Luke Nelson
May 11, 2020
INF 510 – Homework 5

Introduction:
I've been passionate about the outdoors for as long as I can remember. Growing up in Colorado, followed by living in Utah for 4.5 years prior to coming to USC for grad school, served as somewhat of an incubator for falling in love with nature - it allowed me to become an avid hiker, climber, and canyoneer. I started rock climbing with my Father and 3 brothers when I was around 9 years old and have been looking for any excuse to get outside to climb ever since. Over the course of my life, I've enjoyed visiting a number of the national parks, monuments, and historic sites maintained by the U.S. National Park Service. Rocky Mountain National Park, Zion National Park, Dinosaur National Monument, and Arches National Park are just a few of my favorites. Whenever I plan to visit a National Park/Monument with friends or family, I always like to find out if there is quality rock climbing nearby, to complement a day of visiting the park. I've found that it takes a bit of navigating online to find all of the information I need. I will have to navigate to the U.S. National Park Service website to find information about park hours, events, and alerts – this is especially helpful to make sure specific trails or parts of the park aren't shut down for maintenance or road repair. I will then have to look up the weather forecast to make sure it is worth going to the park. As far as rock climbing is concerned, I will have to navigate to Mountain Project and then struggle my way through locating climbing areas nearby the park I plan to visit. Finally, I will have to use google maps to see if the climbs that look interesting are within a reasonable distance of the park. Now that I've provided some context, the aim of this project was to leverage python3 and a number of APIs in conjunction with some webscraping to bring all of the aforementioned data together in just a few simple strokes of the keyboard.

The project brings together data from The US National Park Service (NPS), Mountain Project, the National Weather Service, Inside Hook, and Google Maps. NPS has an API that contains information on National Park Service parks, including but not limited to alerts, events, campgrounds, visitor centers, etc. I utilized the API to access the following data related to all NPS parks: name, state, park Code, park designation, coordinates, park description, weather, url, park events, and park alerts. To do so, I utilized the following NPS API methods: parks, alerts, events. The Mountain Project API contains information on climbing areas and climbing routes, as well as information related to the user's individual MP account, such as the climbs they have marked completed and climbs they would like to climb in the future. I utilized the API to access the following data related to all climbs contained in the Mountain Project API: climbing route name, stars, coordinates, difficulty rating, pitches, url. The specific Mountain Project API method I used was getRoutesForLatLon. I utilized BeautifulSoup to scrape the information on tomorrow's forecast from the National Weather Service, based on the coordinates of the park that the user chooses. I also pulled High and Low temperature data for the ~5 day extended forecast in order to determine the mean high and low temperatures over the extended period. One of the pages on Inside Hook's (an online lifestyle publication) website contains recommendations on what to do in a majority of the US national parks. I thought it would be interesting to do some extra webscraping to bring in recommendations for the best thing to do at a National Park I decide to visit without having to navigate to their site.

The big questions for the project surround whether a program could be developed that is 1) user friendly; 2) comprehensive enough to give the user complete freedom of discretion regarding their interests 3) a superior/preferred method to obtain information rather than using the individual sources already mentioned. The program is successful in that all three questions are addressed and met. The program is user friendly, as it has clearly defined user inputs - input requests include a clear description of the condition in addition to examples of acceptable entries. The program also saves the users API keys in a database so that they are not required to re-input their keys every time they want to utilize the program. Next, the program is comprehensive with regards to user discretion. The user has complete freedom as it relates to the park they would like to visit, the type of climbs they prefer, the difficulty of the climbs, and the distance the climbs are from the park. If a user is manually looking for climbs on mountain project, they have less discretion when navigating among routes. Finally, the program is superior when compared with manually accessing all of the relevant data on the web browser. The program takes less time to retrieve the information than a human would manually, and the final google maps layout is clear and interactive.

Below I will be discussing in further detail the goals of the project, both a high-level and detailed description of the code, a description of additional packages, additional procedures to be followed, and a summary/presentation of results.

Section 1: Goals of the Project
As eluded to in the introduction, the goal of the project was to create a program that would save the user from having to navigate to multiple websites in order to plan a trip of visiting a park and doing some climbing nearby. With only so many hours in the day, it can be exhaustive and inundating to plan a trip – hours spent planning a trip online can even take away from the joy of the adventure.

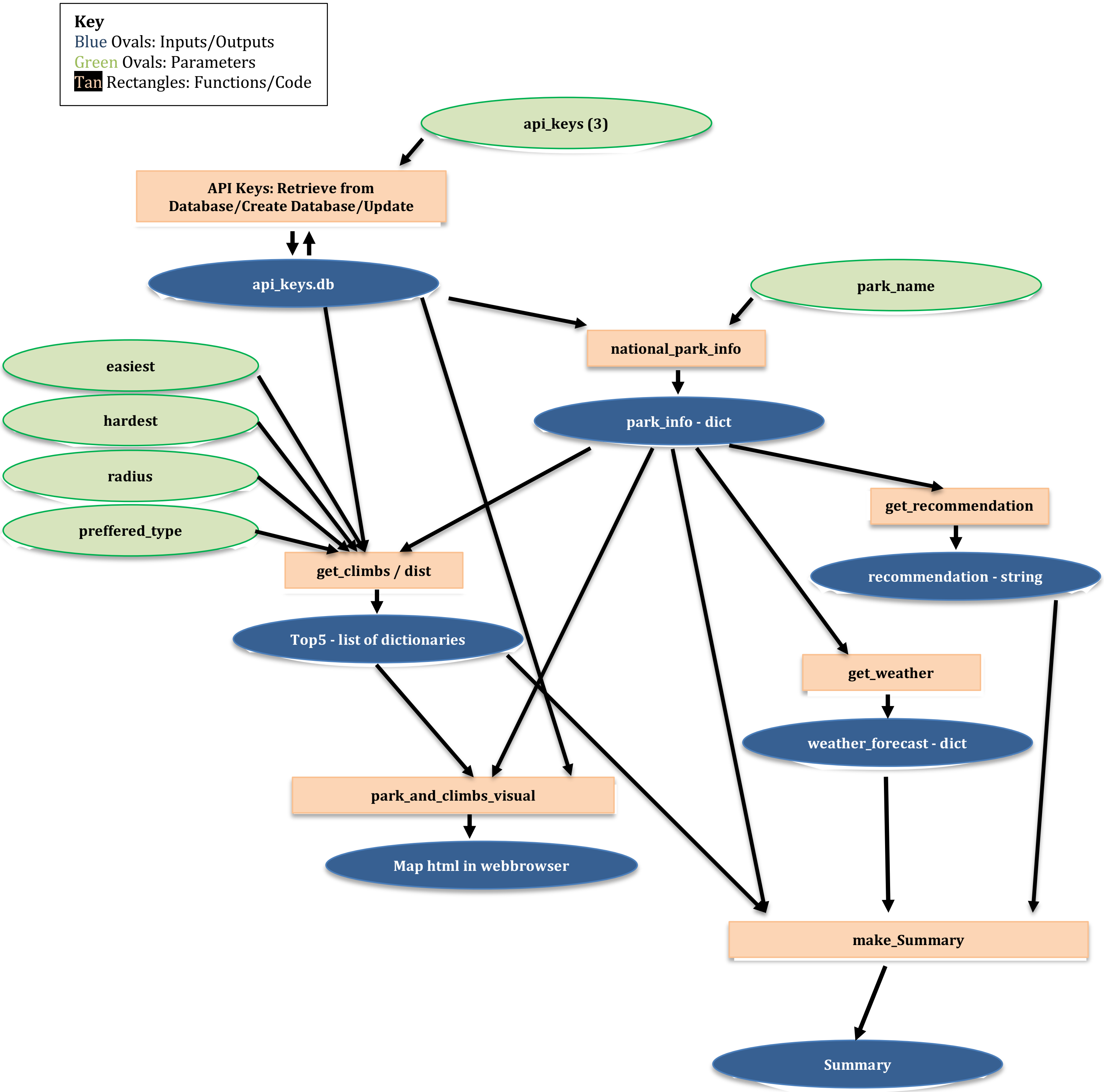
Thus, the goal is to save people time by providing an efficient way to find all the information they may need before visiting a park and going climbing.

The vision was to create a program that allows the user to input the name of a US National Park service maintained park that they would like to visit, along with conditions on the type of rock climbing routes they like, and then returns summary information and a plot of the park and the rock climbing routes. More specifically, the user inputs the following conditions: name of a U.S. National Park Service (NPS) maintained park, a desired climbing skill level (ex. 5.8, 5.10), a climbing type (Sport, Trad, TR), a limit on how far the user is willing to travel to drive from the park to climb. After the user does so, the program will then return some summary information regarding that national park, the weather forecast, possible recommendation as to the best thing to do at that park, and information on the top 5 rated climbing routes within the user-given radius of that park. The program will then plot the locations of the recommended rock climbing routes as well as the specified NPS park on a google map so that the user can visualize where the suggested climbs are relative to the park.

Section 2: Description of Code (High Level/Flow Diagram)
Upon running the program, the user inputs the following conditions: API keys, name of a U.S. National Park Service (NPS) maintained park, a desired climbing skill level (ex. 5.8, 5.10), a climbing type (Sport, Trad, TR), a limit on how far the user is willing to travel to drive from the park to climb. After the user does so, the program will check to see if the park information being returned from the National Park Service is the park intended by the user. If so, the program will then return some summary information regarding that national park, the weather forecast, a possible recommendation as to the best thing to do at that park, and information on the top 5 rated climbing routes within the user-given radius of that park. The program will then plot the locations of the recommended rock climbing routes as well as the specified NPS park on a Google Map so that the user can visualize where the suggested climbs are relative to the park. This plot will be saved as an .html in the “Output” folder of the zip file and should also open automatically in a new window within the user’s internet browser.

My code is contained in two separate .py files: park_and_climb.py and Luke_Nelson_Summary.py. The park_and_climb.py file is a module that contains all of the functions I wrote to pull/extract the data. Luke_Nelson_Summary.py is the code that should be run by the user/grader in order to yield the summary and google map html. It usually takes between 30 seconds to 1 minute to run the code as a result of the multiple API requests being made. The program can be run two different ways: 1) Open Luke_Nelson_Summary.py in your text editor and run the code; 2) in your terminal, set your current directory to the folder where you’ve saved the two .py files (the Code folder). Type “python3 Luke_Nelson_Summary.py” and the program should begin to run. The program begins with code that saves your API keys in a database so that you don’t have to re-enter them every time you want to search for a park and climbing routes. The code checks to see if the table api_keys already exists within a database called park_and_climb. If the table doesn’t exist, you will be prompted to enter your 3 API keys (National Park Service API, Mountain Project API, Google API). If the table already exists (because you’ve run the code and already entered your APIs previously), the program will ask you whether or not you would like to update your API keys. If you say No, the program will use the data already in the table, if you say Yes, the program will ask you to re-input your API key information. Then, the program will ask the user for the other inputs mentioned above: name of a U.S. National Park Service (NPS) maintained park, a desired climbing skill level (ex. 5.8, 5.10), a climbing type (Sport, Trad, TR), a limit on how far the user is willing to travel to drive from the park to climb. After that, the program will utilize the functions in the park_and_climb module to pull information based on the parameters given by the user. The program will produce: the dictionary park_info which contains information on the US National Park Service maintained park; the list of dictionaries Top5 which contains information on the top 5 rated climbing routes that are returned based on the users parameters; the dictionary weather_forecast which contains information on the weather forecast at the park for tomorrow and summary info on the extended forecast; a string with a recommendation from Inside Hook of what to do at the park if it is a national park. The program will take the coordinates from park_info and Top5 to plot the locations of the recommended rock climbing routes as well as the specified NPS park so that the user can visualize where the suggested climbs are relative to the park. This plot will be saved as an .html in the “Output” folder of the zip file and should also open automatically in a new window within the user’s internet browser. The program will finish by taking all outputs and putting them into a clean and easy-to-read summary that is printed in the terminal or text editor.
Below is a workflow to illustrate the general high-level flow of the project. As you can see, many of the data sources are interconnected. Coordinates from park_info and climbs returned by the Mountain Project API query are brought together to

calculate the distance from climbs to the park. Information from all data sources is brought together at the output to give the user a map and summary of the information.



