

Criterion C: Development

Main.py

Importing Libraries

```
# ---LIBRARIES--- #  
  
import pathlib  
  
import wikipedia
```

The pathlib library is used to check if files exist in the program. This functionality is needed so that a new database can be generated whenever one does not exist.

The wikipedia library is used to search for specific wikipedia articles. This is needed as the website will link the wikipedia article for the city and display an image from the wikipedia article.

Reading CSV files

```
def getData(FILENAME):  
    ...  
  
    extract data from file and process into 2D arrays  
    :param FILENAME: (string)  
    :return: (list)  
    ...  
  
    FILE = open(FILENAME, encoding="ISO-8859-1")    # open file  
    RAW_DATA = FILE.readlines()    # separate the content of the file by lines  
    FILE.close()    # close the file  
  
    for i in range(len(RAW_DATA)): # for each line in the original file  
        RAW_DATA[i] = RAW_DATA[i][: -1] # remove the last character (\n)  
        RAW_DATA[i] = RAW_DATA[i].split(',')    # split the line by each comma (CSV)  
  
    return RAW_DATA
```

To read CSV files the program splits the content of the file by each line, then it iterates through these lines and splits them by commas (CSV). This function is used several times through the program to serve the need of reading CSV databases.

Adding the Climate information to the city database

```
def addClimate(CLIMATE_LIST, FULL_LIST):
    """
    Add the information from "climate.csv" to the full list of city information
    :param CLIMATE_LIST: (list) list containing climate information
    :param FULL_LIST: (list) full list of city information
    :return: (list) Updated list of city information
    """

    global CITY_LIST    # List of all cities in the city information database

    CLIMATE_LIST.pop(0)    # Remove the header of the table

    CITY = None

    for item in CLIMATE_LIST:    # for each item in the CLIMATE_LIST array
        if CITY != item[3]:    # if a new city has been reached in the data
            if CITY != None:    # if this is not the first time in the for loop
                if MONTHS == 0:
                    AVERAGE = None
                else:
                    AVERAGE = round(SUM/MONTHS,1)    # calculate the average temperature

                # Convert the temperatures values to Celcius
                LOW = convertToCelcius(LOW)
                HIGH = convertToCelcius(HIGH)
                AVERAGE = convertToCelcius(AVERAGE)

                CITY_INFORMATION = [CITY, None, COUNTRY, LOW, HIGH, AVERAGE, None, None, None, None]
                FULL_LIST.append(CITY_INFORMATION)    # Add the climate data to the list of city information
                CITY_LIST.append(CITY+"_"+COUNTRY)
            CITY = item[3]
            COUNTRY = item[1]

            if COUNTRY == "US":
                COUNTRY = "United States"

            LOW = None
            HIGH = None
            MONTHS = 0
            SUM = 0

            TEMPERATURE = float(item[7])
            if TEMPERATURE != -99.0:
                MONTHS = MONTHS + 1
                SUM = SUM + TEMPERATURE

            # remember the temperature as the highest if it is the highest so far
            if LOW == None or LOW > TEMPERATURE:
                LOW = TEMPERATURE

            # remember the temperature as the lowest if it is the lowest so far
            if HIGH == None or HIGH < TEMPERATURE:
                HIGH = TEMPERATURE

    return FULL_LIST
```

Three values are desired from the climate database: the Lowest Temperature, the Highest temperature, and the average temperature. To obtain these values the climate.csv file is converted into a two dimensional list. This list is iterated through, and these three values are calculated each time. When the program gets to a row about a new city, it then saves the three values into a list of city information and then resets the values. The program continues iterating through the list and repeating this pattern until it has traversed the entire climate list.

Add Population and Coordinates information to the city database

```
# add the data from the population and coordinates csv file to the full city array
def addPopCords(POP_CORDS_LIST, FULL_LIST):
    """
    Add the information from "PopulationAndCords.csv" to the full list of City information
    :param POP_CORDS_LIST: (list) list containing information from "PopulationAndCords.csv"
    :param FULL_LIST: (list) combined list of City information
    :return: (list) updated combined list of city information
    """

    global CITY_LIST

    POP_CORDS_LIST.pop(0) # remove the header
    for item in POP_CORDS_LIST:
        CITY = item[1][1:-1] # remove the quotes (""")
        try:
            POPULATION = int(item[9][1:-1]) # extract the population of a city
        except:
            POPULATION = None

        LATITUDE = item[2][1:-1] # extract the latitude
        LONGITUDE = item[3][1:-1] # extract the longitude
        PROVINCE = item[7][1:-1] # extract the province
        COUNTRY = item[4][1:-1] # extract the country
        CODE = CITY+"_"+COUNTRY

        if CODE in CITY_LIST: # if the is already in the full city database

            # add population and coordinates information
            INDEX = CITY_LIST.index(CODE)
            FULL_LIST[INDEX][1] = PROVINCE
            FULL_LIST[INDEX][6] = LATITUDE
            FULL_LIST[INDEX][7] = LONGITUDE
            FULL_LIST[INDEX][8] = POPULATION
        else:

            # create the city information list with the population and coordinates information
            INFORMATION = [CITY, PROVINCE, COUNTRY, None, None, None, LATITUDE, LONGITUDE, POPULATION, None]

            # add the city information list to the full city database
            FULL_LIST.append(INFORMATION)
            CITY_LIST.append(CODE)

    return FULL_LIST
```

To add the information from this database, it is again converted to a two dimensional array and then iterated over. On each item the program checks if the city is already in the city information list. If it is then the program just adds the data, if it doesn't then the program makes a new list to represent this city.

