Lab - Parsing JSON with a Python Application

Objectives

- Obtain a MapQuest API Key.
- Import necessary modules.
- Create API request variables and construct a URL.
- Add user input functionality.
- Add a quit feature so that the user can end the application.
- Display trip information for time, distance, and fuel usage.
- Iterate through the JSON data to extract and output the directions.
- Display error messages for invalid user input.

Background / Scenario

In this lab, you will create an application 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

- Computer with Python installed according to the **Lab - PC Setup for Workshop**.
- Access to the internet.

Instructions

Part 1: Demonstrate the MapQuest Directions API application.

Your instructor may demonstrate the application and show you the script used to create it. In addition, your instructor may allow you to have the solution script, **08_parse-json_sol.py**. 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=Baltimore
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 will need to replace **your_api_key** with the MapQuest API key you obtained in Step 3.

Part 2: Authenticating a RESTful request.

Before building the application, you will need to obtain a key from MapQuest developer site, which you will do in the next step. Previously, you used no authentication to access the ISS Pass Predictions API. However, many APIs require some form of authentication.

Authenticating a RESTful request is done in one of four ways. Describe each method below:

- **None**: You are free to use the API, but it is very vulnerable to attacks
- **Basic HTTP**: The simplest and easiest way to authorize an API. It is mainly based on creating a username and password to identify yourself within the API
- **Token:** Here the user identifies himself as with HTTP Basic Authentication, using his credentials, username and password. But in this case, the server generates a token that is shared only with the user, and that is required for the API to be valid.
- **Open Authorization (OAuth)**: OAuth 2.0 is a method that is used by large companies to use their services. Its purpose is to allow other providers, services or applications, such as Authenticators, access to information without directly providing user credentials, using encrypted keys that are accepted by service companies.

- **None:** Eres libre de utilizar la API, pero es muy vulnerable a los ataques.
- **HTTP básico:** La forma más sencilla y sencilla de autorizar una API. Se basa principalmente en crear un usuario y contraseña para identificarse dentro de la API
- **Token:** Aquí el usuario se identifica como con Autenticación Básica HTTP, utilizando sus credenciales, nombre de usuario y contraseña. Pero en este caso, el servidor genera un token que se comparte sólo con el usuario y que es necesario para que la API sea válida.
- **Autorización abierta (OAuth):** OAuth 2.0 es un método que utilizan las grandes empresas para utilizar sus servicios. Su finalidad es permitir que otros proveedores, servicios o aplicaciones, como Autenticadores, accedan a la información sin proporcionar directamente las credenciales del usuario, utilizando claves cifradas que sean aceptadas por las empresas de servicios.

For the MapQuest API, you will use token authentication.

Part 3: Get a MapQuest API key.

Complete the following steps to get a MapQuest API key:

Part 4: Go to: https://developer.mapquest.com/.

Part 5: Click **Sign Up** at the top of the page.

Part 6: Fill out the form to create a new account. For **Company**, enter **Cisco Networking Academy Student**.

Part 7: After clicking **Sign Me Up**, you are redirected to the **Manage Keys** page.

Part 8: Click **Approve All Keys** and expand **My Application**.

Part 9: Copy your **Consumer Key** to Notepad 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 for "steps to generate mapquest api key".

Part 10: 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 **request** 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.

Part 11: Open a blank script file and save it **08_parse-json1.py**.

Part 12: Import the **urllib.parse** and **requests** modules.

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

Part 13: Create variables for API request.

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** – This is the main URL that you are accessing.
- **orig** – This is the parameter to specify your point of origin.
- **dest** – This is the parameter to specify your destination.
- **key** – This is the MapQuest API key you retrieved from the developer website.

Part 14: Create variables to build the URL that will be sent in the request. Copy your MapQuest key to the key variable.

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

Part 15: 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 (e.g. space into "+" and a comma into "%2C").

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

Part 16: 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 that is the **json** reply of the **get** method of the **requests** module. The **print** statement is used to check the returned data.

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

Part 17: Test the URL request.