Section 3: Description of Code itself
My code is contained in two separate .py files: park_and_climb.py and Luke_Nelson_Summary.py. This code can be found below in Appendix D. The park_and_climb.py file is a module that contains all of the functions I wrote to pull/extract the data. Luke_Nelson_Summary.py is the code that should be run by the user/grader in order to yield the summary and google maps html. The functions in the park_and_climb.py file are discussed first below, followed by a discussion of the flow of the Luke_Nelson_Summary.py file.

park_and_climb.py code:

The national_park_info function takes two arguments, park_name and nps_api_key. Park_name is the name of the park the user would like to visit. Examples could include Rocky Mountain National Park, Joshua Tree National Park, Dinosaur National Monument, Lincoln Memorial, etc – any park that is maintained by the US National park service. The nps_api_key is the API key you must retrieve from NPS in order to query the API. Based on the park_name entered by the user, the function will try to detect some intial errors. The first protects against the user entering nothing for park_name (‘’). The second keeps the user from entering just one character for a park_name – because when this is sent to the API, it could pull one of any number of random parks. The third protects against an error returned when querying the API, and the fourth protects against the user querying a park_name that does not exist and returning an empty list from the API. If any of these errors occur, the function returns None. The function utilizes requests and json to query the API and return data from the query in JSON. The function then asks the user to confirm that the name of the park returned by the API is the park_name that the user intended. If it is, the function continues, otherwise it returns None. Then, the function adds the desired data to the empty dictionary park_info which was created at the beginning of the function. The function accesses the NPS API two more times, once to get information on real-time alerts in the park based on the parkCode, and the second time to get information on events. The keys for the data in the dictionary are listed below. In the case that there are no alerts or events at the park, the dictionary returns a string saying there are no alerts or no events. Final data is stored in a dictionary called park_info. Please find modeling/formatting info for national_park_info in the Appendix A table “Figure 1. national_park_info.” Keys for the information stored in the dictionary are as follows: ‘name’,’state’,’parkCode’,’designation’,’coordinates’,’description’,’weather’,’url’,’other_parks’,’other_parks_details’,’other_parks_url’,’number_of_alerts’,’’alerts_details’,’number_of_events’,’event_details’. I had to download the following libraries for the code to extract this dataset:

- import requests
- import json
- from datetime import date, timedelta

The get_climbs function takes 6 arguments: park_info, mp_api_key, easiest, hardest, radius, preferred_type. park_info is the dictionary returned by the national_park_info function, mp_api_key is the Mountain Project API key. For context (before discussing the remaining arguments), rock climbing routes are rated by difficulty on a 5. scale (“five-point scale”). Difficulty ratings always include the decimal point, and difficulty ranges from 5.6 to 5.15. The larger the rating, the more difficult the climb. Thus 5.6 is the easiest and is similar to climbing a ladder, and 5.15 is the hardest – only a handful people in the world have ever climbed a 5.15. The argument “easiest” is the easiest difficulty of climb the user is willing to climb, and the argument “hardest” is the hardest difficulty of climb the user is willing to climb. Thus, “easiest” should always be equal to or less than hardest. The argument “radius” is the distance in miles from the park that the person is willing to have climbs returned. Lastly, preferred_type is the users preferred climbing route type. There are different types of climbing. “Sport” is a type where bolts are already located on the wall and are permanently drilled in – as the climber ascends they have to clip the rope into each consecutive bolt to provide a connection to the wall in case they fall. “Trad” involves putting your own nuts/bolts into cracks on the wall as there is no permanent fixed gear. “TR” (also known as Toprope) allows the climber to walk to the top of the climb to fix a rope – this is the safest type of climbing. “Alpine” climbing is a branch of climbing in which the aim is to reach the summit of a mountain – it often requires placing your own screws in ice and using crampons and pick-axes (This type of climbing does exist on Mountain Project, but it is very rare – I simply wanted to make the user is aware of all cases). The program starts by using requests and json libraries to connect to the MP API. I use try/except to return None in the case that no climbing routes are returned by the query. The function then creates an empty dictionary called all_climbs to bring in the desired data from the json returned by the API query. The dictionary has keys of ‘name’ and values are lists of the rest of the data pertinent to each climbing route. The dictionary is turned into a list and sorted by star rating, because the aim of the function is to return the top 5 most popular (highest star rating) climbs. Then the list is sliced so that we are left with the top 5 climbs. We turn the list into a dictionary with the name of the climb as the key. Furthermore, the function uses the dist(point1, point2) function which I created to calculate the distance between each of the 5 climbs and the park, and then append this distance to the dictionary of climbs. The dist function takes two sets of coordinates and calculates the linear distance between them. The get_climbs function finishes by turning each climb into its own dictionary and then putting all 5 dictionaries into a list – as this format will make it easier to plot the climbs via the Google Maps JavaScript API using gmaps later on. Final Data is stored in a list called Top5. It is a list of 1-5 dictionaries, depending on the number of climbing routes returned by the API query. If the API query returns more than 5 climbs, the function will return the 5 top rated climbs. If the API returns 0 climbs, the function will return None. Please find modeling/formatting info for get_climbs in the Appendix A table "Figure 2. get_climbs." I had to download the following libraries for the code to extract this dataset:

- import requests
- import json
- For the distance function:
  - from math import sin, cos, sqrt, atan2, radians

The get_recommendations function takes one argument, park_info. park_info is the dictionary returned by the national_park_info function. The function utilizes BeautifulSoup to webscrape from the following url: https://www.insidehook.com/feature/travel/best-activity-all-59-us-national-parks. After reading in the data as html, the function loops to find recommendations for what to do in the park chosen by the user if the park is a national park. Inside Hook only gives recommendations for what to do at US National Parks, not any of the other parks (ie. Dinosaur National Monument would not have a recommendation on Inside Hook). Final data is stored in a string. The string either tells the user that Inside Hook didn't have a recommendation for the given park, or it gives the recommendation scraped from the aforementioned page. Please find modeling/formatting info for get_recommendations in the Appendix A table "Figure 3. get_recommendations." I downloaded the following libraries for the code to extract this dataset:

- from bs4 import BeautifulSoup
- import urllib
- import requests

There are no special procedures required for this function.

The get_weather function has one argument, park_info. park_info is the dictionary returned by the national_park_info function. The function utilizes BeautifulSoup to webscrape from The National Weather Service (weather.gov). get_weather takes the coordinates from park_info and implements them in the following url: http://forecast.weather.gov/MapClick.php?lat="+lat+"&lon="+lon. After reading in the data as html, the function loops through the html items to locate the forecast for tomorrow. But, since there is no period "tomorrow" in the html, I had to create a switch that turns on after the period "Tonight" is located, as tomorrow always follows the forecast "Tonight." Furthermore, after saving the forecast for tomorrow in a dictionary called weather_forecast, the function compiles a list of the high temperatures in the extended forecast and a list of the low temperatures and adds these lists to the weather_forecast dictionary. The function then takes these lists to count the number of High and Low periods and also calculate the average temperature for High and Low and add this information to the dictionary. The final output of the function is stored in a dictionary titled weather_forecast. Please find modeling/formatting info for get_weather in the Appendix A table "Figure 4. get_weather." Keys for the information stored in the dictionary are as follows: 'temps_high','temps_low','tomorrows_forecast','high_count','high_mean','low_count','low_mean'. I downloaded the following libraries for the code to extract this dataset:

- from bs4 import BeautifulSoup
- import urllib
- import requests
- from statistics import mean

There are no special procedures required for this function.

The park_and_climbs_visual function has two arguments, park_info and Top5. park_info is the dictionary returned by the national_park_info function, and Top5 is the list of dictionaries returned by the get_climbs function. The function starts by asking if there are any dictionaries within the Top5 list, that is to say, it asks if there are any climbs returned by the get_climbs function. If not, the function prints a string saying that there are no climbs and that the program should be re-run with looser constraints. If there are elements in Top5, the function connects to the google Maps JavaScript API. It then utilizes the gmaps library to create a marker layer and symbol layer for the national park. The marker contains the park name and state in the hover text and the park description in the info box. The function also creates marker layers for each of the climbs in Top5. The hover text for these markers is the name of the climb in addition to it's difficulty, and the info box includes the following information about the climb: Name, Rating, Stars, Climb, Pitches, Distance from National Park, a Link to the climb. The function downloads a html to the Output folder in the zip file and titles the file: "park_and_climbs_visual.html." It then opens the html in a new window within the user's internet browser. I downloaded the following libraries for the code to extract this dataset:

- import gmaps
- from ipywidgets.embed import embed_minimal_html
- import webbrowser
- import os

Luke_Nelson_Summary.py code:

The Luke_Nelson_Summary.py file begins with code to save your API keys in a database so that you don't have to re-enter them every time you want to search for a park and climbing routes. The code checks to see if the table api_keys already exists within a database called park_and_climb. If the table doesn't exist, you will be prompted to enter your 3 API keys (National Park Service API, Mountain Project API, Google API). If the table already exists (because you've run the code and already entered your APIs previously), the program will ask you whether or not you would like to update your API keys. If you say No, the program will use the data already in the table, if you say Yes, the program will ask you to re-input your API key information. Then, the program will ask the user for the other inputs mentioned above: name of a U.S. National Park Service (NPS) maintained park, a desired climbing skill level (ex. 5.8, 5.10), a climbing type (Sport, Trad, TR), a limit on how far the user is willing to travel to drive from the park to climb. After that, the program will utilize the functions in the park_and_climb module to pull information based on the parameters given by the user. The program will produce: the dictionary park_info which contains information on the US National Park Service maintained park; The list of dictionaries Top5 which contains information on the top 5 rated climbing routes that are returned based on the users parameters; the dictionary weather_forecast which contains information on the weather forecast at the park for tomorrow and summary info on the extended forecast; a string with a recommendation from Inside Hook of what to do at the park if it is a national park. If park_info is None, the program will finish and instruct the user to 'Please re-run the program and try typing in new conditions.' The program will finish by taking all outputs and printing them in a clean and easy-to-read summary. The program will also take park_info and Top5 to plot the locations of the recommended rock climbing routes as well as the specified NPS park so that the user can visualize where the suggested climbs are relative to the park. This plot will be saved as an .html in the "Output" folder of the zip file and should also open automatically in a new window within the user's internet browser.

Notes:
Please see the requirements.txt file I've created to maintain packages and allow the user/grader to more easily run the code.

If you receive errors regarding any of the modules or libraries I have directed you to import, such as "ImportError: No module named requests", you may have to enter the following command into your terminal:
pip install requests

A description of the data sources can be found in Appendix B, and frequency of update for the data sources can be found in Appendix C.

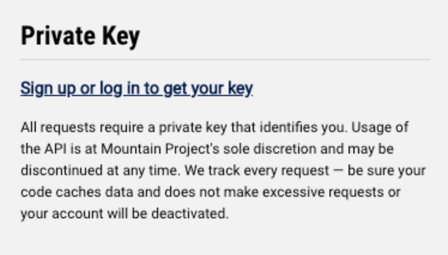
Section 4: Description of Additional Packages
I have used a number of additional packages apart from those discussed in class. They are listed and explained below:

1. from datetime import date, timedelta
    a. The datetime module supplies classes for manipulating dates and times. I used this module in my national_park_info function in order to get the date of today (the current day whenever the code is run) and the date 4 weeks from today. This was done so that we could return any events happening at the specified park within the next 4 weeks

2. from math import sin, cos, sqrt, atan2, radians
    a. This module provides access to the mathematical functions defined by the C standard. I used this in the dist function to calculate the distance between two locations given by coordinates. The dist function sits within the get_climbs function in order to calculate the distance between the park and rock climbs returned by the Mountain Project query.
3. from statistics import mean
    a. This module provides functions for calculating mathematical statistics of numeric (Real-valued) data. I used this in the get_weather function in order to calculate the mean High and Low temperatures over the extended period.
4. import gmaps
    a. gmaps is a Jupyter plugin for embedding Google maps in Jupyter notebooks. It is designed to help visualize and interact with geographical data. I utilized this library in the park_and_climbs_visual function to plot the locations of the park and the climbing routes on a google map that is then saved as an html and opened in the user's browser.
5. from ipywidgets.embed import embed_minimal_html
    a. I used this widget in my park_and_climbs_visual function in order to embed the google map as an html.
6. import webbrowser
    a. This module allows the user to display web pages. I utilized it to display the html created in park_and_climbs_visual in a new window directly in the user's web browser.
7. import os
    a. This module provides a portable way of using operating system dependent functionality. I used it in the park_and_climbs_visual function in order to locate the working directory of the .py file in order to locate and open the html

Within the documentation folder of the zip file I submitted, I have included a requirements file called requirements.txt

Section 5: Additional Procedures to be followed by the grader in order to run the code

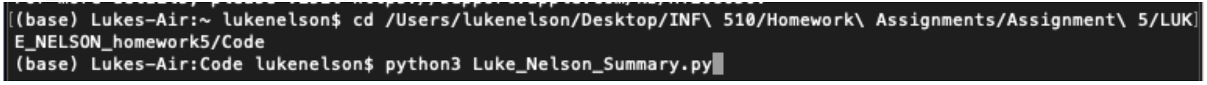
1. To get an API key for the National Park Service API, you must register for an API key at the following URL: https://www.nps.gov/subjects/developer/get-started.htm. Registration is free and requires a first name, last name, and email. Your key will be displayed on the page that is returned after you register, and will also be sent later via email.
2. To get an API key for the Mountain Project API, you must create a free mountain project account. You can do so from the following url: https://www.mountainproject.com/data. Click "Sign up or log in to get your key"
    a. 

       Private Key

       Sign up or log in to get your key

       All requests require a private key that identifies you. Usage of the API is at Mountain Project's sole discretion and may be discontinued at any time. We track every request — be sure your code caches data and does not make excessive requests or your account will be deactivated.
    b. An email will be sent to you to confirm your account and finish your free registration. All it requires is a first name, last name, email, and password. Then if you navigate back to the above url, your Private Key will be given on the right side of the screen.
3. I had to get a Google API key in order to connect to the google Maps JavaScript API. First, you will have to create a google account if you do not already have one. If you need to create a google account, you can do so here: https://accounts.google.com/signup/v2/webcreateaccount?hl=en&flowName=GlifWebSignIn&flowEntry=SignUp. To get the API key, I had to visit the Google Cloud Platform Console (https://console.cloud.google.com/google/maps-apis/overview) then, click on Credentials on the Left. Then, Select Create Project. You will be prompted to enter your name, address, and credit card information, as you will be charged if you utilize more than your free allotted amount. You then will receive an email to confirm and complete your profile and will be prompted to answer a number of questions and then accept the Google Terms and Conditions . After that, navigate back to the "Credentials" tab, click on "Maps JavaScript API", then click "ENABLE." Now, if you go back to Credentials on the left, you will be able select "+CREATE CREDENTIALS" from the top and choose API key. Your API key will be displayed there.