Add Top University rankings to the city database

```
# add the data from the university ranking csv file to the full city array
def addUniversities(UNIVERSITY_LIST, FULL_LIST):
    """
    Add the information from "CityUniversityRankings.csv" to the combined list of City information
    :param UNIVERSITY_LIST: (list) list containing information from "CityUniversityRankings.csv"
    :param FULL_LIST: (list) combined list of City information
    :return: (list) updated combined list of city information
    """

    print("university started")
    global CITY_LIST

    rank = 916
    for UNI in reversed(UNIVERSITY_LIST):
        UNI = UNI[0]
        if UNI in CITY_LIST:      # if the city has been recorded for the full city database
            INDEX = CITY_LIST.index(UNI)    # locate the city in the list
            FULL_LIST[INDEX][9] = rank      # add the university ranking to the city's information list
        else:
            UNI = UNI.split("_")
            if len(UNI) == 3:
                worse = UNI[0]+"_"+UNI[2]
                i = 0
                for CITY in CITY_LIST:
                    CODE = CITY.split("_")
                    #print(CODE)
                    if len(CODE) == 3:
                        CODE = CODE[0] + "_" + CODE[2]
                        if worse == CODE: # if the city has been recorded for the full city database
                            FULL_LIST[i][9] = rank # add the university ranking to the city's information list
                        i = i + 1

            rank = rank - 1    # Count backwards so that higher ranks will overwrite lower ones

    return FULL_LIST
```

To add the top university ranking, the program iterates backwards through the database after it is converted to a two dimensional list. It iterates backwards so that higher ranks overwrite the lower lower ranks instead of the opposite. This allows for cities with multiple universities in the database to be attached to the highest ranking one and not the lowest ranking.

Add a unique code to each city

```
# add a unique code representing the city to the full city array
def addCode(FULL_LIST):
    """
    Add the unique codes of each city to the city information list of each city
    :param FULL_LIST: (list) 2d list of all city information
    :return: (list) updated list of city information
    """
    for i in range(len(FULL_LIST)):

        # set up the items of the unique code for the city
        CITY = FULL_LIST[i][0]
        while " " in CITY:
            CITY = CITY[0:CITY.index(" ")] + CITY[CITY.index(" ")+1:len(CITY)]
        if FULL_LIST[i][1] != None:
            PROV = FULL_LIST[i][1][0:3]
        else:
            PROV = ""
        if " " in FULL_LIST[i][2]:
            COUN = FULL_LIST[i][2][0]+FULL_LIST[i][2][FULL_LIST[i][2].index(" ")+1]
        else:
            COUN = FULL_LIST[i][2][0:3]

        # add the city code to the end of the city information list
        FULL_LIST[i].append(CITY+PROV+COUN)
    return FULL_LIST
```

Convert the city information list into a CSV file

```
# write the contents of the full city array to a CSV file
def createFullCityInfoCSVFile():
    """
    Create a CSV file of the combined city information
    :return:
    """

    global FULL_CITY_CSV
    global FULL_LIST

    SAVED_FILE = open(FULL_CITY_CSV, "w", encoding="utf-8") # open the database
    SAVED_FILE.write("CITY, PROVINCE/STATE, COUNTRY, LOW TEMP, HIGH TEMP, AVERAGE TEMP, LATITUDE, LONGITUDE, POPULATION, UNIVERSITY RANKING, CODE\n")

    for i in FULL_LIST: # for each city in the in the list
        data = ""

        # connect the city information into a string separated by commas (CSV)
        for a in i:
            data = data + str(a) + ","
        data = data + "\n" # end the string with a new line
        SAVED_FILE.write(data) # write the string to the city information CSV file
    SAVED_FILE.close() # close the database
```

This is done to lower the loading time of opening the website. Creating a city information list every time would be inefficient.

