey': key, 'pageNo': '1', 'numOfRows': '1000',
'dataType': 'JSON', #parameters | key, serviceKey, page #, number of row's,
data type: xml/json
                'base_date': '20240516', 'base_time': '0500', 'nx': '55', 'ny':
'127'}
      # base data, base time, nx = longitude, ny latitude

      #request data to connect with the Python program (generate data from the
website and put it into the Python program)
      response = requests.get(url, params=params) #raw data
      #json.loads --> reads the raw data in response  (data reading)
      res_json = json.loads(response.content) #read data is saved in res_jason
which is in a dictionary type
```

The given code above:

- Imports necessary libraries for making HTTP requests and handling JSON data.
- Defines the API endpoint URL and the personal access key required for authentication.
- Set up the API request parameters, including date, time, and geographical coordinates.
- Sends an HTTP GET request to the specified API endpoint with the given parameters.
- Receives and parses the JSON response, converting it into a Python dictionary for further processing.

Then, the following code fetches and processes weather forecast data from the Korea Meteorological Administration's Village Forecast Service:

```
items = res_json['response']['body']['items']['item']
tempInfo = {'Date': [], 'Time': [], 'Temp': [], 'SKY':[], 'POP': [], 'PCP': [],
'PTY':[]}
n = 0

#filtering the raw data (that we need))
for i in items:
   if i['category'] == 'TMP':
       tempInfo['Date'].append(i['fcstDate'])
       tempInfo['Temp'].append(i['fcstValue'])
       tempInfo['Time'].append(i['fcstTime'][:2] + 'H')
       n += 1
   elif i['category'] == 'POP' and tempInfo['Date'][n - 1] == i['fcstDate']:
       tempInfo['POP'].append(i['fcstValue'])
   elif i['category'] == 'PCP' and tempInfo['Date'][n - 1] == i['fcstDate']:
       tempInfo['PCP'].append(i['fcstValue'])
   elif i['category'] == 'SKY' and tempInfo['Date'][n - 1] == i['fcstDate']:
       tempInfo['SKY'].append(i['fcstValue'])
   elif i['category'] == 'PTY' and tempInfo['Date'][n - 1] == i['fcstDate']:
       tempInfo['PTY'].append(i['fcstValue'])
```

The code extracts the relevant forecast items from the JSON response, which are stored in “items”. A dictionary named “tempInfo” is then initialized to hold lists for various weather parameters, such as date, time, temperature (Temp), sky condition (SKY), probability of precipitation (POP),

precipitation amount (PCP), and precipitation type (PTY). A counter “n” is also initialized to keep track of the number of entries.

To see the various weather parameters the following code is used:

```
for i in range(n):
   print(tempInfo['Date'][i], tempInfo['Time'][i], tempInfo['Temp'][i],
tempInfo['SKY'][i], tempInfo['POP'][i], tempInfo['PCP'][i], tempInfo['PTY'][i])
```

When the information is shown, it only shows the first day:

['Wed', 'Jul', '10', '16:38:58', '2024', '4', '29', None, '1', '7.0mm', '66']

To get the weather parameters for the following five days, I had to change the date in the list “params” to collect the information. I tested out the following days to ensure that there is information for the next five days and the results have shown that the weather parameters are only able to reach till the third day. So, a change I made along the way was to shorten the display of the number of days. Instead of six days, it will be three days: the current day, the day after, and the day after tomorrow.

Since the extract time of day is crucial for this code as the purpose of the GUI is to display the weather parameters of the current day and the upcoming days, I created a function to collect the time and the date to easily put the right information into the API to get the weather parameters. In this list “params” requires the date in the order of year, month, and date:

```
    import time
    #shows the current time
    ex) Current Time: Wed Jul  3 17:05:28 2024
    print("Current Time:", time.ctime())

    #converts the time.ctime into a list
    ex) ['Wed', 'Jul', '3', '17:05:28', '2024']
    now = time.ctime().split()
    print(now)



    #the year, month, and day are collected from the list “now”
```

```
      #year, month, day
      year = now[4]
      month = now[1]
      day = now[2]

      #time of day is collected from the list "now"
      #time of day
      clock = now[3][0:2]

#base_date format -> ex) 20240614
      def genBaseDateCode():
         code = year

         mon2num = {'Jan': "01", "Feb":"02", "Mar":"03", "Apr":"04",
"May":"05", "Jun":"06", "Jul":"07", "Aug":"08", "Sep":"09", "Oct":"10",
"Nov":"11", "Dec":"12"}
         code += mon2num[month]
         code += day
         return code
      #the function genBaseDateCode is in the variable dateCode for easier
useage
      dateCode = genBaseDateCode()
```

With the function "genBaseDateCode" it becomes very convenient to collect the weather parameters of the current day, the day after, and the day after tomorrow.

The same went on with the air quality information using the API from Air Korea. The given code with personal information and the function "genBaseDateCode" applied looks like the code below:

