# Lab - Integrate a REST API in a Python Application

## Objectives

**Part 1: Launch the DEVASC VM**

**Part 2: Demonstrate the Graphhopper Directions Application**

**Part 3: Get an API Key**

**Part 4: Build the Graphhopper Geocoding Application**

**Part 5: Build the Graphhopper Routing Application**

**Part 6: Test Full Application Functionality**

## Background / Scenario

In this lab, you will create an application in Visual Studio Code (VS Code) that retrieves JSON data from the Graphhopper Directions API, parses the data, and formats it for output to the user. You will use the GET Route request from the Graphhopper Directions API. Review the Directions API documentation here:

https://docs.graphhopper.com/

**Note**: If the above link no longer works, search for "Graphhopper Directions API Documentation".

## Required Resources

- 1 PC with operating system of your choice
- Virtual Box or VMWare
- DEVASC Virtual Machine

## Instructions

## Part 1: Launch the DEVASC VM

If you have not already completed the **Lab - Install the Virtual Machine Lab Environment**, do so now. If you have already completed that lab, launch the DEVASC VM now.

## Part 2: Demonstrate the Graphhopper Directions Application

Your instructor may demonstrate the Graphhopper Directions Application and show you the script used to create it. However, you will create this script step by step in this lab.

The application asks for a starting location and a destination and the mode of transportation. It then requests JSON data from the Graphhopper Geocoding and Routing APIs, parses it, and displays useful information.

```
+++++++++++++++++++++++++++++++++++++++++++++
Vehicle profiles available on Graphhopper:
+++++++++++++++++++++++++++++++++++++++++++++
car, bike, foot
+++++++++++++++++++++++++++++++++++++++++++++
Enter a vehicle profile from the list above: car
Starting Location: Washington, D.C.
Geocoding API URL for Washington, District of Columbia, United States (Location Type:
city)
```

```
https://graphhopper.com/api/1/geocode?q=Washington%2C+D.C.&limit=1&key=a0552be1-e9b3-
4cc8-b74a-9b4757a7e958
Destination: Baltimore, Maryland
Geocoding API URL for Baltimore, Maryland, United States (Location Type: city)
https://graphhopper.com/api/1/geocode?q=Baltimore%2C+Maryland&limit=1&key=a0552be1-
e9b3-4cc8-b74a-9b4757a7e958
=================================================
Routing API Status: 200
Routing API URL:
https://graphhopper.com/api/1/route?key=a0552be1-e9b3-4cc8-b74a-
9b4757a7e958&vehicle=car&point=38.8950368%2C-77.0365427&point=39.2908816%2C-76.610759
=================================================
Directions from Washington, District of Columbia, United States to Baltimore,
Maryland, United States by car
=================================================
Distance Traveled: 38.6 miles / 62.1 km
Trip Duration: 00:51:29
=================================================
Continue onto 15th Street Northwest ( 0.4 km / 0.3 miles )
Turn right onto New York Avenue Northwest ( 0.9 km / 0.6 miles )
Turn slight right onto K Street Northwest ( 0.2 km / 0.1 miles )
Turn left onto 7th Street Northwest ( 0.1 km / 0.1 miles )
Turn right onto New York Avenue Northwest ( 7.6 km / 4.7 miles )
Keep left onto Baltimore-Washington Parkway and drive toward Baltimore ( 51.6 km /
32.1 miles )
Turn right onto West Baltimore Street ( 0.6 km / 0.4 miles )
Turn left onto North Charles Street ( 0.2 km / 0.1 miles )
Turn right onto East Lexington Street ( 0.4 km / 0.2 miles )
Turn right onto Guilford Avenue ( 0.0 km / 0.0 miles )
Arrive at destination ( 0.0 km / 0.0 miles )
=============================================

+++++++++++++++++++++++++++++++++++++++++++++
Vehicle profiles available on Graphhopper:
+++++++++++++++++++++++++++++++++++++++++++++
car, bike, foot
+++++++++++++++++++++++++++++++++++++++++++++
Enter a vehicle profile from the list above: q
```

**Note**: To see the JSON for the above output, you can copy the URL in a browser tab. However, you will need to replace **your_api_key** with the Graphhopper API key you obtain in Part 3.

## Part 3: Get an API Key

Before building the application, you need to complete the following steps to get a Graphhopper API key.

a. Go to: https://www.graphhopper.com/.

b. Click **Sign Up** at the top of the page.

c. Fill out the form to create a new account. For **Company**, enter **Cisco Networking Academy Student**.

d. After verifying your email account and log into Graphhopper, click **API Keys** at the top of the page. Click **Add API Key** to add a new API key. Provide a description of the API key if desired. Click **Add Key** to create the new API key.

**Note**: Check the SPAM folder for the confirmation email from Graphhopper if you have not received the confirmation email.

e. Copy your **API Key** to a text file for future use. This will be the key you use for the rest of this lab.

**Note**: Graphhopper may change the process for obtaining a key. If the steps above are no longer valid, search the internet for “steps to generate graphhopper api key”.

## Part 4: Build the Graphhopper Gecoding Application

In this part, you will create a Python script to send a URL request to the Graphhopper Geocoding API. You will then test your API calls. Throughout the rest of this lab, you will build your script in parts, saving the file with a new name each time. This will help with learning the pieces of the application as well as provide you a series of scripts that you can return to if you run into any problems in the current version of your application.

### Step 1: Create a New File in VS Code.

You can use any tools you want to enter in Python commands and execute the Python code. However, this lab will demonstrate building the application in VS Code.

a. Create a new folder **graphhopper** in **~/labs/devnet-src** directory.

b. Open **VS Code**. There is a shortcut on the **Desktop**, for your convenience.

c. Select **File** > Open **Folder...**

d. Navigate to the **~/labs/devnet-src/graphhopper** directory and click **OK**.

e. Select **File** > **New File**.

f. Select **File** > **Save as…** and name the file **graphhopper_parse-json_1.py** and click **Save**.

### Step 2: Importing modules for the application.

To begin your script for parsing JSON data, you will need to import two modules from the Python library: **requests** and **urllib.parse**. The **requests** module provides functions for retrieving JSON data from a URL. The **urllib.parse** module provides a variety of functions that will enable you to parse and manipulate the JSON data you receive from a request to a URL.

a. Add the following import statements at the top of your script.

```
import requests
import urllib.parse
```

b. Select **Terminal > New Terminal** to open a Terminal inside VS Code.

c. Save and run your script. You should get no errors. You should save and run your scripts often to test code functionality.

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_1.py
devasc@labvm:~/labs/devnet-src/graphhopper$
```

### Step 3: Build the URL for the request to the Graphhopper Geocoding API.

The first step in creating your API request is to construct the URL that your application will use to make the call. Initially, the URL will be the combination of the following variables:

- **geo_url** – the Geocoding URL to request longitude and latitude information for a location
- **route_url** – the Routing URL to request routing information from the point of origin to the destination
- **loc1** - the parameter to specify the point of origin
- **loc2** - the parameter to specify the destination
- **key** - the Graphhopper API key you retrieved from the Graphhopper website

a. Create variables to build the URL that will be sent in the request. In the following code, replace **your_api_key** with the API Key you copied from your Graphhopper account.