Part 18: Run your **08_json-parse1.py** script and verify that it works. Troubleshoot your code, if necessary. Although your output might be slightly different, you should get a JSON response similar to the following:

```
{'route': {'distance': 38.089, 'hasHighway': True, 'hasUnpaved': False,
'hasAccessRestriction': False, 'options': {'mustAvoidLinkIds':[],
'maxWalkingDistance': -1, 'manmaps': 'true', 'urbanAvoidFactor': -1,
'stateBoundaryDisplay': True, 'cyclingRoadFactor': 1, 'routeType': 'FASTEST',
'countryBoundaryDisplay': True, 'drivingStyle': 2, 'highwayEfficiency': 22,
'narrativeType': 'text', 'routeNumber': 0, 'tryAvoidLinkIds': [], 'generalize': -1,
'returnLinkDirections': False, 'doReverseGeocode': True, 'avoidTripIds': [],
'timeType': 0, 'sideOfStreetDisplay': True, 'filterZoneFactor': -1, 'walkingSpeed': -
1, 'useTraffic': False, 'unit': 'M', 'tr
[output omitted]
>>>
```

Part 19: Change the **orig** and **dest** variables. Rerun the script to get different results.

```python
# Nombre: Eder Gael Saldaña Galván
# Fecha: 23/Octubre/2023
# Descripción: Invocando API
import urllib.parse
import requests
main_api = "https://www.mapquestapi.com/directions/v2/route?"
orig = "Dolores Hidalgo"
dest = "Puerto Vallarta"
key = "1AvgD6pPkSTGJb7jYkYtvLAQxajn0rtu"
url = main_api + urllib.parse.urlencode({"key": key, "from":orig, "to":dest})
json_data = requests.get(url).json()
print(json_data['route']['sessionId'])
```

a) Extrae la distancia y el tiempo

```python
print(json_data['route']['distance'])
print(json_data['route']['time'])
```

b) Extrae el primer elemento llamado formatedTime

```python
print(json_data['route']['formattedTime'])
```

c) Salidas de todos los comandos

```python
# Extrae la distancia y el tiempo
print("Distancia: " + str(json_data['route']['distance']))
print("Tiempo: " + str(json_data['route']['time']))

# Extrae el primer elemento llamado "formattedTime"
print("Tiempo total: " + str(json_data['route']['formattedTime']))
```

```
Distancia: 408.4242
Tiempo: 32027
Tiempo total: 08:53:47
```

Part 20: 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.

Part 21: Save your script as **08_json-parse2.py**.

Part 22: Delete the **print(json_data)** statement because you no longer need to test that the request is properly formatted.

Part 23: Add the statements below, which will do the following:

  o   Print the constructed URL so that the user can see the exact request made by the application.

  o   Parse the JSON data to obtain the **statuscode** value.

  o   Start an **if** loop that checks for a successful call, which has a value of 0. Add a print statement to display the **statuscode** value and its meaning. The **\n** adds a blank line below the output.

```python
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")
```

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

Part 24: Test status and URL print commands.

Run your **08_json-parse2.py** script and verify that it works. Troubleshoot your code, if necessary. You should get output similar to the following:

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

>>>
```

```
# Codificar para manejar el error distinto a 0
elif json_status != 0:
    print("API Status: " + str(json_status) + " = A failure route call.\n")
```

Part 25: Add user input for starting location and destination.

Up to this point, you have used Washington and Baltimore as the static values for the location variables. However, the application requires that the user input these. Complete the following steps to update your application:

Part 26: Save your script as **08_json-parse3.py**.

Part 27: Delete the current **orig** and **dest** variables.

Part 28: 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.

Part 29: Be sure all the remaining code is indented within the **while** loop.

```
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")
```

Part 30: Test user input functionality.

Run your **08_json-parse3.py** script and verify that it works. Troubleshoot your code, if necessary. You should get output similar to what is shown below. You will add quit functionality in the next step. For now, enter **Ctrl+C** to quit the application.

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

Starting Location: <Ctrl+C>
```

```
while True:
    orig = input("Starting Location: ")
    dest = input("Destination: ")
```

Part 31: Add the quit functionality to the application.

Instead of entering **Ctrl+C** to quit the application, 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:

Part 32: Save your script as **08_json-parse4.py**.

Part 33: 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
```

Part 34: Test the quit functionality.

Run your **08_json-parse4.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:

```
Starting Location: q
>>>
Starting Location: quit
>>>
Starting Location: Washington
Destination: q
>>>
Starting Location: Washington
Destination: quit
>>>
```
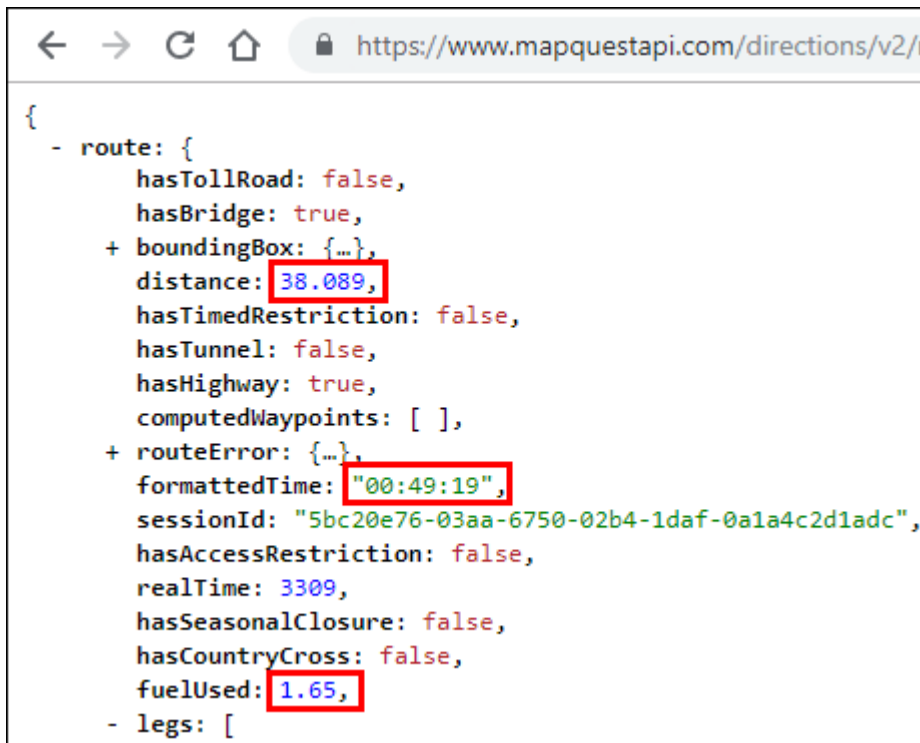

Part 35: Parse and display some data about the trip.

Part 36: Copy your URL into your web browser. If you collapse all the JSON data, you will see that there are two root dictionaries: **route and info**.

Part 37: Expand the **route** dictionary 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, as highlighted below. To parse and display this, specify the **route** dictionary and select key/value pair you want to print.

Part 38: Save your script as **08_json-parse5.py**.

Part 39: Below the API status print command, add print statements that display the from and to locations, as well as the **formattedTime**, **distance**, and **fuelUsed** keys.

Part 40: Add a print statement that will display a double line before the next request for a starting location as shown below.

```
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("=========================================")
```

Part 41: Run **08_json-parse5.py** to see the following output:

```
Starting Location: Washington
Destination: Baltimore
URL: https://www.mapquestapi.com/directions/v2/route?
to=Baltimore&key=Your_api_key&from=Washington
API Status: 0 = A successful route call.

=========================================
Directions from Washington to Baltimore
Trip Duration:  00:49:19
Miles:          38.089
Fuel Used (Gal): 1.65
=========================================
Starting Location: q
>>>
```

Part 42: 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))
```

Part 43: Run the modified **08_json-parse5.py** script to see the following output:

```
Starting Location: Washington
Destination: Baltimore
URL: https://www.mapquestapi.com/directions/v2/route?
key=Your_api_key&to=Baltimore&from=Washington
API Status: 0 = A successful route call.

=============================================
Directions from Washington to Baltimore
Trip Duration:   00:49:19
Kilometers:       61.32329
Fuel Used (Ltr): 6.236999999999999
=============================================
Starting Location: q
>>>
```

Part 44: 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)))
```

Part 45: Test the parsing and formatting functionality.

Run your **08_json-parse5.py** script to verify that 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:

```
Starting Location: Washington
Destination: Baltimore
URL: https://www.mapquestapi.com/directions/v2/route?
key=Your_api_key&to=Baltimore&from=Washington
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
=============================================
Starting Location: q
>>>
```
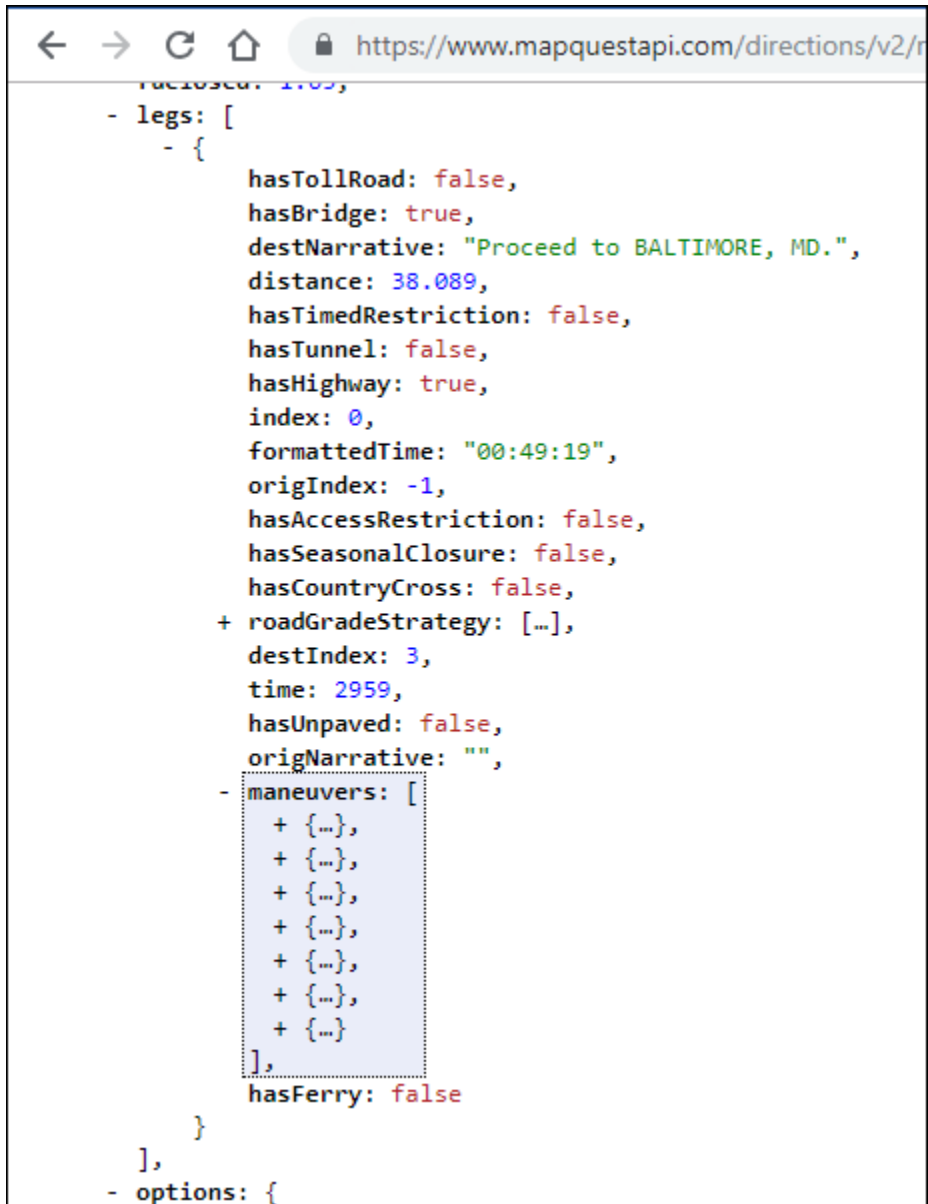
```
=============================================
Directions from Dolores Hidalgo to Puerto Vallarta
Trip Duration:   08:53:27
Miles:           408.4111
Kilometers:      657.54
=============================================
FASTEST
21.318099999999916
-105.40292000000193
```

Part 46: Inspect the maneuvers JSON data.

Part 47: Now you are ready to display the step-by-step directions from the starting location to the destination. Locate the **legs** list inside the route dictionary. The **legs** list includes one big dictionary with most of the JSON data.

Part 48: Find the **maneuvers** list and collapse each of the seven dictionaries inside, as shown below.

Part 49: 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.

Part 50: Add a for loop to iterate through the maneuvers JSON data.

Complete the following steps to update your application:

Part 51: Save your script as **08_json-parse6.py**.

Part 52: Add a **for** loop below the second double line print statement. The **for** loop iterates through each **maneuvers** list and does the following:

Part 53: Prints the **narrative** value.

Part 54: Converts miles to kilometers with ***1.61**.

Part 55: Formats the kilometer value to print only two decimal places with the **"{:.2f}".format** function.

Part 56: Add a print statement that will display a double line before the next request for a starting location, as shown below.

Note: The second double line print statement is not indented within the **for** loop. Therefore, it 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")
```

Part 57: Activity - Test the JSON iteration.