Rank the list of cities

```
def rankCities(ELIGIBLE_CITIES, CRITERIA):
    # CRITERIA = [ [3,-30, 1] ]

    CITIES_SCORES = []
    for CITY in ELIGIBLE_CITIES:
        CITIES_SCORES.append([CITY[0], CITY[-2], 0])
        for ITEM in CRITERIA:
            if CITY[ITEM[0]] != "None":
                CITIES_SCORES[-1][2] = CITIES_SCORES[-1][2] + (abs(float(CITY[ITEM[0]]) - ITEM[1]) * ITEM[2])
            else:
                CITIES_SCORES.pop(-1)
                break

    CITIES_SCORES.sort(key=getDistance)

    LOWER = 0
    UPPER = 1
    MULTIPLIER = 2
    BOUNDARIES = []
    CITIES = []
    while UPPER < len(CITIES_SCORES):
        if UPPER != 1:
            LOWER = UPPER

        if MULTIPLIER == 2:
            MULTIPLIER = 5
        else:
            MULTIPLIER = 2

        UPPER = UPPER * MULTIPLIER

        BOUNDARIES.append("Top " + str(LOWER + 1) + " to " + str(UPPER))
        CITIES.append(CITIES_SCORES[LOWER:UPPER])
    return [BOUNDARIES,CITIES]
```

The program iterates through the cities and compares them with each ranking criteria. The differences between the ideal values and the city's true values are multiplied by their respective weightings and summed up. The list of these sums are ordered from lowest (cities closest to the ideals) and highest (cities farthest from the ideals)

Convert filters into a displayable form

```
# convert the list of filters into a readable form for displaying
def getSpecifications(FILTERS):
    """
    format the list of filters into something readable to display to the user
    :param FILTERS: (list) list of filters on the city data, added by the user
    :return: (list) list of filters in a readable format for displaying to the user
    """
    DISP_FILTERS = []

    TYPES = ["City", "Province/State", "Country", "Lowest Temperature (C°)", "Highest Temperature (C°)", "Average Temperature (C°)", "Latitude",

    for item in FILTERS:
        if item[0] == 9: # if the filter is about university ranking

            # add sentence about university ranking to the list
            DISP_FILTERS.append(TYPES[item[0]] + " is in the top " + str(item[2]))
        else:

            # add a general sentence to the list
            DISP_FILTERS.append(TYPES[item[0]] + " is between " + str(item[1]) + " and " + str(item[2]))

    return DISP_FILTERS
```

Filters are represented by three numbers: a number representing what is being filtered, the lower bound, and the upper bound. To convert this to something readable the first number is mapped to a list of strings where the each value of the first number corresponds to the index of the string that describes it in words

Arrange the Ranking Criteria in a way that is displayable to the user

```
# SUBROUTINES RELATED TO FILTERING CITIES
# convert the list of filters into a readable form for displaying
def getCriteria(CRITERIA):
    """
    format the list of criteria into something readable to display to the user
    :param CRITERIA: (list) list of criteria for the city data, added by the user
    :return: (list) list of criteria in a readable format for displaying to the user
    """
    DISP_CRITERIA = []

    TYPES = ["City", "Province/State", "Country", "lowest temperature", "highest temperature", "average temperature", "latitude", "Longitude", "p

    for item in CRITERIA:
        if item[0] == 3 or item[0] == 4 or item[0] == 5:
            unit = " C° "
        elif item[0] == 8:
            unit = " people "
        else:
            unit = " "
        # add sentence about each item in the criteria list
        DISP_CRITERIA.append("The ideal " + TYPES[item[0]] + " is " + str(item[1]) + unit + "(weighted at x" + str(item[2]) + ")")

    return DISP_CRITERIA
```

Filter out the cities that don't meet the filter to obtain a list of eligible cities

```
# obtain the list of cities that meet the parameters of the filter
def getEligibleCities(SPECIFICATIONS):
    """
    get the a list of eligible cities based on the user inputed filters on city data
    :param SPECIFICATIONS:
    :return: (list) list of cities that meet the filters set by the user
    """
    global FULL_LIST

    ELIGIBLE_CITIES = [] # setup list of eligible cities
    for CITY in FULL_LIST: # for each city in the list of cities
        ELIGIBLE = True # assume the city is eligible
        for SPEC in SPECIFICATIONS: # for each filter in the list of filters
            try:
                if CITY[SPEC[0]] != "None":
                    VALUE = float(CITY[SPEC[0]])
                    if SPEC[1] < VALUE < SPEC[2]:
                        pass
                    else:
                        ELIGIBLE = False # if the city does not meet the filter, mark it as not eligible
                        break
            except:
                pass

        if ELIGIBLE == True:
            ELIGIBLE_CITIES.append([CITY[0],CITY[2],CITY[-2]]) # if city remains eligible, add it to the eligible list
    return ELIGIBLE_CITIES
```

To determine eligibility each city is compared to each specification. If the city gets to the end of the specifications without failing one, it is added to the list of eligible cities

Organize the list of eligible cities into groups of cities that share a country

```
# organize the cities into groups of countries for displaying
def getDisplay(ELIGIBLE_CITIES):
    """
    organize the list of eligible cities into groups of cities that share the same country
    :param ELIGIBLE_CITIES: (list) list of the cities that meet the filters set by the user
    :return: (list) two dimensional array containing a list of counties and a list with the corresponding cities
    """
    COUNTRIES = []
    CITIES = []

    for city in ELIGIBLE_CITIES: # for each city in the list of eligible cities
        if city[1] in COUNTRIES: # if the loop has already encounterd this country
            CITIES[COUNTRIES.index(city[1])].append([city[0],city[2]]) # add the city to a list in the CITY list at the same index as the count
        else:
            COUNTRIES.append(city[1]) # add the country to the end of the countries list
            CITIES.append([city[0],city[2]]) # add the city into a list at the end of the cities list
    return [COUNTRIES, CITIES]
```


