# Lab - Integrate a REST API in a Python Application (Instructor Version)

**Instructor Note**: Red font color or gray highlights indicate text that appears in the instructor copy only.

## **Answers: 4.9.2 Lab - Integrate a REST API in a Python Application**

## Objectives

**Part 1: Launch the DEVASC VM**

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

**Part 3: Get a MapQuest API Key**

**Part 4: Build the Basic MapQuest Direction Application**

**Part 5: Upgrade the MapQuest Directions Application with More Features**

**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 MapQuest Directions API, parses the data, and formats it for output to the user. You will use the GET Route request from the MapQuest Directions API. Review the GET Route Directions API documentation here:

https://developer.mapquest.com/documentation/directions-api/route/get/

**Note**: If the above link no longer works, search for "MapQuest 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 MapQuest Directions Application

Your instructor may demonstrate the MapQuest 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. It then requests JSON data from the MapQuest Directions API, parses it, and displays useful information.

```
>>>
Starting Location: Washington
Destination: Baltimore
```

```
URL:
https://www.mapquestapi.com/directions/v2/route?key=your_api_key&from=Washington&to=Ba
ltimore
API Status: 0 = A successful route call.

Directions from Washington to Baltimore
Trip Duration:   00:49:19
Kilometers:      61.32
Fuel Used (Ltr): 6.24
=============================================
Start out going north on 6th St/US-50 E/US-1 N toward Pennsylvania Ave/US-1 Alt N.
(1.28 km)
Turn right onto New York Ave/US-50 E. Continue to follow US-50 E (Crossing into
Maryland). (7.51 km)
Take the Balt-Wash Parkway exit on the left toward Baltimore. (0.88 km)
Merge onto MD-295 N. (50.38 km)
Turn right onto W Pratt St. (0.86 km)
Turn left onto S Calvert St/MD-2. (0.43 km)
Welcome to BALTIMORE, MD. (0.00 km)
=============================================

Starting Location: quit
>>>
```

**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 MapQuest API key you obtain in Part 3.

**Instructor Note**: The following is the answer script for the MapQuest Directions Application. You may want to show the script to the students and demonstrate its operation. However, we recommend that you do not give students the script. They will build it over the course of this lab.

```
#Replace "your_api_key" with your MapQuest API key

import urllib.parse
import requests

main_api = "https://www.mapquestapi.com/directions/v2/route?"
key = "your_api_key"


while True:
    orig = input("Starting Location: ")
    if orig == "quit" or orig == "q":
        break
    dest = input("Destination: ")
    if dest == "quit" or dest == "q":
        break
    url = main_api + urllib.parse.urlencode({"key":key, "from":orig, "to":dest})
    json_data = requests.get(url).json()
    print("URL: " + (url))
    json_data = requests.get(url).json()
    json_status = json_data["info"]["statuscode"]
    if json_status == 0:
        print("API Status: " + str(json_status) + " = A successful route call.\n")
        print("=============================================")
        print("Directions from " + (orig) + " to " + (dest))
        print("Trip Duration:   " + (json_data["route"]["formattedTime"]))
        print("Kilometers:      " + 
str("{:.2f}".format((json_data["route"]["distance"])*1.61)))
        print("Fuel Used (Ltr): " + 
str("{:.2f}".format((json_data["route"]["fuelUsed"])*3.78)))
        print("=============================================")
        for each in json_data["route"]["legs"][0]["maneuvers"]:
            print((each["narrative"]) + " (" + 
str("{:.2f}".format((each["distance"])*1.61) + " km)"))
        print("=============================================\n")
    elif json_status == 402:
        print("**********************************************")
        print("Status Code: " + str(json_status) + "; Invalid user inputs for one or 
both locations.")
        print("**********************************************\n")
    elif json_status == 611:
        print("**********************************************")
        print("Status Code: " + str(json_status) + "; Missing an entry for one or both 
locations.")
        print("**********************************************\n")
    else:

print("************************************************************************")
        print("For Staus Code: " + str(json_status) + "; Refer to:")
        print("https://developer.mapquest.com/documentation/directions-api/status-
codes")

print("************************************************************************\n")
```

## Part 3: Get a MapQuest API Key

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

a. Go to: https://developer.mapquest.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 clicking **Sign Me Up**, you are redirected to the **Manage Keys** page. If you are redirected elsewhere, click **Manage Keys** from the list of options on the left.

e. Click **Approve All Keys**.

f. Expand **My Application**.

