

DPS PROJECT-1

Reflection:

```
In [8]: def FindBusinessBasedOnCity(cityToSearch, saveLocation1, collection):

    businesses = []
    for idx in range(len(collection.all())):
        data_point = collection.fetch(idx)

        if data_point['city'] == cityToSearch:
            businesses.append([data_point['name'], data_point['full_address'], data_point['city'], data_point['state']])

    f = open(saveLocation1, 'w')
    for attr in businesses:
        f.write('$'.join(str(s) for s in attr))
        f.write('\n')
    f.close()
```

The function FindBusinessBasedOnCity with the parameters cityToSearch, saveLocation1, and collection is created for the project. Assign the data from the "idx" variable to the variable "data point" after creating a list called "businesses" and iterating through the sample.db database. The full information, including "name," "full address," "city," and "state," should be saved to "saveLocation1" if the data point from all the collections in cityToSearch is present. Now use f.write() to divide those strings using a "\$" sign separator, and then save the file.

```
In [10]: def DistanceFunction(lat2, lon2, lat1, lon1):

    R = 3959
    phi1 = math.radians(lat1)
    phi2 = math.radians(lat2)
    delta_phi = math.radians(lat2 - lat1)
    delta_lambda = math.radians(lon2 - lon1)
    a = math.sin(delta_phi/2) * math.sin(delta_phi/2) + math.cos(phi1) * math.cos(phi2) * math.sin(delta_lambda/2) * math.sin(delta_lambda/2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    d = R * c
    return d
```

The distance between the two pairs of latitude and longitude [lat2, lon2] and [lat1, lon1] is determined using the Distance function from the Distance Algorithm. Here, I used the Spherical Law of Cosines with the parameters R, phi1 and phi2, and delta phi as the difference at two latitudes. Similarly, apply the "haversine" formula to longitude pairings with delta lambda. I got the distance "d" and use it in the function to FindBusinessBasedOnLocation.

```
In [9]: def FindBusinessBasedOnLocation(categoriesToSearch, myLocation, maxDistance, saveLocation2, collection):

    names = []

    # get my location
    lat1 = myLocation[0]
    lon1 = myLocation[1]

    # example category attribute in data point: 'categories': ['Restaurants', 'Buffets', 'Italian']
    for idx in range(len(collection.all())):
        data_point = collection.fetch(idx)
        lat2 = data_point['latitude']
        lon2 = data_point['longitude']

        # filter by location and category
        distance = DistanceFunction(lat2, lon2, lat1, lon1)
        if distance <= maxDistance:
            for category in categoriesToSearch:
                if category in data_point['categories']:
                    names.append(data_point['name'])

    f = open(saveLocation2, 'w')
    for name in names:
        f.write(name + '\n')
    f.close()
```

With the parameters categoriesToSearch, myLocation, maxDistance, saveLocation2, and collection, I now build a new function called "FindBusinessBasedOnLocation." Make a list named names, then loop through the database starting with sample. Get the data from the "idx" variable in the database, then get the position information for latitude and longitude [lat2], [lon2], from the allocated data point. Determine the distance using the latitude and longitude pairs that DistanceFunction has returned. if the distance is less than maxDistance (the longest path or distance possible to reach the specified place). Repeat the categoriesToSearch iteration process, adding the "names" list with the "data point" of the "name" if it is in the "data point," and saving it to the "saveLocation2." Using f.write(), record these names in the file, then shut it off.

Lessons Learnt:

I've mastered the use of Jupyter notebooks, an addition to the Anaconda IDE Platform. I was able to run the file using an ipynb file, which produced two output files for the test scenarios below. Which applied the spherical law of cosines' haversine formula to a distance algorithm employing pairs of latitude and longitude.

Reference: <https://www.movable-type.co.uk/scripts/latlong.html>

UnQLite is an embedded, self-contained, serverless, zero-configuration, transactional database design. It directly reads and write to ordinary disk files. The database file format is cross-platform, you can freely copy the database between 32-bit and 64-bit systems.

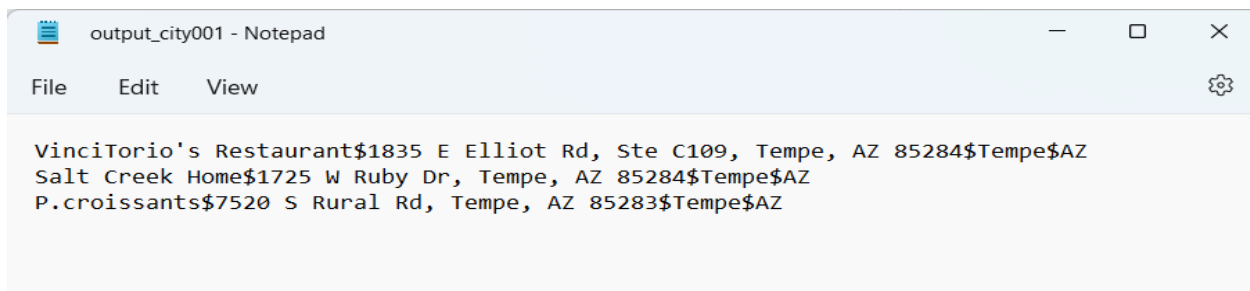
In, python UnQLite is a software-library and can be installed using:

pip install unqlite (or) conda install unqlite

*Make sure to install necessary packages for supporting the unqlite package.