```
geocode_url = "https://graphhopper.com/api/1/geocode?"
route_url = "https://graphhopper.com/api/1/route?"
loc1 = "Washington, D.C."
loc2 = "Baltimore, Maryland"
key = "your_api_key"      # Replace with your Graphhopper API key
```

b. Combine the three variables **geocode_url**, **loc1**, and **key** to format the requested URLs for the origin's longitude and latitude. Use the **urlencode** method to properly format the address value. This function builds the parameters part of the URL and converts possible special characters in the address value into acceptable characters (e.g. space into "+" and a comma into "%2C"). For the **url** variable, we will use the origin location variable **loc1** and limit to one set of the latitude and longitude coordinates, and the Graphhopper API key. You will replace the **loc1** variable when creating a Geocoding function later in the lab.

```
url = geocode_url + urllib.parse.urlencode({"q":loc1, "limit": "1", "key":key})
```

c. Create variables to hold the reply of the requested URL and print the returned JSON data. The **replydata** variable holds the data from the reply of the Graphhopper URL. The **json_data** variable holds a Python's Dictionary representation of the **json** reply of the **get** method of the **requests** module. The **requests.get** will make the API call to the Graphhopper Geocoding API. The **print** statement will be used temporarily to check the returned data. You will replace this print statement with more sophisticated display options later in the lab.

```
replydata = requests.get(url)
json_data = replydata.json()
json_status = replydata.status_code
print(json_data)
```

d. Your final code should look like this, but with a different value for the key.

```
import requests
import urllib.parse

geocode_url = "https://graphhopper.com/api/1/geocode?"
route_url = "https://graphhopper.com/api/1/route?"
loc1 = "Washington, D.C."
loc2 = "Baltimore, Maryland"
key = "a0552be1-e9b3-4cc8-b74a-9b4757a7e958"  ## Replace with your API key

url = geocode_url + urllib.parse.urlencode({"q":loc1, "limit": "1", "key":key})

replydata = requests.get(url)
json_data = replydata.json()
json_status = replydata.status_code
print(json_data)
```

## Step 4: Test the URL request.

a. Save and run your **graphhopper_parse-json_1.py** script and verify it works.

b. Troubleshoot your code, if necessary. Although your output might be slightly different, you should get a JSON response similar to the following. Notice that the output is a dictionary with two key/value pairs. The value for the key **hits** is a list that includes additional dictionaries and lists. The key **point** includes the **lat** and **lng** key/value pair that you will use later in the lab.

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_1.py
{'hits': [{'point': {'lat': 38.8950368, 'lng': -77.0365427}, 'extent': [-77.1197949,
38.7916303, -76.909366, 38.995968], 'name': 'Washington', 'country': 'United States',
'countrycode': 'US', 'state': 'District of Columbia', 'osm_id': 5396194, 'osm_type':
'R', 'osm_key': 'place', 'osm_value': 'city'}], 'locale': 'default'}
devasc@labvm:~/labs/devnet-src/graphhopper$
```

c. Change the **loc1** variable. Rerun the script to get different results. To ensure the results you want, it is best to include both the city and the state for cities in the USA. When referring to cities in other countries, you can usually use either the English name for the city and country or the native name. For example:

```
loc1 = "Rome, Italy"
```

or

```
loc1 = "Roma, Italia"
```

## Step 5: Print the URL and check the status of the JSON request.

Now that you know the JSON request is working, you can add some more functionality to the application.

a. Save your script as **graphhopper_parse-json_2.py**.

b. Delete the **print(json_data)** statement as you no longer need to test that the request is properly formatted.

c. Add the statements below, which will do the following:

Later in this lab, you will add **elif** and **else** statements for different **status_code** values.

```
json_status = replydata.status_code

if json_status == 200:
    print("Geocoding API URL for " + loc1 + ":\n" + url)
```

- Use the **status_code** property to determine the responded HTTP status.
- Start an **if** loop that checks for a successful call, which is indicated by a returned value of 200. If the request is successful, add a print statement to display the constructed Geocoding request URL. The **\n** adds a blank line before displaying the URL.

## Step 6: Display the JSON data in JSONView.

a. Paste the URL for Washington, D.C in the Chromium browser address field.

b. The results only have two root dictionaries: hits and locale. Expand **hits** and investigate the rich data. Continue to expand the result as necessary until latitude and longitude coordinates for the origin or destination are displayed.

```
{
  - hits: [
      - {
          - point: {
                lat: 38.8950368,
                lng: -77.0365427
            },
          + extent: […],
            name: "Washington",
            country: "United States",
            countrycode: "US",
            state: "District of Columbia",
            osm_id: 5396194,
            osm_type: "R",
            osm_key: "place",
            osm_value: "city"
        }
  - ],
    locale: "default"
}
```

## Step 7: Build the Geocoding function.

So far, you have created for URL requesting the latitude and longitude of starting location and received the information from the API request. What about the information for the destination? You can reuse the same code for the second location by using a function.

In this step, you will create a function that accepts the location inputs and return the **status_code** value of the Geocoding API URL, longitude, and latitude of the location and additional Graphhopper information regarding the location.

a. The example here uses the following parameters as the location input into the Geocoding function.

```
loc1 = "Washington, D.C."
loc2 = "Baltimore, Maryland"
```

b. Delete the **geocode_url** statement because it will be part of the geocoding function.

c. Copy and paste the new function and replace all the statements below your API key definition. An URL is created to request the JSON data using the location and key arguments sent into the function. The function returns the values stored in the variables **json_status**, **lat**, and **lng**. if the API call to the Geocoding API was successful. Otherwise, the function will return the error **json_status** and null values for **lat** and **lng** variables.

```
def geocoding (location, key):
    geocode_url = "https://graphhopper.com/api/1/geocode?"
    url = geocode_url + urllib.parse.urlencode({"q":location, "limit": "1",
"key":key})

    replydata = requests.get(url)
    json_data = replydata.json()
    json_status = replydata.status_code
    print("Geocoding API URL for " + location + ":\n" + url)
    if json_status == 200:
        json_data = requests.get(url).json()
        lat=(json_data["hits"][0]["point"]["lat"])
        lng=(json_data["hits"][0]["point"]["lng"])
    else:
        lat="null"
        lng="null"
    return json_status,lat,lng
```

