```
# importing all packages used with the program

import requests
import json

import matplotlib.pyplot as plt
import numpy as np



def weather_api(api_url, location, graph_it):
    # changing the settings as the college network produces errors if not
    requests.adapters.HTTPAdapter(max_retries=5)
    '''verify=False does produce a security issue
     but due to the college security the program will not work without that change'''
    response_api = requests.get(
        api_url,
        verify=False)
    # strips the json to just cleaner text so it can easily be read
    data = response_api.text
    parse_json = json.loads(data)
    # breaks down the json further into just the needed bit
    hourly = parse_json['hourly']
    time = hourly['time']
    temp = hourly['temperature_2m']
    rain = hourly['rain']

    # removes un-needed text from the time list
    time = ([s[11:] for s in time])
    # finds the average temp for the day
    avg_temp = round(sum(temp) / len(temp), 2)
```

```
    if graph_it:
        # creation of graph
        title = f'average temperature of {location}'
        fig, ax = plt.subplots()
        ax.bar(time, temp)
        ax.set_ylabel('temperature (°c)')
        ax.set_title(title)
        plt.xticks(rotation=50)
        # stores it in the static folder so it can be accessed without having to pass a large piece of data
        plt.savefig('static/images/temp.png')

        # creation of graph
        title = f'predicted rainfall of {location}'
        fig, ax = plt.subplots()
        ax.bar(time, rain)
        ax.set_ylabel('rain (mm)')
        ax.set_title(title)
        plt.xticks(rotation=50)
        # stores it in the static folder so it can be accessed without having to pass a large piece of data
        plt.savefig('static/images/rain.png')

    return avg_temp



def air_quality_api(api_url, location):
    # changing the settings as the college network produces errors if not
    requests.adapters.HTTPAdapter(max_retries=5)
    '''verify=False does produce a security issue
     but due to the college security the program will not work without that change'''
    response_api = requests.get(
        api_url,
```

```
    verify=False)
# print(response_API.status_code)
data = response_api.text
parse_json = json.loads(data)
hourly = parse_json['hourly']

time = hourly['time']
time = ([s[11:] for s in time])

# breaks down the json further into just the needed bit
pm10 = hourly['pm10']
pm2_5 = hourly['pm2_5']
alder_pollen = hourly['alder_pollen']
birch_pollen = hourly['birch_pollen']
grass_pollen = hourly['grass_pollen']
olive_pollen = hourly['olive_pollen']

# removes none values which would create errors in list manipulation later
alder_pollen = [x for x in alder_pollen if x is not None]
birch_pollen = [x for x in birch_pollen if x is not None]
grass_pollen = [x for x in grass_pollen if x is not None]
olive_pollen = [x for x in olive_pollen if x is not None]

# if errors are still occurring these try statements will fix it
try:
    alder_average = round(sum(alder_pollen) / len(alder_pollen), 2)
except ZeroDivisionError:
    alder_average = 0
try:
    birch_average = round(sum(birch_pollen) / len(birch_pollen), 2)
except ZeroDivisionError:
```

```
    birch_average = 0
try:
    grass_average = round(sum(grass_pollen) / len(grass_pollen), 2)
except ZeroDivisionError:
    grass_average = 0
try:
    olive_average = round(sum(olive_pollen) / len(olive_pollen), 2)
except ZeroDivisionError:
    olive_average = 0

# replaces none values with a 0 and then converts them to float
pm10 = [str(i or 0) for i in pm10]
pm10 = [float(i) for i in pm10]

pm2_5 = [str(i or 0) for i in pm2_5]
pm2_5 = [float(i) for i in pm2_5]
# puts them into two lists so they can be passes more efficiently through the program
pollen_names = ['alder_pollen', 'birch_pollen', 'grass_pollen', 'olive_pollen']
pollen_averages = [alder_average, birch_average, grass_average, olive_average]
# creation of graph
x_axis = np.arange(len(time))
plt.figure(figsize=(17, 6))
plt.bar(x_axis - 0.2, pm10, 0.4, label='pm10')
plt.bar(x_axis + 0.2, pm2_5, 0.4, label='pm2.5')
plt.xticks(rotation=90)
plt.xticks(x_axis, time)
plt.xlabel("times")
plt.ylabel("amount of particles in the air (µg/m³)")
plt.title(f"amount of particles in the air of different type in {location}")
plt.tick_params(axis='x', which='major', labelsize=8)
plt.legend()
```

```
    # stores it in the static folder so it can be accessed without having to pass a large piece of data
    plt.savefig('static/images/air_quality.png')

    return pollen_names, pollen_averages


def convert_lat(location):
    # changing the settings as the college network produces errors if not
    requests.adapters.HTTPAdapter(max_retries=5)
    '''verify=False does produce a security issue
     but due to the college security the program will not work without that change'''
    response_api = requests.get(
        f'https://geocoding-api.open-meteo.com/v1/search?name={location}',
        verify=False)
    # strips the json to just cleaner text so it can easily be read
    data = response_api.text
    parse_json = json.loads(data)
    # breaks down the json further into just the needed bit
    results = parse_json['results']
    loc = results[0]
    lat = loc['latitude']
    long = loc['longitude']
    return lat, long


def password_checker(password):
    SpecialSym = ['$', '@', '#', '%']
    val = True

    if len(password) < 6:
        val = False
```

```
if not any(char.isdigit() for char in password):
    val = False

if not any(char.isupper() for char in password):
    val = False

if not any(char.islower() for char in password):
    val = False

if not any(char in SpecialSym for char in password):
    val = False

return val
```