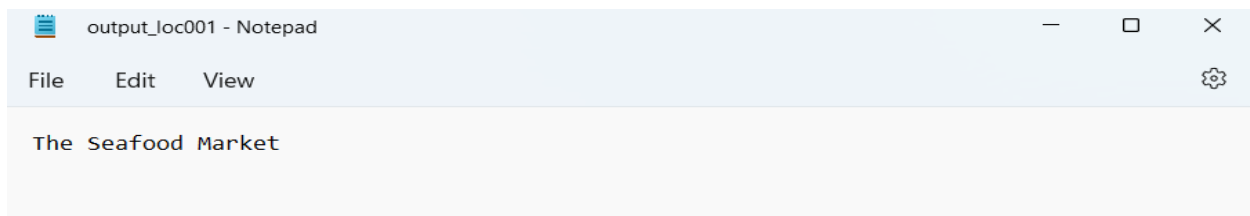
Output:

For the output for function "**FindBusinessBasedOnCity**":



```
output_city001 - Notepad
File Edit View
VinciTorio's Restaurant$1835 E Elliot Rd, Ste C109, Tempe, AZ 85284$Tempe$AZ
Salt Creek Home$1725 W Ruby Dr, Tempe, AZ 85284$Tempe$AZ
P.croissants$7520 S Rural Rd, Tempe, AZ 85283$Tempe$AZ
```

For the output for function "**FindBusinessBasedOnLocation**":



```
output_loc001 - Notepad
File Edit View
The Seafood Market
```

Result of my code:

Printed result for the test-case for “**FindBusinessBasedOnCity**”:

```
In [20]: # Test City 1

true_results = ["VinciTorio's Restaurant$1835 E Elliot Rd, Ste C109, Tempe, AZ 85284$Tempe$AZ", "P.croissants$7520 S Rural Rd, Tempe, AZ 85284$Tempe$AZ"]

try:
    FindBusinessBasedOnCity('Tempe', 'output_city001.txt', data)
except NameError as e:
    print ('The FindBusinessBasedOnCity function is not defined! You must run the cell containing the function before running this test case.')
except TypeError as e:
    print ("The FindBusinessBasedOnCity function is supposed to accept three arguments. Yours does not!")

try:
    opf = open('output_city001.txt', 'r')
except FileNotFoundError as e:
    print ("The FindBusinessBasedOnCity function does not write data to the correct location.")

lines = opf.readlines()
if len(lines) != 3:
    print ("The FindBusinessBasedOnCity function does not find the correct number of results, should be 3.")

lines = [line.strip() for line in lines]
if sorted(lines) == sorted(true_results):
    print ("Correct! You FindBusinessByCity function passes these test cases. This does not cover all possible test edge cases, however, so make sure that your function covers them before submitting!")
```

Correct! You FindBusinessByCity function passes these test cases. This does not cover all possible test edge cases, however, so make sure that your function covers them before submitting!

Printed result for the test-case for “**FindBusinessBasedOnLocation**”:

```
# Test Location 1

true_results = ['The Seafood Market']

try:
    FindBusinessBasedOnLocation(['Specialty Food'], [33.3482589, -111.9088346], 10, 'output_loc001.txt', data)
except NameError as e:
    print ('The FindBusinessBasedOnLocation function is not defined! You must run the cell containing the function before running this test case.')
except TypeError as e:
    print ("The FindBusinessBasedOnLocation function is supposed to accept five arguments. Yours does not!")

try:
    opf = open('output_loc001.txt', 'r')
except FileNotFoundError as e:
    print ("The FindBusinessBasedOnLocation function does not write data to the correct location.")

lines = opf.readlines()
if len(lines) != 1:
    print ("The FindBusinessBasedOnLocation function does not find the correct number of results, should be only 1.")

lines = [line.strip() for line in lines]
if sorted(lines) == sorted(true_results):
    print ("Correct! Your FindBusinessBasedOnLocation function passes these test cases. This does not cover all possible edge cases, however, so make sure your function does before submitting.")
```

Correct! Your FindBusinessBasedOnLocation function passes these test cases. This does not cover all possible edge cases, so make sure your function does before submitting.