## Step 8: Test your new Geocoding function.

a. To use the function, you will create two variables to call the geocoding function and store the values returned from the function. Copy and paste the following lines to the end of script. The geocoding function takes the input variable values for loc1 and key and returns the tuple with the values of the **json_status**, **lat**, and **lng** variables. The function is called again for loc2.

```
orig = geocoding(loc1, key)
print(orig)
dest = geocoding(loc2, key)
print(dest)
```

b. Your final code should look like this, but with a different value for the key.

```
import requests
import urllib.parse

route_url = "https://graphhopper.com/api/1/route?"
loc1 = "Washington, D.C."
loc2 = "Baltimore, Maryland"
key = "a0552be1-e9b3-4cc8-b74a-9b4757a7e958"  ### Replace with your API key

def geocoding (location, key):
    geocode_url = "https://graphhopper.com/api/1/geocode?"
    url = geocode_url + urllib.parse.urlencode({"q":location, "limit": "1", 
"key":key})

    replydata = requests.get(url)
    json_data = replydata.json()
    json_status = replydata.status_code
    print("Geocoding API URL for " + location + ":\n" + url)
    if json_status == 200:
        json_data = requests.get(url).json()
        lat = json_data["hits"][0]["point"]["lat"]
        lng = json_data["hits"][0]["point"]["lng"]
    else:
        lat="null"
        lng="null"
    return json_status,lat,lng

orig = geocoding(loc1, key)
print(orig)
dest = geocoding(loc2, key)
print(dest)
```

c. Save and run your **graphhopper_parse-json_2.py** script and verify it works. Troubleshoot your code, if necessary. You should get output similar to the following. Notice your key and the origin and destination are embedded in the URL request. Notice the returned results from the function as a tuple with the status, latitude and longitude values.

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_2.py
Geocoding API URL for Washington, D.C.:
https://graphhopper.com/api/1/geocode?q=Washington%2C+D.C.&limit=1&key=a0552be1-e9b3-
4cc8-b74a-9b4757a7e958
(200, 38.8950368, -77.0365427)
Geocoding API URL for Baltimore, Maryland:
https://graphhopper.com/api/1/geocode?q=Baltimore%2C+Maryland&limit=1&key=a0552be1-
e9b3-4cc8-b74a-9b4757a7e958
(200, 39.2908816, -76.610759)
devasc@labvm:~/labs/devnet-src/graphhopper$
```

## Step 9: Display additional information.

Besides the point dictionary with latitude and longitude information, you can also display other related information to verify that the coordinate is correct for the input location.

a. Display the JSON data again in JSONview in Chrome browser. Notice the additional information available in the **hits** dictionary.

```
{
  - hits: [
      - {
          + point: {…},
          + extent: […],
            name: "Washington",
            country: "United States",
            countrycode: "US",
            state: "District of Columbia",
            osm_id: 5396194,
            osm_type: "R",
            osm_key: "place",
            osm_value: "city"
        }
  - ],
    locale: "default"
}
```

b. Remove the following print statement within the geocoding function. It will be replaced later in the function.

```
print("Geocoding API URL for " + location + ":\n" + url)
```

c. Within the geocoding function, add the new variables **name**, **state**, **country**, and **value**. Because the keys **state** and **country** are not available for all locations, the **if** statements are used to assign the desired strings for the variable **new_loc** and return the value assigned to **new_loc**. For example, the key value **state** is not in the returned **hits** dictionary for **Beijing**, **China**.

```
    if json_status == 200:
        lat = json_data["hits"][0]["point"]["lat"]
        lng = json_data["hits"][0]["point"]["lng"]
        name = json_data["hits"][0]["name"]
        value = json_data["hits"][0]["osm_value"]

        if "country" in json_data["hits"][0]:
            country = json_data["hits"][0]["country"]
        else:
            country=""

        if "state" in json_data["hits"][0]:
            state = json_data["hits"][0]["state"]
        else:
            state=""

        if len(state) !=0 and len(country) !=0:
            new_loc = name + ", " + state + ", " + country
        elif len(state) !=0:
            new_loc = name + ", " + country
        else:
            new_loc = name

        print("Geocoding API URL for " + new_loc + " (Location Type: " + value + ")\n"
+ url)
    else:
        lat="null"
        lng="null"
        new_loc=location
    return json_status,lat,lng,new_loc
```

d. Save and run your **graphhopper_parse-json_2.py** script and verify it works.

```
devasc@labvm:~/labs/devnet-src/graphhopper$ graphhopper_parse-json_2.py
Geocoding API URL for Washington, D.C.:
https://graphhopper.com/api/1/geocode?q=Washington%2C+D.C.&limit=1&key=a0552be1-e9b3-
4cc8-b74a-9b4757a7e958
(200, 38.8950368, -77.0365427, 'Washington, District of Columbia, United States')
Geocoding API URL for Baltimore, Maryland:
https://graphhopper.com/api/1/geocode?q=Baltimore%2C+Maryland&limit=1&key=a0552be1-
e9b3-4cc8-b74a-9b4757a7e958
(200, 39.2908816, -76.610759, 'Baltimore, Maryland, United States')
```

## Step 10: Add User input for starting location and destination.

You have used static values for the location variables. However, the application requires that the user input these locations. Complete the following steps to update your application:

a. Save your script as **graphhopper_parse-json_3.py**.

b. Delete the current **loc1** and **loc2** variables.

c. Rewrite the **loc1** and **loc2** to be within a **while** loop in which it requests user input for the starting location and destination. The while loop allows the user to continue to make requests for different directions. Add the following code after the geocoding function. Be sure all the remaining code is correctly indented within the while loop.

```
while True:
    loc1 = input("Starting Location: ")
    orig = geocoding(loc1, key)
    print(orig)

    loc2 = input("Destination: ")
    dest = geocoding(loc2, key)
    print(dest)
```

## Step 11: Test user input functionality.

a. Save and run your **graphhopper_parse-json_3.py** script and verify it works. Troubleshoot your code, if necessary. You should get output similar to what is shown below. To end the program, enter **Ctrl+C**. You will get a **KeyboardInterrupt** error as shown in the output below. To stop the application more gracefully, you will add quit functionality in the next step.

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_3.py
Starting Location: Washington, D.C.
Geocoding API URL for Washington, D.C.:
```

```
https://graphhopper.com/api/1/geocode?q=Washington%2C+D.C.&limit=1&key=a0552be1-e9b3-
4cc8-b74a-9b4757a7e958
(200, 38.8950368, -77.0365427, 'Washington, District of Columbia, United States')
Starting Location: Baltimore, Maryland
Geocoding API URL for Baltimore, Maryland:
https://graphhopper.com/api/1/geocode?q=Baltimore%2C+Maryland&limit=1&key=a0552be1-
e9b3-4cc8-b74a-9b4757a7e958
(200, 39.2908816, -76.610759, 'Baltimore, Maryland, United States')
Starting Location: ^CTraceback (most recent call last):
  File "graphhopper_parse-json_3.py", line 46, in <module>
    loc1 = input("Starting Location: ")
KeyboardInterrupt

devasc@labvm:~/labs/devnet-src/graphhopper$
```

## Step 12: Add quit functionality to the application.

Instead of forcing the application to quit with a keyboard interrupt, you will add the ability for the user to enter **q** or **quit** as keywords to quit the application. Complete the following steps to update your application:

a. Add an if statement after each location variable to check if the user enters **q** or **quit**, as shown below.

```
while True:
    loc1 = input("Starting Location: ")
    if loc1 == "quit" or loc1 == "q":
        break
    orig = geocoding(loc1, key)
    print(orig)
    loc2 = input("Destination: ")
    if loc2 == "quit" or loc2 == "q":
        break
    dest = geocoding(loc2, key)
    print(dest)
```

## Step 13: Test the quit functionality.

Run your **graphhopper_parse-json_3.py** script four times to test each location variable. Verify that both **quit** and **q** will end the application. Troubleshoot your code, if necessary. You should get output similar to the following.