Read the URL and convert the information into an array representing the ranking criteria and another representing the filter boundaries

```
def decryptCode(CODE):

    CRITERIA    = []
    FILTER      = []

    if "::" in CODE:
        RAW = CODE.split("::")
        CRITERIA = decryptRaw(RAW[0].split(":"))
        FILTER   = decryptRaw(RAW[1].split(":"))
    else:
        RAW = CODE[4:len(CODE)]
        if "FLT:" in CODE:
            FILTER      = decryptRaw( RAW.split(":") )
        else:
            CRITERIA    = decryptRaw( RAW.split(":") )

    return [CRITERIA, FILTER]

def decryptRaw(RAW):
    CLEAN = []
    for i in RAW:
        NAME = i[0:2]
        if NAME == "LT":
            SELECTION = 3
        elif NAME == "HT":
            SELECTION = 4
        elif NAME == "AT":
            SELECTION = 5
        elif NAME == "PP":
            SELECTION = 8
        else:
            SELECTION = 9

        NUMBERS = i[2:len(i)]
        NUMBERS = NUMBERS.split("_")
        NUM1 = float(NUMBERS[0])
        NUM2 = float(NUMBERS[1])

        CLEAN.append([SELECTION, NUM1, NUM2])
    return CLEAN
```

Get the city information by the city's unique code

```
# Locate the index of a city by it's ID
def getCity(CODE):
    """
    obtain the city information list of a city using it's Code
    :param CODE: (str) unique Code which represents each city
    :return: (list) list of city information
    """
    global FULL_LIST

    INLIST = False

    INDEX = 0

    # Loop through the entire city list until the code matches
    for i in FULL_LIST:
        if i[-2] == CODE:
            # if the code matches, break from the loop and remember that the code is indeed in the city information list
            INLIST = True
            break
        INDEX = INDEX + 1

    # if the program remembers that the code was in the city information list
    if INLIST == True:
        print(FULL_LIST[INDEX])
        return FULL_LIST[INDEX]
    else:
        return None
```

Arrange a city's information into a paragraph

```
# arrange the information from a city in the city array into a paragraph for displaying
def getCityInformation(CITYINFO):
    """
    print the available information about a city to the user
    :param FULL_LIST: (list) combined list of City information
    :return: (str) paragraph on the city
    """

    if CITYINFO != None:

        CITY = CITYINFO[0]
        PROVINCE = CITYINFO[1]
        COUNTRY = CITYINFO[2]
        LOW = CITYINFO[3]
        HIGH = CITYINFO[4]
        AVERAGE = CITYINFO[5]
        LATITUDE = CITYINFO[6]
        LONGITUDE = CITYINFO[7]
        POPULATION = CITYINFO[8]
        UNIVERSITY = CITYINFO[9]

        # add the suffix to the top university's rank
        if UNIVERSITY[-1] == "1":
            UNIVERSITY = UNIVERSITY + "st"
        elif UNIVERSITY[-1] == "2":
            UNIVERSITY = UNIVERSITY + "nd"
        else:
            UNIVERSITY = UNIVERSITY + "th"

        # add the geographic location of the city to the paragraph
        SENTENCE = "\n" + CITY + " is in "
        if PROVINCE != "None":
            SENTENCE = SENTENCE + PROVINCE + ", " + COUNTRY + ".\n"
        else:
            SENTENCE = SENTENCE + " " + COUNTRY + ".\n"

        # add the coordinates of the city to the paragraph
        if LATITUDE != "None":
            SENTENCE = SENTENCE + "The city's coordinates are latitude: " + LATITUDE + " , longitude: " + LONGITUDE + ".\n"

        # add the temperature information of the city to the paragraph
        if LOW != "None":
            SENTENCE = SENTENCE + "The temperature can get as low as " + LOW + " C and as high as " + HIGH + " C.\n"
            SENTENCE = SENTENCE + "The average yearly temperature is " + AVERAGE + " C.\n"

        # add the population of the city to the paragraph
        if POPULATION != "None":
            SENTENCE = SENTENCE + "There are around " + POPULATION + " people in the city.\n"

        # add the top university rank to the paragraph
        if UNIVERSITY[0:-2] != "None":
            SENTENCE = SENTENCE + "The city's top university is ranked " + UNIVERSITY + " in the world."

        return SENTENCE
    else:
        # error message
        return "Sorry! We have no information on this city."
```

As not all cities have an equal amount of information, the paragraph is broken into segments. Each segment is added to the paragraph if the city has the information it requires.

Convert to celsius

```
# convert Fahrenheit (useless unit) to Celcius
def convertToCelcius(Fahrenheit):
    """
    Conver fahrenheit to a better unit of measurement (celcius)
    :param Fahrenheit: (float) temperature in fahrenheit
    :return: (float) temperature in celcius
    """

    CELCIUS = round(((Fahrenheit - 32) * 5) / 9, 1)    # fahrenheit to celcius conversion
    return CELCIUS
```

Find a Wikipedia article

```
# search wikipedia with the search query
RESULT = wikipedia.search(QUERY)
```

This is how to search wikipedia. The function returns a list of results.

```
if len(RESULT) > 0:    # if the search yielded results
    PAGE = wikipedia.page(RESULT[0]).html()
    URL = wikipedia.page(RESULT[0]).url    # get the top result's url
```

This is how a specific page or url is obtained.