Run your **08_json-parse6.py** script and verify that it works. Troubleshoot your code, if necessary. You should get an output similar to the following:

```
Starting Location: Washington
Destination: Baltimore
URL: https://www.mapquestapi.com/directions/v2/route?
key=Your_api_key&to=Baltimore&from=Washington
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: q
>>>
```

```
======================================
Head toward Calle México on Calle Querétaro. Go for 0.2 mi. (0.32 km)
Turn right onto Avenida Norte. Go for 0.3 mi. (0.40 km)
Continue on Calzada Mariano Balleza toward Antigua Estacion FFCC. Go for 0.5 mi. (0.84 km)
Continue on Calzada de los Héroes. Go for 1.5 mi. (2.39 km)
Continue on Carretera Dolores Hidalgo-S Miguel de Allende toward San Luis de La Paz/San Miguel de Allende. Go for 1.2 mi.
(1.97 km)
Keep right onto Carretera Dolores Hidalgo-S Miguel de Allende toward MEX-51/San Miguel de Allende/Guanajuato/MEX-110. Go f
or 19.2 mi. (30.92 km)
Continue toward MEX-51/San Miguel de Allende. Go for 108 ft. (0.03 km)
Take the 2nd exit from roundabout onto Carretera Dolores Hidalgo-S Miguel de Allende toward MEX-51/San Miguel de Allende.
Go for 0.9 mi. (1.48 km)
Take the 1st exit from roundabout onto Carretera Dolores Hidalgo-S Miguel de Allende. Go for 0.1 mi. (0.23 km)
Continue on Prolongación de la Aurora. Go for 0.3 mi. (0.43 km)
Continue on Calzada de la Aurora. Go for 0.2 mi. (0.25 km)
Turn right onto Calzada de la Luz. Go for 0.2 mi. (0.27 km)
Turn left onto Calle Volanteros. Go for 394 ft. (0.12 km)
Turn slightly left onto Calle Quebrada. Go for 0.3 mi. (0.56 km)
```

Part 58: 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")
```

Part 59: 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. Try the following values in your application. You should see results similar to the following:

```
Starting Location: Washington
Destination: Beijing
URL: https://www.mapquestapi.com/directions/v2/route?
to=Beijing&key=your_api_key&from=Washington
Starting Location: Washington
Destination: Balt
URL: https://www.mapquestapi.com/directions/v2/route?
to=Balt&key=your_api_key&from=Washington
Starting Location: Washington
```

```
Destination:
URL: https://www.mapquestapi.com/directions/v2/route?
to=&key=your_api_key&from=Washington
Starting Location: q
```

Part 60: Save your script as **08_jsont-parse7.py**.

Part 61: To provide error information when this happens, add **elif** and **else** statements to your **if** loop. After the last double line print statement under the **if json_status == 0**, add the following **elif** and **else** statements:

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

The **elif** statement prints if the **statuscode** value is 402 for an invalid location. The **else** statement prints for all other **statuscode** values, such as no entry for one or more locations. The **else** statement ends the **if/else** loop and returns the application to the **while** loop.

```
Starting Location: Dolores Hidalgo
Destination: San Felipe
URL: https://www.mapquestapi.com/directions/v2/route?key=1AvgD6pPkSTGJb7jYkYtvLAQxajn0rtu&from=Dolores+Hidalgo&to=San+Feli
pe
API Status: 402 = An error route call.

*********************************************
Status Code: 402; Invalid user inputs for one or both locations.
*********************************************

Starting Location: ▮
```

Part 62: Activity - Test full application functionality.

Run your **08_json-parse7.py** script and verify that it works. Troubleshoot your code, if necessary. Test all the features of the application. You should get output similar to the following:

```
Starting Location: Washington
Destination: Baltimore
URL: https://www.mapquestapi.com/directions/v2/route?
key=your_api_key&from=Washington&to=Baltimore
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: Moscow
Destination: Beijing
URL: https://www.mapquestapi.com/directions/v2/route?
key=your_api_key&from=Moscow&to=Beijing
API Status: 0 = A successful route call.

Directions from Moscow to Beijing
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 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
Destination: Beijing
URL: https://www.mapquestapi.com/directions/v2/route?
key=your_api_key&from=WashingtonTurn+right+onto+%E5%89%8D%E9%97%A8%E8%A5%BF
%E5%A4%A7%E8%A1%97%2FQianmen+West+Street.+%281.01+km%29&to=Beijing

****************************************************************
Status Code: 402; Invalid user inputs for one or both locations.
****************************************************************

Starting Location: Washington
Destination: Balt
URL: https://www.mapquestapi.com/directions/v2/route?
key=your_api_key&from=Washington&to=Balt

*********************************************************************
Status Code: 602; Refer to:
https://developer.mapquest.com/documentation/directions-api/status-codes
*********************************************************************

Starting Location: Washington
Destination:
URL: https://www.mapquestapi.com/directions/v2/route?
key=your_api_key&from=Washington&to=

*********************************************************************
Status Code: 611; Refer to:
https://developer.mapquest.com/documentation/directions-api/status-codes
*********************************************************************

Starting Location: q
>>>
```