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

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

## Part 4: Build the Basic MapQuest Direction Application

In this Part, you will create a Python script to send a URL request to the MapQuest directions API. You will then test your API call. 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. Open **VS Code**. There is a shortcut on the **Desktop**, for your convenience.

b. Select File > Open Folder...

c. Navigate to the **~/labs/devnet-src/mapquest** directory and click **OK**. This directory is currently empty and is where you will store each iteration of your application.

d. Select **File** > **New File**.

e. Select **File** > **Save as…** and name the file **mapquest_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 urllib.parse
import requests
```

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

### Step 3: Build the URL for the request to the MapQuest directions 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:

- **main_api** - the main URL that you are accessing
- **orig** - the parameter to specify your point of origin
- **dest** - the parameter to specify your destination
- **key** - the MapQuest API key you retrieved from the developer 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 Consumer Key you copied from your MapQuest developer account.

```
main_api = "https://www.mapquestapi.com/directions/v2/route?"
orig = "Washington, D.C."
dest = "Baltimore, Md"
key = "your_api_key"
```

b. Combine the four variables **main_api**, **orig**, **dest**, and **key** to format the requested URL. 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").

```
url = main_api + urllib.parse.urlencode({"key":key, "from":orig, "to":dest})
```

c. Create a variable to hold the reply of the requested URL and print the returned JSON data. 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 MapQuest 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.

```
json_data = requests.get(url).json()
print(json_data)
```

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

```
import urllib.parse
import requests

main_api = "https://www.mapquestapi.com/directions/v2/route?"
orig = "Washington, D.C."
dest = "Baltimore, Md"
key = "fZadaFOY22VIEEemZcBFfxl5vjSXIPpZ" #Replace with your MapQuest key

url = main_api + urllib.parse.urlencode({"key":key, "from":orig, "to":dest})

json_data = requests.get(url).json()
print(json_data)
```

## Step 4: Test the URL request.

a. Save and run your **mapquest_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 **route** is another dictionary that includes additional dictionaries and lists. The key **info** includes the **statuscode** key/value pair that you will use later in the lab.

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-json_1.py
{'route': {'hasTollRoad': False, 'hasBridge': True, 'boundingBox': {'lr': {'lng': -
76.612137, 'lat': 38.892063}, 'ul': {'lng': -77.019913, 'lat': 39.290443}},
'distance': 38.089, 'hasTimedRestriction': False, 'hasTunnel': False, 'hasHighway':
True, 'computedWaypoints': [], 'routeError': {'errorCode': -400, 'message': ''},
'formattedTime': '00:49:29', 'sessionId': '5eadfc17-00ee-5f21-02b4-1a24-0647e6e69816',
'hasAccessRestriction': False, 'realTime': 2915, 'hasSeasonalClosure': False,
'hasCountryCross': False, 'fuelUsed': 1.65, 'legs': [{'hasTollRoad': False,
'hasBridge': True, 'destNarrative': 'Proceed to BALTIMORE, MD.', 'distance': 38.089,
'hasTimedRestriction': False, 'hasTunnel': False, 'hasHighway': True, 'index': 0,
'formattedTime': '00:49:29', 'origIndex': -1, 'hasAccessRestriction': False,
'hasSeasonalClosure': False, 'hasCountryCross': False, 'roadGradeStrategy': [[]],
'destIndex': 3, 'time': 2969, 'hasUnpaved': False, 'origNarrative': '', 'maneuvers':
[{'distance': 0.792, 'streets': ['6th St', 'US-50 E', 'US-1 N'], 'narrative': 'Start
out going north on 6th St/US-50 E/US-1 N toward Pennsylvania Ave/US-1 Alt N.',
'turnType': 0, 'startPoint': {'lng': -77.019913, 'lat': 38.892063}, 'index': 0,
'formattedTime': '00:02:06', 'directionName': 'North', 'maneuverNotes': [], 'linkIds':
[], 'signs': [{'extraText': '', 'text': '50', 'type': 2, 'url':
'http://icons.mqcdn.com/icons/rs2.png?n=50&d=EAST', 'direction': 8}, {'extraText': '',
'text': '1', 'type': 2, 'url': 'http://icons.mqcdn.com/icons/rs2.png?n=1&d=NORTH',
<<<<<                  >>>>>
      output omitted
<<<<<                  >>>>>
'geocodeQuality': 'CITY', 'adminArea1Type': 'Country', 'adminArea3Type': 'State',
'latLng': {'lng': -76.61233, 'lat': 39.29044}}], 'time': 2969, 'hasUnpaved': False,
'locationSequence': [0, 1], 'hasFerry': False}, 'info': {'statuscode': 0, 'copyright':
{'imageAltText': '© 2019 MapQuest, Inc.', 'imageUrl':
'http://api.mqcdn.com/res/mqlogo.gif', 'text': '© 2019 MapQuest, Inc.'}, 'messages':
[]}}
devasc@labvm:~/labs/devnet-src/mapquest$
```