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_3.py
Starting Location: q
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_3.py
Starting Location: quit
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_3.py
Starting Location: Washington, D.C.
Geocoding API URL for Washington, District of Columbia, United States (Location Type:
city)
https://graphhopper.com/api/1/geocode?q=Washington%2C+D.C.&limit=1&key=a0552be1-e9b3-
4cc8-b74a-9b4757a7e958
(200, 38.8950368, -77.0365427, 'Washington, District of Columbia, United States')
Destination q
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_3.py
Starting Location: Washington, D.C.
```

```
Geocoding API URL for Washington, District of Columbia, United States (Location Type: city)
https://graphhopper.com/api/1/geocode?q=Washington%2C+D.C.&limit=1&key=a0552be1-e9b3-4cc8-b74a-9b4757a7e958
(200, 38.8950368, -77.0365427, 'Washington, District of Columbia, United States')
Destination: quit
devasc@labvm:~/labs/devnet-src/graphhopper$
```

## Step 14: Handling errors within Geocoding function.

But what happens if the user did not enter any location information? For example, if the user just pressed enter accidently and did not input any locations. What if the static API key is not correct or no longer valid? What if the user entered an invalid location, such as #? This can all be addressed within the geocode function.

a. Save your script as **graphhopper_parse-json_4.py**.

b. A **while** statement can address no location information. Copy and add the following highlighted statement to the function. The function will continue to ask for the location until the user provides the information.

```
def geocoding (location, key):
    while location == "":
        location = input("Enter the location again: ")
    geocode_url = "https://graphhopper.com/api/1/geocode?"
    url = geocode_url + urllib.parse.urlencode({"q":location, "limit": "1", "key":key})
```

c. Save and run **graphhopper_parse-json_4.py**. Press Enter without entering any starting location information to verify the while loop is working correctly.

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_4.py
Starting Location:
Enter the location again: Washington, D.C.
Geocoding API URL for Washington, District of Columbia, United States (Location Type: city)
https://graphhopper.com/api/1/geocode?q=Washington%2C+D.C.&limit=1&key=a0552be1-e9b3-4cc8-b74a-9b4757a7e958
(200, 38.8950368, -77.0365427, 'Washington, District of Columbia, United States')
Destination:
Enter the location again: Baltimore, Maryland
Geocoding API URL for Baltimore, Maryland, United States (Location Type: city)
https://graphhopper.com/api/1/geocode?q=Baltimore%2C+Maryland&limit=1&key=a0552be1-e9b3-4cc8-b74a-9b4757a7e958
(200, 39.2908816, -76.610759, 'Baltimore, Maryland, United States')
Starting Location: q
devasc@labvm:~/labs/devnet-src/graphhopper$
```

d. To test the invalid API key, copy and paste one of the Geocoding API URL into the Chromium browser address field and add an extra character to the API key at the end of URL. You will receive a message, similar to the example below:

```
{
       message: "Wrong credentials. Register and get a valid API key at https://www.graphhopper.com/developers/"
}
```

e. To display the error message in the script, add the highlighted print statement in Geocoding function.

```
            print("Geocoding API URL for " + new_loc + " (Location Type: " + value + ")\n" 
+ url)
        else:
            lat="null"
            lng="null"
            new_loc=location
            print("Geocode API status: " + str(json_status) + "\nError message: " + 
json_data["message"])
        return json_status,lat,lng,new_loc

while True:
    loc1 = input("Starting Location: ")
```

f. Edit the API key by adding an extra character. Save and run the script to verify the JSON response status code and the error message is displayed when there is an invalid key.

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_4.py
Starting Location: Washington, D.C.
Geocode API status: 401
Error message: Wrong credentials. Register and get a valid API key at 
https://www.graphhopper.com/developers/
(401, 'null', 'null', 'Washington, D.C.')
Destination: q
devasc@labvm:~/labs/devnet-src/graphhopper$
```

g. Correct the API key and save the script again before moving to the next part.

h. What if the user entered invalid characters, such as **#** or **!**? Or other invalid inputs?

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_4.py
Starting Location: slkdjf
Traceback (most recent call last):
  File "graphhopper_parse-json_4.py", line 54, in <module>
    orig = geocoding(loc1, key)
  File "graphhopper_parse-json_4.py", line 20, in geocoding
    lat = json_data["hits"][0]["point"]["lat"]
IndexError: list index out of range
devasc@labvm:~/labs/devnet-src/graphhopper$
```

i. Copy and paste any successful Geocoding API URL from previous output into the Chromium browser address field. For example, replace **Washington%2C+D.C.** with **slkdjf**.

```
https://graphhopper.com/api/1/geocode?q=slkdjf&limit=1&key=a0552be1-e9b3-4cc8-b74a-
9b4757a7e958
```

j. In this example, you received an empty **hits** list for the successful API request.

```
{
hits: [ ],
locale: "default"
}
```

k. Update the test condition of the **if** statement so it tests for a successful API request and the **hits** results is not empty.

```
    if json_status == 200 and len(json_data["hits"]) !=0:
        json_data = requests.get(url).json()
```

l. Save and test **graphhopper_parse-json_4.py**.

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_4.py
Starting Location: slkdjf
Traceback (most recent call last):
  File "graphhopper_parse-json_4.py", line 52, in <module>
    orig = geocoding(loc1, key)
  File "graphhopper_parse-json_4.py", line 45, in geocoding
    print("Geocode API status: " + str(json_status) + "\nError message: " +
json_data["message"])
KeyError: 'message'
devasc@labvm:~/labs/devnet-src/graphhopper$
```

With the **and** operator, both **json_status == 200** and **len(json_data["hits"]) !=0** have to be true for the statement to be true. In this example, the len(json_data["hits"] is 0, but the json_status is 200, so the statement is false. However, no error message was returned from the API request and json_data["message"] is not valid. To correct this, add a nested **if** statement within else portion of the **if** statement. Indent the print statement for error message.

```
    else:
        lat="null"
        lng="null"
        new_loc=location
        if json_status != 200:
            print("Geocode API status: " + str(json_status) + "\nError message: " +
json_data["message"])
    return json_status,lat,lng,new_loc
```

## Part 5: Build the Graphhopper Routing Application

In the previous part, you have built an application that accepts user input and returns the latitude and longitude information for an origin and destination location.

In this part, you will finish the application to provide routing information between the starting and ending locations. Furthermore, you will also provide information regarding total travelled distance and trip duration. In addition, you will add the vehicle feature to your application, where the user can choose the method of travel on land.

### Step 1: Build the URL for the request to Graphhopper Routing API.

In this step, you will build the URL for the Routing API request if the Geocoding API calls were successfully.

a. Save your script as **graphhopper_parse-json_5.py**.

b. The **print(orig)** and **print(dest)** statements can be deleted at this time because you have verified the status, latitude, and longitude values returned from the Geocoding function multiple times.

c. The Geocoding function returns a tuple with these three values: **json_status**, **latitude**, and **longitude**. Add a printer statement with a divider and an **if** statement to check if the geocoding function returns a successful status for both locations.