4. There are two ways to run the program:
    a. Open Luke_Nelson_Summary.py in your text editor(PyCharm) and run the code
    b. In your terminal, set your current directory to the folder where you've saved the two .py files. Then type "python3 Luke_Nelson_Summary.py" and the program should begin to run.
        i.



```
[(base) Lukes-Air:~ lukenelson$ cd /Users/lukenelson/Desktop/INF\ 510/Homework\ Assignments/Assignment\ 5/LUK]
E_NELSON_homework5/Code
(base) Lukes-Air:Code lukenelson$ python3 Luke_Nelson_Summary.py
```

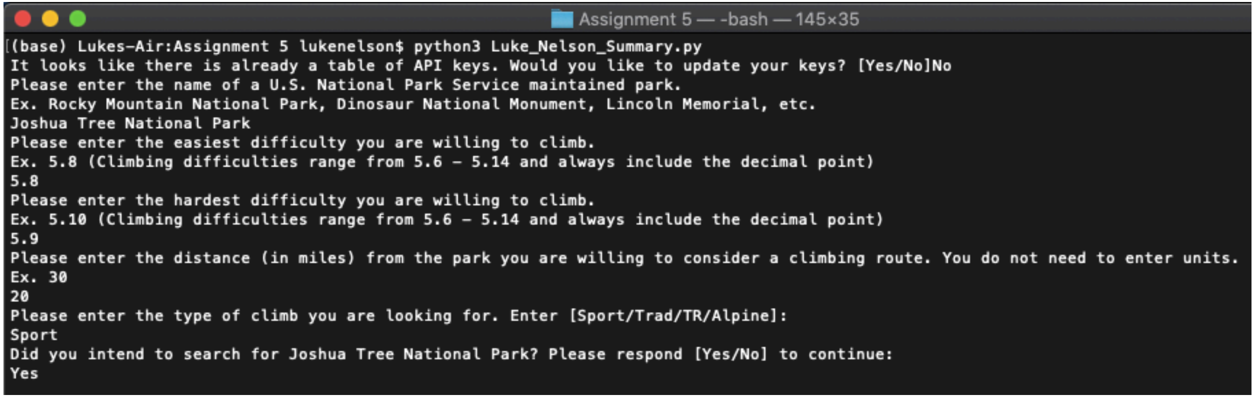
Section 6: Summary/Presentation of Results

The program successfully brings together data from The US National Park Service, Mountain Project, the National Weather Service, Inside Hook, and Google Maps in a way that is user friendly, comprehensive, and superior/preferred to manually accessing the sites individually. As mentioned earlier in the paper, the final output is a summary of information on the park of interest, the weather forecast, recommendations of what to do at the park, and information on the most popular rock climbing routes that meet the users specifications. In addition to the summary, the program outputs a google map that shows the locations of the park and rock climbing routes as well as information about them.

6.1 User Friendly

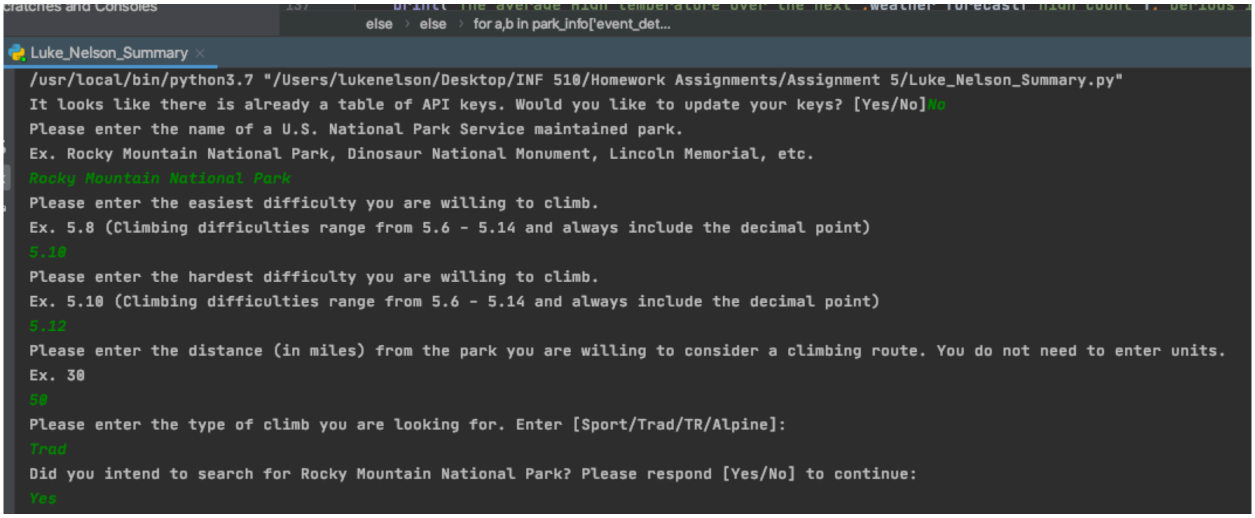
The program was written in a way that makes is easy to use. First, you can run the program directly from your terminal. If you change the directory in your terminal to the folder where the .py files are stored and then type "python3 Luke_Nelson_Summary.py", the program will begin. Second, the program checks to see if there is already a table of api keys stored by the user in the same folder as the code. If the table already exists, the user has the option to not re-enter the keys. This saves the user time when wanting to re-run the program over and over again. Next, while the program asks for a number of inputs, these input requests are clearly defined and examples are given so that the user can easily interact with the program without causing errors. Additionally, the code incorporates a number of error "catches" to protect the program from user error. For example, if the user does not enter a park name, or enters an incorrect park name, the code will be able to detect this and will ask the user to restart the program. After the user enters all inputs, the code will pull the park name from NPS and ask the user if this is their intended park, in order to allow the user to restart the program instead of waiting for further information to be pulled on the wrong park if that be the case. Lastly, the program is user friendly regarding the output, because it returns a clear and easy to read summary with all of the important information. It also automatically opens an html google map in the user's browser so that the user does not have to navigate to any folders to open the map.

Below you will see a screenshot of the program running in the terminal. You can see that the program is asking the user if they would like to update their API keys and also confirming if the park found is the one intended by the user:



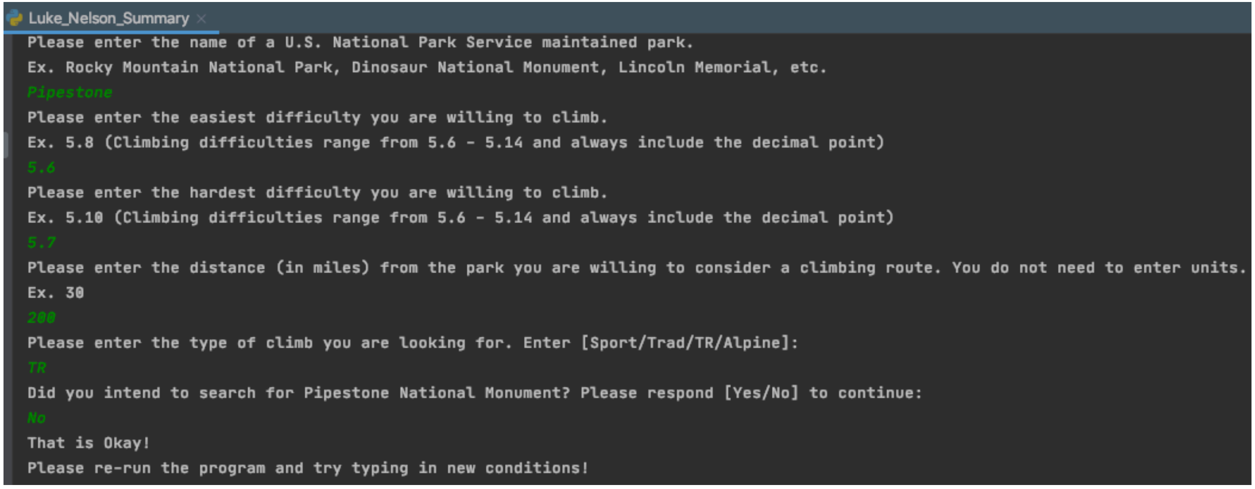
```
(base) Lukes-Air:Assignment 5 lukenelson$ python3 Luke_Nelson_Summary.py
It looks like there is already a table of API keys. Would you like to update your keys? [Yes/No]No
Please enter the name of a U.S. National Park Service maintained park.
Ex. Rocky Mountain National Park, Dinosaur National Monument, Lincoln Memorial, etc.
Joshua Tree National Park
Please enter the easiest difficulty you are willing to climb.
Ex. 5.8 (Climbing difficulties range from 5.6 - 5.14 and always include the decimal point)
5.8
Please enter the hardest difficulty you are willing to climb.
Ex. 5.10 (Climbing difficulties range from 5.6 - 5.14 and always include the decimal point)
5.9
Please enter the distance (in miles) from the park you are willing to consider a climbing route. You do not need to enter units.
Ex. 30
20
Please enter the type of climb you are looking for. Enter [Sport/Trad/TR/Alpine]:
Sport
Did you intend to search for Joshua Tree National Park? Please respond [Yes/No] to continue:
Yes
```

Below is the same, but run in the text editor:



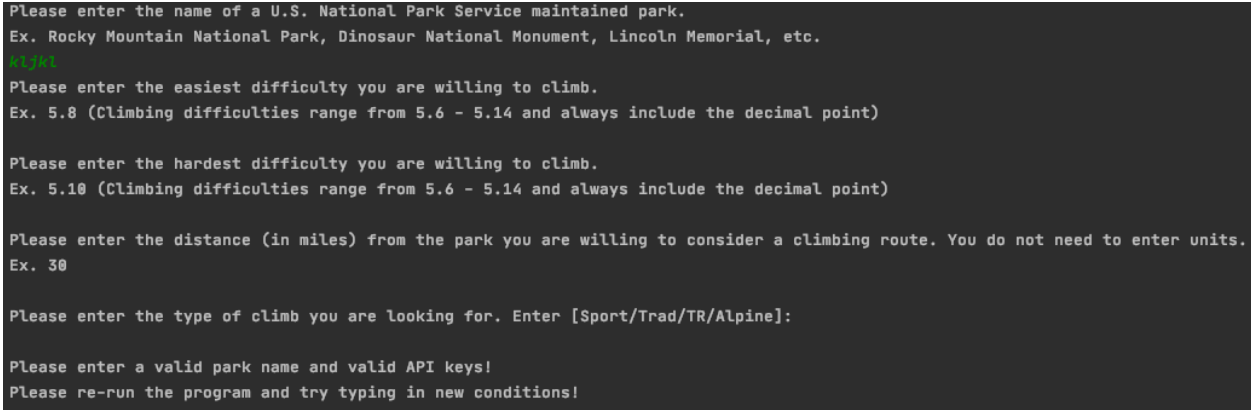
```
/usr/local/bin/python3.7 "/Users/lukenelson/Desktop/INF 510/Homework Assignments/Assignment 5/Luke_Nelson_Summary.py"
It looks like there is already a table of API keys. Would you like to update your keys? [Yes/No]No
Please enter the name of a U.S. National Park Service maintained park.
Ex. Rocky Mountain National Park, Dinosaur National Monument, Lincoln Memorial, etc.
Rocky Mountain National Park
Please enter the easiest difficulty you are willing to climb.
Ex. 5.8 (Climbing difficulties range from 5.6 - 5.14 and always include the decimal point)
5.10
Please enter the hardest difficulty you are willing to climb.
Ex. 5.10 (Climbing difficulties range from 5.6 - 5.14 and always include the decimal point)
5.12
Please enter the distance (in miles) from the park you are willing to consider a climbing route. You do not need to enter units.
Ex. 30
50
Please enter the type of climb you are looking for. Enter [Sport/Trad/TR/Alpine]:
Trad
Did you intend to search for Rocky Mountain National Park? Please respond [Yes/No] to continue:
Yes
```

Here is an example of the program checking to see if the park returned is the one intended by the User. If the user says "No", then the program will simply end and allow the user to restart:



```
Please enter the name of a U.S. National Park Service maintained park.
Ex. Rocky Mountain National Park, Dinosaur National Monument, Lincoln Memorial, etc.
Pipestone
Please enter the easiest difficulty you are willing to climb.
Ex. 5.8 (Climbing difficulties range from 5.6 - 5.14 and always include the decimal point)
5.6
Please enter the hardest difficulty you are willing to climb.
Ex. 5.10 (Climbing difficulties range from 5.6 - 5.14 and always include the decimal point)
5.7
Please enter the distance (in miles) from the park you are willing to consider a climbing route. You do not need to enter units.
Ex. 30
200
Please enter the type of climb you are looking for. Enter [Sport/Trad/TR/Alpine]:
TR
Did you intend to search for Pipestone National Monument? Please respond [Yes/No] to continue:
No
That is Okay!
Please re-run the program and try typing in new conditions!
```

Lastly, the program will catch errors, such as the user entering an incorrect park name that doesn't exist in the NPS system:



```
Please enter the name of a U.S. National Park Service maintained park.
Ex. Rocky Mountain National Park, Dinosaur National Monument, Lincoln Memorial, etc.
kljkl
Please enter the easiest difficulty you are willing to climb.
Ex. 5.8 (Climbing difficulties range from 5.6 - 5.14 and always include the decimal point)

Please enter the hardest difficulty you are willing to climb.
Ex. 5.10 (Climbing difficulties range from 5.6 - 5.14 and always include the decimal point)

Please enter the distance (in miles) from the park you are willing to consider a climbing route. You do not need to enter units.
Ex. 30

Please enter the type of climb you are looking for. Enter [Sport/Trad/TR/Alpine]:

Please enter a valid park name and valid API keys!
Please re-run the program and try typing in new conditions!
```

6.2 Comprehensive

The user has freedom as it concerns the park they would like to visit, the type of climbs the user prefers, the difficulty of the climbs, and the distance from the park the climbs are located. If a user is manually looking for climbs on Mountain Project, they have less discretion when navigating climbing routes. For instance, users of Mountain Project are not given the opportunity to set a radius from a specific location they would be willing to travel in order to climb. As it concerns the National Park service page, the site does not give descriptive statistics surrounding the other parks in the state. My program gives a comprehensive snapshot of other types of parks within the state along with their count, so that the user can use the program down the road to look visit other parks and try new climbs nearby.