Get an image from wikipedia

```
TERMS = ["montage", "at_night", "downtown", "skyline", "business", "hall", "aerial", "bridge", "stadium", "street", "mount", "lake"]
if len(RESULT) > 0:    # if the search yielded results
    PAGE = wikipedia.page(RESULT[0])
    URL = wikipedia.page(RESULT[0]).url    # get the top result's url
    if len(PAGE.images) > 0:
        for index in range(len(PAGE.images)):
            x = PAGE.images[index].lower()

            for TERM in TERMS:
                if TERM in x:
                    MONTAGE.append(PAGE.images[index])
                    break
```

The wikipedia api has a dot function `.images` which returns a list of the addresses of images which appear in a page. This program iterates through these addresses and stores the ones which certain key terms appear.

Get information from Wikipedia (full function)

```
# get the wikipedia page and an image from the wikipedia page of a city
def getWikipedia(CITY):
    """
    retrieve wikipedia page on the city and the first image in it
    :param city: (list) list of city information
    :return: (list) list containing the link to the wikipedia page and the link to the first image
    """
    URL = "None"
    MONTAGE = []

    # create the search query
    if CITY[0] == CITY[1] or CITY[1] == "None":
        QUERY = CITY[0] + ", " + CITY[2]
    else:
        QUERY = CITY[0] + ", " + CITY[1] + " " + CITY[2]
    print("Query: " + QUERY)

    try:
        # search wikipedia with the search query
        RESULT = wikipedia.search(QUERY)

        TERMS = ["montage", "at_night", "downtown", "skyline", "business", "hall", "aerial", "bridge", "stadium", "street", "mount", "lake"]
        if len(RESULT) > 0: # if the search yielded results
            PAGE = wikipedia.page(RESULT[0])
            URL = wikipedia.page(RESULT[0]).url # get the top result's url
            if len(PAGE.images) > 0:
                for index in range(len(PAGE.images)):
                    x = PAGE.images[index].lower()

                    for TERM in TERMS:
                        if TERM in x:
                            MONTAGE.append(PAGE.images[index])
                            break

    except:
        pass
    print(MONTAGE)
    print(URL)
    return [URL, MONTAGE]
```

Guess the URL for a country's image on countryflags.com

```
# get the flag of the country of a specific city
def getFlag(CITY):
    """
    get the flag of the country of a city
    :param city: (list) city information list
    :return:
    """
    try:
        COUNTRY = CITY[2].lower()
        if COUNTRY == "united states": # united states has an exception as it is represented differently in the website
            FLAG = "https://cdn.countryflags.com/thumbs/united-states-of-america/flag-800.png"
        else:
            # replace any spaces in the country name with dashes
            while " " in COUNTRY:
                COUNTRY = COUNTRY[:COUNTRY.index(" ")] + "-" + COUNTRY[COUNTRY.index(" ")+1:len(COUNTRY)]
            FLAG = "https://cdn.countryflags.com/thumbs/" + COUNTRY + "/flag-800.png"
    except:
        FLAG = "None"
    print(FLAG)
    return FLAG
```

The countryflags website has a fairly consistent method of naming images which means the program can apply this method to most countries and receive the correct URL for the country's flag. This method is superior to other ways of obtaining a country's flag as it does not require interacting with another website or downloading all the flags. Interacting with another system is often bit rate limited and can slow the website down by several seconds. Downloading all the flags would be a faster system but would be extremely memory intensive. Guessing the countryflags URLs is less accurate but is worth it overall.

Check if a file exists

```
FIRST_RUN = True
if (pathlib.Path.cwd() / FULL_CITY_CSV).exists():
    FIRST_RUN = False
```

This is how the library path pathlib checks if a file exists.