```
while True:
    loc1 = input("Starting Location: ")
    if loc1 == "quit" or loc1 == "q":
        break
    orig = geocoding(loc1, key)
    loc2 = input("Destination: ")
    if loc2 == "quit" or loc2 == "q":
        break
    dest = geocoding(loc2, key)
    print("=================================================")
    if orig[0] == 200 and dest[0] == 200:
```

d. At the end of the script, add the following lines of code to request the routing information for the Graphhopper Routing API and print out the URL for validation using JSONView.

- **op**: the string that provides the latitude and longitude of the origin
- **dp**: the string that provides the latitude and longitude of the destination
- **paths_url**: URL to request the route between the origin and destination
- **paths_status**: the status code returned from URL to request the route between the origin and destination
- **paths_data**: the JSON data returned from the URL to request the route between the origin and destination

```
    if orig[0] == 200 and dest[0] == 200:
        op="&point="+str(orig[1])+"%2C"+str(orig[2])
        dp="&point="+str(dest[1])+"%2C"+str(dest[2])
        paths_url = route_url + urllib.parse.urlencode({"key":key}) + op + dp
        paths_status = requests.get(paths_url).status_code
        paths_data = requests.get(paths_url).json()
        print("Routing API Status: " + str(paths_status) + "\nRouting API URL:\n" +
paths_url)
```

e. Run your **graphhopper_parse-json_5.py** script. Verify that both **quit** and **q** will end the application. Troubleshoot your code, if necessary. You should get output similar to the following. The highlighted text below is the URL to request the routing information from the Routing API.

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_5.py
Starting Location: Washington, D.C.
Geocoding API URL for Washington, District of Columbia, United States (Location Type:
city)
https://graphhopper.com/api/1/geocode?q=Washington%2C+D.C.&limit=1&key=a0552be1-e9b3-
4cc8-b74a-9b4757a7e958
Destination: Baltimore, Maryland
Geocoding API URL for Baltimore, Maryland, United States (Location Type: city)
https://graphhopper.com/api/1/geocode?q=Baltimore%2C+Maryland&limit=1&key=a0552be1-
e9b3-4cc8-b74a-9b4757a7e958
=================================================
Routing API Status: 200
Routing API URL:
https://graphhopper.com/api/1/route?key=a0552be1-e9b3-4cc8-b74a-
9b4757a7e958&point=38.8950368%2C-77.0365427&point=39.2908816%2C-76.610759
Starting Location: q
devasc@labvm:~/labs/devnet-src/graphhopper$
```

## Step 2: Display trip summary information to include duration and distance.

a. Copy and paste the Routing URL in the Chromium browser address field from the previous step.

b. The results only have three root dictionaries: **hints**, **info**, and **paths**. Expand **paths** if necessary and investigate the rich data. The highlighted information below provides the duration total distance travelled.

```
{
  + hints: {…},
  + info: {…},
  - paths: [
      - {
            distance: 62074.783,
            weight: 4020.981119,
            time: 3089864,
            transfers: 0,
            points_encoded: true,
            bbox: […],
            points:
<output omitted>
```

c. The following **if** statement tests the response status for the Routing APIs. When the response status returns as 200 for the Routing API, the variable **paths_data** stores the JSON data returned from Routing API. Several **print** statements are added displaying the from and to locations, as well as the **time** and **distance** keys for the trip as strings.

The additional statements also include print statements that will display a double line before the next request for a starting location. Make sure these statements are embedded in the **while True** function.

```
        print("=================================================")
        print("Directions from " + orig[3] + " to " + dest[3])
        print("=================================================")
        if paths_status == 200:
            print("Distance Traveled: " + str(paths_data["paths"][0]["distance"]) + " m")
            print("Trip Duration: " + str(paths_data["paths"][0]["time"]) + " millisec")
            print("=================================================")
```

d. Save and run **graphhopper_parse-json_5.py** to see the following output.

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_5.py
Starting Location: Washington, D.C.
Geocoding API URL for Washington, District of Columbia, United States (Location Type: city)
https://graphhopper.com/api/1/geocode?q=Washington%2C+D.C.&limit=1&key=a0552be1-e9b3-4cc8-b74a-9b4757a7e958
Destination: Baltimore, Maryland
Geocoding API URL for Baltimore, Maryland, United States (Location Type: city)
https://graphhopper.com/api/1/geocode?q=Baltimore%2C+Maryland&limit=1&key=a0552be1-e9b3-4cc8-b74a-9b4757a7e958
=================================================
Routing API Status: 200
Routing API URL:
https://graphhopper.com/api/1/route?key=a0552be1-e9b3-4cc8-b74a-9b4757a7e958&point=38.8950368%2C-77.0365427&point=39.2908816%2C-76.610759
=================================================
Directions from Washington, District of Columbia, United States to Baltimore, Maryland, United States
=================================================
Distance Traveled: 62074.812 m
Trip Duration: 3089865 millisec
=================================================
Starting Location: q
devasc@labvm:~/labs/devnet-src/graphhopper$
```

e. By default, Graphhopper uses the metric system and displays the distance in the unit of meters. There is no request parameter to change data to the imperial system. Therefore, you can convert your application to display imperial values and convert. Create two variables, **miles** and **km**, to convert the unit of distance results.

```
        if paths_status == 200:
            miles = (paths_data["paths"][0]["distance"])/1000/1.61
            km = (paths_data["paths"][0]["distance"])/1000
```

f. Replace the Distance Traveled print statement as shown below that displays the distance in both measuring systems using the **miles** and **km** variables. The extra decimal places for kilometers and miles are not helpful. Use the **"{:.1f}".format** argument to format the float values to 1 decimal place, as shown below.

```
        if paths_status == 200:
            miles = (paths_data["paths"][0]["distance"])/1000/1.61
            km = (paths_data["paths"][0]["distance"])/1000
            print("Distance Traveled: {0:.1f} miles / {1:.1f} km".format(miles, km))
            print("Trip Duration: " + str(paths_data["paths"][0]["time"]) + "
millisec")
            print("=================================================")
```

g. Graphhopper reports the elapsed time in milliseconds. Add these lines of code to calculate the trip duration before the block of print statements to convert the elapse time so the trip duration time can be displayed in the form of **hh:mm:ss**. The modulo operator (**%**) (mod) returns the reminder, instead of the quotient.

```
        if paths_status == 200:
            miles = (paths_data["paths"][0]["distance"])/1000/1.61
            km = (paths_data["paths"][0]["distance"])/1000
            sec = int(paths_data["paths"][0]["time"]/1000%60)
            min = int(paths_data["paths"][0]["time"]/1000/60%60)
            hr = int(paths_data["paths"][0]["time"]/1000/60/60)
```

h. Replace the Trip Duration print statement as shown below. To display the time in the form of **hh:mm:ss**, The displayed numbers should be in the form of two digital whole number.. Use the **"{:02d}".format** argument to format the float values to a digital whole number, as shown below.