c. Change the **orig** and **dest** variables. 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:

```
orig = "Rome, Italy"
dest = "Frascati, Italy"
```

or

```
orig = "Roma, Italia"
dest = "Frascati, 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 **mapquest_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:

- Print the constructed URL so that the user can see the exact request made by the application.
- Parse the JSON data to obtain the **statuscode** value.
- Start an **if** loop that checks for a successful call, which is indicated by a returned value of 0. Add a print statement to display the **statuscode** value and its meaning. The **\n** adds a blank line below the output.

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

```
print("URL: " + (url))

json_data = requests.get(url).json()
json_status = json_data["info"]["statuscode"]

if json_status == 0:
    print("API Status: " + str(json_status) + " = A successful route call.\n")
```

## Step 6: Test status and URL print commands.

a. The example here uses the following parameters.

```
orig = "Washington, D.C."
dest = "Baltimore, Md"
```

b. Save and run your **mapquest_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 is embedded in the URL request.

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-json_2.py
```

```
URL:
https://www.mapquestapi.com/directions/v2/route?key=fZadaFOY22VIEEemZcBFfxl5vjSXIPpZ&f
rom=Washington%2C+D.C.&to=Baltimore%2C+Md
API Status: 0 = A successful route call.

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

## Step 7: 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. Complete the following steps to update your application:

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

b. Delete the current **orig** and **dest** variables.

c. Rewrite the **orig** and **dest** 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, highlighted below, after the key parameter. Be sure all the remaining code is correctly indented within the while loop.

```
import urllib.parse
import requests

main_api = "https://www.mapquestapi.com/directions/v2/route?"
key = "fZadaFOY22VIEEemZcBFfxl5vjSXIPpZ"

while True:
   orig = input("Starting Location: ")
   dest = input("Destination: ")
   url = main_api + urllib.parse.urlencode({"key": key, "from":orig, "to":dest})
   print("URL: " + (url))
   json_data = requests.get(url).json()
   json_status = json_data["info"]["statuscode"]
   if json_status == 0:
       print("API Status: " + str(json_status) + " = A successful route call.\n")
```

## Step 8: Test user input functionality.

Run your **mapquest_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/mapquest$ python3 mapquest_parse-json_3.py
Starting Location: Washington, D.C.
Destination: Baltimore, Md
URL:
https://www.mapquestapi.com/directions/v2/route?key=fZadaFOY22VIEEemZcBFfxl5vjSXIPpZ&f
rom=Washington%2C+D.C.&to=Baltimore%2C+Md
API Status: 0 = A successful route call.

Starting Location: ^CTraceback (most recent call last):
  File "mapquest_parse-json_3.py", line 9, in <module>
    orig = input("Starting Location: ")
KeyboardInterrupt
```

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

## Step 9: 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. Save your script as **mapquest_parse-json_4.py**.

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

```
while True:
    orig = input("Starting Location: ")
    if orig == "quit" or orig == "q":
        break
    dest = input("Destination: ")
    if dest == "quit" or dest == "q":
        break
```