6.3 Superior/preferred to navigating web pages.

My program is a preferred method for extracting information when compared to simply navigating the National Park Service, National Weather Service, Mountain Project, and Inside Hook websites. My program pulls only the important information from each page, whereas many of the websites provide details and information that is not useful for planning a trip. Because of my program, the user no longer has to waste time weeding through what is and isn't important. For example, finding out about events at the park requires navigating to a calendar on the NPS website and entering a number of inputs in order to see what will be happening. My program yields that information without the hassle of having to navigate the website. Furthermore, navigating Mountain Project is difficult and slow. It is more efficient and less cryptic to view the climb locations relative to the park in the google map created by my program. Mountain Project does not provide coordinates for the climb on the website, so trying to map climbs relative to the park manually proves difficult if not in some cases impossible.

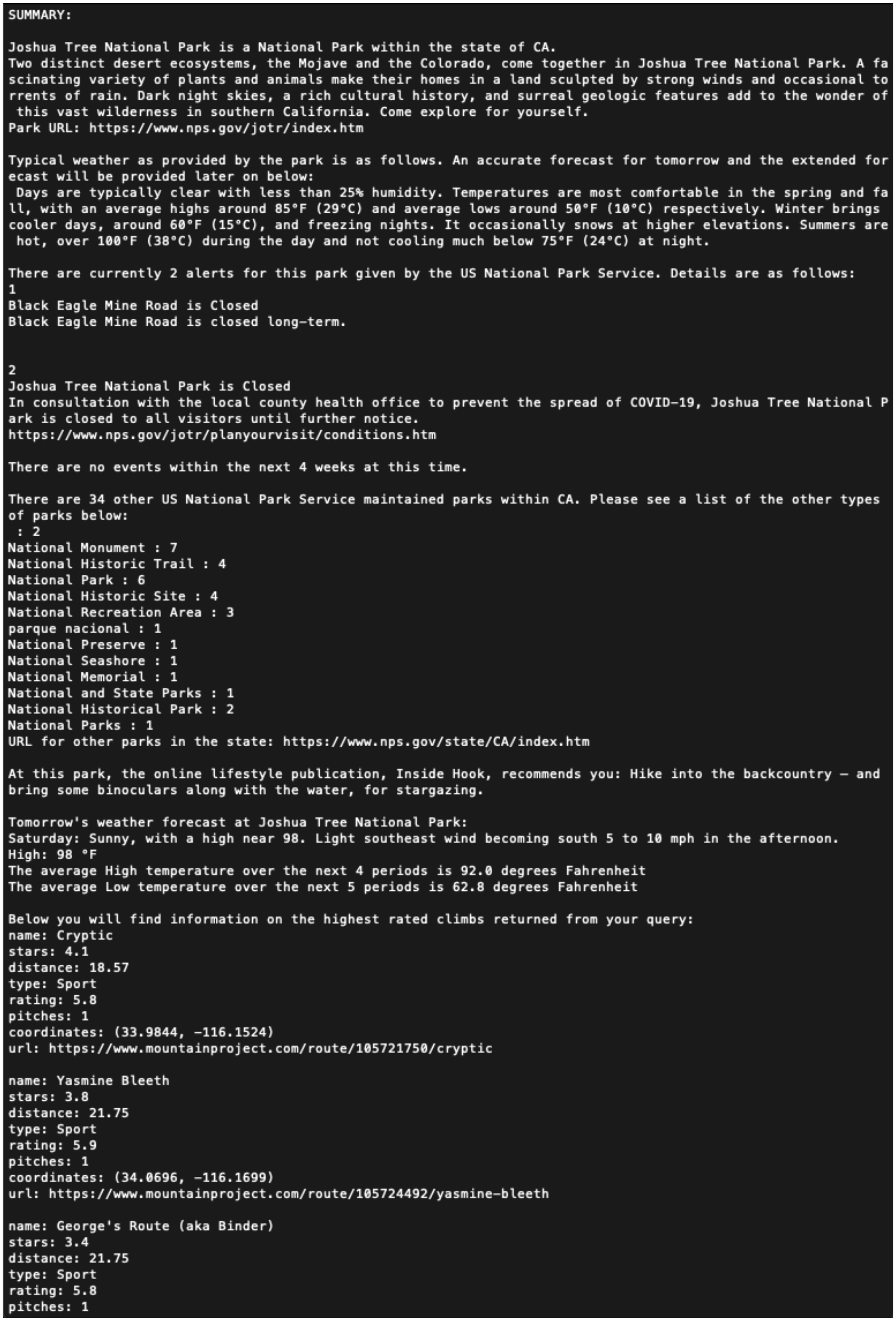
My code performs a number of functions and calculations that are worth highlighting. The most notable are listed here:

- Calculate distance between two sets of coordinates
- Count the number of other parks within the state of the park of interest and place this information in a dictionary
- Sort a dictionary of climbs by climb popularity and slice the dictionary to include no more than 5 climbs
- Determine tomorrow's forecast by using a switch as a result of forecasts being given by day of the week
- Calculate mean temperature over the extended periods for High and Low
- Plott climbs and park in google map using gmaps and iterating through climbs to create info box
- Check for table of API keys and create new table if it does not exist

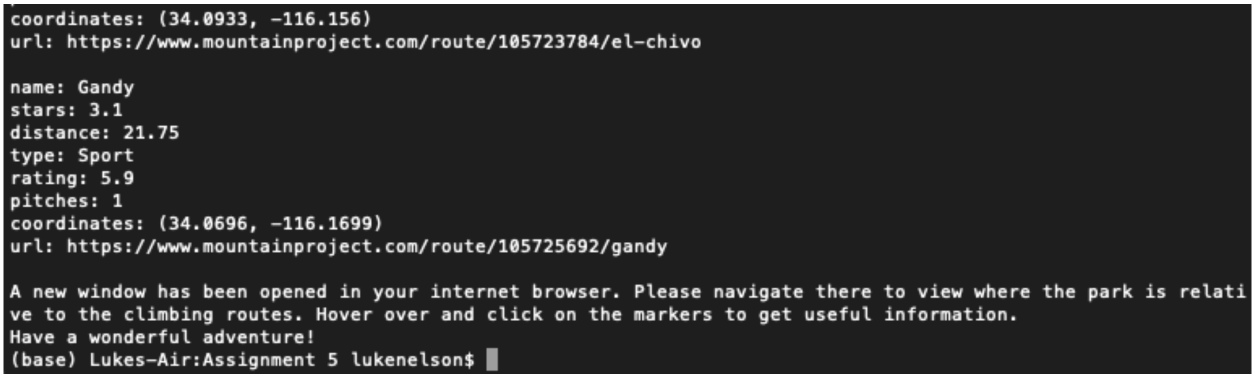
Output:

The output and result of the program are a cleanly laid-out and easy to read summary with important information, in addition to an html google map showing the location of the park and the climbing routes. After the program is run, this html will be saved in the Output folder of the zip file.

Below is a picture of the summary when the program is run in the terminal:

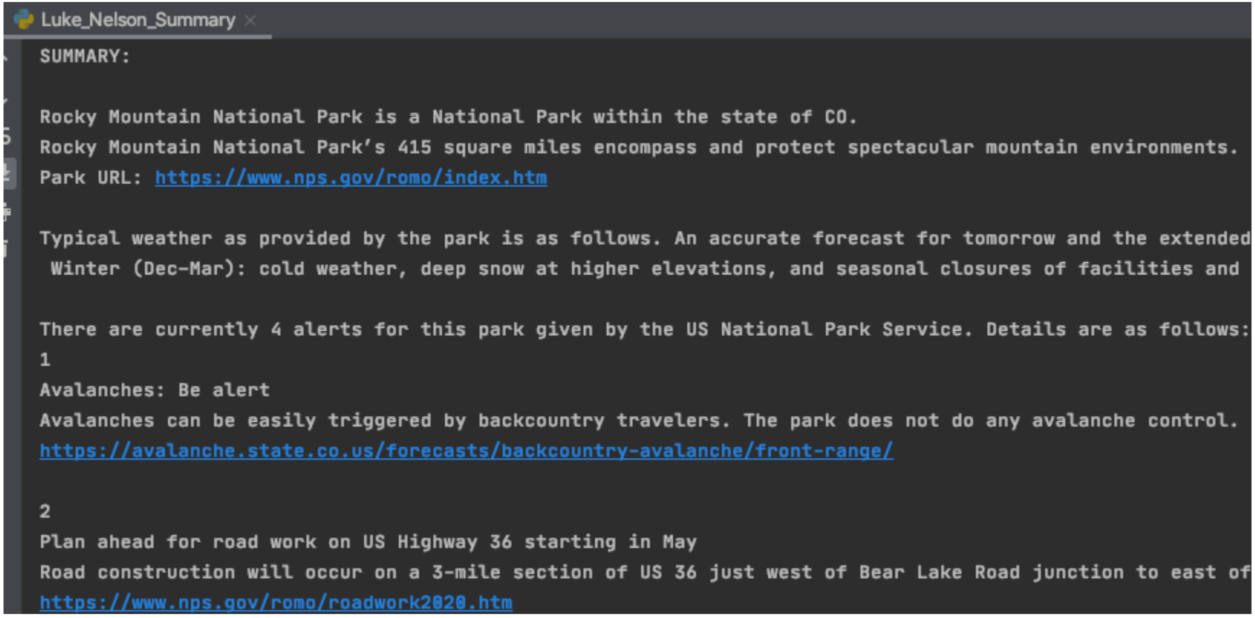


```
SUMMARY:

Joshua Tree National Park is a National Park within the state of CA.
Two distinct desert ecosystems, the Mojave and the Colorado, come together in Joshua Tree National Park. A fa
scinating variety of plants and animals make their homes in a land sculpted by strong winds and occasional to
rrents of rain. Dark night skies, a rich cultural history, and surreal geologic features add to the wonder of
 this vast wilderness in southern California. Come explore for yourself.
Park URL: https://www.nps.gov/jotr/index.htm

Typical weather as provided by the park is as follows. An accurate forecast for tomorrow and the extended for
ecast will be provided later on below:
 Days are typically clear with less than 25% humidity. Temperatures are most comfortable in the spring and fa
ll, with an average highs around 85°F (29°C) and average lows around 50°F (10°C) respectively. Winter brings
cooler days, around 60°F (15°C), and freezing nights. It occasionally snows at higher elevations. Summers are
 hot, over 100°F (38°C) during the day and not cooling much below 75°F (24°C) at night.

There are currently 2 alerts for this park given by the US National Park Service. Details are as follows:
1
Black Eagle Mine Road is Closed
Black Eagle Mine Road is closed long-term.


2
Joshua Tree National Park is Closed
In consultation with the local county health office to prevent the spread of COVID-19, Joshua Tree National P
ark is closed to all visitors until further notice.
https://www.nps.gov/jotr/planyourvisit/conditions.htm

There are no events within the next 4 weeks at this time.

There are 34 other US National Park Service maintained parks within CA. Please see a list of the other types
of parks below:
 : 2
National Monument : 7
National Historic Trail : 4
National Park : 6
National Historic Site : 4
National Recreation Area : 3
parque nacional : 1
National Preserve : 1
National Seashore : 1
National Memorial : 1
National and State Parks : 1
National Historical Park : 2
National Parks : 1
URL for other parks in the state: https://www.nps.gov/state/CA/index.htm

At this park, the online lifestyle publication, Inside Hook, recommends you: Hike into the backcountry — and
bring some binoculars along with the water, for stargazing.

Tomorrow's weather forecast at Joshua Tree National Park:
Saturday: Sunny, with a high near 98. Light southeast wind becoming south 5 to 10 mph in the afternoon.
High: 98 °F
The average High temperature over the next 4 periods is 92.0 degrees Fahrenheit
The average Low temperature over the next 5 periods is 62.8 degrees Fahrenheit

Below you will find information on the highest rated climbs returned from your query:
name: Cryptic
stars: 4.1
distance: 18.57
type: Sport
rating: 5.8
pitches: 1
coordinates: (33.9844, -116.1524)
url: https://www.mountainproject.com/route/105721750/cryptic

name: Yasmine Bleeth
stars: 3.8
distance: 21.75
type: Sport
rating: 5.9
pitches: 1
coordinates: (34.0696, -116.1699)
url: https://www.mountainproject.com/route/105724492/yasmine-bleeth

name: George's Route (aka Binder)
stars: 3.4
distance: 21.75
type: Sport
rating: 5.8
pitches: 1
```