```
            print("Trip Duration: {0:02d}:{1:02d}:{2:02d}".format(hr, min, sec))
```

## Step 3: Test the parsing and formatting functionality.

Save and run your **graphhopper_parse-json_5.py** script to verify it works. Troubleshoot your code, if necessary. Make sure you have all the proper opening and closing parentheses. You should get output similar to the following.

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_5.py
Starting Location: Washington, D.C.
Geocoding API URL for Washington, District of Columbia, United States (Location Type:
city)
https://graphhopper.com/api/1/geocode?q=Washington%2C+D.C.&limit=1&key=a0552be1-e9b3-
4cc8-b74a-9b4757a7e958
Destination: Baltimore, Maryland
Geocoding API URL for Baltimore, Maryland, United States (Location Type: city)
https://graphhopper.com/api/1/geocode?q=Baltimore%2C+Maryland&limit=1&key=a0552be1-
e9b3-4cc8-b74a-9b4757a7e958
```

```
=================================================
Routing API Status: 200
Routing API URL:
https://graphhopper.com/api/1/route?key=a0552be1-e9b3-4cc8-b74a-
9b4757a7e958&point=38.8950368%2C-77.0365427&point=39.2908816%2C-76.610759
=================================================
Directions from Washington, District of Columbia, United States to Baltimore,
Maryland, United States
=================================================
Distance Traveled: 38.6 miles / 62.1 km
Trip Duration: 00:51:29
=================================================
Starting Location: q
devasc@labvm:~/labs/devnet-src/graphhopper$
```

## Step 4: Inspect the instructions list in the JSON data.

a. Now you are ready to display the step-by-step directions from the starting location to the destination. Return to the Chromium browser where earlier you viewed the output in JSONView. If you closed the browser, copy the Routing URL from last time you ran the program and paste it into the browser address bar.

b. Inside the **paths** list. The **paths** list includes one big dictionary with most of the JSON data. Find the **instructions** list and collapse each of the seven dictionaries inside, as shown below (click the "**-**" **minus** sign to toggle it to a "**+**" **plus** sign). If you are using different locations, you will probably have a different number of instructions list.

The highlighted information will be used as output for the completed application.

```
{
  + hints: {…},
  + info: {…},
  - paths: [
      - {
            distance: 62074.812,
            weight: 4020.982119,
            time: 3089865,
            transfers: 0,
            points_encoded: true,
            bbox: […],
            points:
<output omitted>
            instructions: [
              + {…},
              + {…},
              + {…},
              + {…},
            ],
            legs: [ ],
            details: { },
            ascend: 713.5285040140152,
            descend: 689.9790037870407,
            snapped_waypoints: "utklFdsduMadoAqmpA"
            }
        ]
    }
```

c. Expand the first dictionary in the **instructions** list. Each dictionary contains a **text** key with a value, such as “Continue onto ...”, as shown below. You need to parse the JSON data to extract the value for the **text** key to display inside your application.

```
  - paths: [
      - {
            distance: 62074.812,
            weight: 4020.982119,
            time: 3089865,
            transfers: 0,
            points_encoded: true,
            bbox: […],
            points:
<output omitted>
            instructions: [
              - {
                    distance: 424.291,
                    heading: 1.09,
                    sign: 0,
                  + interval: […],
                    text: "Continue onto 15th Street Northwest",
                    time: 47733,
                    street_name: "15th Street Northwest"
                },
              - {
                    distance: 912.037,
                    sign: 2,
                  + interval: […],
                    text: "Turn right onto New York Avenue Northwest",
                    time: 93195,
                    street_name: "New York Avenue Northwest"
                },
<output omitted>
```

## Step 5: Add a for loop to iterate through the instructions JSON data.

Complete the following steps to upgrade the application to display the value for the **text** key. You will do this by creating a for loop to iterate through the instructions list, displaying the text value for the **instructions** list from starting location to destination.

a. Save your script as **graphhopper_parse-json_6.py**.

b. Add a **for** loop, highlighted below, after the second double line print statement. The **for** loop iterates through the **instructions** list and does the following:

1) Creates two variables to store the text and distance data from the Routing API JSON data

2) Prints the **text** value.

3) Converts meter to miles.

4) Formats the mile and kilometer value to print only two decimal places with the **"{:.2f}".format** function.

c. Add a **print** statement that will display a double line before the application asks for another starting location, as shown below.

**Note**: The double line print statement is not indented within the for loop

```
            print("Trip Duration: {0:02d}:{1:02d}:{2:02d}".format(hr, min, sec))
            print("=============================================")
            for each in range(len(paths_data["paths"][0]["instructions"])):
                path = paths_data["paths"][0]["instructions"][each]["text"]
                distance = paths_data["paths"][0]["instructions"][each]["distance"]
                print("{0} ( {1:.1f} km / {2:.1f} miles )".format(path, distance/1000, 
distance/1000/1.61))
            print("=============================================")
```

## Step 6: Test the JSON iteration.

Save and run your **graphhopper_parse-json_6.py** script to verify it works. Troubleshoot your code, if necessary. You should get an output similar to the following:

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_6.py
Starting Location: Washington, D.C.
Geocoding API URL for Washington, District of Columbia, United States (Location Type: 
city)
https://graphhopper.com/api/1/geocode?q=Washington%2C+D.C.&limit=1&key=a0552be1-e9b3-
4cc8-b74a-9b4757a7e958
Destination: Baltimore, Maryland
Geocoding API URL for Baltimore, Maryland, United States (Location Type: city)
https://graphhopper.com/api/1/geocode?q=Baltimore%2C+Maryland&limit=1&key=a0552be1-
e9b3-4cc8-b74a-9b4757a7e958
=================================================
Routing API Status: 200
Routing API URL:
https://graphhopper.com/api/1/route?key=a0552be1-e9b3-4cc8-b74a-
9b4757a7e958&point=38.8950368%2C-77.0365427&point=39.2908816%2C-76.610759
=================================================
Directions from Washington, District of Columbia, United States to Baltimore, 
Maryland, United States
=================================================
Distance Traveled: 38.6 miles / 62.1 km
Trip Duration: 00:51:29
=================================================
Continue onto 15th Street Northwest ( 0.4 km / 0.3 miles )
Turn right onto New York Avenue Northwest ( 0.9 km / 0.6 miles )
Turn slight right onto K Street Northwest ( 0.2 km / 0.1 miles )
Turn left onto 7th Street Northwest ( 0.1 km / 0.1 miles )
Turn right onto New York Avenue Northwest ( 7.6 km / 4.7 miles )
Keep left onto Baltimore-Washington Parkway and drive toward Baltimore ( 51.6 km / 
32.1 miles )
Turn right onto West Baltimore Street ( 0.6 km / 0.4 miles )
Turn left onto North Charles Street ( 0.2 km / 0.1 miles )
Turn right onto East Lexington Street ( 0.4 km / 0.2 miles )
Turn right onto Guilford Avenue ( 0.0 km / 0.0 miles )
```