## Step 10: Test the quit functionality.

Run your **mapquest_parse-json_4.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/mapquest$ python3 mapquest_parse-json_4.py
Starting Location: q
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-json_4.py
Starting Location: quit
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-json_4.py
Starting Location: Washington, D.C
Destination: q
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-json_4.py
Starting Location: Washington, D.C.
Destination: quit
devasc@labvm:~/labs/devnet-src/mapquest$
```

## Step 11: Display the JSON data in JSONView.

The Chromium browser in the DEVASC VM includes the JSONView extension. You can use this to view a JSON object in a readable, colored, and collapsible format.

a. Run your **mapquest_parse-json_4.py** again and copy the code returned for the URL. Do not use the code below. Your result will include your API key.

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-json_4.py
Starting Location: Washington, D.C.
Destination: Baltimore, Md
URL:
https://www.mapquestapi.com/directions/v2/route?key=fZadaFOY22VIEEemZcBFfxl5vjSXIPpZ&f
rom=Washington%2C+D.C.&to=Baltimore%2C+Md
API Status: 0 = A successful route call.

Starting Location: quit
```

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

b. Paste the URL in the Chromium browser address field.

c. Collapse the JSONView data by selecting the dash "**-**" before **route**, you will see that there are two root dictionaries: **route** and **info**.

```
{
 - route:{
      hasTollRoad: false,
      hasBridge: true,
 <output omitted>
```

You will see that there are two root dictionaries: **route** and **info**. Notice that **info** has the **statuscode** key/value paired used in your code.

```
{
 + route: {},
 - info: {
       statuscode: 0,
     - copyright: {
           imageAltText: "© 2019 MapQuest, Inc.",
           imageUrl: "http://api.mqcdn.com/res/mqlogo.gif",
           text: "© 2019 MapQuest, Inc."
       },
       messages: [ ]
   }
}
```

d. Expand the **route** dictionary (click on the plus sign "**+**" before **route**) and investigate the rich data. There are values to indicate whether the route has toll roads, bridges, tunnels, highways, closures, or crosses into other countries. You should also see values for distance, the total time the trip will take, and fuel usage. To parse and display this data in your application, you would specify the **route** dictionary and then select key/value pair you want to print. You will do some parsing of the route dictionary in the next part of the lab.

## Part 5: Upgrade the MapQuest Directions Application with More Features

In this Part, you will add additional features to your MapQuest Directions Application to provide more information to the user. You will include some trip summary information and then a list of the directions parsed from the legs dictionary. As a final step, you will add some basic error checking to validate user input.

### Step 1: Display trip summary information to include duration, distance, and fuel used.

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

b. Below the API status **print** command, add several **print** statements that display the from and to locations, as well as the **formattedTime**, **distance**, and **fuelUsed** keys.

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.

```
    if json_status == 0:
        print("API Status: " + str(json_status) + " = A successful route call.\n")
        print("=============================================")
        print("Directions from " + (orig) + " to " + (dest))
        print("Trip Duration:   " + (json_data["route"]["formattedTime"]))
        print("Miles:           " + str(json_data["route"]["distance"]))
        print("Fuel Used (Gal): " + str(json_data["route"]["fuelUsed"]))
        print("=============================================")
```

c. Save and run **mapquest_parse-json_5.py** to see the following output.

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-json_5.py
Starting Location: Washington, D.C.
Destination: Baltimore, Md
URL:
https://www.mapquestapi.com/directions/v2/route?key=fZadaFOY22VIEEemZcBFfxl5vjSXIPpZ&f
rom=Washington%2C+D.C.&to=Baltimore%2C+Md
API Status: 0 = A successful route call.

=============================================
Directions from Washington, D.C. to Baltimore, Md
Trip Duration:   00:49:29
Miles:           38.089
Fuel Used (Gal): 1.65
=============================================
Starting Location: q
devasc@labvm:~/labs/devnet-src/mapquest$
```

d. By defauilt, MapQuest uses the imperial system and there is not a request parameter to change data to the metric system. Therefore, you should probably convert your application to display metric values, as shown below.