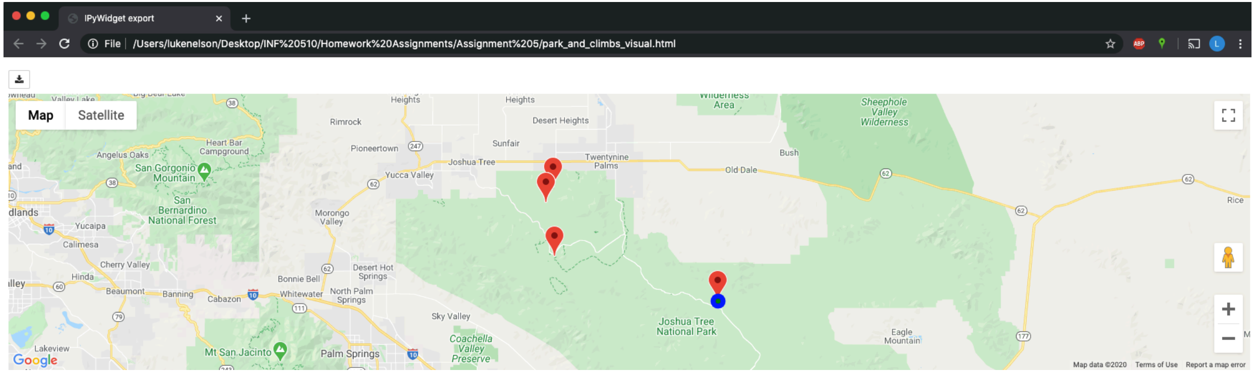
```
coordinates: (34.0933, -116.156)
url: https://www.mountainproject.com/route/105723784/el-chivo

name: Gandy
stars: 3.1
distance: 21.75
type: Sport
rating: 5.9
pitches: 1
coordinates: (34.0696, -116.1699)
url: https://www.mountainproject.com/route/105725692/gandy

A new window has been opened in your internet browser. Please navigate there to view where the park is relati
ve to the climbing routes. Hover over and click on the markers to get useful information.
Have a wonderful adventure!
(base) Lukes-Air:Assignment 5 lukenelson$ 
```

Here is a screenshot of the summary when the program is run in the Text Editor:



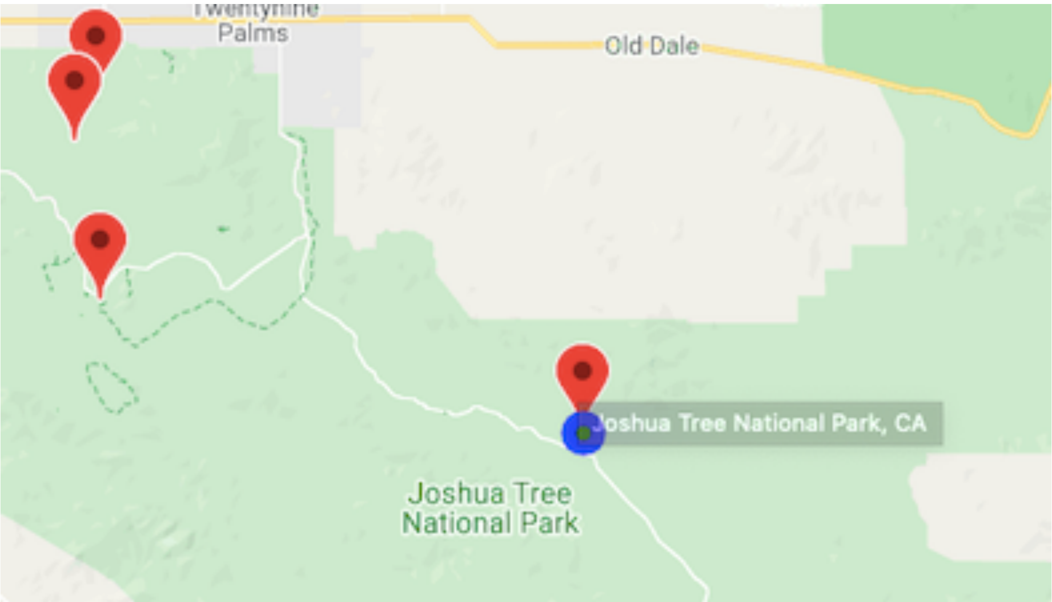
```
SUMMARY:

Rocky Mountain National Park is a National Park within the state of CO.
Rocky Mountain National Park’s 415 square miles encompass and protect spectacular mountain environments.
Park URL: https://www.nps.gov/romo/index.htm

Typical weather as provided by the park is as follows. An accurate forecast for tomorrow and the extended
 Winter (Dec-Mar): cold weather, deep snow at higher elevations, and seasonal closures of facilities and

There are currently 4 alerts for this park given by the US National Park Service. Details are as follows:
1
Avalanches: Be alert
Avalanches can be easily triggered by backcountry travelers. The park does not do any avalanche control.
https://avalanche.state.co.us/forecasts/backcountry-avalanche/front-range/

2
Plan ahead for road work on US Highway 36 starting in May
Road construction will occur on a 3-mile section of US 36 just west of Bear Lake Road junction to east of
https://www.nps.gov/romo/roadwork2020.htm
```

The program also creates an interactive google map. The html map provides the user with both hover text and info boxes.

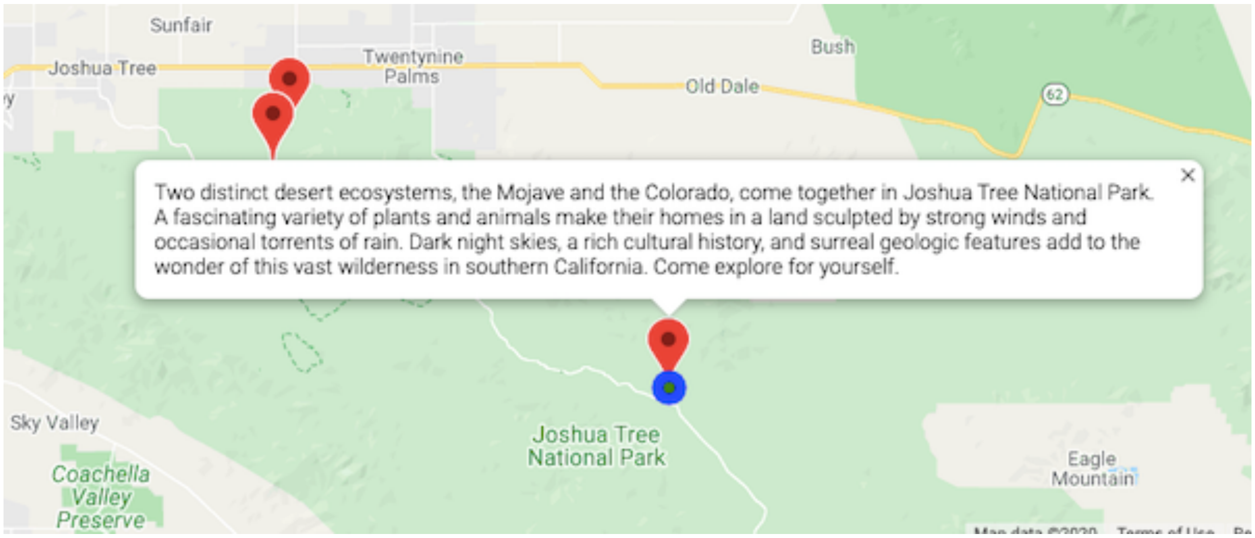


The national park is identifiable among the other markers as it has been marked with a dark blue symbol at its base. The hover text for the park includes the park name and the state in which it is located, and it’s info box contains the description of the park.

National Park Hover Text:



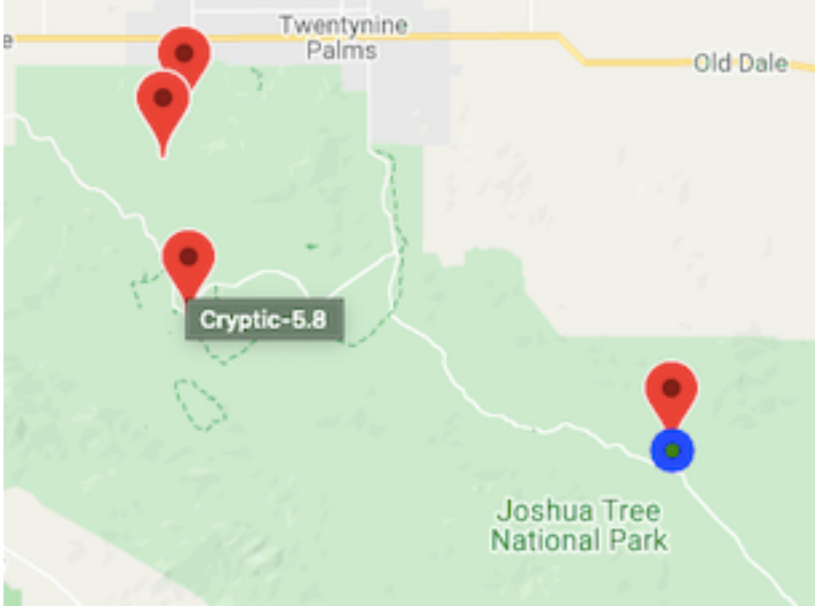
National Park info box:


Sunfair
Joshua Tree
Twentynine Palms
Bush
Old Dale
62
Sky Valley
Coachella Valley Preserve
Joshua Tree National Park
Eagle Mountain
Map data ©2020 Terms of Use

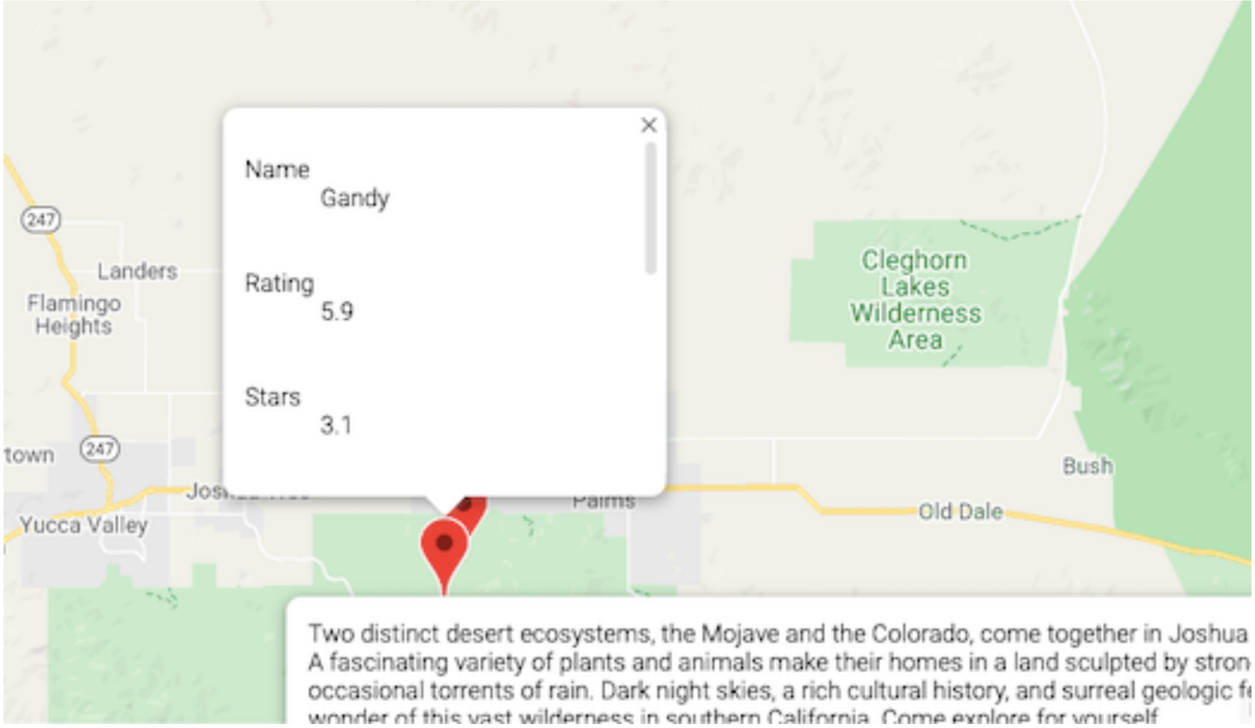
Two distinct desert ecosystems, the Mojave and the Colorado, come together in Joshua Tree National Park. A fascinating variety of plants and animals make their homes in a land sculpted by strong winds and occasional torrents of rain. Dark night skies, a rich cultural history, and surreal geologic features add to the wonder of this vast wilderness in southern California. Come explore for yourself.

The hover text for the rock climbing routes displays the climb name and rating. The info box for the climbing routes contains all the information in the Top5 dictionary: name, stars, distance, type, rating, pitches, coordinates, url.

Climbing route hover text:



Climbing route info box:



Name
Gandy

Rating
5.9

Stars
3.1

Two distinct desert ecosystems, the Mojave and the Colorado, come together in Joshua
A fascinating variety of plants and animals make their homes in a land sculpted by stron
occasional torrents of rain. Dark night skies, a rich cultural history, and surreal geologic fe
wonder of this vast wilderness in southern California. Come explore for yourself

Conclusion:
This was a wonderful project. It has already proven very useful to me. I’m planning a trip to Joshua Tree after the parks open back up (weather permitting) and will be able to find the top rated climbs within a 15 mile radius of the park, mapped out for me and with all of the information I require. My brothers are also big climbers who love to visit parks and have told me that they would like to use this program, as it is comprehensive and easier than searching blindly for climbs on mountain project’s website. The program is certainly user friendly, comprehensive, and it is a preferred method for returning data when compared to manually accessing the information online.