Get City Information

```
# VARIABLES

CLIMATE_CSV      = "databases/climate.csv"
POP_CORDS_CSV    = "databases/PopulationAndCords.csv"
FULL_CITY_CSV    = "databases/fullCityInfo.csv"
UNIVERSITY_CSV   = "databases/CityUniversityRankings.csv"

FULL_LIST = []
CITY_LIST = []

# Get Data
FIRST_RUN = True
if (pathlib.Path.cwd() / FULL_CITY_CSV).exists():
    FIRST_RUN = False

if FIRST_RUN == True:

    # Create a combined array of city information
    CLIMATE_LIST = getData(CLIMATE_CSV) # convert the climate.csv to a list
    POP_CORDS_LIST = getData(POP_CORDS_CSV) # convert the PopulationAndCords.csv to a list
    UNIVERSITY_LIST = getData(UNIVERSITY_CSV) # convert the CityUniversityRankings.csv to a list
    FULL_LIST = addClimate(CLIMATE_LIST, FULL_LIST) # add data from CLIMATE_LIST list to the full city list
    del CLIMATE_LIST # remove the CLIMATE_LIST list from memory
    FULL_LIST = addPopCords(POP_CORDS_LIST, FULL_LIST) # add data from POP_CORDS_LIST list to full city list
    del POP_CORDS_LIST # remove the POP_CORDS_LIST list from memory
    FULL_LIST = addUniversities(UNIVERSITY_LIST, FULL_LIST) # add data from UNIVERSITY_LIST list to full city list
    del UNIVERSITY_LIST # remove the UNIVERSITY_LIST list from memory
    FULL_LIST = addCode(FULL_LIST) # add the unique city code to each city
    createFullCityInfoCSVFile() # Save the full city information list in CSV format
    # so that the data does not have to be compiled every time the website is run
else:
    # Get city information from the saved CSV file
    FULL_LIST = getData(FULL_CITY_CSV) # convert the fullCityInfo.csv to a list
    FULL_LIST.pop(0) # remove the header
```

If the FULL_CITY.csv file does not exist, then arrange and create the FULL_CITY.csv file. If the file does exist, then convert the csv file to a two dimensional array.

website.py

Import libraries

```
from flask import Flask, render_template, request, redirect, url_for
import main
```

The flask library is used to control the webpage and interact with html

The program imports main so that it can use the functions from main.

Redirecting the home page to /filter/0

```
@app.route('/', methods=["POST", "GET"])
def index():

    return redirect(url_for('filter', id="0"))
```

Control the main page

```
@app.route('/filter/<id>', methods=["POST", "GET"])
def filter(id):

    global FILTERS

    if id == "0":
        print("None")
        FILTERS = []
        CRITERIA = []
    else:
        DECRYPTED = main.decryptCode(id)
        CRITERIA = DECRYPTED[0]
        FILTERS = DECRYPTED[1]
        print(CRITERIA)
        print(FILTERS)

    ELIGIBLE = main.getEligibleCities(FILTERS)

    if not CRITERIA:
        DISPLAY = main.getDisplay(ELIGIBLE)
    else:
        DISPLAY = main.rankCities(ELIGIBLE, CRITERIA)

    DISPLAY_CRITERIA = main.getCriteria(CRITERIA)

    DISPLAY_FILTERS = main.getSpecifications(FILTERS)
    COUNTRIES = DISPLAY[0]
    CITIES = DISPLAY[1]

    print(FILTERS)

    return render_template('index.html', code=id, countries=COUNTRIES, cities=CITIES, filters=DISPLAY_FILTERS, rankings=DISPLAY_CRITERIA,)
```

The functions from main.py are referenced to get the displayable list of cities, filters, and rankings

Remove a filter or ranking criteria

```
def remove(code,type, id):
    id = int(id)
    if "RNK:" in code or "FLT:" in code :
        other = None
        code = code[4:len(code)]
    else:
        code = code.split("::")
        if type == "R":
            other = code[1]
            code = code[0]
        else:
            other = code[0]
            code = code[1]
    changed = code.split(":")
    changed.pop(id)
    changed = ":".join(changed)

    if other != None and changed != "":
        if type == "R":
            newcode = changed+"::"+other
        else:
            newcode = other+"::"+changed
    elif changed != "":
        if type == "R":
            newcode = "RNK:"+changed
        else:
            newcode = "FLT:"+changed
    elif other != None:
        if type == "F":
            newcode = "RNK:"+other
        else:
            newcode = "FLT:"+other
    else:
        newcode = "0"

    print("Removed Filter! New Code is " + newcode)
    return redirect(url_for('filter',id=newcode))
```

The program slices the “code” which represents the filters and criteria. It then removes the filter or criteria that is defined by the type (Filter to Criteria) and id (the index of the item to remove with the type). The program then reconstructs the code without the removed item.

Control the City page

```
@app.route("/city/<id>")
def city(id):
    CITY = main.getCity(id) # get the city information list
    paragraph = main.getCityInformation(CITY) # information in the city list in paragraph form
    TYPES = ["City", "Province/State", "Country", "Lowest Temperature (C°)", "Highest Temperature (C°)", "Average Temperature (C°)", "Latitude", "Longitude"]
    WIKI = main.getWikipedia(CITY)
    PAGE = WIKI[0] # get wikipedia page
    MONTAGE = WIKI[1] # get wikipedia's "montage" of the city

    FLAG = main.getFlag(CITY) # get the url of the flag of the city's country

    # get the Latitude and Longitude of the city
    if CITY[6] != "None":
        LATITUDE = CITY[6]
        LONGITUDE = CITY[7]
    else:
        LATITUDE = "None"
        LONGITUDE = "None"

    return render_template('city.html',LONGITUDE=LONGITUDE, LATITUDE=LATITUDE, types=TYPES, city=CITY,wikipedia=PAGE, paragraph=paragraph, flag=FLAG)
```

Use the functions from main.py to receive information about the city and send it to the city.html page