```
#AIR QUALITY
dateCode = dateCode[:4]+'-'+dateCode[4:6]+'-'+dateCode[6:]

#Today and Tomorrow data (Bad / Normal / Good)
url =
'http://apis.data.go.kr/B552584/ArpltnInforInqireSvc/getMinuDustFrcstDspth'
```

```
key='Ts+NMfEsOgySQHW4+bWRkZP21N8vBTl4bqCkgyB8uzQMES9dcxmiADWp5qopwg9j0CqzjRWst+
RYPI6ElYSbiA=='
params ={'serviceKey' : key, 'returnType' : 'JSON', 'numOfRows' : '100',
'pageNo' : '1', 'searchDate':dateCode}

response = requests.get(url, params=params) #there is a limit to how much data
can be collected
res_json = json.loads(response.content)
items = res_json['response']['body']['items']

result = dict()
for row in items:
   raw = row['informGrade'].split(',')
   for pos in raw:
       if '인천 :' in pos and row['informCode'] == 'PM10':
           result[row['informData']] = pos[5:]
           break

result = list(result.values())
#current day
day0[2] = result[0]
#the day after
day1[2] = result[1]
#the day after tomorrow (this API wasn't able to predict the air quality for
the day after tomorrow)
day2[2] = "N/A"
```

A difference from the weather API was that the furthest information the air quality API could get was tomorrow. So, since most of the information for the day after tomorrow is gathered from the weather API, the air quality API just had to be not applicable (N/A) for the third day.

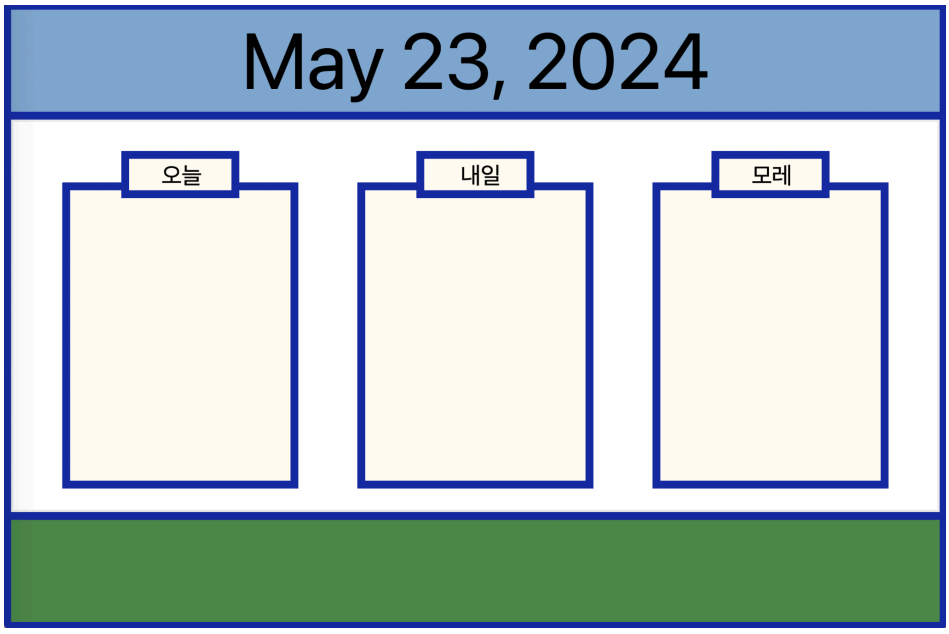
After the weather parameters and the air quality information had been collected by using the API, I worked on the GUI. It didn't take much time since I already had a basic sketch of how I wanted the GUI to look. However, now that I know what information is available and what is not, there have to be a few things to be fixed on my sketch.