We could extend this model by also allowing for the user to enter the name of state parks or national forests. Currently, information is coming from the National Park Service API, which does not include information for state parks or forests as these types of parks are not maintained by the National Park Service. I would need to bring in data from every US state's park repository in order to make this a reality. I could also utilize additional methods from the Mountain Project API to bring in information about the user's mountain project account, related specifically to whether or not the user has already climbed certain climbs or whether the user has marked specific climbs that they plan to climb. I could then structure the model/program so that the user is only given climbs that they have not marked as having already climbed. For the weather forecast, I could provide the user more flexibility and lets them enter what day in the extended period they want a forecast for. This way, the user could ask for the forecast for 3 days from now for example as opposed to just getting tomorrow's forecast (which is the default). Lastly, I could work to turn this code into an application that is more user friendly and allows the user to select options rather than having to type the desired information in.

I am very excited to use this program and to further refine it down the road as I see fit and as my interests change!

Appendix A:

Figure 1. national_park_info

Modeling/information of the data returned by the function, including format, units, description, and connection to other functions within the code.

Key	**Format**	**Units**	**Description**	**Connection to other data**
name	string	NA	name of US NPS maintained park	get_recommendations, park_and_climbs_visual
state	string	NA	State where park is located	park_and_climbs_visual
parkCode	string	NA	Unique identifier given by NPS	
designation	string	NA	Type of park (ex. National Park, National Monument)	
coordinates	tuple of floats	degrees	location of park	get_climbs, get_weather, park_and_climbs_visual, dist
description	string	NA	Description of park	park_and_climbs_visual
weather	string	NA	Typical weather at the park (Not Live)	
url	string	NA	url for park on NPS site	
other_parks	string	NA	# of other parks in the state	
other_parks_details	dictionary with keys as strings and values as integers	NA	designations of other parks and the counts	
other_parks_url	string	NA	NPS url to all other parks in the state	
number_of_alerts	string	NA	# of alerts for the park	
alerts_details	dictionary with keys as strings and values as lists of two strings	NA	alert descriptions and their urls	

number_of_events	string	NA	# of events at the park within the next 4 weeks	
event_details	string or dictionary with keys and values as strings	NA	description of event and the date	

Figure 2. get_climbs
Modeling/information of the data returned by the function, including format, units, description, and connection to other functions within the code

Key	**Format**	**Units**	**Description**	**Connection to other data**
name	string	NA	name of the rock climbing route	park_and_climbs_visual
stars	float	stars	the user voted quality rating of the route	park_and_climbs_visual
distance	float	miles	distance in miles the climb is from the park	park_and_climbs_visual
type	string	NA	type of climbing (Sport, Trad, TR, Alpine)	park_and_climbs_visual
rating	string	NA	difficulty of the climb on the 5. scale	park_and_climbs_visual
pitches	integer	pitches	the number of climb lengths (height) of the route	park_and_climbs_visual
coordinates	tuple of floats	degrees	the location of the climbing route	dist, park_and_climbs_visual
url	string	NA	the mountain project url for the climb	park_and_climbs_visual

Figure 3. get_recommendations
Modeling/information of the data returned by the function, including format, units, description, and connection to other functions within the code.

Output	**Format**	**Units**	**Description**	**Connection to other data**
recommendation	string	NA	The string either tells the user that Inside Hook didn't have a recommendation for the given park, or it gives the recommendation scraped from	

Figure 4. get_weather
Modeling/information of the data returned by the function, including format, units, description, and connection to other functions within the code.

Key	**Format**	**Units**	**Description**	**Connection to other data**
tomorrows forecast	list of strings	NA	the forecast for the park tomorrow	

temps_high	list of floats	degrees fahrenheit	high temps over the extended forecast	
temps_low	list of floats	degrees fahrenheit	low temps over the extended forecast	
high_count	integer	NA	number of periods in high extended forecast	
high_mean	float	degrees fahrenheit	the mean temperature in temps_high	
low_count	integer	NA	number of periods in low extended forecast	
low_mean	float	degrees fahrenheit	the mean temperature in temps_low	

Appendix B :
Data Source Information

My program utilizes 4 different data sources for the project. I utilized APIs to retrieve data from 2 of them, and webscraping for the remaining two.

The first data source I utilized is the the U.S. National Park Service (NPS) external public API. The U.S. National Park Service homepage can be found here: https://www.nps.gov/index.htm, and information related to the API can be found here: https://www.nps.gov/subjects/digital/nps-data-api.htm. The U.S. National Park Service (NPS) is an agency of the US federal government that manages 419 parks/areas covering more than 85 million acres across the entire Unites States, and including American Samoa, Guam, Puerto Rico, and the Virgin Islands. The areas it maintains include National Parks (Joshua Tree National Park, Zion National Park), National Monuments (Dinosaur National Monument), battlefields, historic parks, historic sites (Lincoln Memorial), lakeshores, seashores, historic trails, etc. NPS has an API that contains information on National Park Service parks, including but not limited to alerts, events, campgrounds, visitor centers, etc. I utilized the API to access the following data related to all NPS parks: name, state, park Code, park designation, coordinates, park description, weather, url, park events, and park alerts. To do so, I utilized the following NPS API methods: parks, alerts, events.

The second data source I utilized is the Mountain Project (MP) external public API. Mountain Project is a climbing resource for rock climbers across the globe. It serves as a place and forum for climbers to discover climbing areas and routes, rate routes, and share their comments, opinions, questions, and photos. This website serves as a guidebook to more than 170,000 climbing routes across the world, with routes being put up and added every week. Mountain Project is owned and maintained by REI. The Mountain Project homepage can be found here: https://www.mountainproject.com/, and information regarding the API can be found here: https://www.mountainproject.com/data. The Mountain Project API contains information on climbing areas and climbing routes, as well as information related to the user's individual MP account, such as the climbs they have marked completed and climbs they would like to climb in the future. I utilized the API to access the following data related to all climbs contained in the Mountain Project API: climbing route name, stars, coordinates, difficulty rating, pitches, url. The specific Mountain Project API method I used was getRoutesForLatLon.

The third data source I used for my project was the Extended Forecast from the National Weather Service. The National Weather Service is an agency of the US federal government that is tasked with providing weather forecasts, hazardous weather warnings, and other weather-related information to the public. National Weather Service extended forecasts can be found via the following: https://forecast.weather.gov/. While the National Weather Service does have an API, I decided to utilize webscraping to gather the live data I desired in order to validate the project requirement that we utilize webscraping for at least one data source. I ran this by Yigal and he said it was okay to use webscraping for this dataset, even though an API is available. I utilized BeautifulSoup to scrape the information on tomorrow's forecast for the coordinates of the park that the user chooses. I also pulled High and Low temperature data for the ~5 day extended forecast in order to determine the mean high and low temperatures over the extended period.

Another data source I utilized for this project comes from Inside Hook. Inside Hook is an online news/lifestyle platform that provides its users with content on goods, where to go, what to buy, and how to live. One of the pages on Inside Hook's website contains recommendations on what to do in a majority of the US national parks. This is not a live dataset as the page is not regularly changed/updated, but I thought it would be interesting to do some extra webscraping to bring in recommendations for the best thing to do at a National Park I decide to visit without having to navigate to their site. The link to Inside Hook's homepage is as follows: https://www.insidehook.com/, and here is the link to the particular page I scraped: https://www.insidehook.com/feature/travel/best-activity-all-59-us-national-parks.

Appendix C:
Frequency of Data Update of the Data Source

Data from the U.S. National Park Service API is Live, as it requires daily updates for park events and alerts. The park service is constantly updating alerts such as rock falls, trail closures, park closures, road work, incoming hazardous weather, etc. The park service is also continually creating and updating events in the park that can occur as often as daily – examples include scheduled hikes and history/information sessions regarding animals and plants that occur in the various parts of the park. Though not as often, new parks and areas are being added to the national park service's purview every year.

The Mountain Project API is Live, as users of the website/application are constantly giving their input in terms of climb difficulty and rating. As a result of the climb star rating and difficulty rating being based on consensus, the popularity and difficulty of climbs is updated daily based on votes from the climbing community. New climbs are also constantly being added to the Mountain Project as the climbing community is very active in setting and creating new routes in designated climbing areas. As a result of this, Mountain project recommends leaving data cached for no more than a week.

National Weather Service forecasts are certainly Live, as data collected from the national weather service observation stations is updated at least once an hour. New data which comes in constantly changes the forecasts that are given for the rest of the day, tomorrow, and the extended forecast – which is the data with which my program is concerned.

Inside Hook's recommendations for what to do at the National Parks is NOT Live, as the recommendations are not changed or updated regularly. However, this is not one of my main data sources, I just wanted to bring in another example of webscraping and some additional useful information for my project.