```
    print("Kilometers:      " + str((json_data["route"]["distance"])*1.61))
    print("Fuel Used (Ltr): " + str((json_data["route"]["fuelUsed"])*3.78))
```

e. Run the modified **mapquest_parse-json_5.py** script to see the following output:

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-json_5.py
Starting Location: Washington, D.C.
Destination: Baltimore, Md
URL:
https://www.mapquestapi.com/directions/v2/route?key=fZadaFOY22VIEEemZcBFfxl5vjSXIPpZ&f
rom=Washington%2C+D.C.&to=Baltimore%2C+Md
API Status: 0 = A successful route call.

=============================================
Directions from Washington, D.C. to Baltimore, Md
Trip Duration:   00:49:29
Kilometers:      61.32329
Fuel Used (Ltr): 6.236999999999999
=============================================
Starting Location: q
devasc@labvm:~/labs/devnet-src/mapquest$
```

f. The extra decimal places for Kilometers and Fuel Used are not helpful. Use the **"{:.2f}".format** argument to format the float values to 2 decimal places before converting them to string values, as shown below. Each statement should be on one line.

```
    print("Kilometers:      " +
str("{:.2f}".format((json_data["route"]["distance"])*1.61)))
```

```
        print("Fuel Used (Ltr): " +
    str("{:.2f}".format((json_data["route"]["fuelUsed"])*3.78)))
```

## Step 2: Test the parsing and formatting functionality.

Save and run your **mapquest_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/mapquest$ python3 mapquest_parse-json_5.py
Starting Location: Washington, D.C.
Destination: Baltimore, Md
URL:
https://www.mapquestapi.com/directions/v2/route?key=fZadaFOY22VIEEemZcBFfxl5vjSXIPpZ&f
rom=Washington%2C+D.C.&to=Baltimore%2C+Md
API Status: 0 = A successful route call.

=============================================
Directions from Washington, D.C. to Baltimore, Md
Trip Duration:   00:49:29
Kilometers:      61.32
Fuel Used (Ltr): 6.24
=============================================
Starting Location: q
devasc@labvm:~/labs/devnet-src/mapquest$
```

## Step 3: Inspect the maneuvers 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 URL from last time you ran the program and paste it into the browser address bar.

b. Inside the **route** dictionary, locate the **legs** list. The **legs** list includes one big dictionary with most of the JSON data. Find the **maneuvers** 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 maneuver dictionaries.

```
- legs: [
    - {
         hasTollRoad: false,
         hasBridge: true,
         destNarrative: "Proceed to BALTIMORE, MD.",
         distance: 38.089,
         hasTimedRestriction: false,
         hasTunnel: false,
         hasHighway: true,
         index: 0,
         formattedTime: "00:49:29",
         origIndex: -1,
         hasAccessRestriction: false,
         hasSeasonalClosure: false,
         hasCountryCross: false,
       - roadGradeStrategy: [
                [ ]
            ],
            destIndex: 3,
            time: 2969,
                  hasUnpaved: false,
            origNarrative: "",
         - maneuvers: [
               + {…},
               + {…},
               + {…},
               + {…},
               + {…},
               + {…},
               + {…}
            ],
            hasFerry: false
        }
          ],
         - options: {
```

c. Expand the first dictionary in the **maneuvers** list. Each dictionary contains a **narrative** key with a value, such as “Start out going north...”, as shown below. You need to parse the JSON data to extract the value for the **narrative** key to display inside your application.

```
- legs: [
    - {
        hasTollRoad: false,
        hasBridge: true,
        destNarrative: "Proceed to BALTIMORE, MD.",
        distance: 38.089,
        hasTimedRestriction: false,
        hasTunnel: false,
        hasHighway: true,
        index: 0,
        formattedTime: "00:49:29",
        origIndex: -1,
        hasAccessRestriction: false,
        hasSeasonalClosure: false,
        hasCountryCross: false,
      - roadGradeStrategy: [
            [ ]
        ],
        destIndex: 3,
        time: 2969,
              hasUnpaved: false,
        origNarrative: "",
    - maneuvers: [
        - {
              distance: 0.792,
            - streets: [
                  "6th St",
                  "US-50 E",
                  "US-1 N"
              ],
              narrative: "Start out going north on 6th St/US-50E/US-1 N toward
 Pennsylvania Ave/US-1 Alt N.",
              turnType: 0,
            - startPoint: {
                 lng: -77.019913,
                 lat: 38.892063
              },
              index: 0,
              formattedTime: "00:02:06",
              directionName: "North",
              maneuverNotes: [ ],
              linkIds: [ ],
            - signs: [
                - {
                      extraText: "",
                      ext: "50",
                      type: 2,
 <output omitted>
```

**Note**: Word wrap was added for the value in the narrative for display purposes.

## Step 4: Add a for loop to iterate through the maneuvers JSON data.

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

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

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

   1) Prints the **narrative** value.

   2) Converts miles to kilometers with ***1.61**.

   3) Formats the 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. It therefore is part of the previous **if** statement that checks the **statuscode** parameter.