First, I had to change the number of days the weather parameters will show. Instead of six like the sketch, I can only display three. I have been using certain weather and air quality APIs to gather data. However, I have encountered a limitation: these APIs only provided weather parameters and air quality information for today, tomorrow, and the day after tomorrow which is only three days. So, I made the section for each day was horizontally longer.

Second, is the titles that are in each day on the top. I drew some cloud-shaped titles on the top for each section of the day in the sketch, however, I figured out that Tkinter is not that flexible in forming unique shapes. I would have to draw it and load it as an image on a label in the GUI, which will create a frame around the shape, making it difficult to create a cloud shape title which is why instead of doing this design I changed it to a rectangle shape and add the rectangle shape title instead.

Lastly, I decided to have most of the data displayed in Korean. Since I decided to create a weather display for a city in Korea and the API I'm using is from Korea, it felt right to make the information in Korean instead of English. The weather parameters that didn't give numerical data and instead had words were said in Korean and it would be difficult to translate every word that was used in the API.

After making some changes to my plan, I continued with the overall design. It all first began with the simple shapes which were easily created with frames in Tkinter Python which divided the page into sections where I would place certain things at those areas like the top section where the date goes:

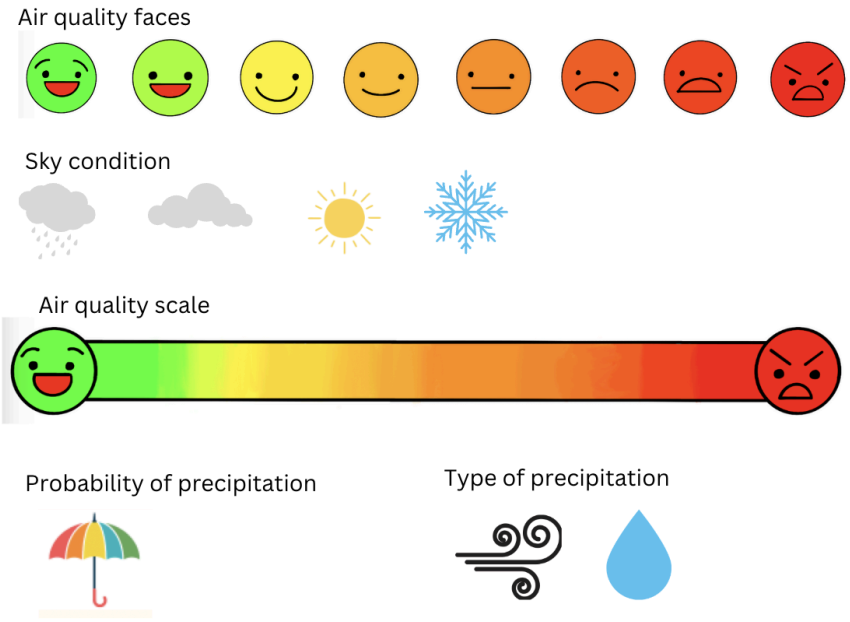


The following code that was programmed for the display above looked like this:

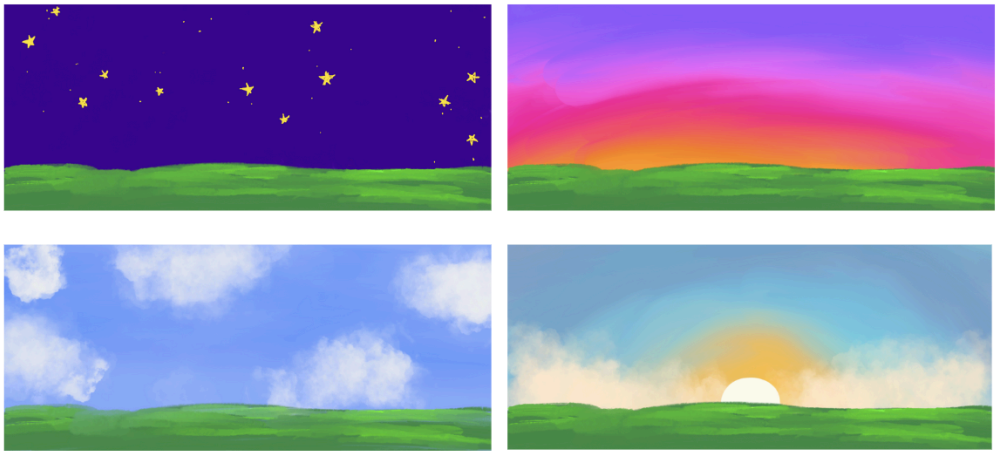
```
from tkinter import*

#GUI
window = Tk()
window.title('weather page')
window.geometry("1200x800")

#colors
outlinecolor = "#152CA2"
sky_color = "#7EA6CD"
grass_color = "#4B884A"
minside = "#FFFCF3"

#Outer territories
outline_bg = Frame(window, width = 1200, height = 800,bg = outlinecolor)
outline_bg.place(x=0,y=0)