Appendix D:
Code

Below is the code contained in the two .py files

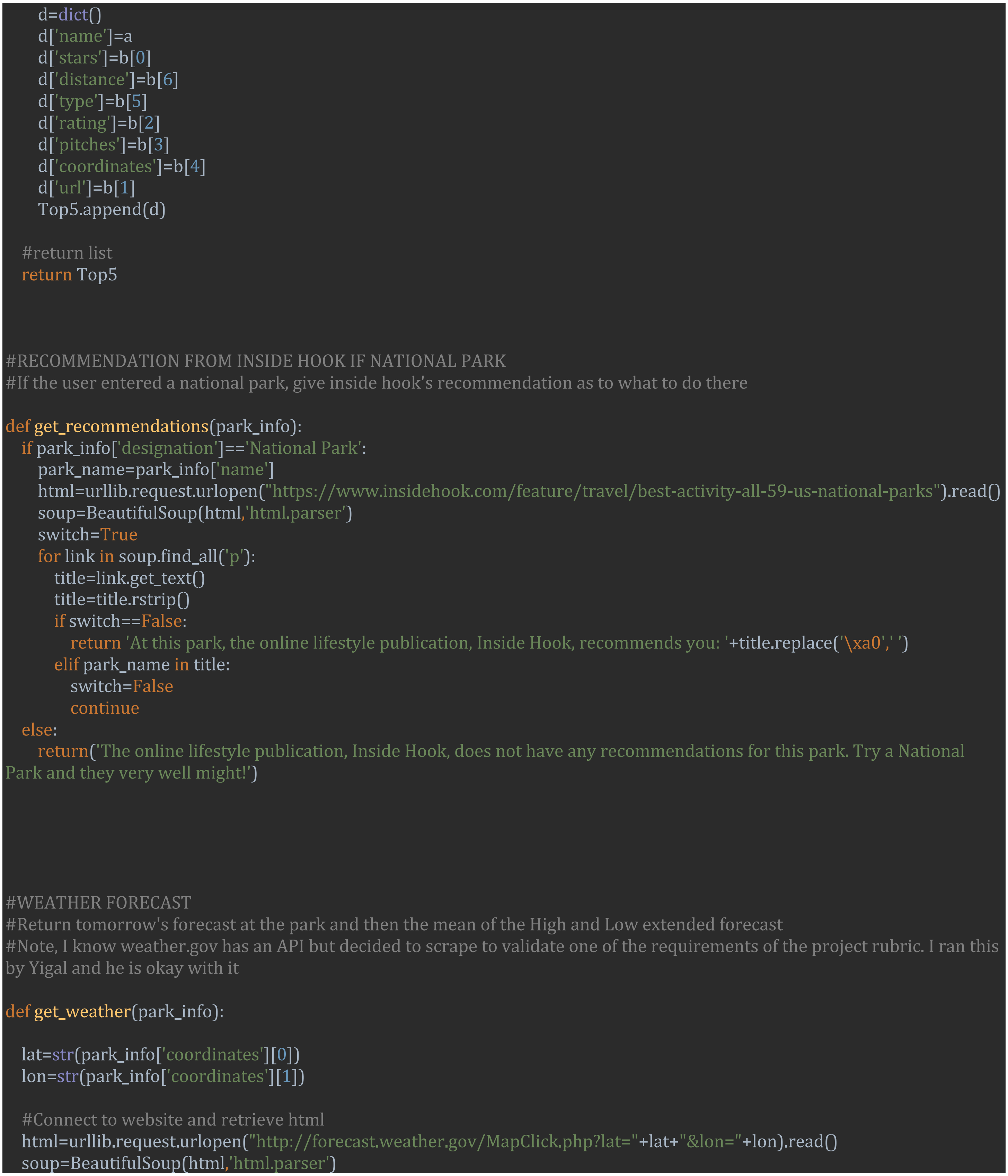
park_and_climb.py

```
#LIBRARIES WE NEED:


import json    #NPS; MP
from datetime import date, timedelta   #NPS
from math import sin, cos, sqrt, atan2, radians    #DISTANCE FUNCTION
from bs4 import BeautifulSoup    #WEATHER, INSIDE HOOK
```

```
import urllib    #WEATHER, INSIDE HOOK
import requests    #WEATHER, INSIDE HOOK, NPS, MP
from statistics import mean    #WEATHER
import gmaps #GOOGLE MAPS
from ipywidgets.embed import embed_minimal_html    #GOOGLE MAPS
import webbrowser    #GOOGLE MAPS
import os    #GOOGLE MAPS


#NPS FUNCTION
#Function to print all the summary info about the park

def national_park_info(park_name, nps_api_key):
    park_info=dict()

    try:
        #Access the NPS API with park name
        error1=park_name[0]    #This protects against the user entering '' for the park_name
        error2=len(park_name)/(len(park_name)-1)    #Park name has to be more than 1 character, otherwise yields random
match
        resp=requests.get('https://developer.nps.gov/api/v1/parks?q='+park_name+'&api_key='+nps_api_key)
        js=resp.json()
        error3=js['data'][0] #this protects against the user entering a park name that does not exist

    except:
        print('Please enter a valid park name and valid API keys!')
        return

    #Check to see if the park returned by the JSON is the park intended by the user
    park_checker=input('Did you intend to search for '+js['data'][0]['fullName']+'? Please respond [Yes/No] to continue:\n')
    if park_checker != 'Yes':
        print('That is Okay!')
        return
    else:

        #Add park name to dict
        park_info['name']=js['data'][0]['fullName']

        #The state the park is in
        state=js['data'][0]['states']
        park_info['state']=state

        #ParkCode
        parkCode=js['data'][0]['parkCode']
        park_info['parkCode']=parkCode

        #ParkDesignation
        parkDesignation=js['data'][0]['designation']
        park_info['designation']=parkDesignation

        #The coordinates for the park
        coordinates=js['data'][0]['latLong']
        coordinates=coordinates.split(',')
        lat_colon=coordinates[0].find(':')
```

```
latitude=float(coordinates[0][lat_colon+1:])
lon_colon=coordinates[1].find(':')
longitude=float(coordinates[1][lon_colon+1:])
new_coordinates=(latitude,longitude)
park_info['coordinates']=new_coordinates

#Park Description
description=js['data'][0]['description']
park_info['description']=description

#General Weather
weather=js['data'][0]['weatherInfo']
park_info['weather']=weather

#URL.
url=js['data'][0]['url']
park_info['url']=url

#Get information about the other parks within the state
resp2=requests.get('https://developer.nps.gov/api/v1/parks?stateCode='+state+'&api_key='+nps_api_key)
js2=resp2.json()

sites=dict()
total_parks=js2['total']
site_info=js2['data']
for item in site_info:
    x=item['designation']
    sites[x]=sites.get(x,0)+1
park_info['other_parks']=total_parks
park_info['other_parks_details']=sites

#URL for all parks in the state
park_info['other_parks_url']='https://www.nps.gov/state/'+state+'/index.htm'

#Get information on Live-real time alerts within the park
resp3=requests.get('https://developer.nps.gov/api/v1/alerts?parkCode='+parkCode+'&api_key='+nps_api_key)
js3=resp3.json()
number_of_alerts=js3['total']
park_info['number_of_alerts']=number_of_alerts
if number_of_alerts != '0':
    alerts=dict()
    alert_info=js3['data']
    for item in alert_info:
        x=item['title']
        y=item['description']
        z=item['url']
        alerts[x]=[y,z]
    park_info['alert_details']=alerts
else:
    park_info['alert_details']='There are no alerts at this time.'

#Get data on Live Events happening tomorrow or within 4 weeks
startDate=str(date.today()+timedelta(days=1))
endDate=str(date.today()+timedelta(days=28))
```

```
resp4=requests.get('https://developer.nps.gov/api/v1/events?parkCode='+parkCode+'&api_key='+nps_api_key+'&dateStart=
'+startDate+'&dateEnd='+endDate)
        js4=resp4.json()
        number_of_events=js4['total']
        park_info['number_of_events']=number_of_events
        if number_of_events != '0':
            events=dict()
            event_info=js4['data']
            for item in event_info:
                try:
                    x=item['description']
                    x=x.replace('<p>','')
                    x=x.replace('</p>','')
                    z=item['datestart']
                    events[x]=z
                except:
                    events[item['description']]='Unfortunately there is no datestart'
            park_info['event_details']=events
        else:
            park_info['event_details']='There are no events within the next 4 weeks at this time.'

        #return dictionary
        return park_info



#DISTANCE FUNCTION
#This formula takes the distance (in miles) bewteen two points of the form parameters tuple - (lat1, long1), tuple - (lat2, long2)
and returns distance in kms between both (lat, long) tuples.

def dist(point1, point2):
    # approximate radius of earth in km
    R = 6373.0
    lat1 = radians(point1[0])
    lon1 = radians(point1[1])
    lat2 = radians(point2[0])
    lon2 = radians(point2[1])
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    distance = R * c #in km
    distance=distance * 0.62137 #converted to miles
    distance=round(distance,2)
    return distance




#MOUNTAIN PROJECT FUNCTION
def get_climbs(park_info, mp_api_key, easiest, hardest, radius, preferred_type):
  latitude=str(park_info['coordinates'][0])
  longitude=str(park_info['coordinates'][1])
  park_coords=park_info['coordinates']

  try:
    resp=requests.get('https://www.mountainproject.com/data/get-routes-for-lat-
lon?lat='+latitude+'&lon='+longitude+'&maxDistance='+radius+'&minDiff='+easiest+'&maxDiff='+hardest+'&key='+mp_api_k
ey)
    js=resp.json()
    data=js['routes']
  except:
    print('Please enter valid inputs!')
    return

  #Create a dictionary for all climbs of the specified type
  all_climbs=dict()
  for item in data:
    if preferred_type in item['type']:

all_climbs[item['name']]=[item['stars'],item['url'],item['rating'],item['pitches'],(item['latitude'],item['longitude']),item['type']]
    else:
      continue

  #Sort dictionary by Star Rating. tmp stands for temporary since this is a temporary list for sorting
  tmp=[]
  for a,b in all_climbs.items():
    newtup=(b,a)
    tmp.append(newtup)
  tmp.sort(reverse=True)

  #If there are zero climbs or fewer than 5 climbs, let the user know:
  if len(tmp)== 0:
    #print('Unfortunately there are no climbs within your specified constraints. Please try broadening them.')
    return

  #Turn the list into the top 5 rated climbs
  climbs=tmp[:5]


  #Turn back into a dictionary with name as the key and star rating back as a value
  climbs_dct=dict()
  for a,b in climbs:
    climbs_dct[b]=a

  #Calculate distance between the 5 climbs and the national park
  #Then feed the lat lon and the park_coords into the distance function and append the distance to climbs_dct
  for a,b in climbs_dct.items():
    distance_from_park=dist(park_coords,b[4])
    climbs_dct[a].append(distance_from_park)

  #Make each climb into a dictionary and make a list of dictionaries so making the google map will be easier. Top5 refers to
our Top5 climbs
  Top5=list()
  for a,b in climbs_dct.items():
```

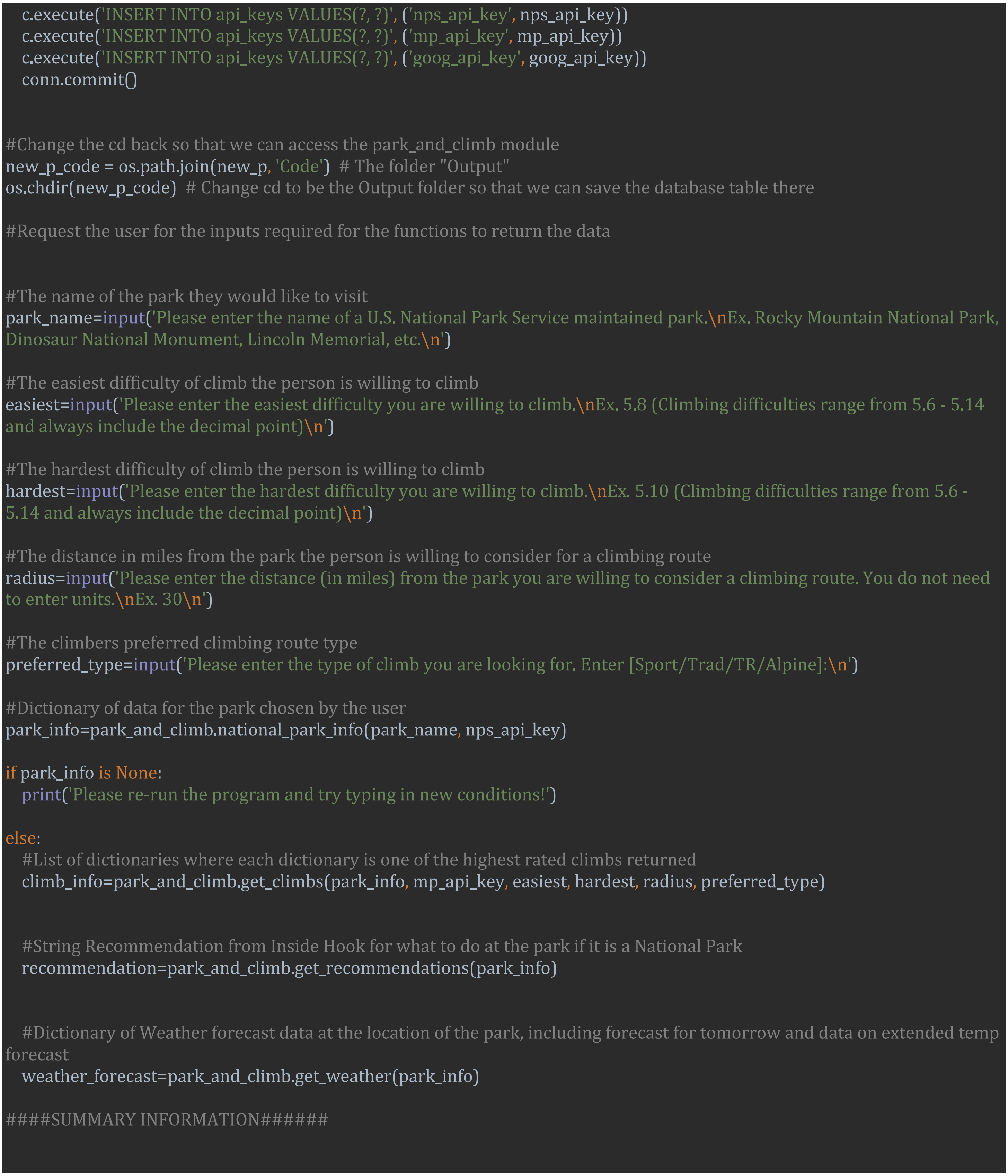


```
        d=dict()
        d['name']=a
        d['stars']=b[0]
        d['distance']=b[6]
        d['type']=b[5]
        d['rating']=b[2]
        d['pitches']=b[3]
        d['coordinates']=b[4]
        d['url']=b[1]
        Top5.append(d)

    #return list
    return Top5



#RECOMMENDATION FROM INSIDE HOOK IF NATIONAL PARK
#If the user entered a national park, give inside hook's recommendation as to what to do there

def get_recommendations(park_info):
    if park_info['designation']=='National Park':
        park_name=park_info['name']
        html=urllib.request.urlopen("https://www.insidehook.com/feature/travel/best-activity-all-59-us-national-parks").read()
        soup=BeautifulSoup(html,'html.parser')
        switch=True
        for link in soup.find_all('p'):
            title=link.get_text()
            title=title.rstrip()
            if switch==False:
                return 'At this park, the online lifestyle publication, Inside Hook, recommends you: '+title.replace('\xa0',' ')
            elif park_name in title:
                switch=False
                continue
    else:
        return('The online lifestyle publication, Inside Hook, does not have any recommendations for this park. Try a National
Park and they very well might!')





#WEATHER FORECAST
#Return tomorrow's forecast at the park and then the mean of the High and Low extended forecast
#Note, I know weather.gov has an API but decided to scrape to validate one of the requirements of the project rubric. I ran this
by Yigal and he is okay with it

def get_weather(park_info):

    lat=str(park_info['coordinates'][0])
    lon=str(park_info['coordinates'][1])

    #Connect to website and retrieve html
    html=urllib.request.urlopen("http://forecast.weather.gov/MapClick.php?lat="+lat+"&lon="+lon).read()
    soup=BeautifulSoup(html,'html.parser')
    forecast=soup.find(id="seven-day-forecast")
    forecast_info = forecast.find_all(class_="tombstone-container")

    weather_forecast=dict()
    weather_forecast['temps_high']=[]
    weather_forecast['temps_low']=[]
    switch=False
    count=0

    #Get the forecast for tomorrow. Since there is no period "tomorrow" in the html, we have to create a switch, because
tomorrow could be any day of the week.
    #the html has "Today, This Afternoon, Tonight, etc." but no "Tomorrow"
    for item in forecast_info:
        period=item.find(class_="period-name").get_text()
        description=item.find("img")
        description2=description['title']
        temperature=item.find(class_="temp").get_text()

        if switch==True:
            weather_forecast['tomorrows_forecast']=[period,description2,temperature]
            break
        elif period=='Tonight':
            switch=True
            continue
        else:
            continue

    #Get the high and low temps in the extended forecast so we can return and average
    for item in forecast_info:
        temperature=item.find(class_="temp").get_text()
        temperature=temperature.split(': ')
        temperature[1]=temperature[1][:-3]
        temperature[1]=float(temperature[1])
        if temperature[0]=='High':
            weather_forecast['temps_high'].append(temperature[1])
        elif temperature[0]=='Low':
            weather_forecast['temps_low'].append(temperature[1])

    #Get the average and count of forecasted high and low temps
    #This way we can return the mean high and low temperatures for the following number of periods given in the extended
forecast (Ex. The 5 period average high is 45 degrees F)
    weather_forecast['high_count']=len(weather_forecast['temps_high'])
    weather_forecast['high_mean']=mean(weather_forecast['temps_high'])
    weather_forecast['low_count']=len(weather_forecast['temps_low'])
    weather_forecast['low_mean']=mean(weather_forecast['temps_low'])

    return weather_forecast


# GOOGLE MAPS
# mention that this required pip install gmaps

def park_and_climbs_visual(park_info, Top5, goog_api_key):
    if Top5 is None:
```