Run the Flask web app

```
# run the app
if __name__ == "__main__":
    app.run(debug=True)
```

index.html

Display the list of cities

```
<list>
  <h3>Elegible Cities</h3>
  {% for catagory in catagories %}
    <h5>{{catagory}}</h5>
    {% for city in cities[loop.index0] %}
      <div style="display: inline-block">
        <a href="/city/{{city[1]}}"> {{city[0]}}</a>
      </div>
    {% endfor %}
  {% endfor %}
<div style="height: 50px">

</div>
</list>
```

iterate through each category and for each category write each city in the category to the html page

Getting input from the webpage

```
<h3 style="width: 100px; display: inline-block">Ranking</h3>

<clickable onclick="document.getElementById('formR').style.display = 'block';"></clickable>

<div id = "formR">
  <input type="radio" id="lowtempR" name="selectionR" value=3 onclick="updateSelection('R');">
  <label for="lowtempR">Lowest Temperature</label><br>

  <input type="radio" id="hightempR" name="selectionR" value=4 onclick="updateSelection('R');">
  <label for="hightempR">Highest Temperature</label><br>

  <input type="radio" id="avetempR" name="selectionR" value=5 onclick="updateSelection('R');">
  <label for="avetempR">Average Temperature</label><br>

  <input type="radio" id="popR" name="selectionR" value=8 onclick="updateSelection('R');">
  <label for="popR">Population</label><br>

  <input type="radio" id="unirankR" name="selectionR" value=9 onclick="updateSelection('R');">
  <label for="unirankR">University Rank</label><br>

  <p id="num1labelR" >Ideal</p>
  <p><input type="number" id="num1R" name="num1R"/></p>



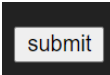
  <p id="num2labelR" >Weight</p>
  <p><input type="number" id="num2R" name="num2R"/></p>

  <clickable style="display: inline-block" onclick='updatePage1("R");'>submit</clickable>
  <clickable style="display: inline-block" onclick="document.getElementById('formR').style.display = 'none';">cancel</clickable>
</div>
```

This division has the `id = "formR"` so when javascript function `addFilter()` is run it shows on the webpage. The user may interact with several input types as explained in this table:

Each input has a type, id, name, and value

type	The type is written like <code>type="radio"</code> controls what kind of input the user is entering and how it appears on the webpage.
id	The id of the inputs as written like <code>id="lowtemp"</code> this allows the input to be referenced again in the HTML which is used to add labels (text next to) to the inputs like: <code><label for="lowtemp">Lowest Temperature (C°)</label>
</code>
name	The name is written like <code>name="selection"</code> . This is how the python script website.py receives refererrs to the these inputs like this <code>SELECTION = request.form["selection"]</code>
value	The value is written like <code>value=3</code> . The value is what is sent to the python script website.py. For the type numbers the value is controlled by the user.

type	Explanation
radio	The radio input looks like this  Lowest Temperature (C°). It allows the user to choose one from a set by clicking on one item of a list. This type was chosen because only one filter can be added at a time and it is convenient for the user to click on an item in a list as compared to typing out the name of the filter or entering a number that relates to the filter.
number	The number input looks like  . It allows the user to enter an integer.
submit	The submit input looks like  it is a button that sends the forum to the python script website.py when clicked

city.html

Embed a google map in the page

```
{% if LATITUDE != "None" %}
    <div id="map" style="width: 45%; height: 60%; margin-bottom: 1%;margin-right: 1%; display: inline-block;"></div>

    <script type="text/javascript">
        var myOptions =
        {
            zoom: 12,
            center: new google.maps.LatLng({{LATITUDE}}, {{LONGITUDE}}),
            mapTypeId: google.maps.MapTypeId.ROADMAP
        };

        var map = new google.maps.Map(document.getElementById("map"), myOptions);
    </script>
{% endif %}
```

Display a table in the page

```
<table style="width: 45%; height: 60%; overflow:scroll; border: solid; display: inline-block; margin:0;">
    {% for info in city %}
    <tr>
        <td>{{types[loop.index0]}}</td>
        <td>{{info}}</td>
    </tr>
    {% endfor %}
</table>
```

<tr> represents a table row and <td> represents a cell in the row

Link to Google Maps

```
<a href = "https://www.google.com/maps/search/?api=1&query={{city[0]}}%2C{{city[1]}}%2C{{city[2]}}">Google Maps</a>
```

Display images sent from website.py

```
<filter>
    {% if montage|length > 0 %}
        {% for image in montage %}
            
        {% endfor %}
    {% elif flag != "None" %}
        
    {% endif %}
</filter>
```

website.js

Get the item that is checked in the forum

```
function updateSelection(type)
{
    var selection = document.getElementsByName("selection" + type);

    if (selection[0].checked == true)
    {
        displayTemp(type);
        selectionText = "LT";
    }
    else if (selection[1].checked == true)
    {
        displayTemp(type);
        selectionText = "HT";
    }
    else if (selection[2].checked == true)
    {
        displayTemp(type);
        selectionText = "AT";
    }
    else if (selection[3].checked == true)
    {
        displayPop(type);
        selectionText = "PP";
    }
    else if (selection[4].checked == true)
    {
        displayUni(type);
        selectionText = "UR";
    }
}
```