```
print("Fuel Used (Ltr): " + str("{:.2f}".format((json_data["route"]["fuelUsed"])*3.78)))
print("=============================================")
for each in json_data["route"]["legs"][0]["maneuvers"]:
    print((each["narrative"]) + " (" + str("{:.2f}".format((each["distance"])*1.61) + " km)"))
print("=============================================\n")
```

## Step 5: Activity - Test the JSON iteration.

Save and run your **mapquest_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/mapquest$ python3 mapquest_parse-json_6.py
```

```
Starting Location: Washington, D.C.
Destination: Baltimore, Md
URL:
https://www.mapquestapi.com/directions/v2/route?key=fZadaFOY22VIEEemZcBFfxl5vjSXIPpZ&f
rom=Washington%2C+D.C.&to=Baltimore%2C+Md
API Status: 0 = A successful route call.

=============================================
Directions from Washington, D.C. to Baltimore, Md
Trip Duration:   00:49:29
Kilometers:      61.32
Fuel Used (Ltr): 6.24
=============================================
Start out going north on 6th St/US-50 E/US-1 N toward Pennsylvania Ave/US-1 Alt N.
(1.28 km)
Turn right onto New York Ave/US-50 E. Continue to follow US-50 E (Crossing into
Maryland). (7.51 km)
Take the Balt-Wash Parkway exit on the left toward Baltimore. (0.88 km)
Merge onto MD-295 N. (50.38 km)
Turn right onto W Pratt St. (0.86 km)
Turn left onto S Calvert St/MD-2. (0.43 km)
Welcome to BALTIMORE, MD. (0.00 km)
=============================================

Starting Location: q
devasc@labvm:~/labs/devnet-src/mapquest$
```

### Step 6: Check for invalid user input.

Now you are ready to add one final feature to your application to report an error when the user enters invalid data. Recall that you started an if loop to make sure the returned **statuscode** equals 0 before parsing the JSON data:

```
json_status = json_data["info"]["statuscode"]

if json_status == 0:
    print("API Status: " + str(json_status) + " = A successful route call.\n")
```

But what happens if the **statuscode** is not equal to 0? For example, the user might enter an invalid location or might not enter one or more locations. 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 your application to fail without user notification, try the following values in your application. You should see similar results.

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-json_6.py
Starting Location: Washington, D.C.
Destination: Beijing, China
URL:
https://www.mapquestapi.com/directions/v2/route?key=fZadaFOY22VIEEemZcBFfxl5vjSXIPpZ&f
rom=Washington%2C+D.C.&to=Beijing%2C+China
Starting Location: Washington, D.C.
Destination: Bal
```

```
URL:
https://www.mapquestapi.com/directions/v2/route?key=fZadaFOY22VIEEmZcBFfxl5vjSXIPpZ&f
rom=Washington%2C+D.C.&to=Bal
Starting Location: q
devasc@labvm:~/labs/devnet-src/mapquest$
```

b. Copy one of the URLs to a Chromium browser tab. Notice that the only entry in **route** dictionary is a **routeError** dictionary with the **errorCode** 2. In the info dictionary, the **statuscode** is **402**. Therefore, your if loop never executed the code for when the **statuscode** is **0**.

c. Save your script as **mapquest_parse-json_7.py**.

d. To provide error information when **statuscode** is equal to **402**, **611**, or another value, add two **elif** statements and an **else** statement to your if loop. Your **elif** and **else** statements must align with the previous **if** statement. After the last double line print statement under the **if json_status == 0**, add the following **elif** and **else** statements:

```
    if json_status == 0:
       <statements omitted>
       for each in json_data["route"]["legs"][0]["maneuvers"]:
           print((each["narrative"]) + " (" +
str("{:.2f}".format((each["distance"])*1.61) + " km)"))
        print("=============================================\n")
    elif json_status == 402:
        print("**********************************************")
        print("Status Code: " + str(json_status) + "; Invalid user inputs for one or both
locations.")
        print("**********************************************\n")
    elif json_status == 611:
        print("**********************************************")
        print("Status Code: " + str(json_status) + "; Missing an entry for one or both
locations.")
        print("**********************************************\n")
    else:
        print("************************************************************************")
        print("For Staus Code: " + str(json_status) + "; Refer to:")
        print("https://developer.mapquest.com/documentation/directions-api/status-codes")
        print("************************************************************************\n")
```

The first **elif** statement prints if the **statuscode** value is **402** for an invalid location. The second **elif** statement prints if the **statuscode** value is **611** because the user did not provide an entry for one or both locations. The **else** statement prints for all other **statuscode** values, such as when the MapQuest site returns an error. The **else** statement ends the **if/else** loop and returns the application to the while loop.

## Part 6: Test Full Application Functionality

Run your **mapquest_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/mapquest$ python3 mapquest_parse-json_7.py
Starting Location: Washington, D.C.
Destination: Baltimore, Md
URL:
https://www.mapquestapi.com/directions/v2/route?key=fZadaFOY22VIEEmZcBFfxl5vjSXIPpZ&f
rom=Washington%2C+D.C.&to=Baltimore%2C+Md
API Status: 0 = A successful route call.
```

```
=============================================
Directions from Washington, D.C. to Baltimore, Md
Trip Duration:   00:49:29
Kilometers:      61.32
Fuel Used (Ltr): 6.24
=============================================
Start out going north on 6th St/US-50 E/US-1 N toward Pennsylvania Ave/US-1 Alt N.
(1.28 km)
Turn right onto New York Ave/US-50 E. Continue to follow US-50 E (Crossing into
Maryland). (7.51 km)
Take the Balt-Wash Parkway exit on the left toward Baltimore. (0.88 km)
Merge onto MD-295 N. (50.38 km)
Turn right onto W Pratt St. (0.86 km)
Turn left onto S Calvert St/MD-2. (0.43 km)
Welcome to BALTIMORE, MD. (0.00 km)
=============================================

Starting Location: Moscow, Russia
Destination: Beijing, China
URL:
https://www.mapquestapi.com/directions/v2/route?key=fZadaFOY22VIEEemZcBFfxl5vjSXIPpZ&f
rom=Moscow%2C+Russia&to=Beijing%2C+China
API Status: 0 = A successful route call.

=============================================
Directions from Moscow, Russia to Beijing, China
Trip Duration:   84:31:10
Kilometers:      7826.83
Fuel Used (Ltr): 793.20
=============================================
Start out going west on Кремлёвская набережная/Kremlin Embankment. (0.37 km)
Turn slight right onto ramp. (0.15 km)
Turn slight right onto Боровицкая площадь. (0.23 km)
<output omitted>
Turn slight left onto 前门东大街/Qianmen East Street. (0.31 km)
Turn left onto 广场东侧路/E. Guangchang Rd. (0.82 km)
广场东侧路/E. Guangchang Rd becomes 东长安街/E. Chang'an Str. (0.19 km)
Welcome to BEIJING. (0.00 km)
=============================================

Starting Location: Washington, D.C.
Destination: Beijing, China
URL:
https://www.mapquestapi.com/directions/v2/route?key=fZadaFOY22VIEEemZcBFfxl5vjSXIPpZ&f
rom=Washington%2C+D.C.&to=Beijing%2C+China
**********************************************
Status Code: 402; Invalid user inputs for one or both locations.
**********************************************
```

```
Starting Location: Washington, D.C.
Destination: Bal
URL:
https://www.mapquestapi.com/directions/v2/route?key=fZadaFOY22VIEEemZcBFfxl5vjSXIPpZ&f
rom=Washington%2C+D.C.&to=Bal
**********************************************
Status Code: 402; Invalid user inputs for one or both locations.
**********************************************

Starting Location: Washington, D.C.
Destination:
URL:
https://www.mapquestapi.com/directions/v2/route?key=ANUqwkHlgDv1vlsyBPtVrFeX8
uu6agjA&from=Washington%2C+D.C.&to=
**********************************************
Status Code: 611; Missing an entry for one or both locations.
**********************************************

Starting Location: quit
devasc@labvm:~/labs/devnet-src/mapquest$
```