```
Arrive at destination ( 0.0 km / 0.0 miles )
=================================================
Starting Location: q
devasc@labvm:~/labs/devnet-src/graphhopper$
```

## Step 7: Report Routing API errors

Now you are ready to add a feature to let the users know that Graphhopper cannot provide a route between the two locations.

For example, if a user might enter locations that will return an invalid route. If so, then the application displays the URL and asks for a new starting location. The user has no idea what happened.

a. To cause the Routing API request to fail without user notification, try the following values in your application. You should see similar results.

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-json_6.py
Starting Location: Beijing, China
Geocoding API URL for 北京市, 中国 (Location Type: state)
https://graphhopper.com/api/1/geocode?q=Beijing%2C+China&limit=1&key=a0552be1-e9b3-4cc8-b74a-9b4757a7e958
Destination: Washington, D.C.
Geocoding API URL for Washington, District of Columbia, United States (Location Type: city)
https://graphhopper.com/api/1/geocode?q=Washington%2C+D.C.&limit=1&key=a0552be1-e9b3-4cc8-b74a-9b4757a7e958
=================================================
Routing API Status: 400
Routing API URL:
https://graphhopper.com/api/1/route?key=a0552be1-e9b3-4cc8-b74a-9b4757a7e958&point=40.190632%2C116.412144&point=38.8950368%2C-77.0365427
=================================================
Directions from 北京市, 中国 to Washington, District of Columbia, United States
=================================================
Starting Location: q
```

b. Note the Routing API status was returned as 400. Copy the URL to a Chromium browser tab. Notice that there are two key/value entries in the dictionary. By adding an else statement to the **if paths_status == 200:** statement, the application notifies the users with the error message received from the failed Graphhopper Routing API request.

```
            print("=================================================")
            for each in range(len(paths_data["paths"][0]["instructions"])):
                path = paths_data["paths"][0]["instructions"][each]["text"]
                distance = paths_data["paths"][0]["instructions"][each]["distance"]
                print("{0} ( {1:.1f} km / {2:.1f} miles )".format(path, distance/1000, distance/1000/1.61))
            print("=================================================")
        else:
            print("Error message: " + paths_data["message"])
            print("*************************************************")
```

## Step 8: Add more modes of transportation.

With free access to Graphhopper, you have a few different modes of transportation to choose from. The default method is by car.

a. Save your script as **graphhopper_parse-json_7.py**.

b. By adding the following lines of code after **while True:**, you allow the users to choose their mode of land transportation available in Graphhopper.

   When users enter a listed method, the method is assigned to the variable **vehicle**. If no match is found, then **car** is assigned to **vehicle**.

```
while True:
    print("\n+++++++++++++++++++++++++++++++++++++++++++++")
    print("Vehicle profiles available on Graphhopper:")
    print("+++++++++++++++++++++++++++++++++++++++++++++")
    print("car, bike, foot")
    print("+++++++++++++++++++++++++++++++++++++++++++++")
    profile=["car", "bike", "foot"]
    vehicle = input("Enter a vehicle profile from the list above: ")
    if vehicle == "quit" or vehicle == "q":
        break
    elif vehicle in profile:
        vehicle = vehicle
    else:
        vehicle = "car"
        print("No valid vehicle profile was entered. Using the car profile.")
```

c. Locate the variable **paths_url**. Update the query by adding a new field **vehicle**.

```
paths_url = route_url + urllib.parse.urlencode({"key":key, "vehicle":vehicle}) + op +
dp
```

d. Locate the print statement regarding the directions. Update the print statement with the highlighted text.

```
        print("=================================================")
        print("Directions from " + orig[3] + " to " + dest[3] + " by " + vehicle)
        print("=================================================")
```