#date banner
Date_bg = Frame(window, width=1180, height=130, bg= sky_color)
Date_bg.place(x= 10, y= 10)

datenumber = Label(window,text = "May 23, 2024", font=("Ariel", 100),bg = sky_color,
fg = "black")
datenumber.place(x = 300, y = 10)

#main banner
Main_bg = Frame(window, width = 1180, height = 500, bg= sky_color)
Main_bg.place(x=10, y=150)

C = Canvas(Main_bg, bg="white", width = 1174, height = 494)
C.place(x=0, y = 0)


def DAY1():
   global tempImg, airImg, mistImg, windImg

   page1 = Frame(window, width=300, height=390, bg=minside,
highlightbackground=outlinecolor, highlightthickness=10)
   page1.place(x=75, y=230)
```

```
   page1L = Frame(window, width=150, height=60, bg="#FFFCF3",
highlightbackground=outlinecolor, highlightthickness=10)
   page1L.place(x=150, y=190)

   pg1L = Label(page1L, bg=minside, font=("Arial", 30))
   pg1L.place(x=30, y=0, width=75, height=40)
   pg1L["text"] = "오늘"

def DAY2():
   page2 = Frame(window, width = 300, height = 390, bg = "#FFFCF3",
highlightbackground = outlinecolor, highlightthickness= 10)
   page2.place(x=450, y = 230)

   page2L = Frame(window, width=150, height=60, bg="#FFFCF3", highlightbackground =
outlinecolor, highlightthickness= 10)
   page2L.place(x= 525, y=190)

   pg2L = Label(page2L, bg = minside, font = ("Arial",30))
   pg2L.place(x = 35 , y = 0, width = 75, height = 40)
   pg2L["text"] = "내일"

def DAY3():
   page3 = Frame(window, width = 300, height = 390, bg = "#FFFCF3",
highlightbackground = outlinecolor, highlightthickness= 10)
   page3.place(x=825, y = 230)

   page3L = Frame(window, width=150, height=60, bg="#FFFCF3",
highlightbackground=outlinecolor, highlightthickness=10)
   page3L.place(x=900, y=190)

   pg3L = Label(page3L, bg=minside, font=("Arial", 30))
   pg3L.place(x=30, y=0, width=75, height=40)
   pg3L["text"] = "모레"

DAY1()
DAY2()
DAY3()