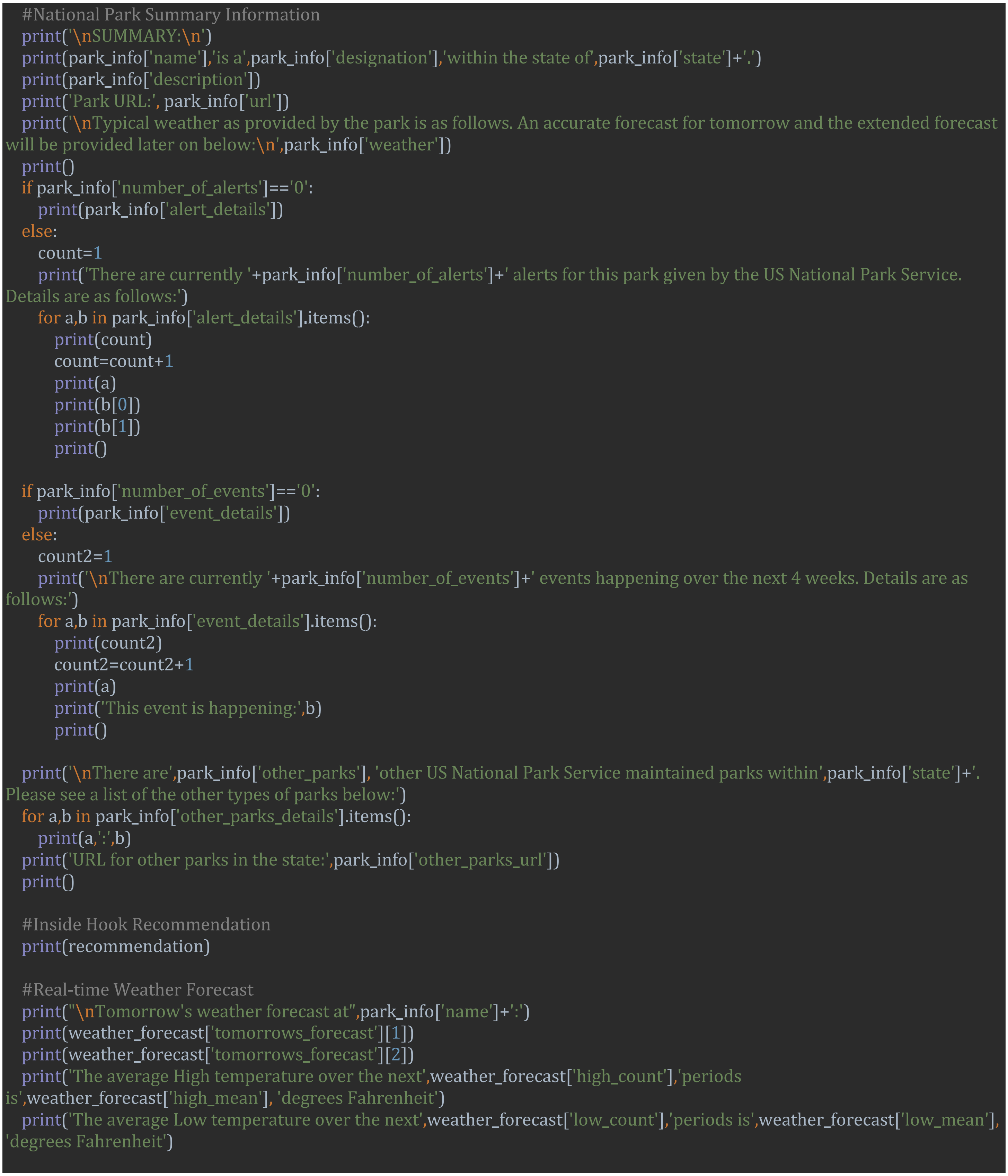
```
    print('There are no climbs to be shown. Please re-run the program with broader specifications')
    return
  else:
    gmaps.configure(api_key=goog_api_key)

    ntl_park_coords = []
    ntl_park_coords.append(park_info['coordinates'])

    # Create the map figure
    fig = gmaps.figure()

    # Create the National Park symbol that is blue so it stands out from the climbs
    park_dot = gmaps.symbol_layer(ntl_park_coords, fill_color='green', stroke_color='blue', scale=7)

    # Create Markers for the Top 5 climbing routes
    climb_locations = [climb['coordinates'] for climb in Top5]
    climb_names = [climb['name'] + '-' + climb['rating'] for climb in Top5]
    info_box_template = """
    <dl>
    <dt>Name</dt><dd>{name}</dd><br><br>
    <dt>Rating</dt><dd>{rating}</dd><br><br>
    <dt>Stars</dt><dd>{stars}</dd><br><br>
    <dt>Climb Type</dt><dd>{type}</dd><br><br>
    <dt>Pitches</dt><dd>{pitches}</dd><br><br>
    <dt>Distance From National Park (miles)</dt><dd>{distance}</dd><br><br>
    <a href={url}>Link to Climb Info</a>
    </dl>
    """
    climb_info = [info_box_template.format(**climb) for climb in Top5]

    # Create park and climb marker layers and add them to the GMAPS figure (fig for short)
    climb_marker_layer = gmaps.marker_layer(climb_locations, hover_text=climb_names, info_box_content=climb_info)
    park_marker_layer = gmaps.marker_layer(ntl_park_coords,
                         hover_text=park_info['name'] + ', ' + park_info['state'],
                         info_box_content=park_info['description'])
    fig = gmaps.figure()
    fig.add_layer(climb_marker_layer)
    fig.add_layer(park_marker_layer)
    fig.add_layer(park_dot)


    new_p=os.path.normpath(os.getcwd() + os.sep + os.pardir)   #The parent Directory LUKE_NELSON_homework5
    new_p_out=os.path.join(new_p,'Output')   #The folder "Output"
    os.chdir(new_p_out)   #Change cd to be the Output folder so that we can save html there
    embed_minimal_html('park_and_climbs_visual.html', views=[fig])   #Save the google map as html in Output folder
    url='file://'+os.path.join(new_p_out,'park_and_climbs_visual.html')   #Create url for Mac users
    webbrowser.open_new_tab('park_and_climbs_visual.html')   #For PC open the html in web browser
    webbrowser.open(url, new=2)  # For Mac open the html in web browser
```

Luke_Nelson_Summary.py

```
#code to pull from module, save API Keys, Print summary, create google map

import sqlite3
import park_and_climb
from sys import argv
import os

#Change cd to Output folder so that database table is saved there instead of in the code folder
new_p = os.path.normpath(os.getcwd() + os.sep + os.pardir)  # The parent Directory LUKE_NELSON_homework5
new_p_out = os.path.join(new_p, 'Output')  # The folder "Output"
os.chdir(new_p_out)  # Change cd to be the Output folder so that we can save the database table there


#Create a database to store your API keys so you don't have to manually re-enter them every time you run the program
conn = sqlite3.connect('park_and_climb')
c = conn.cursor()

#get the count of tables with the name to see if there are already API keys
c.execute(''' SELECT count(name) FROM sqlite_master WHERE type='table' AND name='api_keys' ''')

#if the count is 1, then table exists
#Allow for the user to enter new api keys if they want otherwise they dont have to go through the hassle of doing so each time
if c.fetchone()[0]==1 :
   re_create=input('It looks like there is already a table of API keys. Would you like to update your keys? [Yes/No]')
   if re_create=='Yes':
      c.execute('''DROP TABLE api_keys''')
      c.execute('''CREATE TABLE api_keys (key_type TEXT, key TEXT)''')
      nps_api_key=input('Please enter your NPS API Key:\n')
      mp_api_key=input('Please enter your Mountain Project API Key:\n')
      goog_api_key=input('Please enter your Google API Key:\n')
      c.execute('INSERT INTO api_keys VALUES(?, ?)', ('nps_api_key', nps_api_key))
      c.execute('INSERT INTO api_keys VALUES(?, ?)', ('mp_api_key', mp_api_key))
      c.execute('INSERT INTO api_keys VALUES(?, ?)', ('goog_api_key', goog_api_key))
      conn.commit()
   else:
      c.execute('SELECT key FROM api_keys WHERE key_type = "nps_api_key"')
      nps_api_key=c.fetchall()
      nps_api_key=nps_api_key[0][0]

      c.execute('SELECT key FROM api_keys WHERE key_type = "mp_api_key"')
      mp_api_key=c.fetchall()
      mp_api_key=mp_api_key[0][0]

      c.execute('SELECT key FROM api_keys WHERE key_type = "goog_api_key"')
      goog_api_key=c.fetchall()
      goog_api_key=goog_api_key[0][0]


else:
   c.execute('''CREATE TABLE api_keys (key_type TEXT, key TEXT)''')
   nps_api_key=input('Please enter your NPS API Key:\n')
   mp_api_key=input('Please enter your Mountain Project API Key:\n')
   goog_api_key=input('Please enter your Google API Key:\n')
```



```
    c.execute('INSERT INTO api_keys VALUES(?, ?)', ('nps_api_key', nps_api_key))
    c.execute('INSERT INTO api_keys VALUES(?, ?)', ('mp_api_key', mp_api_key))
    c.execute('INSERT INTO api_keys VALUES(?, ?)', ('goog_api_key', goog_api_key))
    conn.commit()


#Change the cd back so that we can access the park_and_climb module
new_p_code = os.path.join(new_p, 'Code')  # The folder "Output"
os.chdir(new_p_code)  # Change cd to be the Output folder so that we can save the database table there

#Request the user for the inputs required for the functions to return the data


#The name of the park they would like to visit
park_name=input('Please enter the name of a U.S. National Park Service maintained park.\nEx. Rocky Mountain National Park, Dinosaur National Monument, Lincoln Memorial, etc.\n')

#The easiest difficulty of climb the person is willing to climb
easiest=input('Please enter the easiest difficulty you are willing to climb.\nEx. 5.8 (Climbing difficulties range from 5.6 - 5.14 and always include the decimal point)\n')

#The hardest difficulty of climb the person is willing to climb
hardest=input('Please enter the hardest difficulty you are willing to climb.\nEx. 5.10 (Climbing difficulties range from 5.6 - 5.14 and always include the decimal point)\n')

#The distance in miles from the park the person is willing to consider for a climbing route
radius=input('Please enter the distance (in miles) from the park you are willing to consider a climbing route. You do not need to enter units.\nEx. 30\n')

#The climbers preferred climbing route type
preferred_type=input('Please enter the type of climb you are looking for. Enter [Sport/Trad/TR/Alpine]:\n')

#Dictionary of data for the park chosen by the user
park_info=park_and_climb.national_park_info(park_name, nps_api_key)

if park_info is None:
    print('Please re-run the program and try typing in new conditions!')

else:
    #List of dictionaries where each dictionary is one of the highest rated climbs returned
    climb_info=park_and_climb.get_climbs(park_info, mp_api_key, easiest, hardest, radius, preferred_type)


    #String Recommendation from Inside Hook for what to do at the park if it is a National Park
    recommendation=park_and_climb.get_recommendations(park_info)


    #Dictionary of Weather forecast data at the location of the park, including forecast for tomorrow and data on extended temp forecast
    weather_forecast=park_and_climb.get_weather(park_info)

####SUMMARY INFORMATION######
```



```
  #National Park Summary Information
  print('\nSUMMARY:\n')
  print(park_info['name'],'is a',park_info['designation'],'within the state of',park_info['state']+'.')
  print(park_info['description'])
  print('Park URL:', park_info['url'])
  print('\nTypical weather as provided by the park is as follows. An accurate forecast for tomorrow and the extended forecast
will be provided later on below:\n',park_info['weather'])
  print()
  if park_info['number_of_alerts']=='0':
    print(park_info['alert_details'])
  else:
    count=1
    print('There are currently '+park_info['number_of_alerts']+' alerts for this park given by the US National Park Service.
Details are as follows:')
    for a,b in park_info['alert_details'].items():
      print(count)
      count=count+1
      print(a)
      print(b[0])
      print(b[1])
      print()

  if park_info['number_of_events']=='0':
    print(park_info['event_details'])
  else:
    count2=1
    print('\nThere are currently '+park_info['number_of_events']+' events happening over the next 4 weeks. Details are as
follows:')
    for a,b in park_info['event_details'].items():
      print(count2)
      count2=count2+1
      print(a)
      print('This event is happening:',b)
      print()

  print('\nThere are',park_info['other_parks'], 'other US National Park Service maintained parks within',park_info['state']+'.
Please see a list of the other types of parks below:')
  for a,b in park_info['other_parks_details'].items():
    print(a,':',b)
  print('URL for other parks in the state:',park_info['other_parks_url'])
  print()

  #Inside Hook Recommendation
  print(recommendation)

  #Real-time Weather Forecast
  print("\nTomorrow's weather forecast at",park_info['name']+':')
  print(weather_forecast['tomorrows_forecast'][1])
  print(weather_forecast['tomorrows_forecast'][2])
  print('The average High temperature over the next',weather_forecast['high_count'],'periods
is',weather_forecast['high_mean'], 'degrees Fahrenheit')
  print('The average Low temperature over the next',weather_forecast['low_count'],'periods is',weather_forecast['low_mean'],
'degrees Fahrenheit')
```

```
    #Information on returned climbing routes
    if climb_info is None:
        print('\nUnfortunately there are no climbs within your specified constraints. Please try broadening them. As a result, a map will not be printed at this time.')
    else:
        print('\nBelow you will find information on the highest rated climbs returned from your query:')
        for i in climb_info:
            for a,b in i.items():
                print(a+':',b)
            print()

        #See Google Map
        print('A new window should have been opened in your internet browser. Please navigate there to view where the park is relative to the climbing routes. Otherwise, an html has been saved in the Output folder - please open it to see the map. Hover over and click on the markers to get useful information.\nHave a wonderful adventure!')
        park_and_climb.park_and_climbs_visual(park_info,climb_info,goog_api_key)
```