## Part 6: Test Full Application Functionality

Run your **graphhopper_parse-json_7.py** script and verify it works. Troubleshoot your code, if necessary. Test all the features of the application. You should get output similar to the following.

```
devasc@labvm:~/labs/devnet-src/graphhopper$ python3 graphhopper_parse-
json_7.py

+++++++++++++++++++++++++++++++++++++++++++++
Vehicle profiles available on Graphhopper:
+++++++++++++++++++++++++++++++++++++++++++++
car, bike, foot
+++++++++++++++++++++++++++++++++++++++++++++
Enter a vehicle profile from the list above: bike
Starting Location: Washington, D.C.
Geocoding API URL for Washington, District of Columbia, United States (Location Type:
city)
https://graphhopper.com/api/1/geocode?q=Washington%2C+D.C.&limit=1&key=a0552be1-e9b3-
4cc8-b74a-9b4757a7e958
Destination: Baltimore, Maryland
Geocoding API URL for Baltimore, Maryland, United States (Location Type: city)
```

```
https://graphhopper.com/api/1/geocode?q=Baltimore%2C+Maryland&limit=1&key=a0552be1-
e9b3-4cc8-b74a-9b4757a7e958
=================================================
Routing API Status: 200
Routing API URL:
https://graphhopper.com/api/1/route?key=a0552be1-e9b3-4cc8-b74a-
9b4757a7e958&vehicle=bike&point=38.8950368%2C-77.0365427&point=39.2908816%2C-76.610759
=================================================
Directions from Washington, District of Columbia, United States to Baltimore,
Maryland, United States by bike
=================================================
Distance Traveled: 44.3 miles / 71.4 km
Trip Duration: 04:06:10
=================================================
Continue ( 0.0 km / 0.0 miles )
Turn left ( 0.0 km / 0.0 miles )
Turn right onto Ellipse Road Northwest ( 0.3 km / 0.2 miles )
Keep left onto Ellipse Road Northwest ( 0.2 km / 0.1 miles )
Turn left onto Constitution Avenue Northwest ( 0.0 km / 0.0 miles )
Turn left onto 15th St NW Cycle Track ( 0.4 km / 0.2 miles )
<output omitted>
Turn left onto North Charles Street ( 0.1 km / 0.1 miles )
Turn right onto East Lexington Street ( 0.4 km / 0.2 miles )
Turn right onto Guilford Avenue ( 0.0 km / 0.0 miles )
Arrive at destination ( 0.0 km / 0.0 miles )
=================================================


+++++++++++++++++++++++++++++++++++++++++++++
Vehicle profiles available on Graphhopper:
+++++++++++++++++++++++++++++++++++++++++++++
car, bike, foot
+++++++++++++++++++++++++++++++++++++++++++++
Enter a vehicle profile from the list above:
No valid vehicle profile was entered. Using the car profile.
Starting Location: Washington, D.C.
Geocoding API URL for Washington, District of Columbia, United States (Location Type:
city)
https://graphhopper.com/api/1/geocode?q=Washington%2C+D.C.&limit=1&key=a0552be1-e9b3-
4cc8-b74a-9b4757a7e958
Destination: Baltimore, Maryland
Geocoding API URL for Baltimore, Maryland, United States (Location Type: city)
https://graphhopper.com/api/1/geocode?q=Baltimore%2C+Maryland&limit=1&key=a0552be1-
e9b3-4cc8-b74a-9b4757a7e958
=================================================
Routing API Status: 200
Routing API URL:
https://graphhopper.com/api/1/route?key=a0552be1-e9b3-4cc8-b74a-
9b4757a7e958&vehicle=car&point=38.8950368%2C-77.0365427&point=39.2908816%2C-76.610759
=================================================
Directions from Washington, District of Columbia, United States to Baltimore,
Maryland, United States by car
```

```
=================================================
Distance Traveled: 38.6 miles / 62.1 km
Trip Duration: 00:51:29
=================================================
Continue onto 15th Street Northwest ( 0.4 km / 0.3 miles )
Turn right onto New York Avenue Northwest ( 0.9 km / 0.6 miles )
Turn slight right onto K Street Northwest ( 0.2 km / 0.1 miles )
Turn left onto 7th Street Northwest ( 0.1 km / 0.1 miles )
Turn right onto New York Avenue Northwest ( 7.6 km / 4.7 miles )
Keep left onto Baltimore-Washington Parkway and drive toward Baltimore ( 51.6 km /
32.1 miles )
Turn right onto West Baltimore Street ( 0.6 km / 0.4 miles )
Turn left onto North Charles Street ( 0.2 km / 0.1 miles )
Turn right onto East Lexington Street ( 0.4 km / 0.2 miles )
Turn right onto Guilford Avenue ( 0.0 km / 0.0 miles )
Arrive at destination ( 0.0 km / 0.0 miles )
=================================================

+++++++++++++++++++++++++++++++++++++++++++++
Vehicle profiles available on Graphhopper:
+++++++++++++++++++++++++++++++++++++++++++++
car, bike, foot
+++++++++++++++++++++++++++++++++++++++++++++
Enter a vehicle profile from the list above: car
Starting Location: Beijing, China
Geocoding API URL for 北京市, 中国 (Location Type: state)
https://graphhopper.com/api/1/geocode?q=Beijing%2C+China&limit=1&key=a0552be1-e9b3-
4cc8-b74a-9b4757a7e958
Destination: Washington, D.C.
Geocoding API URL for Washington, District of Columbia, United States (Location Type:
city)
https://graphhopper.com/api/1/geocode?q=Washington%2C+D.C.&limit=1&key=a0552be1-e9b3-
4cc8-b74a-9b4757a7e958
=================================================
Routing API Status: 400
Routing API URL:
https://graphhopper.com/api/1/route?key=a0552be1-e9b3-4cc8-b74a-
9b4757a7e958&vehicle=scooter&point=40.190632%2C116.412144&point=38.8950368%2C-
77.0365427
=================================================
Directions from 北京市, 中国 to Washington, District of Columbia, United States by
scooter
=================================================
Error message: Connection between locations not found
*************************************************

+++++++++++++++++++++++++++++++++++++++++++++
Vehicle profiles available on Graphhopper:
+++++++++++++++++++++++++++++++++++++++++++++
car, bike, foot
```

```
+++++++++++++++++++++++++++++++++++++++++++++
Enter a vehicle profile from the list above:
No valid vehicle profile was entered. Using the car profile.
Starting Location:
Enter location again: **Leeds, England**
Geocoding API URL for Leeds, England, United Kingdom (Location Type: city)
https://graphhopper.com/api/1/geocode?q=Leeds%2C+England&limit=1&key=a0552be1-e9b3-
4cc8-b74a-9b4757a7e958
Destination: **London, England**
Geocoding API URL for London, England, United Kingdom (Location Type: city)
https://graphhopper.com/api/1/geocode?q=London%2C+England&limit=1&key=a0552be1-e9b3-
4cc8-b74a-9b4757a7e958
=================================================
Routing API Status: 200
Routing API URL:
https://graphhopper.com/api/1/route?key=a0552be1-e9b3-4cc8-b74a-
9b4757a7e958&vehicle=car&point=53.7974185%2C-1.5437941&point=51.5074456%2C-0.1277653
=================================================
Directions from Leeds, England, United Kingdom to London, England, United Kingdom by
car
=================================================
Distance Traveled: 195.2 miles / 314.3 km
Trip Duration: 03:33:56
=================================================
Continue onto Albion Street ( 0.3 km / 0.2 miles )
Turn sharp right onto Great George Street ( 0.3 km / 0.2 miles )
Keep right onto New Briggate ( 0.5 km / 0.3 miles )
<output omitted>
Keep left onto Haymarket ( 0.0 km / 0.0 miles )
Turn left onto Pall Mall East ( 0.3 km / 0.2 miles )
Enter roundabout ( 0.1 km / 0.1 miles )
Arrive at destination ( 0.0 km / 0.0 miles )
=================================================

+++++++++++++++++++++++++++++++++++++++++++++
Vehicle profiles available on Graphhopper:
+++++++++++++++++++++++++++++++++++++++++++++
car, bike, foot
+++++++++++++++++++++++++++++++++++++++++++++
Enter a vehicle profile from the list above: foot
Starting Location: Tempe, Arizona
Geocoding API URL for Tempe, Arizona, United States (Location Type: city)
https://graphhopper.com/api/1/geocode?q=Tempe%2C+Arizona&limit=1&key=a0552be1-e9b3-
4cc8-b74a-9b4757a7e958
(200, 33.4255117, -111.940016, 'Tempe, Arizona, United States')
Destination: Mesa, Arizona
Geocoding API URL for Mesa, Arizona, United States (Location Type: city)
https://graphhopper.com/api/1/geocode?q=Mesa%2C+Arizona&limit=1&key=a0552be1-e9b3-
4cc8-b74a-9b4757a7e958
=================================================
Routing API Status: 200
Routing API URL:
https://graphhopper.com/api/1/route?key=a0552be1-e9b3-4cc8-b74a-
9b4757a7e958&vehicle=foot&point=33.4255117%2C-111.940016&point=33.4151005%2C-
111.831455
=================================================
Directions from Tempe, Arizona, United States to Mesa, Arizona, United States by foot
=================================================
Distance Traveled: 7.0 miles / 11.2 km
Trip Duration: 02:15:56
=================================================
Continue onto East 5th Street ( 0.0 km / 0.0 miles )
Turn sharp right ( 0.0 km / 0.0 miles )
Turn left ( 0.5 km / 0.3 miles )
Turn right ( 0.0 km / 0.0 miles )
<output omitted>
Turn right ( 0.0 km / 0.0 miles )
Turn left onto West Main Street ( 0.0 km / 0.0 miles )
Turn right onto North Center Street ( 0.0 km / 0.0 miles )
Arrive at destination ( 0.0 km / 0.0 miles )
=============================================

+++++++++++++++++++++++++++++++++++++++++++++
Vehicle profiles available on Graphhopper:
+++++++++++++++++++++++++++++++++++++++++++++
car, bike, foot
+++++++++++++++++++++++++++++++++++++++++++++
Enter a vehicle profile from the list above: q
devasc@labvm:~/labs/devnet-src/graphhopper$
```