#air quality scale banner
Airquality_bg = Frame(window, width = 1180, height =130, bg = grass_color)
Airquality_bg.place(x= 10, y= 660)
window.mainloop()
```

After the basic frames were made, I started on the icons which were mostly drawn on my iPad (and the rest on Canva). The icons that were needed were the air quality faces, the different weather conditions, and icons that can represent sky conditions, probability of precipitation, and type of precipitation:



Then, the background of the GUI had to be created. The sketch includes a background that has a landscape with a blue sky with clouds. It needed to change the sky based on the time of day. So, from 6:00 to 11:59 the sky will be the color when the sun rises, from 12:00 to 16:59 the background will be blue, from 17:00 to 18:59 the sky will be the color when the sun goes down, and from 19:00 to 5:59 the sky will the color when it's night time:



Since the weather parameters for each day have a list with all the information, all that had to be done was moving the information into the labels created with Tkinter Python. However, some had to be displayed in icons. For example, the weather's temperature data just had to be displayed in Celsius, which already did. However, for the air quality, the results can be very good, good, normal, bad, or very bad; and since the data is from an API made in Korea, the results are in Korean: 아주 좋음, 좋음, 보통, 나쁨, or 매우 나쁨. The air quality needs a label and an icon representing the condition. Because the label is in words and the API is already in words, I just needed to write the index of where the air quality information was in the list that was created just like the temperature information.

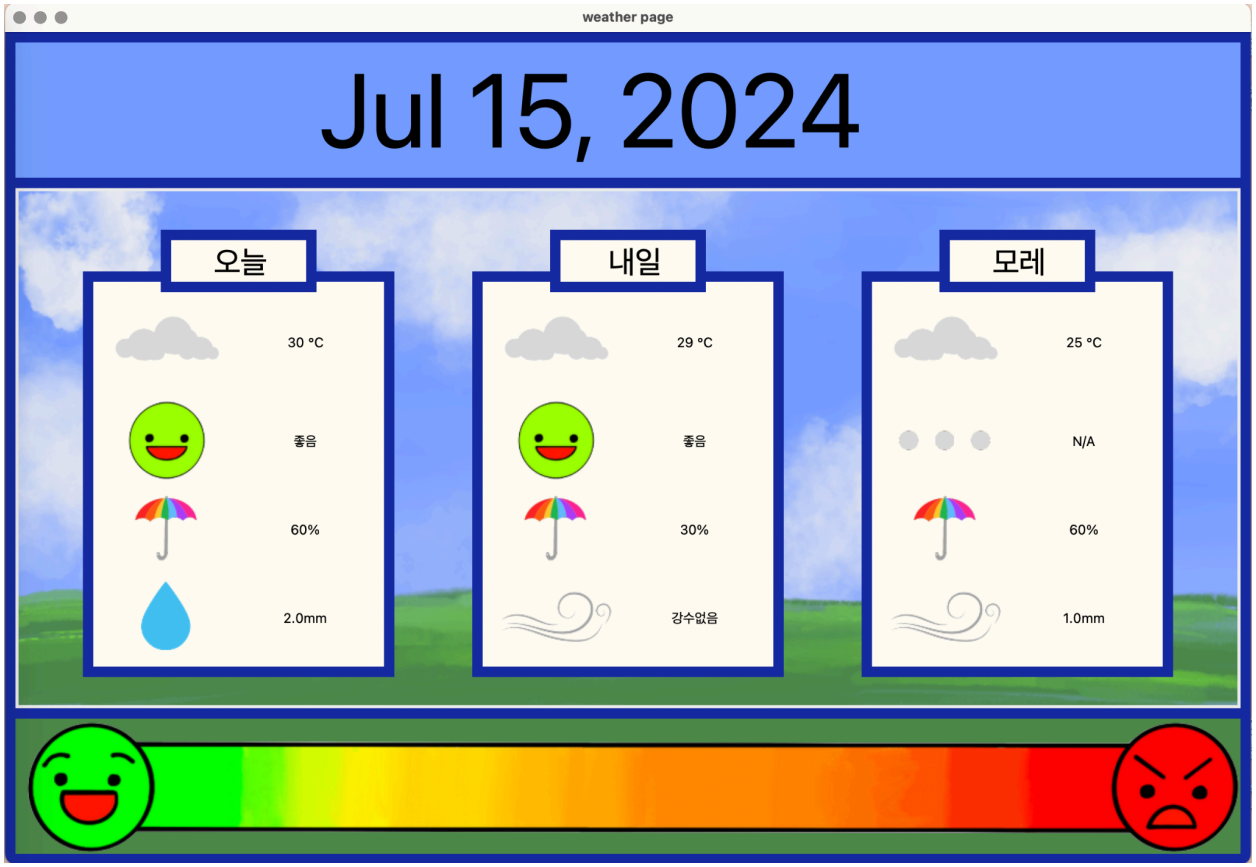
However, for the icons, they had to go through a process in a code to open the image so that it could be displayed in the GUI. To do this, instead of doing it one by one which would be a tedious process, I created separate lists for each weather parameter that I needed for organization and then loaded them in the order it was in the list. So, for the air quality icons, they were in the order of very good to terrible:

```
#Created a list for all the air quality face icons to be loaded in
airFaces = [] #verygood, good, normal, bad, terrible

#loading the image and inserting it into the list airFaces
for FACE in
['verygood.png','good.png','normal.png','bad.png','terrible.png']:
   FACE_img = Image.open("./weather_images/"+ FACE)
   FACE_img = ImageTk.PhotoImage(FACE_img)
   airFaces.append(FACE_img)

dataToIndex = {'아주 좋음':0, '좋음':1, '보통':2, '나쁨':3, '매우
나쁨':4}
```

Then I created a dictionary that scaled the air quality condition from zero to four instead of it being in words so that it is easier to select the image representing the condition based on the data. So when the result of the current day is 아주 좋음, which is very good, the numerical scale for very good will be zero:

```
    #creating a label for the air quality face icon for the day2.
    p2_air = Label(page2, bg= minside)
    #placing the label inside frame page2
    p2_air.place(x=15, y=115, width=110, height=75)
    #collecting the air quality information form the API and
selecting the icon based on how the air quality is from the airFaces
list.
    p2_air["image"] = airFaces[dataToIndex[day1[2]]]
```

This process was done on all the weather parameters resulting in the screenshot of the GUI below:



After finishing the weather page, I think that I have slightly reached the goal of understanding the use of API because I have understood what API I used in this project and made use of it to create the GUI. However, I think that I would need to explore this concept more as there are many different ways of how API is used and this is just one of the ways.

Work Cited

Jain, Sandeep. “What is an API (Application Programming Interface).” *GeeksforGeeks*, 16 February 2024, https://www.geeksforgeeks.org/what-is-an-api/. Accessed 26 June 2024.