Open a new page with a new filter or criteria

```
function updatePage1(type)
{
    document.getElementById('form' + type).style.display = 'none';

    var URL = window.location.href;

    URL = URL.split("/");
    URL = URL[URL.length - 1];

    if (URL != "0")
    {
        if (URL.includes("::"))
        {
            URL = URL.split("::");
            RCODE = URL[0];
            FCODE = URL[1];
        }
        else if (URL.includes("RNK:"))
        {
            RCODE = URL.slice(4, URL.length);
            FCODE = "";
        }
        else if (URL.includes("FLT:"))
        {
            RCODE = "";
            FCODE = URL.slice(4, URL.length);
        }
    }
    else
    {
        RCODE = "";
        FCODE = "";
    }

    updatePage2(type, RCODE, FCODE);
}
```

Define the URL code for the filters and criteria separately

```

function NewCode(CODE, type)
{
    var NEWCODE = "";
    if (CODE.includes(selectionText) == true)
    {
        var NEWCODE = "";
        var RAWFILTERS = CODE.split(":");
        var FILTERS = [];
        for (let i = 0; i < RAWFILTERS.length; i++)
        {
            if (RAWFILTERS[i].includes(selectionText) != true)
            {
                NEWCODE = NEWCODE + RAWFILTERS[i] + ":";
            }
        }
    }
    else
    {
        if (CODE != "")
        {
            NEWCODE = CODE + ":";
        }
    }

    var NUM1 = document.getElementById("num1" + type).value;
    var NUM2 = document.getElementById("num2" + type).value;

    if (NUM1 == "")
    {
        NUM1 = 1;
    }
    if (NUM2 == "")
    {
        NUM2 = 1;
    }

    if (parseInt(NUM1) > parseInt(NUM2) && type == "F")
    {
        alert("The Lower bound (" + String(NUM1) + ") cannot be higher than the Upper Bound(" + String(NUM2) + ")!");
        return "ERROR";
    }
    else if (selectionText == 0)
    {
        alert("Please select a filter!");
        return "ERROR";
    }
    else
    {
        NUM1 = String(NUM1);
        NUM2 = String(NUM2);
        NEWCODE = NEWCODE + selectionText + NUM1 + "_" + NUM2;
        return NEWCODE;
    }
}

```

Add the new criteria or filter to the code

```

function updatePage2(type, RCODE, FCODE)
{
    if (type == "R")
    {
        RCODE = NewCode(RCODE, type);
        if (RCODE != "ERROR")
        {
            if (FCODE.length == 0)
            {
                window.location.replace('/filter/RNK:' + RCODE);
            }
            else
            {
                window.location.replace('/filter/' + RCODE + "::" + FCODE);
            }
        }
    }
    else
    {
        FCODE = NewCode(FCODE, type);
        if (FCODE != "ERROR")
        {
            if (RCODE.length == 0)
            {
                window.location.replace('/filter/FLT:' + FCODE);
            }
            else
            {
                window.location.replace('/filter/' + RCODE + "::" + FCODE);
            }
        }
    }
}

```

Open a new page with the new filter or criteria

style.css

Style for buttons

```
clickable
{
  background-color: #bb5555;
  margin: 5%;
  padding: 1%;
  color: white;
  cursor: pointer;

  -webkit-user-select: none;
  -moz-user-select: none;
  -ms-user-select: none;
  user-select: none;
}
clickable:hover
{
  background-color: #995555;
}
```

<code>background-color: #bb5555;</code>	fill colour of the button
<code>color: white;</code>	text colour
<code>cursor: pointer;</code>	changes the cursor to the pointer image
<code>-webkit-user-select: none;</code> <code>-moz-user-select: none;</code> <code>-ms-user-select: none;</code> <code>user-select: none;</code>	stops the user from selecting the text
<code>clickable:hover</code>	defines style when the cursor is hovering over the button

Style for the filter

```
filter
{
    font-size: 15px;
    position: absolute;
    display: block;
    overflow: scroll;
    left: 66%;
    width: 30%;
    padding: 2%;
    height: 100%;
    background-color: #222222;
    color: #eeeeee;
    float: right;
    margin: 0;
}
```

<code>overflow: scroll;</code>	Add scrolling when the content overflows
<code>left: 66%;</code> <code>width: 30%;</code>	Definite the position
<code>font-size: 15px;</code>	define the font size

Style for Scrollbar

```
::-webkit-scrollbar
{
    display: block;
    width: 5px;
    height: 5px;
    padding: 10%;
}
::-webkit-scrollbar-thumb
{
    background: #555555;
    border-radius: 10px;
}
::-webkit-scrollbar-corner
{
    background: transparent;
}
::-webkit-scrollbar-track-piece-start
{
    background: transparent;
    margin-top: 10px;
}

::-webkit-scrollbar-track-piece-end
{
    background: transparent;
    margin-bottom: 10px;
}
```

Word Count: 929 (excluding code, tables, and titles)